

# Attention and Transformers

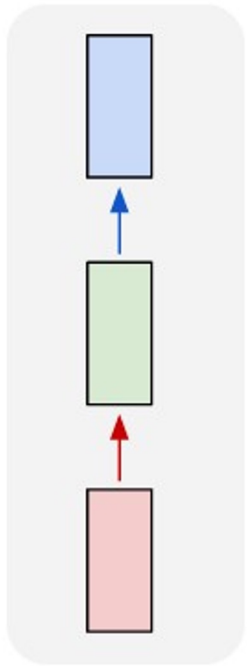
Arjun Majumdar  
Georgia Tech

# Lecture Outline

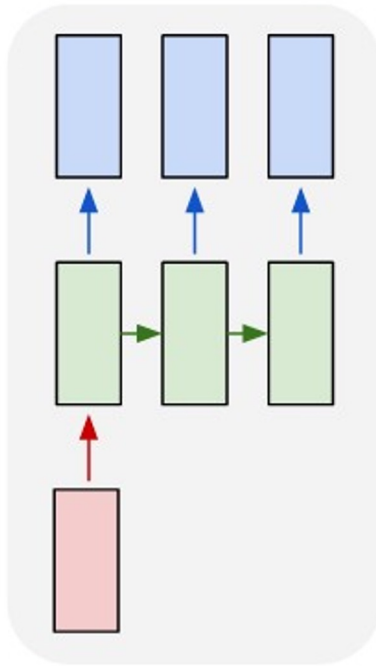
- Machine Translation with RNNs
- RNNs with Attention
- From Attention to Transformers
- What can Transformers do?

# Sequence Modeling with RNNs

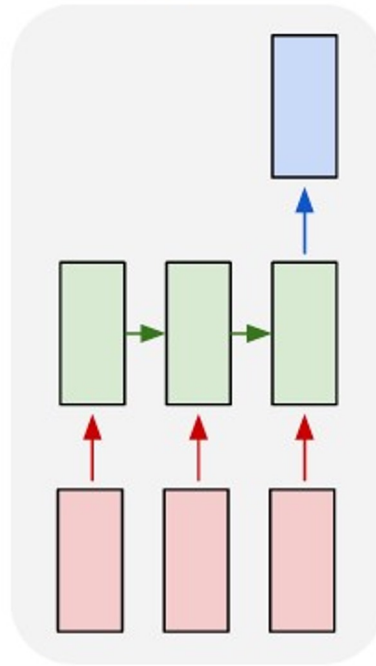
one to one



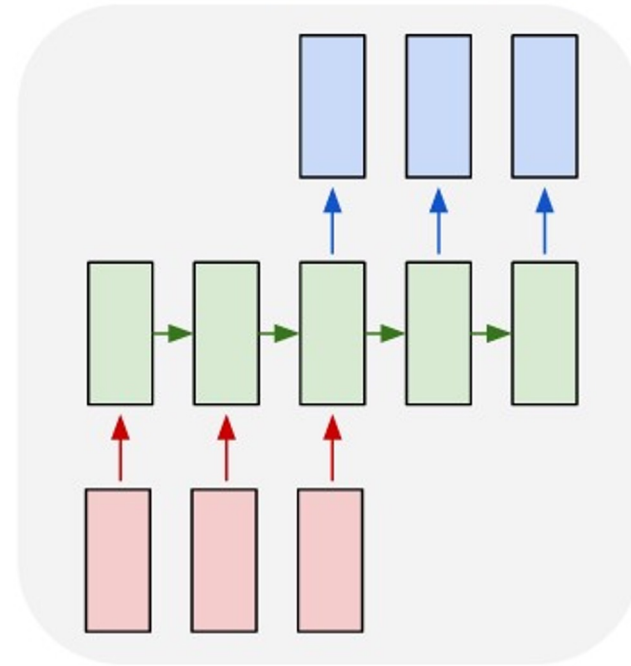
one to many



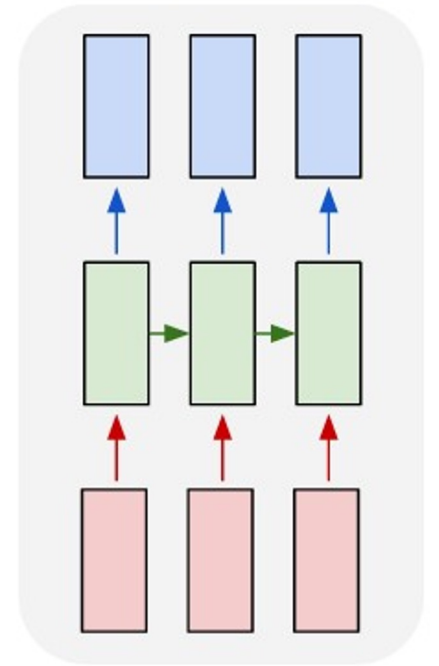
many to one



many to many



many to many



# Machine Translation

we are eating bread



estamos comiendo pan

# Machine Translation

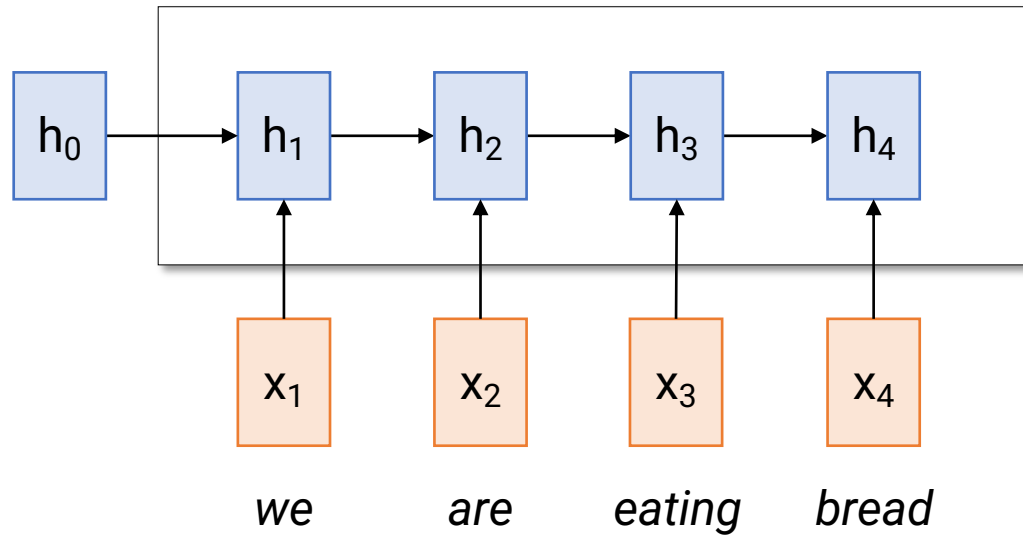


we are eating bread

estamos comiendo pan

# Machine Translation with RNNs

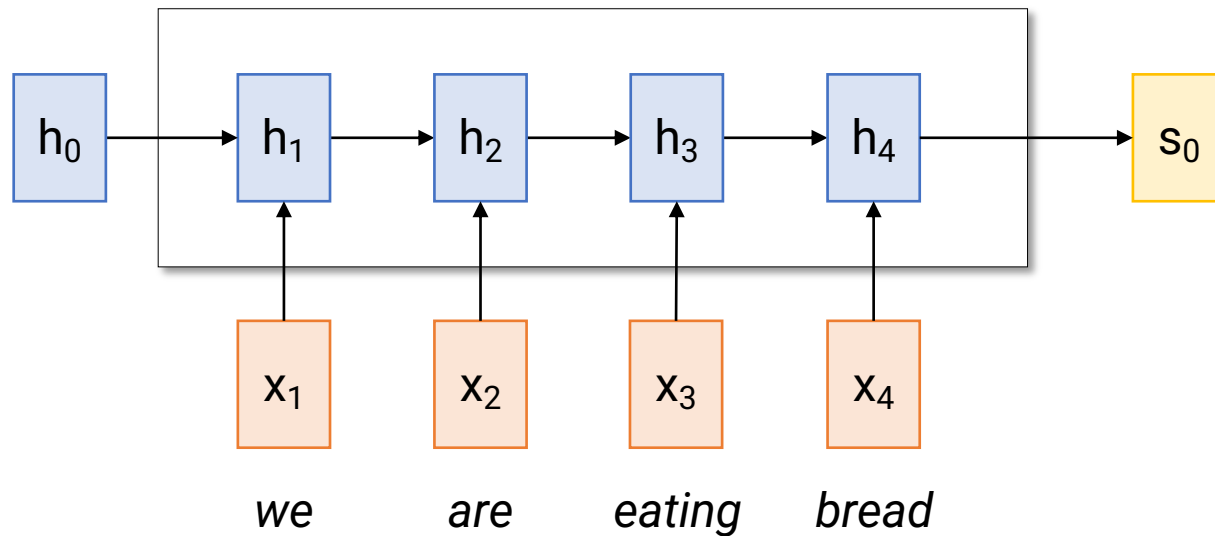
Encoder:  $h_t = f_W(x_t, h_{t-1})$



# Machine Translation with RNNs

Encoder:  $h_t = f_W(x_t, h_{t-1})$

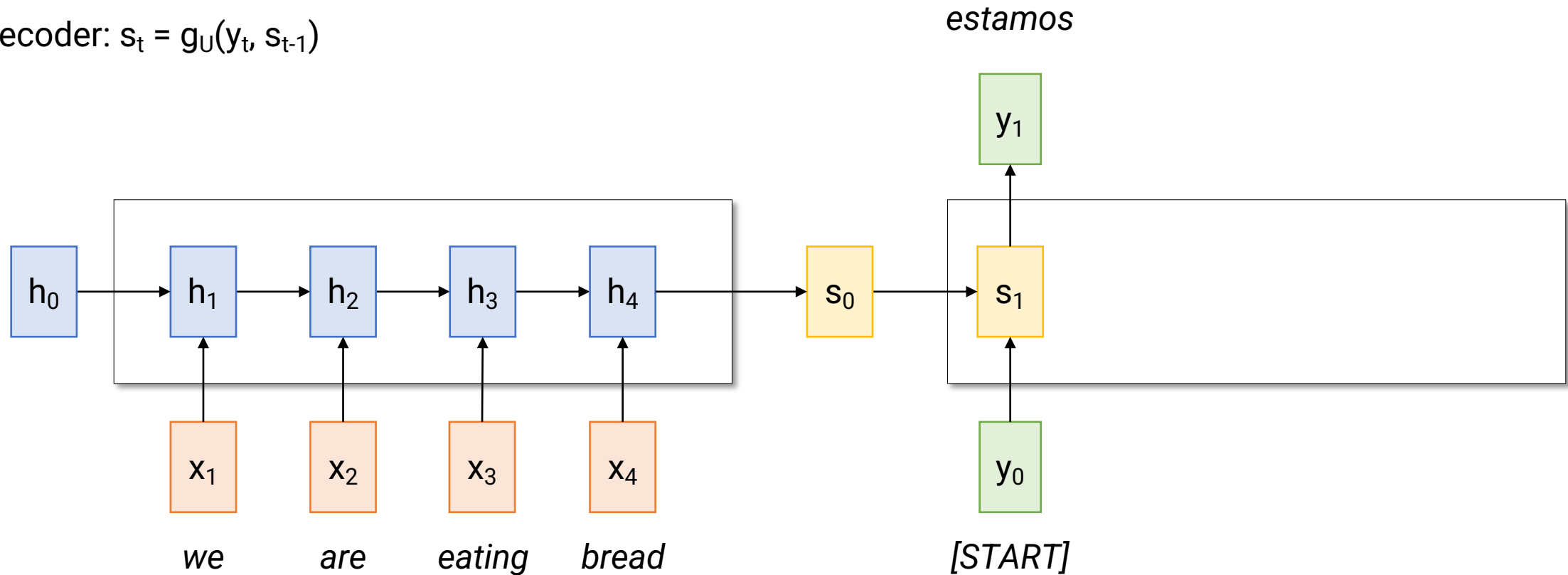
$s_0 = h_4$



# Machine Translation with RNNs

Encoder:  $h_t = f_W(x_t, h_{t-1})$

Decoder:  $s_t = g_U(y_t, s_{t-1})$

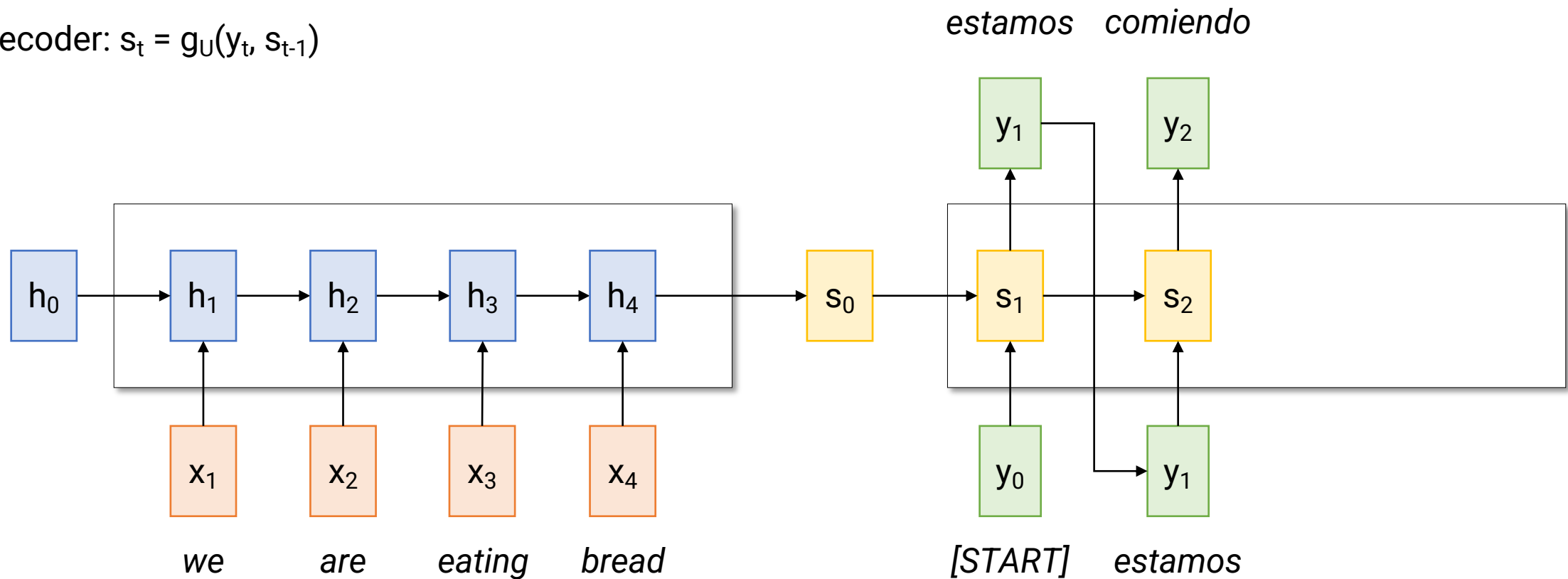




# Machine Translation with RNNs

Encoder:  $h_t = f_W(x_t, h_{t-1})$

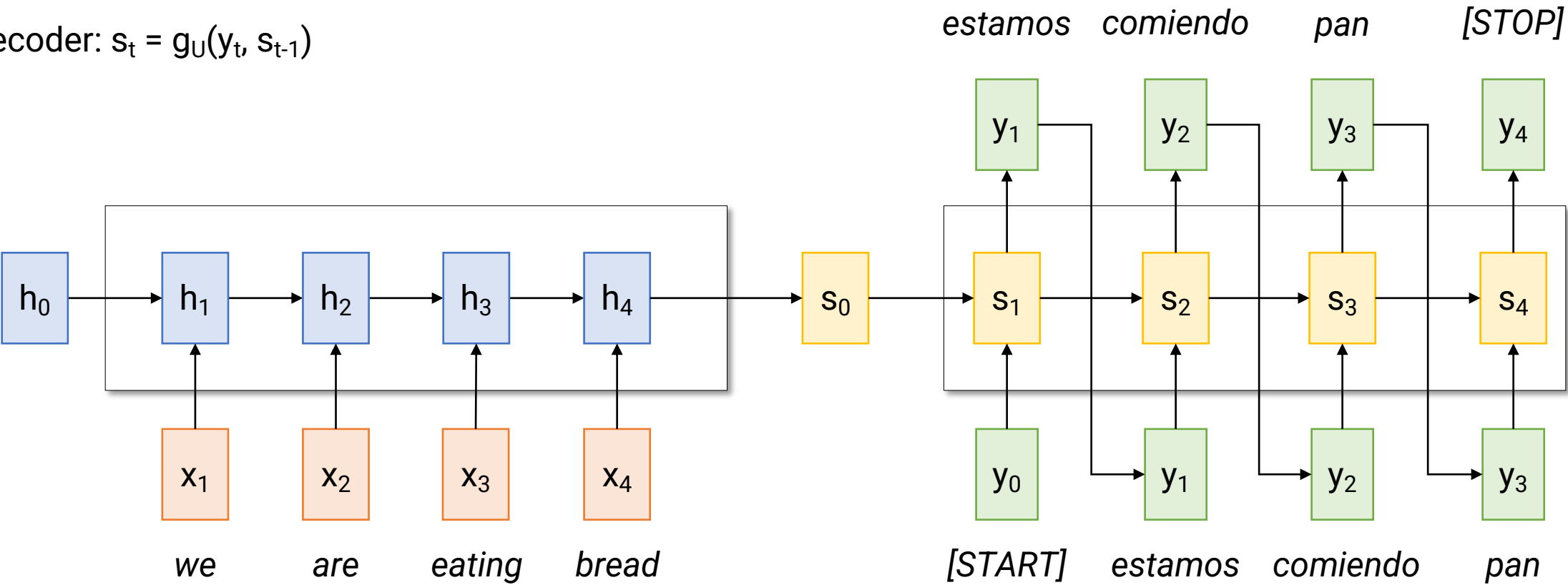
Decoder:  $s_t = g_U(y_t, s_{t-1})$



# Machine Translation with RNNs

Encoder:  $h_t = f_W(x_t, h_{t-1})$

Decoder:  $s_t = g_U(y_t, s_{t-1})$

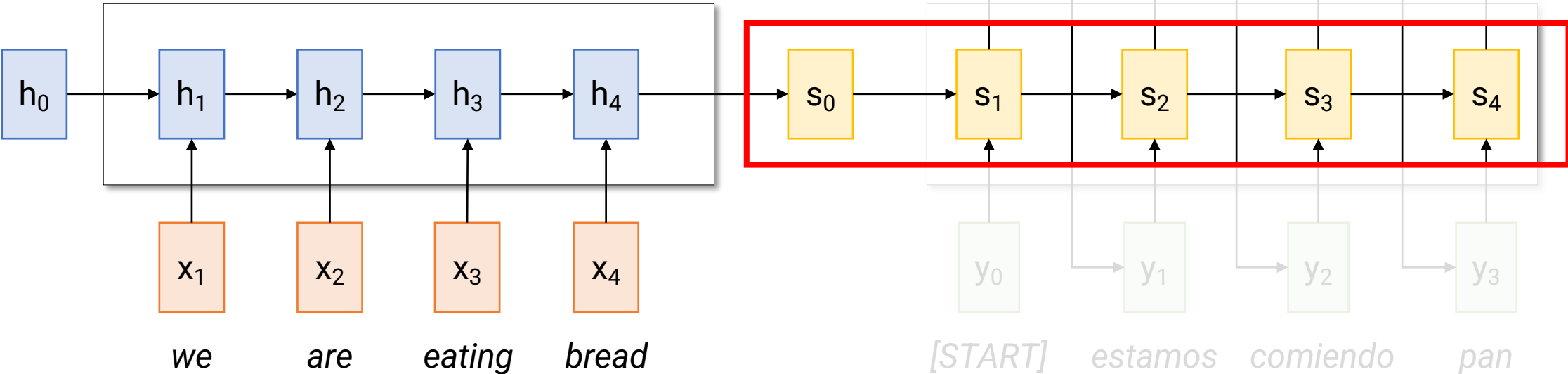


# Machine Translation with RNNs

Encoder:  $h_t = f_W(x_t, h_{t-1})$

Decoder:  $s_t = g_U(y_t, s_{t-1})$

Problem:  $s_i$  is used to encode input and maintain decoder state

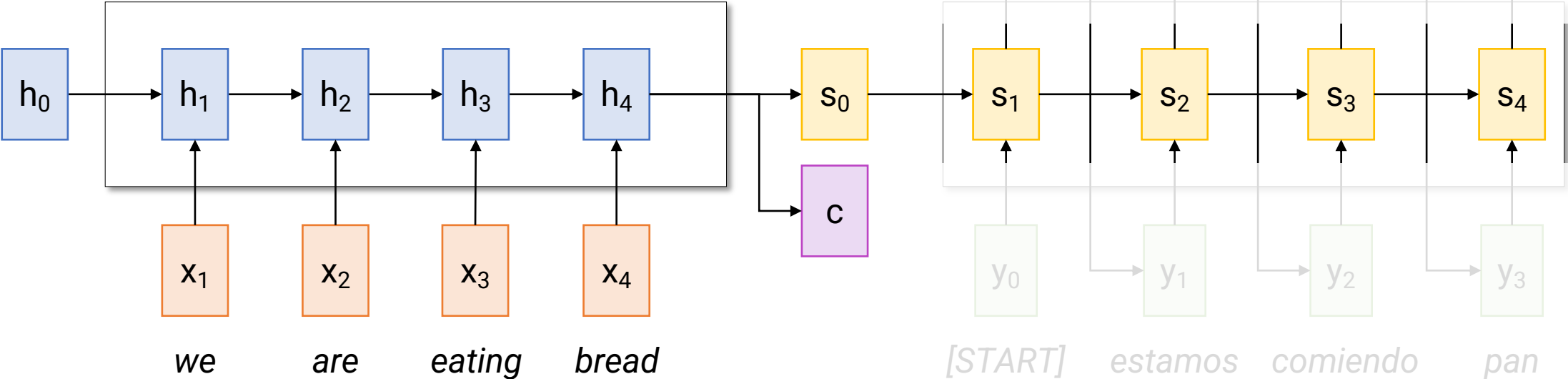


# Machine Translation with RNNs

Encoder:  $h_t = f_W(x_t, h_{t-1})$

Decoder:  $s_t = g_U(y_t, s_{t-1}, c)$

Solution: add a context vector  $c = h_4$  and predict  $s_0$  from  $h_4$

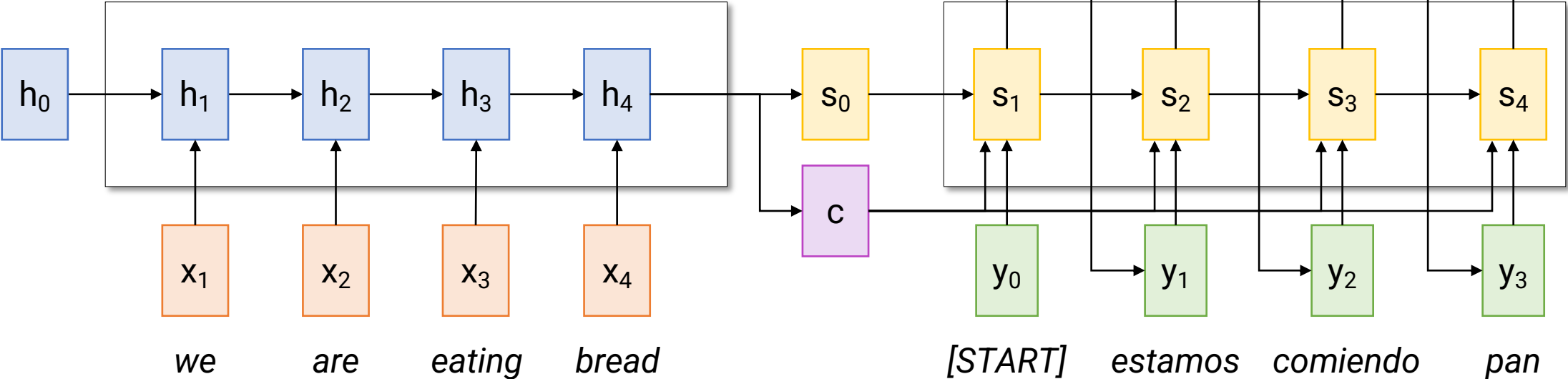


# Machine Translation with RNNs

Encoder:  $h_t = f_W(x_t, h_{t-1})$

Decoder:  $s_t = g_U(y_t, s_{t-1}, c)$

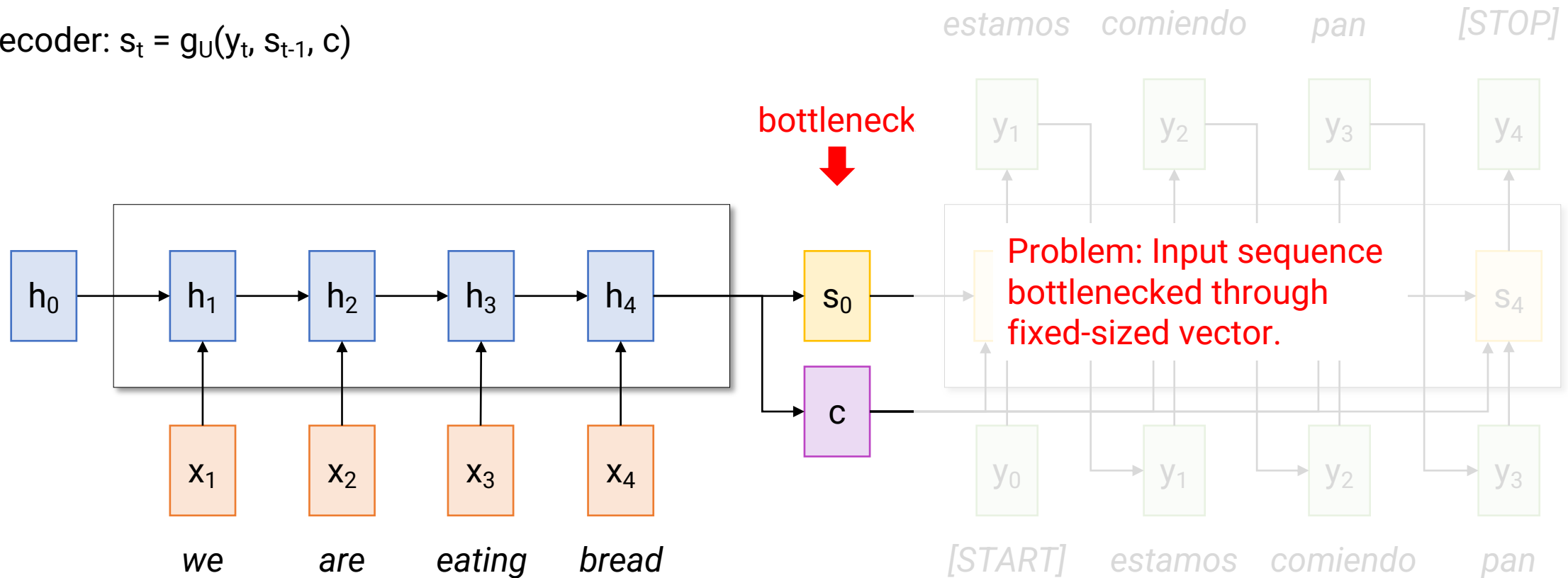
Solution: add a context vector  $c = h_4$  and predict  $s_0$  from  $h_4$



# Machine Translation with RNNs

Encoder:  $h_t = f_W(x_t, h_{t-1})$

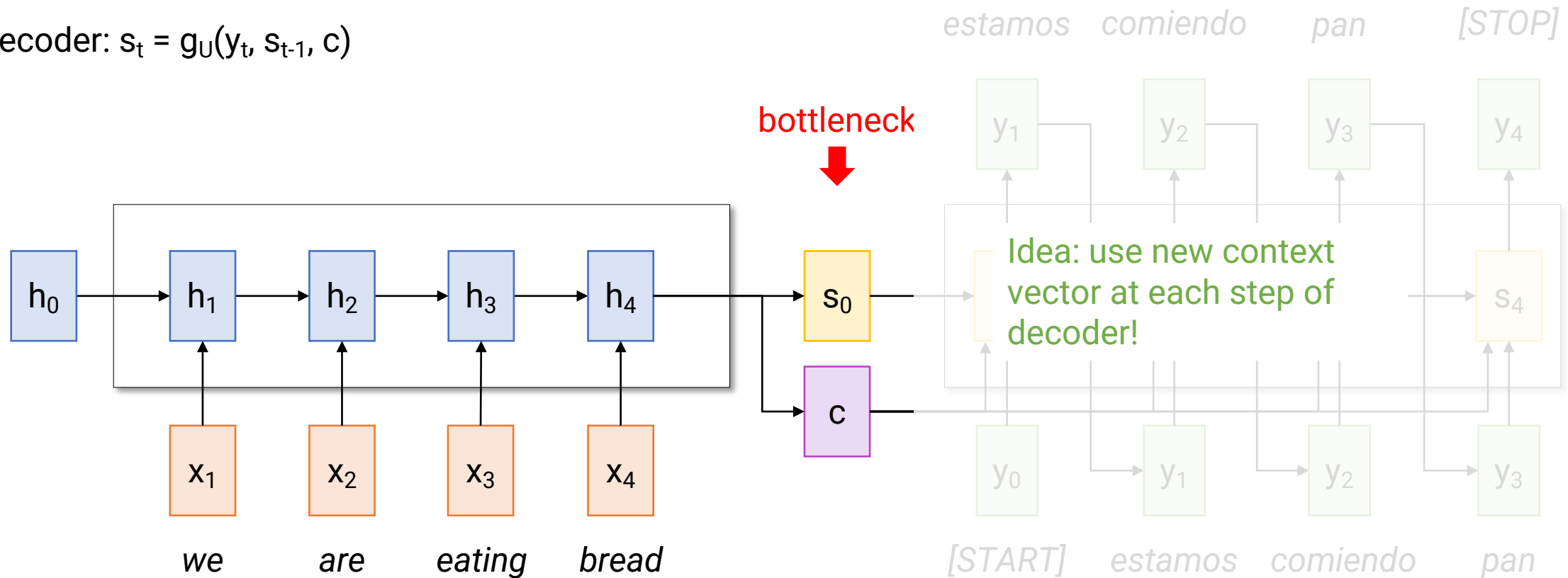
Decoder:  $s_t = g_U(y_t, s_{t-1}, c)$



# Machine Translation with RNNs

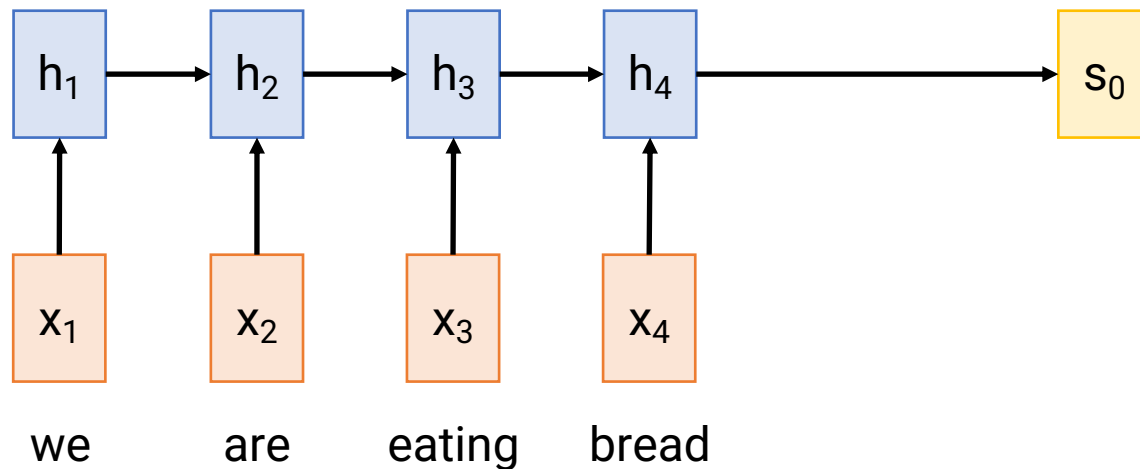
Encoder:  $h_t = f_W(x_t, h_{t-1})$

Decoder:  $s_t = g_U(y_t, s_{t-1}, c)$



# Machine Translation with RNNs **and Attention**

From final hidden state:  
**Initial decoder state  $s_0$**

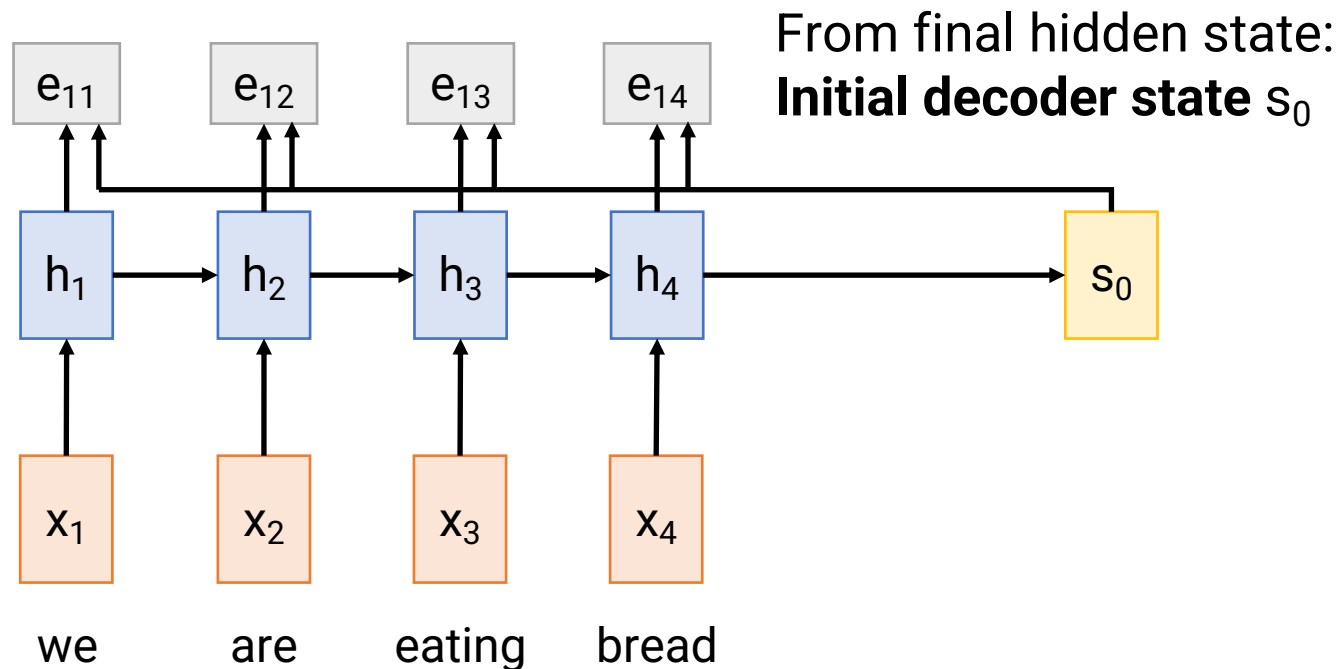




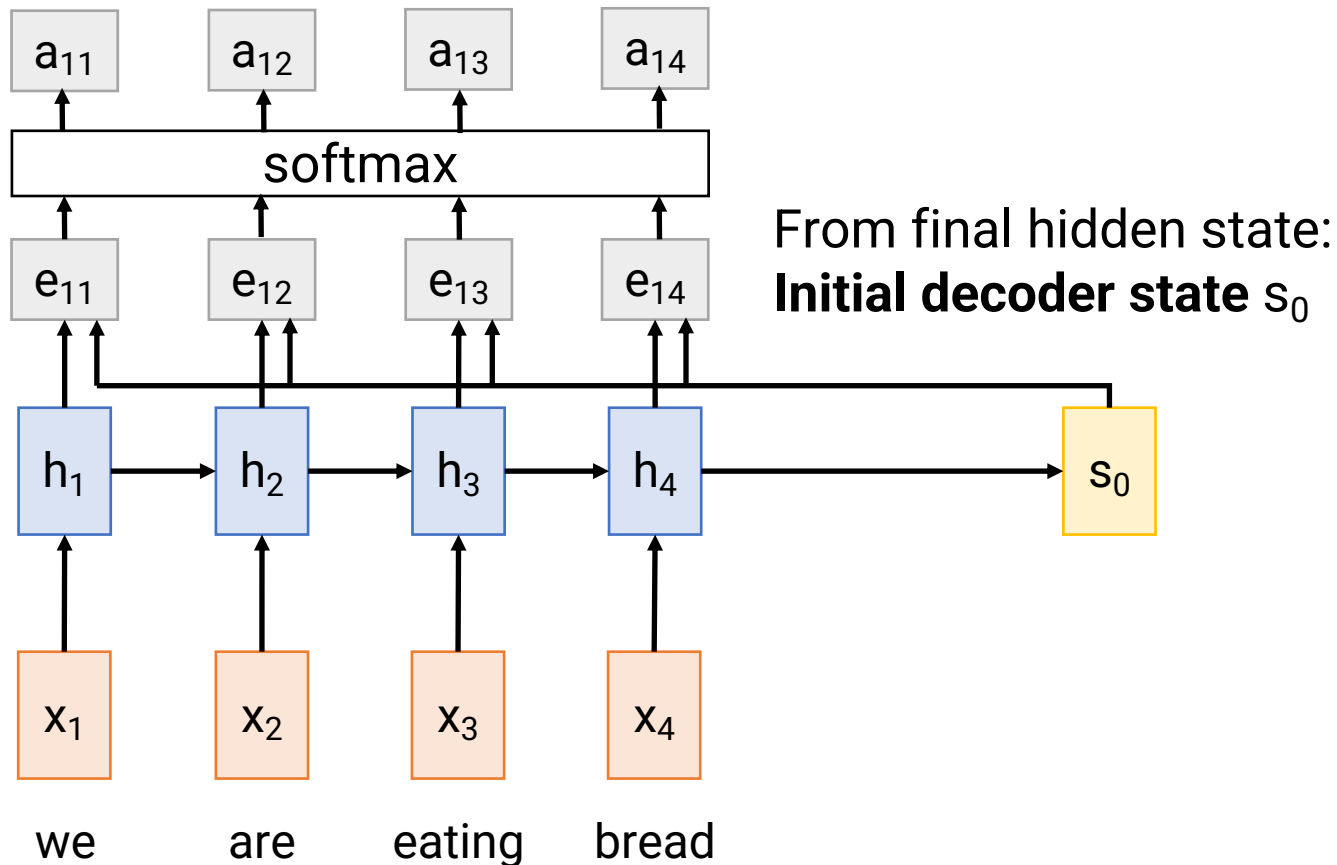
# Machine Translation with RNNs **and Attention**

Compute **alignment scores**

$$e_{t,i} = f_{\text{att}}(s_{t-1}, h_i) \quad (f_{\text{att}} \text{ is an MLP})$$



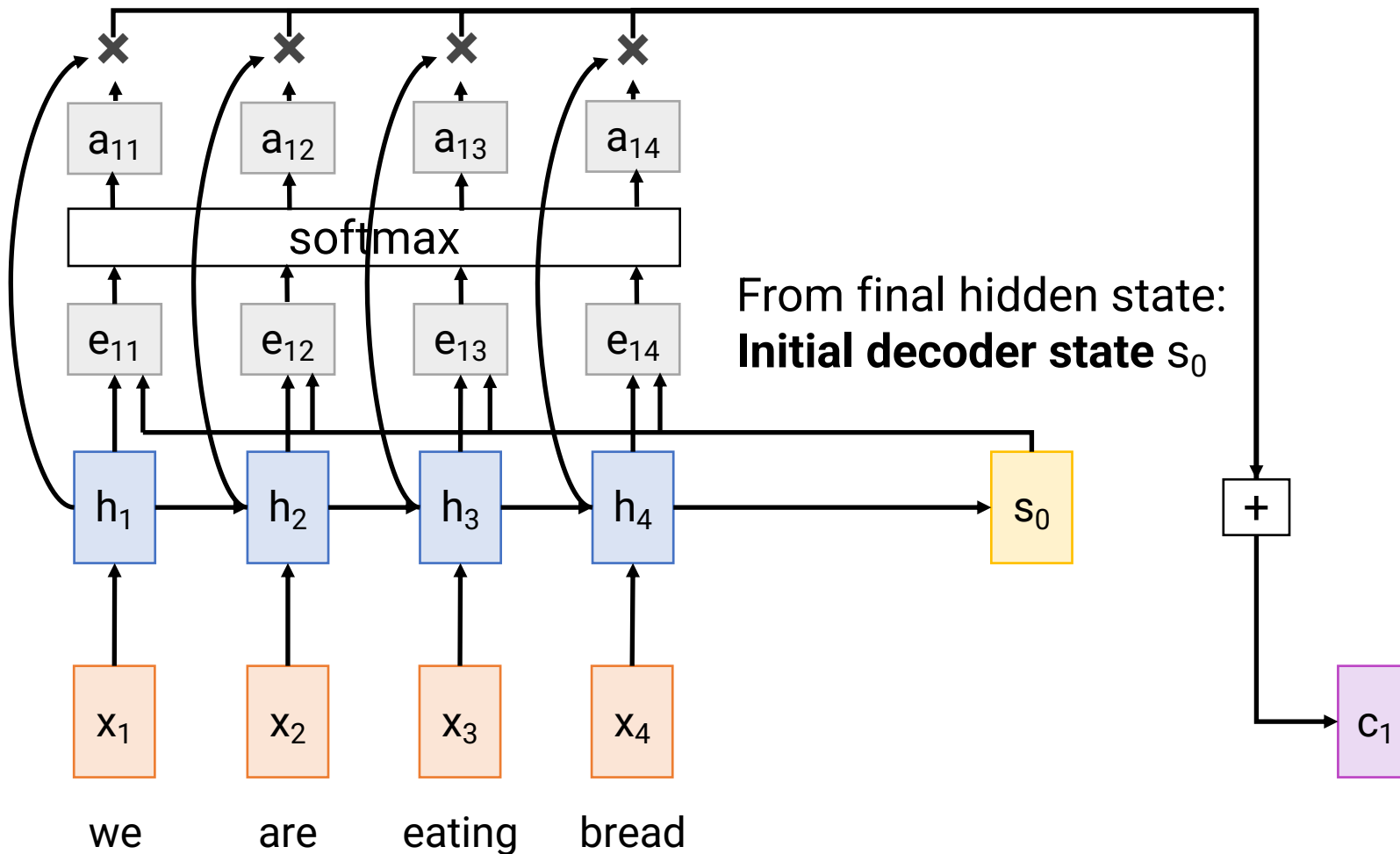
# Machine Translation with RNNs **and Attention**



Compute **alignment scores**  
 $e_{t,i} = f_{\text{att}}(s_{t-1}, h_i)$  ( $f_{\text{att}}$  is an MLP)

Normalize to get  
**attention weights**  
 $0 < a_{t,i} < 1 \quad \sum_i a_{t,i} = 1$

# Machine Translation with RNNs **and Attention**

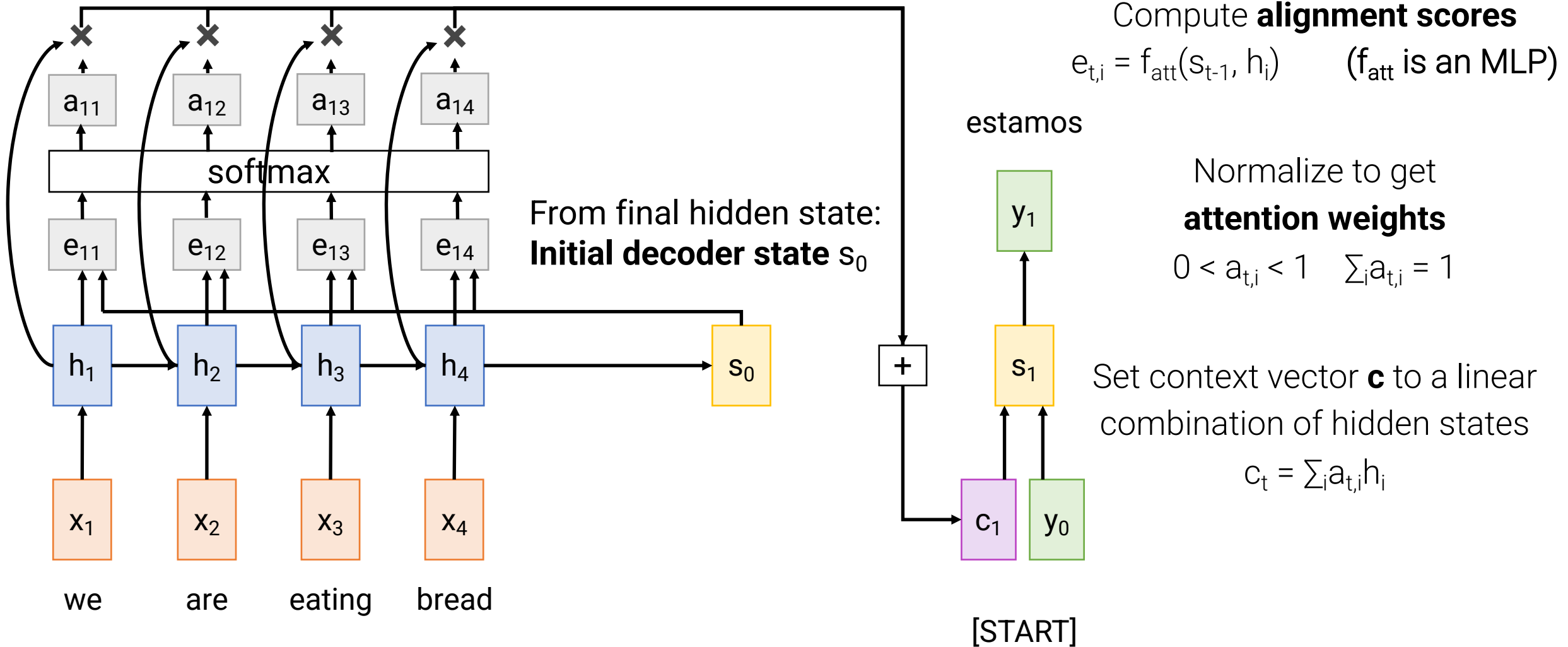


Compute **alignment scores**  
 $e_{t,i} = f_{\text{att}}(s_{t-1}, h_i)$  ( $f_{\text{att}}$  is an MLP)

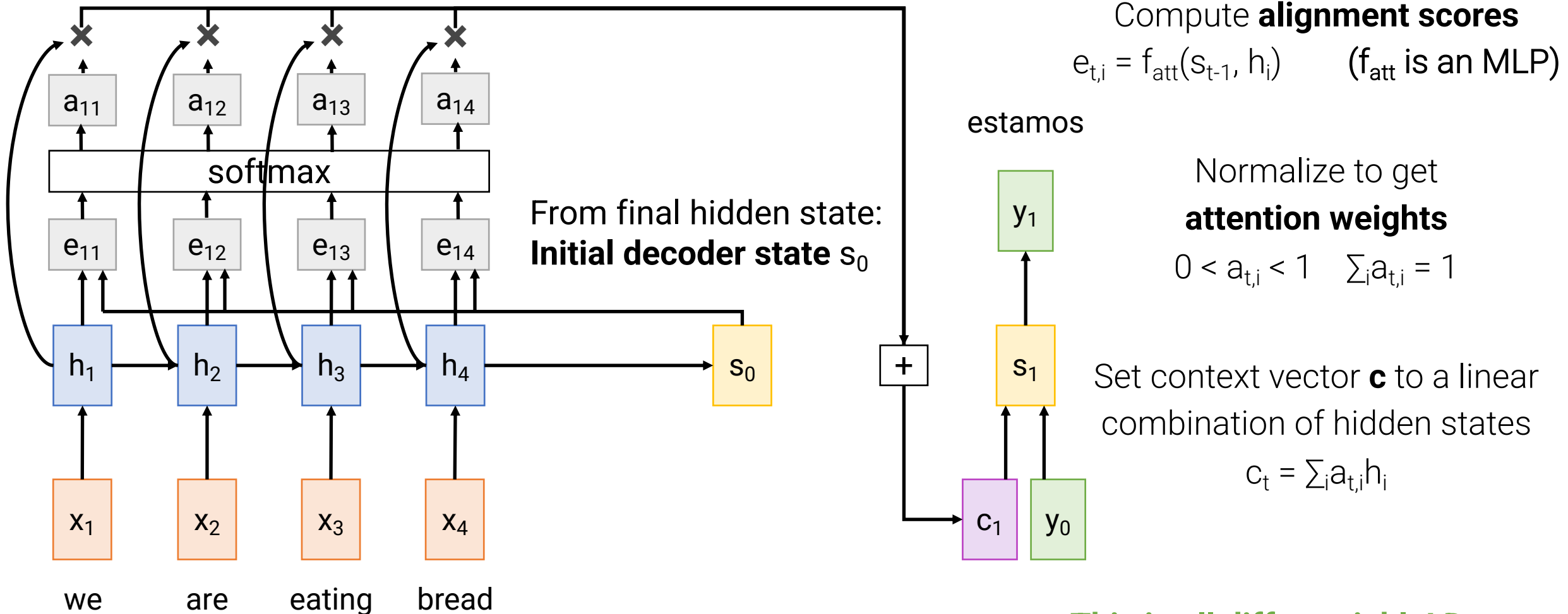
Normalize to get  
**attention weights**  
 $0 < a_{t,i} < 1 \quad \sum_i a_{t,i} = 1$

Set context vector **c** to a linear  
combination of hidden states  
 $c_t = \sum_i a_{t,i} h_i$

# Machine Translation with RNNs **and Attention**

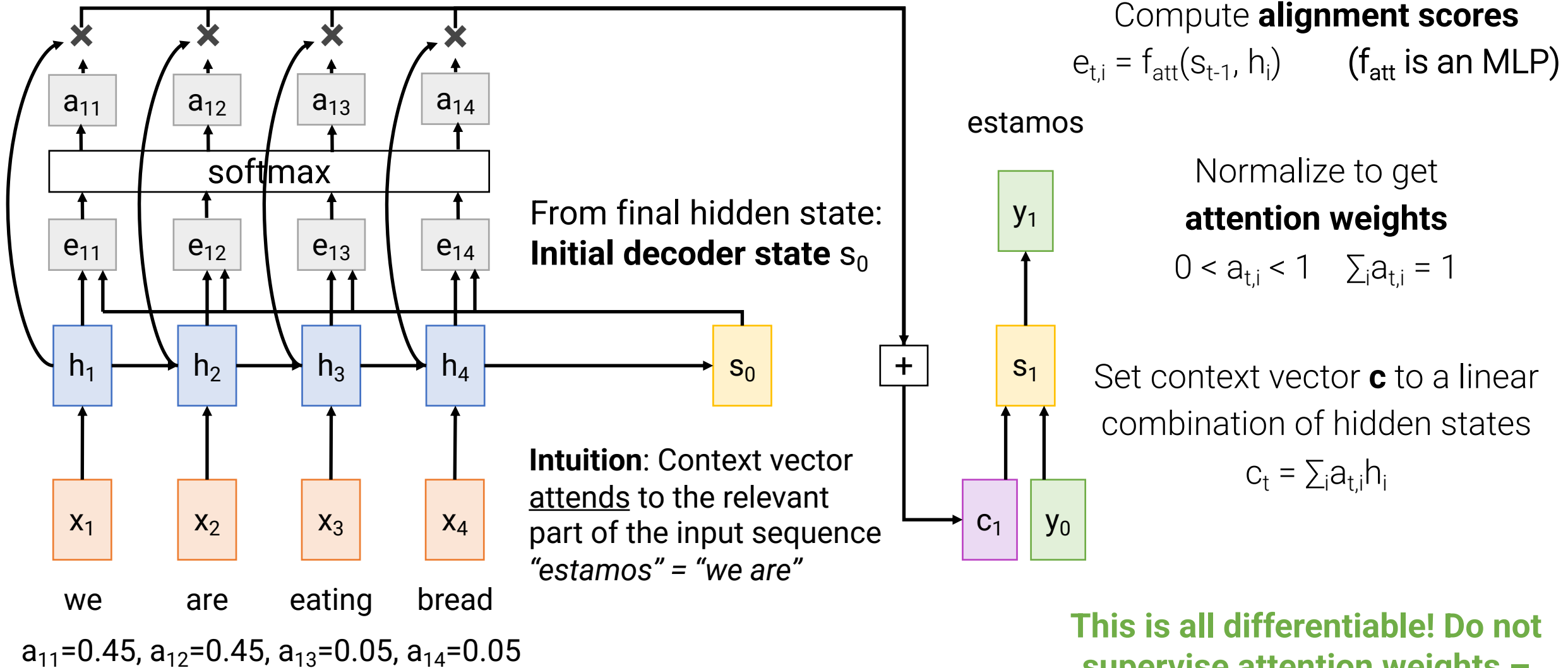


# Machine Translation with RNNs **and Attention**

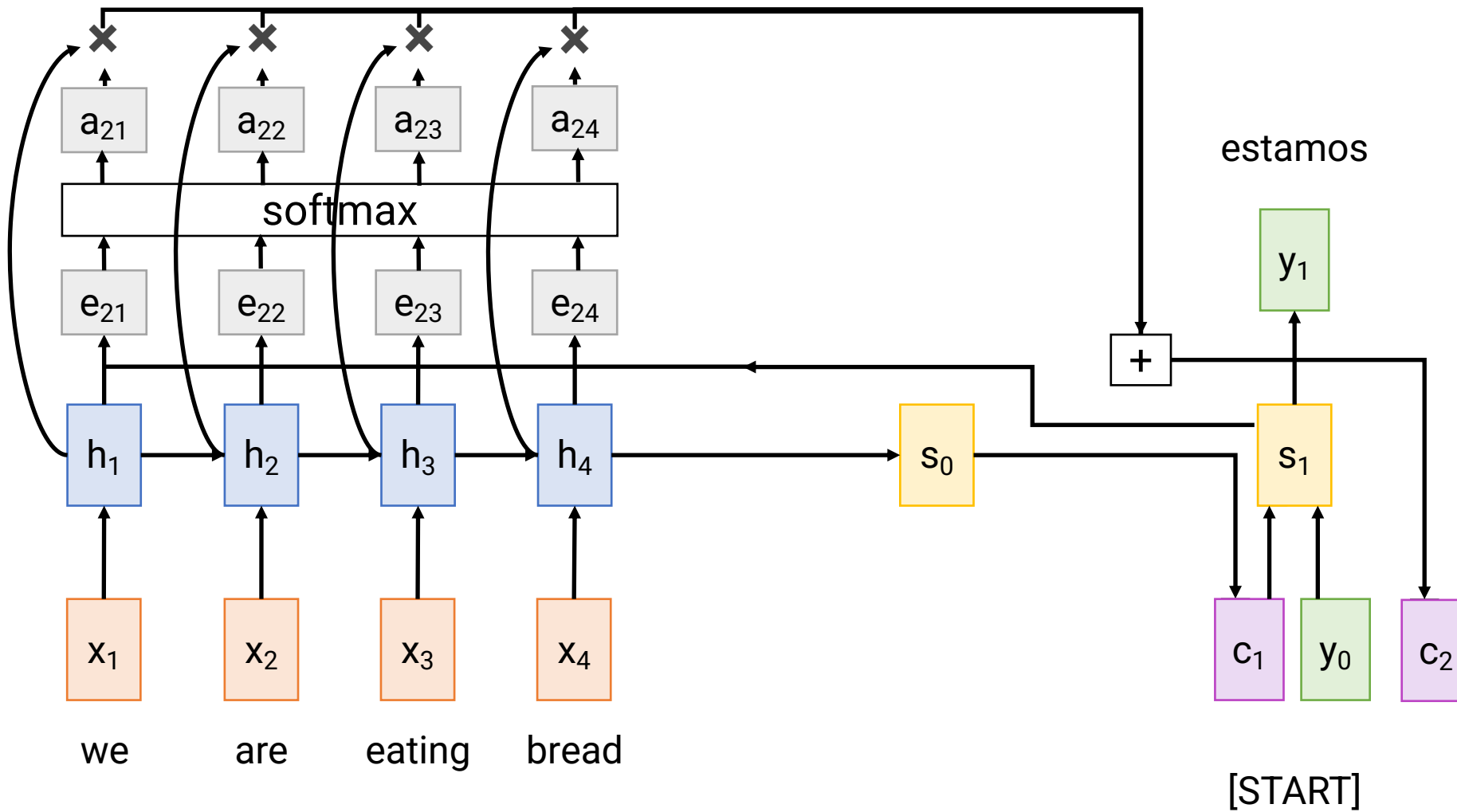


**This is all differentiable! Do not supervise attention weights – backprop through everything**

# Machine Translation with RNNs **and Attention**

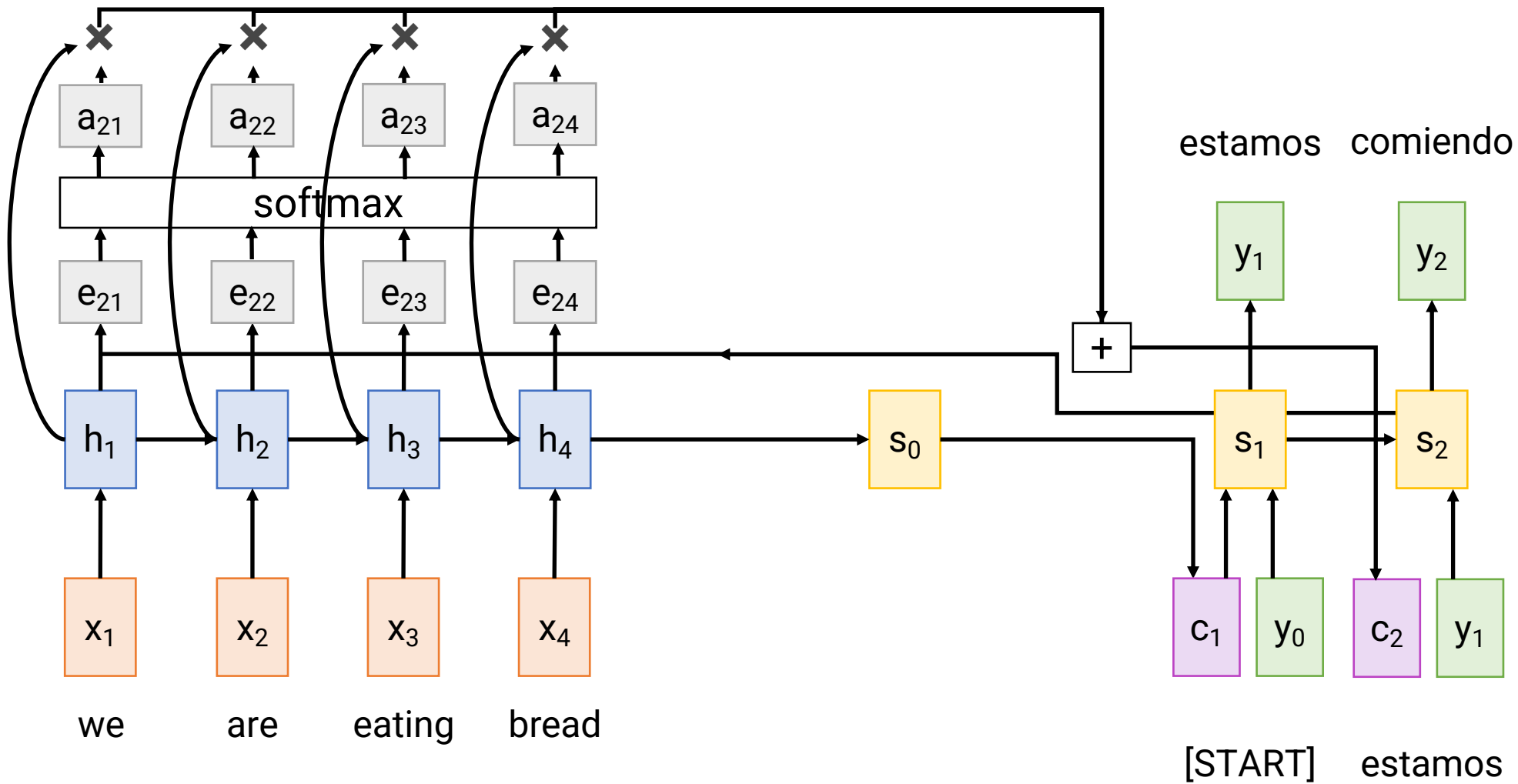


# Machine Translation with RNNs and Attention



Repeat: Use  $s_1$  to compute new context vector  $c_2$

# Machine Translation with RNNs and Attention

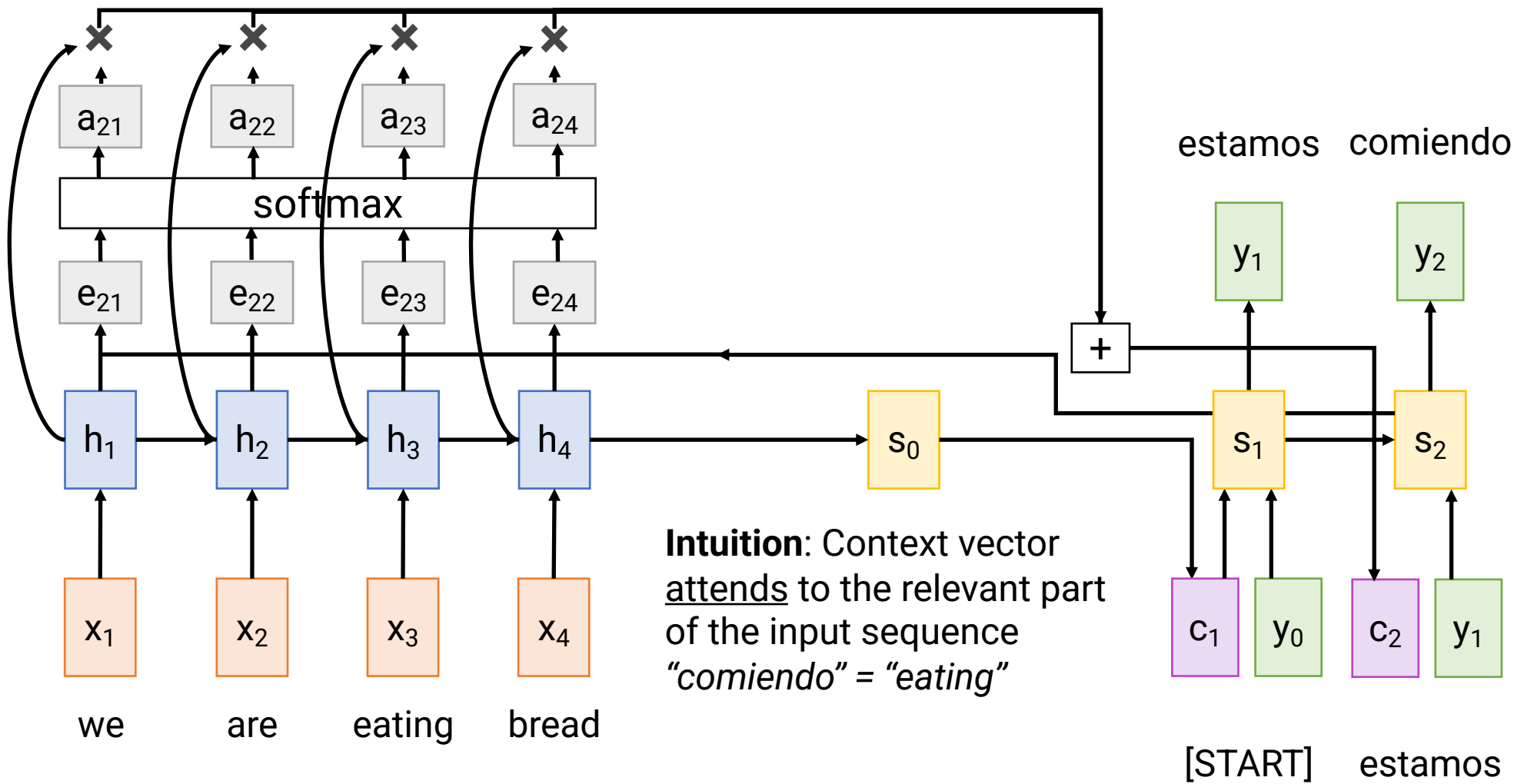


Repeat: Use  $s_1$  to compute new context vector  $c_2$

Use  $c_2$  to compute  $s_2, y_2$



# Machine Translation with RNNs **and Attention**



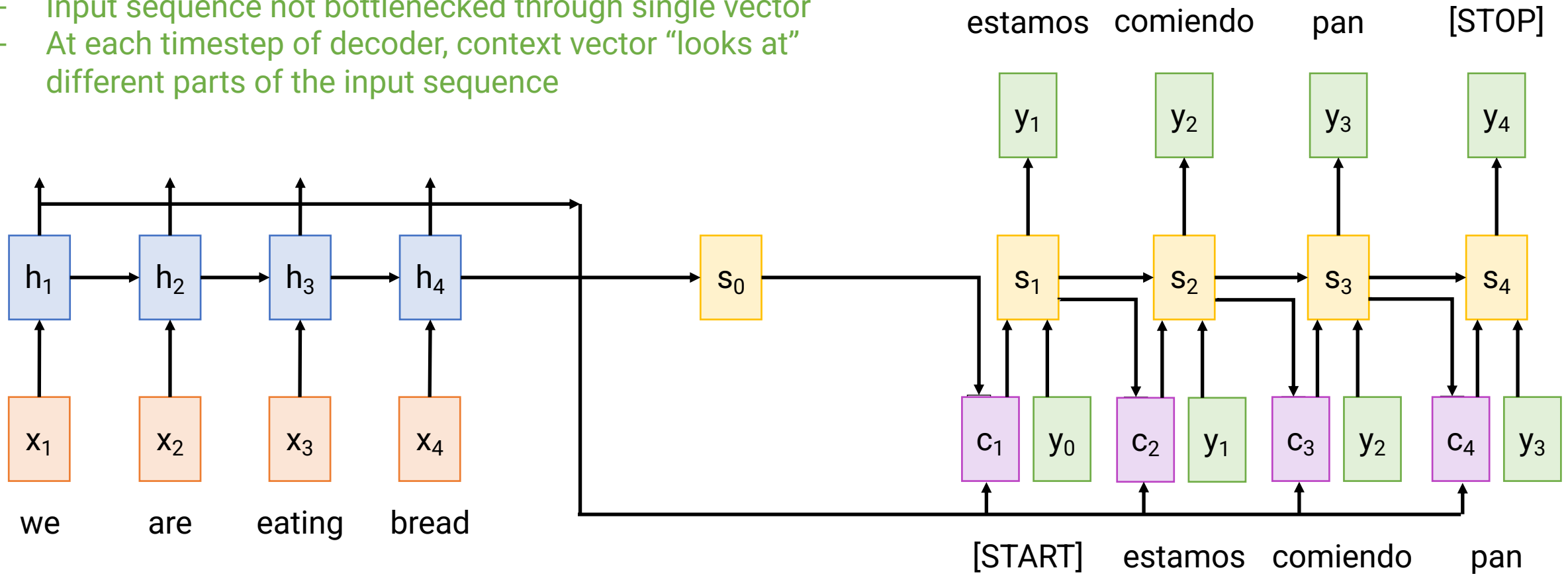
**Intuition:** Context vector attends to the relevant part of the input sequence  
*"comiendo" = "eating"*

Repeat: Use  $s_1$  to compute new context vector  $c_2$

Use  $c_2$  to compute  $s_2, y_2$

# Machine Translation with RNNs **and Attention**

- Use a different context vector in each timestep of decoder
- Input sequence not bottlenecked through single vector
  - At each timestep of decoder, context vector “looks at” different parts of the input sequence



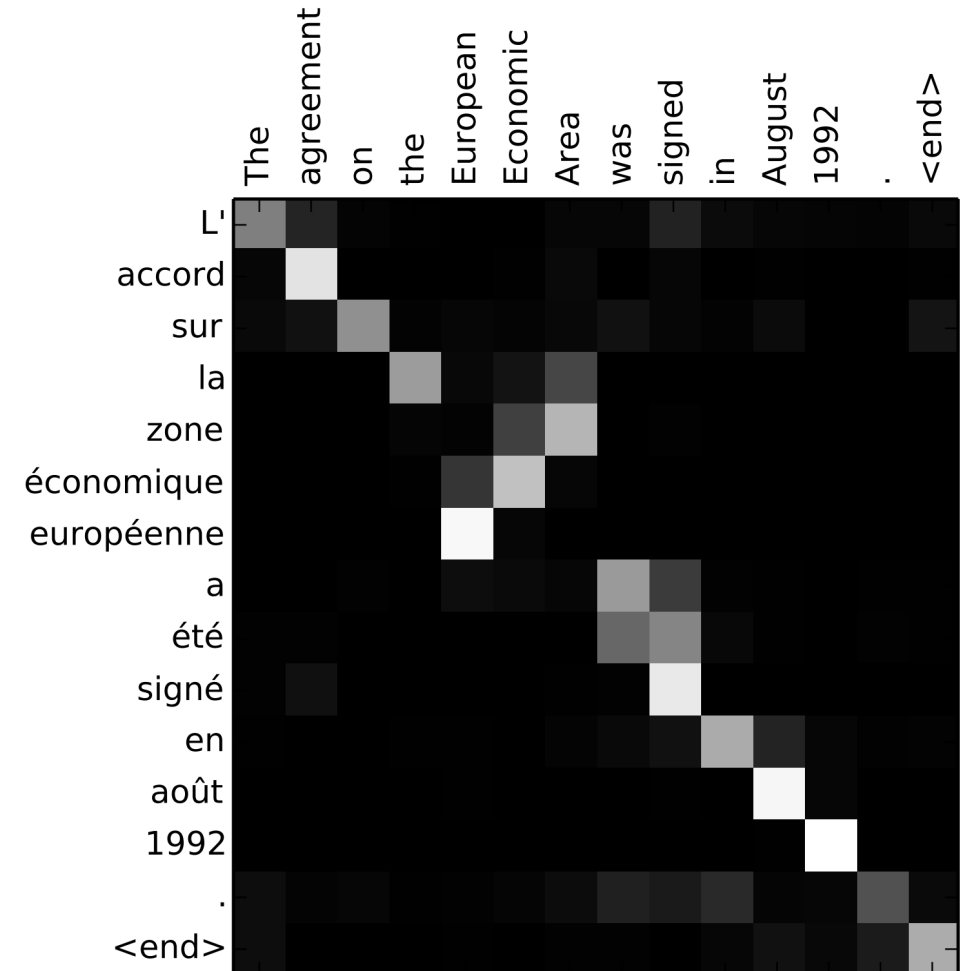
# Machine Translation with RNNs **and Attention**

**Example:** English to French translation

**Input:** “The agreement on the European Economic Area was signed in August 1992.”

**Output:** “L’accord sur la zone économique européenne a été signé en août 1992.”

Visualize attention weights  $a_{t,i}$



# Machine Translation with RNNs **and Attention**

**Example:** English to French translation

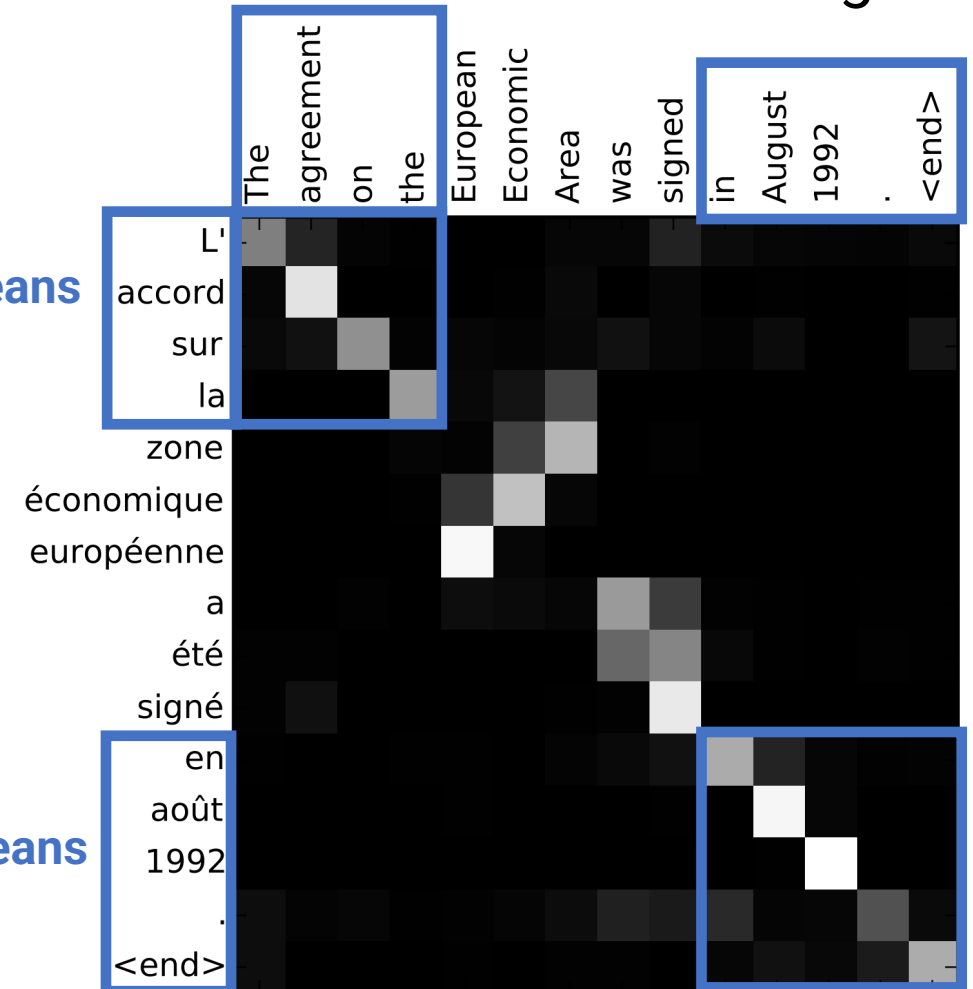
**Input:** “**The agreement on the** European Economic Area was signed **in August 1992.**”

**Output:** “**L'accord sur la** zone économique européenne a été signé **en août 1992.**”

Diagonal attention means words correspond in order

Diagonal attention means words correspond in order

Visualize attention weights  $a_{t,i}$



# Machine Translation with RNNs **and Attention**

**Example:** English to French translation

**Input:** “**The agreement on the European Economic Area** was signed **in August 1992.**”

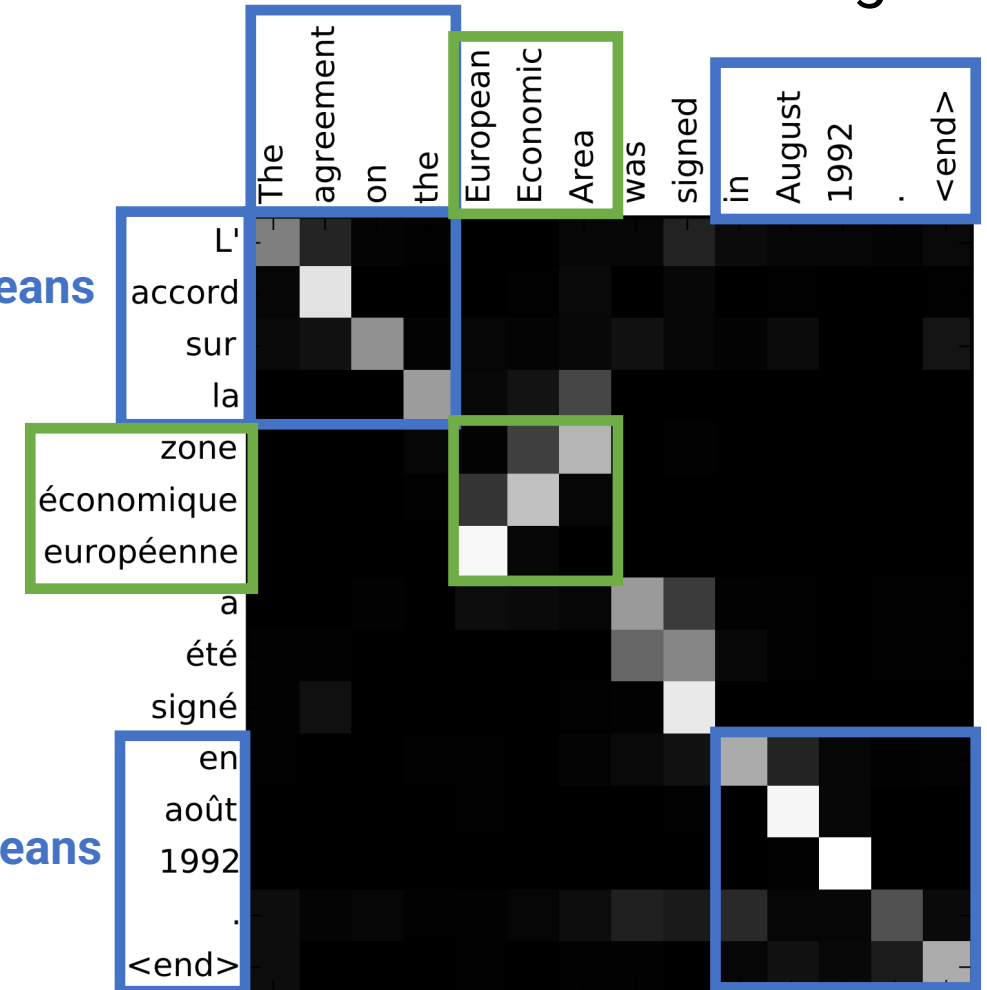
**Output:** “**L'accord sur la zone économique européenne** a été signé **en août 1992.**”

Diagonal attention means words correspond in order

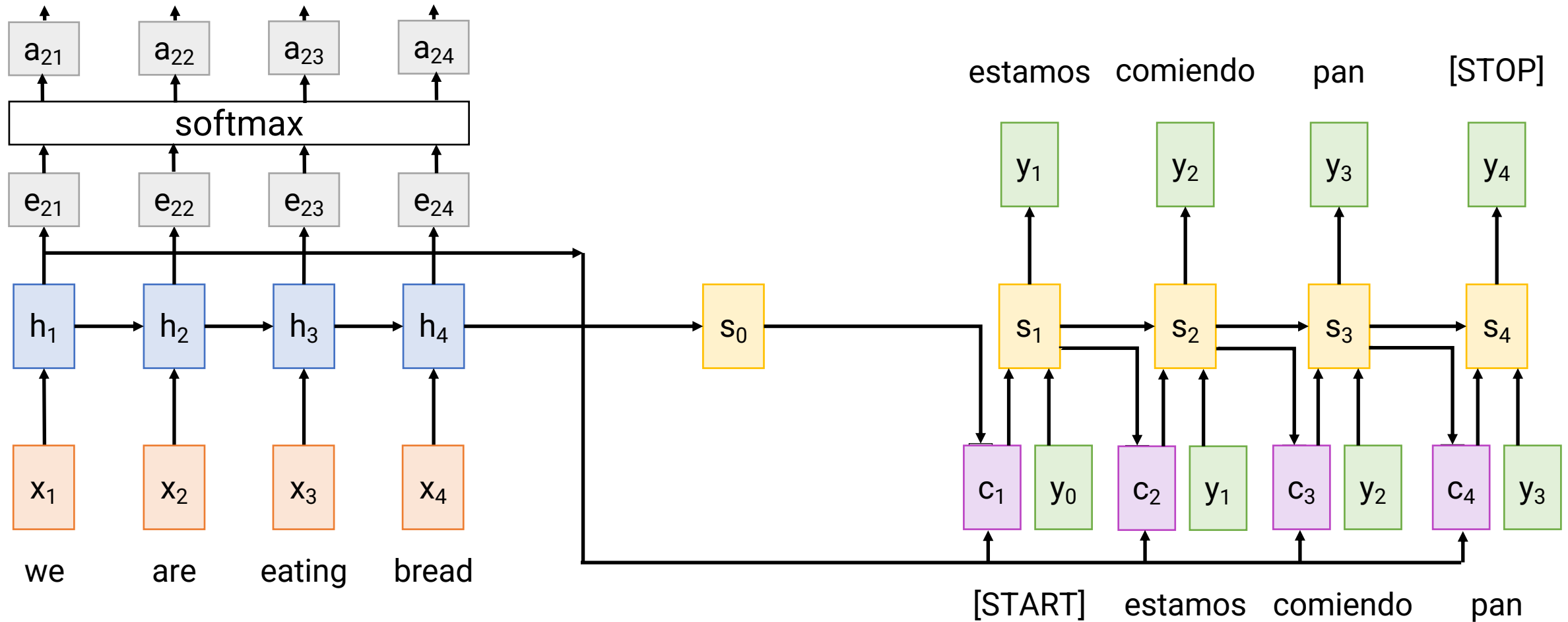
Attention figures out different word orders

Diagonal attention means words correspond in order

Visualize attention weights  $a_{t,i}$



# Machine Translation with RNNs and Attention



# Attention Layer

## Inputs:

**State vector:**  $\mathbf{s}_i$  (Shape:  $D_Q$ )

**Hidden vectors:**  $\mathbf{h}_i$  (Shape:  $N_X \times D_H$ )

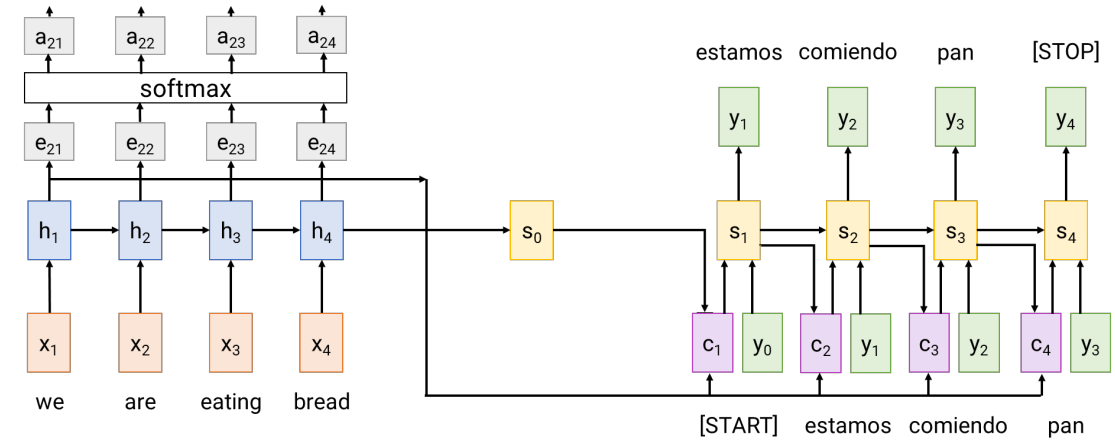
**Similarity function:**  $f_{\text{att}}$

## Computation:

**Similarities:**  $\mathbf{e}$  (Shape:  $N_X$ )  $e_i = f_{\text{att}}(\mathbf{s}_{t-1}, \mathbf{h}_i)$

**Attention weights:**  $\mathbf{a} = \text{softmax}(\mathbf{e})$  (Shape:  $N_X$ )

**Output vector:**  $\mathbf{y} = \sum_i a_i \mathbf{h}_i$  (Shape:  $D_X$ )



# Attention Layer

## Inputs:

**Query vector:**  $\mathbf{q}$  (Shape:  $D_Q$ )

**Input vectors:**  $\mathbf{X}$  (Shape:  $N_X \times D_X$ )

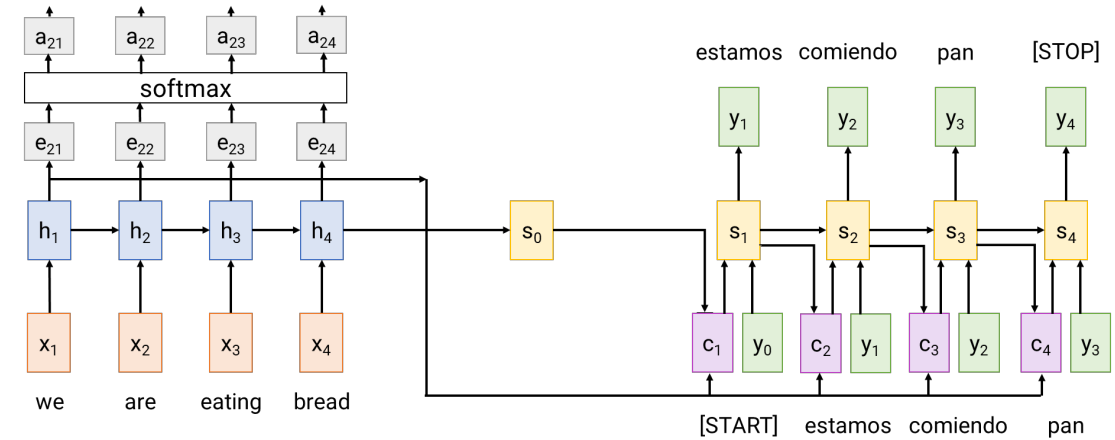
**Similarity function:**  $f_{\text{att}}$

## Computation:

**Similarities:**  $\mathbf{e}$  (Shape:  $N_X$ )  $e_i = f_{\text{att}}(\mathbf{q}, \mathbf{X}_i)$

**Attention weights:**  $\mathbf{a} = \text{softmax}(\mathbf{e})$  (Shape:  $N_X$ )

**Output vector:**  $\mathbf{y} = \sum_i a_i \mathbf{X}_i$  (Shape:  $D_X$ )





# Attention Layer

## Inputs:

**Query vector:**  $\mathbf{q}$  (Shape:  $D_Q$ )

**Input vectors:**  $\mathbf{X}$  (Shape:  $N_X \times D_Q$ )

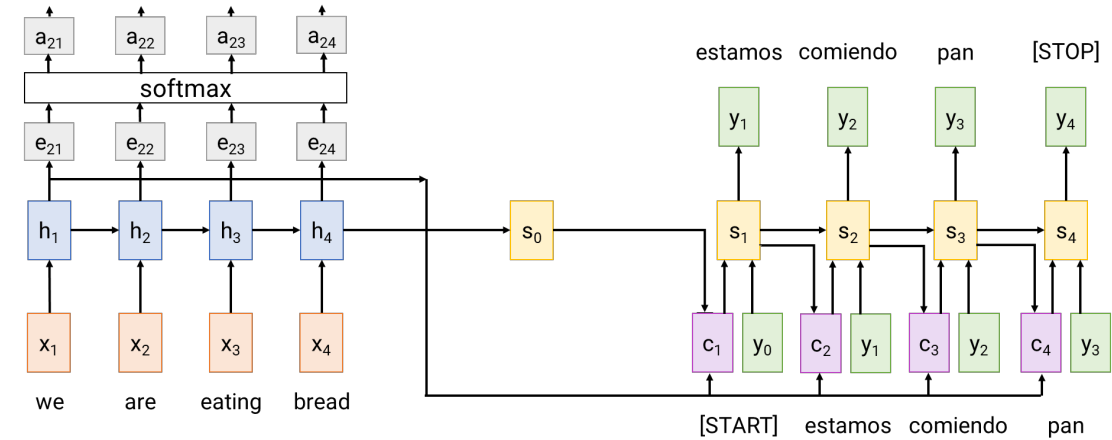
**Similarity function:** dot product

## Computation:

**Similarities:**  $\mathbf{e}$  (Shape:  $N_X$ )  $e_i = \mathbf{q} \cdot \mathbf{X}_i$

**Attention weights:**  $\mathbf{a} = \text{softmax}(\mathbf{e})$  (Shape:  $N_X$ )

**Output vector:**  $\mathbf{y} = \sum_i a_i \mathbf{X}_i$  (Shape:  $D_X$ )



Changes:

- Use dot product for similarity

# Attention Layer

## Inputs:

**Query vector:**  $\mathbf{q}$  (Shape:  $D_Q$ )

**Input vectors:**  $\mathbf{X}$  (Shape:  $N_X \times D_X$ )

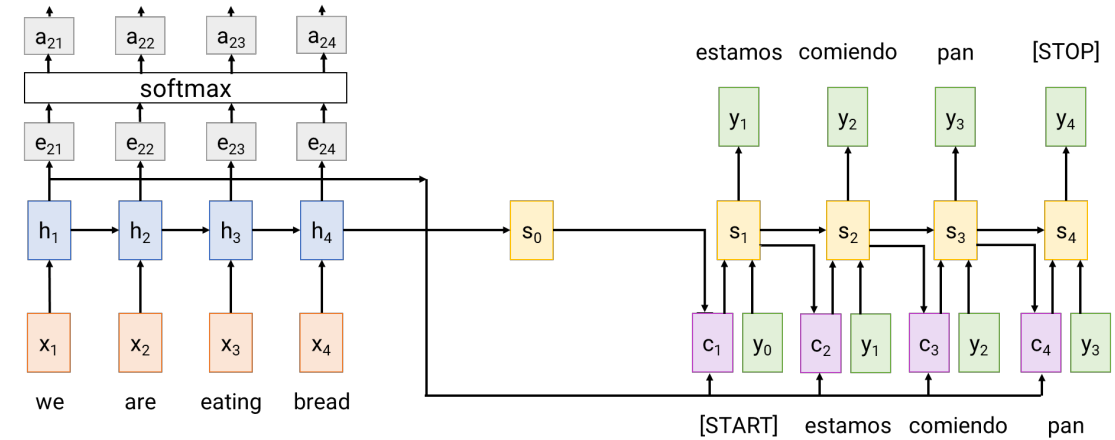
**Similarity function:** scaled dot product

## Computation:

**Similarities:**  $\mathbf{e}$  (Shape:  $N_X$ )  $e_i = \mathbf{q} \cdot \mathbf{X}_i / \text{sqrt}(D_Q)$

**Attention weights:**  $\mathbf{a} = \text{softmax}(\mathbf{e})$  (Shape:  $N_X$ )

**Output vector:**  $\mathbf{y} = \sum_i a_i \mathbf{X}_i$  (Shape:  $D_X$ )



Changes:

- Use **scaled** dot product for similarity

# Attention Layer

## Inputs:

Query vectors: **Q** (Shape:  $N_Q \times D_Q$ )

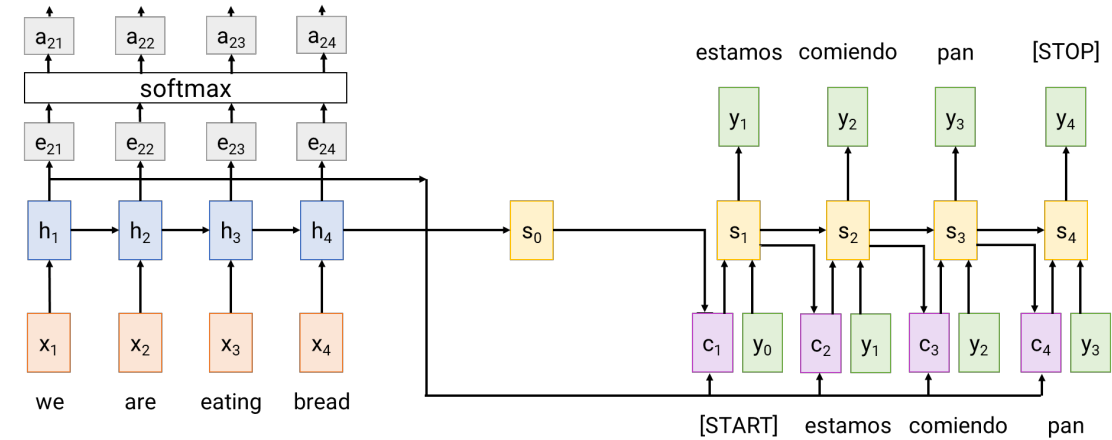
Input vectors: **X** (Shape:  $N_X \times D_Q$ )

## Computation:

Similarities:  $E = QX^T$  (Shape:  $N_Q \times N_X$ )  $E_{ij} = Q_i \cdot X_j / \text{sqrt}(D_Q)$

Attention weights:  $A = \text{softmax}(E, \text{dim}=1)$  (Shape:  $N_Q \times N_X$ )

Output vectors:  $Y = AX$  (Shape:  $N_Q \times D_X$ )  $Y_i = \sum_j A_{i,j} X_j$



## Changes:

- Use dot product for similarity
- Multiple **query** vectors

# Attention Layer

## Inputs:

**Query vectors:**  $\mathbf{Q}$  (Shape:  $N_Q \times D_Q$ )

**Input vectors:**  $\mathbf{X}$  (Shape:  $N_X \times D_X$ )

**Key matrix:**  $\mathbf{W}_K$  (Shape:  $D_X \times D_Q$ )

**Value matrix:**  $\mathbf{W}_V$  (Shape:  $D_X \times D_V$ )

## Computation:

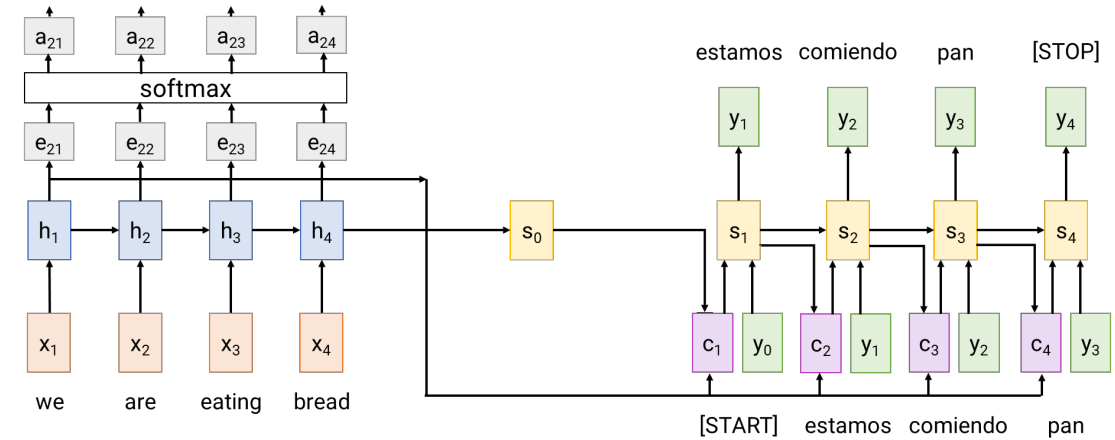
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

**Value vectors:**  $\mathbf{V} = \mathbf{XW}_V$  (Shape:  $N_X \times D_V$ )

**Similarities:**  $\mathbf{E} = \mathbf{QK}^T$  (Shape:  $N_Q \times N_X$ )  $E_{ij} = \mathbf{Q}_i \cdot \mathbf{K}_j / \text{sqrt}(D_Q)$

**Attention weights:**  $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$  (Shape:  $N_Q \times N_X$ )

**Output vectors:**  $\mathbf{Y} = \mathbf{AV}$  (Shape:  $N_Q \times D_V$ )  $Y_i = \sum_j A_{ij} \mathbf{V}_j$



## Changes:

- Use dot product for similarity
- Multiple **query** vectors
- Separate **key** and **value**

# Attention Layer

## Inputs:

**Query vectors:**  $\mathbf{Q}$  (Shape:  $N_Q \times D_Q$ )

**Input vectors:**  $\mathbf{X}$  (Shape:  $N_X \times D_X$ )

**Key matrix:**  $\mathbf{W}_K$  (Shape:  $D_X \times D_Q$ )

**Value matrix:**  $\mathbf{W}_V$  (Shape:  $D_X \times D_V$ )

## Computation:

**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

**Value vectors:**  $\mathbf{V} = \mathbf{XW}_V$  (Shape:  $N_X \times D_V$ )

**Similarities:**  $\mathbf{E} = \mathbf{QK}^T$  (Shape:  $N_Q \times N_X$ )  $E_{ij} = \mathbf{Q}_i \cdot \mathbf{K}_j / \text{sqrt}(D_Q)$

**Attention weights:**  $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$  (Shape:  $N_Q \times N_X$ )

**Output vectors:**  $\mathbf{Y} = \mathbf{AV}$  (Shape:  $N_Q \times D_V$ )  $Y_i = \sum_j A_{ij} \mathbf{V}_j$

$X_1$

$X_2$

$X_3$

$Q_1$

$Q_2$

$Q_3$

$Q_4$

# Attention Layer

## Inputs:

**Query vectors:**  $\mathbf{Q}$  (Shape:  $N_Q \times D_Q$ )

**Input vectors:**  $\mathbf{X}$  (Shape:  $N_X \times D_X$ )

**Key matrix:**  $\mathbf{W}_K$  (Shape:  $D_X \times D_Q$ )

**Value matrix:**  $\mathbf{W}_V$  (Shape:  $D_X \times D_V$ )

## Computation:

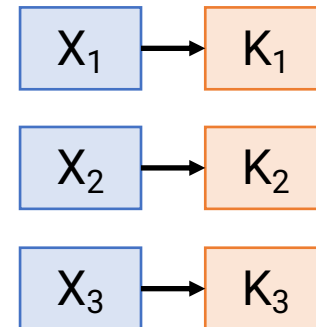
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

**Value vectors:**  $\mathbf{V} = \mathbf{XW}_V$  (Shape:  $N_X \times D_V$ )

**Similarities:**  $\mathbf{E} = \mathbf{QK}^T$  (Shape:  $N_Q \times N_X$ )  $E_{ij} = \mathbf{Q}_i \cdot \mathbf{K}_j / \text{sqrt}(D_Q)$

**Attention weights:**  $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$  (Shape:  $N_Q \times N_X$ )

**Output vectors:**  $\mathbf{Y} = \mathbf{AV}$  (Shape:  $N_Q \times D_V$ )  $Y_i = \sum_j A_{ij} \mathbf{V}_j$



# Attention Layer

## Inputs:

**Query vectors:**  $\mathbf{Q}$  (Shape:  $N_Q \times D_Q$ )

**Input vectors:**  $\mathbf{X}$  (Shape:  $N_X \times D_X$ )

**Key matrix:**  $\mathbf{W}_K$  (Shape:  $D_X \times D_Q$ )

**Value matrix:**  $\mathbf{W}_V$  (Shape:  $D_X \times D_V$ )

## Computation:

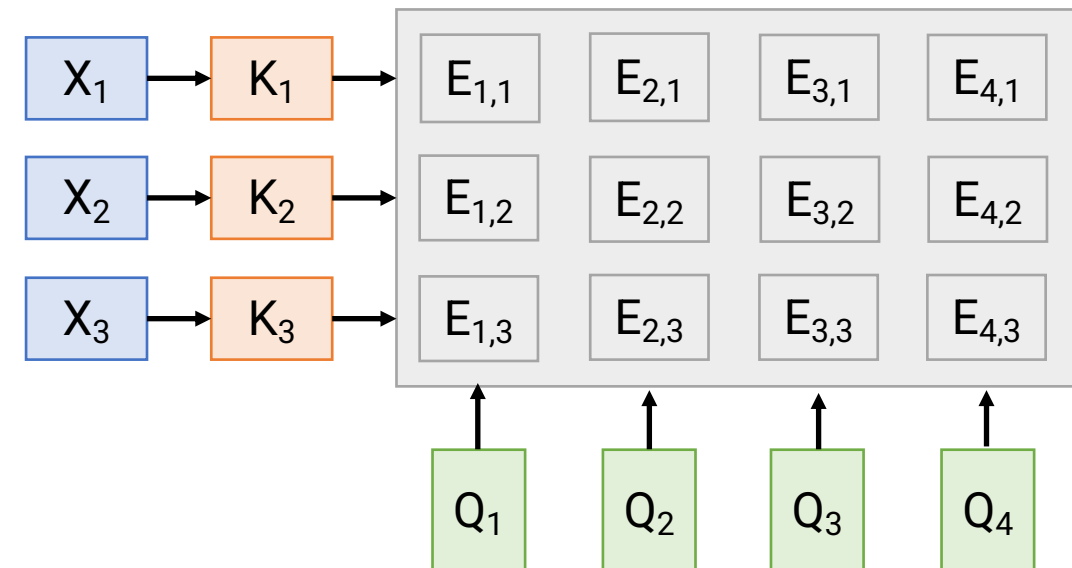
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

**Value vectors:**  $\mathbf{V} = \mathbf{XW}_V$  (Shape:  $N_X \times D_V$ )

**Similarities:**  $\mathbf{E} = \mathbf{QK}^T$  (Shape:  $N_Q \times N_X$ )  $E_{i,j} = \mathbf{Q}_i \cdot \mathbf{K}_j / \text{sqrt}(D_Q)$

**Attention weights:**  $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$  (Shape:  $N_Q \times N_X$ )

**Output vectors:**  $\mathbf{Y} = \mathbf{AV}$  (Shape:  $N_Q \times D_V$ )  $Y_i = \sum_j A_{i,j} \mathbf{V}_j$



# Attention Layer

## Inputs:

**Query vectors:**  $\mathbf{Q}$  (Shape:  $N_Q \times D_Q$ )

**Input vectors:**  $\mathbf{X}$  (Shape:  $N_X \times D_X$ )

**Key matrix:**  $\mathbf{W}_K$  (Shape:  $D_X \times D_Q$ )

**Value matrix:**  $\mathbf{W}_V$  (Shape:  $D_X \times D_V$ )

## Computation:

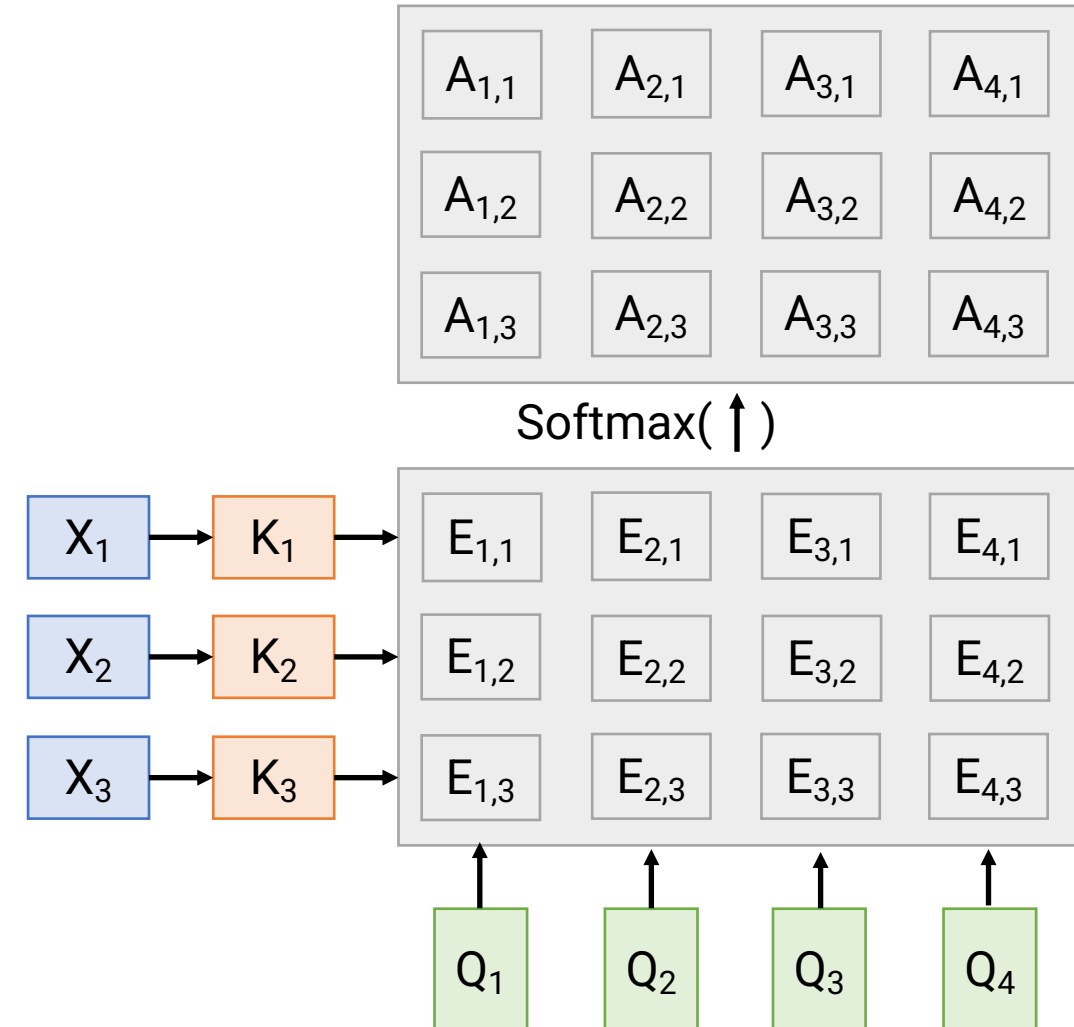
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

**Value vectors:**  $\mathbf{V} = \mathbf{XW}_V$  (Shape:  $N_X \times D_V$ )

**Similarities:**  $\mathbf{E} = \mathbf{QK}^T$  (Shape:  $N_Q \times N_X$ )  $E_{ij} = \mathbf{Q}_i \cdot \mathbf{K}_j / \text{sqrt}(D_Q)$

**Attention weights:**  $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$  (Shape:  $N_Q \times N_X$ )

**Output vectors:**  $\mathbf{Y} = \mathbf{AV}$  (Shape:  $N_Q \times D_V$ )  $Y_i = \sum_j A_{ij} \mathbf{V}_j$





# Attention Layer

## Inputs:

**Query vectors:**  $\mathbf{Q}$  (Shape:  $N_Q \times D_Q$ )

**Input vectors:**  $\mathbf{X}$  (Shape:  $N_X \times D_X$ )

**Key matrix:**  $\mathbf{W}_K$  (Shape:  $D_X \times D_Q$ )

**Value matrix:**  $\mathbf{W}_V$  (Shape:  $D_X \times D_V$ )

## Computation:

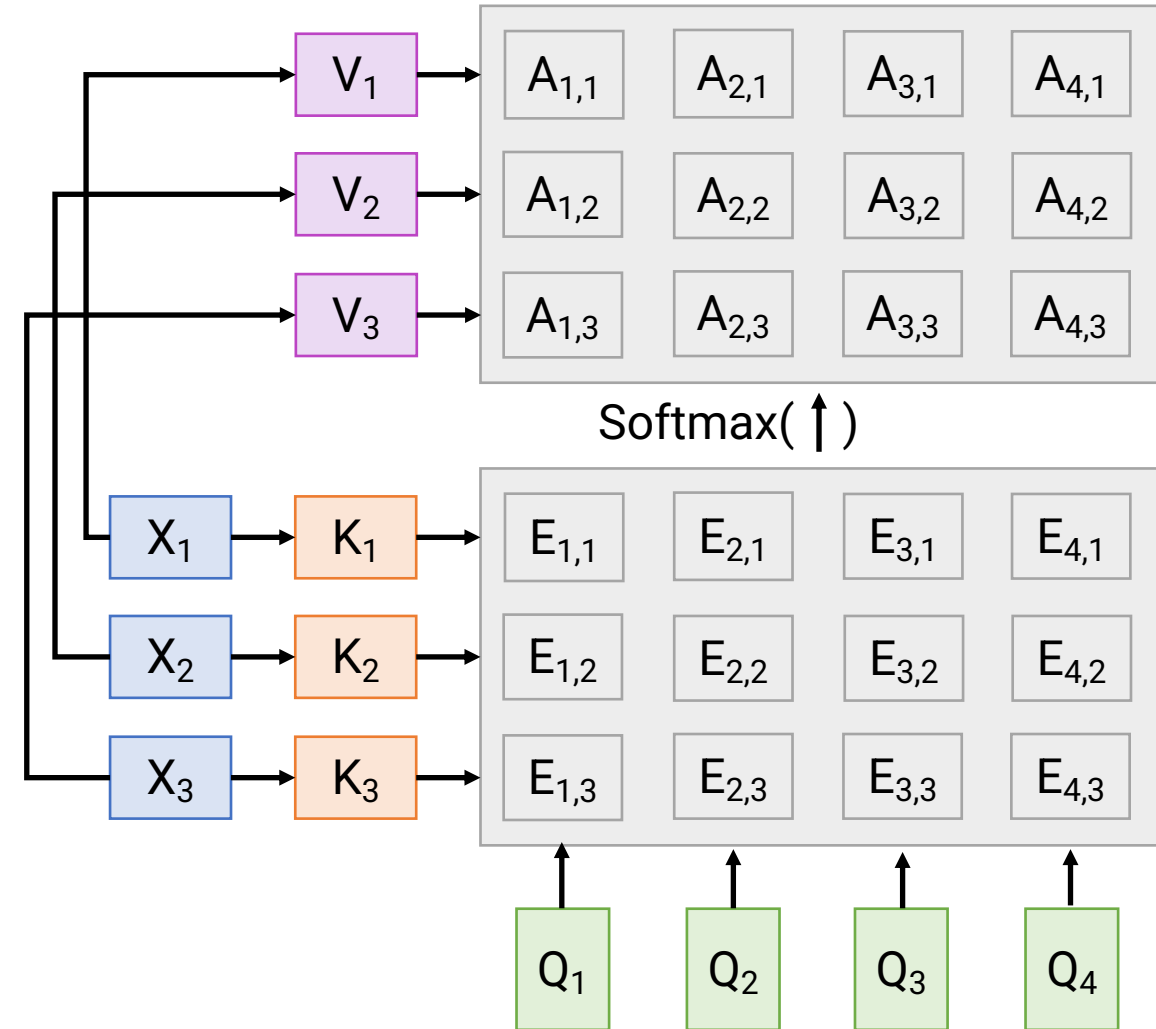
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

**Value vectors:**  $\mathbf{V} = \mathbf{XW}_V$  (Shape:  $N_X \times D_V$ )

**Similarities:**  $\mathbf{E} = \mathbf{QK}^T$  (Shape:  $N_Q \times N_X$ )  $E_{i,j} = \mathbf{Q}_i \cdot \mathbf{K}_j / \text{sqrt}(D_Q)$

**Attention weights:**  $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$  (Shape:  $N_Q \times N_X$ )

**Output vectors:**  $\mathbf{Y} = \mathbf{AV}$  (Shape:  $N_Q \times D_V$ )  $Y_i = \sum_j A_{i,j} \mathbf{V}_j$



# Attention Layer

## Inputs:

**Query vectors:**  $\mathbf{Q}$  (Shape:  $N_Q \times D_Q$ )

**Input vectors:**  $\mathbf{X}$  (Shape:  $N_X \times D_X$ )

**Key matrix:**  $\mathbf{W}_K$  (Shape:  $D_X \times D_Q$ )

**Value matrix:**  $\mathbf{W}_V$  (Shape:  $D_X \times D_V$ )

## Computation:

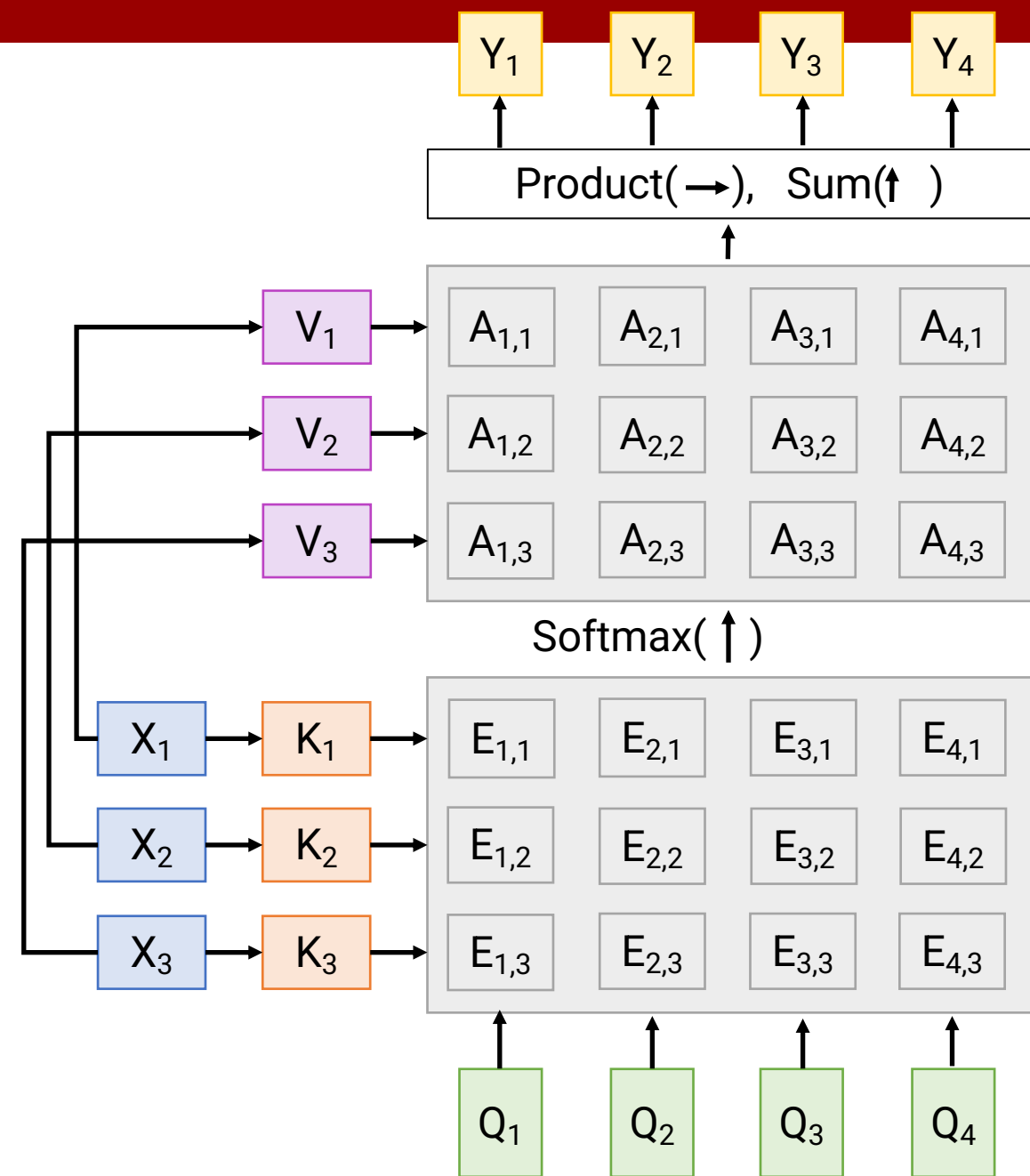
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

**Value vectors:**  $\mathbf{V} = \mathbf{XW}_V$  (Shape:  $N_X \times D_V$ )

**Similarities:**  $\mathbf{E} = \mathbf{QK}^T$  (Shape:  $N_Q \times N_X$ )  $E_{i,j} = \mathbf{Q}_i \cdot \mathbf{K}_j / \text{sqrt}(D_Q)$

**Attention weights:**  $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$  (Shape:  $N_Q \times N_X$ )

**Output vectors:**  $\mathbf{Y} = \mathbf{AV}$  (Shape:  $N_Q \times D_V$ )  $Y_i = \sum_j A_{i,j} \mathbf{V}_j$



# Self-Attention Layer

One **query** per **input vector**

## Inputs:

**Input vectors:**  $\mathbf{X}$  (Shape:  $N_X \times D_X$ )

**Key matrix:**  $\mathbf{W}_K$  (Shape:  $D_X \times D_Q$ )

**Value matrix:**  $\mathbf{W}_V$  (Shape:  $D_X \times D_V$ )

**Query matrix:**  $\mathbf{W}_Q$  (Shape:  $D_X \times D_Q$ )

## Computation:

**Query vectors:**  $\mathbf{Q} = \mathbf{XW}_Q$

**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

**Value vectors:**  $\mathbf{V} = \mathbf{XW}_V$  (Shape:  $N_X \times D_V$ )

**Similarities:**  $\mathbf{E} = \mathbf{QK}^T$  (Shape:  $N_X \times N_X$ )  $E_{ij} = \mathbf{Q}_i \cdot \mathbf{K}_j / \text{sqrt}(D_Q)$

**Attention weights:**  $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$  (Shape:  $N_X \times N_X$ )

**Output vectors:**  $\mathbf{Y} = \mathbf{AV}$  (Shape:  $N_X \times D_V$ )  $Y_i = \sum_j A_{ij} \mathbf{V}_j$

$X_1$

$X_2$

$X_3$

# Self-Attention Layer

One **query** per **input vector**

## Inputs:

**Input vectors:**  $\mathbf{X}$  (Shape:  $N_X \times D_X$ )

**Key matrix:**  $\mathbf{W}_K$  (Shape:  $D_X \times D_Q$ )

**Value matrix:**  $\mathbf{W}_V$  (Shape:  $D_X \times D_V$ )

**Query matrix:**  $\mathbf{W}_Q$  (Shape:  $D_X \times D_Q$ )

## Computation:

**Query vectors:**  $\mathbf{Q} = \mathbf{XW}_Q$

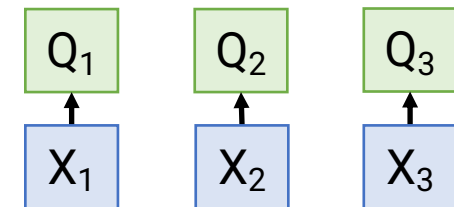
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

**Value vectors:**  $\mathbf{V} = \mathbf{XW}_V$  (Shape:  $N_X \times D_V$ )

**Similarities:**  $\mathbf{E} = \mathbf{QK}^T$  (Shape:  $N_X \times N_X$ )  $E_{ij} = \mathbf{Q}_i \cdot \mathbf{K}_j / \text{sqrt}(D_Q)$

**Attention weights:**  $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$  (Shape:  $N_X \times N_X$ )

**Output vectors:**  $\mathbf{Y} = \mathbf{AV}$  (Shape:  $N_X \times D_V$ )  $Y_i = \sum_j A_{i,j} \mathbf{V}_j$



# Self-Attention Layer

One **query** per **input vector**

## Inputs:

**Input vectors:**  $\mathbf{X}$  (Shape:  $N_X \times D_X$ )

**Key matrix:**  $\mathbf{W}_K$  (Shape:  $D_X \times D_Q$ )

**Value matrix:**  $\mathbf{W}_V$  (Shape:  $D_X \times D_V$ )

**Query matrix:**  $\mathbf{W}_Q$  (Shape:  $D_X \times D_Q$ )

## Computation:

**Query vectors:**  $\mathbf{Q} = \mathbf{XW}_Q$

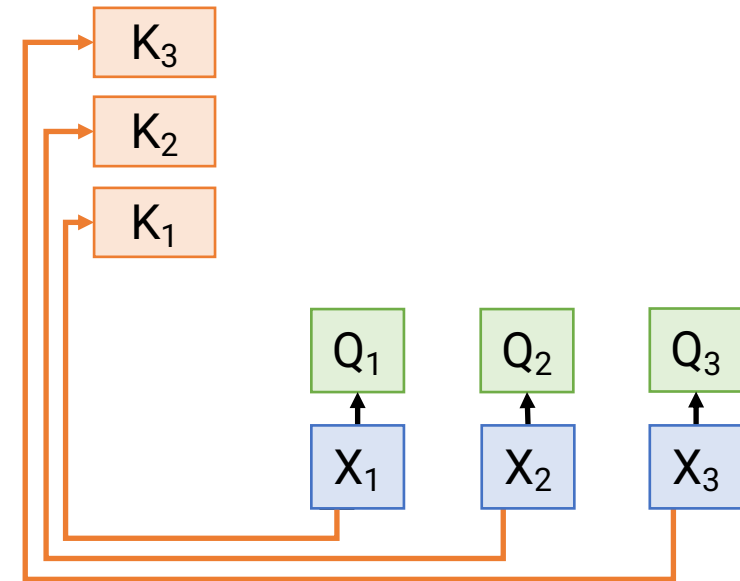
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

**Value vectors:**  $\mathbf{V} = \mathbf{XW}_V$  (Shape:  $N_X \times D_V$ )

**Similarities:**  $\mathbf{E} = \mathbf{QK}^T$  (Shape:  $N_X \times N_X$ )  $E_{ij} = \mathbf{Q}_i \cdot \mathbf{K}_j / \text{sqrt}(D_Q)$

**Attention weights:**  $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$  (Shape:  $N_X \times N_X$ )

**Output vectors:**  $\mathbf{Y} = \mathbf{AV}$  (Shape:  $N_X \times D_V$ )  $Y_i = \sum_j A_{i,j} \mathbf{V}_j$



# Self-Attention Layer

One **query** per **input vector**

## Inputs:

**Input vectors:**  $\mathbf{X}$  (Shape:  $N_X \times D_X$ )

**Key matrix:**  $\mathbf{W}_K$  (Shape:  $D_X \times D_Q$ )

**Value matrix:**  $\mathbf{W}_V$  (Shape:  $D_X \times D_V$ )

**Query matrix:**  $\mathbf{W}_Q$  (Shape:  $D_X \times D_Q$ )

## Computation:

**Query vectors:**  $\mathbf{Q} = \mathbf{XW}_Q$

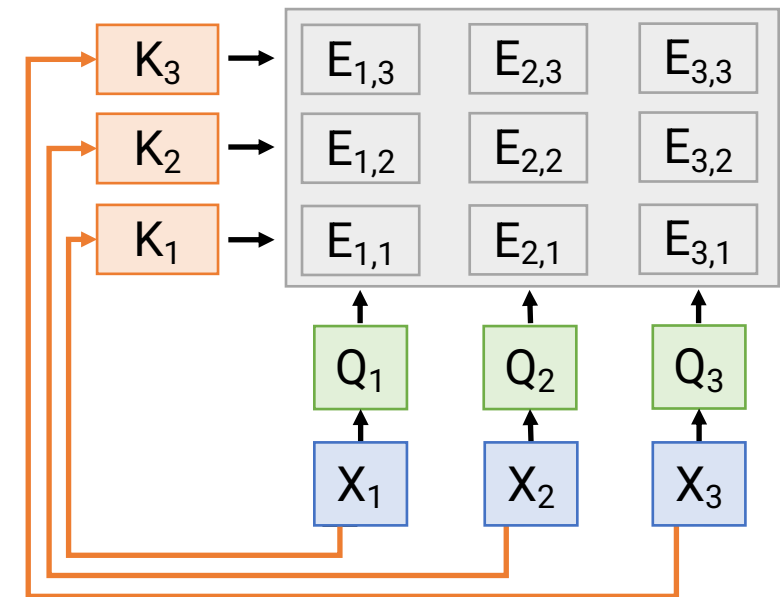
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

**Value vectors:**  $\mathbf{V} = \mathbf{XW}_V$  (Shape:  $N_X \times D_V$ )

**Similarities:**  $\mathbf{E} = \mathbf{QK}^T$  (Shape:  $N_X \times N_X$ )  $E_{ij} = \mathbf{Q}_i \cdot \mathbf{K}_j / \text{sqrt}(D_Q)$

**Attention weights:**  $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$  (Shape:  $N_X \times N_X$ )

**Output vectors:**  $\mathbf{Y} = \mathbf{AV}$  (Shape:  $N_X \times D_V$ )  $Y_i = \sum_j A_{i,j} \mathbf{V}_j$



# Self-Attention Layer

One **query** per **input vector**

## Inputs:

**Input vectors:**  $\mathbf{X}$  (Shape:  $N_X \times D_X$ )

**Key matrix:**  $\mathbf{W}_K$  (Shape:  $D_X \times D_Q$ )

**Value matrix:**  $\mathbf{W}_V$  (Shape:  $D_X \times D_V$ )

**Query matrix:**  $\mathbf{W}_Q$  (Shape:  $D_X \times D_Q$ )

## Computation:

**Query vectors:**  $\mathbf{Q} = \mathbf{XW}_Q$

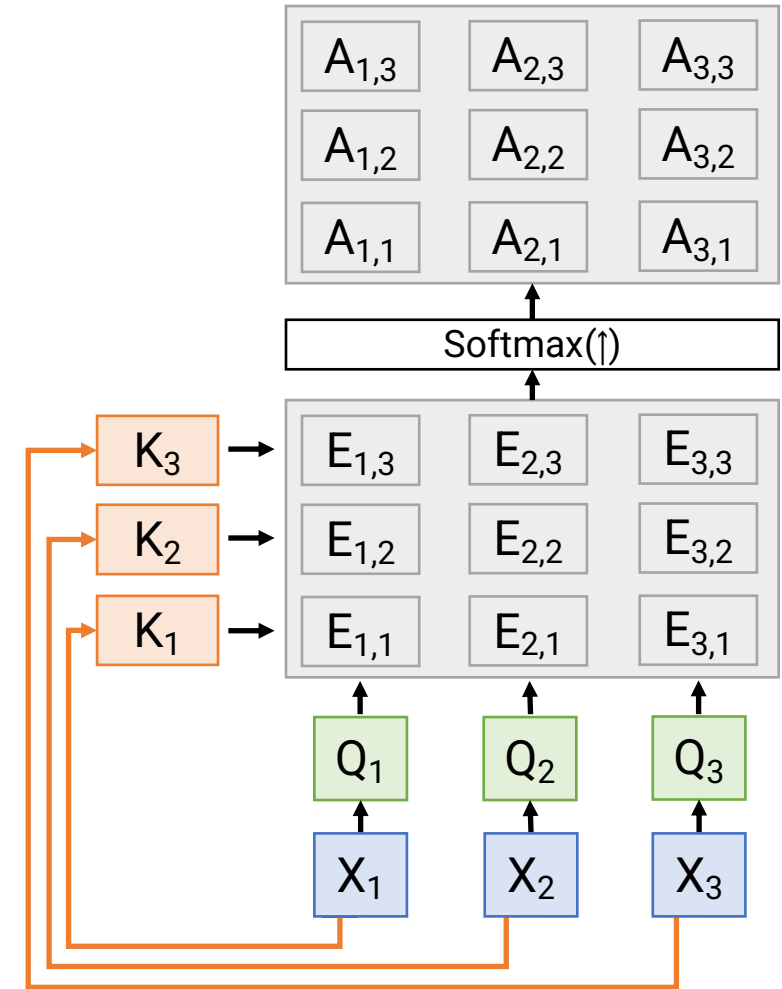
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

**Value vectors:**  $\mathbf{V} = \mathbf{XW}_V$  (Shape:  $N_X \times D_V$ )

**Similarities:**  $\mathbf{E} = \mathbf{QK}^T$  (Shape:  $N_X \times N_X$ )  $E_{ij} = \mathbf{Q}_i \cdot \mathbf{K}_j / \text{sqrt}(D_Q)$

**Attention weights:**  $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$  (Shape:  $N_X \times N_X$ )

**Output vectors:**  $\mathbf{Y} = \mathbf{AV}$  (Shape:  $N_X \times D_V$ )  $Y_i = \sum_j A_{i,j} \mathbf{V}_j$



# Self-Attention Layer

One **query** per **input vector**

## Inputs:

**Input vectors:**  $\mathbf{X}$  (Shape:  $N_X \times D_X$ )

**Key matrix:**  $\mathbf{W}_K$  (Shape:  $D_X \times D_Q$ )

**Value matrix:**  $\mathbf{W}_V$  (Shape:  $D_X \times D_V$ )

**Query matrix:**  $\mathbf{W}_Q$  (Shape:  $D_X \times D_Q$ )

## Computation:

**Query vectors:**  $\mathbf{Q} = \mathbf{XW}_Q$

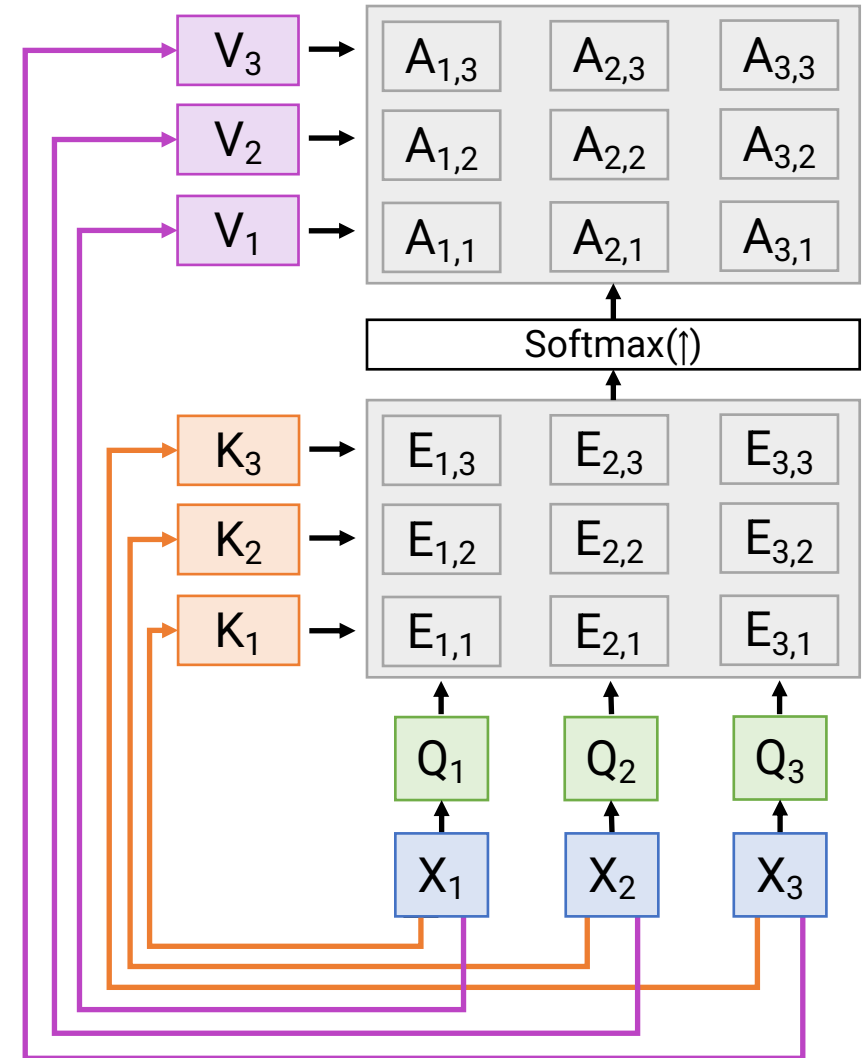
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

**Value vectors:**  $\mathbf{V} = \mathbf{XW}_V$  (Shape:  $N_X \times D_V$ )

**Similarities:**  $\mathbf{E} = \mathbf{QK}^T$  (Shape:  $N_X \times N_X$ )  $E_{ij} = \mathbf{Q}_i \cdot \mathbf{K}_j / \text{sqrt}(D_Q)$

**Attention weights:**  $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$  (Shape:  $N_X \times N_X$ )

**Output vectors:**  $\mathbf{Y} = \mathbf{AV}$  (Shape:  $N_X \times D_V$ )  $Y_i = \sum_j A_{i,j} \mathbf{V}_j$





# Self-Attention Layer

One **query** per **input vector**

## Inputs:

**Input vectors:**  $\mathbf{X}$  (Shape:  $N_X \times D_X$ )

**Key matrix:**  $\mathbf{W}_K$  (Shape:  $D_X \times D_Q$ )

**Value matrix:**  $\mathbf{W}_V$  (Shape:  $D_X \times D_V$ )

**Query matrix:**  $\mathbf{W}_Q$  (Shape:  $D_X \times D_Q$ )

## Computation:

**Query vectors:**  $\mathbf{Q} = \mathbf{XW}_Q$

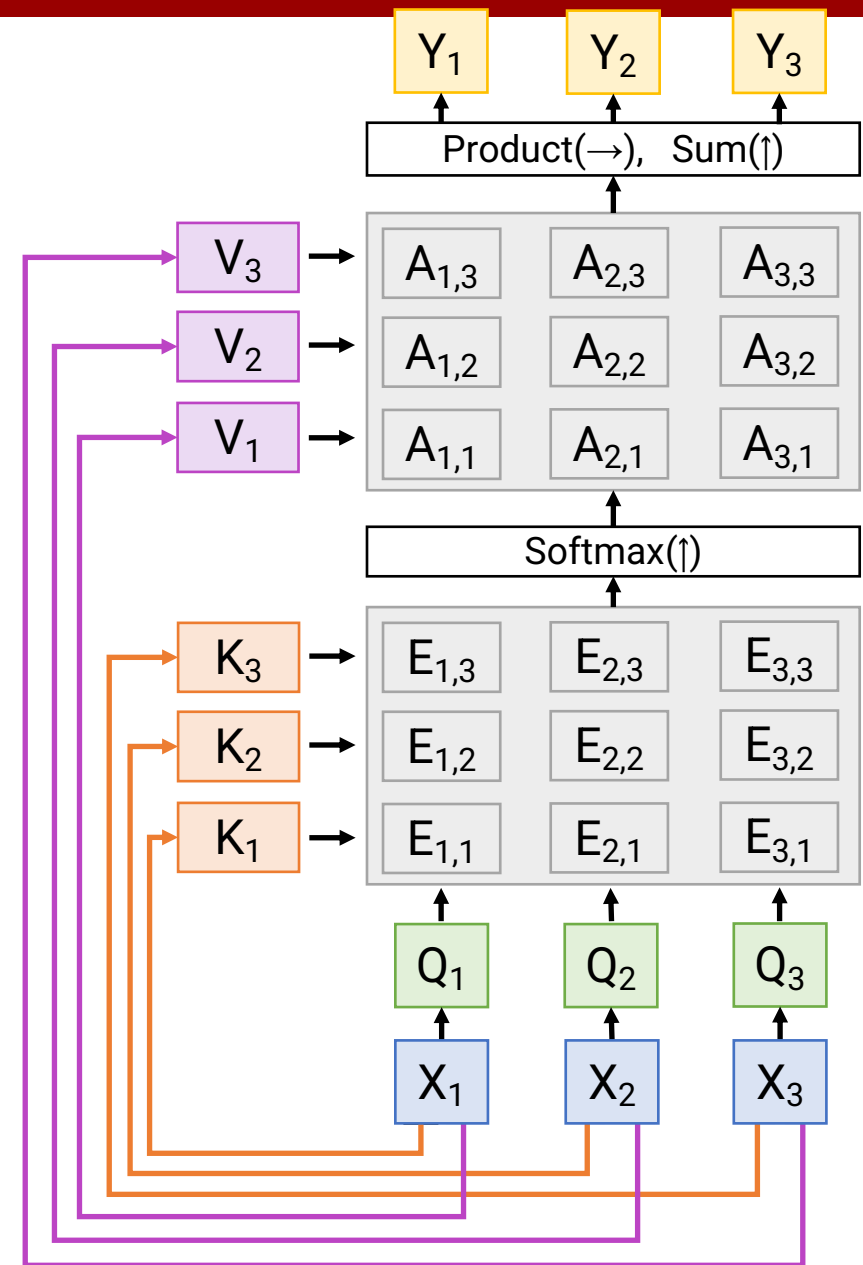
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

**Value vectors:**  $\mathbf{V} = \mathbf{XW}_V$  (Shape:  $N_X \times D_V$ )

**Similarities:**  $\mathbf{E} = \mathbf{QK}^T$  (Shape:  $N_X \times N_X$ )  $E_{ij} = \mathbf{Q}_i \cdot \mathbf{K}_j / \text{sqrt}(D_Q)$

**Attention weights:**  $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$  (Shape:  $N_X \times N_X$ )

**Output vectors:**  $\mathbf{Y} = \mathbf{AV}$  (Shape:  $N_X \times D_V$ )  $Y_i = \sum_j A_{i,j} \mathbf{V}_j$



# Self-Attention Layer

## Inputs:

Input vectors:  $\mathbf{X}$  (Shape:  $N_X \times D_X$ )

Key matrix:  $\mathbf{W}_K$  (Shape:  $D_X \times D_Q$ )

Value matrix:  $\mathbf{W}_V$  (Shape:  $D_X \times D_V$ )

Query matrix:  $\mathbf{W}_Q$  (Shape:  $D_X \times D_Q$ )

## Computation:

Query vectors:  $\mathbf{Q} = \mathbf{XW}_Q$

Key vectors:  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

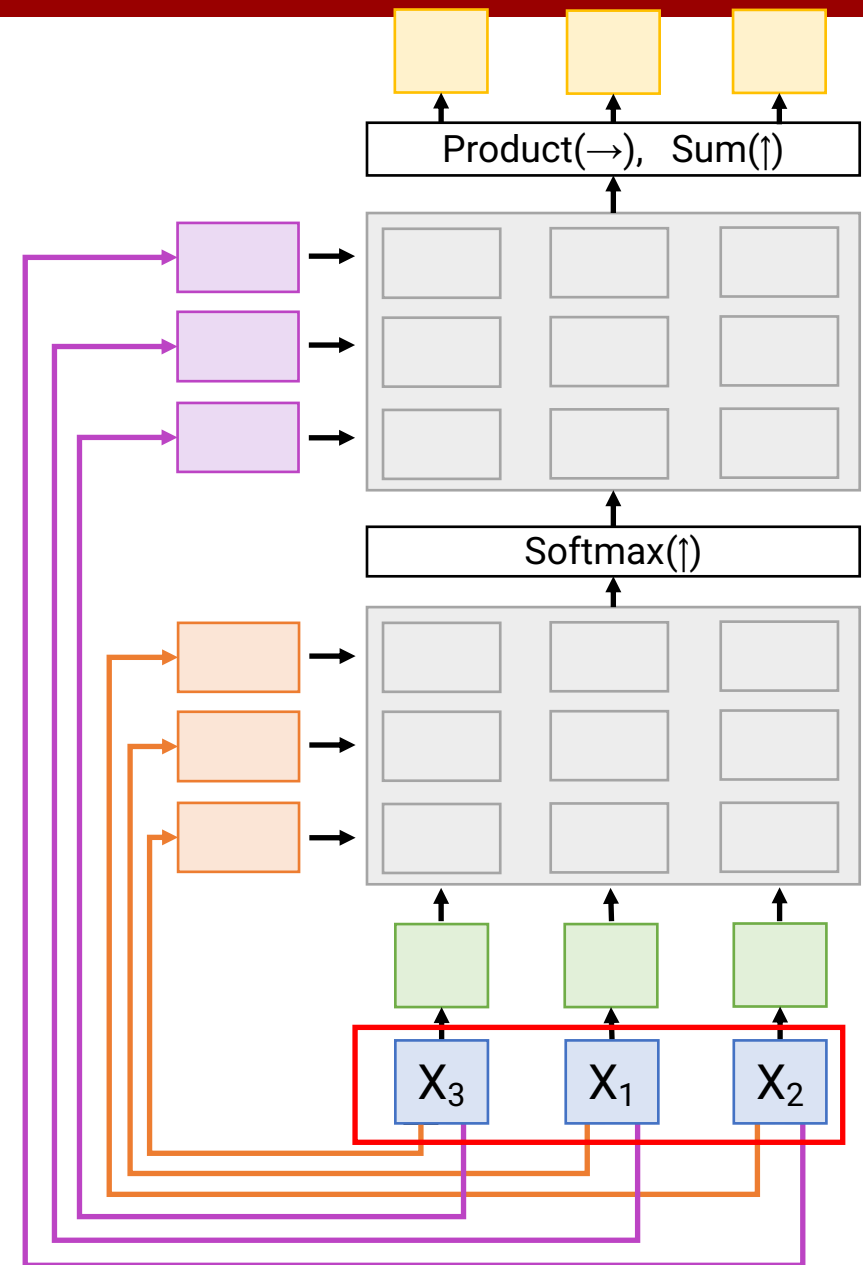
Value vectors:  $\mathbf{V} = \mathbf{XW}_V$  (Shape:  $N_X \times D_V$ )

Similarities:  $\mathbf{E} = \mathbf{QK}^T$  (Shape:  $N_X \times N_X$ )  $E_{ij} = \mathbf{Q}_i \cdot \mathbf{K}_j / \text{sqrt}(D_Q)$

Attention weights:  $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$  (Shape:  $N_X \times N_X$ )

Output vectors:  $\mathbf{Y} = \mathbf{AV}$  (Shape:  $N_X \times D_V$ )  $Y_i = \sum_j A_{i,j} \mathbf{V}_j$

Consider **permuting**  
the input vectors:



# Self-Attention Layer

## Inputs:

Input vectors:  $\mathbf{X}$  (Shape:  $N_X \times D_X$ )

Key matrix:  $\mathbf{W}_K$  (Shape:  $D_X \times D_Q$ )

Value matrix:  $\mathbf{W}_V$  (Shape:  $D_X \times D_V$ )

Query matrix:  $\mathbf{W}_Q$  (Shape:  $D_X \times D_Q$ )

## Computation:

Query vectors:  $\mathbf{Q} = \mathbf{XW}_Q$

Key vectors:  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

Value Vectors:  $\mathbf{V} = \mathbf{XW}_V$  (Shape:  $N_X \times D_V$ )

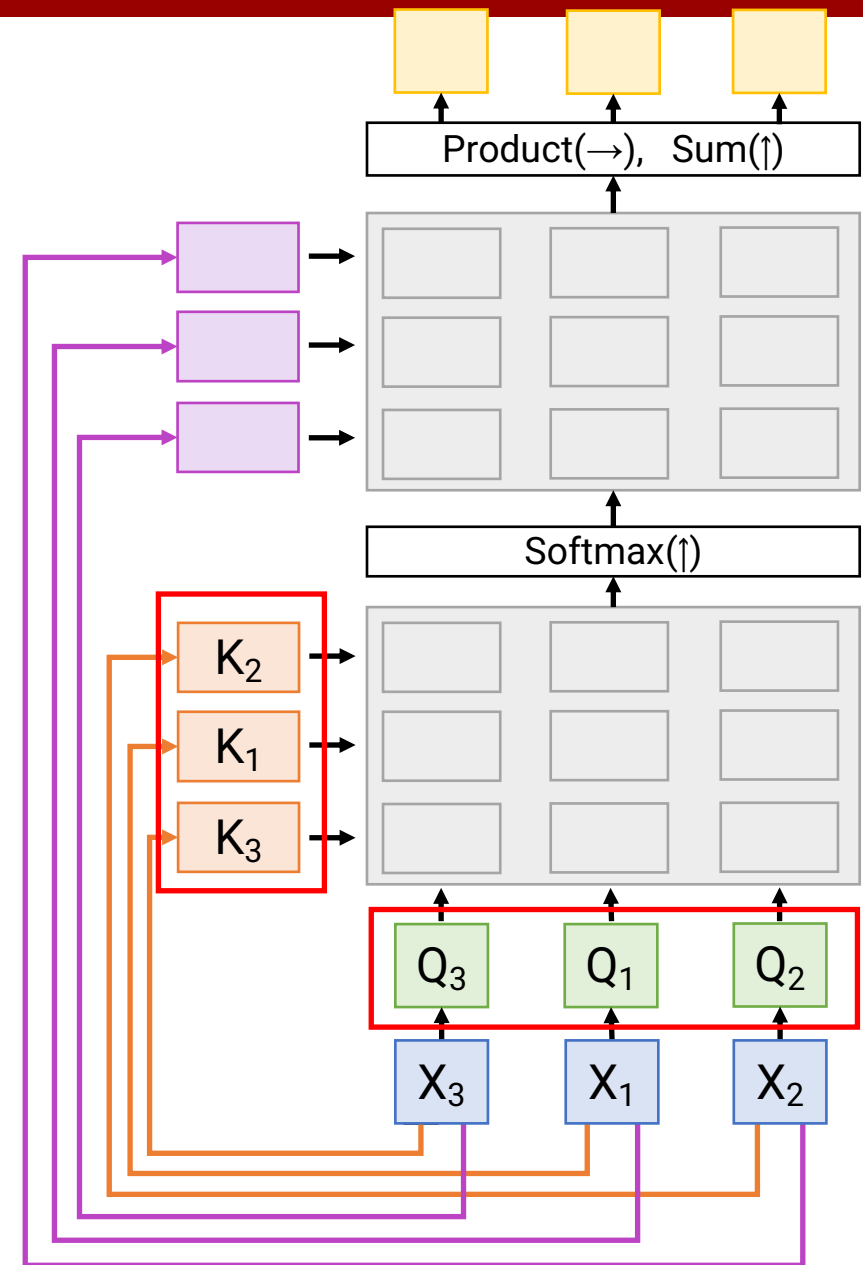
Similarities:  $\mathbf{E} = \mathbf{QK}^T$  (Shape:  $N_X \times N_X$ )  $E_{ij} = \mathbf{Q}_i \cdot \mathbf{K}_j / \text{sqrt}(D_Q)$

Attention weights:  $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$  (Shape:  $N_X \times N_X$ )

Output vectors:  $\mathbf{Y} = \mathbf{AV}$  (Shape:  $N_X \times D_V$ )  $Y_i = \sum_j A_{i,j} \mathbf{V}_j$

Consider **permuting**  
the input vectors:

Queries and Keys will  
be the same, but  
permuted



# Self-Attention Layer

## Inputs:

Input vectors:  $\mathbf{X}$  (Shape:  $N_X \times D_X$ )

Key matrix:  $\mathbf{W}_K$  (Shape:  $D_X \times D_Q$ )

Value matrix:  $\mathbf{W}_V$  (Shape:  $D_X \times D_V$ )

Query matrix:  $\mathbf{W}_Q$  (Shape:  $D_X \times D_Q$ )

## Computation:

Query vectors:  $\mathbf{Q} = \mathbf{XW}_Q$

Key vectors:  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

Value vectors:  $\mathbf{V} = \mathbf{XW}_V$  (Shape:  $N_X \times D_V$ )

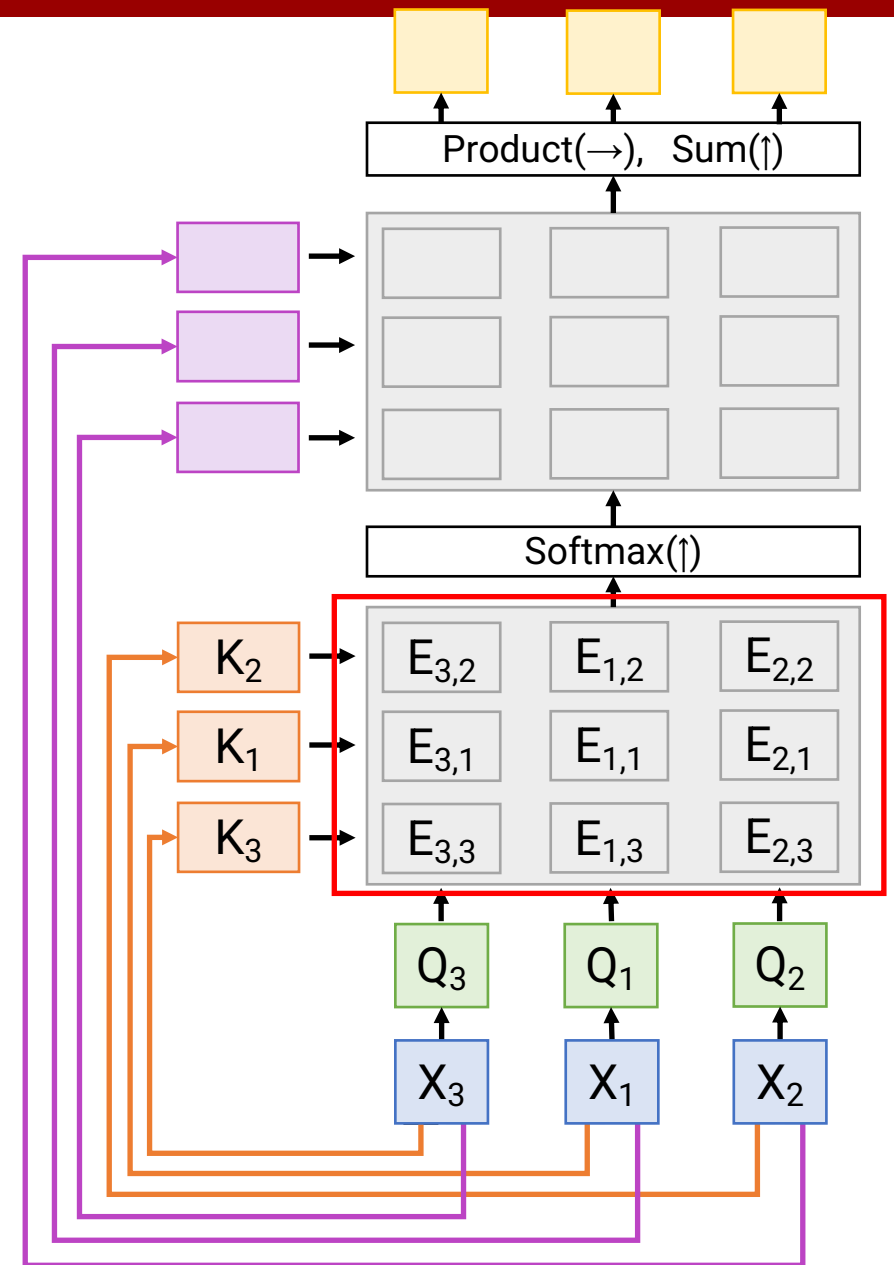
Similarities:  $\mathbf{E} = \mathbf{QK}^T$  (Shape:  $N_X \times N_X$ )  $E_{ij} = \mathbf{Q}_i \cdot \mathbf{K}_j / \text{sqrt}(D_Q)$

Attention weights:  $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$  (Shape:  $N_X \times N_X$ )

Output vectors:  $\mathbf{Y} = \mathbf{AV}$  (Shape:  $N_X \times D_V$ )  $Y_i = \sum_j A_{i,j} \mathbf{V}_j$

Consider **permuting**  
the input vectors:

Similarities will be the  
same, but permuted



# Self-Attention Layer

## Inputs:

**Input vectors:**  $\mathbf{X}$  (Shape:  $N_X \times D_X$ )

**Key matrix:**  $\mathbf{W}_K$  (Shape:  $D_X \times D_Q$ )

**Value matrix:**  $\mathbf{W}_V$  (Shape:  $D_X \times D_V$ )

**Query matrix:**  $\mathbf{W}_Q$  (Shape:  $D_X \times D_Q$ )

## Computation:

**Query vectors:**  $\mathbf{Q} = \mathbf{XW}_Q$

**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

**Value vectors:**  $\mathbf{V} = \mathbf{XW}_V$  (Shape:  $N_X \times D_V$ )

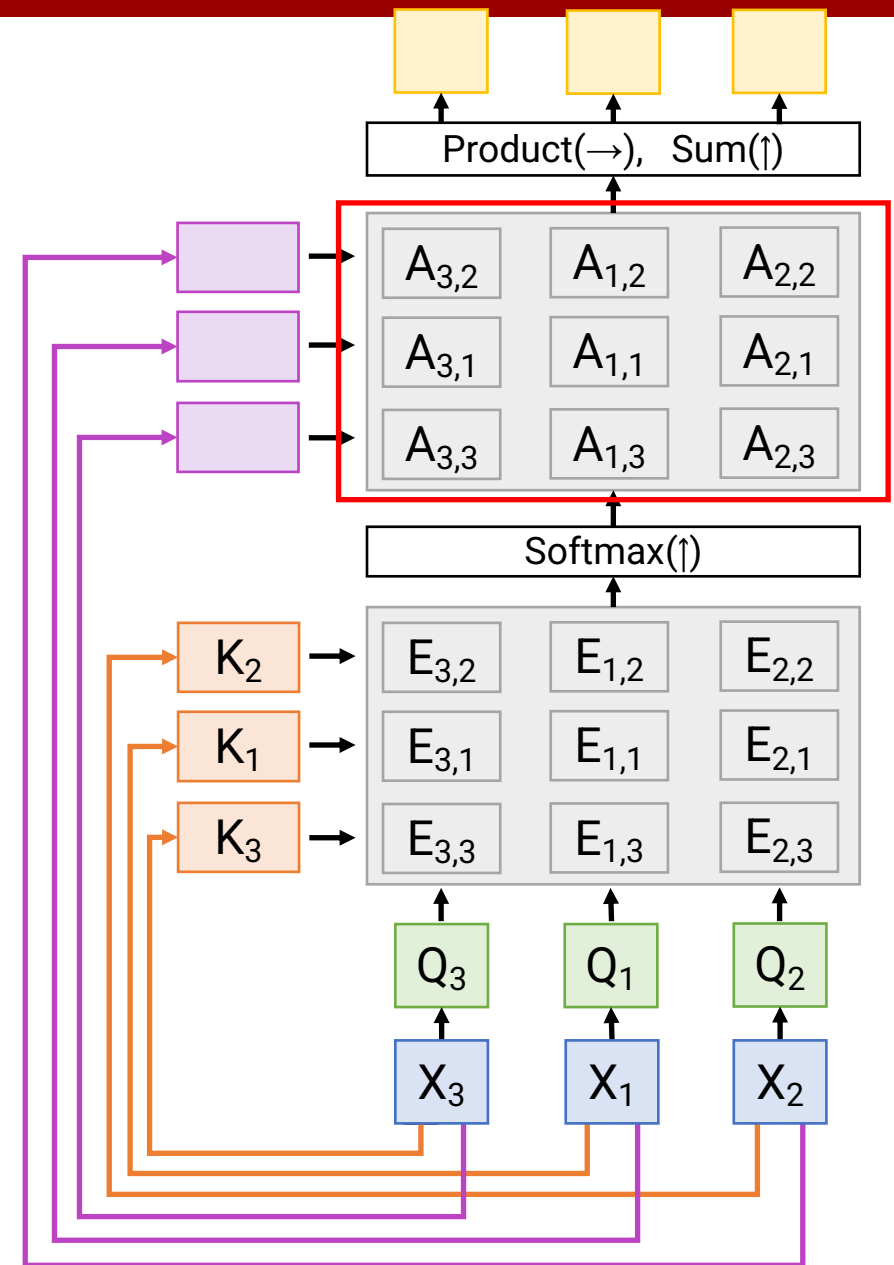
**Similarities:**  $\mathbf{E} = \mathbf{QK}^T$  (Shape:  $N_X \times N_X$ )  $E_{ij} = \mathbf{Q}_i \cdot \mathbf{K}_j / \text{sqrt}(D_Q)$

**Attention weights:**  $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$  (Shape:  $N_X \times N_X$ )

**Output vectors:**  $\mathbf{Y} = \mathbf{AV}$  (Shape:  $N_X \times D_V$ )  $Y_i = \sum_j A_{i,j} \mathbf{V}_j$

Consider **permuting**  
the input vectors:

Attention weights will  
be the same, but  
permuted



# Self-Attention Layer

## Inputs:

**Input vectors:**  $X$  (Shape:  $N_X \times D_X$ )

**Key matrix:**  $W_K$  (Shape:  $D_X \times D_Q$ )

**Value matrix:**  $W_V$  (Shape:  $D_X \times D_V$ )

**Query matrix:**  $W_Q$  (Shape:  $D_X \times D_Q$ )

## Computation:

**Query vectors:**  $Q = XW_Q$

**Key vectors:**  $K = XW_K$  (Shape:  $N_X \times D_Q$ )

**Value vectors:**  $V = XW_V$  (Shape:  $N_X \times D_V$ )

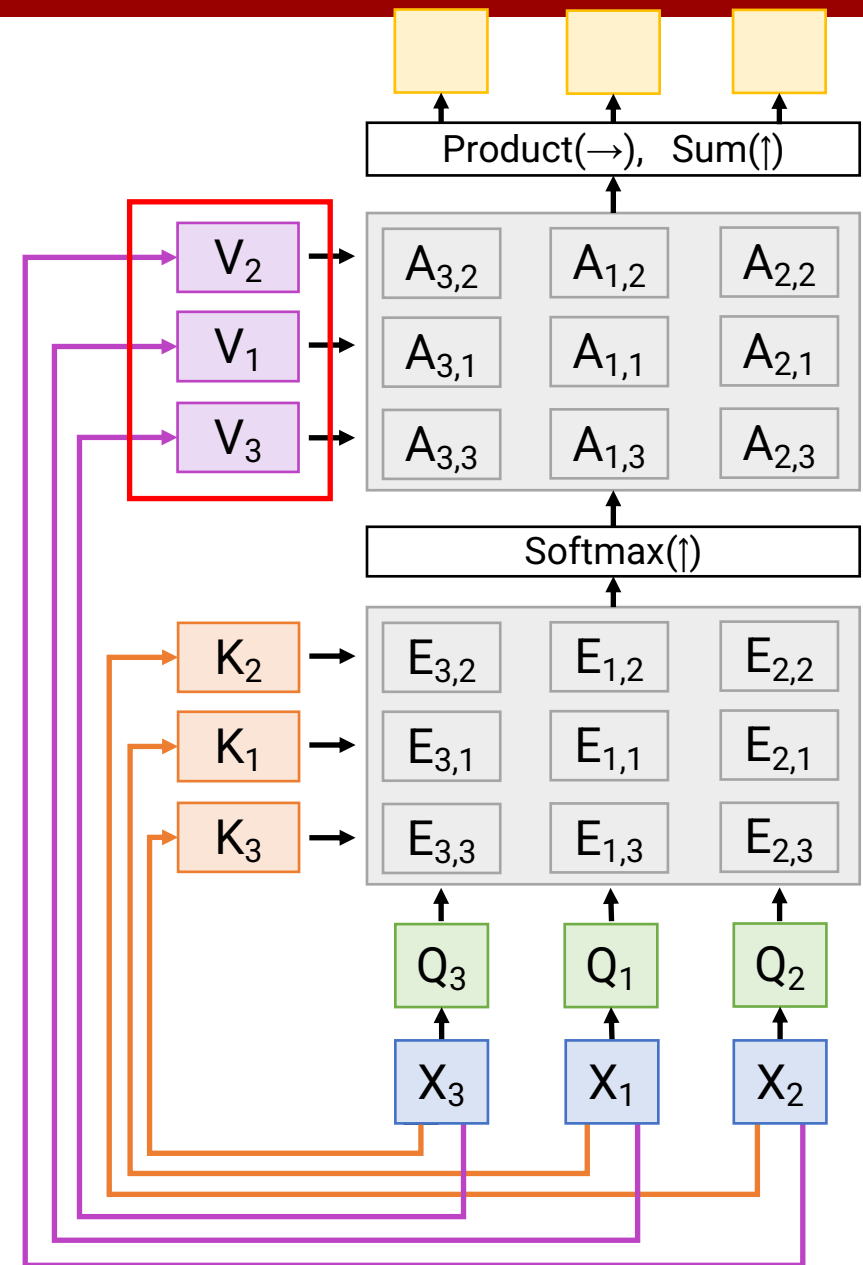
**Similarities:**  $E = QK^T$  (Shape:  $N_X \times N_X$ )  $E_{ij} = Q_i \cdot K_j / \text{sqrt}(D_Q)$

**Attention weights:**  $A = \text{softmax}(E, \text{dim}=1)$  (Shape:  $N_X \times N_X$ )

**Output vectors:**  $Y = AV$  (Shape:  $N_X \times D_V$ )  $Y_i = \sum_j A_{i,j} V_j$

Consider **permuting**  
the input vectors:

Values will be the  
same, but permuted



# Self-Attention Layer

## Inputs:

**Input vectors:**  $\mathbf{X}$  (Shape:  $N_X \times D_X$ )

**Key matrix:**  $\mathbf{W}_K$  (Shape:  $D_X \times D_Q$ )

**Value matrix:**  $\mathbf{W}_V$  (Shape:  $D_X \times D_V$ )

**Query matrix:**  $\mathbf{W}_Q$  (Shape:  $D_X \times D_Q$ )

## Computation:

**Query vectors:**  $\mathbf{Q} = \mathbf{XW}_Q$

**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

**Value vectors:**  $\mathbf{V} = \mathbf{XW}_V$  (Shape:  $N_X \times D_V$ )

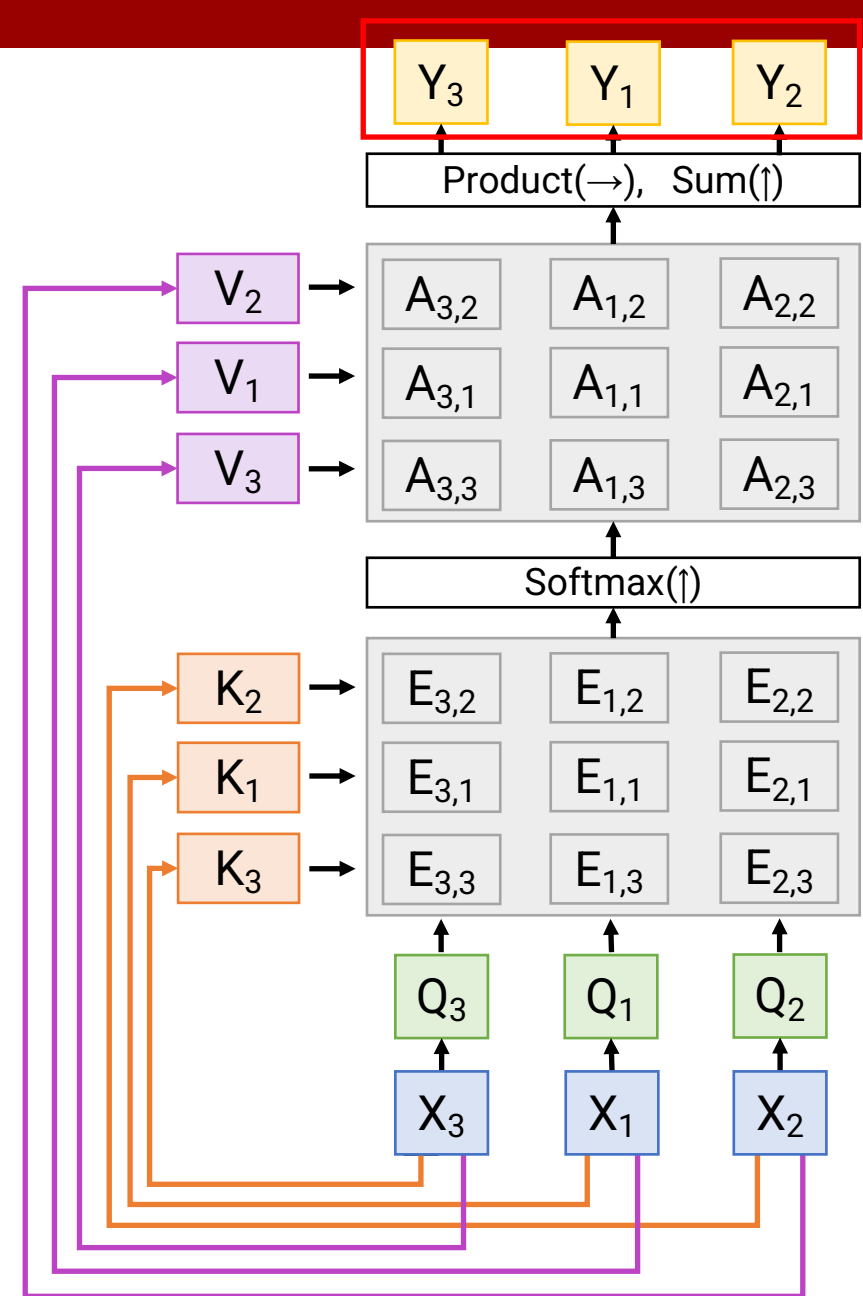
**Similarities:**  $\mathbf{E} = \mathbf{QK}^T$  (Shape:  $N_X \times N_X$ )  $E_{ij} = \mathbf{Q}_i \cdot \mathbf{K}_j / \text{sqrt}(D_Q)$

**Attention weights:**  $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$  (Shape:  $N_X \times N_X$ )

**Output vectors:**  $\mathbf{Y} = \mathbf{AV}$  (Shape:  $N_X \times D_V$ )  $Y_i = \sum_j A_{i,j} \mathbf{V}_j$

Consider **permuting**  
the input vectors:

Outputs will be the  
same, but **permuted**



# Self-Attention Layer

## Inputs:

**Input vectors:**  $\mathbf{X}$  (Shape:  $N_X \times D_X$ )

**Key matrix:**  $\mathbf{W}_K$  (Shape:  $D_X \times D_Q$ )

**Value matrix:**  $\mathbf{W}_V$  (Shape:  $D_X \times D_V$ )

**Query matrix:**  $\mathbf{W}_Q$  (Shape:  $D_X \times D_Q$ )

## Computation:

**Query vectors:**  $\mathbf{Q} = \mathbf{XW}_Q$

**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

**Value vectors:**  $\mathbf{V} = \mathbf{XW}_V$  (Shape:  $N_X \times D_V$ )

**Similarities:**  $\mathbf{E} = \mathbf{QK}^T$  (Shape:  $N_X \times N_X$ )  $E_{ij} = \mathbf{Q}_i \cdot \mathbf{K}_j / \text{sqrt}(D_Q)$

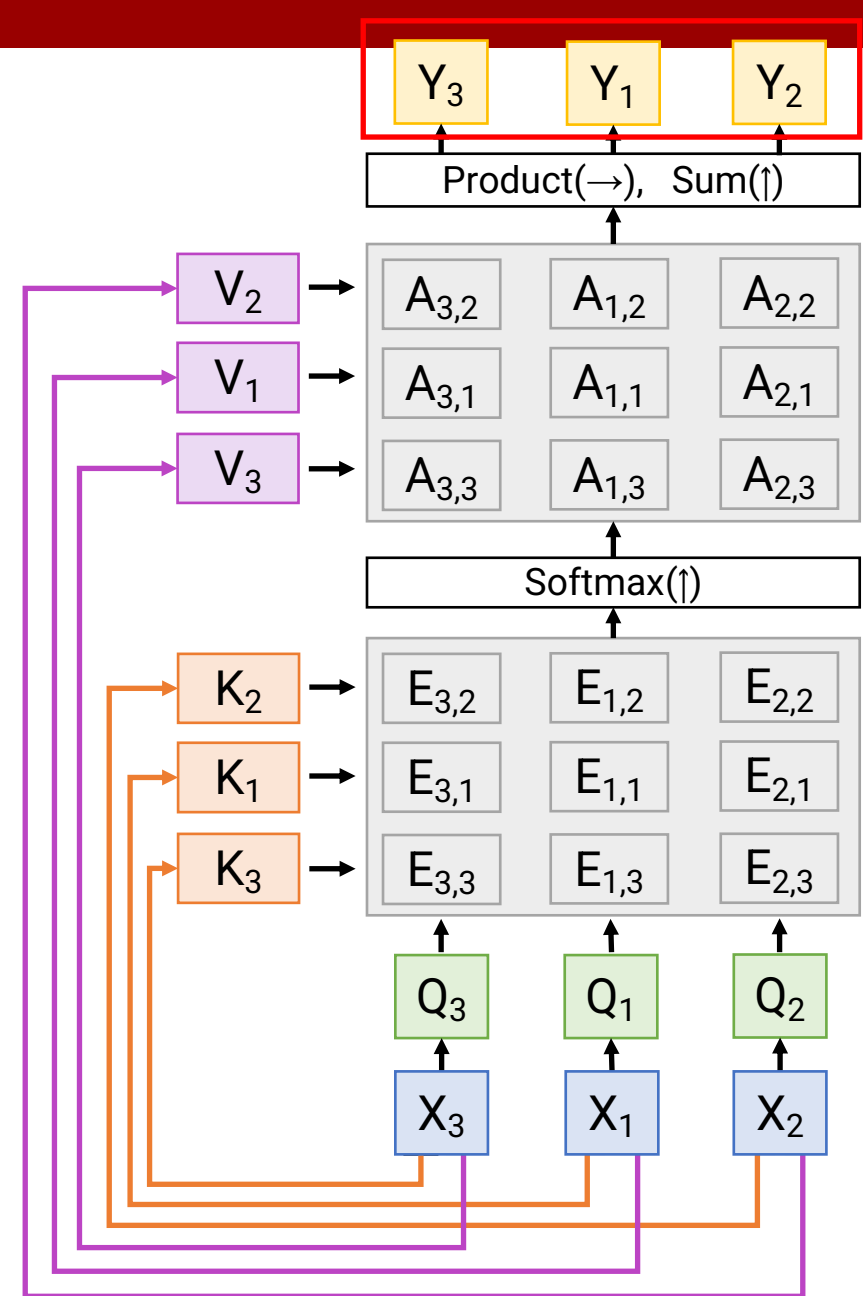
**Attention weights:**  $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$  (Shape:  $N_X \times N_X$ )

**Output vectors:**  $\mathbf{Y} = \mathbf{AV}$  (Shape:  $N_X \times D_V$ )  $Y_i = \sum_j A_{i,j} \mathbf{V}_j$

Consider **permuting**  
the input vectors:

Outputs will be the  
same, but permuted

Self-attention layer is  
**Permutation  
Equivariant**  
 $f(s(x)) = s(f(x))$





# Self-Attention Layer

## Inputs:

**Input vectors:**  $X$  (Shape:  $N_X \times D_X$ )

**Key matrix:**  $W_K$  (Shape:  $D_X \times D_Q$ )

**Value matrix:**  $W_V$  (Shape:  $D_X \times D_V$ )

**Query matrix:**  $W_Q$  (Shape:  $D_X \times D_Q$ )

## Computation:

**Query vectors:**  $Q = XW_Q$

**Key vectors:**  $K = XW_K$  (Shape:  $N_X \times D_Q$ )

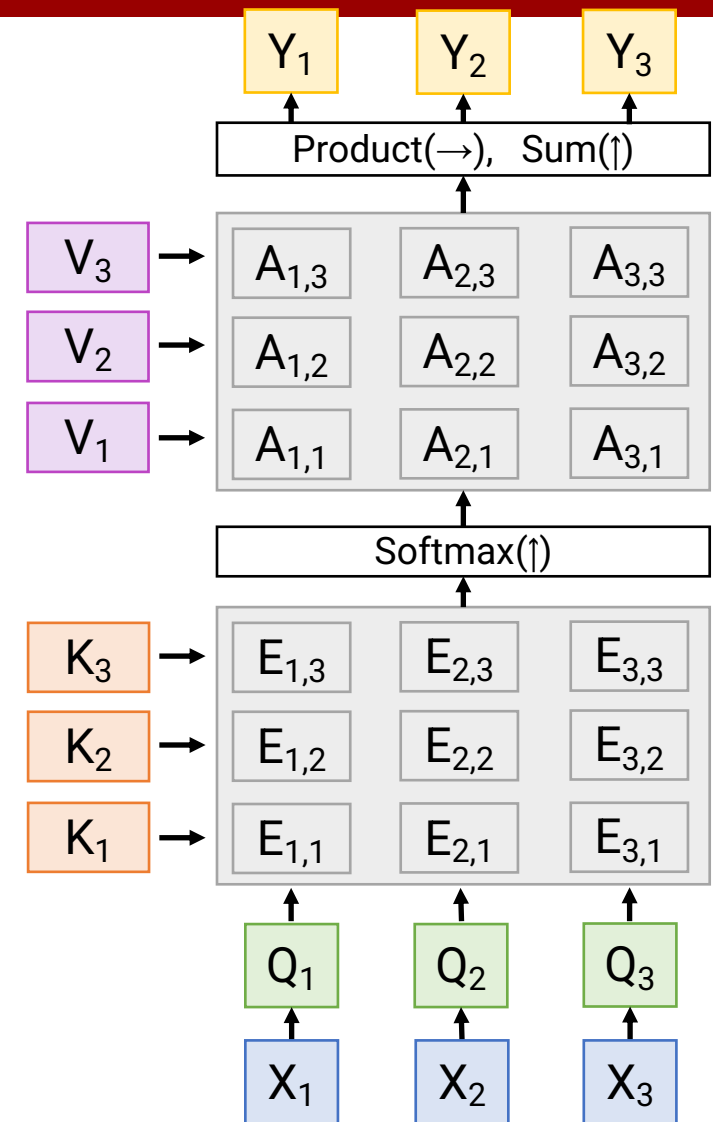
**Value vectors:**  $V = XW_V$  (Shape:  $N_X \times D_V$ )

**Similarities:**  $E = QK^T$  (Shape:  $N_X \times N_X$ )  $E_{i,j} = Q_i \cdot K_j / \text{sqrt}(D_Q)$

**Attention weights:**  $A = \text{softmax}(E, \text{dim}=1)$  (Shape:  $N_X \times N_X$ )

**Output vectors:**  $Y = AV$  (Shape:  $N_X \times D_V$ )  $Y_i = \sum_j A_{i,j} V_j$

Self attention doesn't "know" the order of the vectors it is processing!



# Self-Attention Layer

## Inputs:

**Input vectors:**  $X$  (Shape:  $N_X \times D_X$ )

**Key matrix:**  $W_K$  (Shape:  $D_X \times D_Q$ )

**Value matrix:**  $W_V$  (Shape:  $D_X \times D_V$ )

**Query matrix:**  $W_Q$  (Shape:  $D_X \times D_Q$ )

## Computation:

**Query vectors:**  $Q = XW_Q$

**Key vectors:**  $K = XW_K$  (Shape:  $N_X \times D_Q$ )

**Value vectors:**  $V = XW_V$  (Shape:  $N_X \times D_V$ )

**Similarities:**  $E = QK^T$  (Shape:  $N_X \times N_X$ )  $E_{ij} = Q_i \cdot K_j / \text{sqrt}(D_Q)$

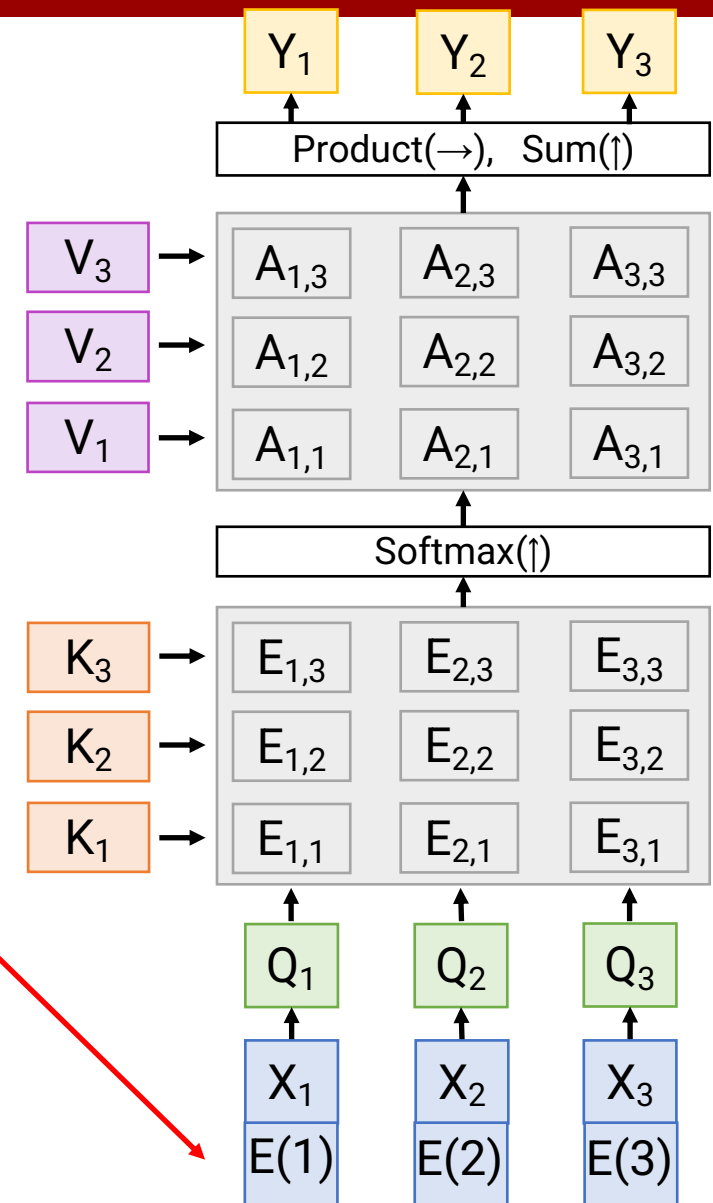
**Attention weights:**  $A = \text{softmax}(E, \text{dim}=1)$  (Shape:  $N_X \times N_X$ )

**Output vectors:**  $Y = AV$  (Shape:  $N_X \times D_V$ )  $Y_i = \sum_j A_{i,j} V_j$

Self attention doesn't "know" the order of the vectors it is processing!

In order to make processing position-aware, concatenate input with **positional encoding**

$E$  can be learned lookup table, or fixed function



# Masked Self-Attention Layer

## Inputs:

**Input vectors:**  $\mathbf{X}$  (Shape:  $N_X \times D_X$ )

**Key matrix:**  $\mathbf{W}_K$  (Shape:  $D_X \times D_Q$ )

**Value matrix:**  $\mathbf{W}_V$  (Shape:  $D_X \times D_V$ )

**Query matrix:**  $\mathbf{W}_Q$  (Shape:  $D_X \times D_Q$ )

## Computation:

**Query vectors:**  $\mathbf{Q} = \mathbf{XW}_Q$

**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

**Value vectors:**  $\mathbf{V} = \mathbf{XW}_V$  (Shape:  $N_X \times D_V$ )

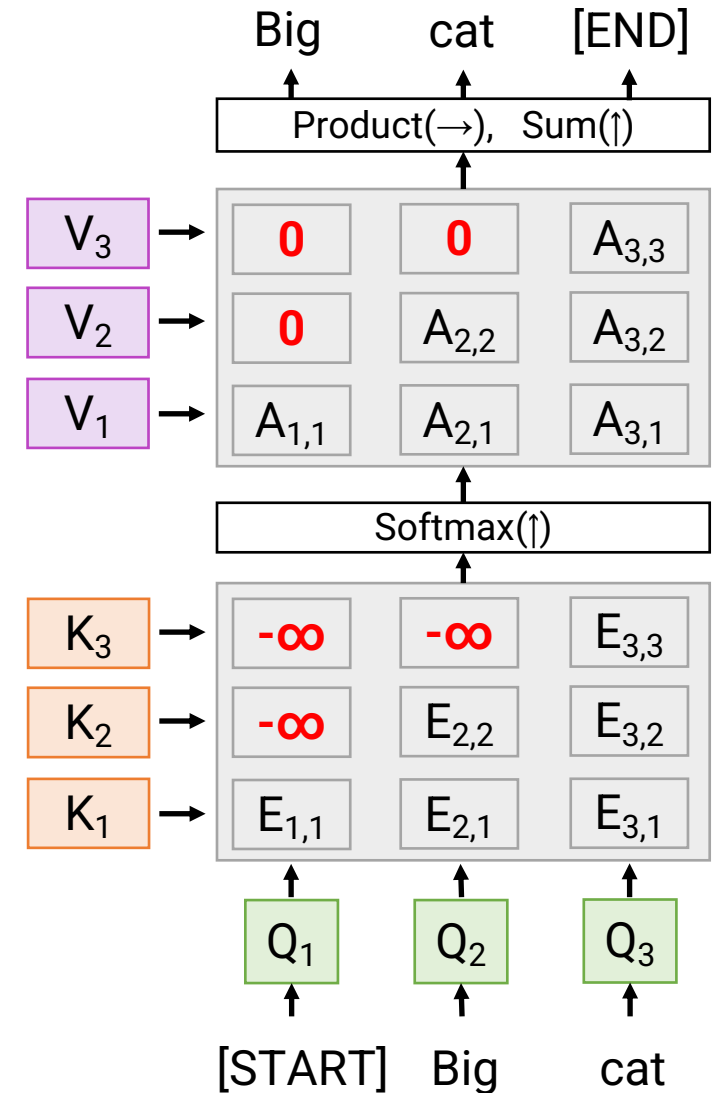
**Similarities:**  $\mathbf{E} = \mathbf{QK}^T$  (Shape:  $N_X \times N_X$ )  $E_{ij} = \mathbf{Q}_i \cdot \mathbf{K}_j / \text{sqrt}(D_Q)$

**Attention weights:**  $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$  (Shape:  $N_X \times N_X$ )

**Output vectors:**  $\mathbf{Y} = \mathbf{AV}$  (Shape:  $N_X \times D_V$ )  $Y_i = \sum_j A_{i,j} \mathbf{V}_j$

Don't let vectors "look ahead" in the sequence

Used for language modeling (predict next word)



# Multihead Self-Attention Layer

## Inputs:

Input vectors:  $\mathbf{X}$  (Shape:  $N_X \times D_X$ )

Key matrix:  $\mathbf{W}_K$  (Shape:  $D_X \times D_Q$ )

Value matrix:  $\mathbf{W}_V$  (Shape:  $D_X \times D_V$ )

Query matrix:  $\mathbf{W}_Q$  (Shape:  $D_X \times D_Q$ )

## Computation:

Query vectors:  $\mathbf{Q} = \mathbf{XW}_Q$

Key vectors:  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

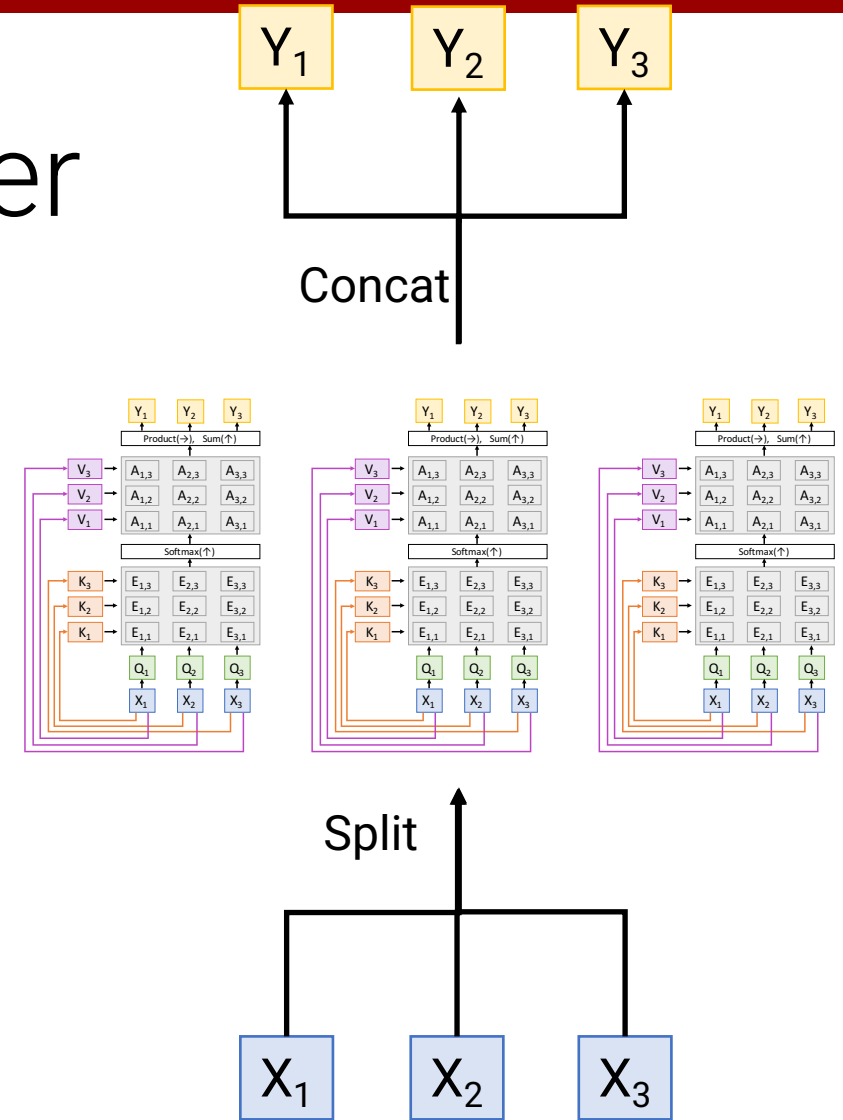
Value vectors:  $\mathbf{V} = \mathbf{XW}_V$  (Shape:  $N_X \times D_V$ )

Similarities:  $\mathbf{E} = \mathbf{QK}^T$  (Shape:  $N_X \times N_X$ )  $E_{ij} = \mathbf{Q}_i \cdot \mathbf{K}_j / \text{sqrt}(D_Q)$

Attention weights:  $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$  (Shape:  $N_X \times N_X$ )

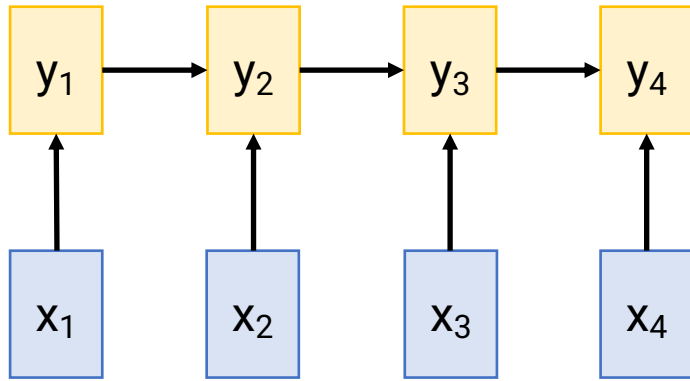
Output vectors:  $\mathbf{Y} = \mathbf{AV}$  (Shape:  $N_X \times D_V$ )  $Y_i = \sum_j A_{ij} \mathbf{V}_j$

Use H independent  
“Attention Heads” in  
parallel



# Three Ways of Processing Sequences

## Recurrent Neural Network



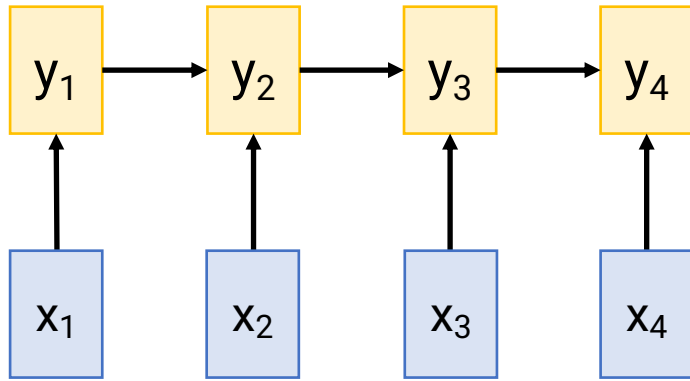
Works on **Ordered Sequences**

(+) **Good at long sequences: After one RNN layer,  $h_T$  "sees" the whole sequence**

(-) **Not parallelizable: need to compute hidden states sequentially**

# Three Ways of Processing Sequences

## Recurrent Neural Network

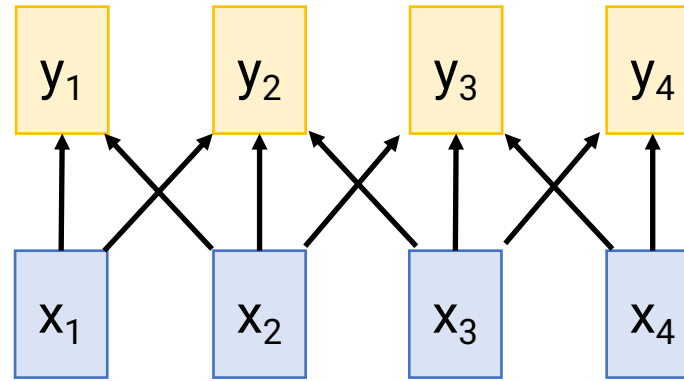


Works on **Ordered Sequences**

(+) **Good at long sequences: After one RNN layer,  $h_T$  "sees" the whole sequence**

(-) **Not parallelizable: need to compute hidden states sequentially**

## 1D Convolution



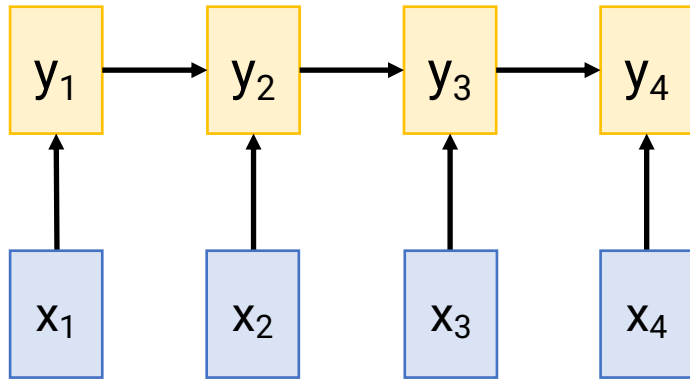
Works on **Multidimensional Grids**

(-) **Bad at long sequences: Need to stack many conv layers for outputs to "see" the whole sequence**

(+) **Highly parallel: Each output can be computed in parallel**

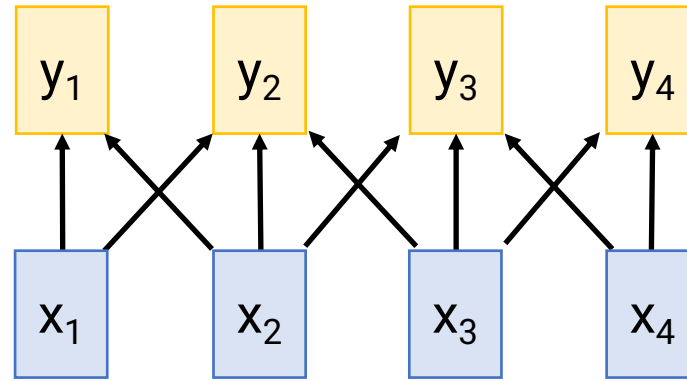
# Three Ways of Processing Sequences

## Recurrent Neural Network



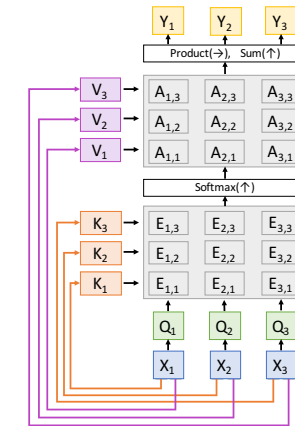
Works on **Ordered Sequences**  
(+) **Good at long sequences:** After one RNN layer,  $h_T$  "sees" the whole sequence  
(-) **Not parallelizable:** need to compute hidden states sequentially

## 1D Convolution



Works on **Multidimensional Grids**  
(-) **Bad at long sequences:** Need to stack many conv layers for outputs to "see" the whole sequence  
(+) **Highly parallel:** Each output can be computed in parallel

## Self-Attention



Works on **Sets of Vectors**  
(+) **Good at long sequences:** after one self-attention layer, each output "sees" all inputs!  
(+) **Highly parallel:** Each output can be computed in parallel  
(-) **Very memory intensive**

# Three Ways of Processing Sequences

Recurrent Neural Network

1D Convolution

Self-Attention

# Attention is all you need

Vaswani et al, NeurIPS 2017

Works on **Ordered Sequences**

(+) **Good at long sequences: After one RNN layer,  $h_T$  "sees" the whole sequence**

(-) **Not parallelizable: need to compute hidden states sequentially**

Works on **Multidimensional Grids**

(-) **Bad at long sequences: Need to stack many conv layers for outputs to "see" the whole sequence**

(+) **Highly parallel: Each output can be computed in parallel**

Works on **Sets of Vectors**

(+) **Good at long sequences: after one self-attention layer, each output "sees" all inputs!**

(+) **Highly parallel: Each output can be computed in parallel**

(-) **Very memory intensive**

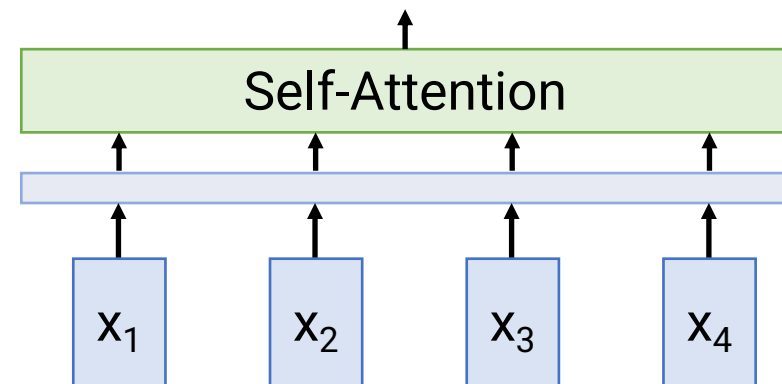


# The Transformer



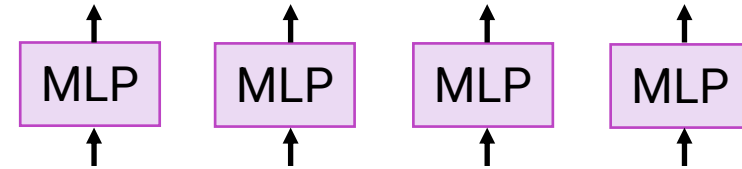
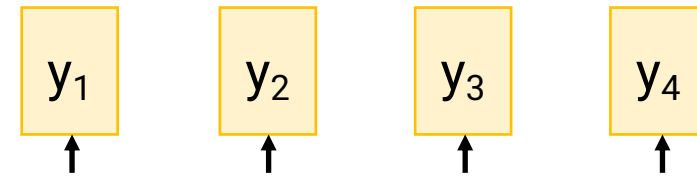
# The Transformer

All vectors interact  
with each other

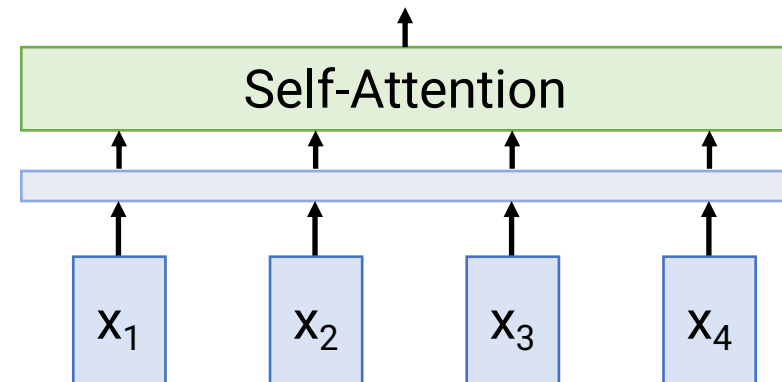


# The Transformer

MLP independently  
on each vector

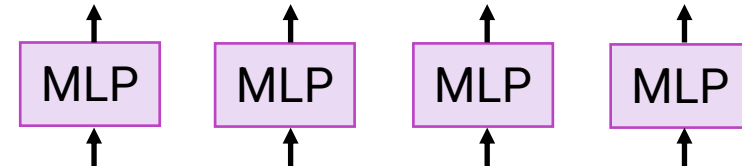
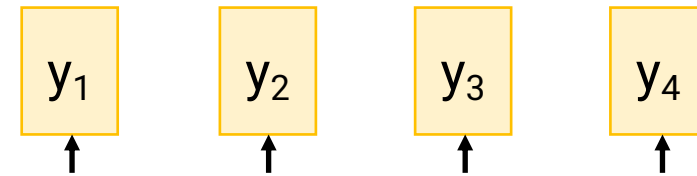


All vectors interact  
with each other



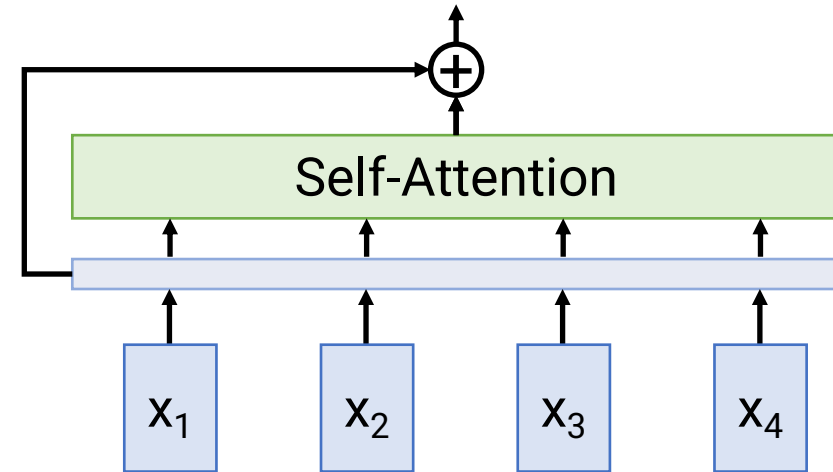
# The Transformer

MLP independently  
on each vector



Residual connection

All vectors interact  
with each other



# The Transformer

## Recall **Layer Normalization**:

Given  $h_1, \dots, h_N$  (Shape: D)

scale:  $\gamma$  (Shape: D)

shift:  $\beta$  (Shape: D)

$\mu_i = (1/D) \sum_j h_{i,j}$  (scalar)

$\sigma_i = (\sum_j (h_{i,j} - \mu_i)^2)^{1/2}$  (scalar)

$z_i = (h_i - \mu_i) / \sigma_i$

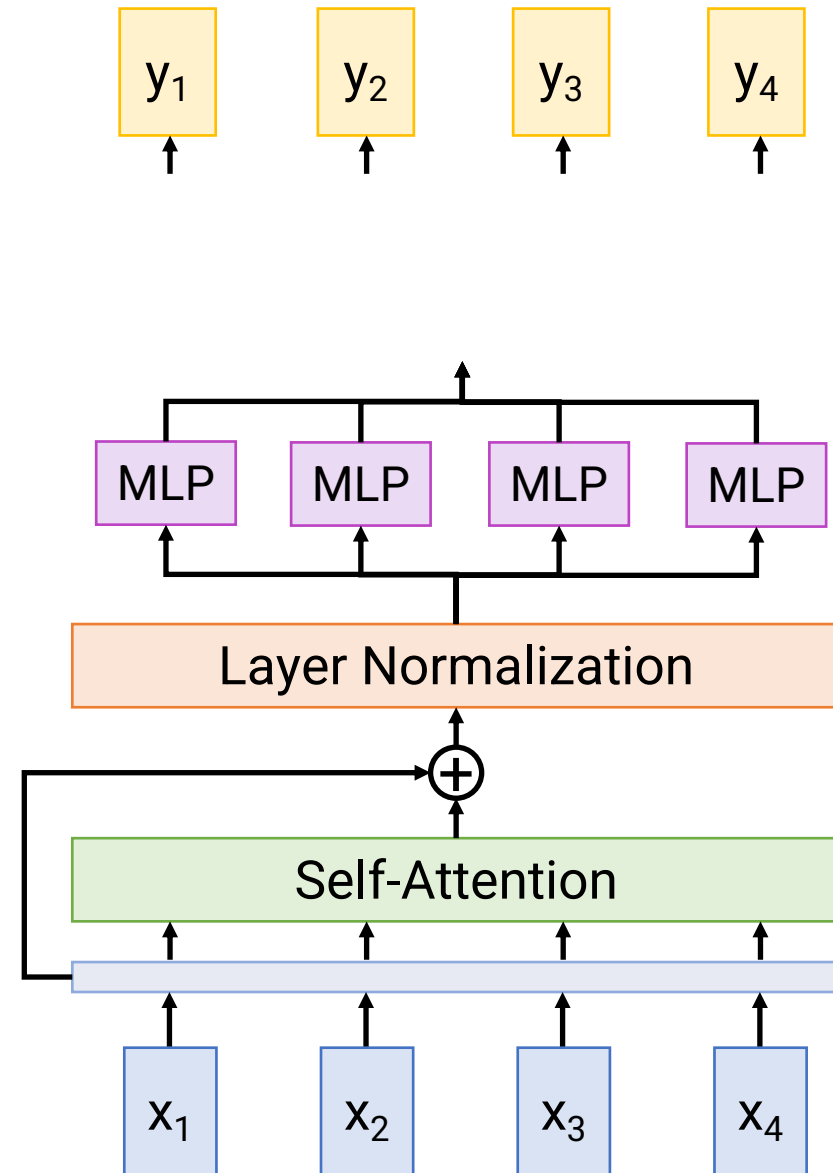
$y_i = \gamma * z_i + \beta$

Ba et al, 2016

MLP independently  
on each vector

Residual connection

All vectors interact  
with each other

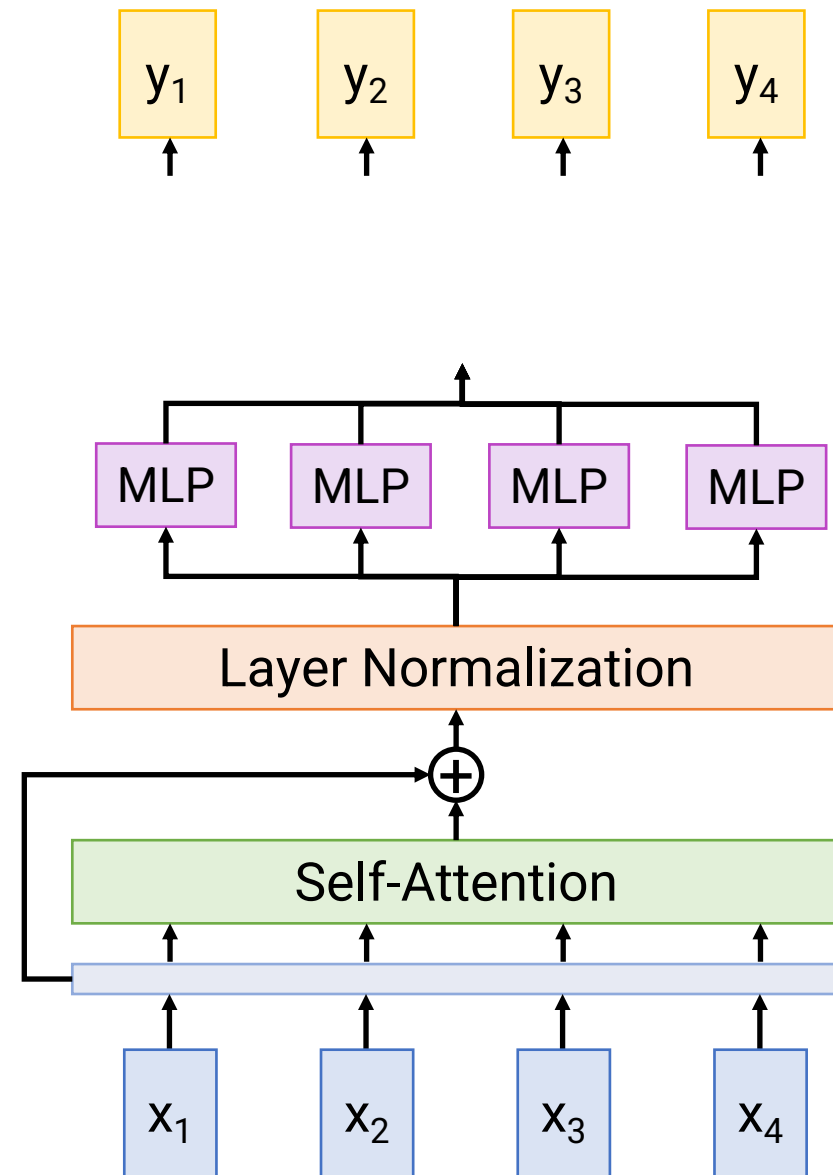


# The Transformer

MLP independently  
on each vector

Residual connection

All vectors interact  
with each other



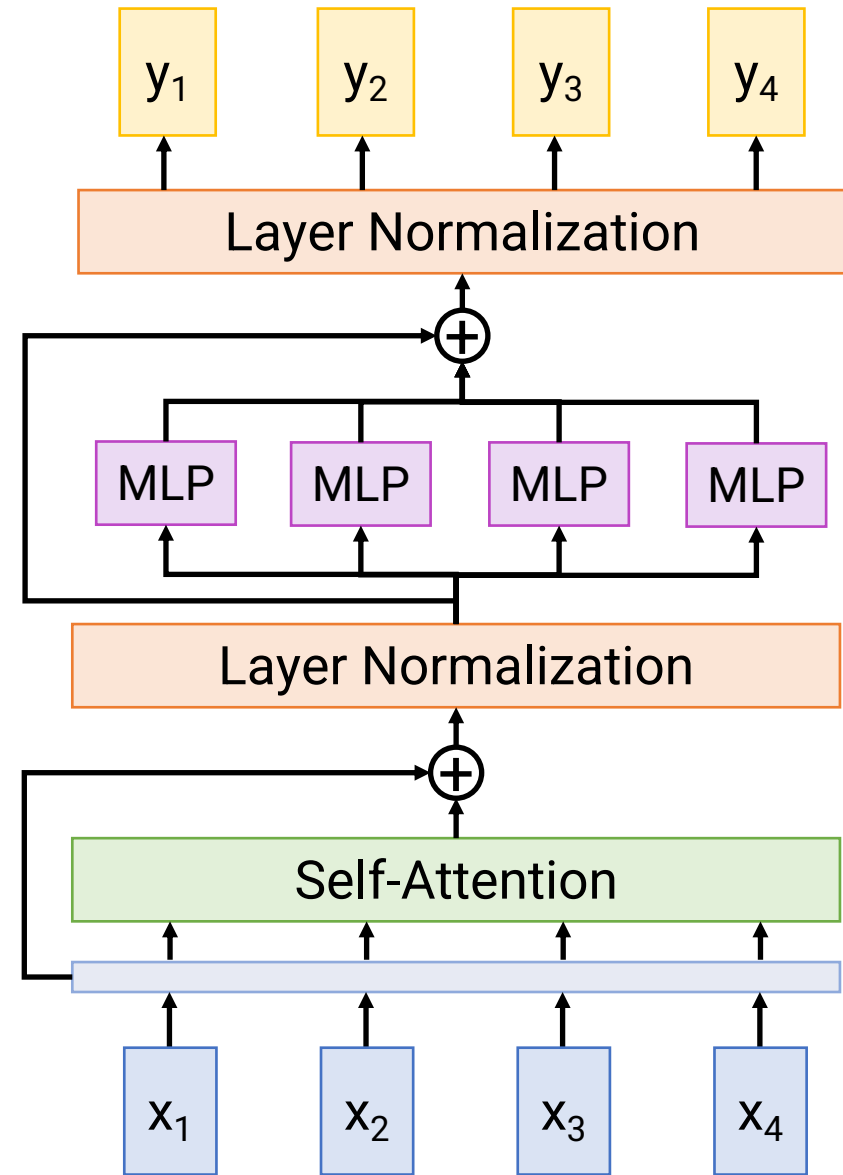
# The Transformer

Residual connection

MLP independently  
on each vector

Residual connection

All vectors interact  
with each other



# The Transformer

## **Transformer Block:**

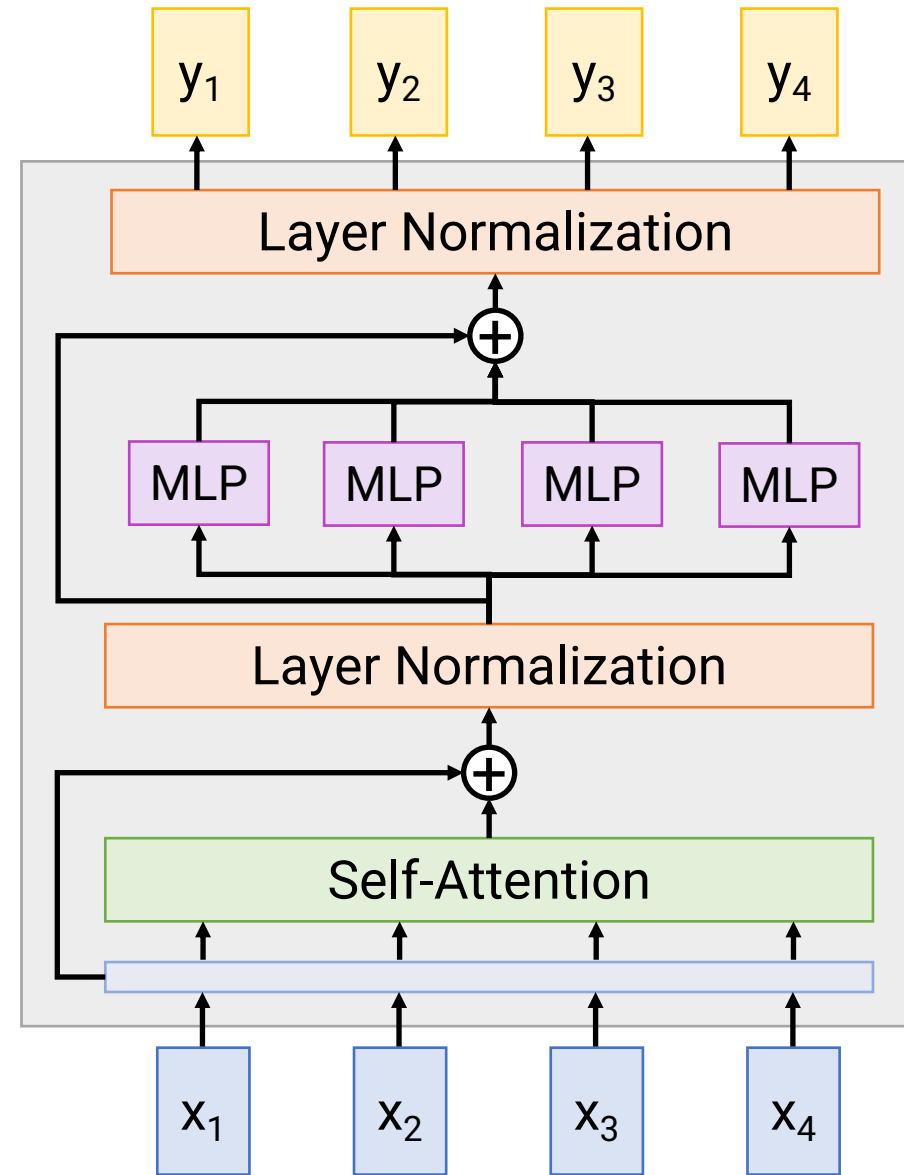
**Input:** Set of vectors  $x$

**Output:** Set of vectors  $y$

Self-attention is the only interaction between vectors!

Layer norm and MLP work independently per vector

Highly scalable, highly parallelizable





# The Transformer

## Transformer Block:

**Input:** Set of vectors  $x$

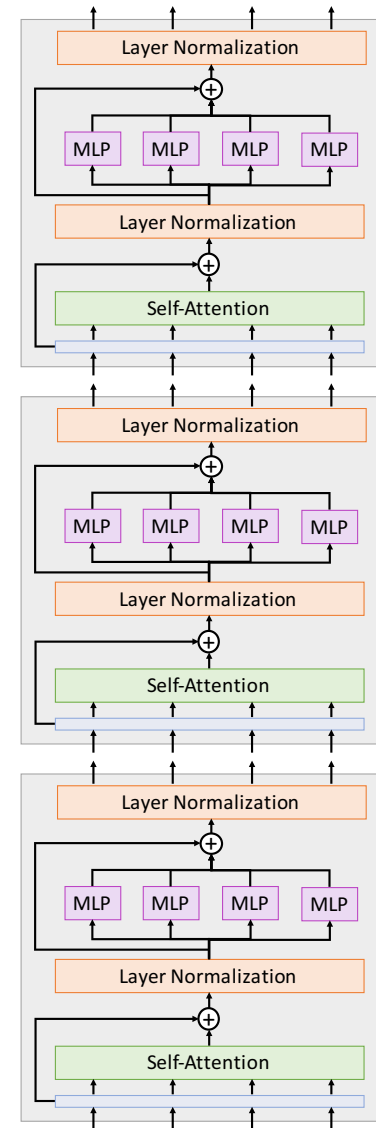
**Output:** Set of vectors  $y$

Self-attention is the only interaction between vectors!

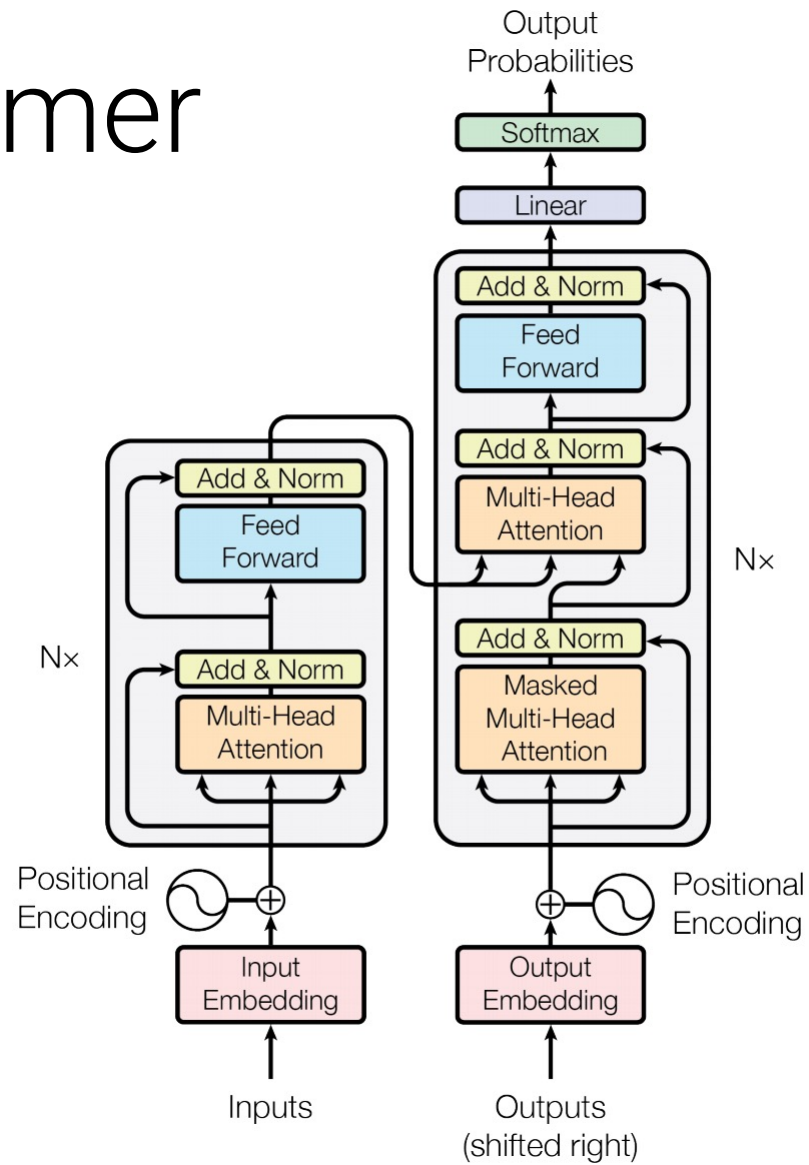
Layer norm and MLP work independently per vector

Highly scalable, highly parallelizable

A **Transformer** is a sequence of transformer blocks



# The Transformer



## Encoder-Decoder

# GLUE Benchmark

Rank	Name	Model	URL	Score	CoLA	SST-2	MRPC	STS-B	QQP	MNLI-m	MNLI-mm	QNLI	RTE	WNLI	AX	
1	HFL iFLYTEK	MacALBERT + DKM		90.7	74.8	97.0	94.5/92.6	92.8/92.6	74.7/90.6	91.3	91.1	97.8	92.0	94.5	52.6	
+	2	Alibaba DAMO NLP	StructBERT + TAPT	<a href="#">↗</a>	90.6	75.3	97.3	93.9/91.9	93.2/92.7	74.8/91.0	90.9	90.7	97.4	91.2	94.5	49.1
+	3	PING-AN Omni-Sinitic	ALBERT + DAAF + NAS		90.6	73.5	97.2	94.0/92.0	93.0/92.4	76.1/91.0	91.6	91.3	97.5	91.7	94.5	51.2
	4	ERNIE Team - Baidu	ERNIE	<a href="#">↗</a>	90.4	74.4	97.5	93.5/91.4	93.0/92.6	75.2/90.9	91.4	91.0	96.6	90.9	94.5	51.7
	5	T5 Team - Google	T5	<a href="#">↗</a>	90.3	71.6	97.5	92.8/90.4	93.1/92.8	75.1/90.6	92.2	91.9	96.9	92.8	94.5	53.1
	6	Microsoft D365 AI & MSR AI & GATECH	MT-DNN-SMART	<a href="#">↗</a>	89.9	69.5	97.5	93.7/91.6	92.9/92.5	73.9/90.2	91.0	90.8	99.2	89.7	94.5	50.2
+	7	Zihang Dai	Funnel-Transformer (Ensemble B10-10-10H1024)	<a href="#">↗</a>	89.7	70.5	97.5	93.4/91.2	92.6/92.3	75.4/90.7	91.4	91.1	95.8	90.0	94.5	51.6
+	8	ELECTRA Team	ELECTRA-Large + Standard Tricks	<a href="#">↗</a>	89.4	71.7	97.1	93.1/90.7	92.9/92.5	75.6/90.8	91.3	90.8	95.8	89.8	91.8	50.7
+	9	Huawei Noah's Ark Lab	NEZHA-Large		89.1	69.9	97.3	93.3/91.0	92.4/91.9	74.2/90.6	91.0	90.7	95.7	88.7	93.2	47.9
+	10	Microsoft D365 AI & UMD	FreeLB-RoBERTa (ensemble)	<a href="#">↗</a>	88.4	68.0	96.8	93.1/90.8	92.3/92.1	74.8/90.3	91.1	90.7	95.6	88.7	89.0	50.1
	11	Junjie Yang	HIRE-RoBERTa	<a href="#">↗</a>	88.3	68.6	97.1	93.0/90.7	92.4/92.0	74.3/90.2	90.7	90.4	95.5	87.9	89.0	49.3
	12	Facebook AI	RoBERTa	<a href="#">↗</a>	88.1	67.8	96.7	92.3/89.8	92.2/91.9	74.3/90.2	90.8	90.2	95.4	88.2	89.0	48.7
+	13	Microsoft D365 AI & MSR AI	MT-DNN-ensemble	<a href="#">↗</a>	87.6	68.4	96.5	92.7/90.3	91.1/90.7	73.7/89.9	87.9	87.4	96.0	86.3	89.0	42.8
	14	GLUE Human Baselines	GLUE Human Baselines	<a href="#">↗</a>	87.1	66.4	97.8	86.3/80.8	92.7/92.6	59.5/80.4	92.0	92.8	91.2	93.6	95.9	-
	15	Stanford Hazy Research	Snorkel MeTaL	<a href="#">↗</a>	83.2	63.8	96.2	91.5/88.5	90.1/89.7	73.1/89.9	87.6	87.2	93.9	80.9	65.1	39.9

# GLUE Benchmark

Rank	Name	Model	URL	Score	CoLA	SST-2	MRPC	STS-B	QQP	MNLI-m	MNLI-mm	QNLI	RTE	WNLI	AX	
1	HFL iFLYTEK	MacALBERT + DKM		90.7	74.8	97.0	94.5/92.6	92.8/92.6	74.7/90.6	91.3	91.1	97.8	92.0	94.5	52.6	
+	2	Alibaba DAMO NLP	StructBERT + TAPT	<a href="#">↗</a>	90.6	75.3	97.3	93.9/91.9	93.2/92.7	74.8/91.0	90.9	90.7	97.4	91.2	94.5	49.1
+	3	PING-AN Omni-Sinitic	ALBERT + DAAF + NAS		90.6	73.5	97.2	94.0/92.0	93.0/92.4	76.1/91.0	91.6	91.3	97.5	91.7	94.5	51.2
	4	ERNIE Team - Baidu	ERNIE	<a href="#">↗</a>	90.4	74.4	97.5	93.5/91.4	93.0/92.6	75.2/90.9	91.4	91.0	96.6	90.9	94.5	51.7
	5	T5 Team - Google	T5	<a href="#">↗</a>	90.3	71.6	97.5	92.8/90.4	93.1/92.8	75.1/90.6	92.2	91.9	96.9	92.8	94.5	53.1
	6	Microsoft D365 AI & MSR AI & GATECH	MT-DNN-SMART	<a href="#">↗</a>	89.9	69.5	97.5	93.7/91.6	92.9/92.5	73.9/90.2	91.0	90.8	99.2	89.7	94.5	50.2
+	7	Zihang Dai	Funnel-Transformer (Ensemble B10-10-10H1024)	<a href="#">↗</a>	89.7	70.5	97.5	93.4/91.2	92.6/92.3	75.4/90.7	91.4	91.1	95.8	90.0	94.5	51.6
+	8	ELECTRA Team	ELECTRA-Large + Standard Tricks	<a href="#">↗</a>	89.4	71.7	97.1	93.1/90.7	92.9/92.5	75.6/90.8	91.3	90.8	95.8	89.8	91.8	50.7
+	9	Huawei Noah's Ark Lab	NEZHA-Large		89.1	69.9	97.3	93.3/91.0	92.4/91.9	74.2/90.6	91.0	90.7	95.7	88.7	93.2	47.9
+	10	Microsoft D365 AI & UMD	FreeLB-RoBERTa (ensemble)	<a href="#">↗</a>	88.4	68.0	96.8	93.1/90.8	92.3/92.1	74.8/90.3	91.1	90.7	95.6	88.7	89.0	50.1
	11	Junjie Yang	HIRE-RoBERTa	<a href="#">↗</a>	88.3	68.6	97.1	93.0/90.7	92.4/92.0	74.3/90.2	90.7	90.4	95.5	87.9	89.0	49.3
	12	Facebook AI	RoBERTa	<a href="#">↗</a>	88.1	67.8	96.7	92.3/89.8	92.2/91.9	74.3/90.2	90.8	90.2	95.4	88.2	89.0	48.7
+	13	Microsoft D365 AI & MSR AI	MT-DNN-ensemble	<a href="#">↗</a>	87.6	68.4	96.5	92.7/90.3	91.1/90.7	73.7/89.9	87.9	87.4	96.0	86.3	89.0	42.8
	14	GLUE Human Baselines	GLUE Human Baselines	<a href="#">↗</a>	87.1	66.4	97.8	86.3/80.8	92.7/92.6	59.5/80.4	92.0	92.8	91.2	93.6	95.9	-
	15	Stanford Hazy Research	Snorkel MeTaL	<a href="#">↗</a>	83.2	63.8	96.2	91.5/88.5	90.1/89.7	73.1/89.9	87.6	87.2	93.9	80.9	65.1	39.9

## SYSTEM PROMPT (HUMAN-WRITTEN)

In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

MODEL COMPLETION (MACHINE-WRITTEN, 10 TRIES)

The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Pérez and the others then ventured further into the valley. "By the time we reached the top of one peak, the water looked blue, with some crystals on top," said Pérez.

Source: OpenAI, "Better Language Models and Their Implications"  
<https://openai.com/blog/better-language-models/>

Can Attention/Transformers be used  
from more than text processing?

# ViLBERT: A Visolinguistic Transformer



pop artist performs at the festival in a city.

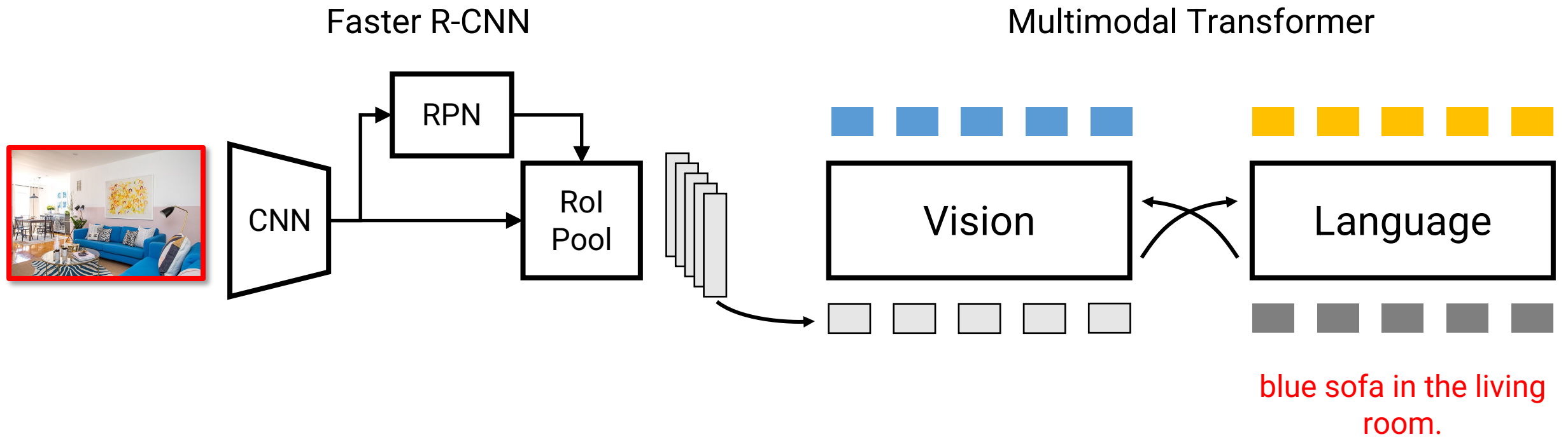


a worker helps to clear the debris.



blue sofa in the living room.

# ViLBERT: A Visolinguistic Transformer

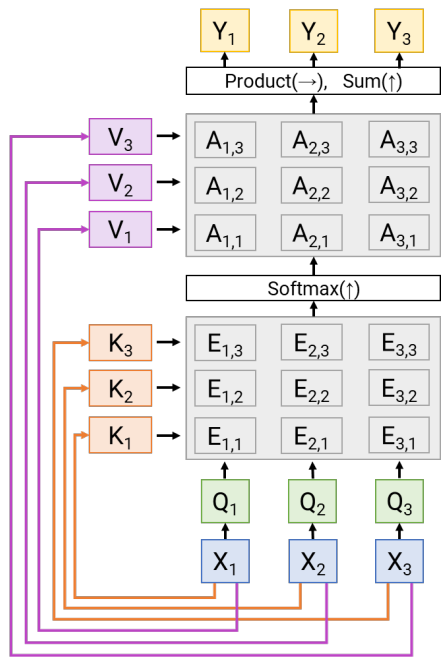




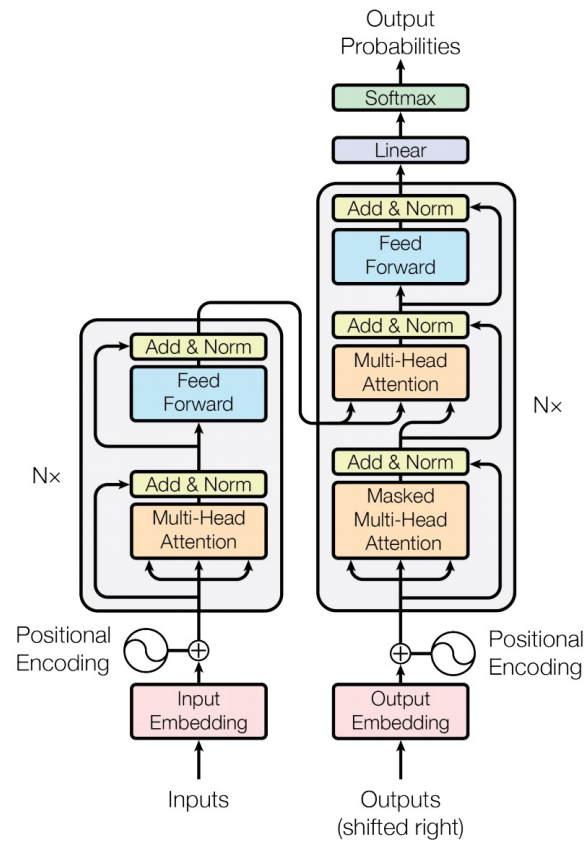
ViLBERT Demo:  
<https://vilbert.cloudcv.org/>

# Summary

## Self-Attention



## Transformer Model



## ViLBERT

