

Topics:

- Attention and Transformers

**CS 4644-DL / 7643-A**

**ZSOLT KIRA**

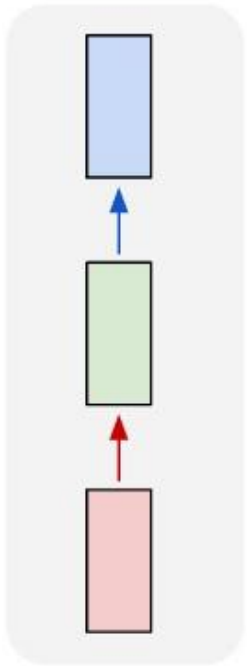
- **Assignment 3 out**
  - **Due March 8th 11:59pm EST**
- **Meta office hours Friday 3pm ET on Attention/Language Models**

# Lecture Outline

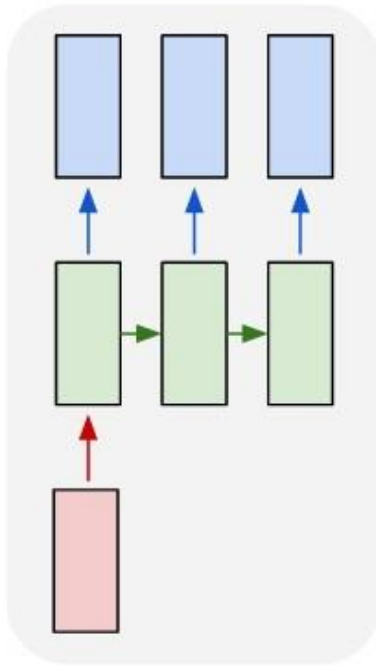
- Machine Translation with RNNs
- RNNs with Attention
- From Attention to Transformers
- What can Transformers do?

# Sequence Modeling with RNNs

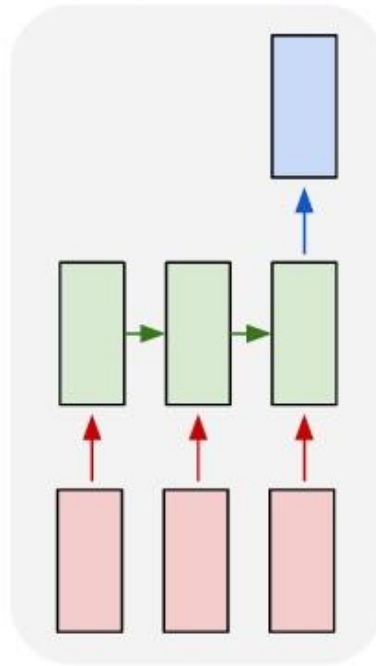
one to one



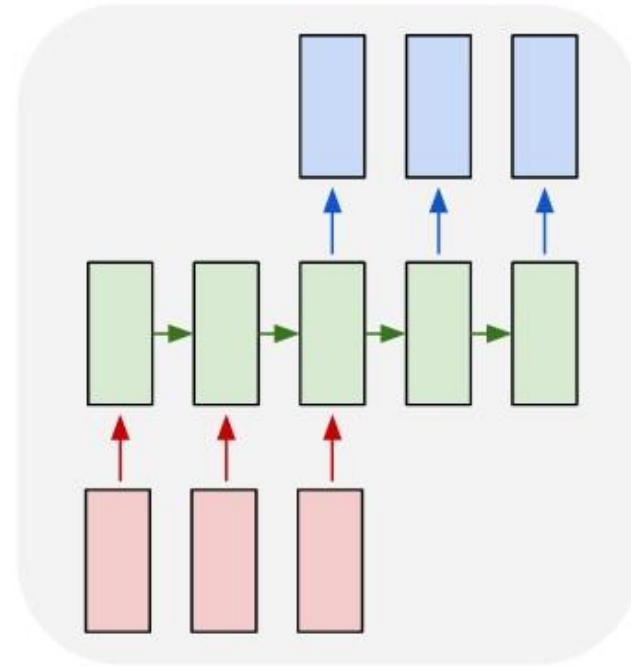
one to many



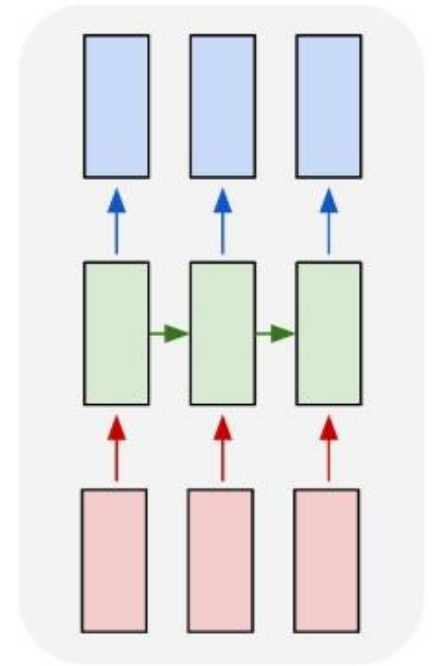
many to one



many to many

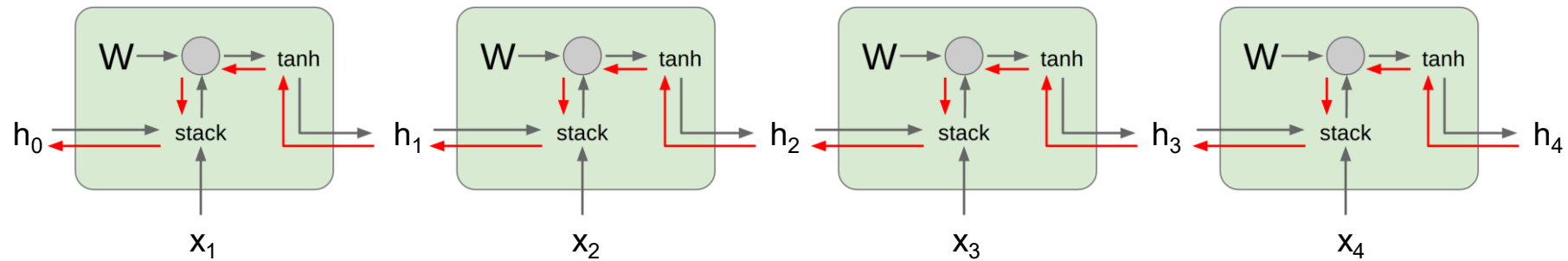


many to many



# Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994  
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



$$\begin{aligned}h_t &= \tanh(W_{hh}h_{t-1} + W_{hx}x_t) \\ &= \tanh\left((W_{hh} \quad W_{hx}) \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right) \\ &= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)\end{aligned}$$

# How can we train this on language?

- Supervised Learning:
  - Sentiment analysis (sentence -> negative/neutral/positive) labeled by humans
  - Translation -> English and equivalent other language
- Self-supervised: Predict the next letter or word!
  - This is **extremely powerful!!**
  - In order to predict what's next, it needs to really understand not just language statistics but world knowledge!
    - Of course, we need scale for this level of loss reduction / understanding

- **Training:** A large corpus of text from the web
  - Note: No annotation required! It's just "the text"
- **Inference:** Just generate me new text
  - Can condition on some initial input (**prompt**)

```
#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setew.h>
#include <asm/pgproto.h>

#define REG_PG    vesa_slot_addr_pack
#define PFM_NOCOMP  AFSR(0, load)
#define STACK_DDR(type)      (func)

#define SWAP_ALLOCATE(nr)      (e)
#define emulate_sigs()  arch_get_unaligned_child()
#define access_rw(TST)  asm volatile("movd %%esp, %0, %3" : : "r" (0)); \
    if (__type & DO_READ)

static void stat_PC_SEC __read_mostly offsetof(struct seq_argsqueue, \
    pC>[1]);

static void
os_prefix(unsigned long sys)
{
#ifdef CONFIG_PREEMPT
    PUT_PARAM_RAID(2, sel) = get_state_state();
    set_pid_sum((unsigned long)state, current_state_str(),
        (unsigned long)-1->lr_full; low;
}

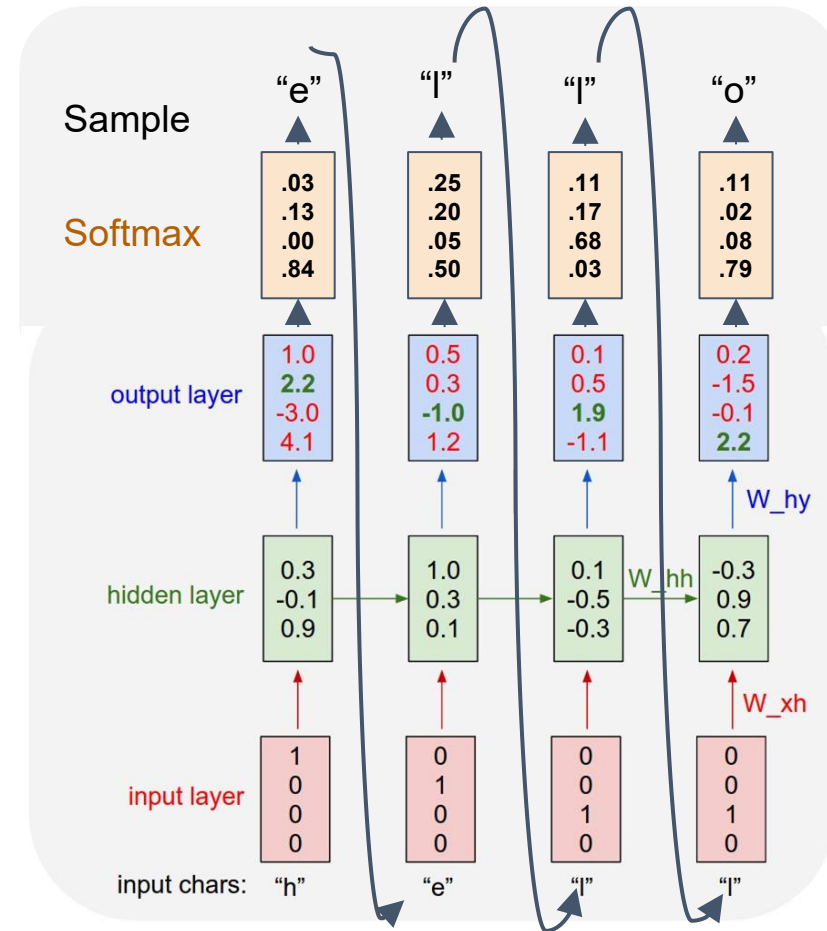
```

# Test Time: Sample / Argmax / Beam Search

## Example: Character-level Language Model Sampling

Vocabulary:  
[h,e,l,o]

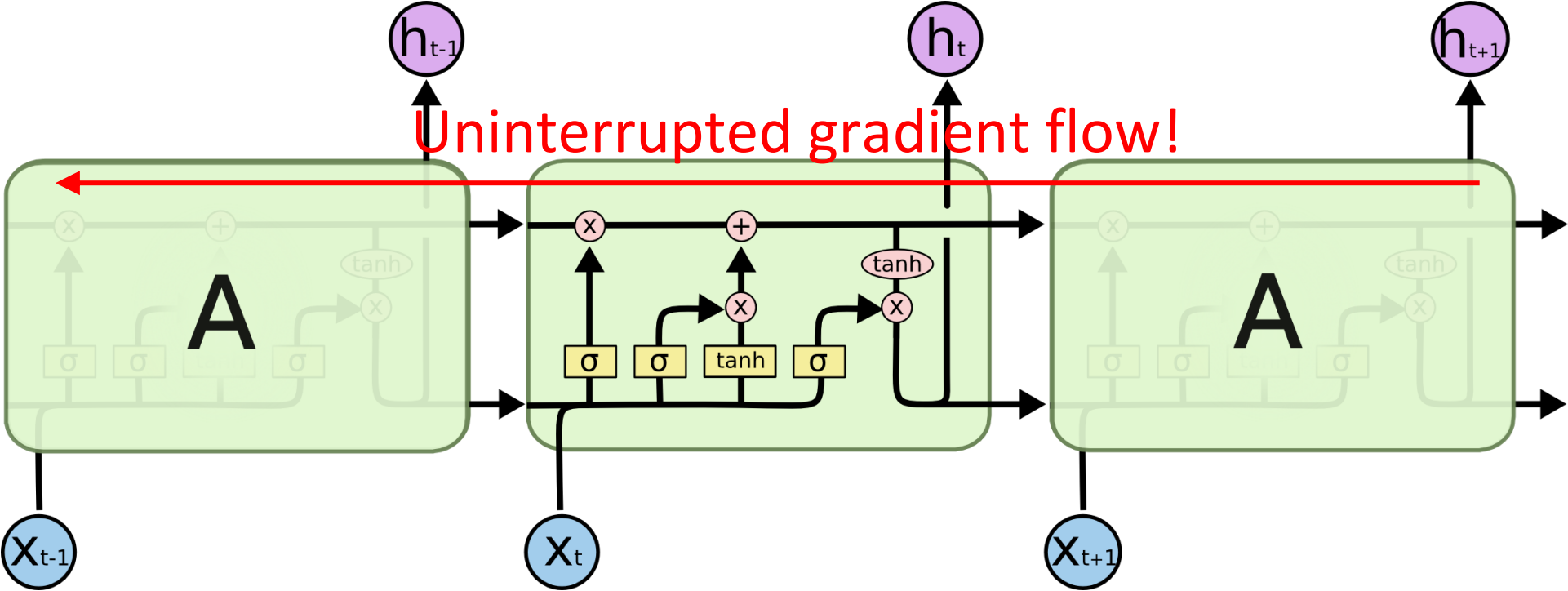
At test-time sample  
characters one at a  
time, feed back to  
model



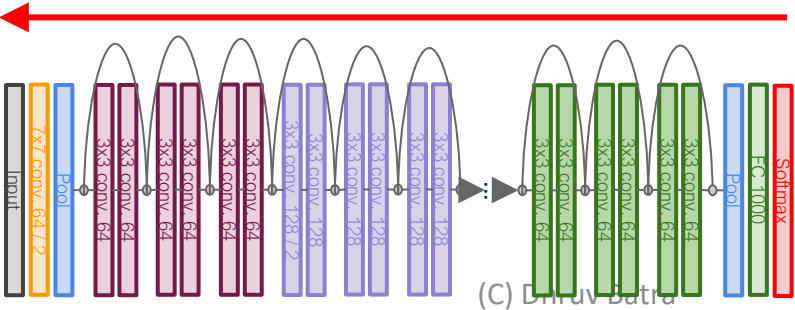
Can also feed in predictions during training (student forcing)



# LSTMs Intuition: Additive Updates



Similar to ResNet!



(C) Dhruv Batra

```

#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setew.h>
#include <asm/pgproto.h>

#define REG_PG    vesa_slot_addr_pack
#define PFM_NOCOMP  AFSR(0, load)
#define STACK_DDR(type)      (func)

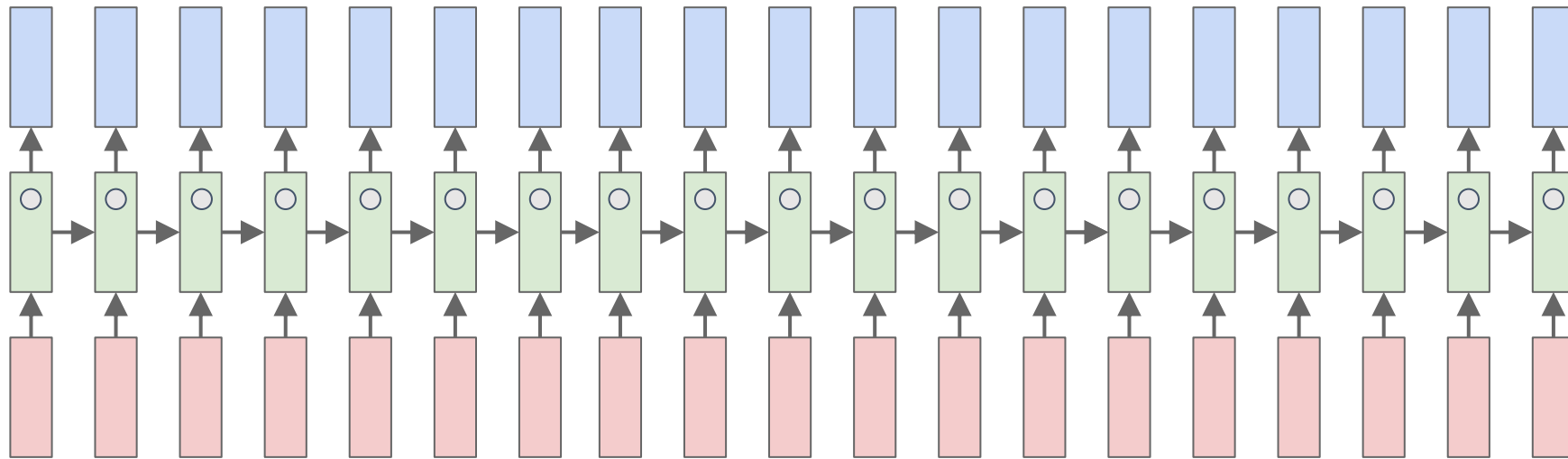
#define SWAP_ALLOCATE(nr)      (e)
#define emulate_sigs()  arch_get_unaligned_child()
#define access_rw(TST)  asm volatile("movd %%esp, %0, %3" : : "r" (0)); \
    if (__type & DO_READ)

static void stat_PC_SEC __read_mostly offsetof(struct seq_argsqueue, \
    pC>[1]);

static void
os_prefix(unsigned long sys)
{
#ifdef CONFIG_PREEMPT
    PUT_PARAM_RAID(2, sel) = get_state_state();
    set_pid_sum((unsigned long)state, current_state_str(),
        (unsigned long)-1->lr_full; low;
}

```

# Searching for interpretable cells



Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

# Searching for interpretable cells

```
/* Unpack a filter field's string representation from user-space
 * buffer. */
char *audit_unpack_string(void **bufp, size_t *remain, size_t len)
{
    char *str;
    if (!*bufp || (len == 0) || (len > *remain))
        return ERR_PTR(-EINVAL);
    /* Of the currently implemented string fields, PATH_MAX
     * defines the longest valid length.
     */
}
```

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission

# Searching for interpretable cells

"You mean to imply that I have nothing to eat out of.... On the contrary, I can supply you with everything even if you want to give dinner parties," warmly replied Chichagov, who tried by every word he spoke to prove his own rectitude and therefore imagined Kutuzov to be animated by the same desire.

Kutuzov, shrugging his shoulders, replied with his subtle penetrating smile: "I meant merely to say what I said."

quote detection cell

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission

# Searching for interpretable cells

Cell sensitive to position in line:

The sole importance of the crossing of the Berezina lies in the fact that it plainly and indubitably proved the fallacy of all the plans for cutting off the enemy's retreat and the soundness of the only possible line of action--the one Kutuzov and the general mass of the army demanded--namely, simply to follow the enemy up. The French crowd fled at a continually increasing speed and all its energy was directed to reaching its goal. It fled like a wounded animal and it was impossible to block its path. This was shown not so much by the arrangements it made for crossing as by what took place at the bridges. When the bridges broke down, unarmed soldiers, people from Moscow and women with children who were with the French transport, all--carried on by vis inertiae--pressed forward into boats and into the ice-covered water and did not, surrender.

line length tracking cell

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission

# Searching for interpretable cells

```
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
                           siginfo_t *info)
{
    int sig = next_signal(pending, mask);
    if (sig) {
        if (current->notifier) {
            if (sigismember(current->notifier_mask, sig)) {
                if (!(current->notifier)(current->notifier_data)) {
                    clear_thread_flag(TIF_SIGPENDING);
                    return 0;
                }
            }
        }
        collect_signal(sig, pending, info);
    }
    return sig;
}
```

if statement cell

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission

# Searching for interpretable cells

Cell that turns on inside comments and quotes:

```
/* Duplicate LSM field information. The lsm_rule is opaque, so
 * re-initialized. */
static inline int audit_dupe_lsm_field(struct audit_field *df,
                                     struct audit_field *sf)
{
    int ret = 0;
    char *lsm_str;
    /* our own copy of lsm_str */
    lsm_str = kstrdup(sf->lsm_str, GFP_KERNEL);
    if (unlikely(!lsm_str))
        return -ENOMEM;
    df->lsm_str = lsm_str;
    /* our own (refreshed) copy of lsm_rule */
    ret = security_audit_rule_init(df->type, df->op, df->lsm_str,
                                  (void **)&df->lsm_rule);
    /* Keep currently invalid fields around in case they
     * become valid after a policy reload. */
    if (ret == -EINVAL) {
        pr_warn("audit rule for LSM '%s' is invalid\n",
               df->lsm_str);
        ret = 0;
    }
    return ret;
}
```

quote/comment cell

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission



# Searching for interpretable cells

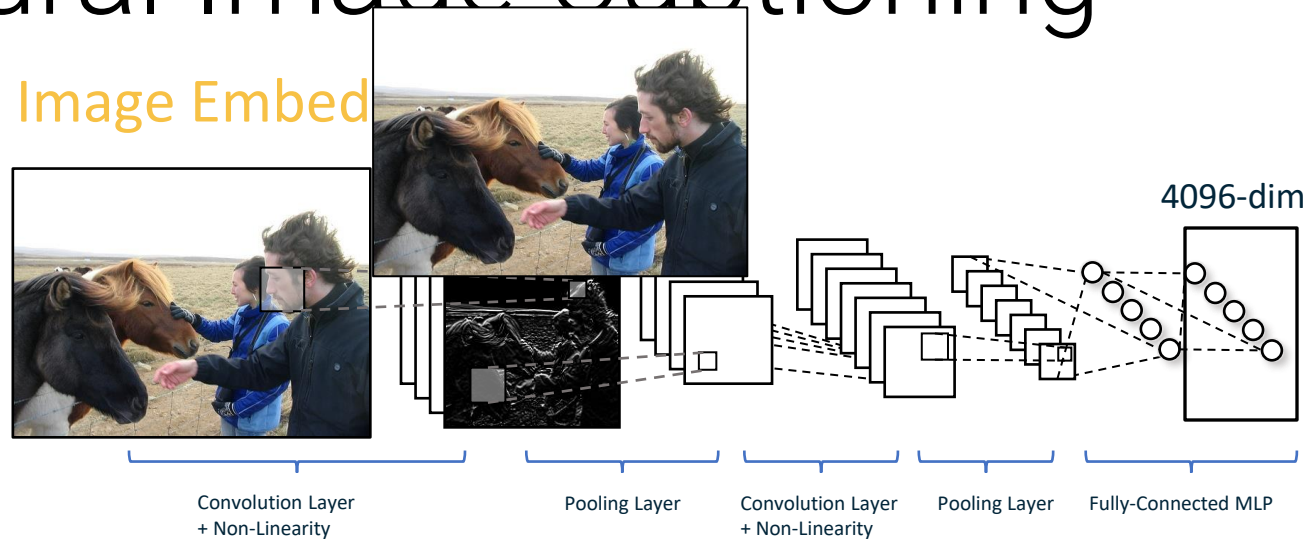
```
#ifdef CONFIG_AUDIT_SYSCALL
static inline int audit_match_class_bits(int class, u32 *mask)
{
    int i;
    if (classes[class]) {
        for (i = 0; i < AUDIT_BITMASK_SIZE; i++)
            if (mask[i] & classes[class][i])
                return 0;
    }
    return 1;
}
```

code depth cell

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

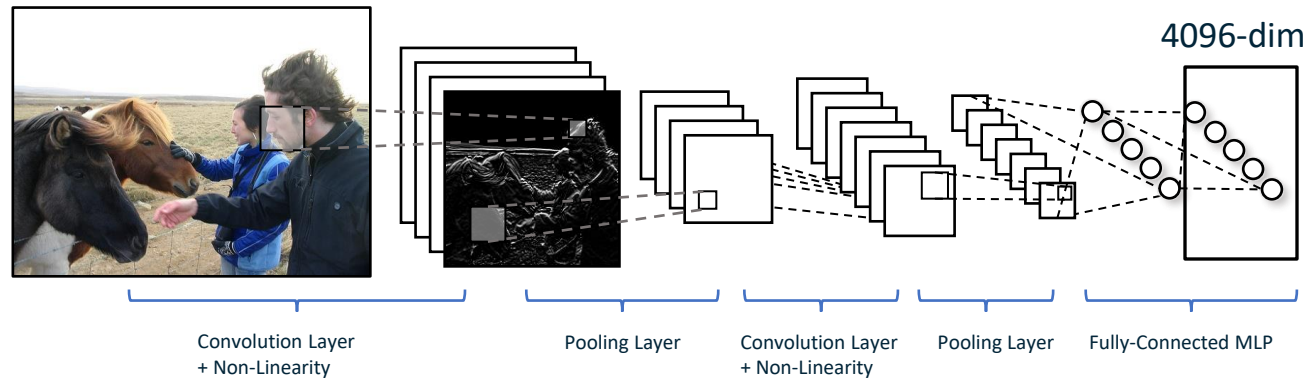
Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission

# Neural Image Captioning

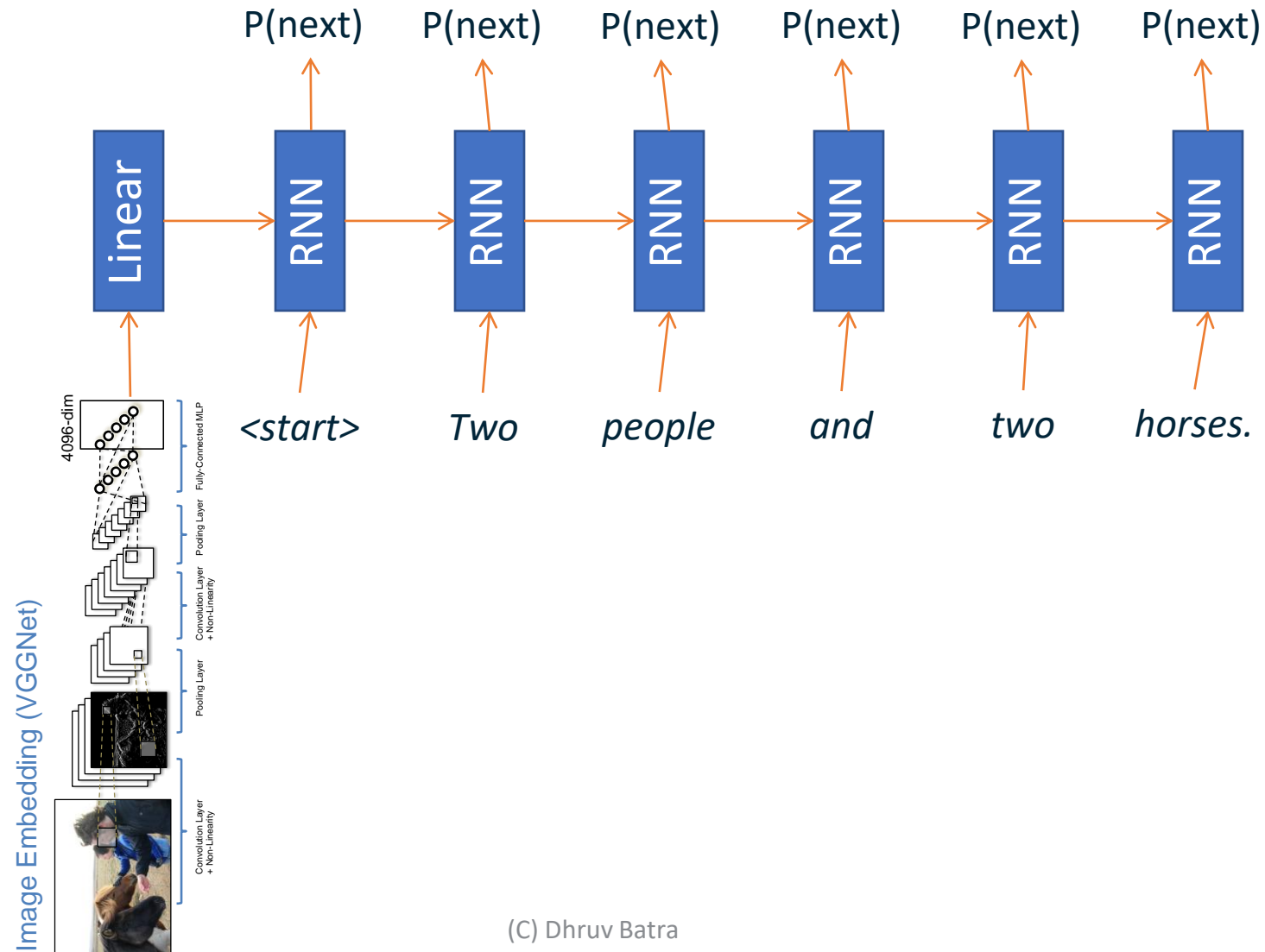


# Neural Image Captioning

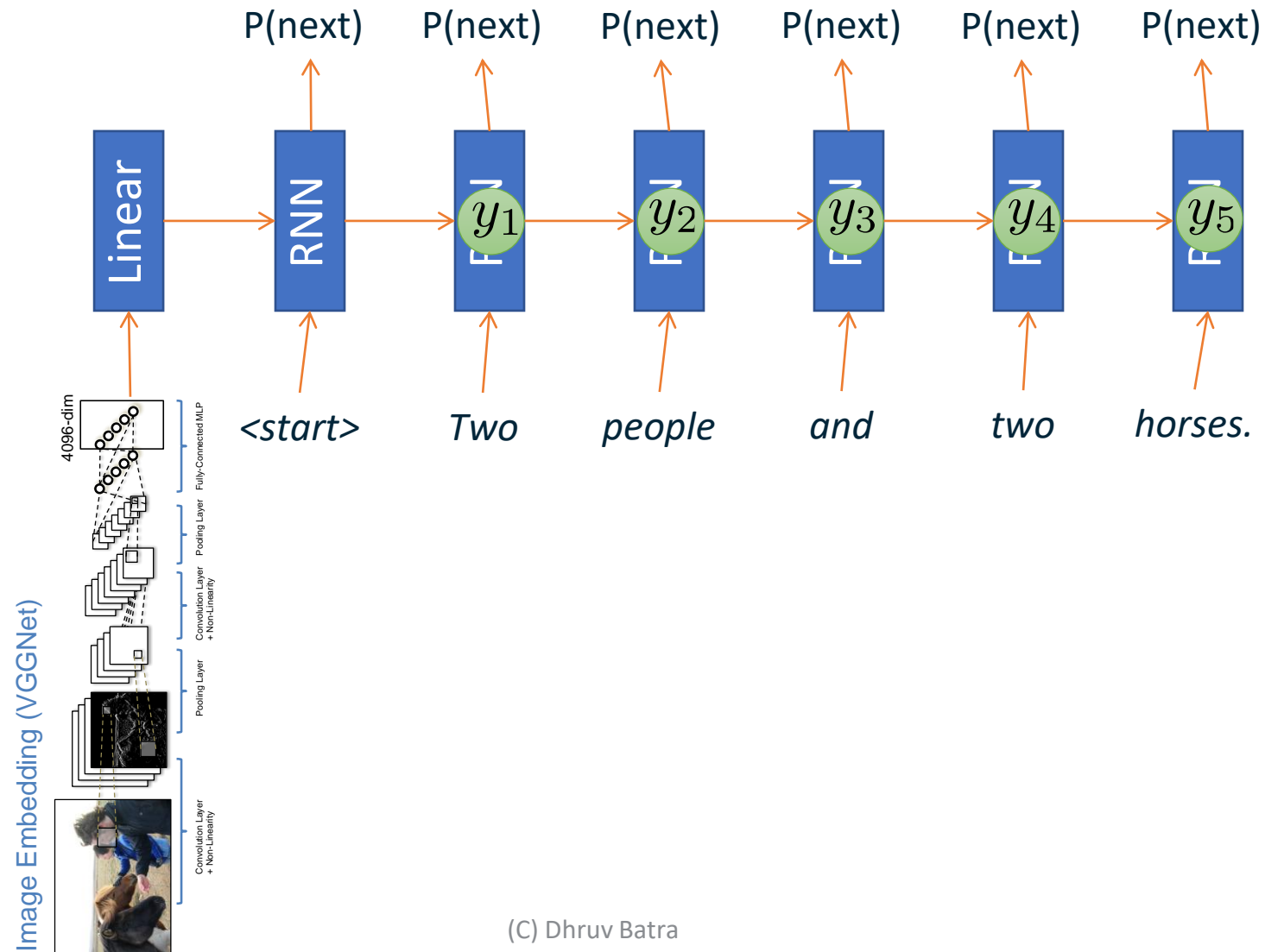
## Image Embedding (VGGNet)



# Neural Image Captioning



# Neural Image Captioning



# Machine Translation

we are eating bread



estamos comiendo pan

# Machine Translation



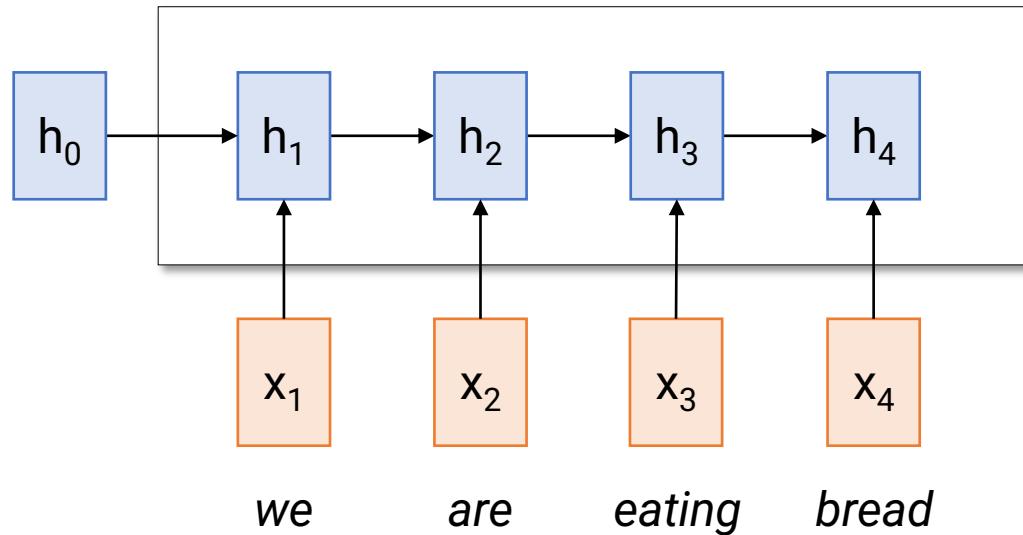
we are eating bread



estamos comiendo pan

# Machine Translation with RNNs

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

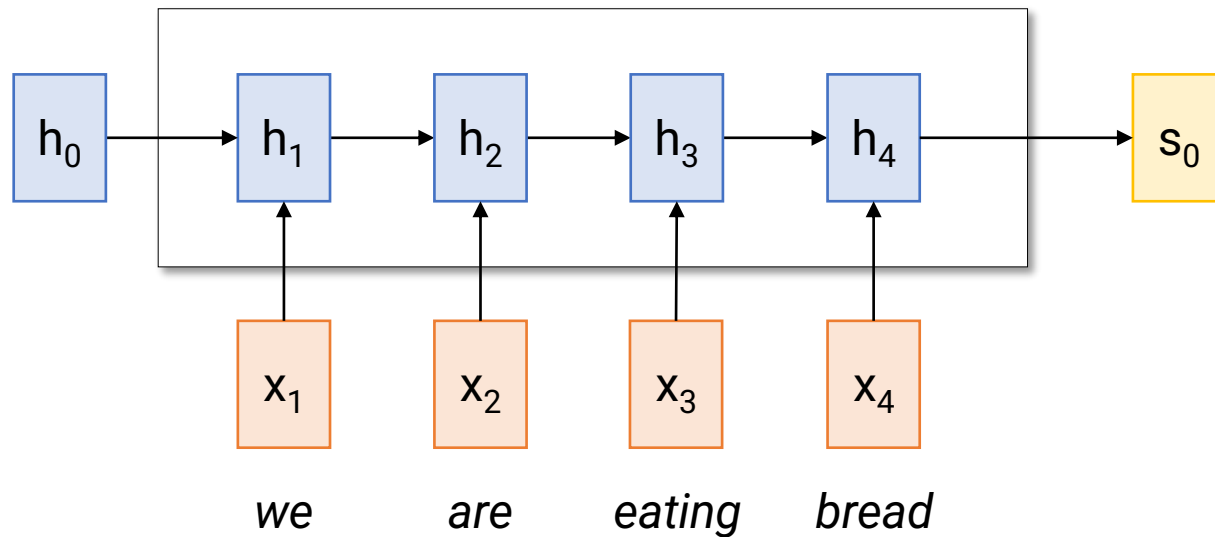




# Machine Translation with RNNs

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

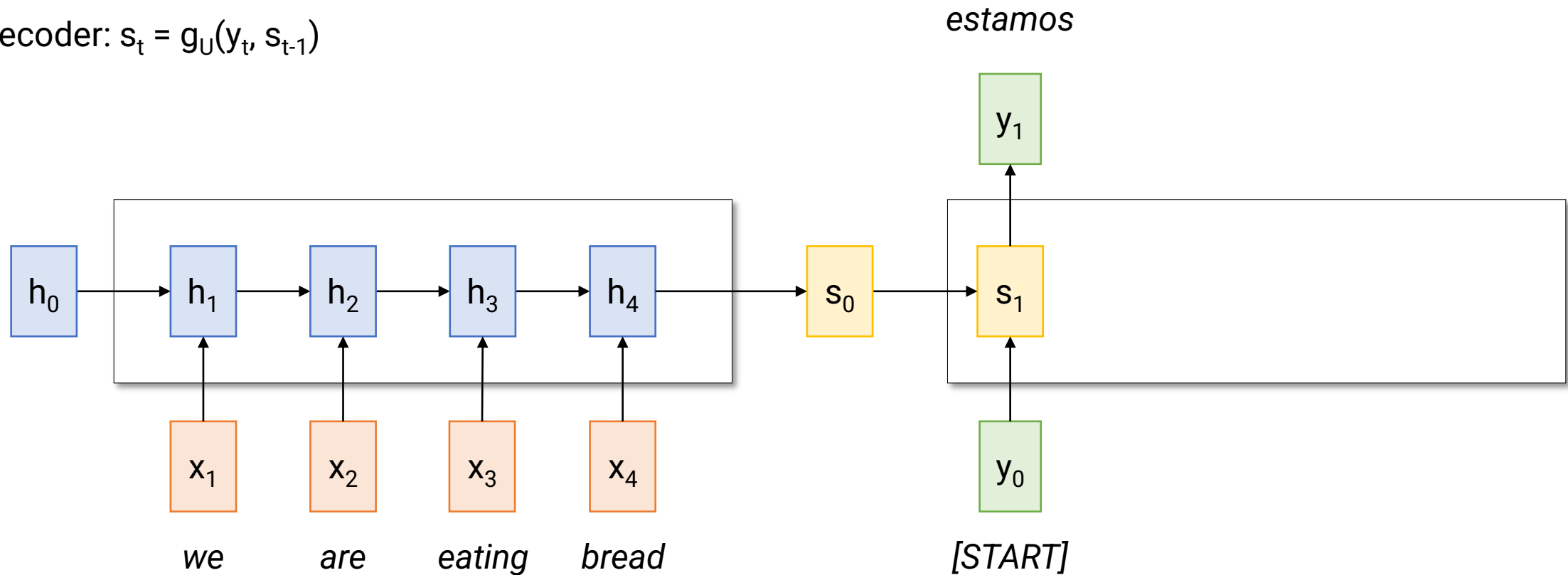
$s_0 = h_4$



# Machine Translation with RNNs

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

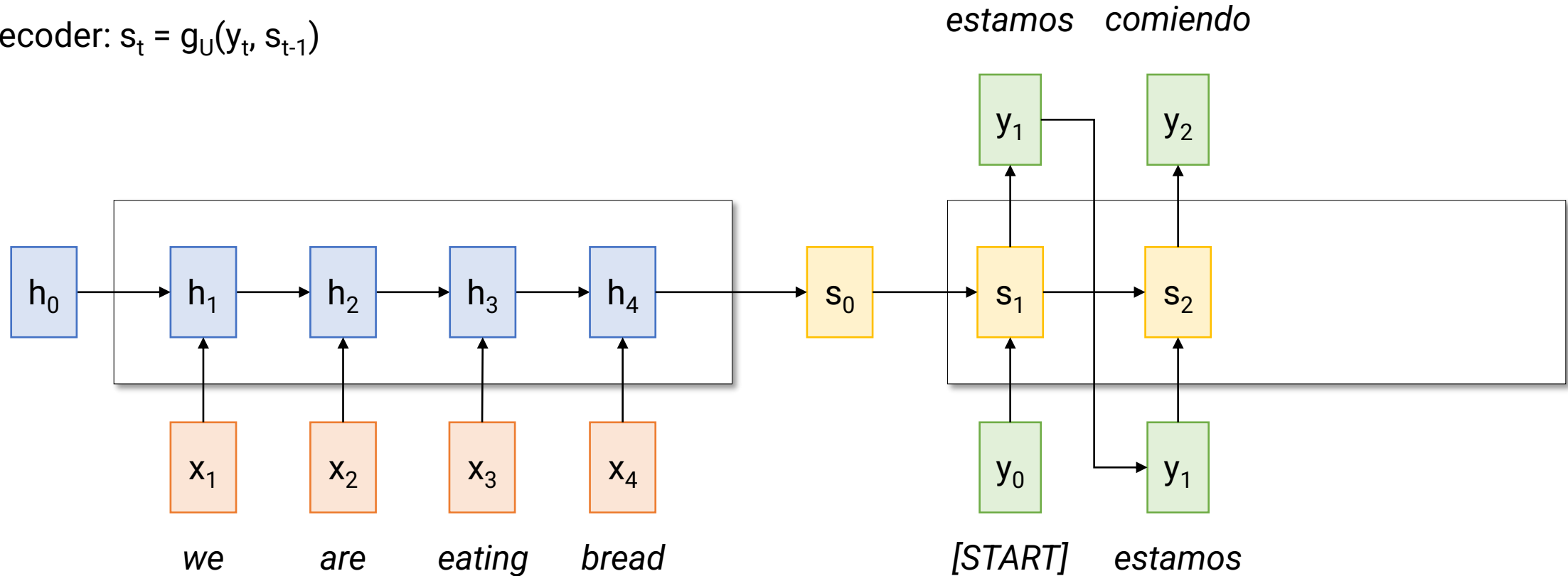
Decoder:  $s_t = g_U(y_t, s_{t-1})$



# Machine Translation with RNNs

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

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

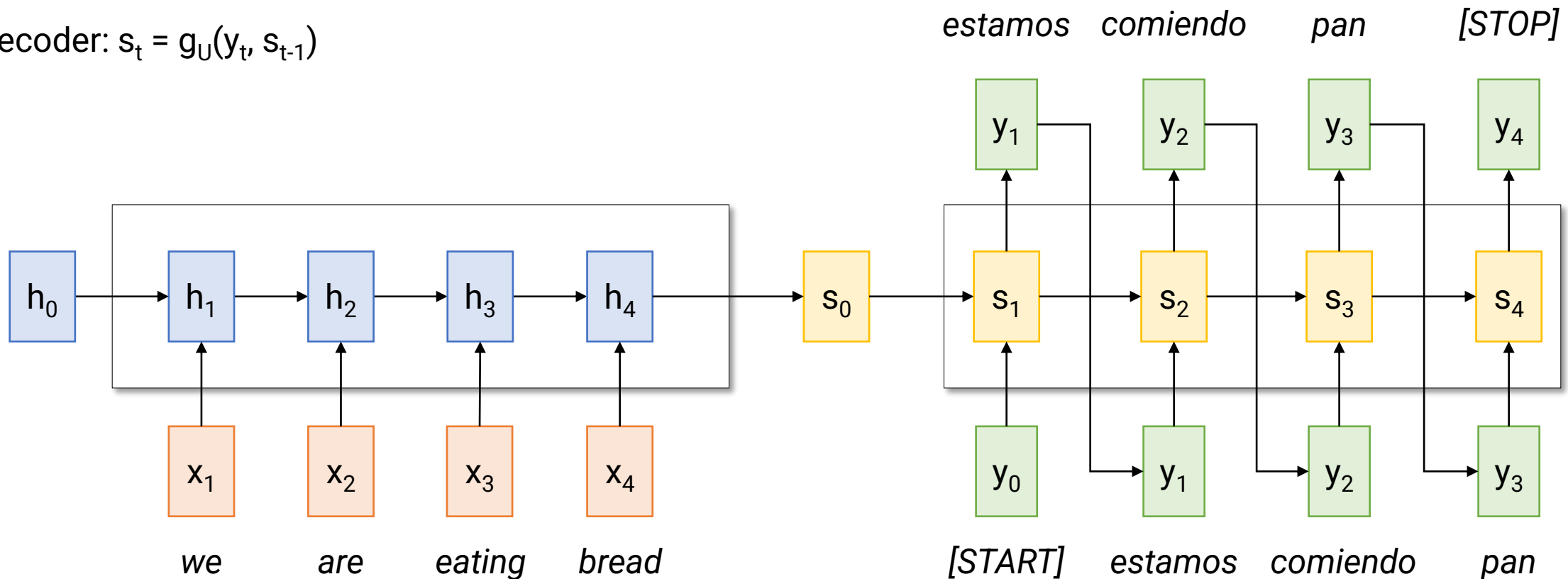


# Machine Translation with RNNs

Note [START]/[STOP] words. This can be treated as representation for entire sentence

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

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

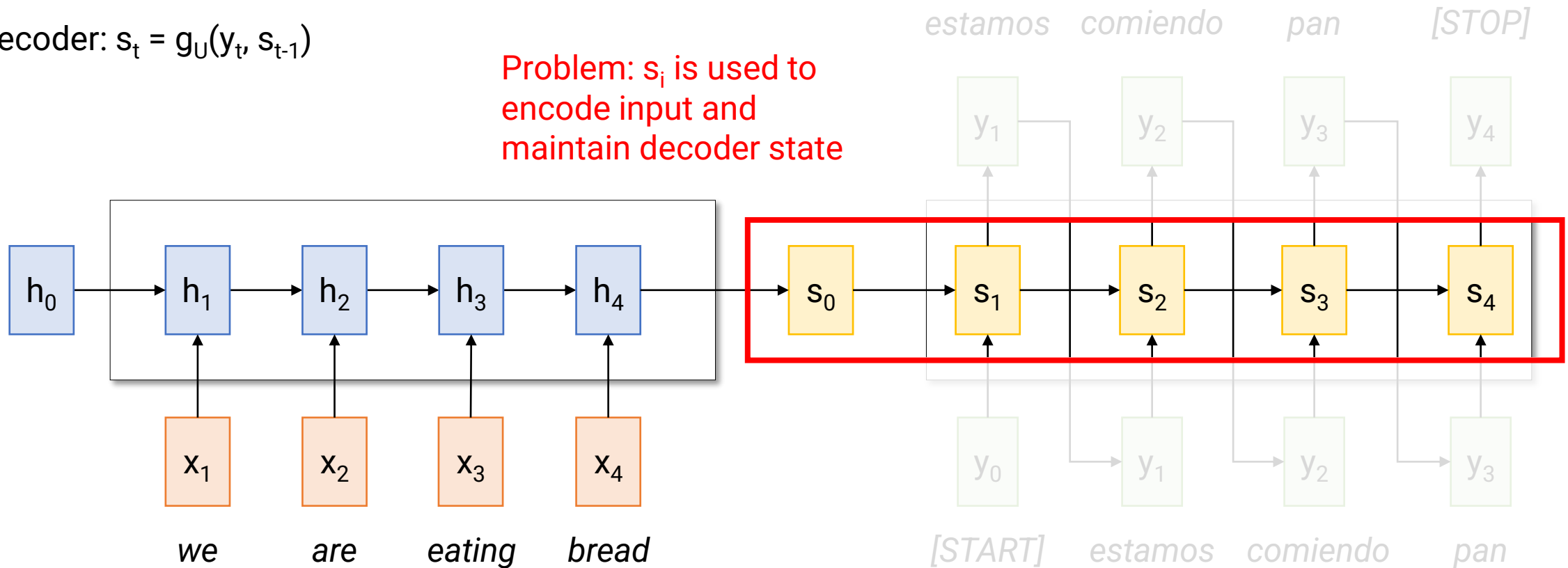


# Machine Translation with RNNs

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

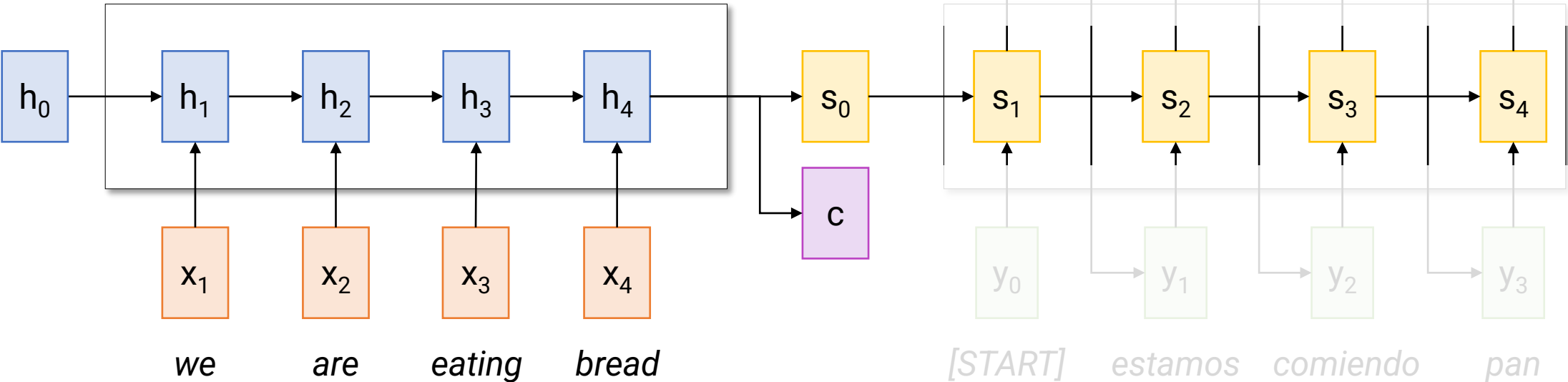


# Machine Translation with RNNs

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  $h_4$

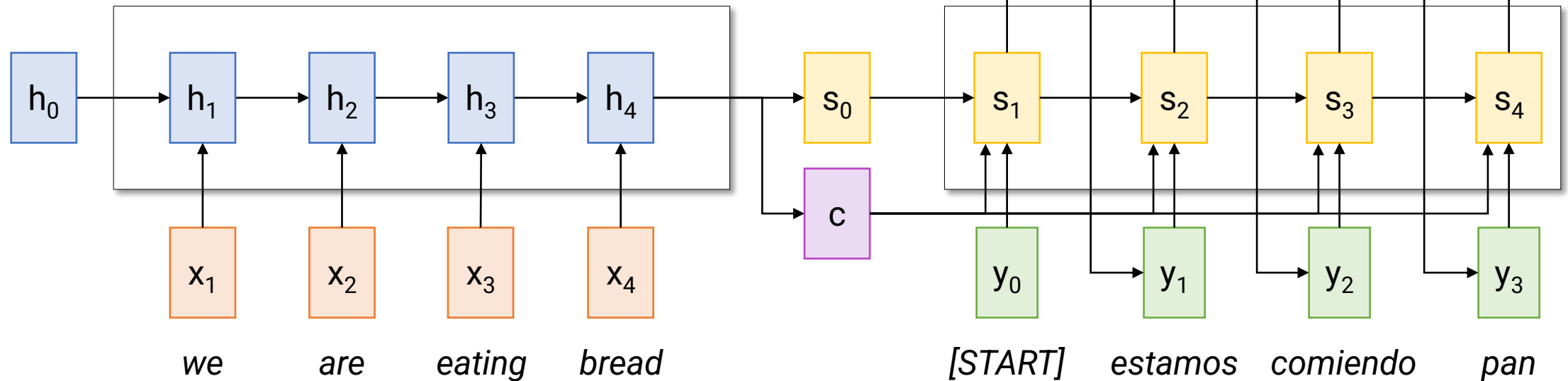


# Machine Translation with RNNs

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

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

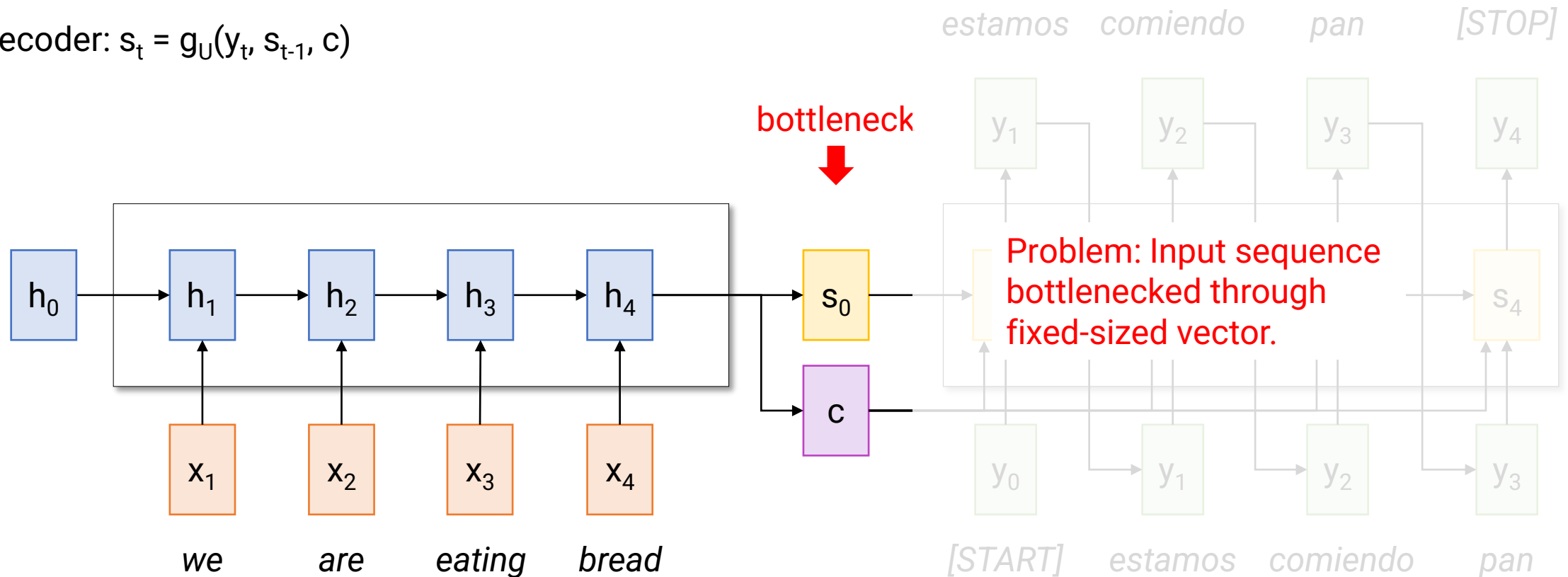
Solution: add a context vector  $\mathbf{c} = h_4$  and predict  $s_0$  from  $h_4$



# Machine Translation with RNNs

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

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

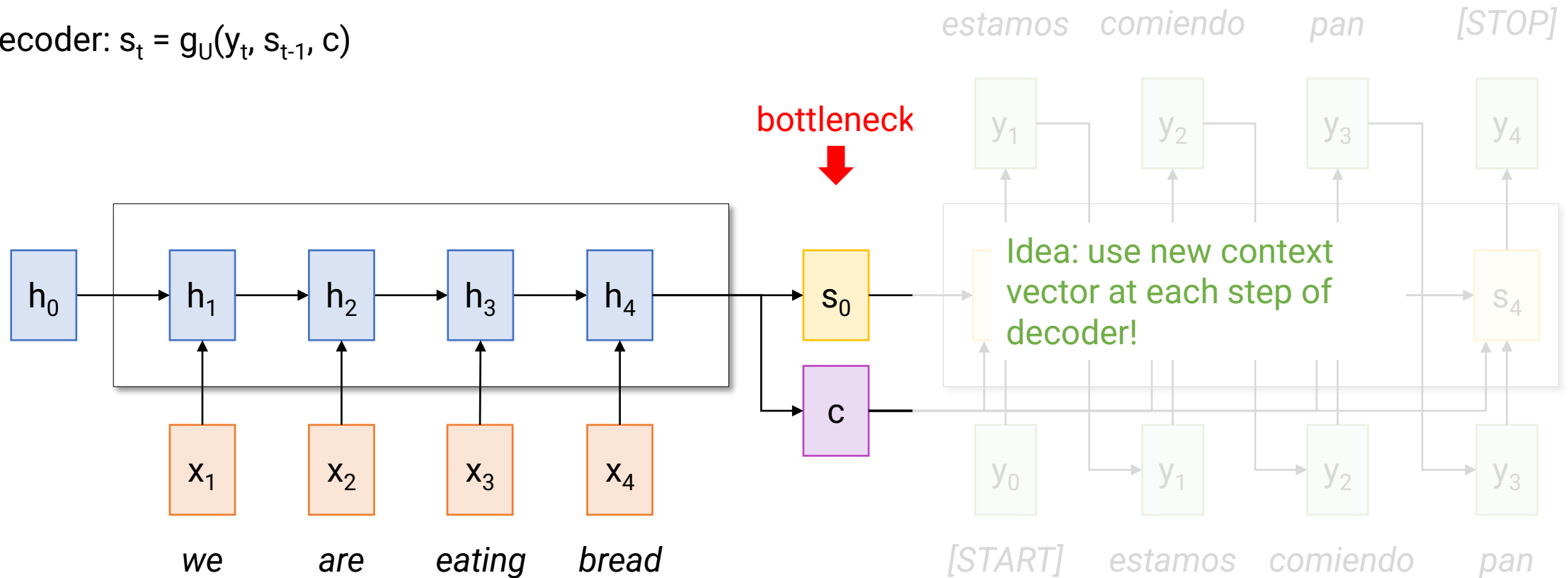




# Machine Translation with RNNs

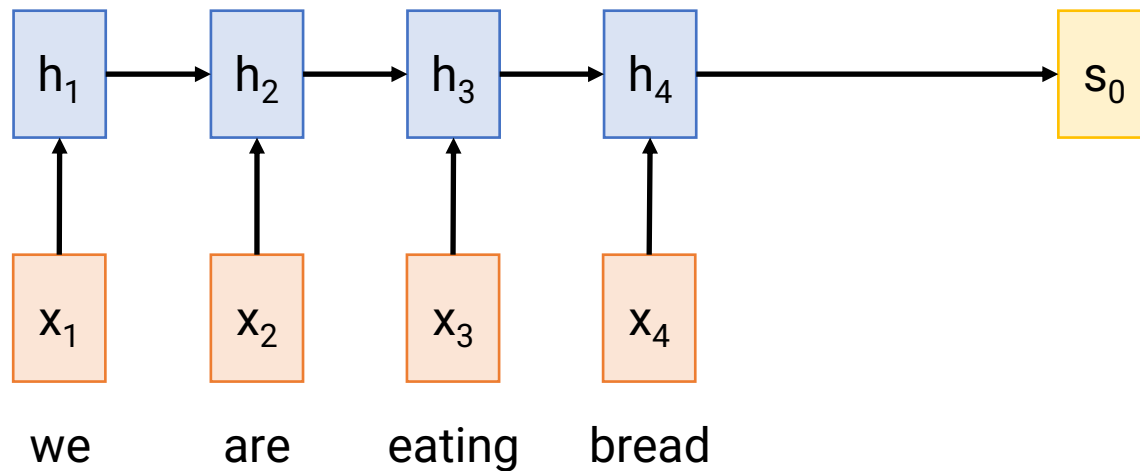
Encoder:  $h_t = f_W(x_t, h_{t-1})$

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



# Machine Translation with RNNs **and Attention**

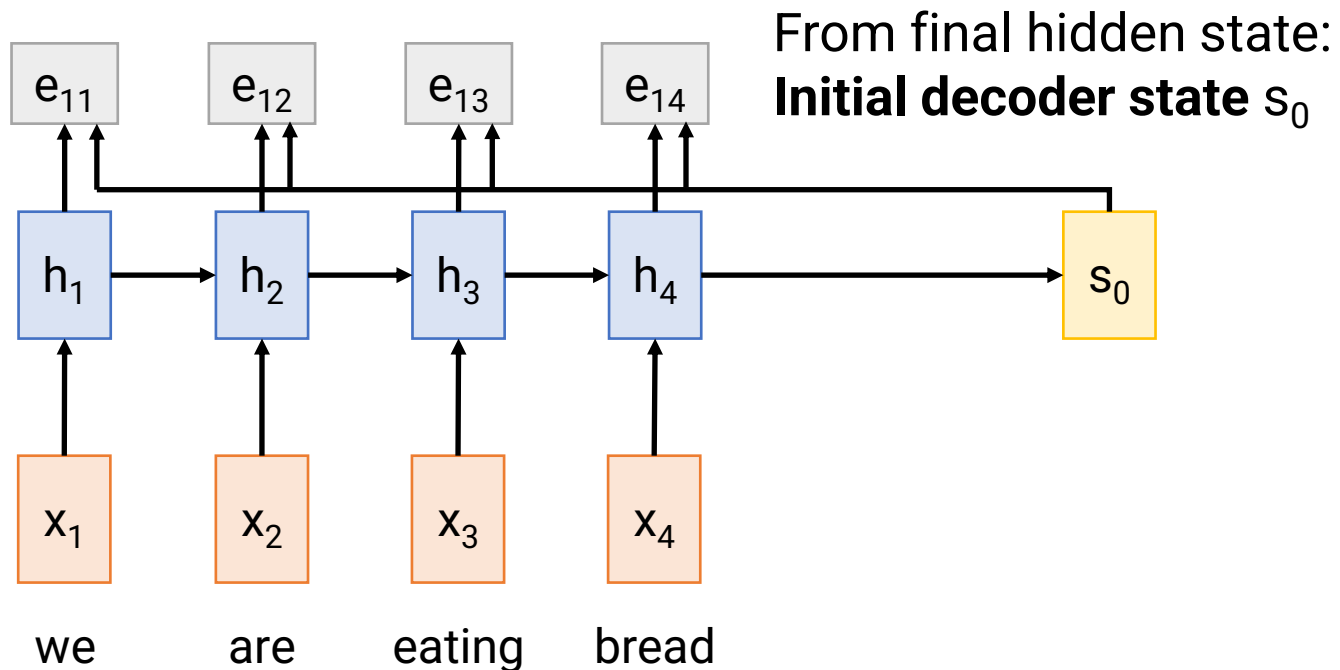
From final hidden state:  
**Initial decoder state  $s_0$**



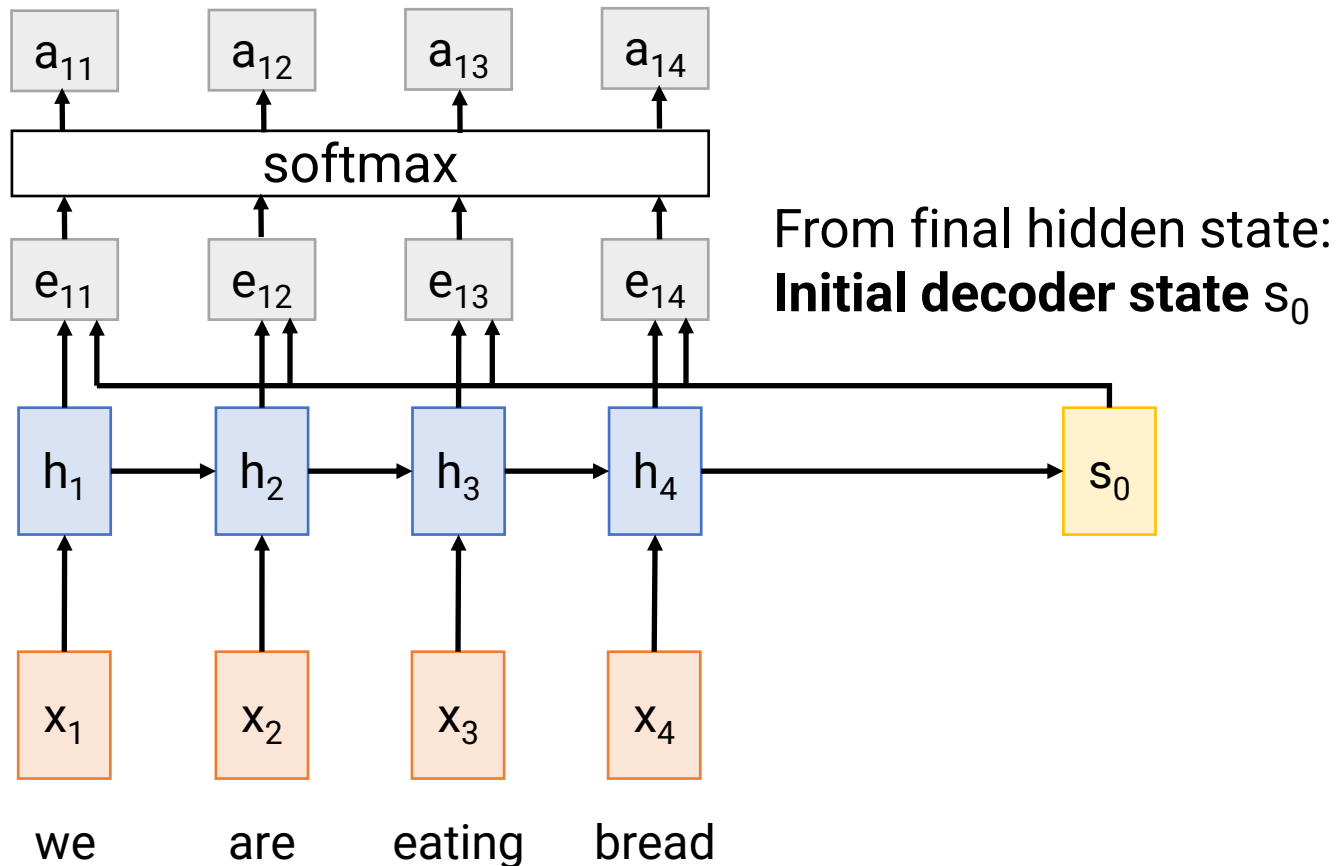
# Machine Translation with RNNs **and Attention**

Compute **alignment scores**

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



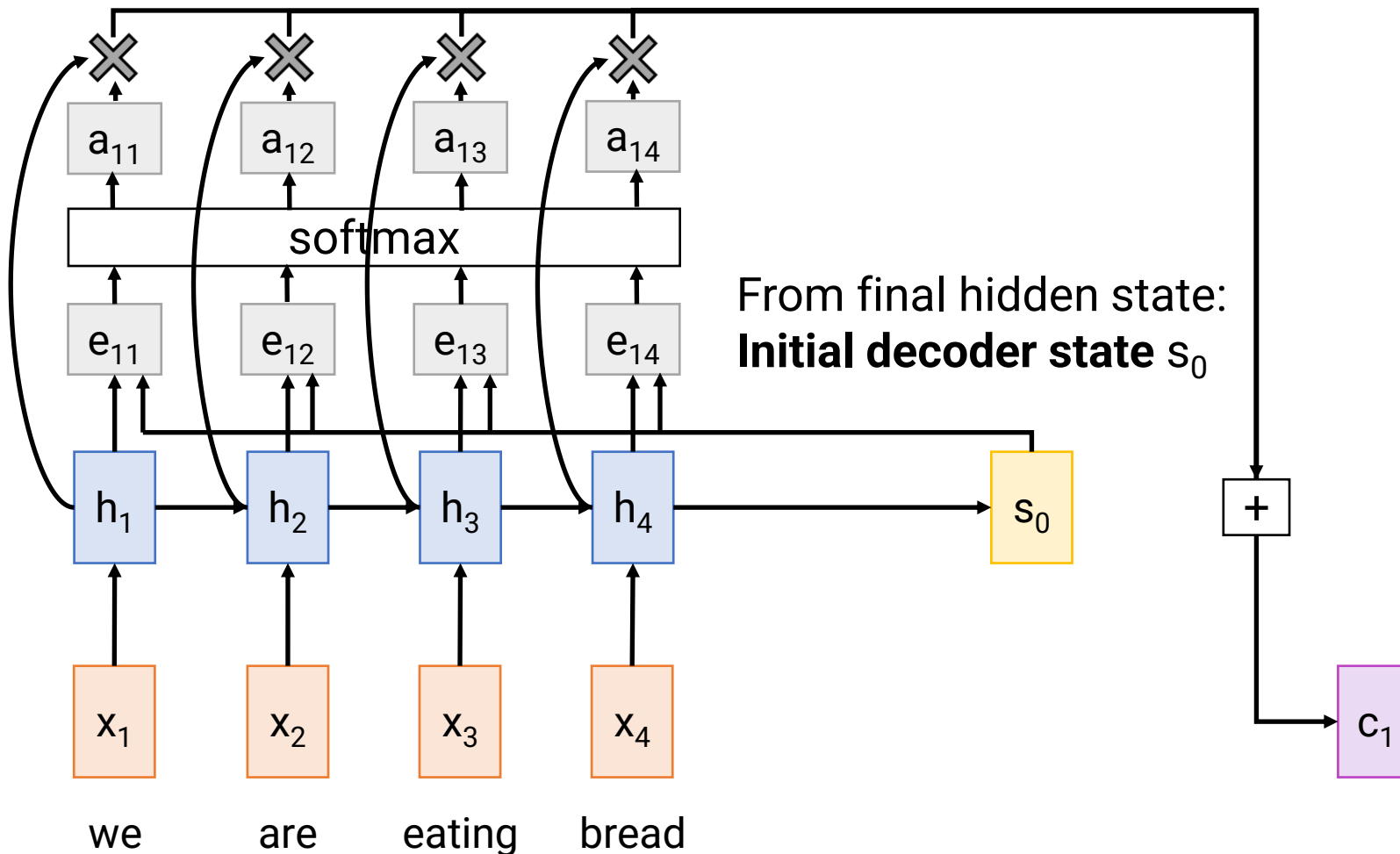
# Machine Translation with RNNs **and Attention**



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**

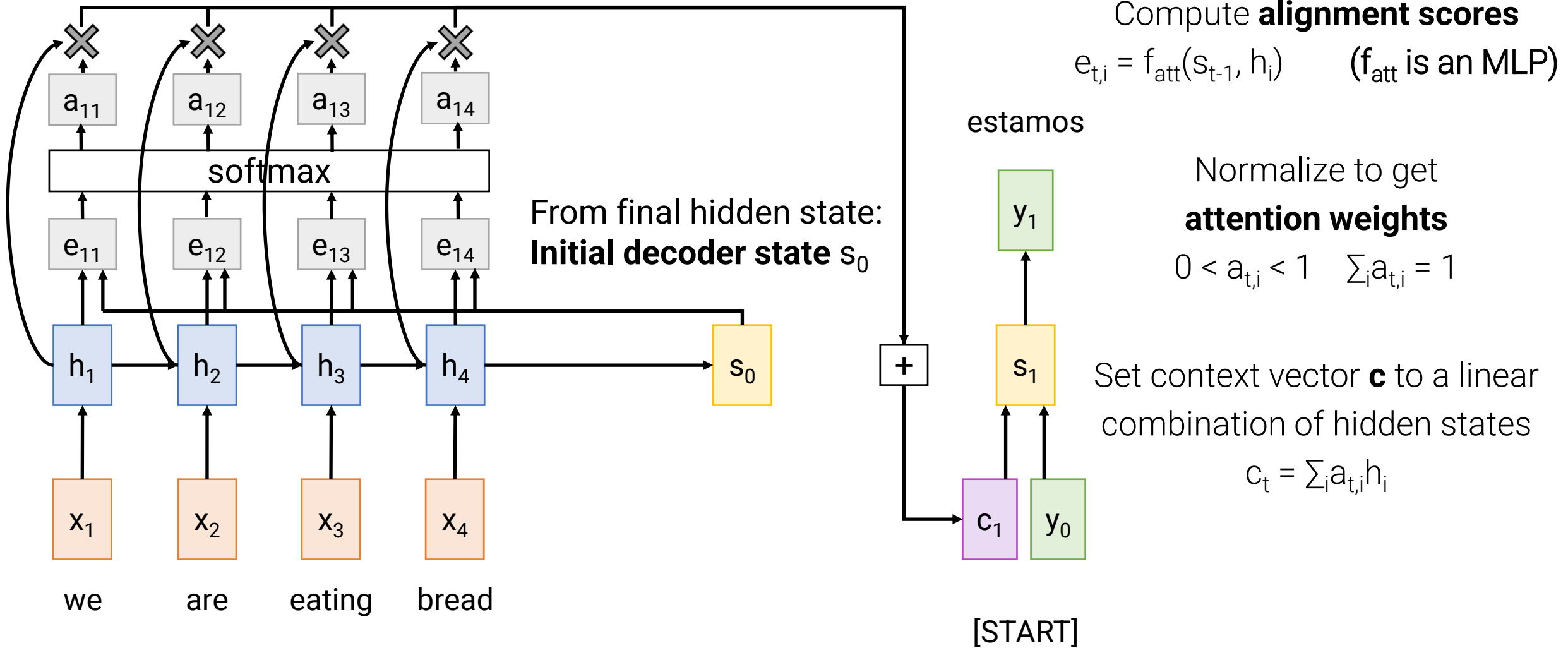


Compute **alignment scores**  
 $e_{t,i} = f_{\text{att}}(s_{t-1}, h_i)$  ( $f_{\text{att}}$  is an MLP)

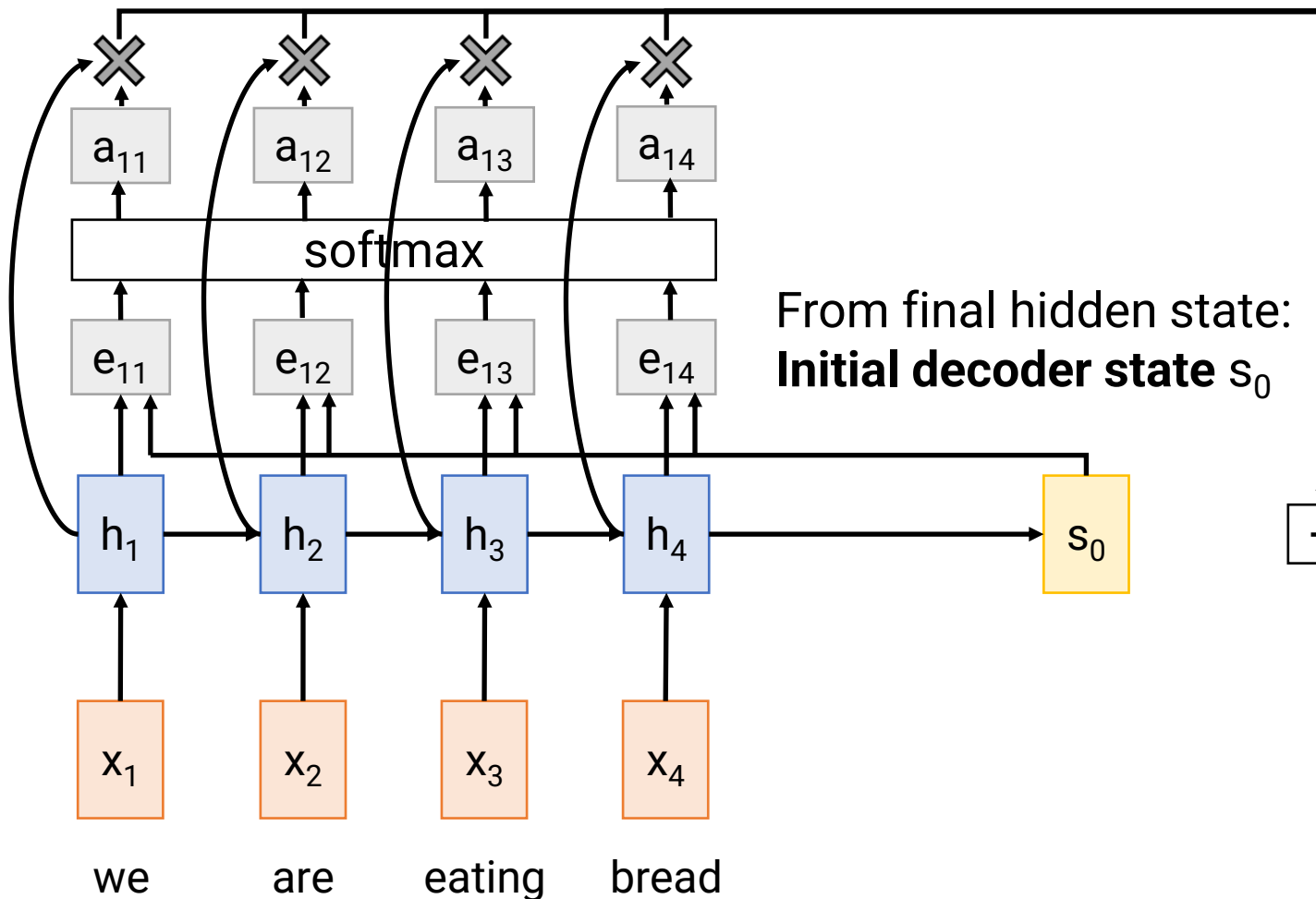
Normalize to get  
**attention weights**  
 $0 < a_{t,i} < 1 \quad \sum_i a_{t,i} = 1$

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**



Compute **alignment scores**

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

Normalize to get

**attention weights**

$$0 < a_{t,i} < 1 \quad \sum_i a_{t,i} = 1$$

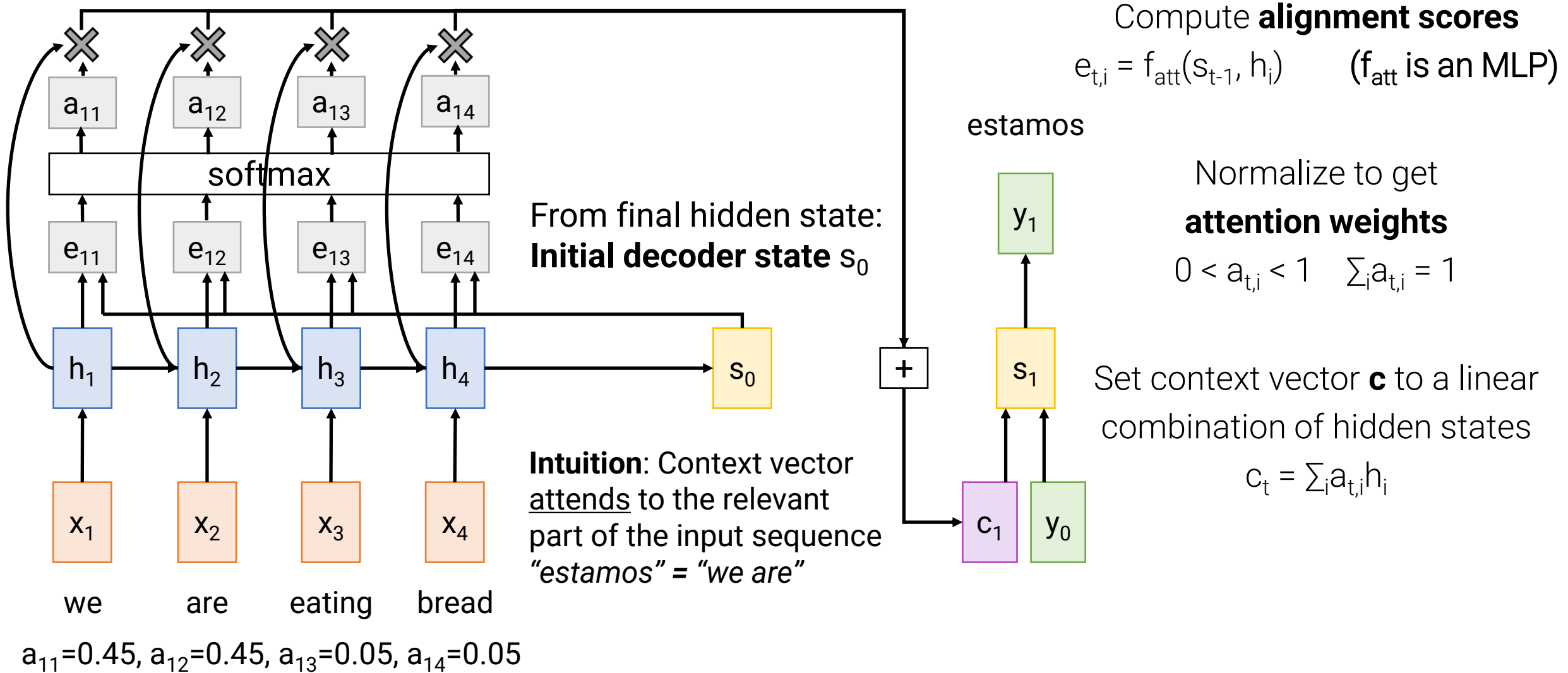
Set context vector  **$\mathbf{c}$**  to a linear combination of hidden states

$$c_t = \sum_i a_{t,i} h_i$$

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

**Can be seen as a input-dependent weighting (rather than MLP)**

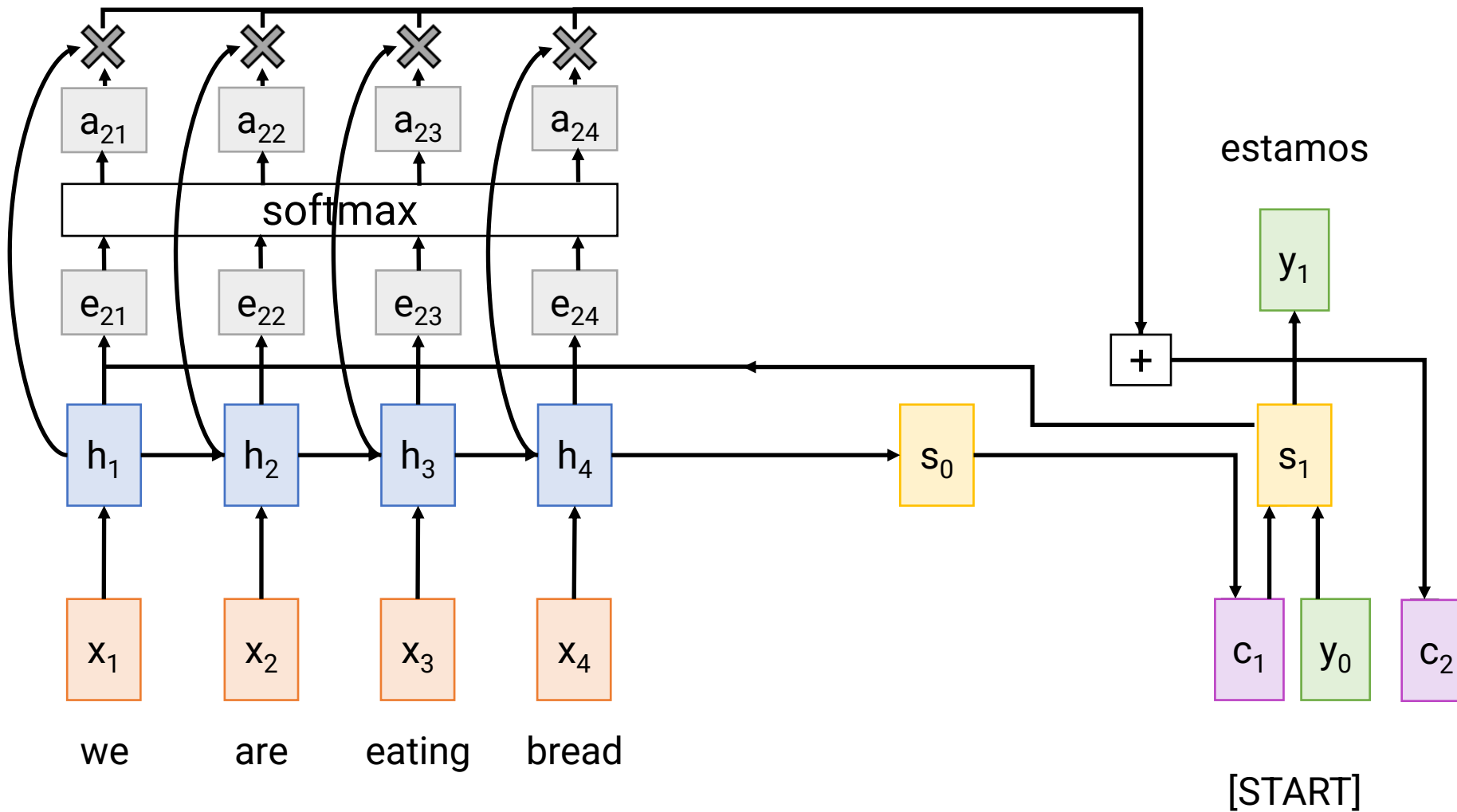
# Machine Translation with RNNs **and Attention**



**This is an inductive bias we think is reasonable for this task. Need to verify empirically though!**

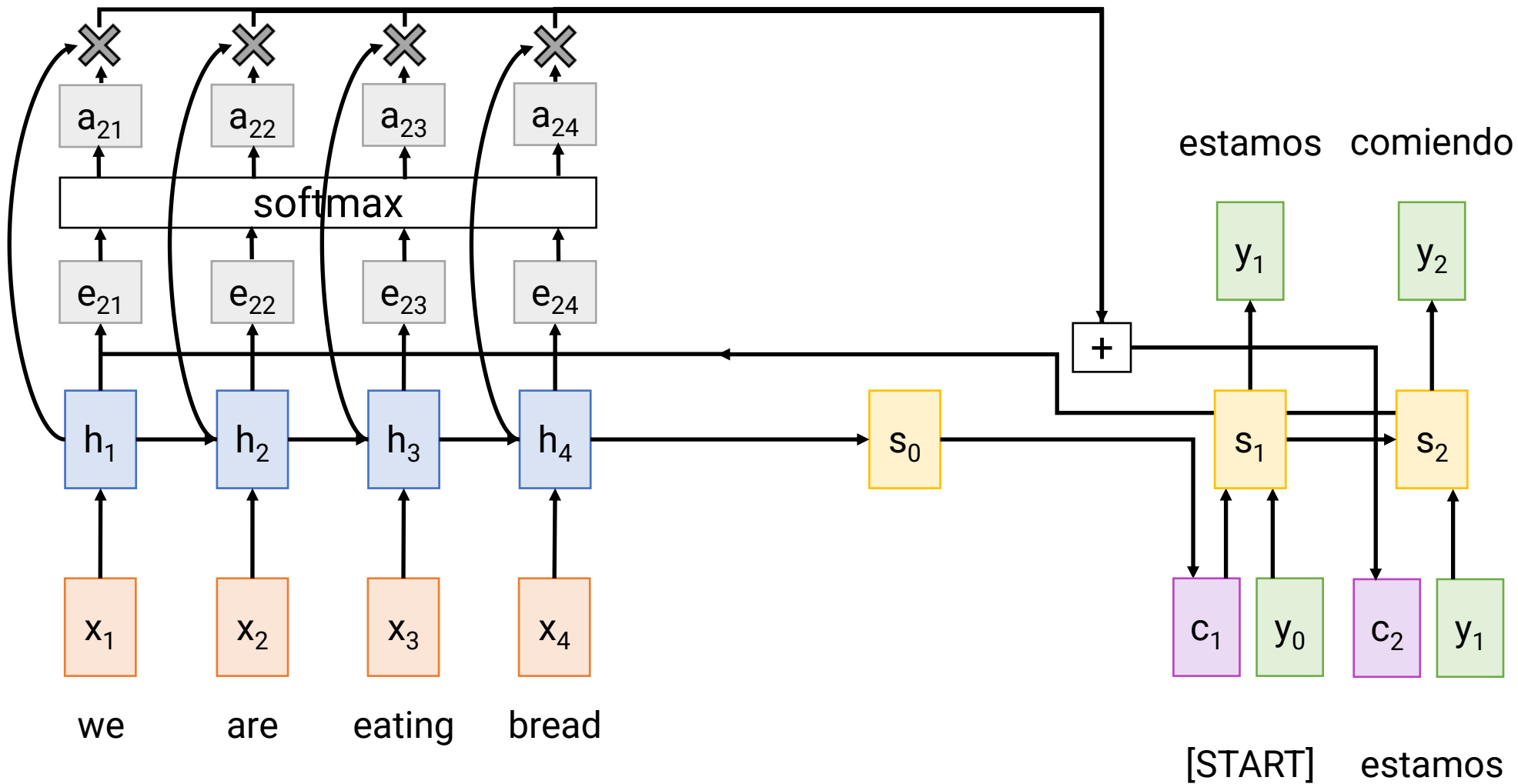


# Machine Translation with RNNs and Attention



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

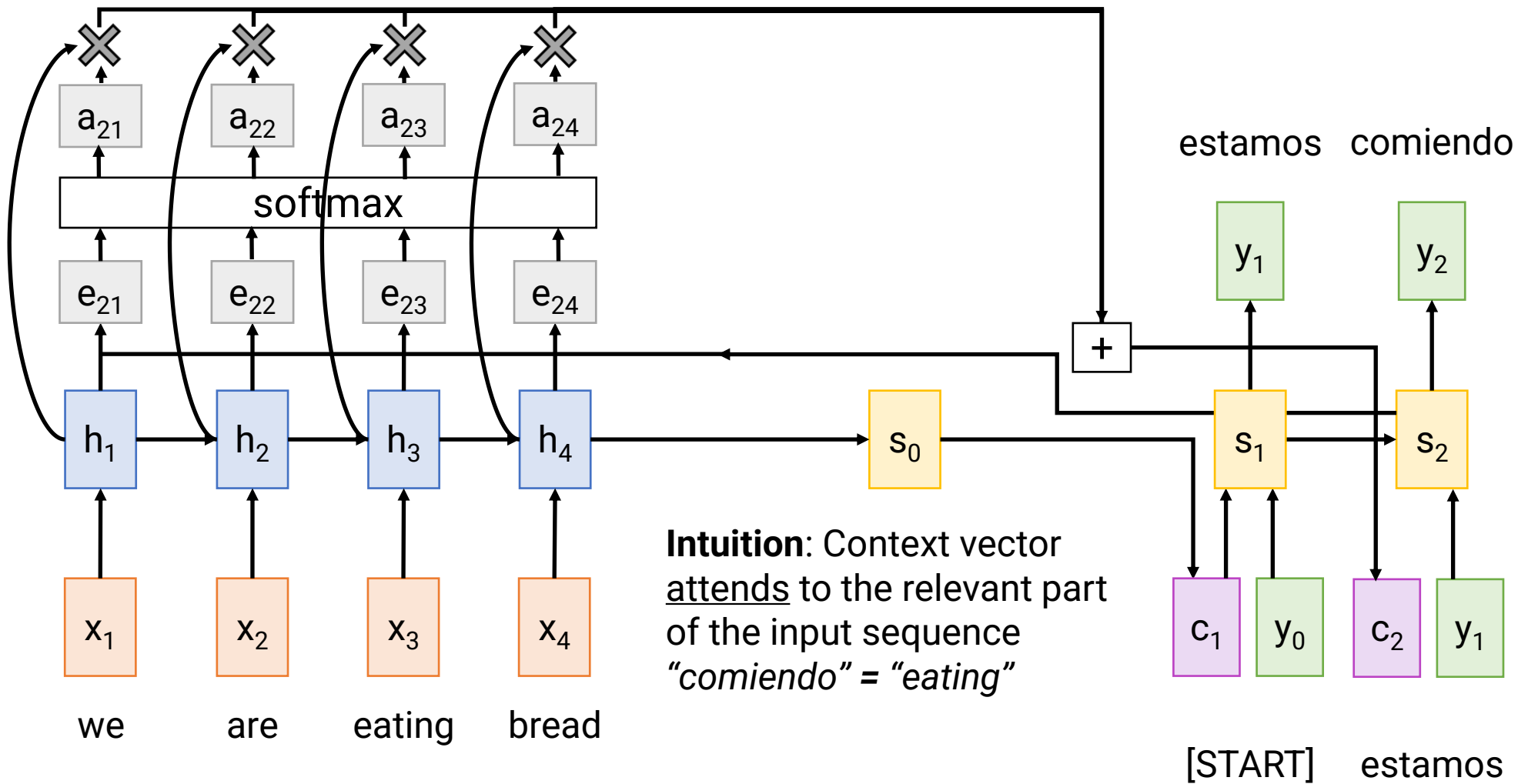
# Machine Translation with RNNs and Attention



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

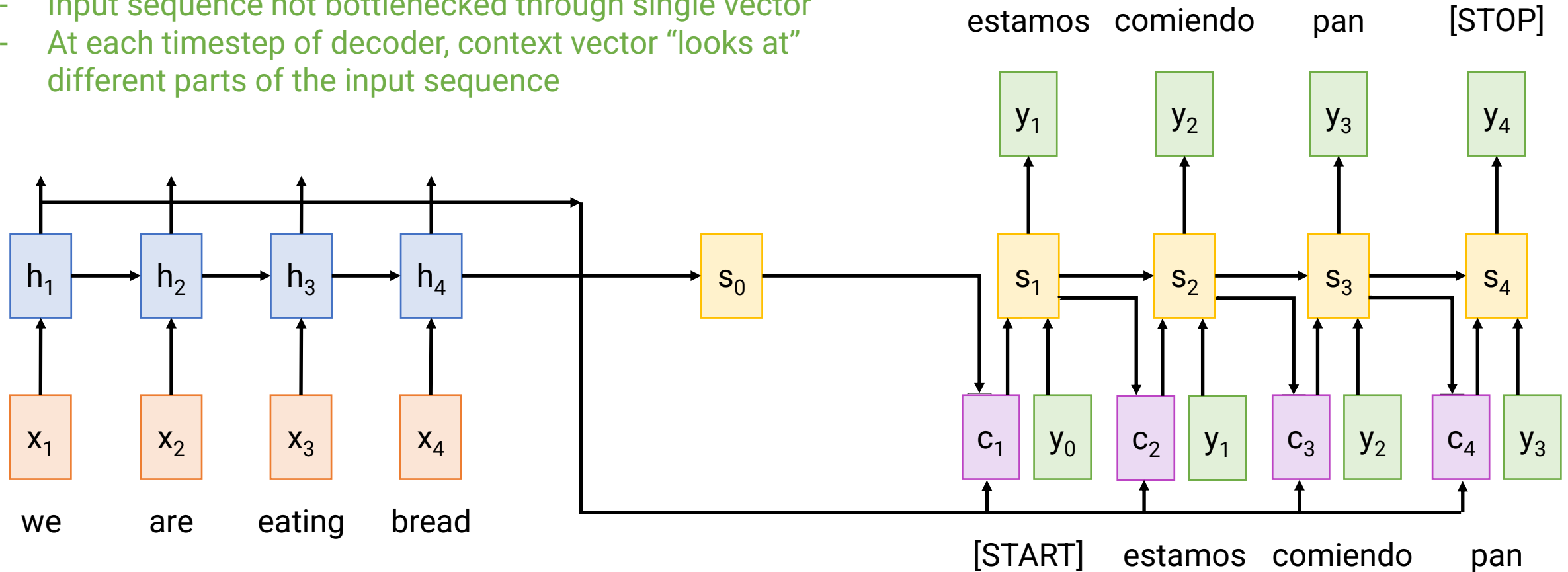


Repeat: Use  $s_1$  to compute new context vector  $c_2$

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



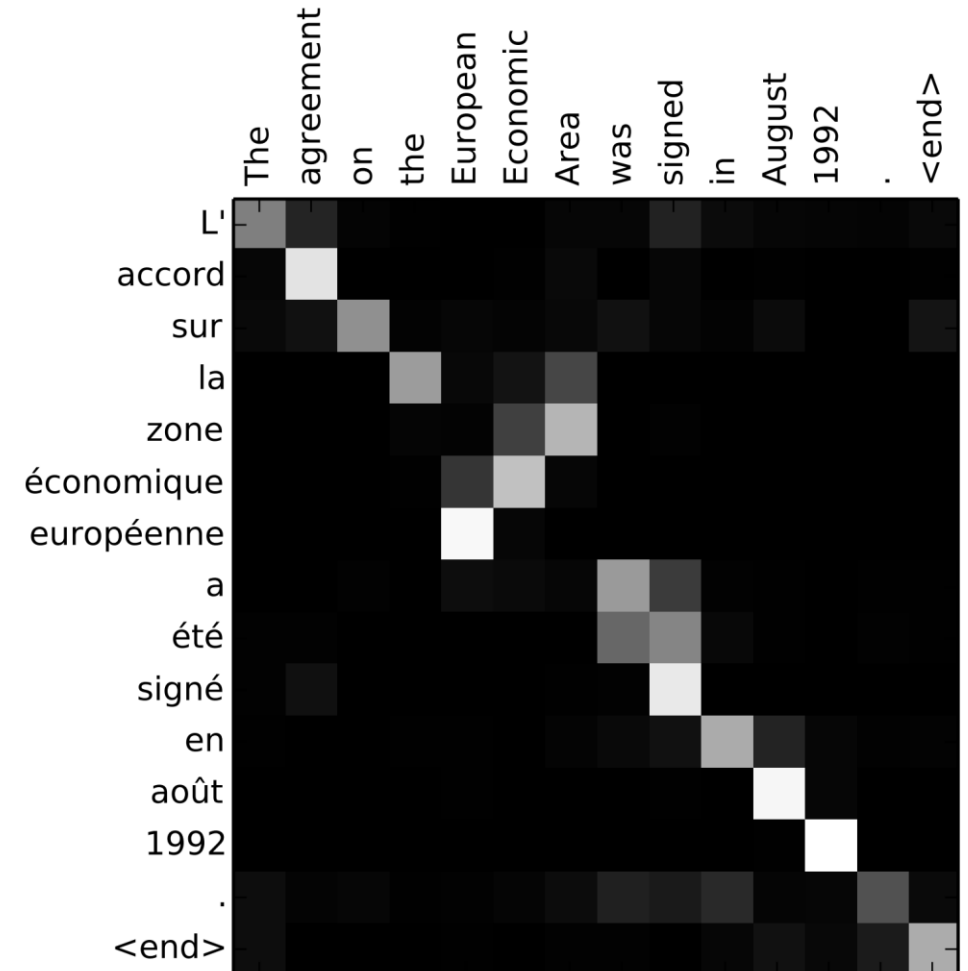
# Machine Translation with RNNs **and Attention**

**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

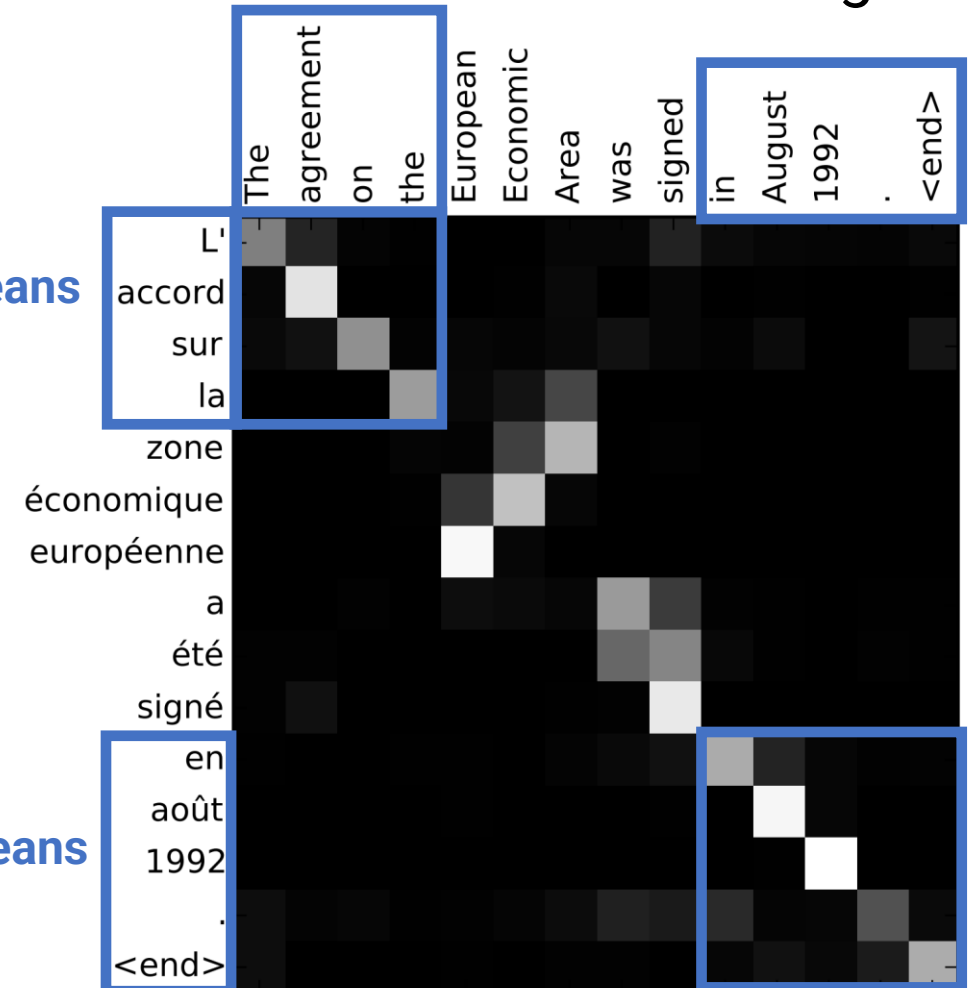
**Input:** “**The agreement on the** European Economic Area was signed **in August 1992.**”

**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.**”

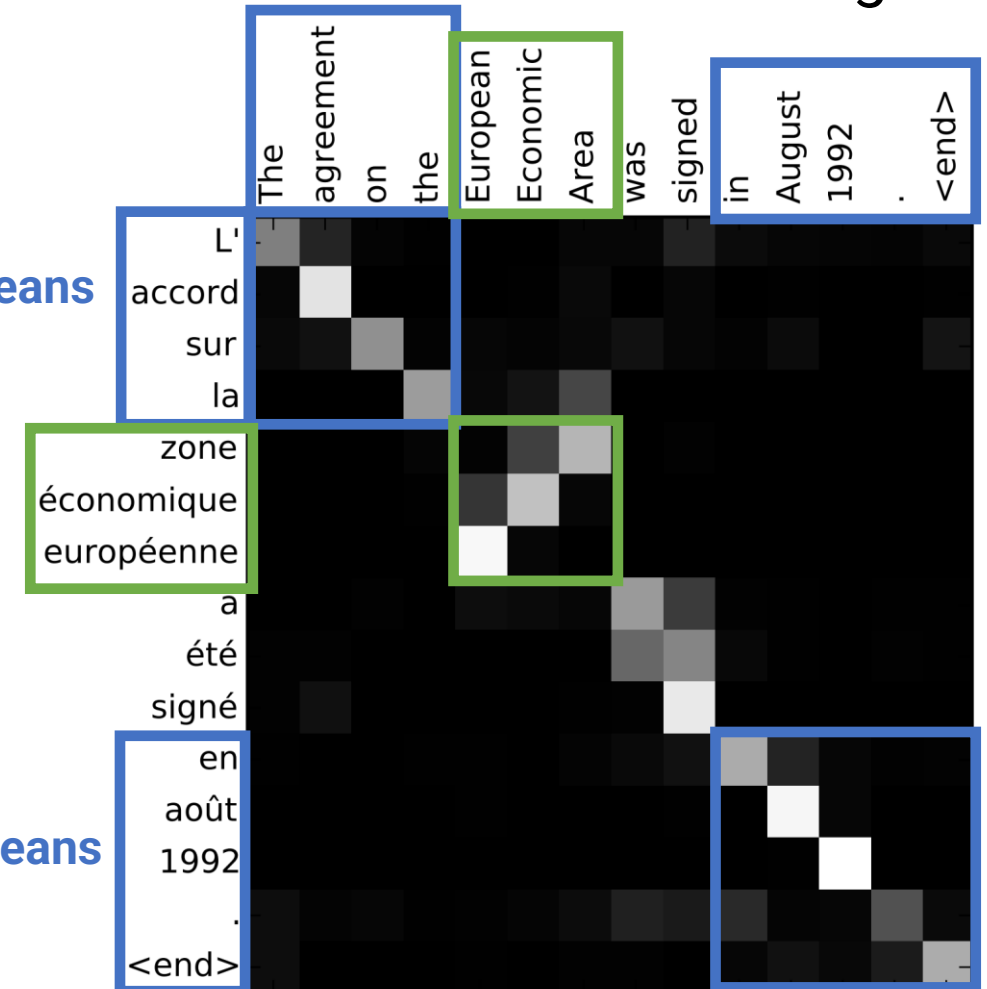
**Output:** “**L'accord sur la zone économique européenne** a été signé **en août 1992.**”

Diagonal attention means words correspond in order

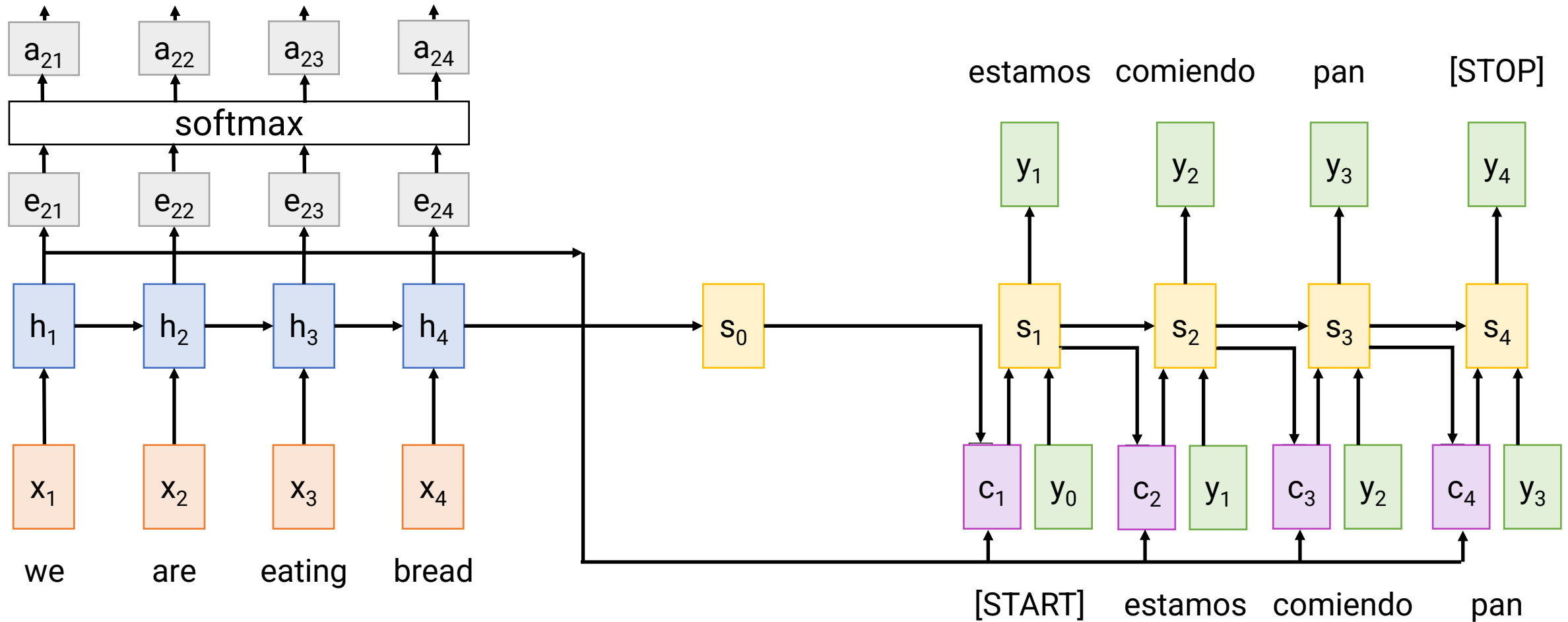
Attention figures out different word orders

Diagonal attention means words correspond in order

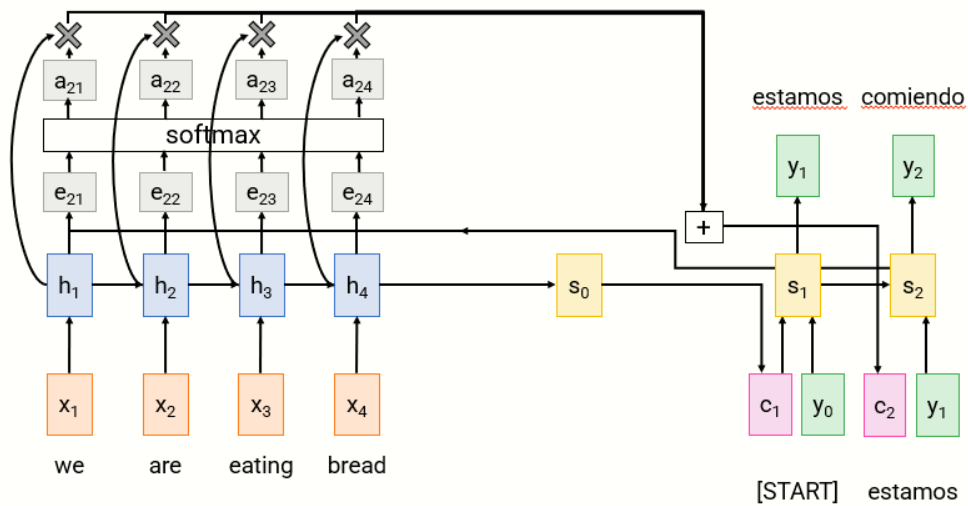
Visualize attention weights  $a_{t,i}$



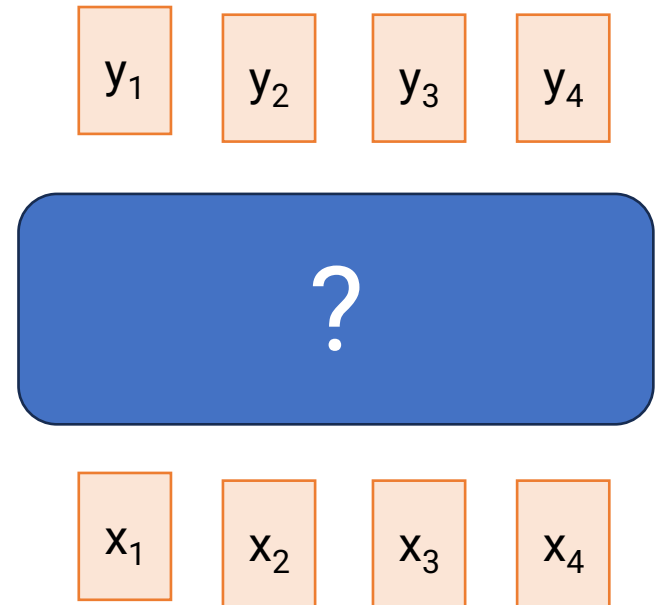
# Machine Translation with RNNs **and Attention**







Idea: Can we use **attention** as a fundamental building block for a generic sequence (input) to sequence (output) layer?



**Note:** We just want a generic sequence-in, sequence-out model that will represent each input *contextualized* with rest of inputs, and encode meaning of entire sequence

We will progressively develop a generic mechanism using idea of attention.  
 Don't try to map to RNN translation example!

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

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

**Output vector:**  $y = \sum_i a_i \mathbf{h}_i$  (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:  $f_{\text{att}}$

Make the module generic:

Input ( $\mathbf{X}$ ), Query ( $\mathbf{q}$ )

Output (Weighted sum of inputs)

## 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:  $N_X \times D_Q$ )

**Similarity function:** scaled dot product

## Computation:

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

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

**Output vector:**  $y = \sum_i a_i \mathbf{X}_i$  (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:  $N_X \times D_Q$ )

Make the module generic:

Sequence Input (**X**), Sequence Query (**Q**)

Output: Sequence (Weighted sum/mixture of inputs)

## Computation:

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

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

Output vectors:  $Y = \mathbf{AX}$  (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:  $D_X \times D_V$ )

## Separate concerns:

- 1) *Matching* (similarity) -> Key,
- 2) *Output given weighting* -> Value

## 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:

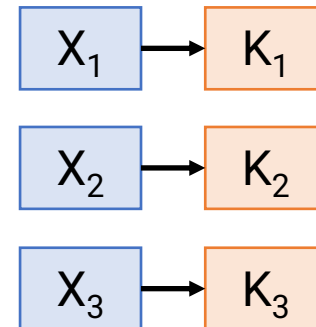
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

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

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

**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:

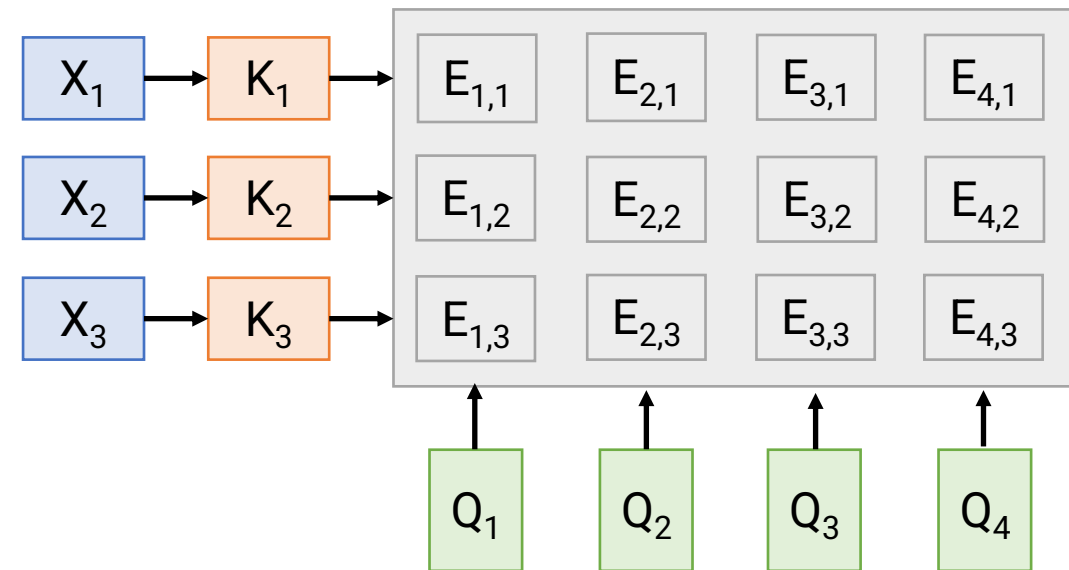
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

**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:  $N_Q \times D_V$ )  $Y_i = \sum_j A_{i,j} \mathbf{V}_j$



# 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:

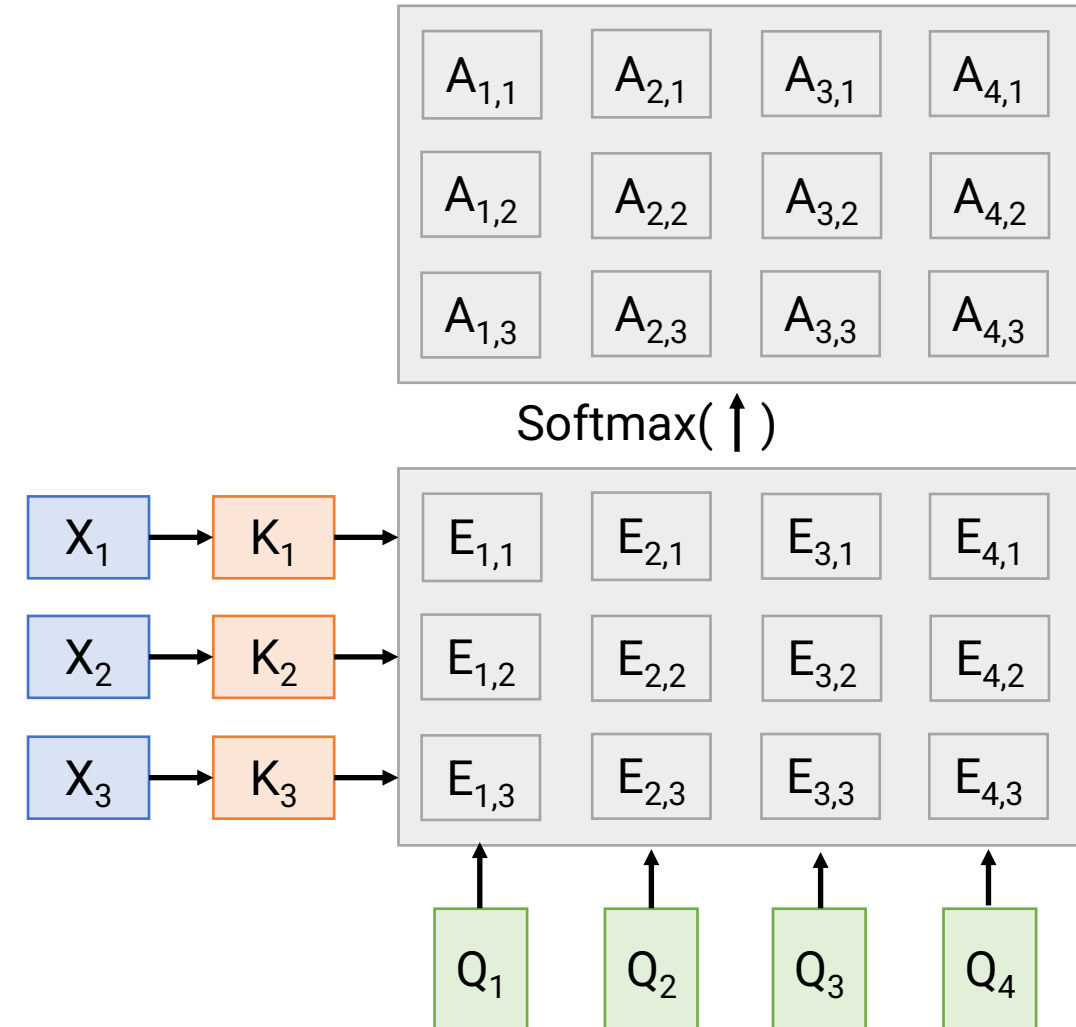
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

**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:

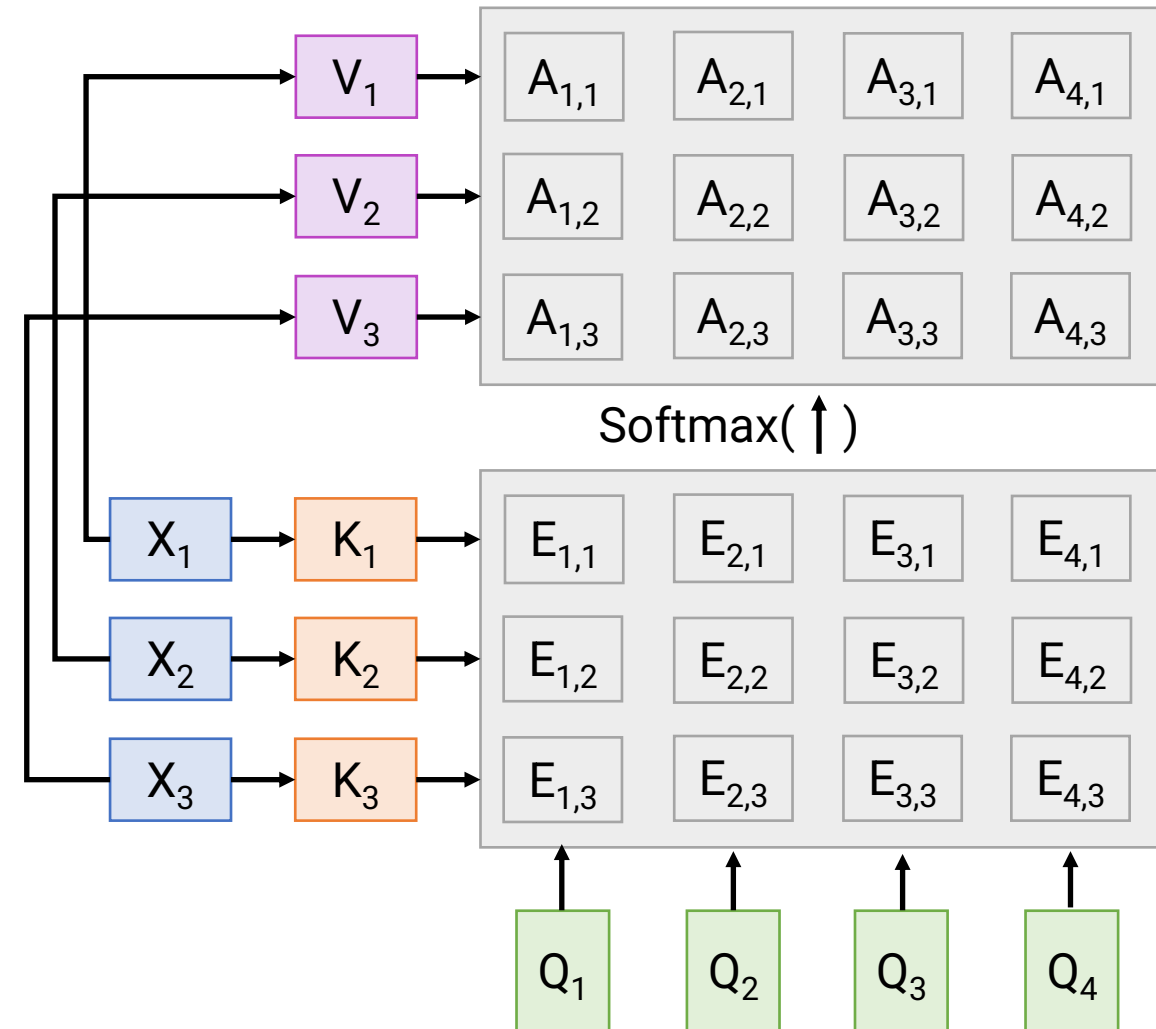
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

**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:  $N_Q \times D_V$ )  $Y_i = \sum_j A_{i,j} \mathbf{V}_j$



# 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:

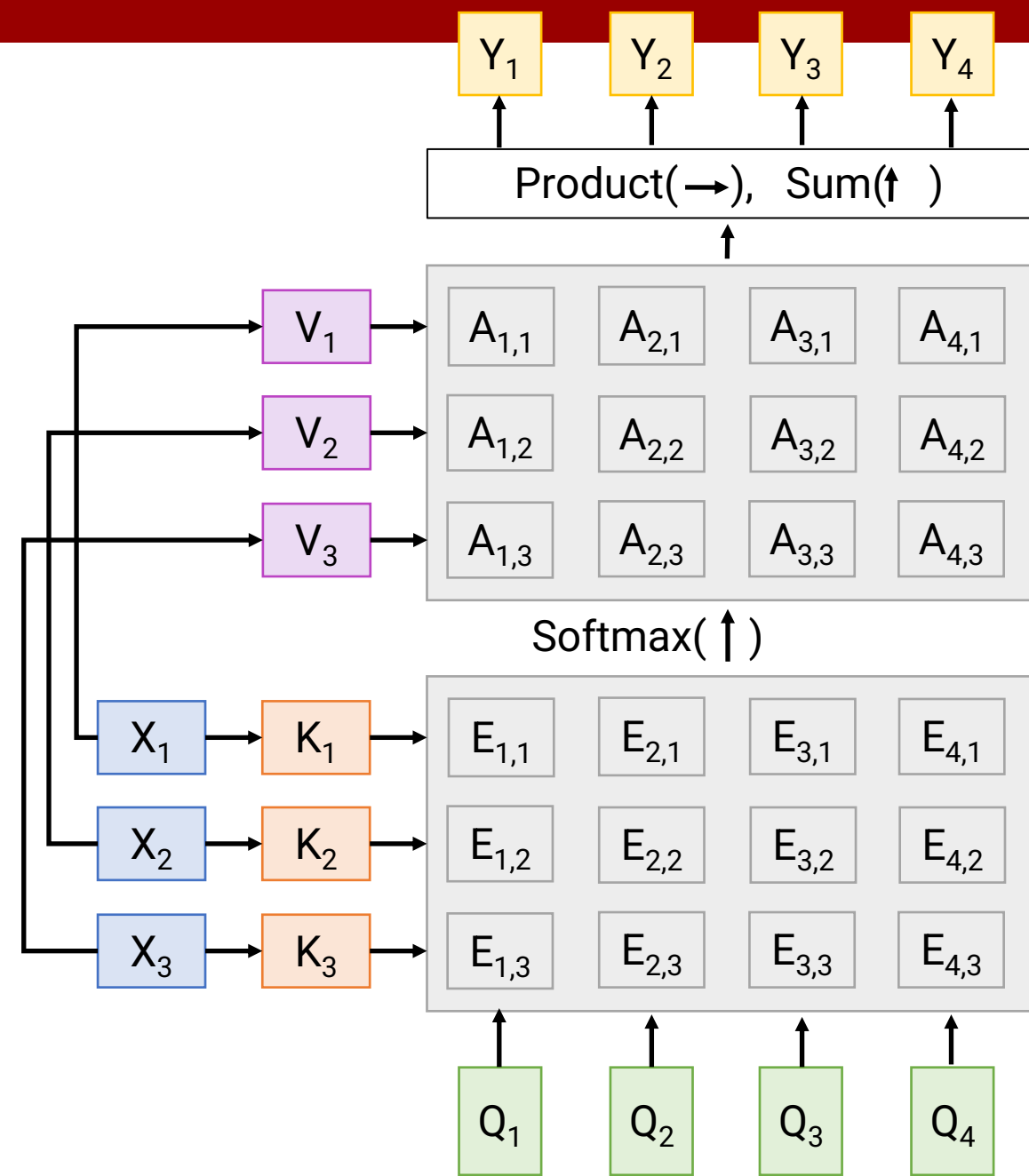
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

**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:  $D_x \times D_Q$ )

**Make the module generic:**

**Input: Sequence ( $\mathbf{X}$ )**

**Output: Sequence (Weighted sum/mixture of inputs)**

## 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:**  $\mathbf{Q} = \mathbf{XW}_Q$

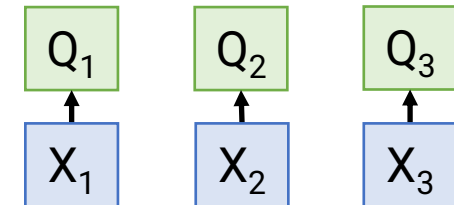
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_k$  (Shape:  $N_x \times D_Q$ )

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

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

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

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

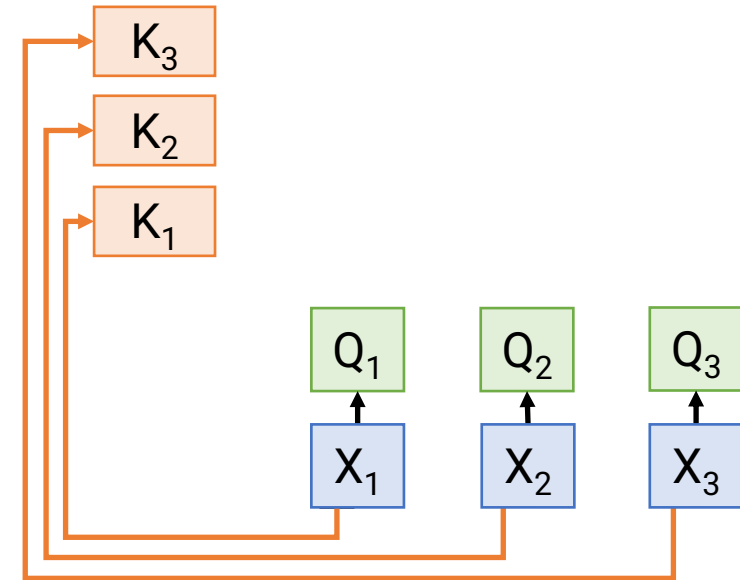
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_k$  (Shape:  $N_x \times D_Q$ )

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

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

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

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

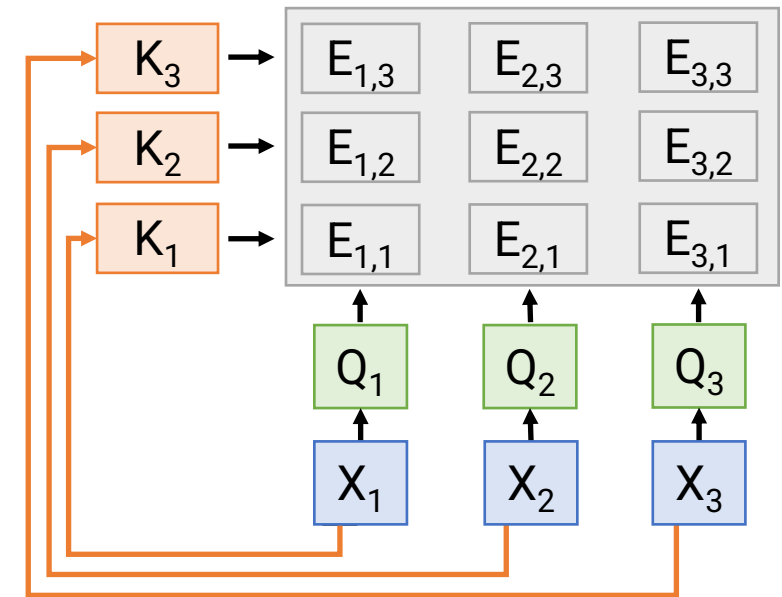
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_k$  (Shape:  $N_x \times D_Q$ )

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

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

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

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

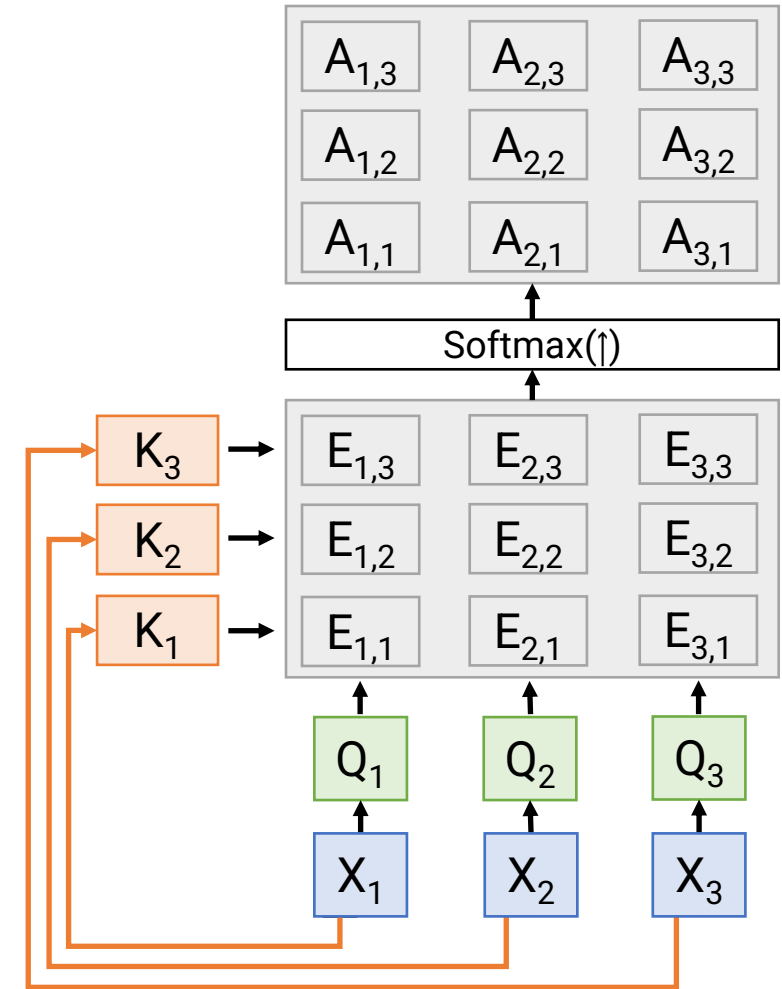
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_k$  (Shape:  $N_x \times D_Q$ )

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

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

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

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

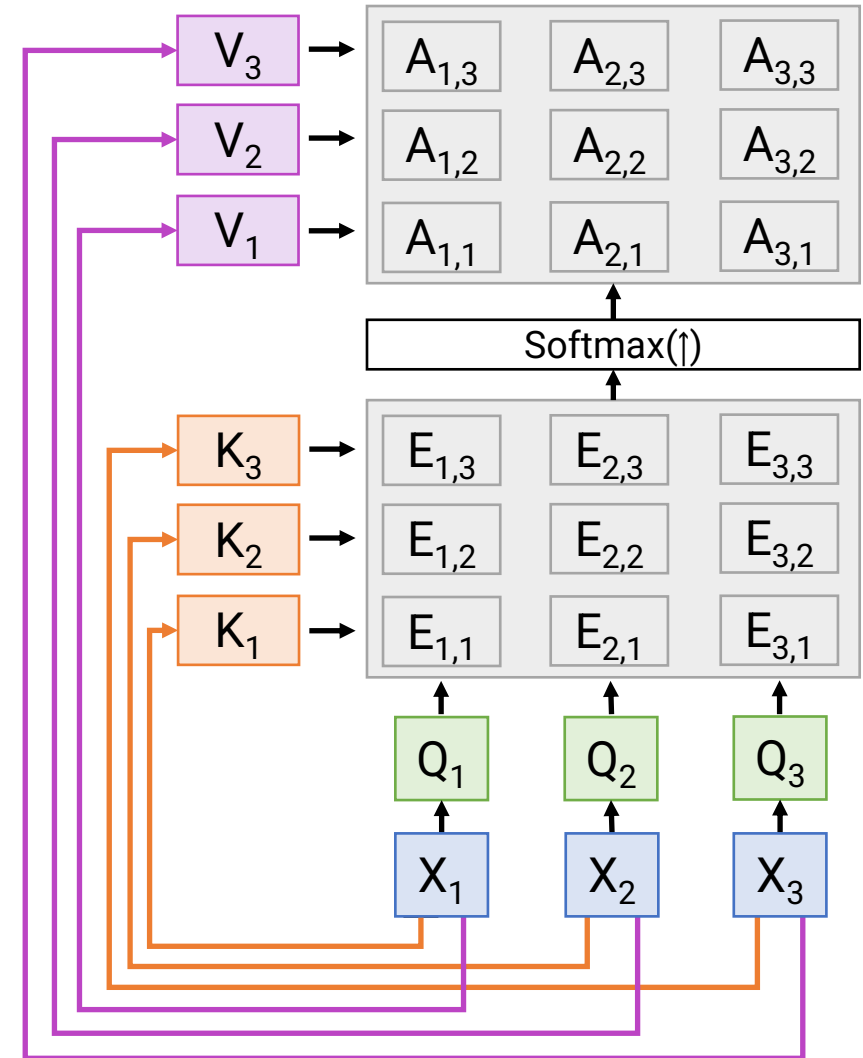
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

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

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

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

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

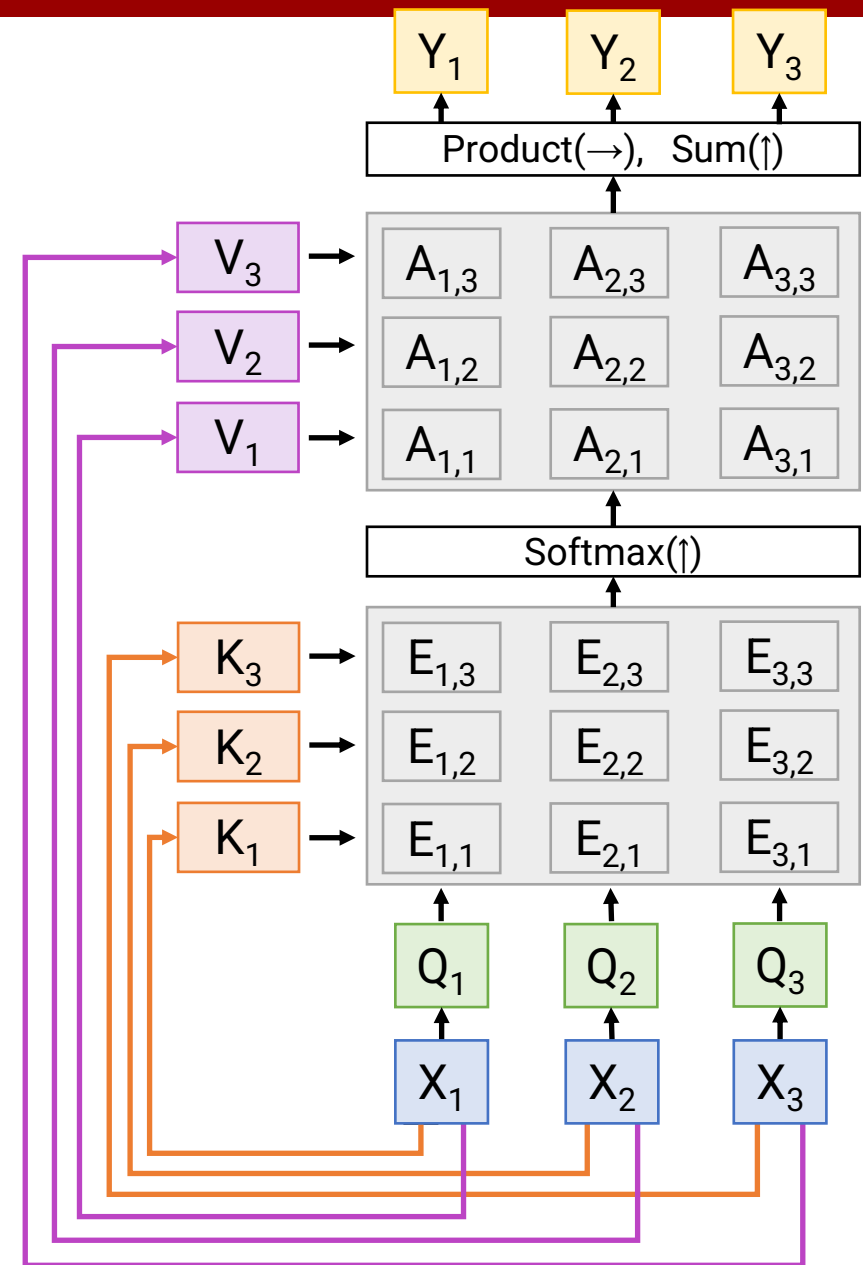
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_k$  (Shape:  $N_x \times D_Q$ )

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

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

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

**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:  $N_X \times D_Q$ )

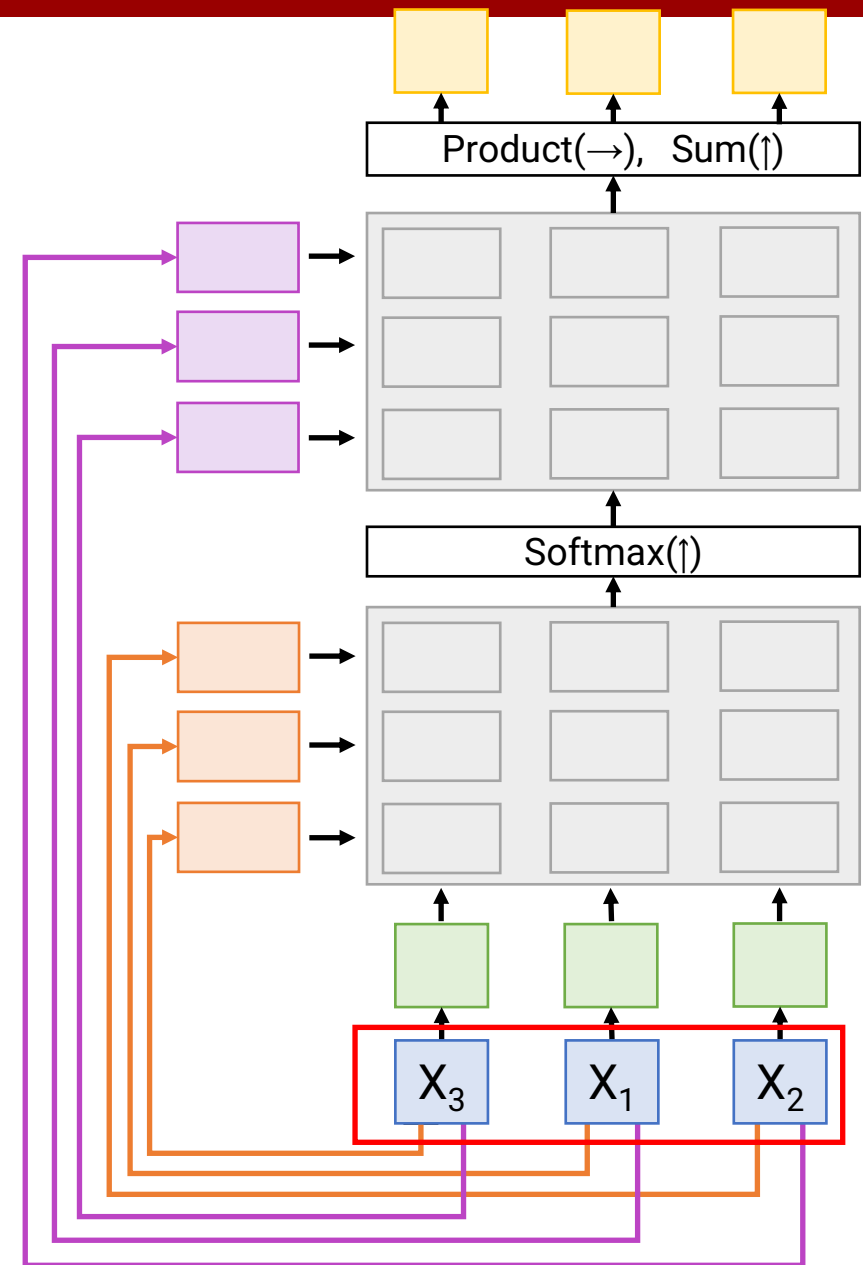
Value vectors:  $\mathbf{V} = \mathbf{XW}_V$  (Shape:  $N_X \times D_V$ )

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

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

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:  $N_X \times D_V$ )

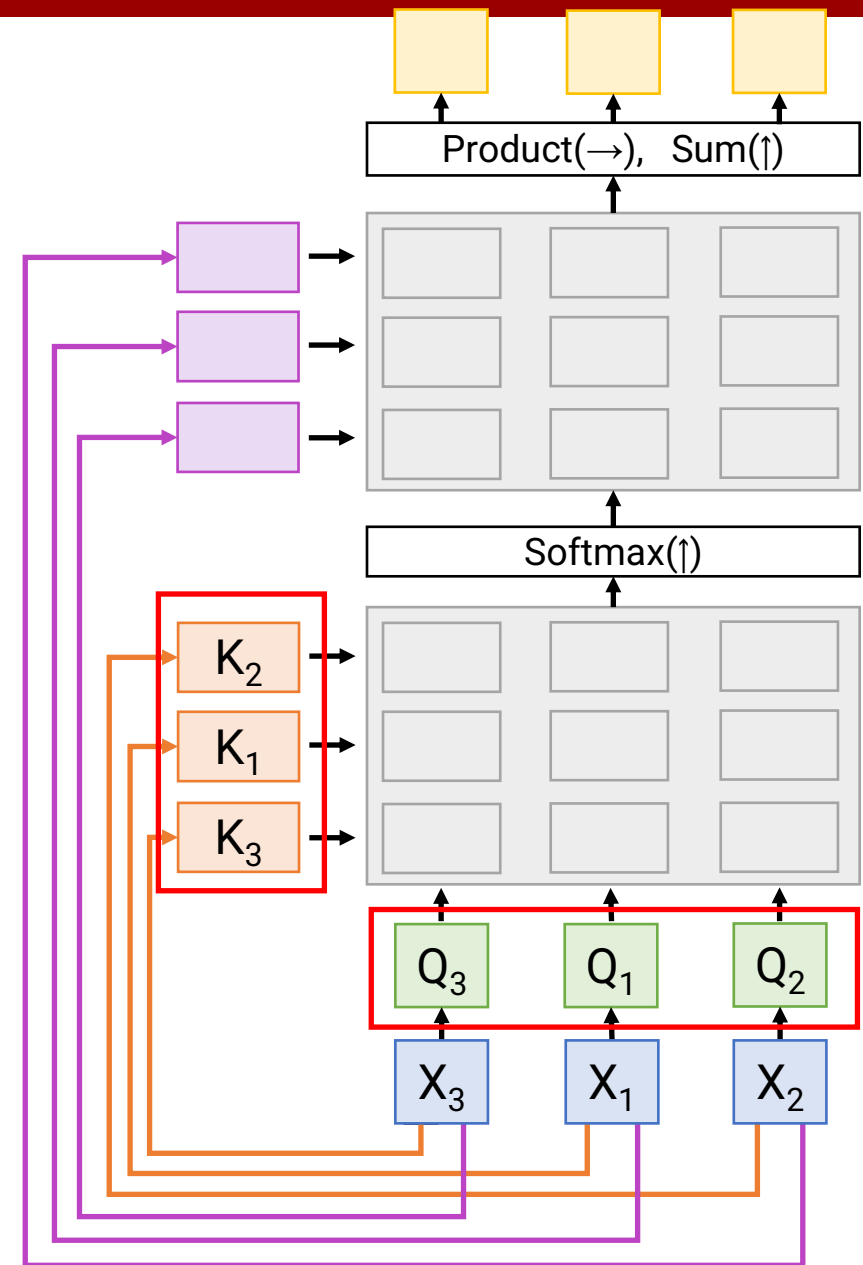
**Similarities:**  $\mathbf{E} = \mathbf{QK}^T$  (Shape:  $N_X \times N_X$ )  $E_{i,j} = \mathbf{Q}_i \cdot \mathbf{K}_j / \text{sqrt}(D_Q)$

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

**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:  $N_x \times D_V$ )

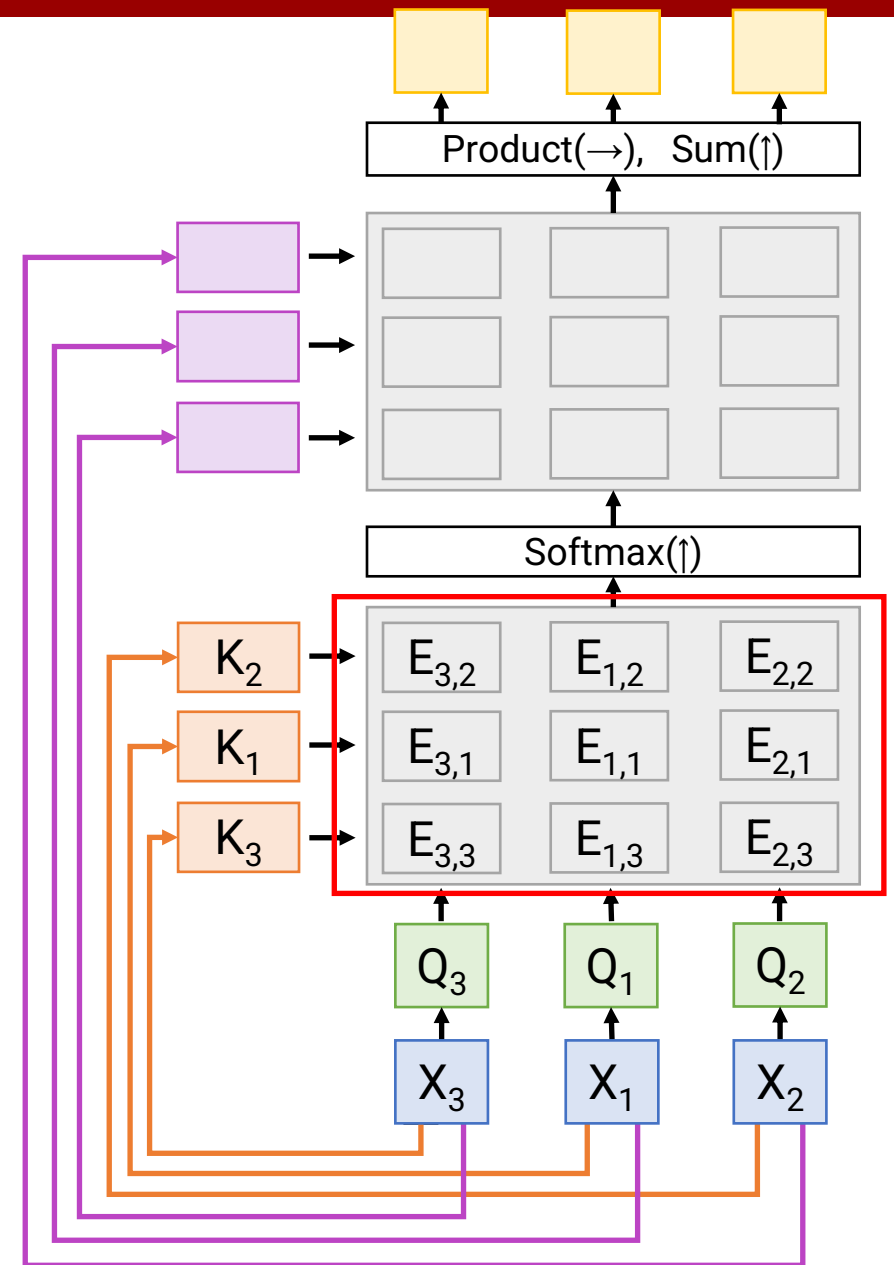
Similarities:  $\mathbf{E} = \mathbf{QK}^T$  (Shape:  $N_x \times N_x$ )  $E_{i,j} = \mathbf{Q}_i \cdot \mathbf{K}_j / \text{sqrt}(D_Q)$

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

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:  $N_x \times D_V$ )

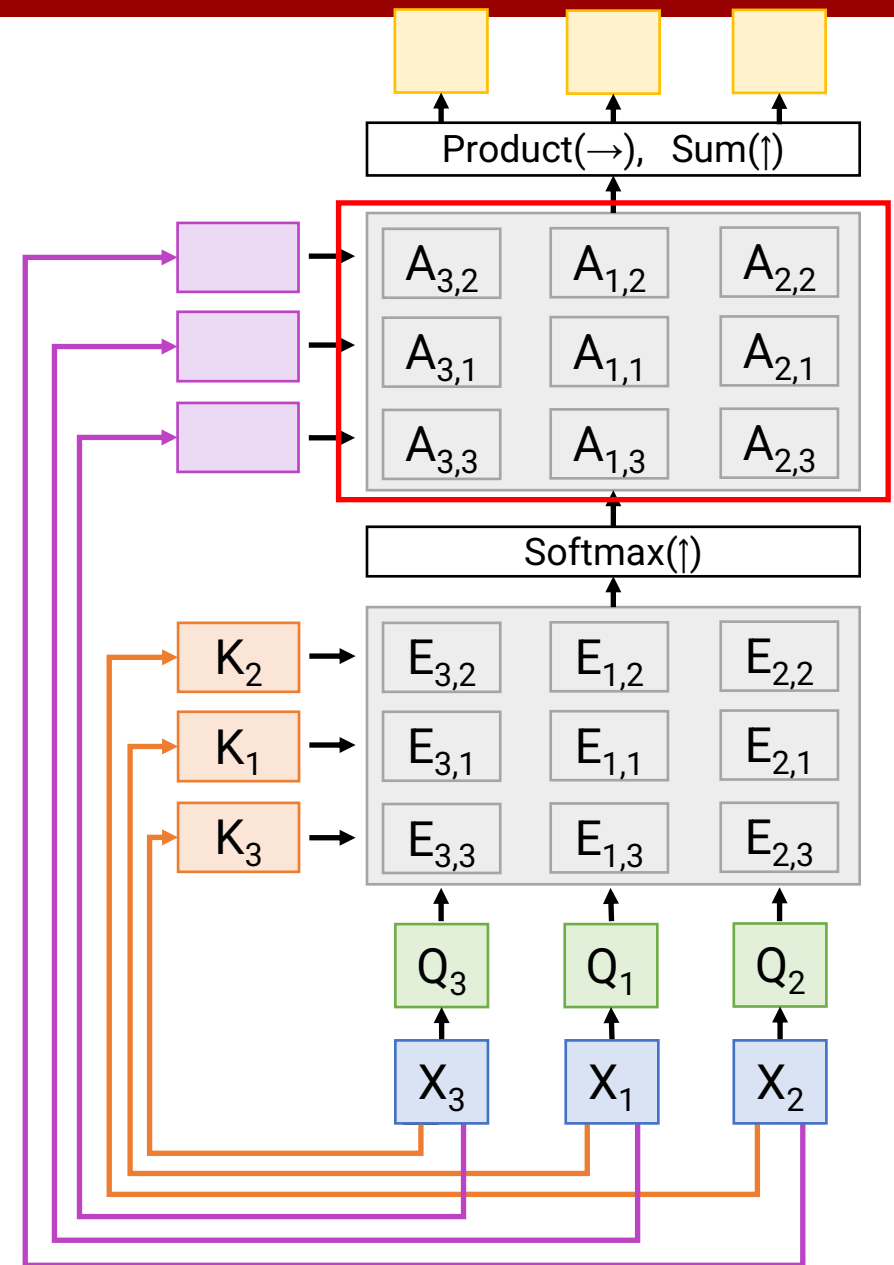
**Similarities:**  $\mathbf{E} = \mathbf{QK}^T$  (Shape:  $N_x \times N_x$ )  $E_{i,j} = \mathbf{Q}_i \cdot \mathbf{K}_j / \text{sqrt}(D_Q)$

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

**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:  $N_x \times D_V$ )

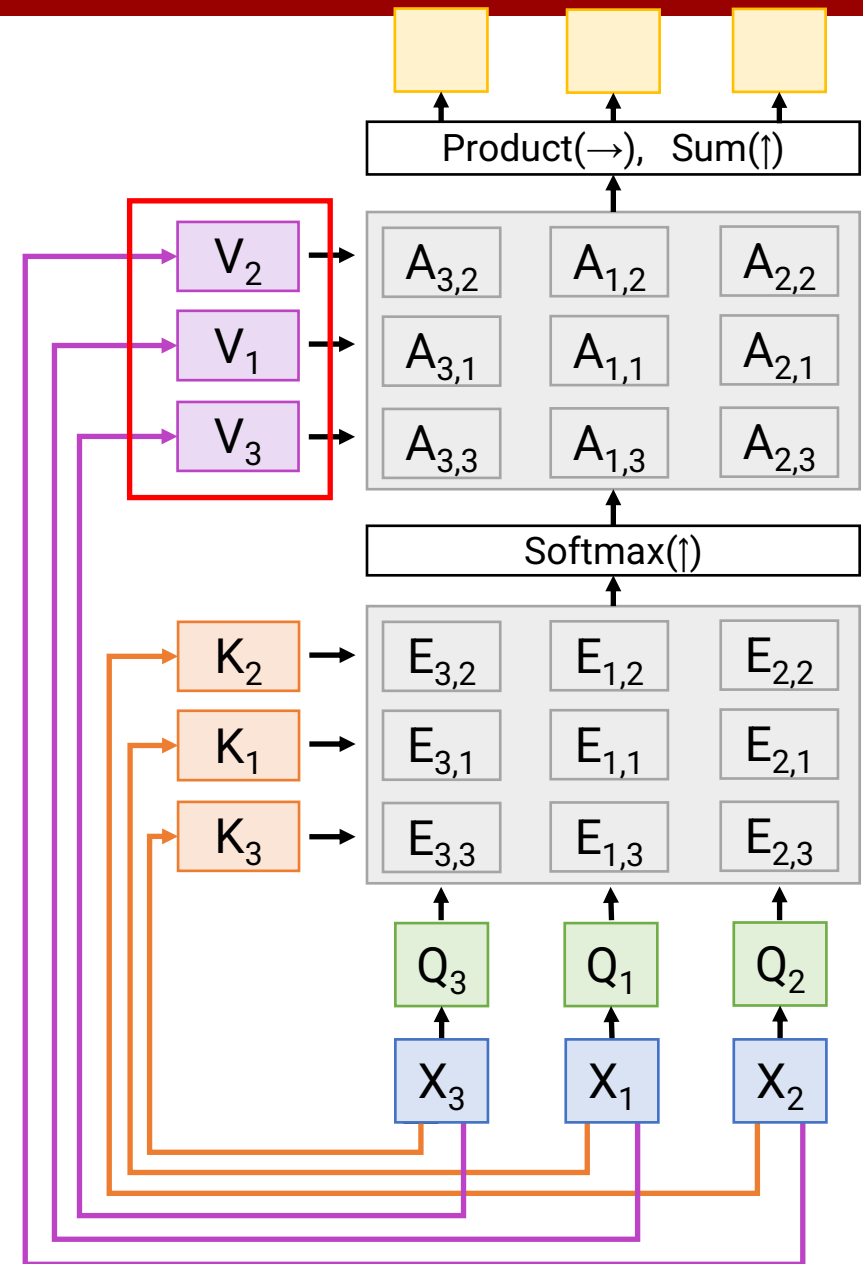
**Similarities:**  $E = QK^T$  (Shape:  $N_x \times N_x$ )  $E_{i,j} = Q_i \cdot K_j / \text{sqrt}(D_Q)$

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

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

Consider **permuting**  
the input vectors:

Values 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:  $N_x \times D_V$ )

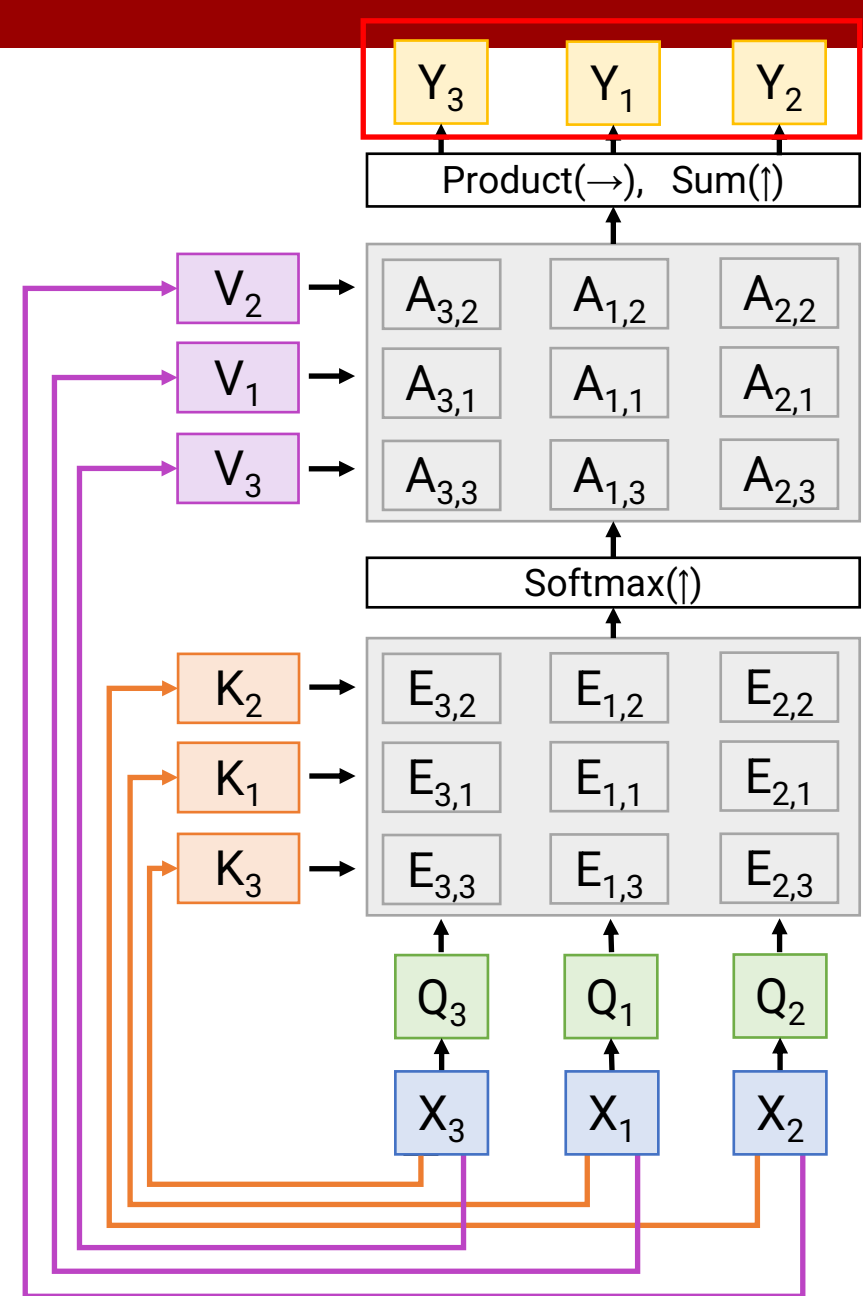
**Similarities:**  $E = QK^T$  (Shape:  $N_x \times N_x$ )  $E_{i,j} = Q_i \cdot K_j / \text{sqrt}(D_Q)$

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

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

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:  $N_x \times N_x$ )  $E_{i,j} = \mathbf{Q}_i \cdot \mathbf{K}_j / \text{sqrt}(D_Q)$

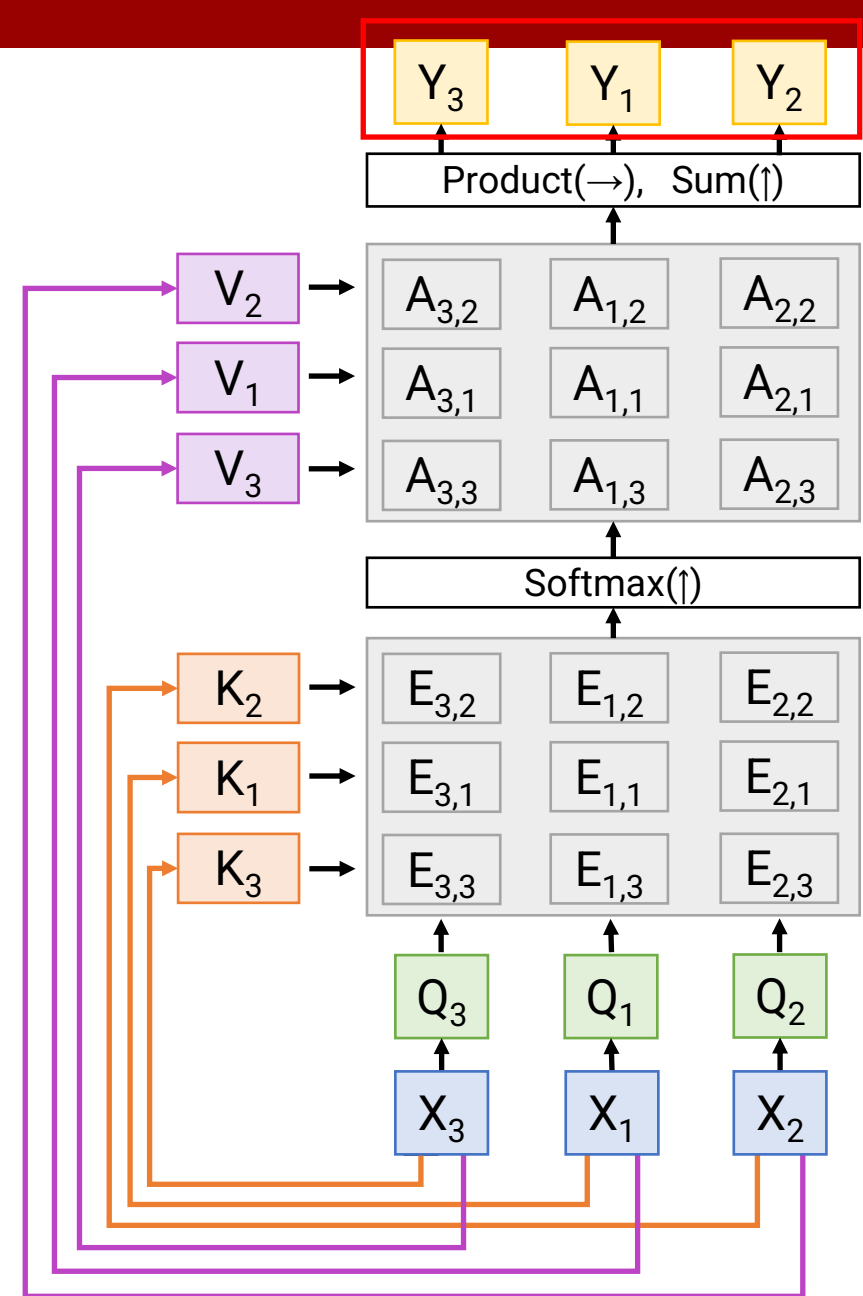
**Attention weights:**  $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$  (Shape:  $N_x \times N_x$ )

**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:**  $\mathbf{X}$  (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:  $N_x \times D_Q$ )

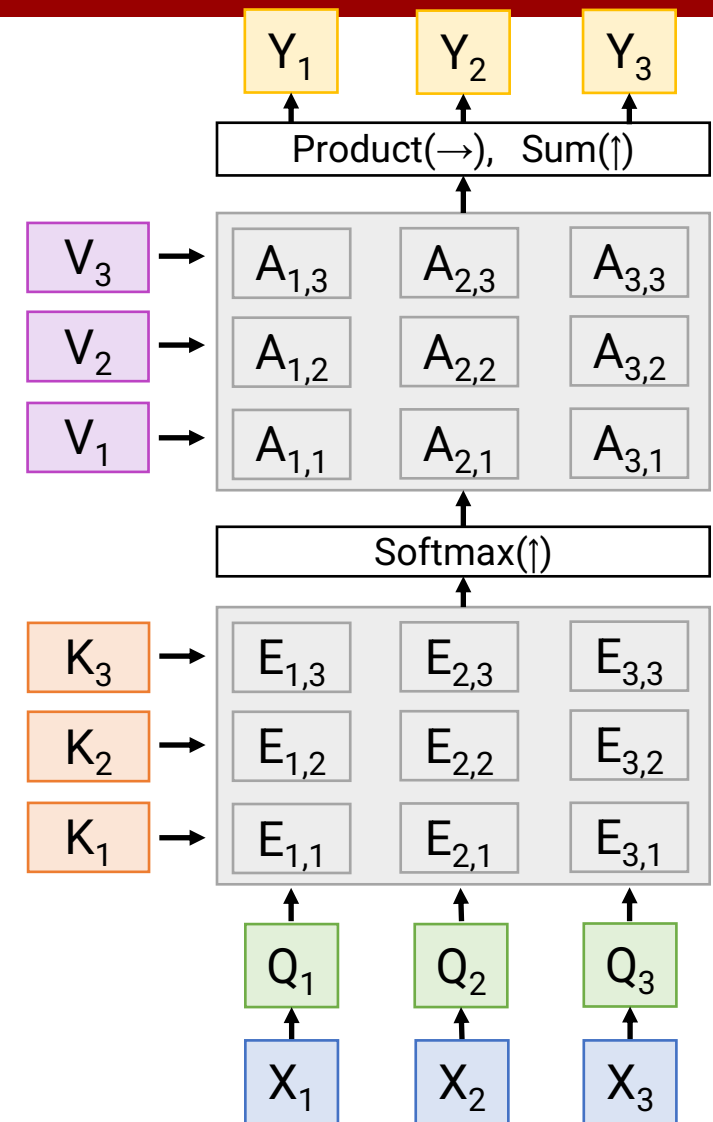
**Value vectors:**  $\mathbf{V} = \mathbf{XW}_v$  (Shape:  $N_x \times D_V$ )

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

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

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

Self attention 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:  $N_x \times N_x$ )  $E_{i,j} = Q_i \cdot K_j / \text{sqrt}(D_Q)$

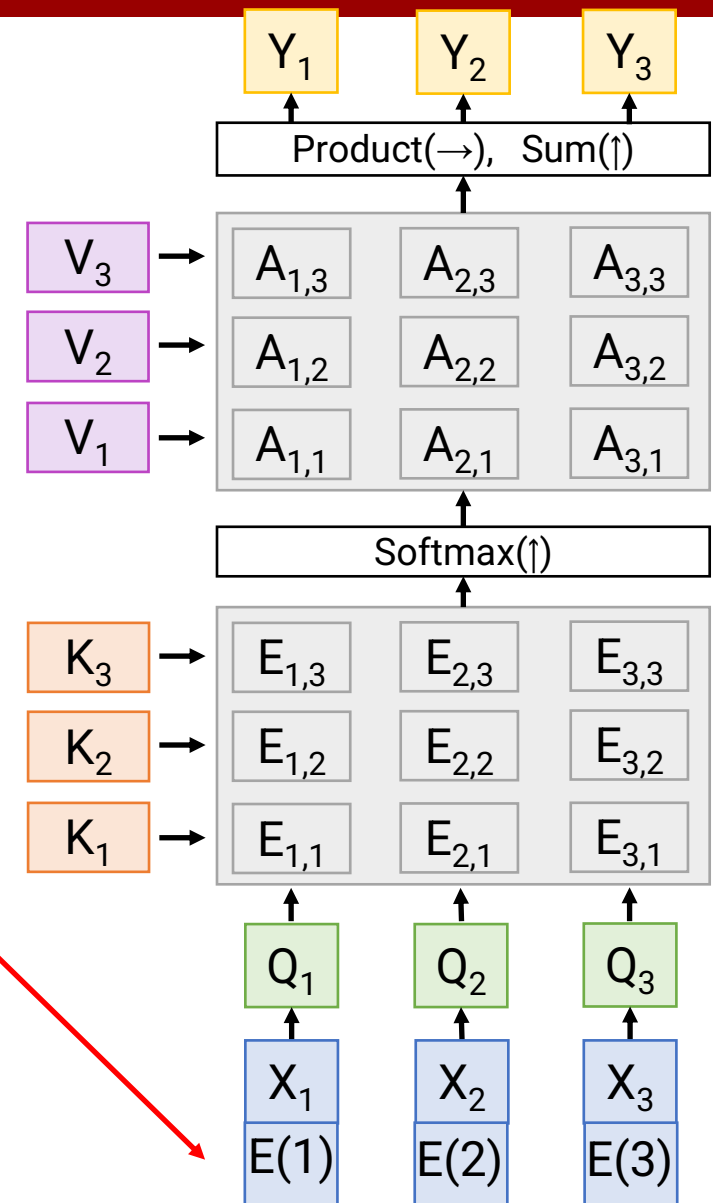
**Attention weights:**  $A = \text{softmax}(E, \text{dim}=1)$  (Shape:  $N_x \times N_x$ )

**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!

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

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



# Summary

- We have made a generic sequence-in to sequence-out layer
  - This is what we want for language processing!
  - Each output is a contextualized representation of the corresponding input word
  - Vector for stop word can be treated as representation of entire sentence (e.g. project its output to classifier and add loss)
- Unlike RNNs/LSTMs, it processes all inputs (e.g. entire sentence) **at once**
  - **Highly parallelizable**
  - **-> SCALE! -> Reduction of loss -> Magic**
- Next time: Entire transformer architecture that combines this new layer with other layers/concepts we know about (fully-connected, normalization, residual/skip connections)