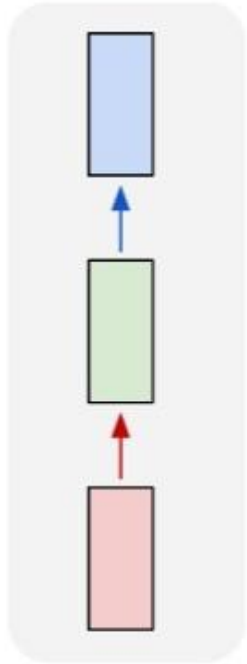


# Deep Learning

# So far: “Feedforward” Neural Networks

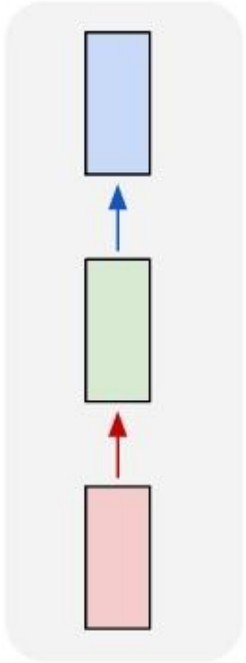
one to one



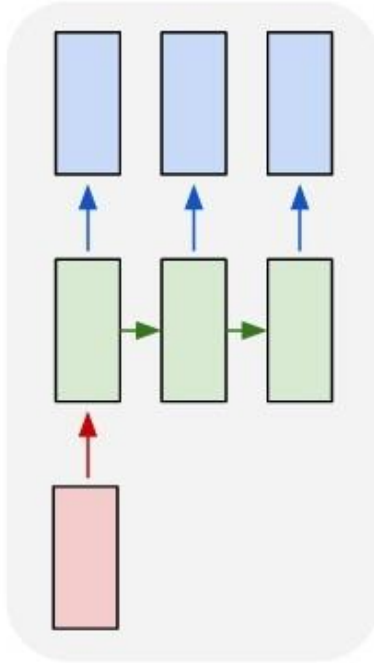
e.g. **Image classification**  
Image -> Label

# Recurrent Neural Networks: Process Sequences

one to one



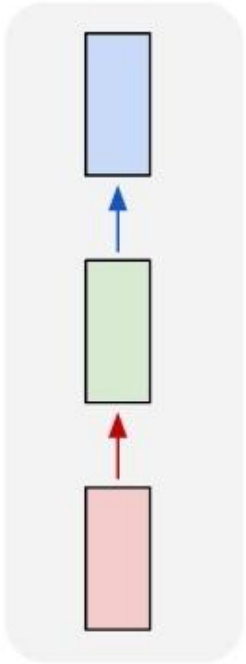
one to many



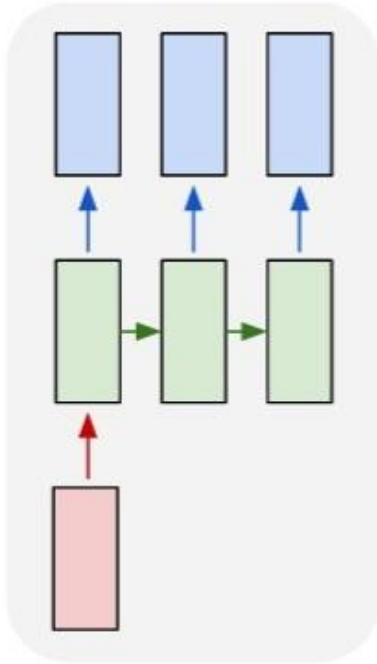
e.g. **Image Captioning:**  
Image -> sequence of words

# Recurrent Neural Networks: Process Sequences

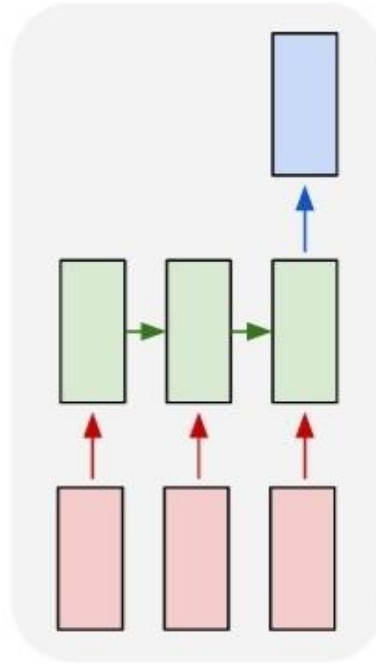
one to one



one to many



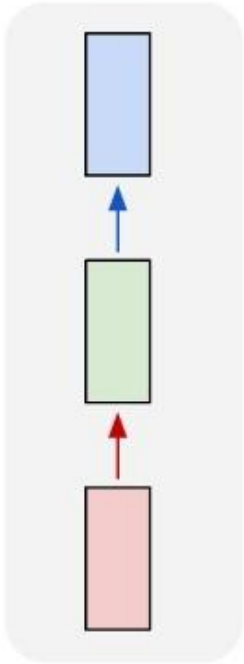
many to one



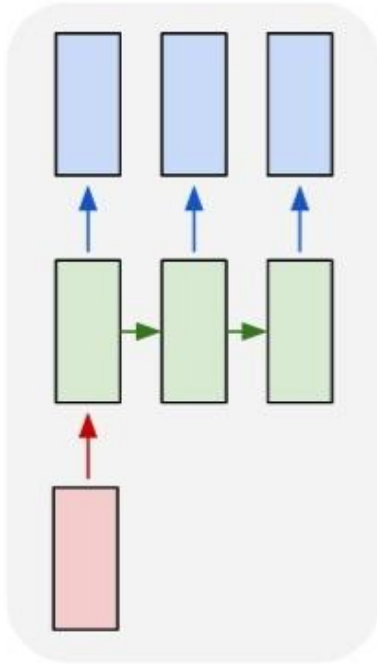
↖ e.g. **Video classification:**  
Sequence of images -> label

# Recurrent Neural Networks: Process Sequences

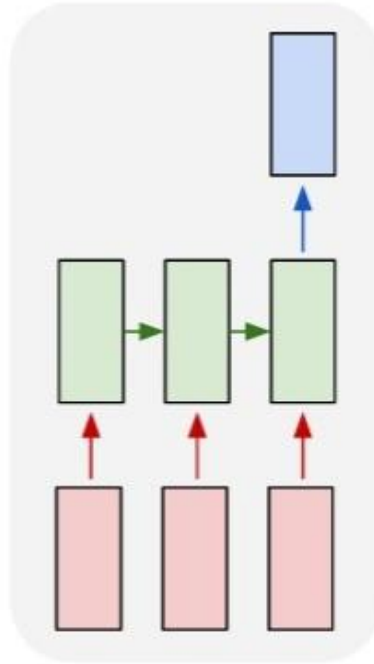
one to one



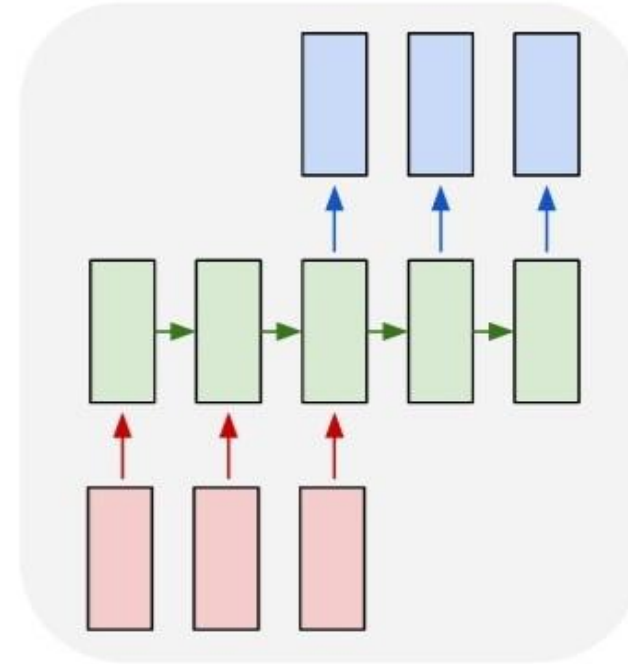
one to many



many to one



many to many



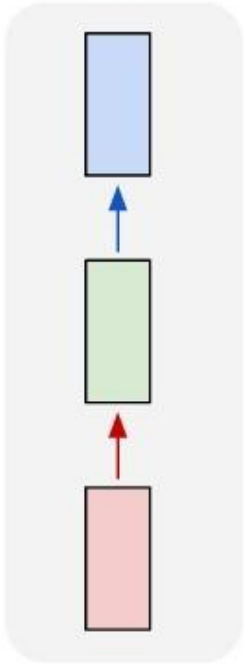
e.g. **Machine Translation:**

Sequence of words -> Sequence of words

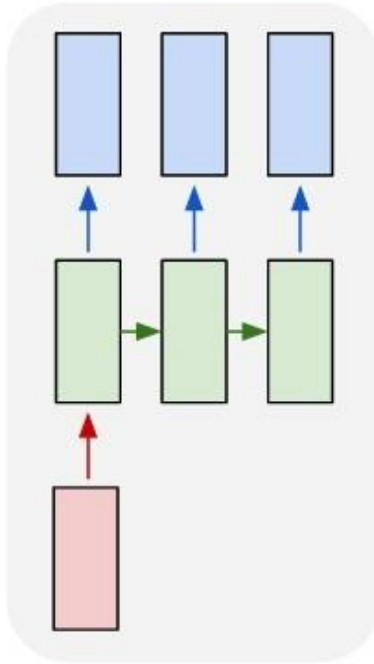


# Recurrent Neural Networks: Process Sequences

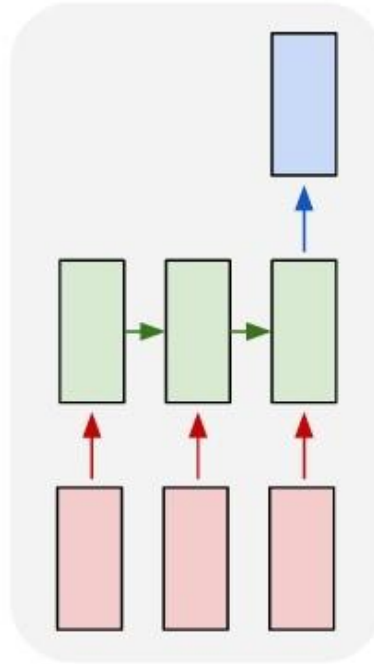
one to one



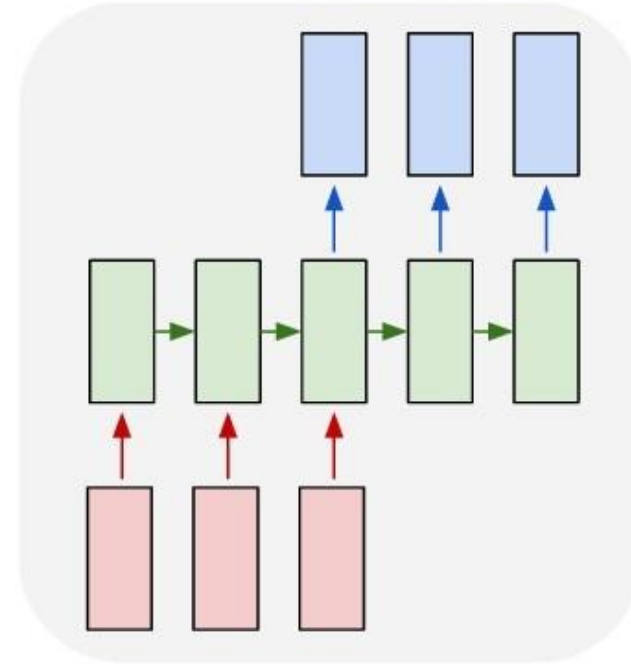
one to many



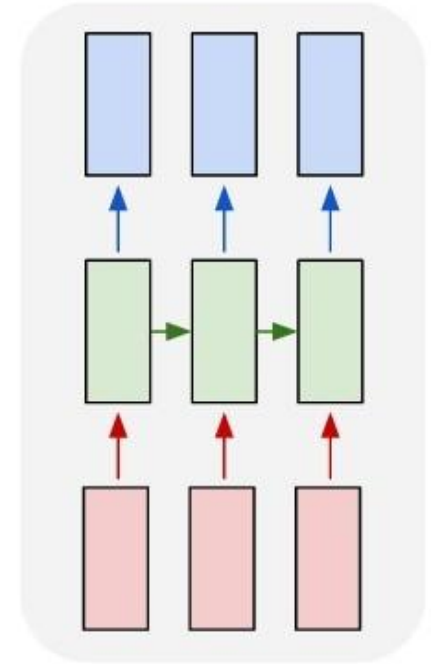
many to one



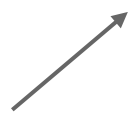
many to many



many to many

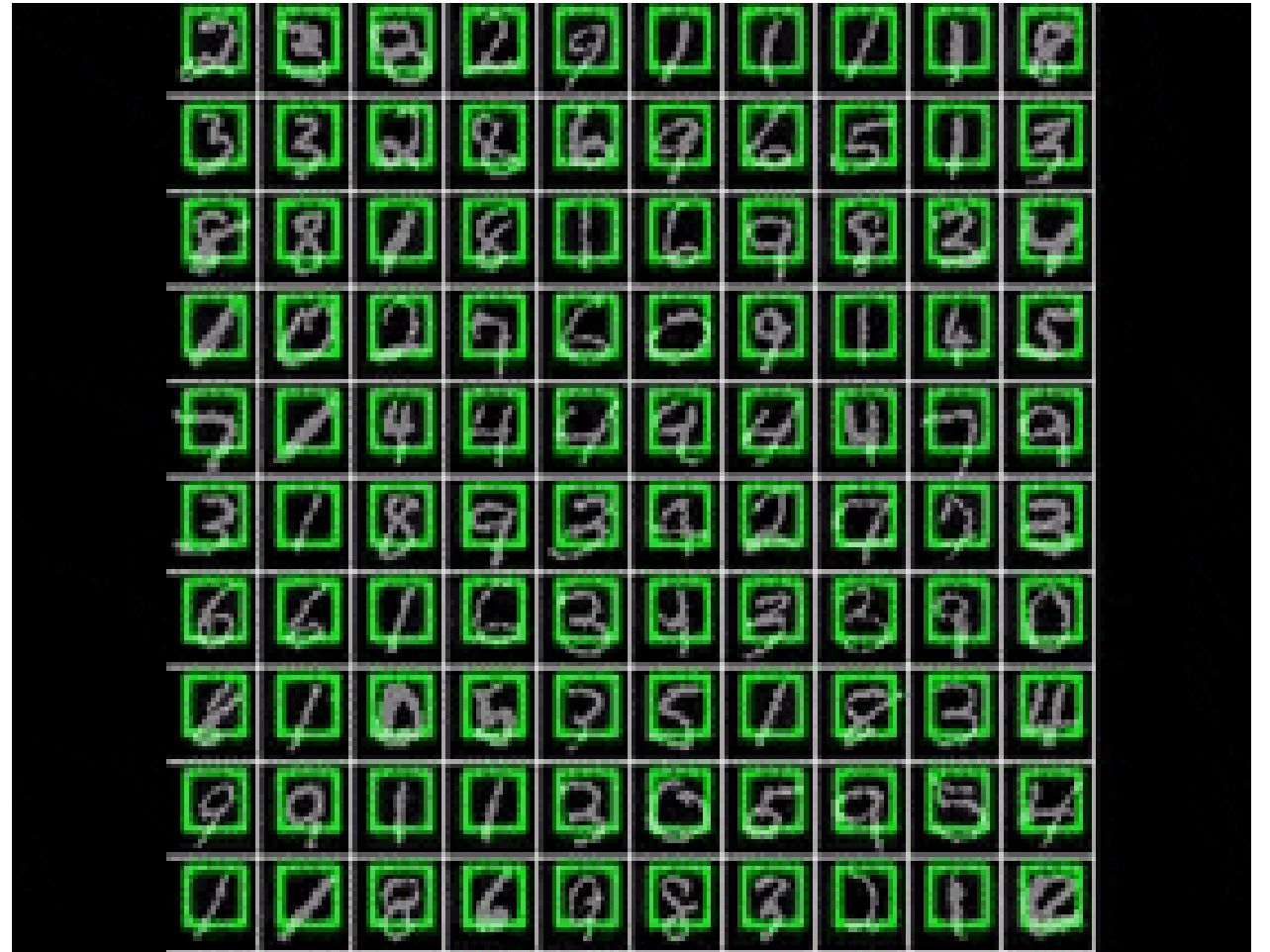


e.g. **Per-frame video classification:**  
Sequence of images -> Sequence of labels



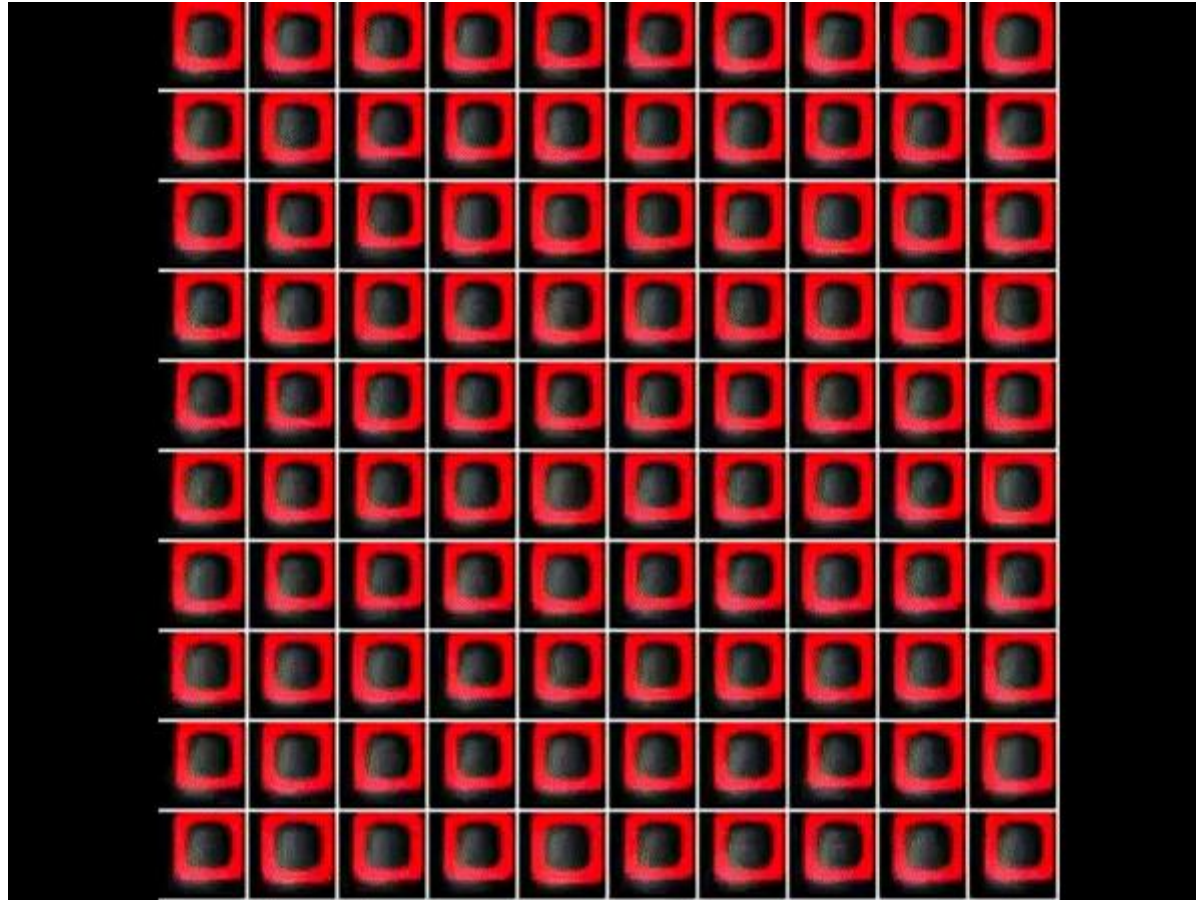
# Sequential Processing of Non-Sequential Data

Classify images by taking  
a series of “glimpses”



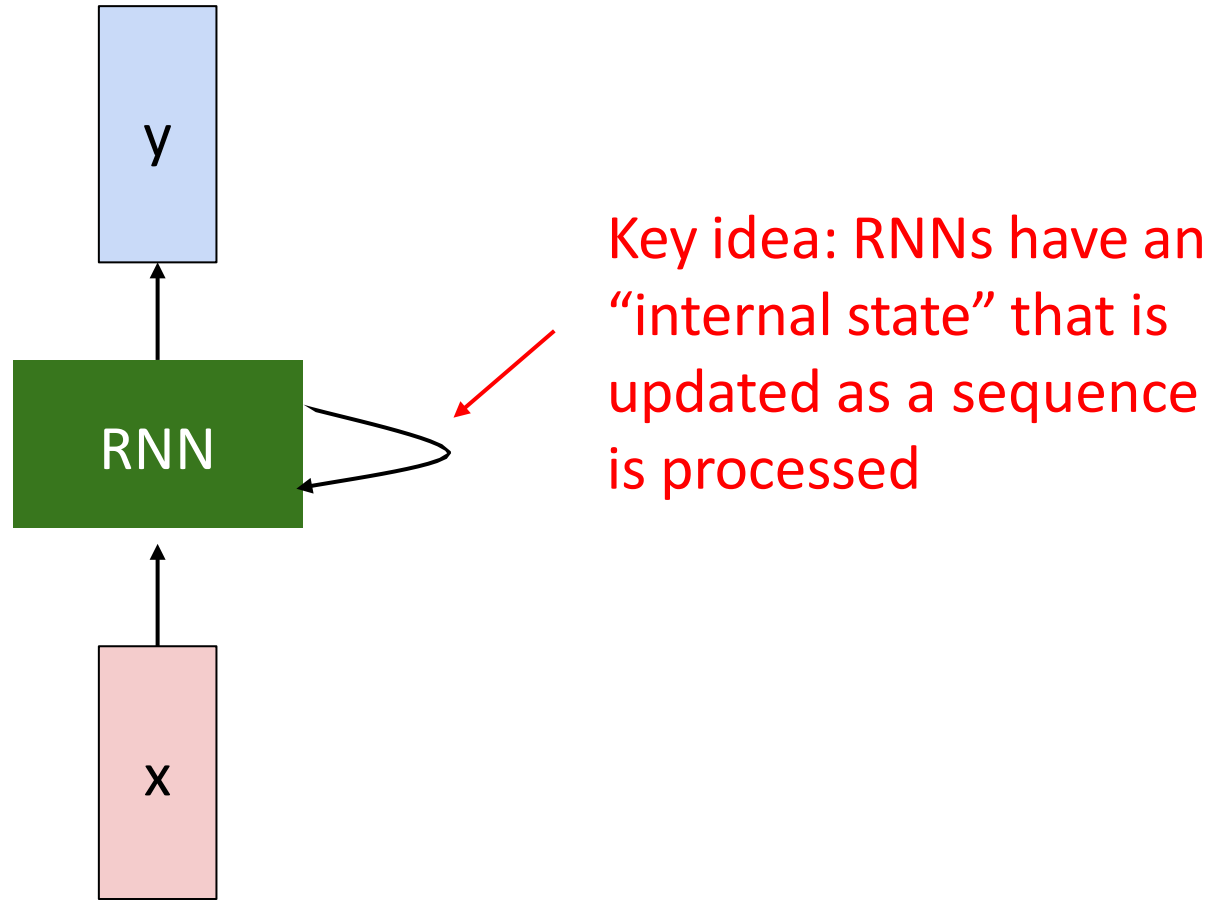
# Sequential Processing of Non-Sequential Data

Generate images one piece at a time

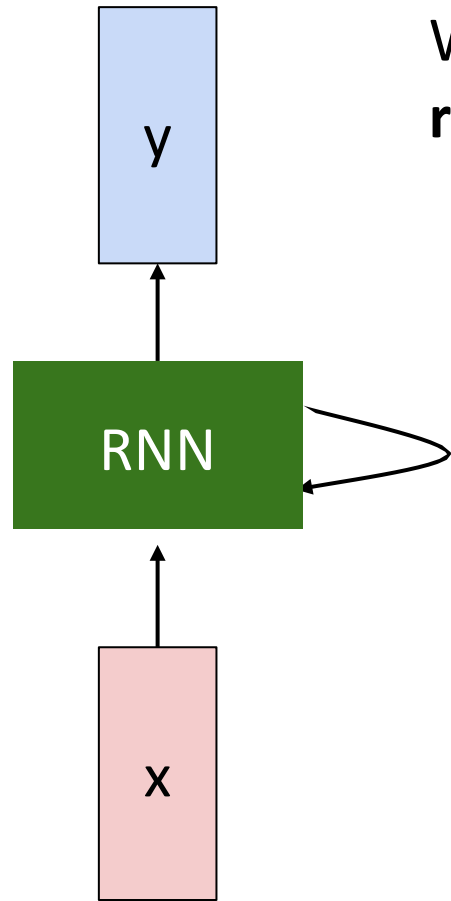




# Recurrent Neural Networks



# Recurrent Neural Networks



We can process a sequence of vectors  $\mathbf{x}$  by applying a **recurrence formula** at every time step:

$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state                      some function with parameters  $W$                       old state                      input vector at some time step

# (Vanilla) Recurrent Neural Networks

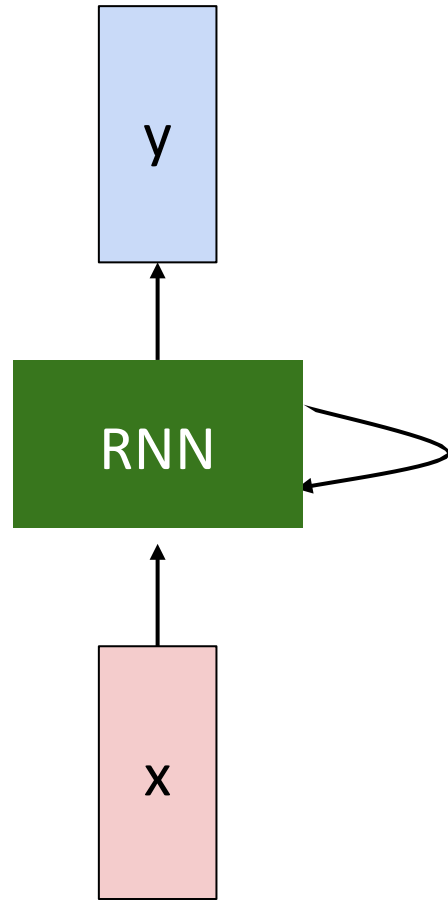
The state consists of a single “*hidden*” vector  $\mathbf{h}$ :

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

(also bias term)

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

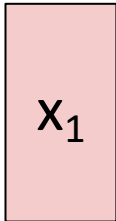
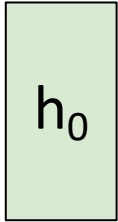
$$y_t = W_{hy}h_t$$



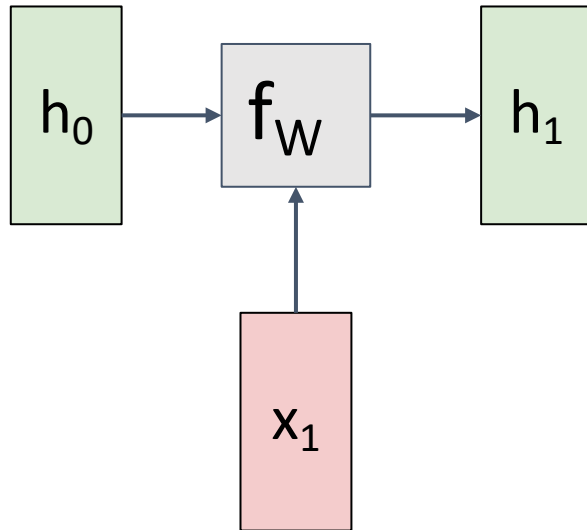
Sometimes called a “Vanilla RNN” or an “Elman RNN” after Prof. Jeffrey Elman

# RNN Computational Graph

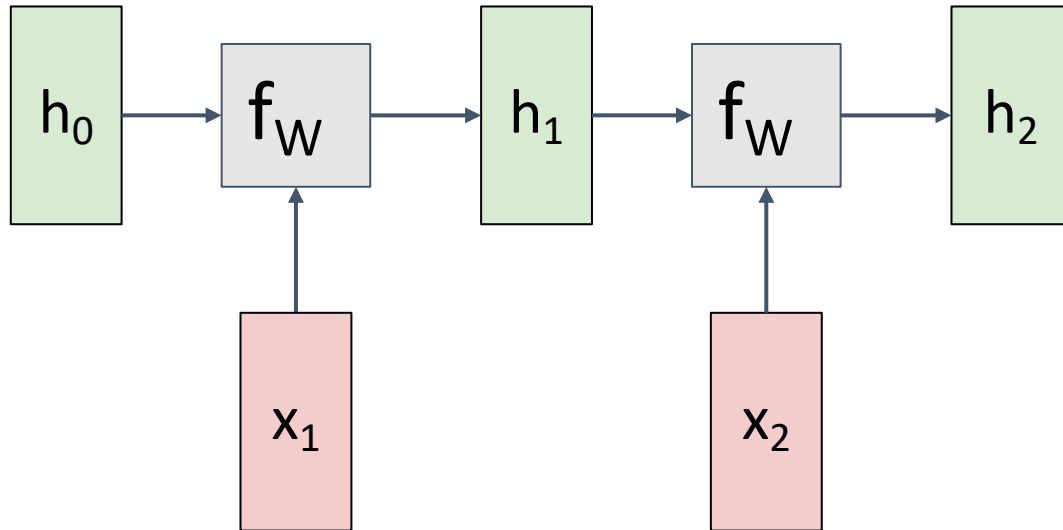
Initial hidden state  
Either set to all 0,  
Or learn it



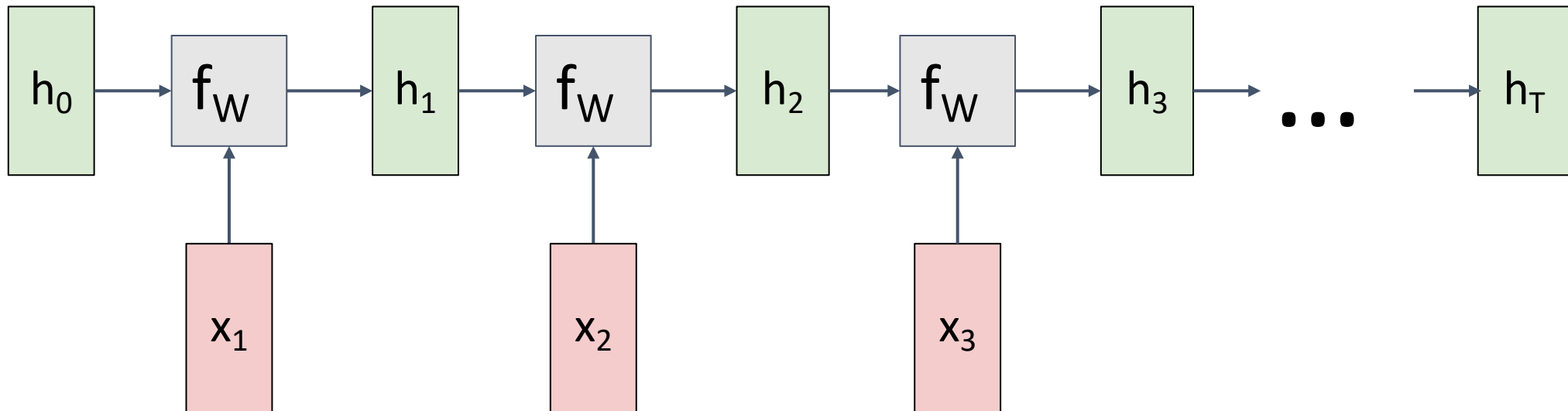
# RNN Computational Graph



# RNN Computational Graph

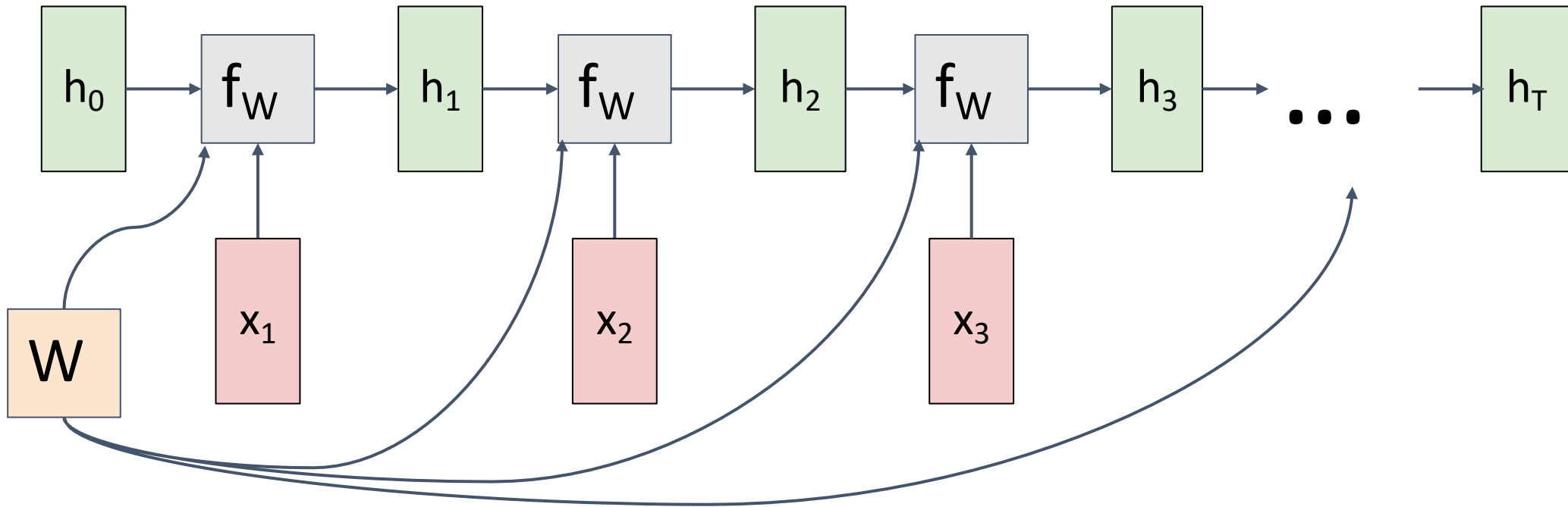


# RNN Computational Graph



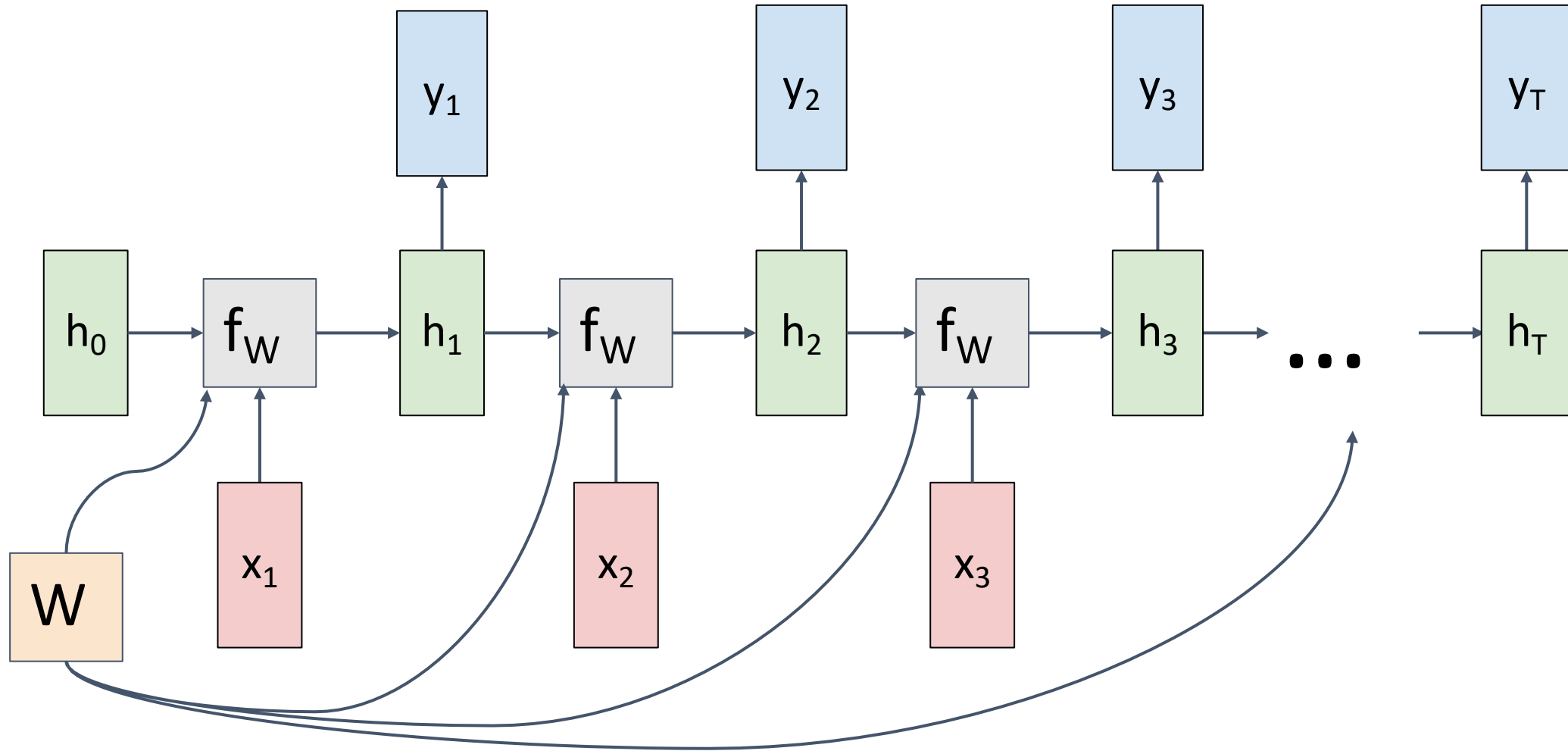
# RNN Computational Graph

Re-use the same weight matrix at every time-step

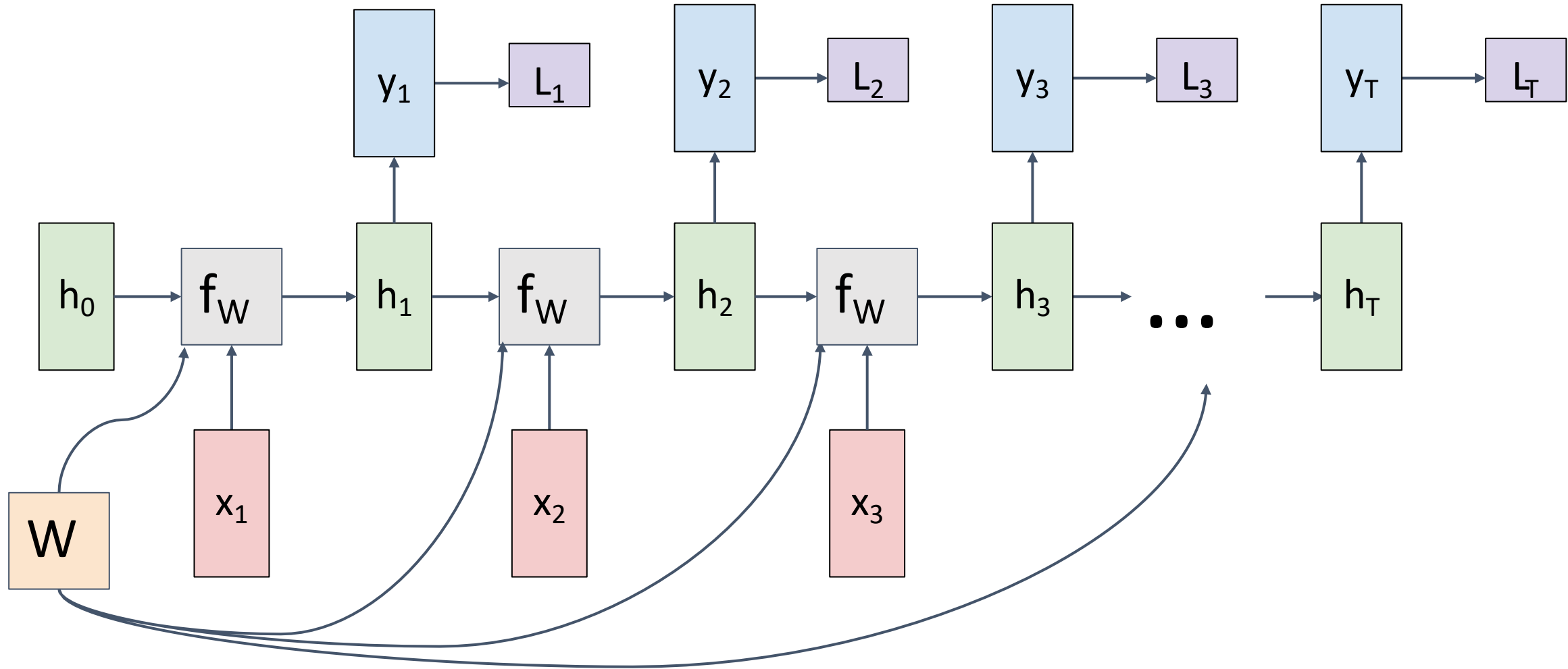




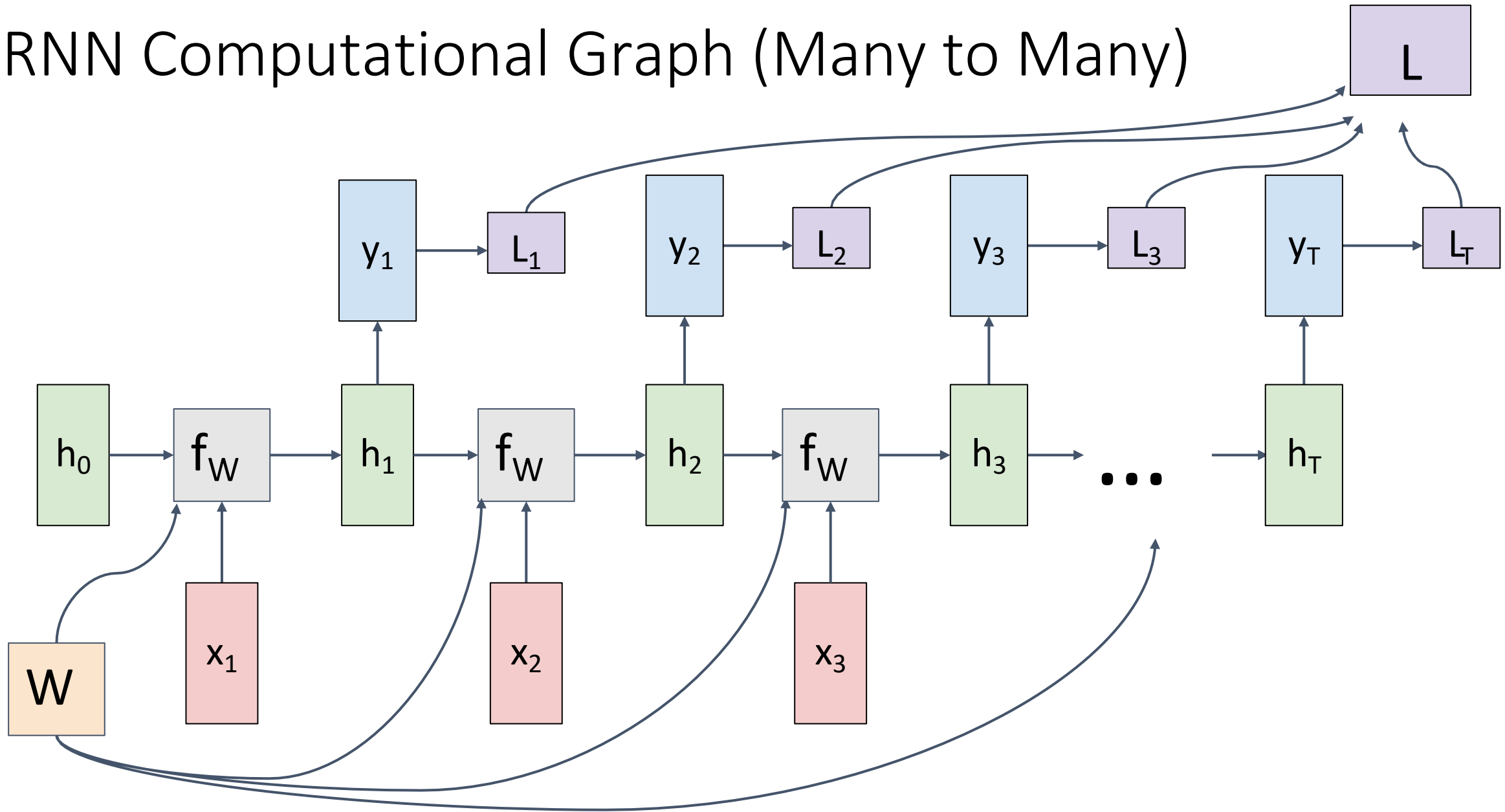
# RNN Computational Graph (Many to Many)



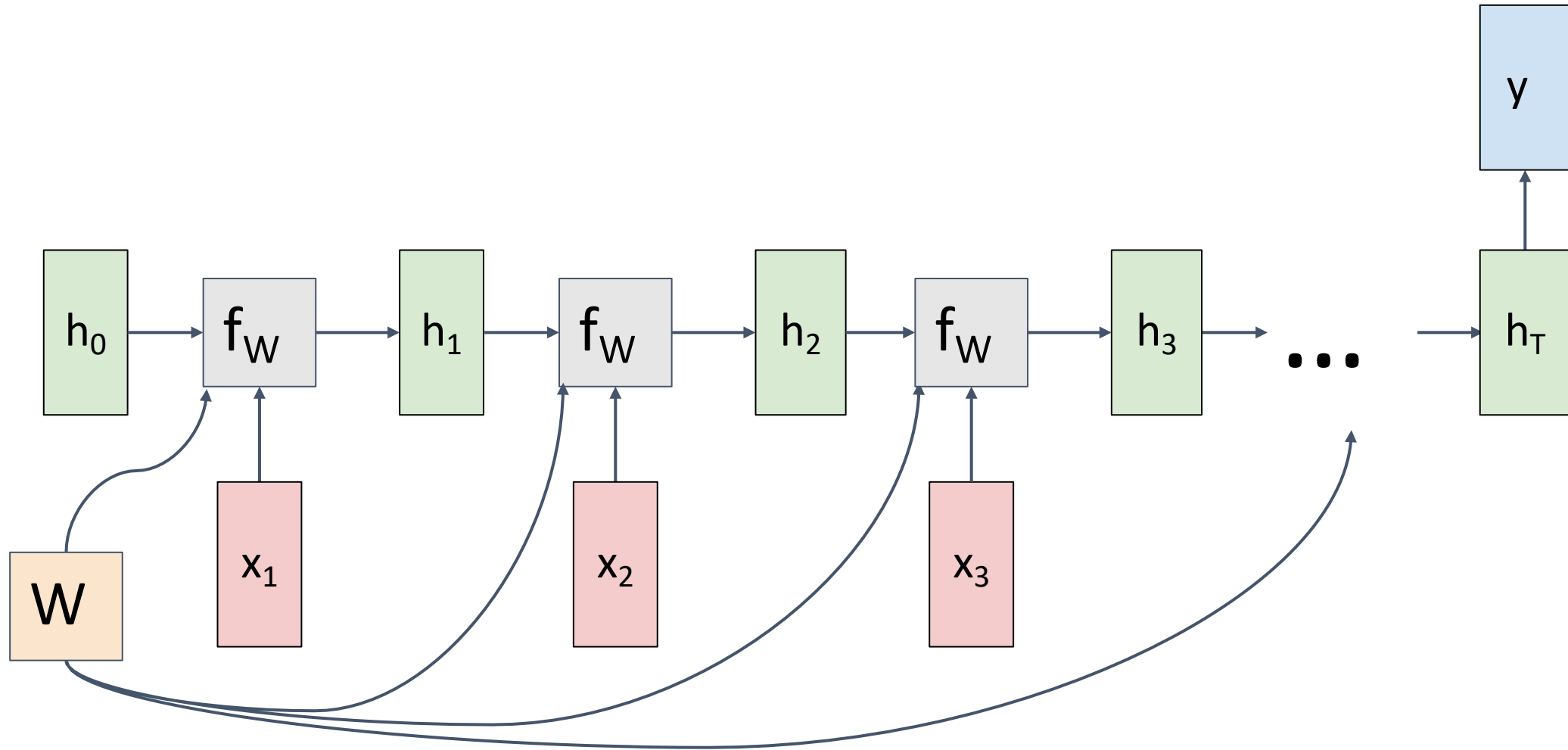
# RNN Computational Graph (Many to Many)



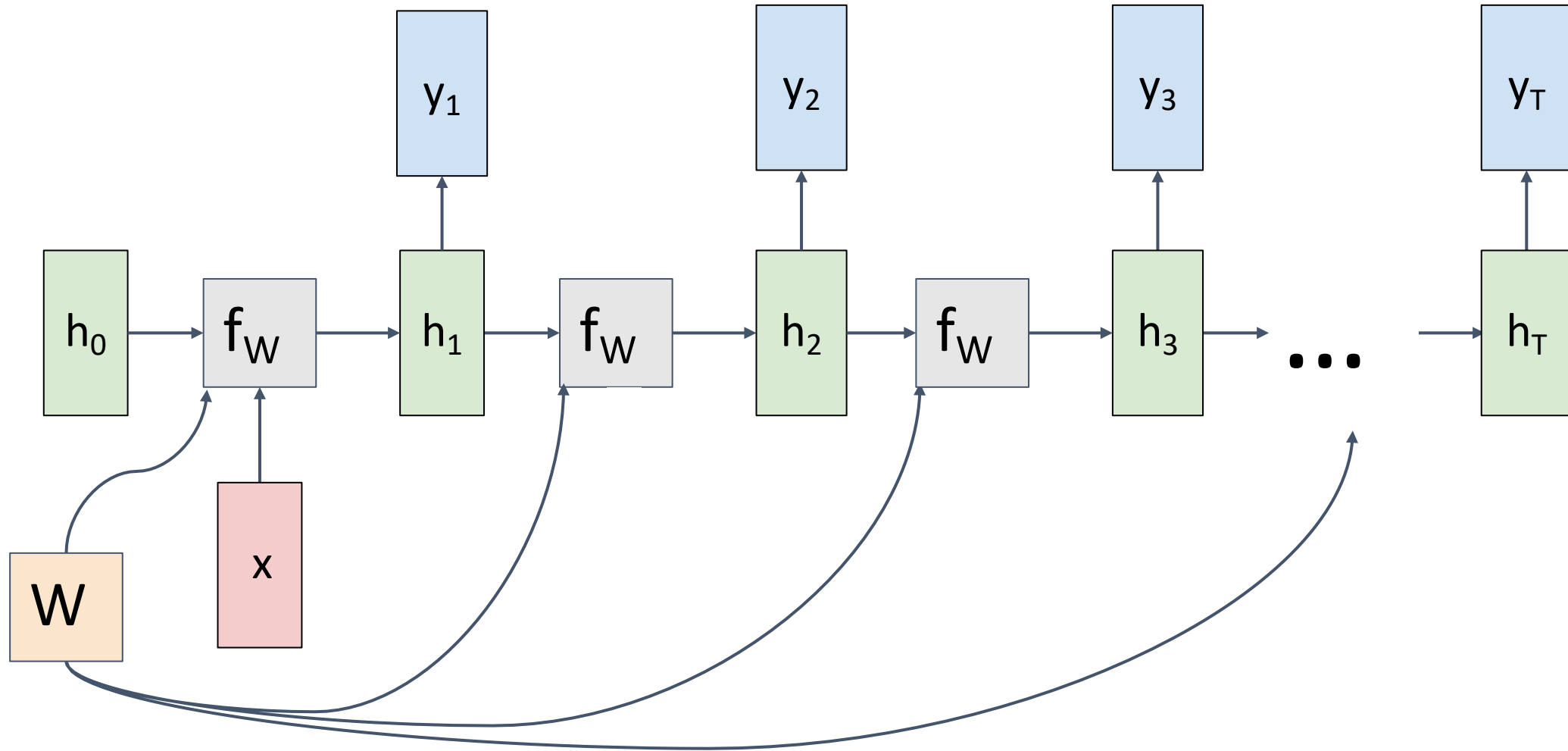
# RNN Computational Graph (Many to Many)



# RNN Computational Graph (Many to One)

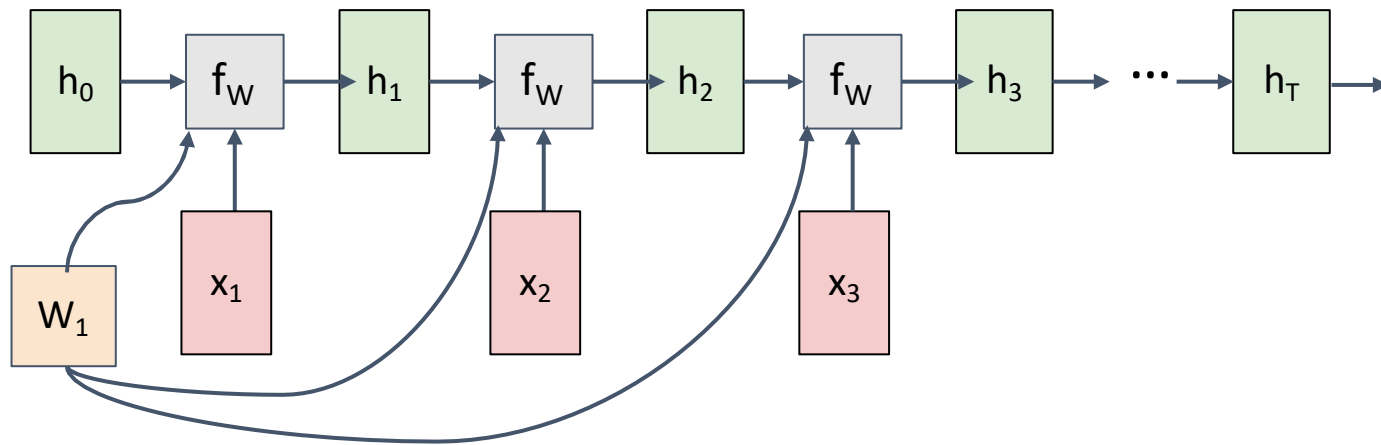


# RNN Computational Graph (One to Many)



# Sequence to Sequence (seq2seq) (Many to one) + (One to many)

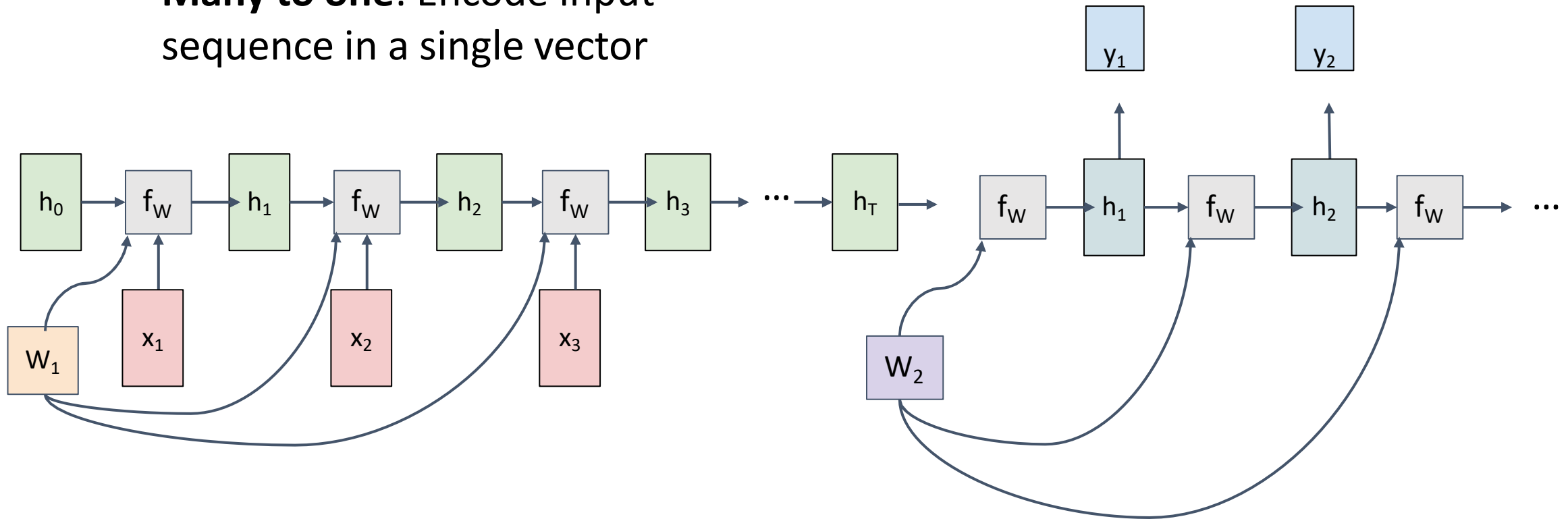
**Many to one:** Encode input sequence in a single vector



# Sequence to Sequence (seq2seq) (Many to one) + (One to many)

**One to many:** Produce output sequence from single input vector

**Many to one:** Encode input sequence in a single vector

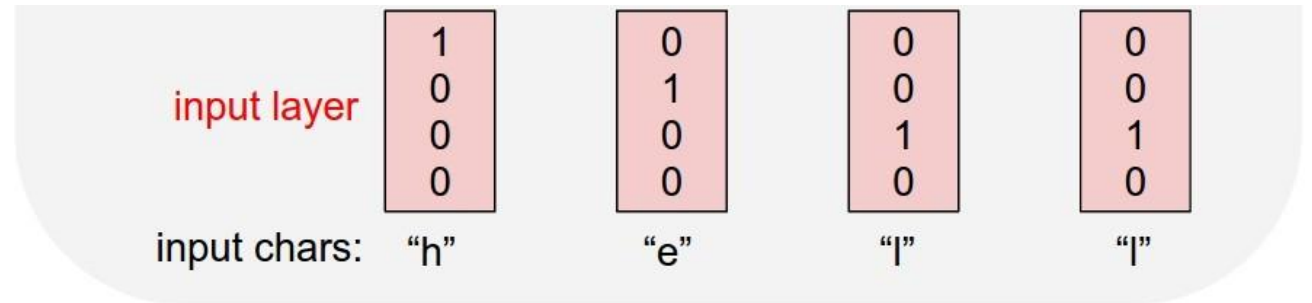


# Example: Language Modeling

Given characters 1, 2, ..., t,  
model predicts character t

Training sequence: "hello"

Vocabulary: [h, e, l, o]





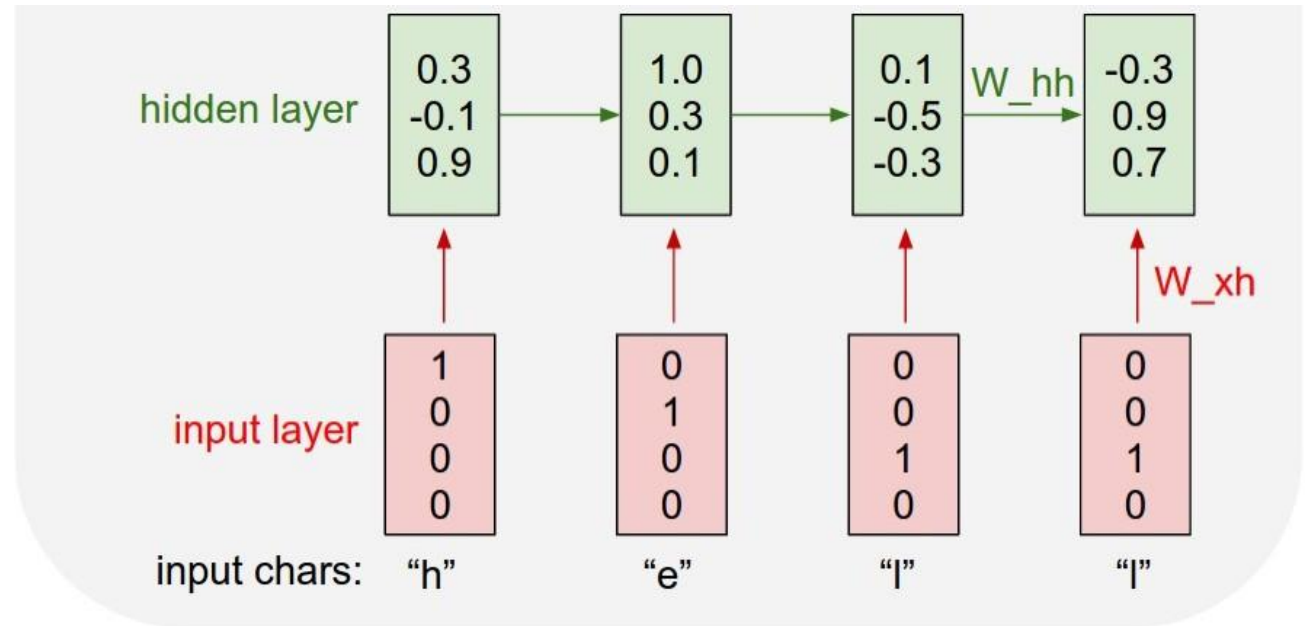
# Example: Language Modeling

Given characters 1, 2, ..., t,  
model predicts character t

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Training sequence: "hello"

Vocabulary: [h, e, l, o]



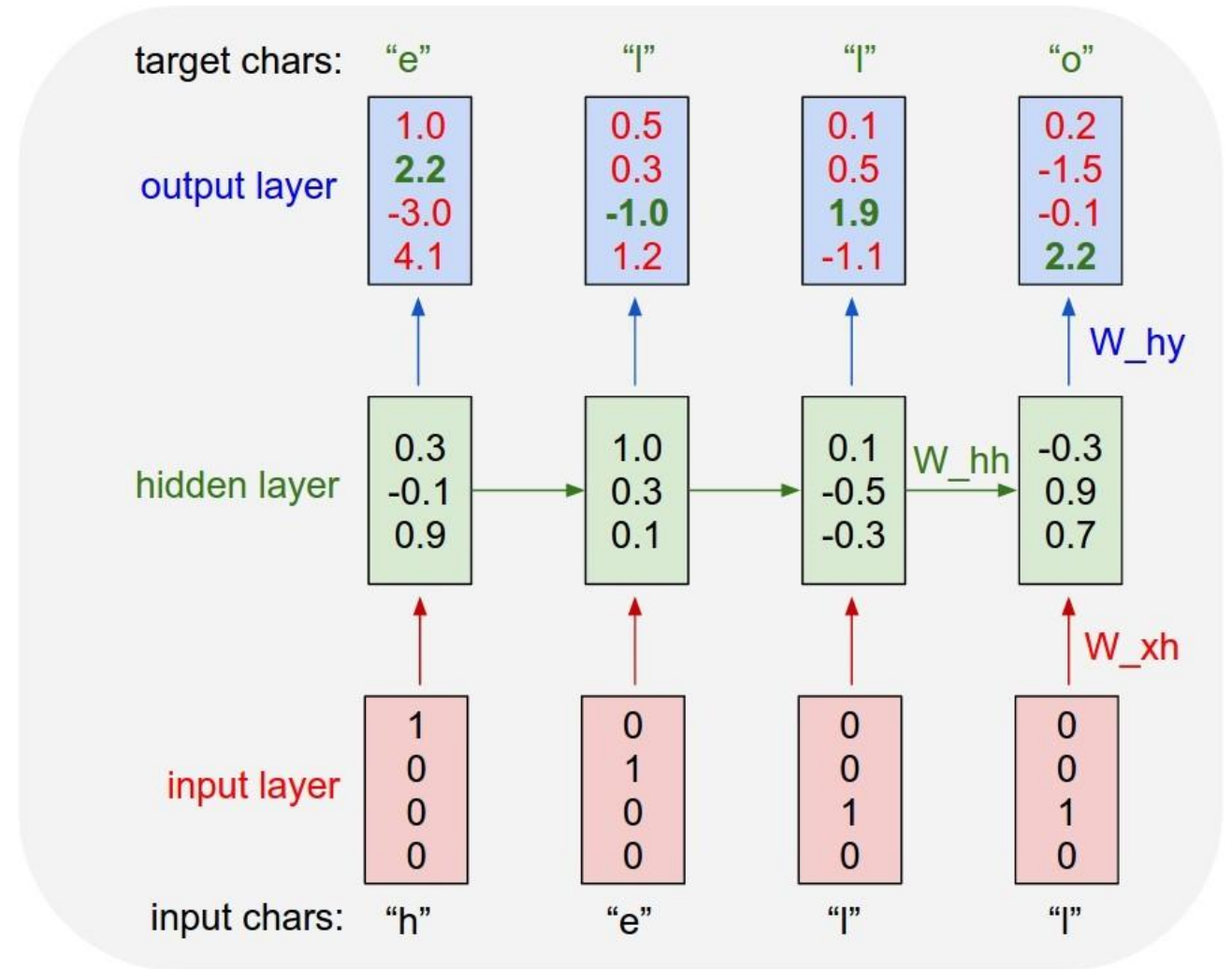
# Example: Language Modeling

Given characters 1, 2, ..., t,  
model predicts character t

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Training sequence: "hello"

Vocabulary: [h, e, l, o]



# Example: Language Modeling

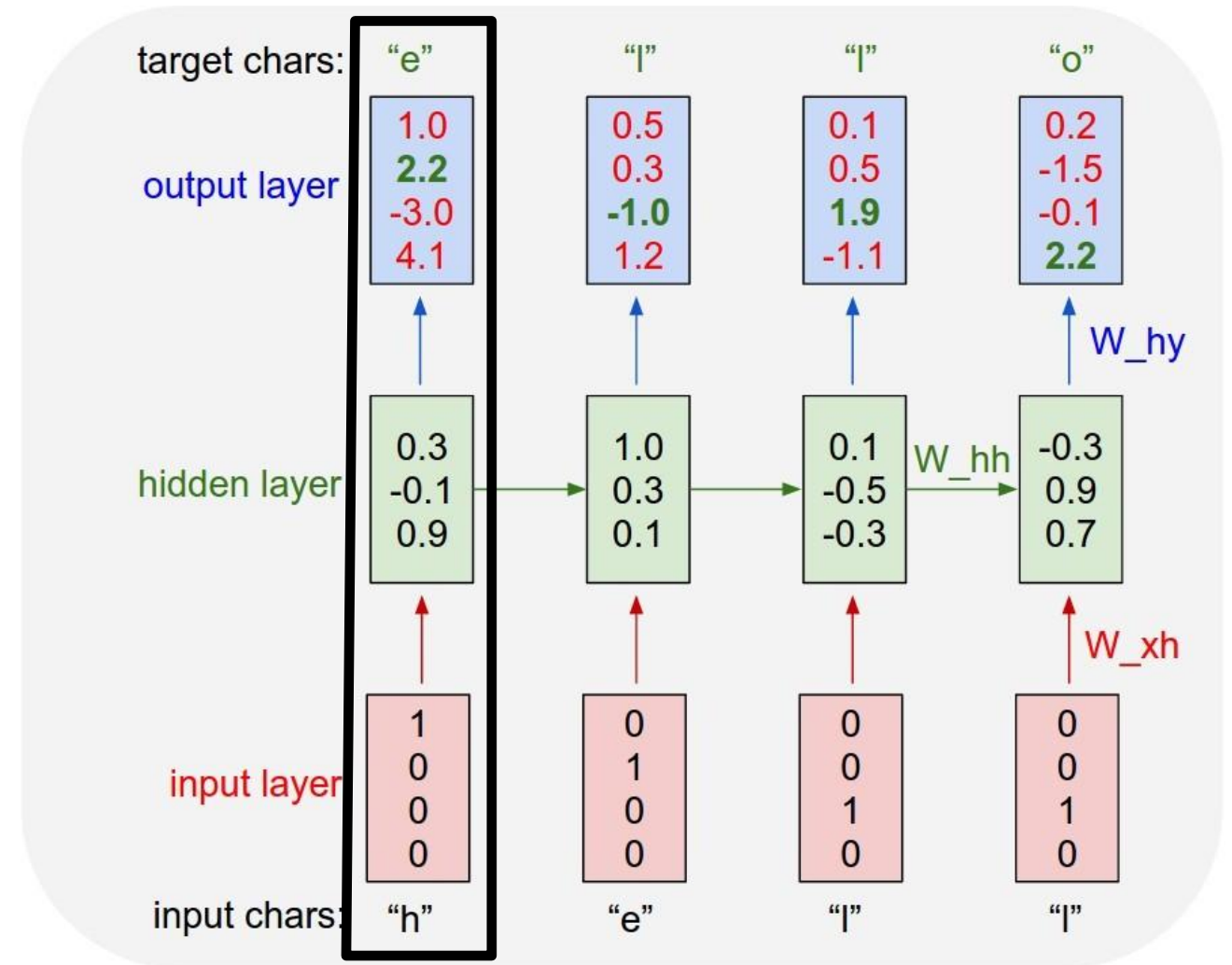
Given "h", predict "e"

Given characters 1, 2, ..., t,  
model predicts character t

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Training sequence: "hello"

Vocabulary: [h, e, l, o]



# Example: Language Modeling

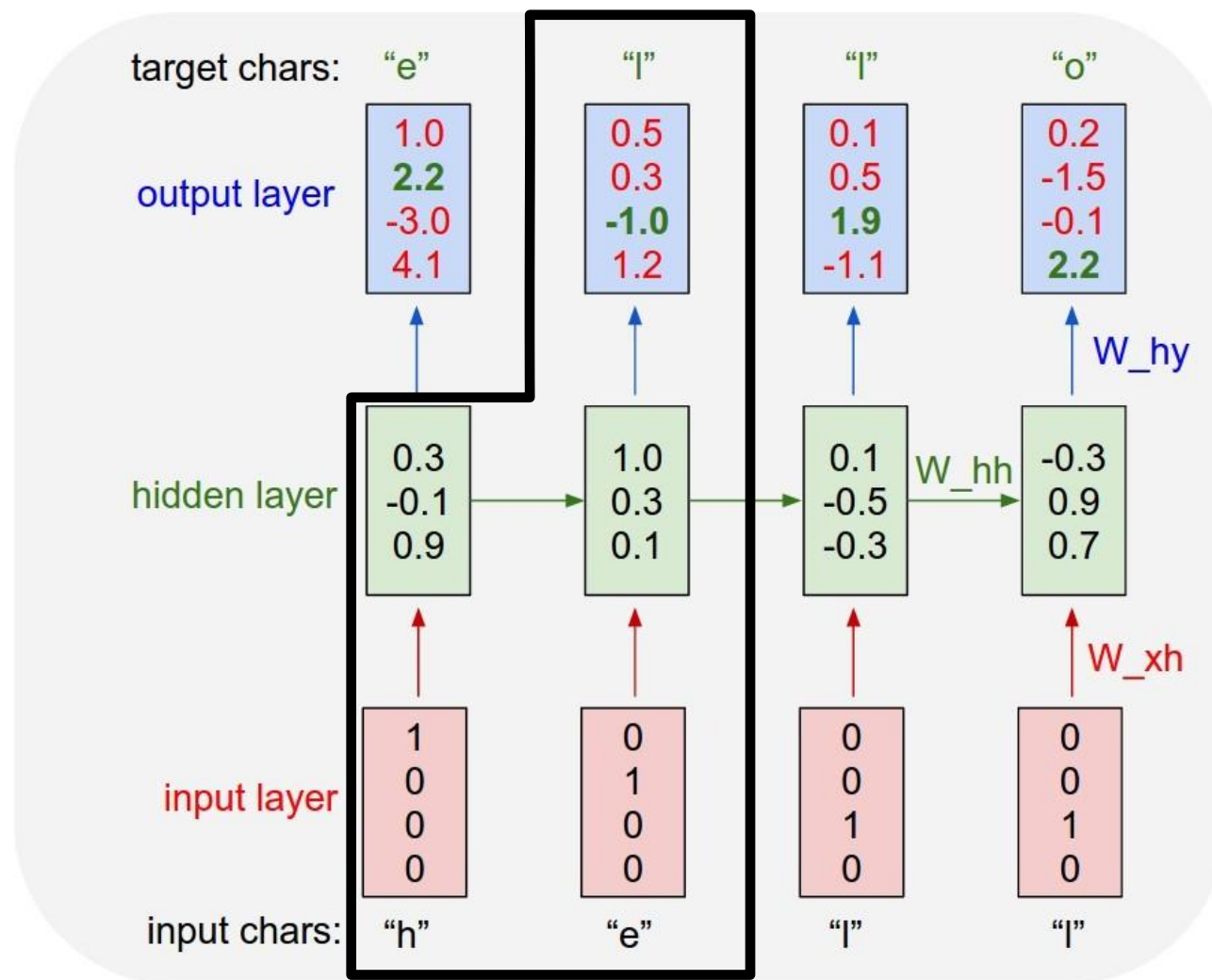
Given “he”, predict “l”

Given characters 1, 2, ..., t,  
model predicts character t

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Training sequence: “hello”

Vocabulary: [h, e, l, o]



# Example: Language Modeling

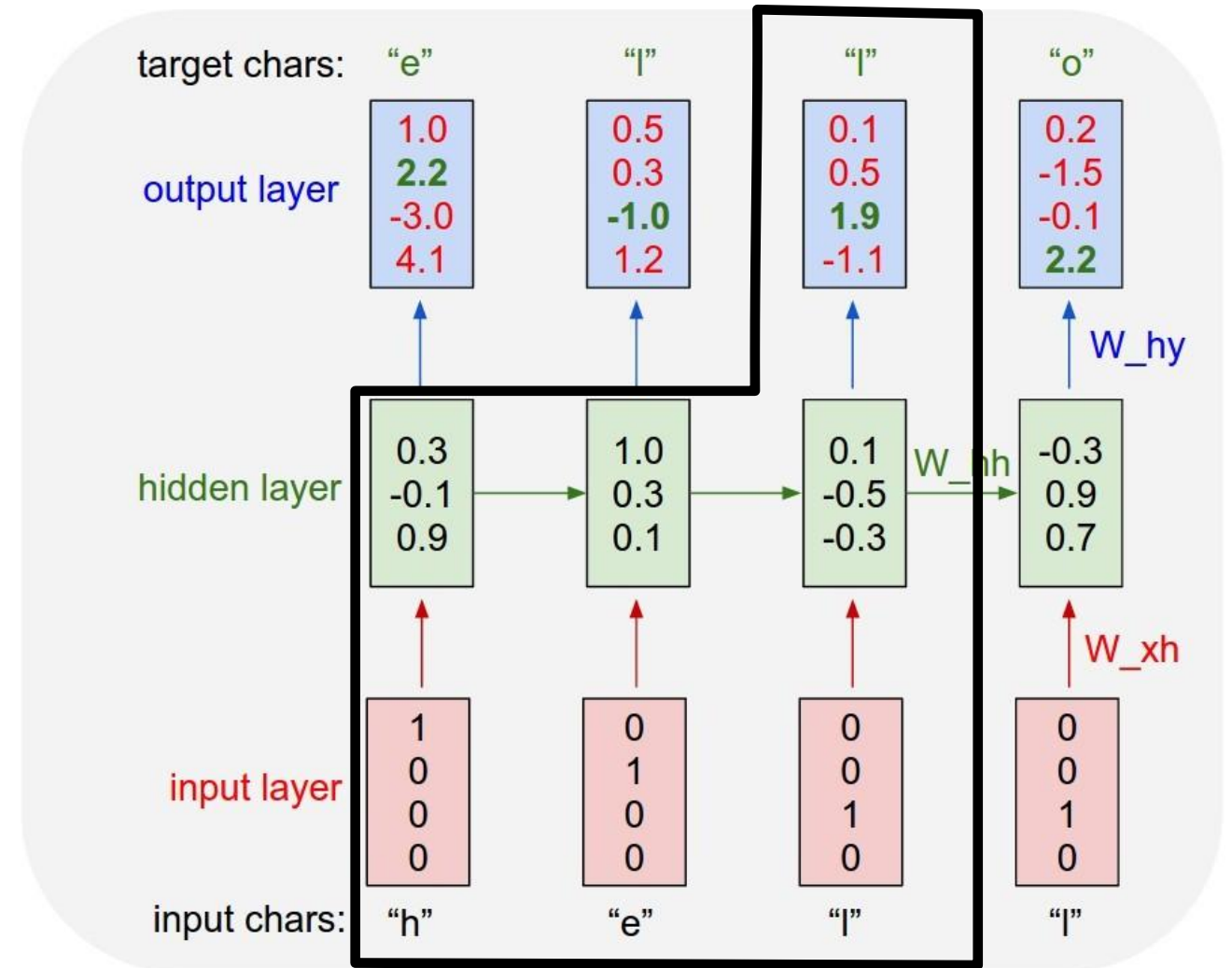
Given “hel”, predict “l”

Given characters 1, 2, ..., t,  
model predicts character t

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Training sequence: “hello”

Vocabulary: [h, e, l, o]



# Example: Language Modeling

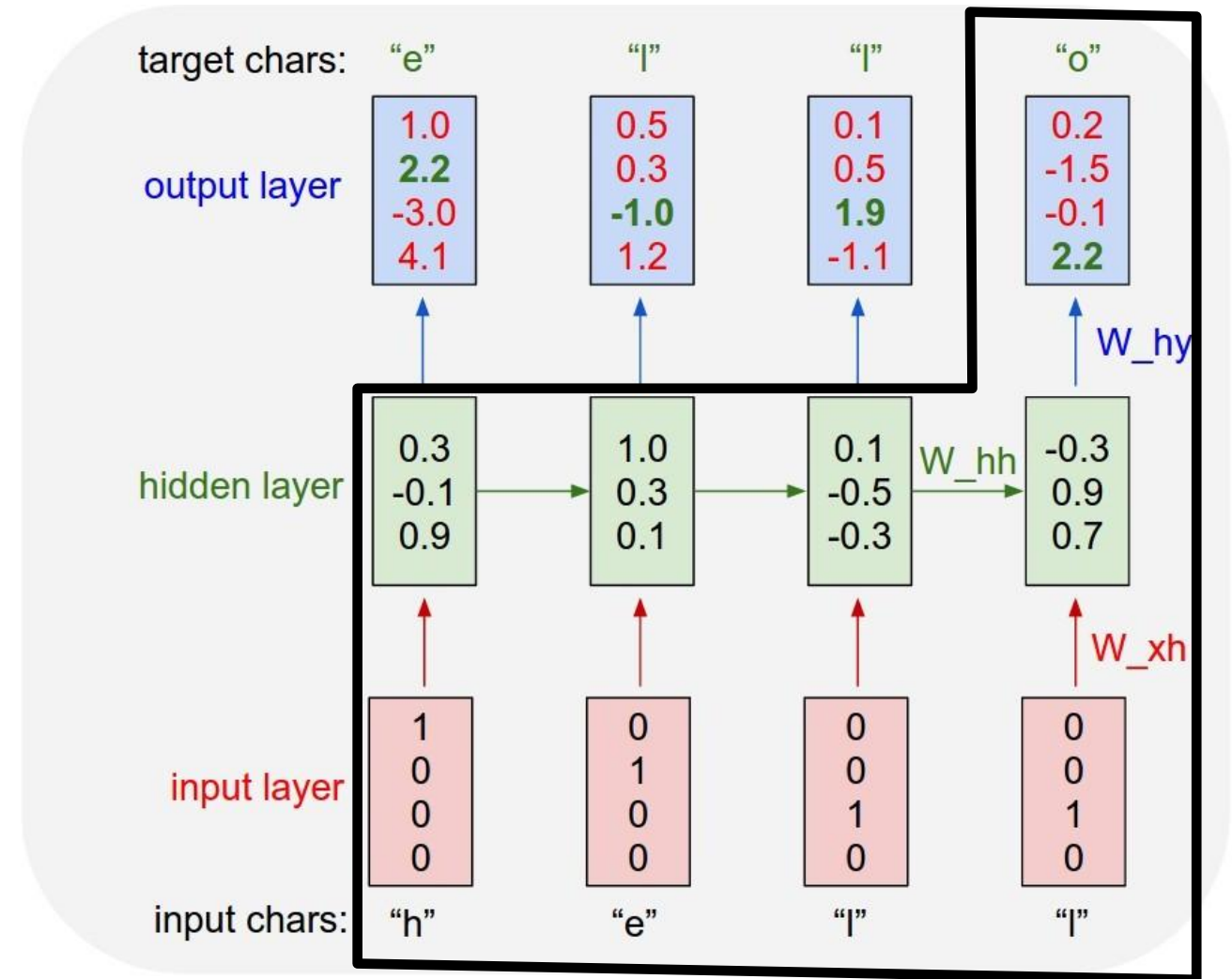
Given “hell”, predict “o”

Given characters 1, 2, ..., t,  
model predicts character t

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Training sequence: “hello”

Vocabulary: [h, e, l, o]



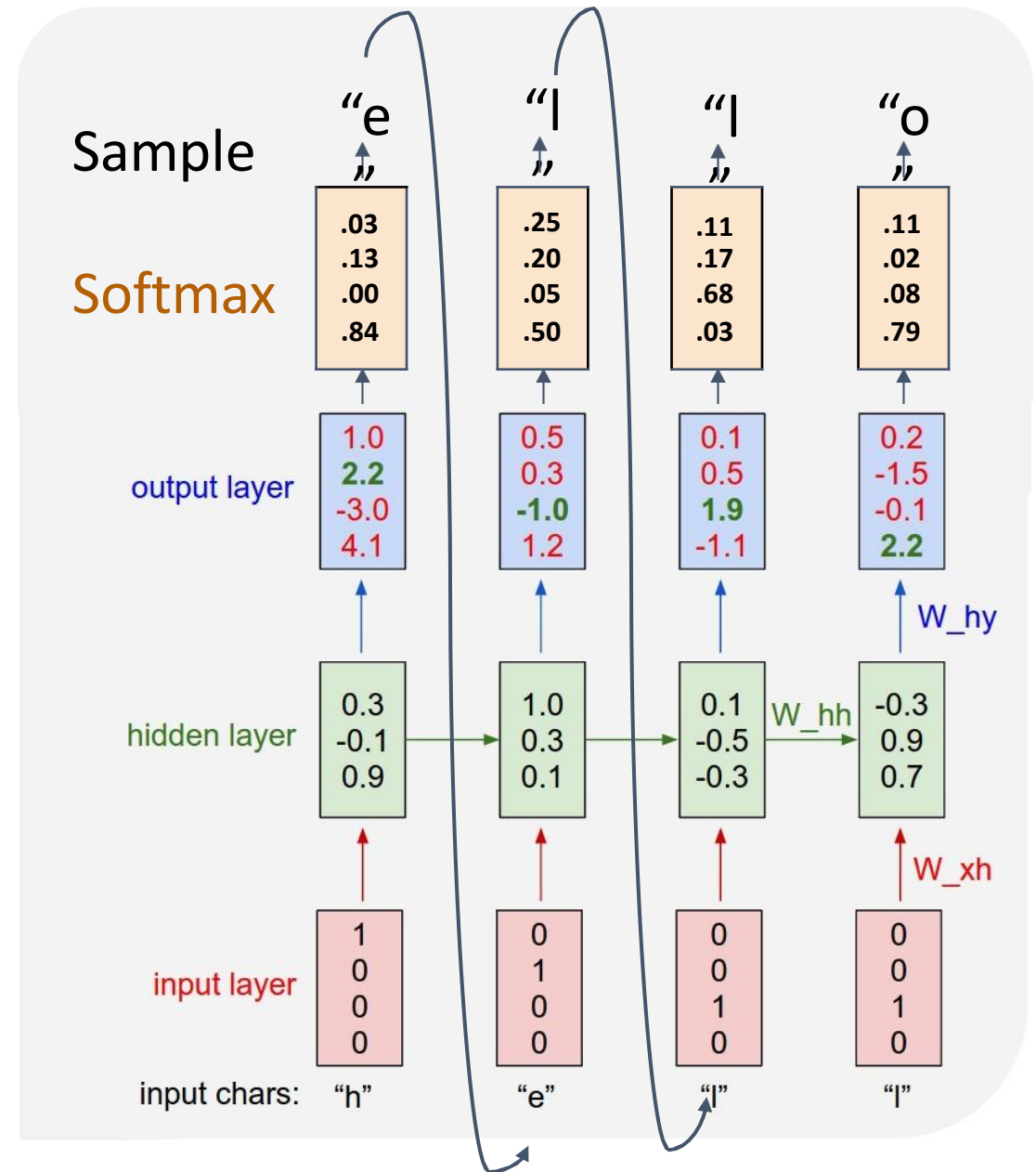


# Example: Language Modeling

So far: encode inputs  
as **one-hot-vector**

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{14} \\ w_{31} & w_{32} & w_{33} & w_{14} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} w_{11} \\ w_{21} \\ w_{31} \end{bmatrix}$$

Matrix multiply with a one-hot vector just  
extracts a column from the weight matrix.  
Often extract this into a separate  
**embedding** layer

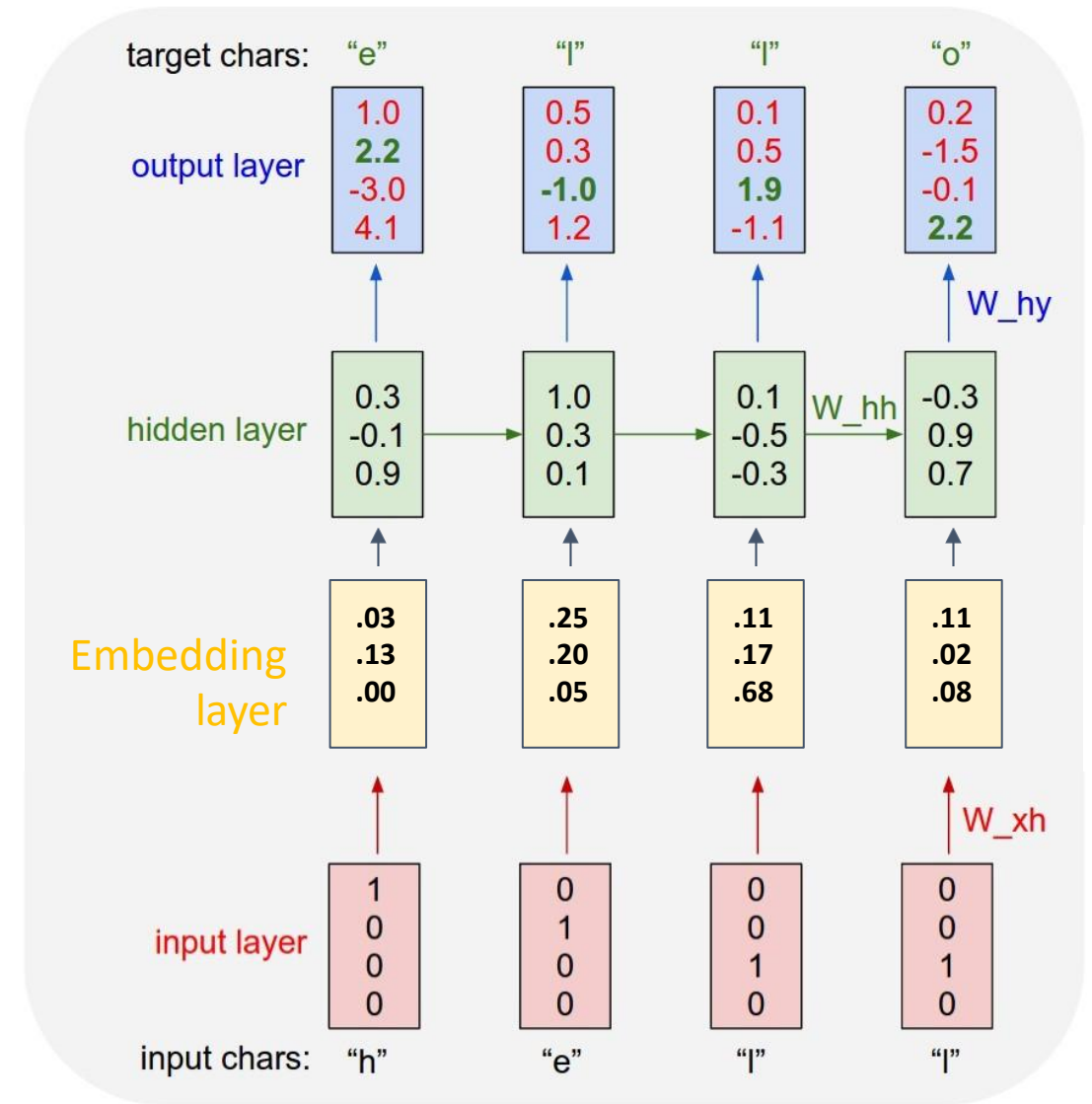


# Example: Language Modeling

So far: encode inputs  
as **one-hot-vector**

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{14} \\ w_{31} & w_{32} & w_{33} & w_{14} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} w_{11} \\ w_{21} \\ w_{31} \end{bmatrix}$$

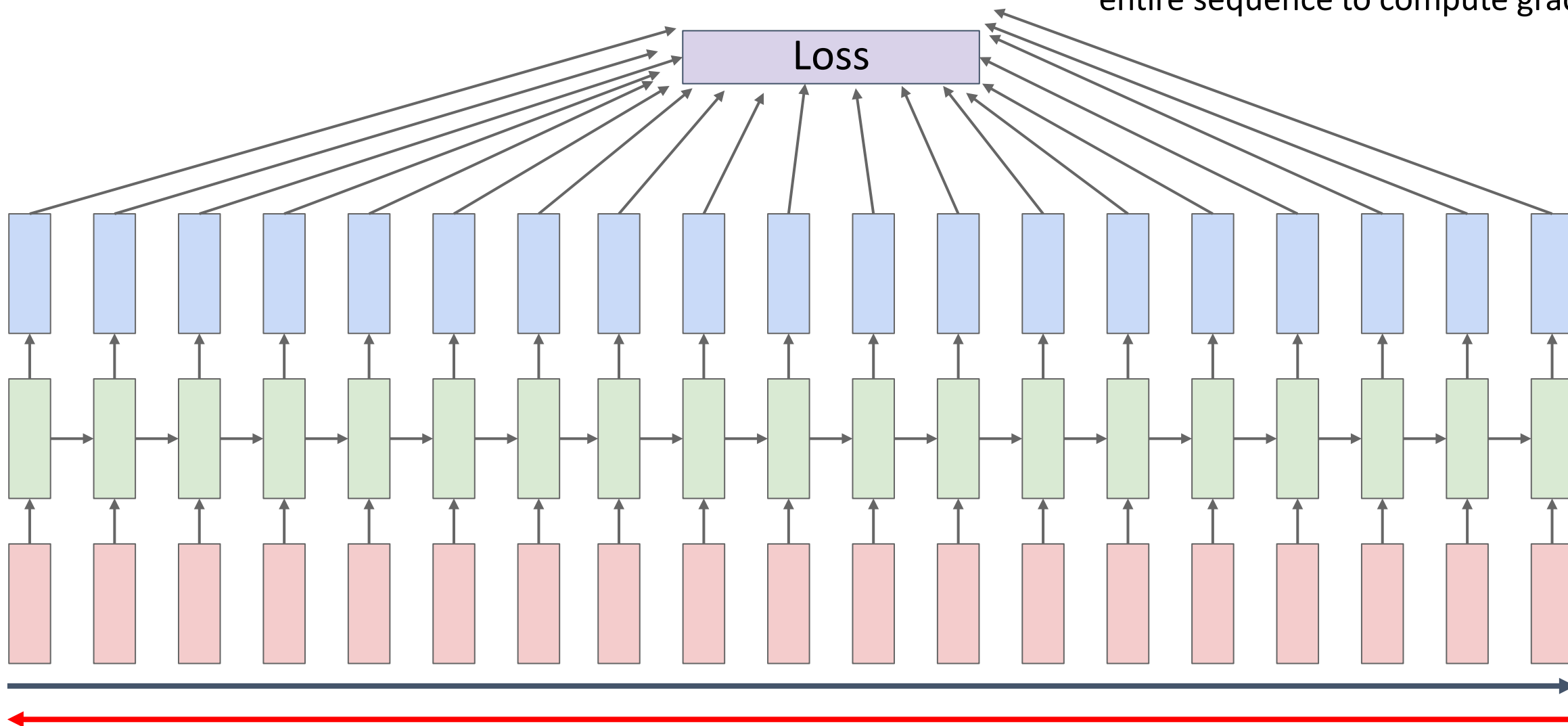
Matrix multiply with a one-hot vector just  
extracts a column from the weight matrix.  
Often extract this into a separate  
**embedding** layer



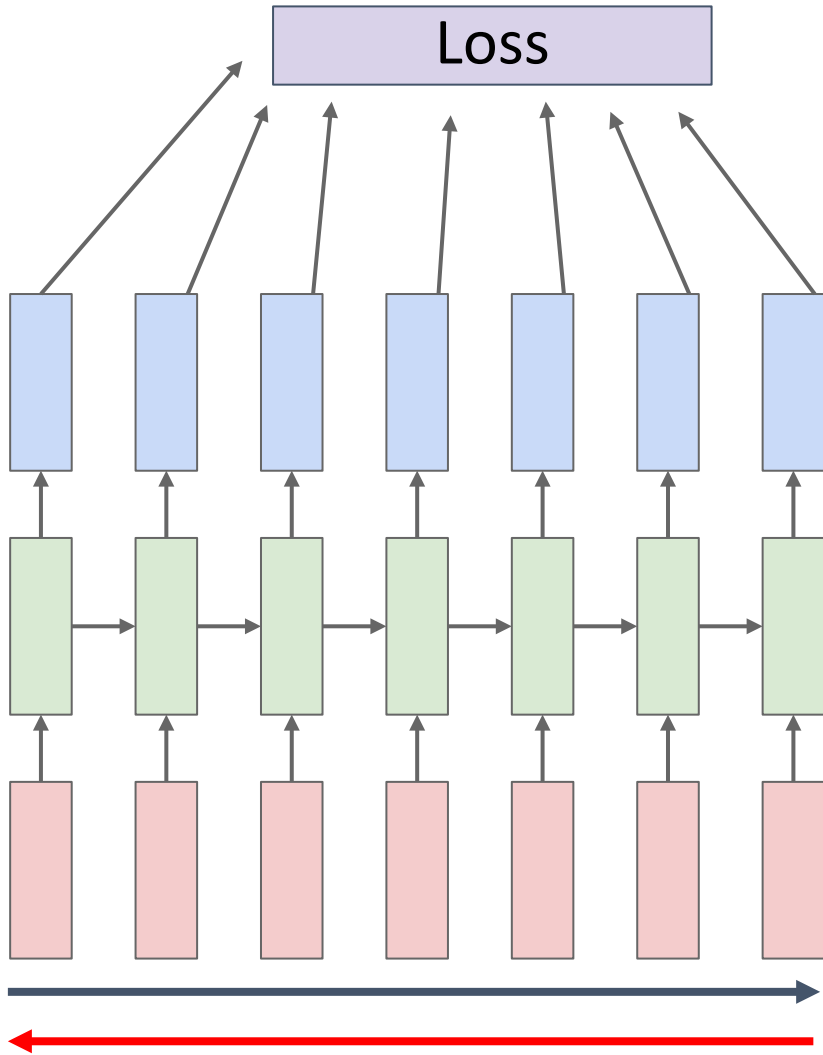


# Backpropagation Through Time

Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient

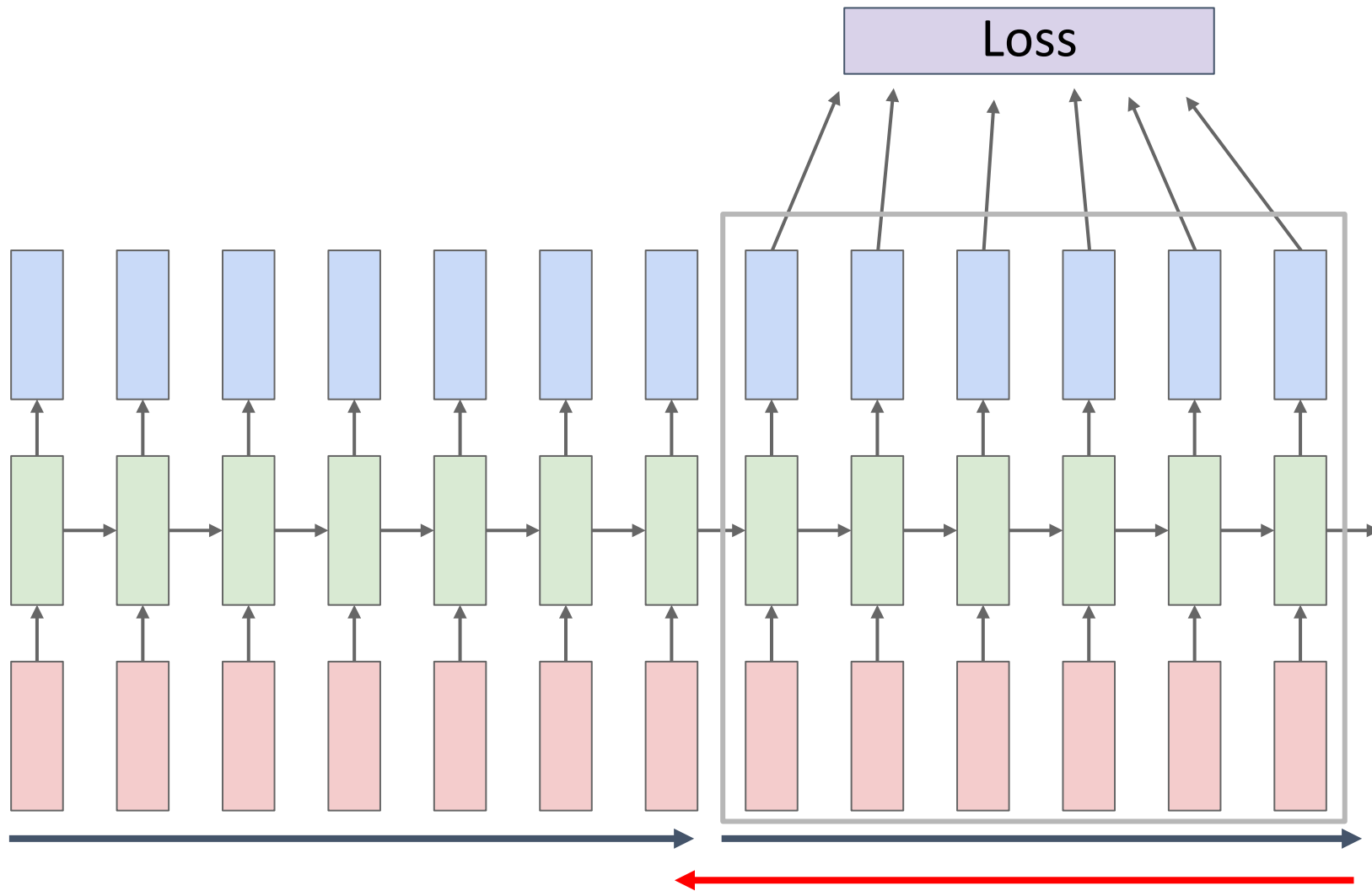


# Truncated Backpropagation Through Time



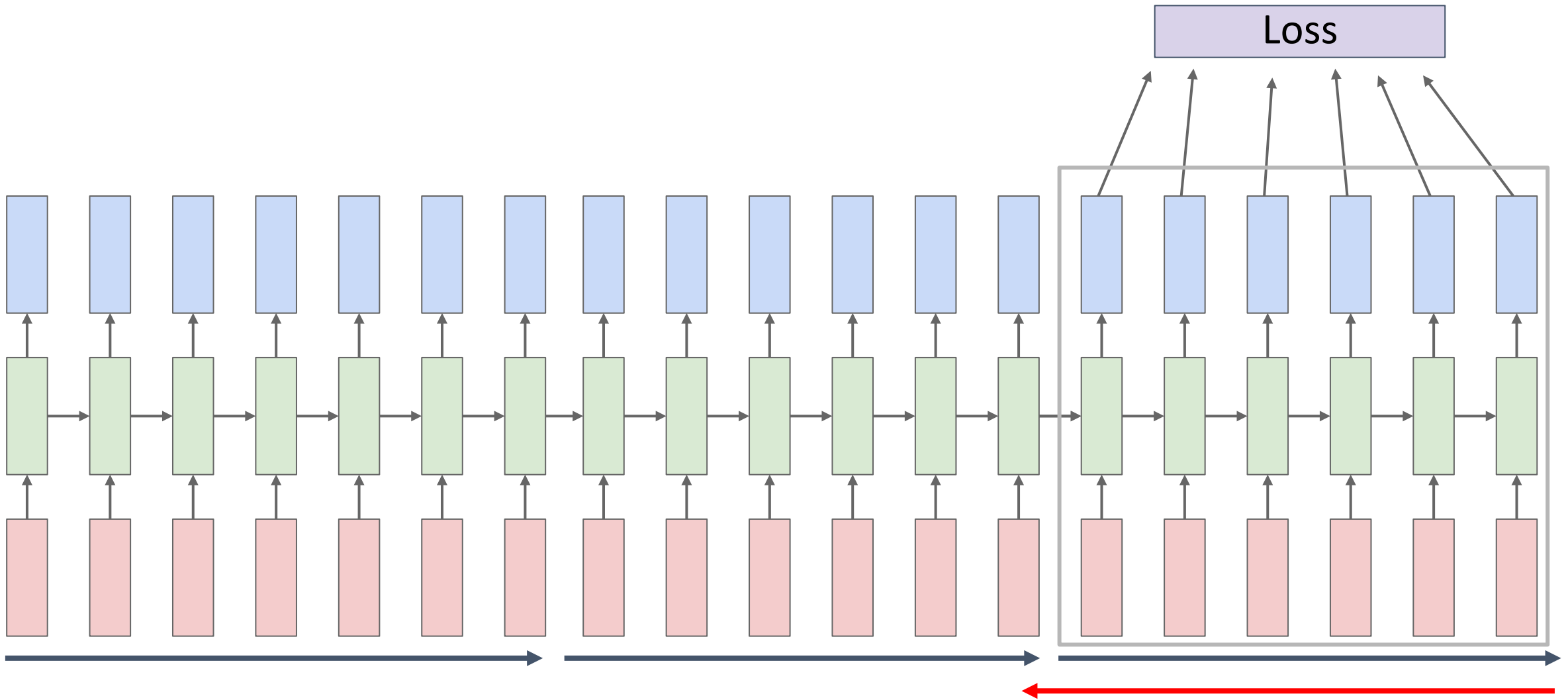
Run forward and backward through chunks of the sequence instead of whole sequence

# Truncated Backpropagation Through Time



Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

# Truncated Backpropagation Through Time

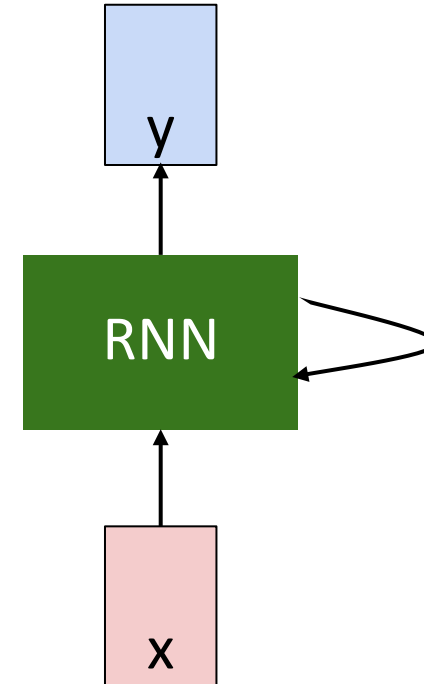


# THE SONNETS

by William Shakespeare

From fairest creatures we desire increase,  
That thereby beauty's rose might never die,  
But as the ripper should by time decease,  
His tender heir might bear his memory:  
But thou, contracted to thine own bright eyes,  
Feed'st thy light's flame with self-substantial fuel,  
Making a famine where abundance lies,  
Thyself thy foe, to thy sweet self too cruel:  
Thou that art now the world's fresh ornament,  
And only herald to the gaudy spring,  
Within thine own bud buriest thy content,  
And tender churl mak'st waste in niggarding:  
    Pity the world, or else this glutton be,  
    To eat the world's due, by the grave and thee.

When forty winters shall besiege thy brow,  
And dig deep trenches in thy beauty's field,  
Thy youth's proud livery so gazed on now,  
Will be a tatter'd weed of small worth held:  
Then being asked, where all thy beauty lies,  
Where all the treasure of thy lusty days;  
To say, within thine own deep sunken eyes,  
Were an all-eating shame, and thriftless praise.  
How much more praise deserv'd thy beauty's use,  
If thou couldst answer 'This fair child of mine  
Shall sum my count, and make my old excuse,'  
Proving his beauty by succession thine!  
    This were to be new made when thou art old,  
    And see thy blood warm when thou feel'st it cold.



at first:

tyntd-iafhatawiaoighrdemot lytdws e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e  
plia tklrqd t o idoe ns,smtt h ne etie h,hregtrs nigtike,aoaenns lng

at first:

tyntd-iafhatawiaoihrdemot lytdws e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e  
plia tklrqd t o idoe ns,smtt h ne etie h,hregtrs nigtike,aoaenns lng



train more

"Tmont thithey" fomesscerliund  
Keushey. Thom here  
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome  
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."

at first:

tyntd-iafhatawiaoihrdemot lytdws e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e  
plia tklrqd t o idoe ns,smtt h ne etie h,hregtrs nigtike,aoaenns lng



train more

"Tmont thithey" fomesscerliund  
Keushey. Thom here  
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome  
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."



train more

Aftair fall unsuch that the hall for Prince Velzonski's that me of  
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort  
how, and Gogition is so overelical and ofter.



at first:

tyntd-iafhatawiaoihrdemot lytdws e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e  
plia tklrqd t o idoe ns,smtt h ne etie h,hregtrs nigtike,aoaenns lng

train more

"Tmont thithey" fomesscerliund  
Keushey. Thom here  
sheulke, anmerenith ol sivh I lalterthend Bleipile shuw y fil on aseterlome  
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."

train more

Aftair fall unsuch that the hall for Prince Velzonski's that me of  
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort  
how, and Gogition is so overelical and ofter.

train more

"Why do what that day," replied Natasha, and wishing to himself the fact the  
princess, Princess Mary was easier, fed in had oftened him.  
Pierre aking his soul came to the packs and drove up his father-in-law women.

PANDARUS:

Alas, I think he shall be come approached and the day  
When little strain would be attain'd into being never fed,  
And who is but a chain and subjects of his death,  
I should not sleep.

Second Senator:

They are away this miseries, produced upon my soul,  
Breaking and strongly should be buried, when I perish  
The earth and thoughts of many states.

DUKE VINCENTIO:

Well, your wit is in the care of side and that.

Second Lord:

They would be ruled after this chamber, and  
my fair nudes begun out of the fact, to be conveyed,  
Whose noble souls I'll have the heart of the wars.

Clown:

Come, sir, I will make did behold your worship.

VIOLA:

I'll drink it.

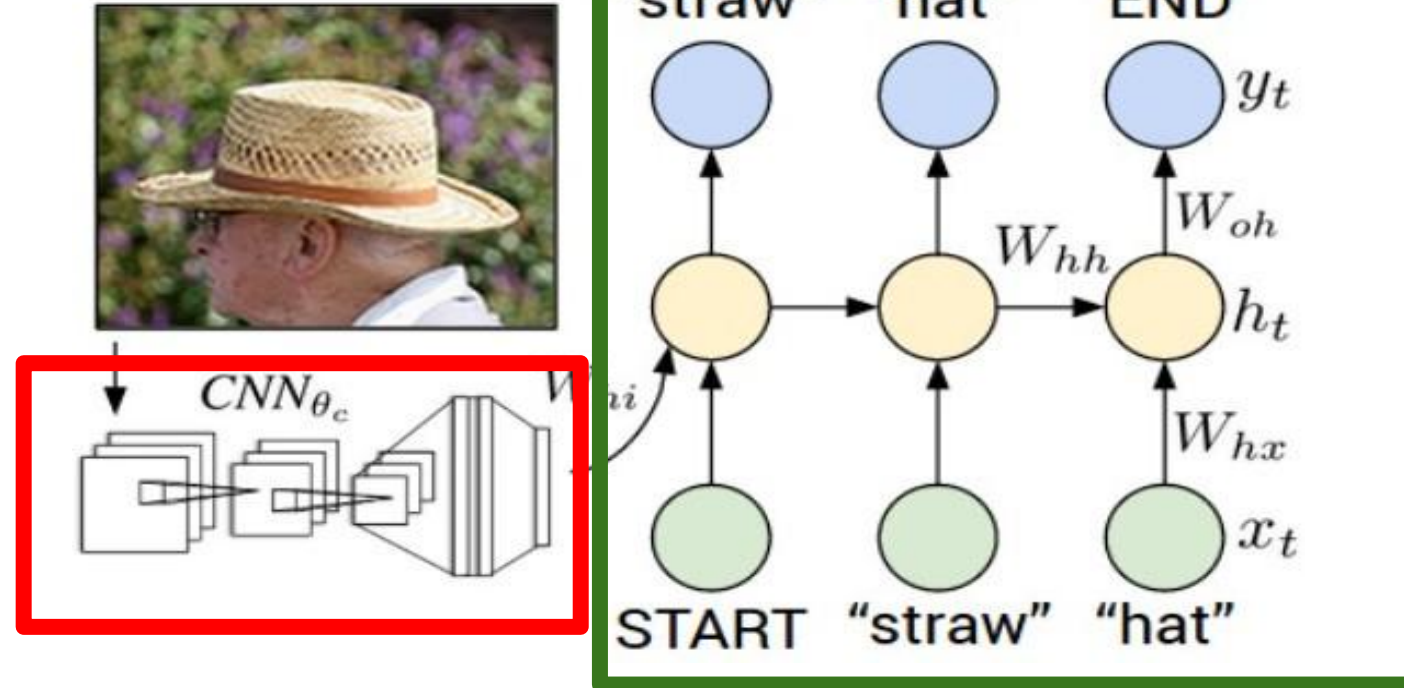
VIOLA:

Why, Salisbury must find his flesh and thought  
That which I am not apt, not a man and in fire,  
To show the reining of the raven and the wars  
To grace my hand reproach within, and not a fair are hand,  
That Caesar and my goodly father's world;  
When I was heaven of presence and our fleets,  
We spare with hours, but cut thy council I am great,  
Murdered and by thy master's ready there  
My power to give thee but so much as hell:  
Some service in the noble bondman here,  
Would show him to her wine.

KING LEAR:

O, if you were a feeble sight, the courtesy of your law,  
Your sight and several breath, will wear the gods  
With his heads, and my hands are wonder'd at the deeds,  
So drop upon your lordship's head, and your opinion  
Shall be against your honour.

# Example: Image Captioning



**Recurrent  
Neural  
Network**

**Convolutional Neural Network**

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096

FC-1000

softmax



**Transfer learning:** Take  
CNN trained on ImageNet,  
chop off last layer

X



image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096



x0

<START>

image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096



**before:**

$$h = \tanh(W_{xh} * x + W_{hh} * h)$$

**now:**

$$h = \tanh(W_{xh} * x + W_{hh} * h + W_{ih} * v)$$

y0

h0

x0

<START>

$W_{ih}$

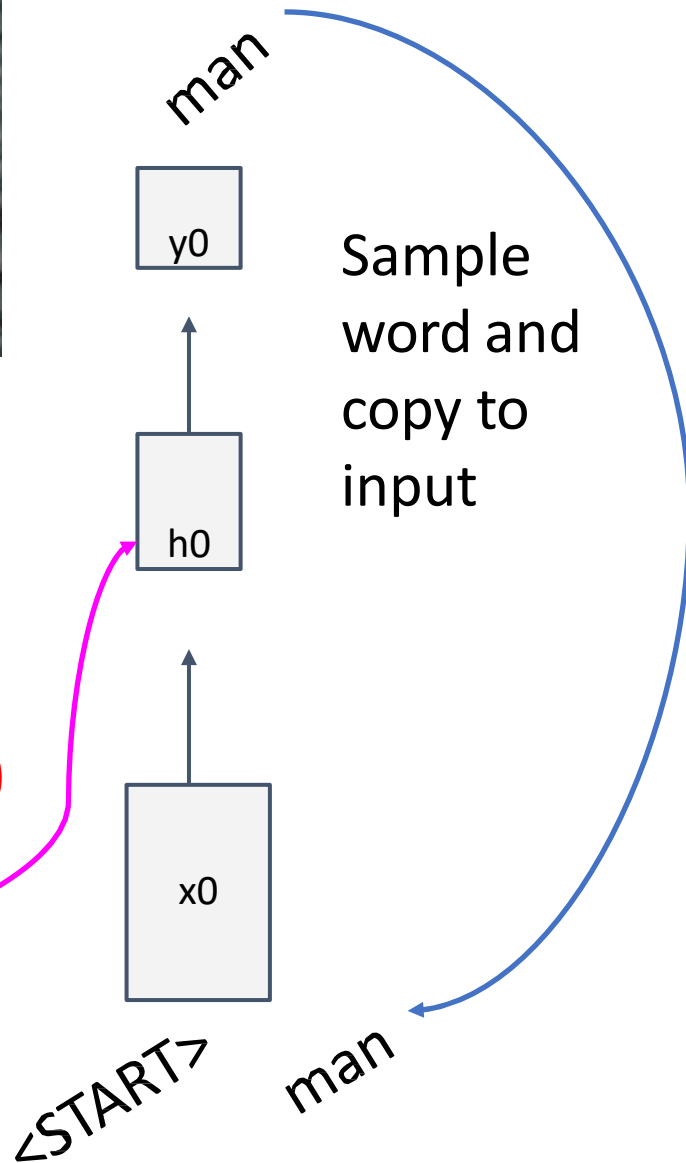


**before:**

$$h = \tanh(W_{xh} * x + W_{hh} * h)$$

**now:**

$$h = \tanh(W_{xh} * x + W_{hh} * h + W_{ih} * v)$$





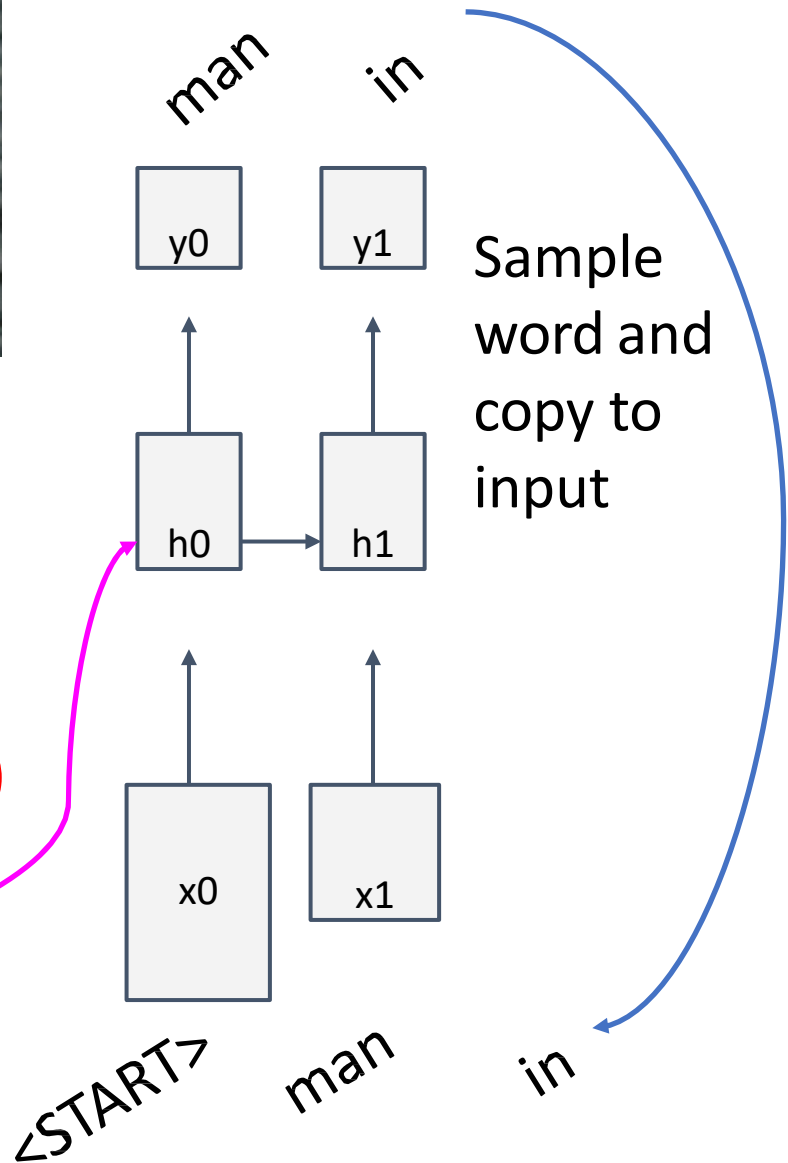
**before:**

$$h = \tanh(W_{xh} * x + W_{hh} * h)$$

**now:**

$$h = \tanh(W_{xh} * x + W_{hh} * h + W_{ih} * v)$$

$W_{ih}$







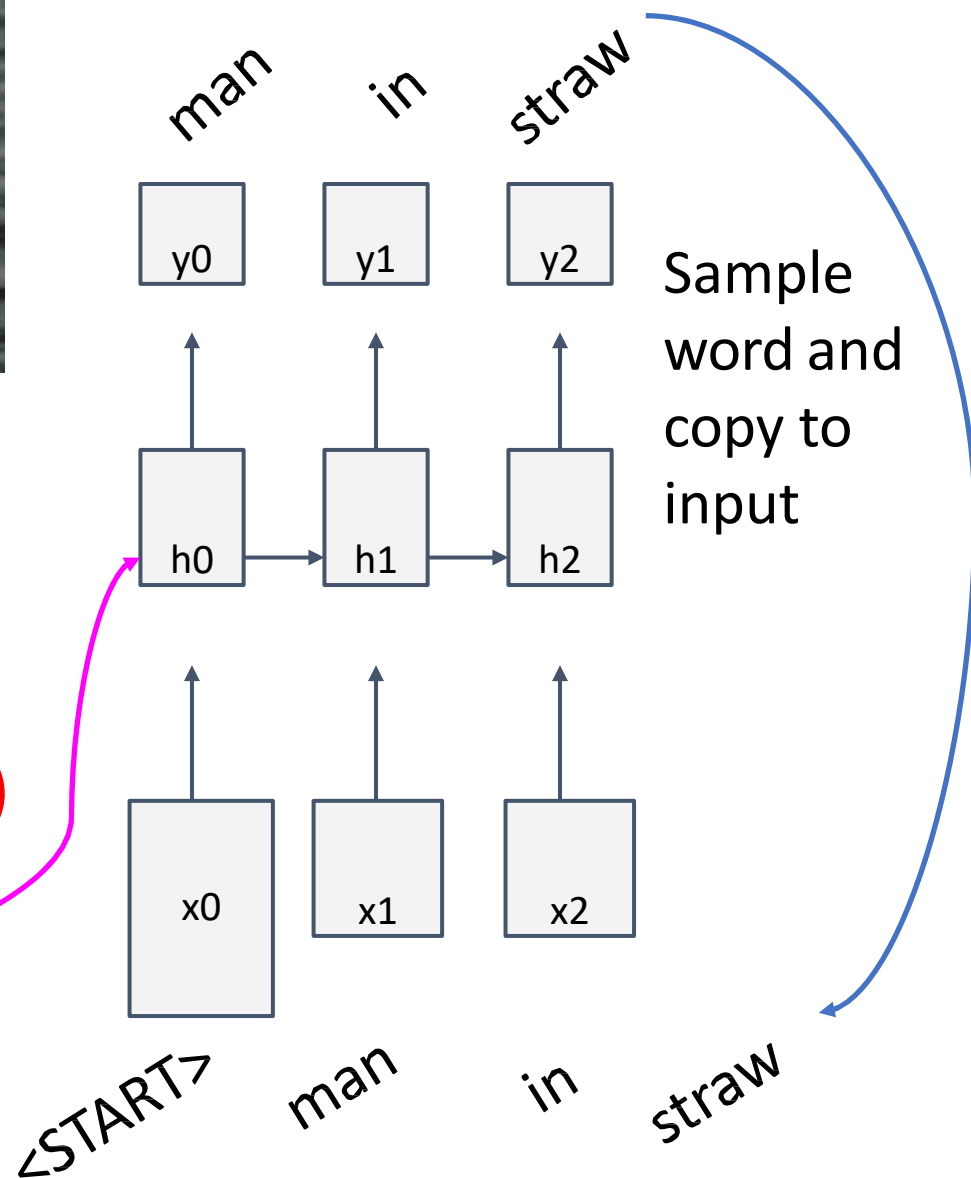
**before:**

$$h = \tanh(W_{xh} * x + W_{hh} * h)$$

**now:**

$$h = \tanh(W_{xh} * x + W_{hh} * h + W_{ih} * v)$$

$W_{ih}$





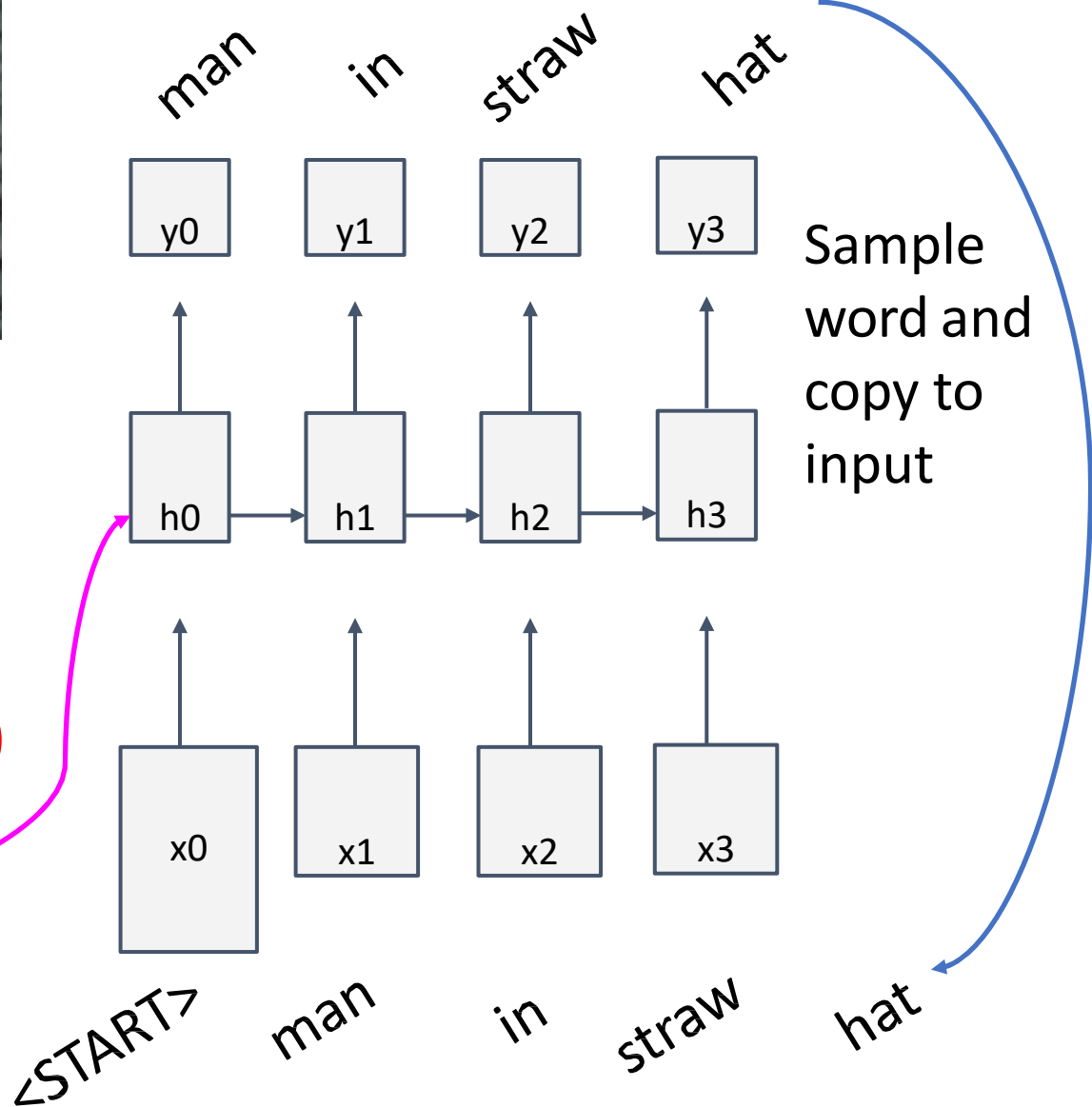
**before:**

$$h = \tanh(W_{xh} * x + W_{hh} * h)$$

**now:**

$$h = \tanh(W_{xh} * x + W_{hh} * h + W_{ih} * v)$$

$W_{ih}$





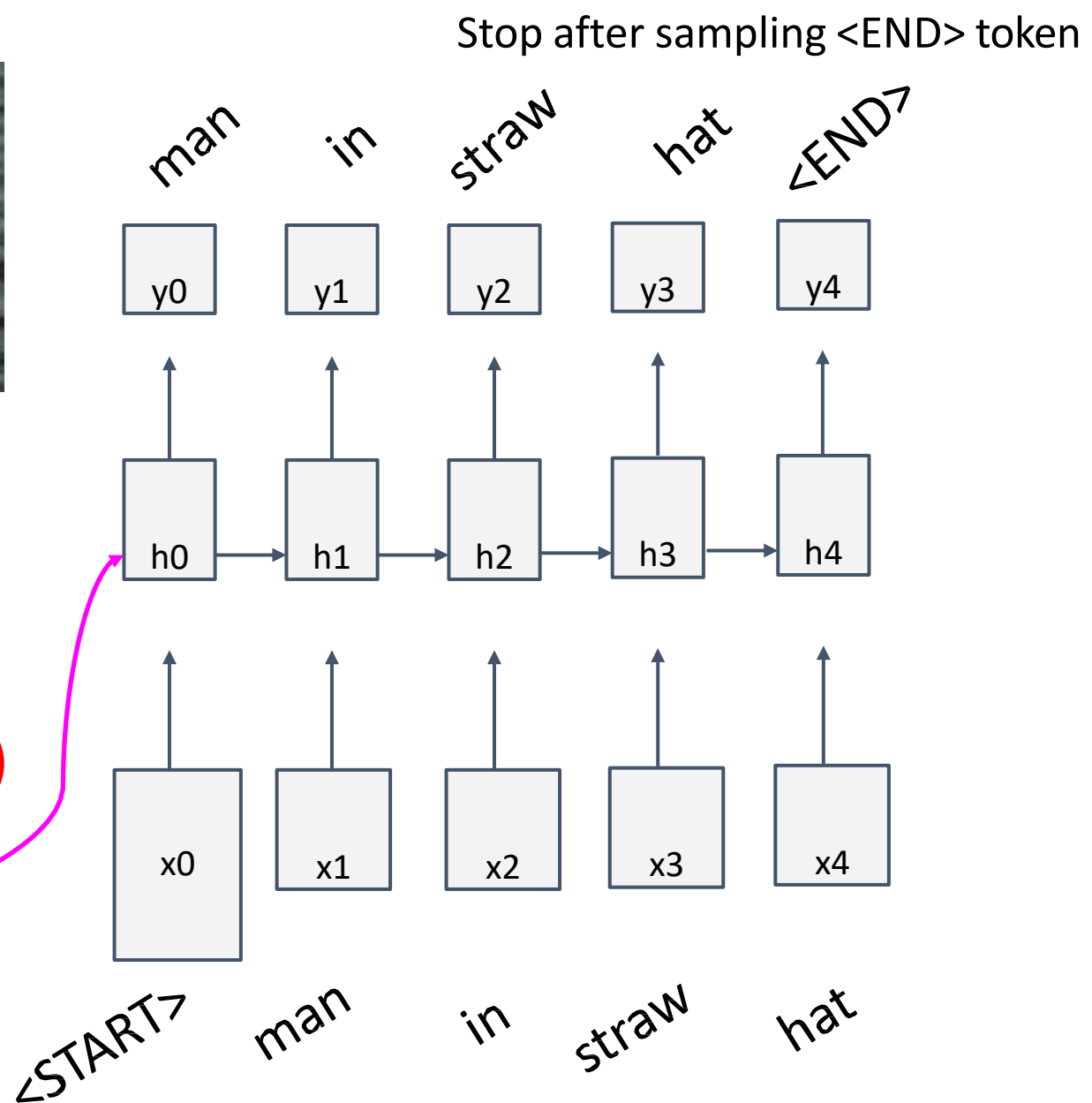
**before:**

$$h = \tanh(W_{xh} * x + W_{hh} * h)$$

**now:**

$$h = \tanh(W_{xh} * x + W_{hh} * h + W_{ih} * v)$$

$W_{ih}$





# Image Captioning: Example Results



*A cat sitting on a suitcase on the floor*



*A cat is sitting on a tree branch*



*A dog is running in the grass with a frisbee*



*A white teddy bear sitting in the grass*



*Two people walking on the beach with surfboards*



*A tennis player in action on the court*



*Two giraffes standing in a grassy field*



*A man riding a dirt bike on a dirt track*



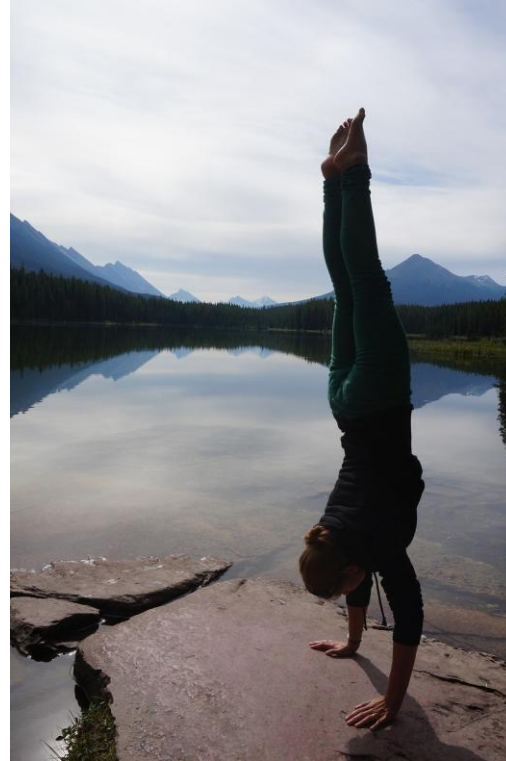
# Image Captioning: Failure Cases



*A woman is holding a cat  
in her hand*



*A person holding a computer  
mouse on a desk*



*A woman standing on a beach  
holding a surfboard*



*A bird is perched on a  
tree branch*



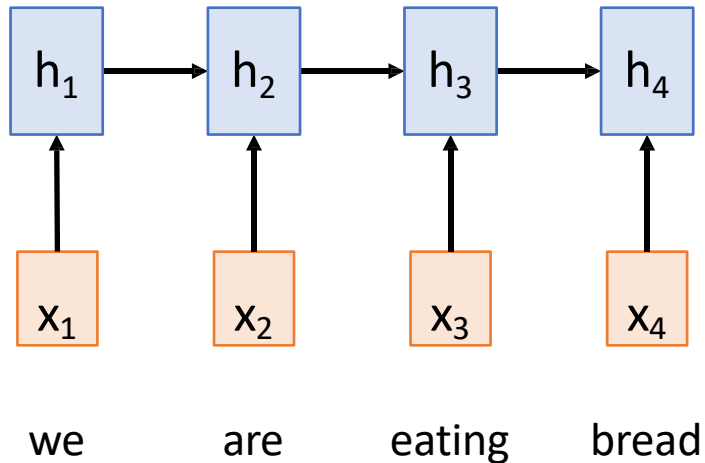
*A man in a  
baseball uniform  
throwing a ball*

# Sequence-to-Sequence with RNNs

**Input:** Sequence  $x_1, \dots, x_T$

**Output:** Sequence  $y_1, \dots, y_{T'}$

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



# Sequence-to-Sequence with RNNs

**Input:** Sequence  $x_1, \dots, x_T$

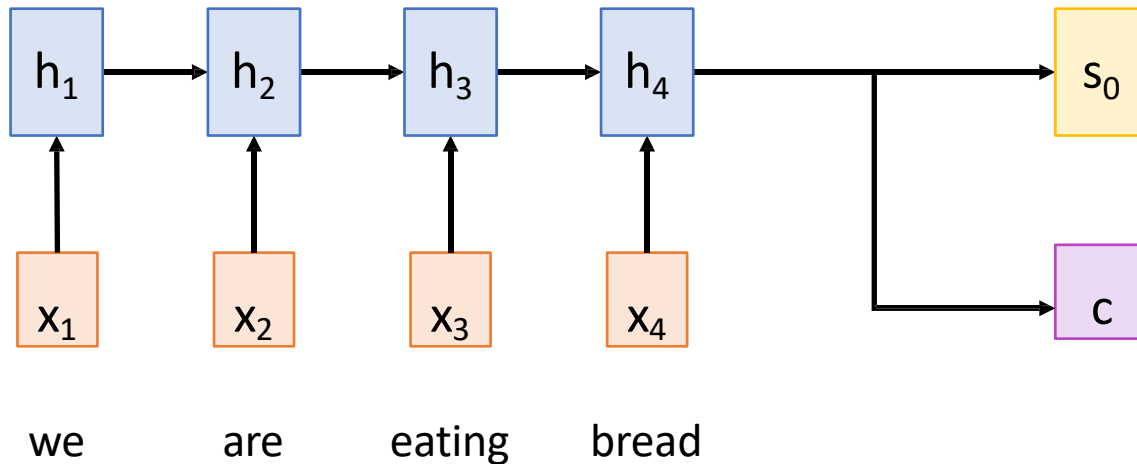
**Output:** Sequence  $y_1, \dots, y_{T'}$

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

From final hidden state predict:

**Initial decoder state**  $s_0$

**Context vector**  $c$  (often  $c=h_T$ )



# Sequence-to-Sequence with RNNs

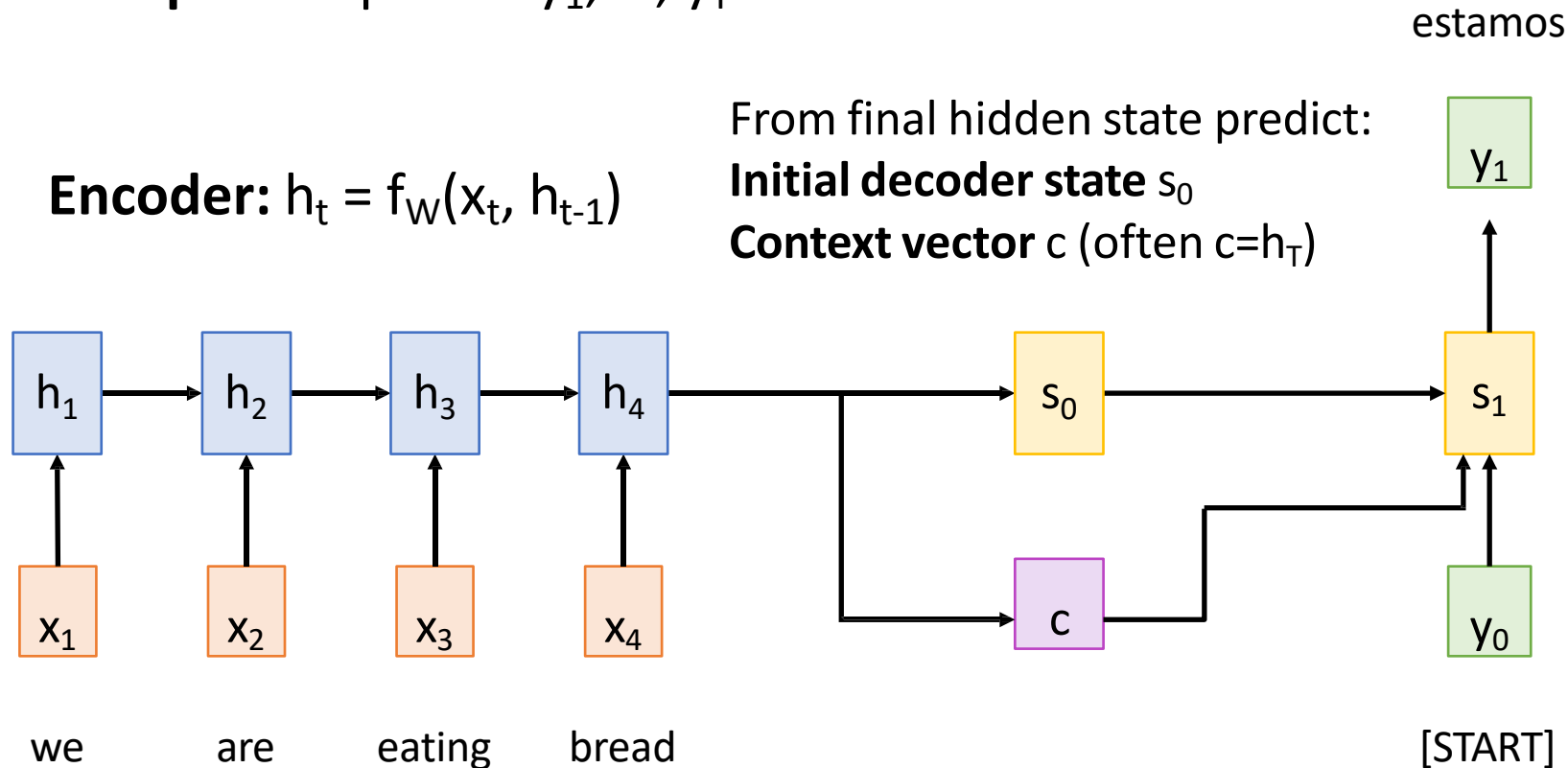
**Input:** Sequence  $x_1, \dots, x_T$

**Output:** Sequence  $y_1, \dots, y_{T'}$

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

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

From final hidden state predict:  
**Initial decoder state**  $s_0$   
**Context vector**  $c$  (often  $c=h_T$ )





# Sequence-to-Sequence with RNNs

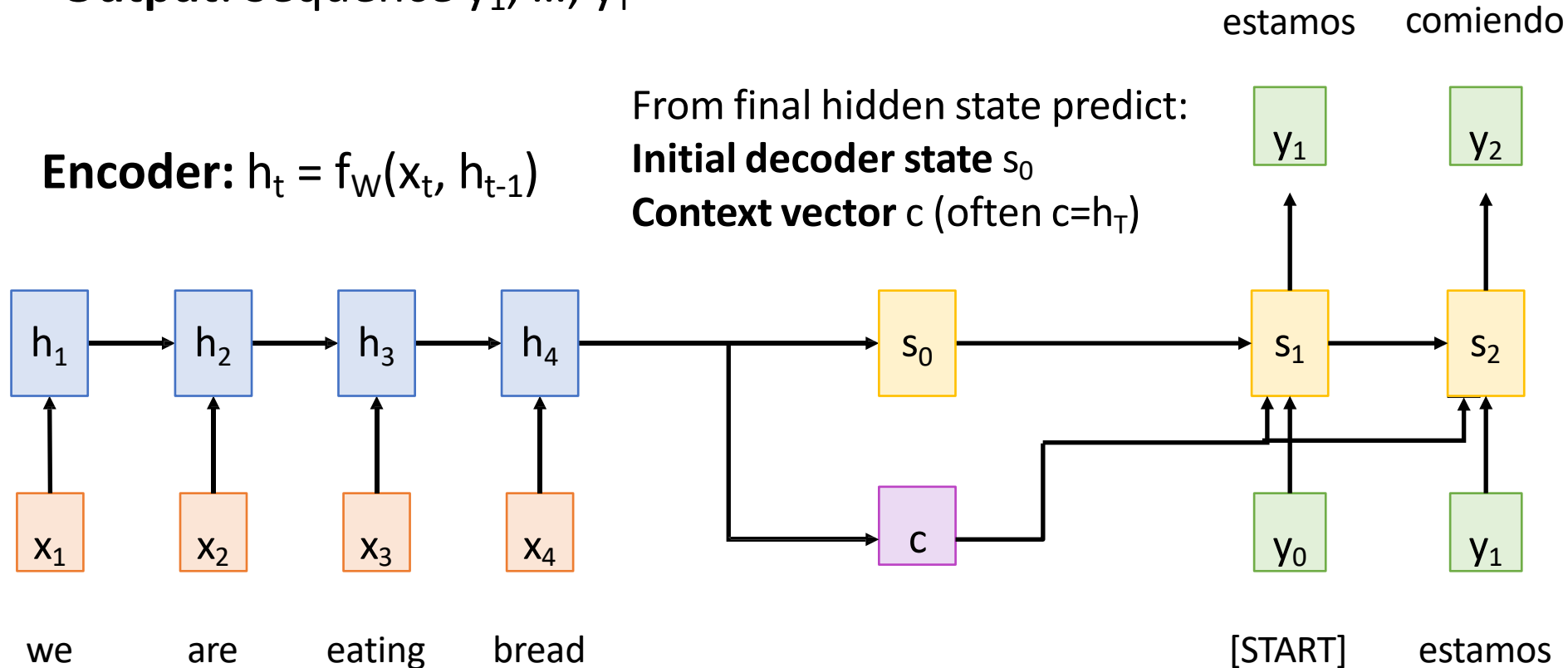
**Input:** Sequence  $x_1, \dots, x_T$

**Output:** Sequence  $y_1, \dots, y_{T'}$

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

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

From final hidden state predict:  
**Initial decoder state**  $s_0$   
**Context vector**  $c$  (often  $c=h_T$ )



# Sequence-to-Sequence with RNNs

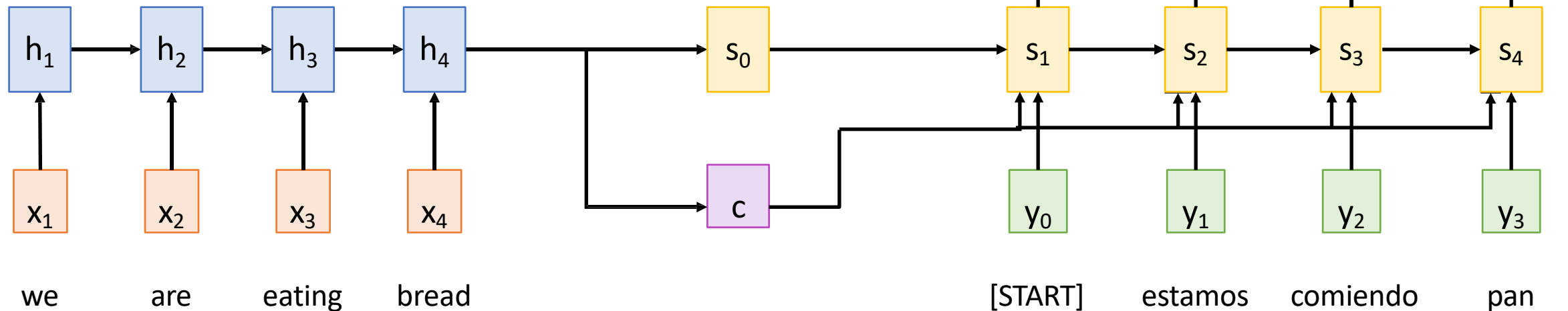
**Input:** Sequence  $x_1, \dots, x_T$

**Output:** Sequence  $y_1, \dots, y_{T'}$

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

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

From final hidden state predict:  
**Initial decoder state**  $s_0$   
**Context vector**  $c$  (often  $c=h_T$ )



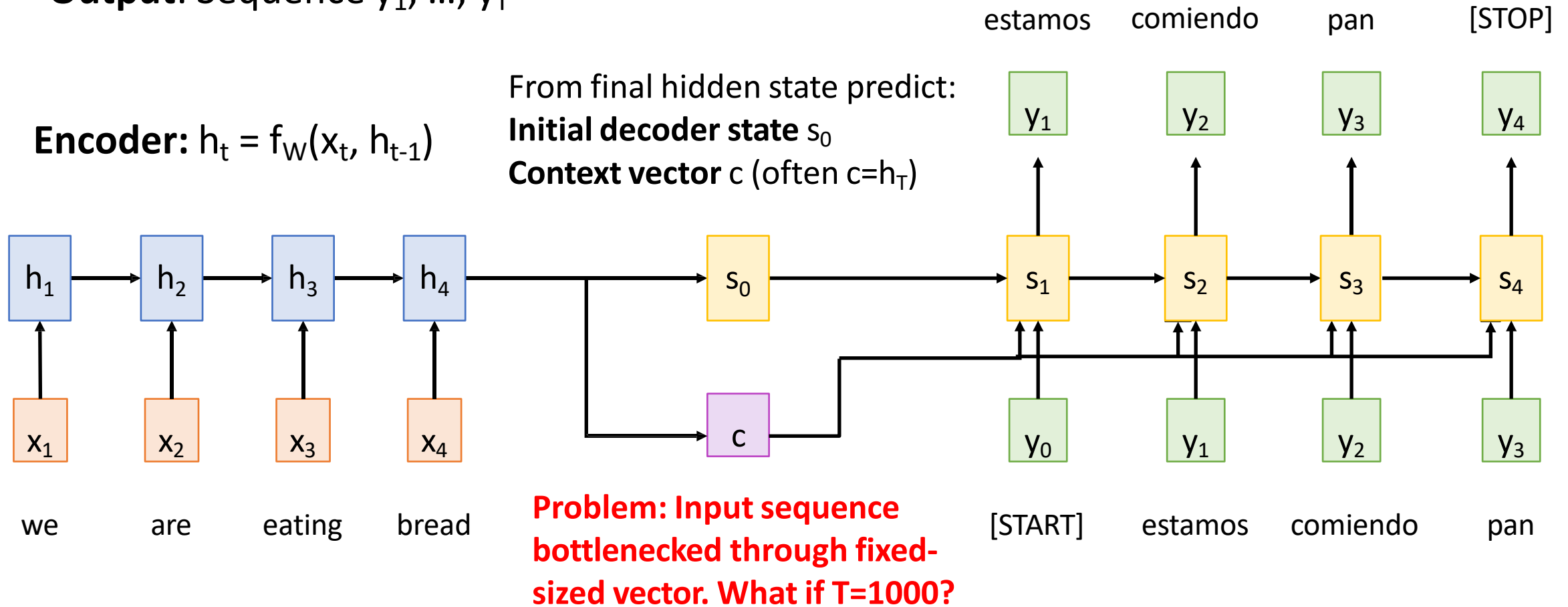
# Sequence-to-Sequence with RNNs

**Input:** Sequence  $x_1, \dots, x_T$

**Output:** Sequence  $y_1, \dots, y_{T'}$

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

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



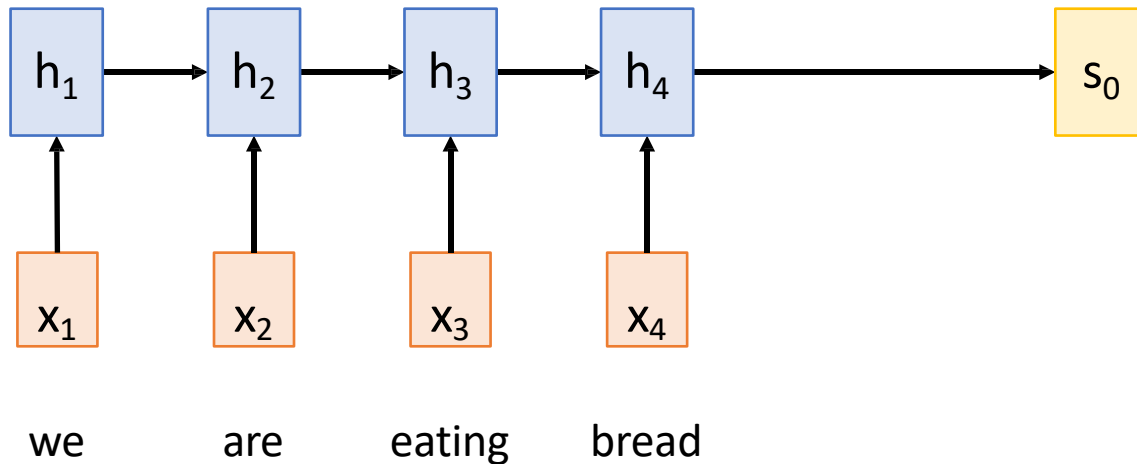
# Sequence-to-Sequence with RNNs and Attention

**Input:** Sequence  $x_1, \dots, x_T$

**Output:** Sequence  $y_1, \dots, y_{T'}$

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

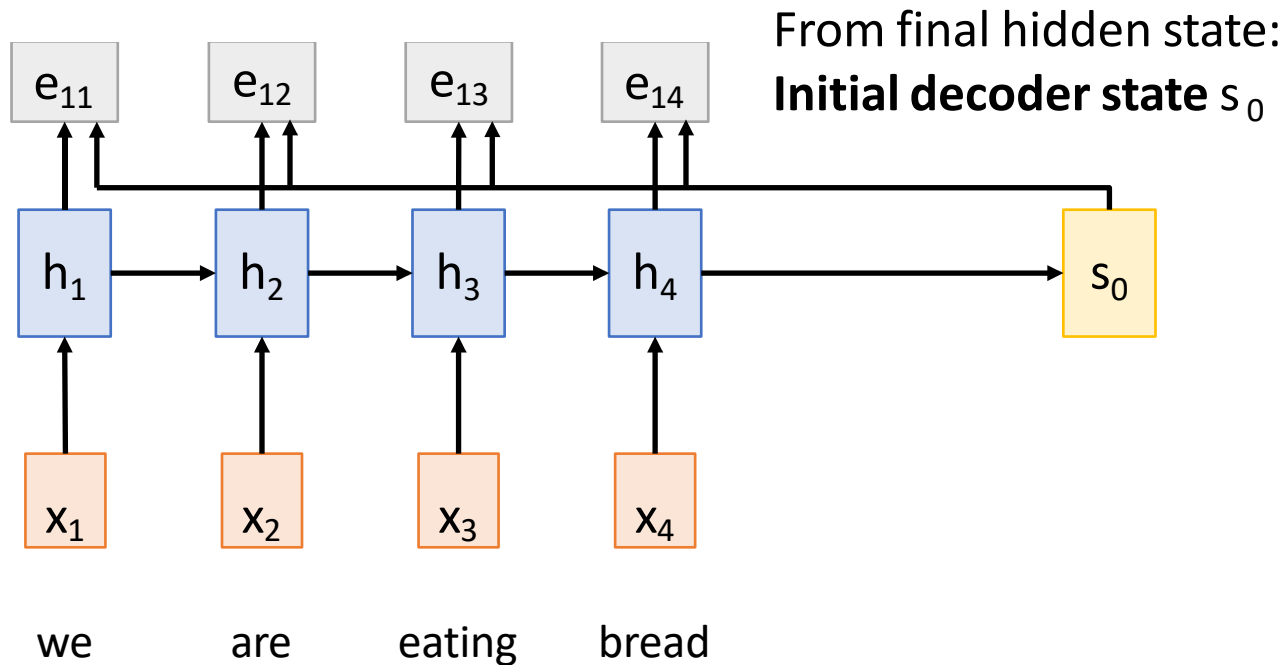
From final hidden state:  
**Initial decoder state**  $s_0$



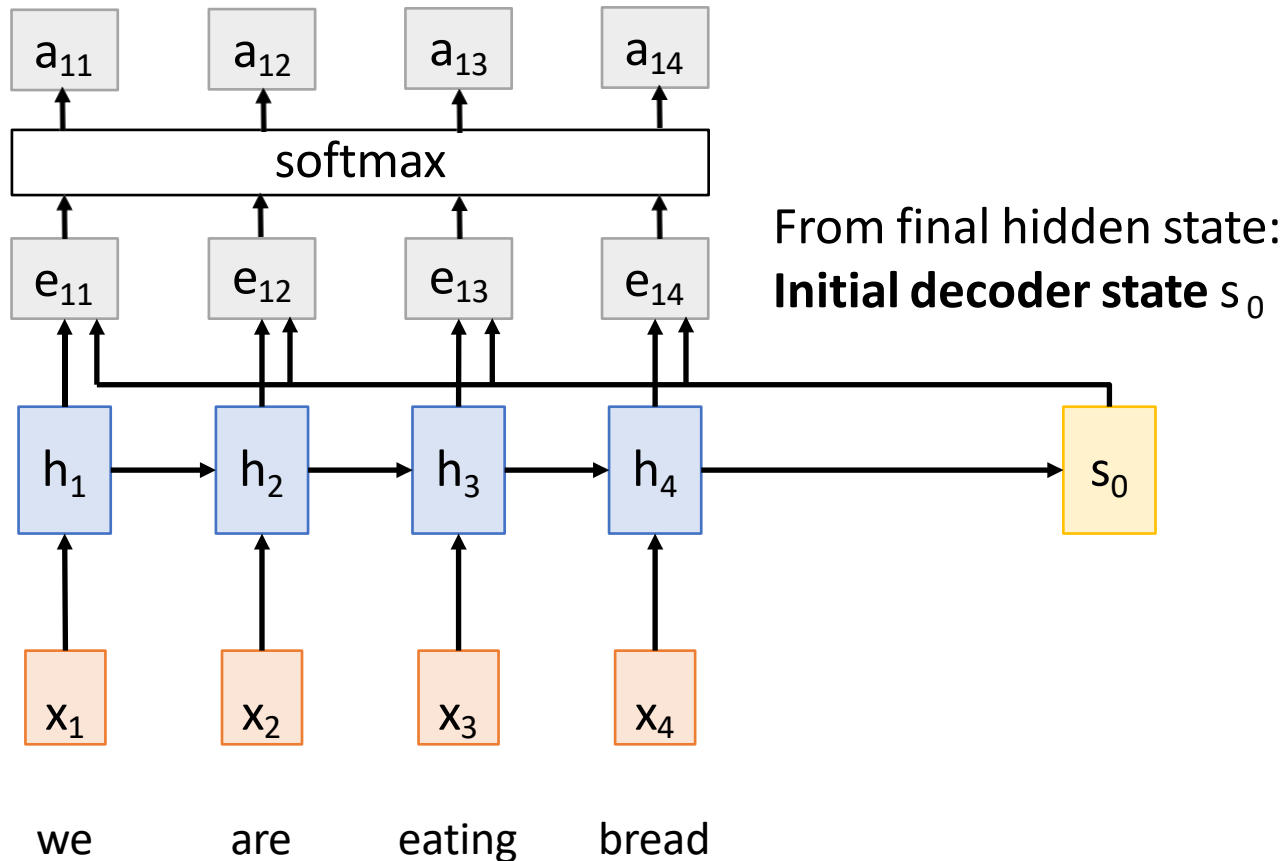
# Sequence-to-Sequence with RNNs and Attention

Compute (scalar) **alignment scores**

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



# Sequence-to-Sequence with RNNs and Attention



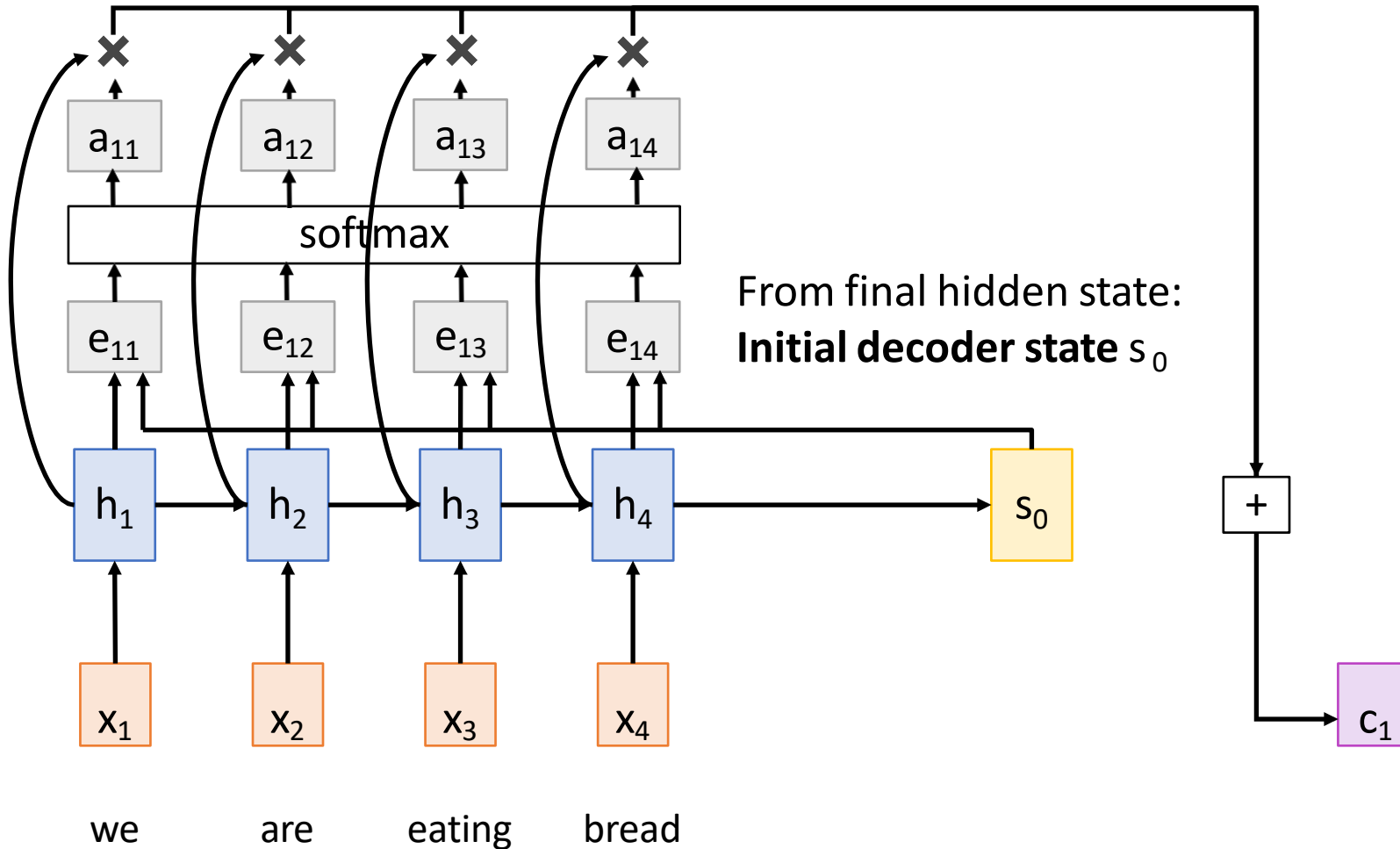
Compute (scalar) **alignment scores**

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

Normalize alignment scores  
to get **attention weights**

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

# Sequence-to-Sequence with RNNs and Attention

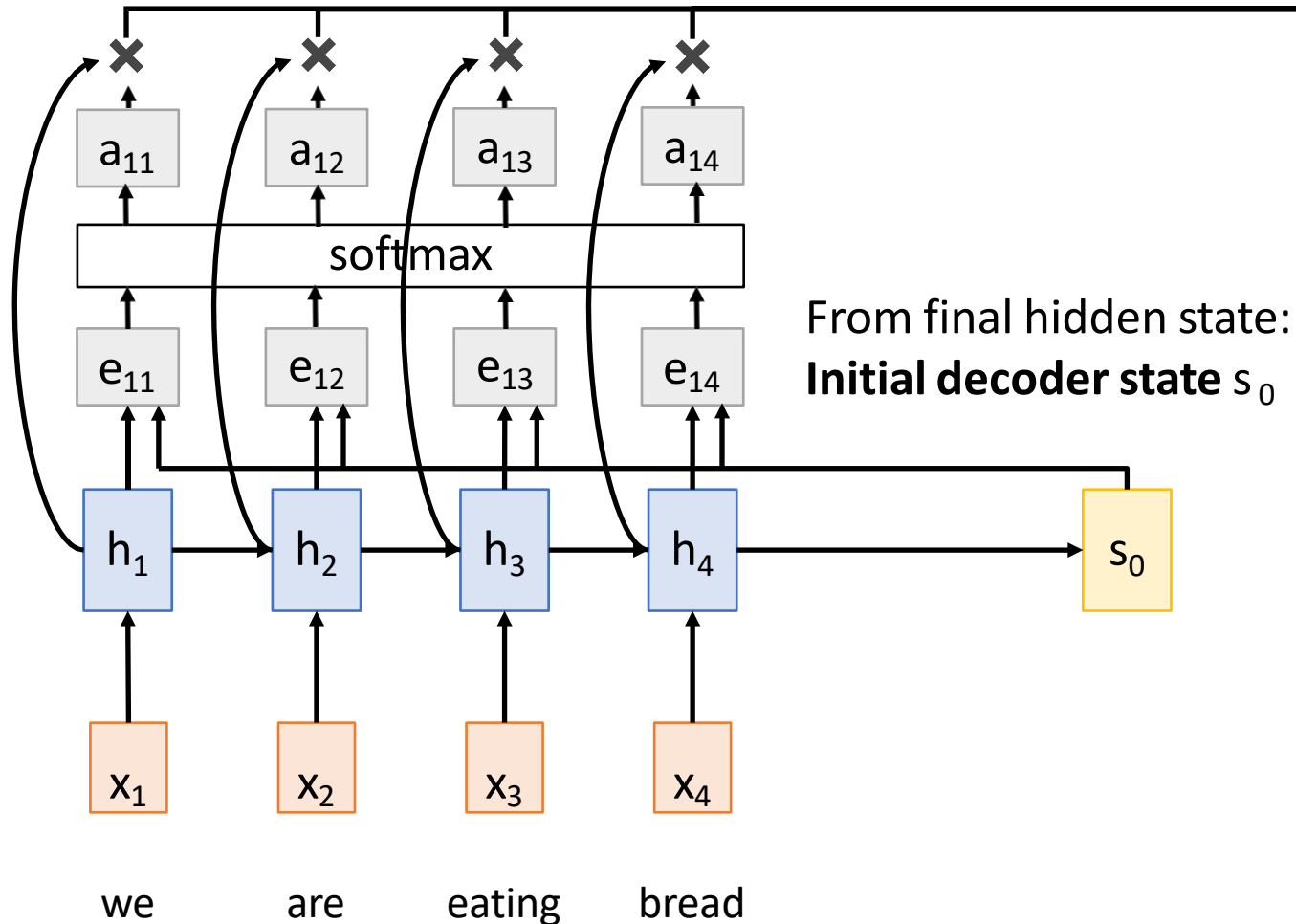


Compute (scalar) **alignment scores**  
 $e_{t,i} = f_{\text{att}}(s_{t-1}, h_i)$  ( $f_{\text{att}}$  is an MLP)

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

Compute context vector as linear  
combination of hidden states  
 $c_t = \sum_i a_{t,i} h_i$

# Sequence-to-Sequence with RNNs and Attention



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

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

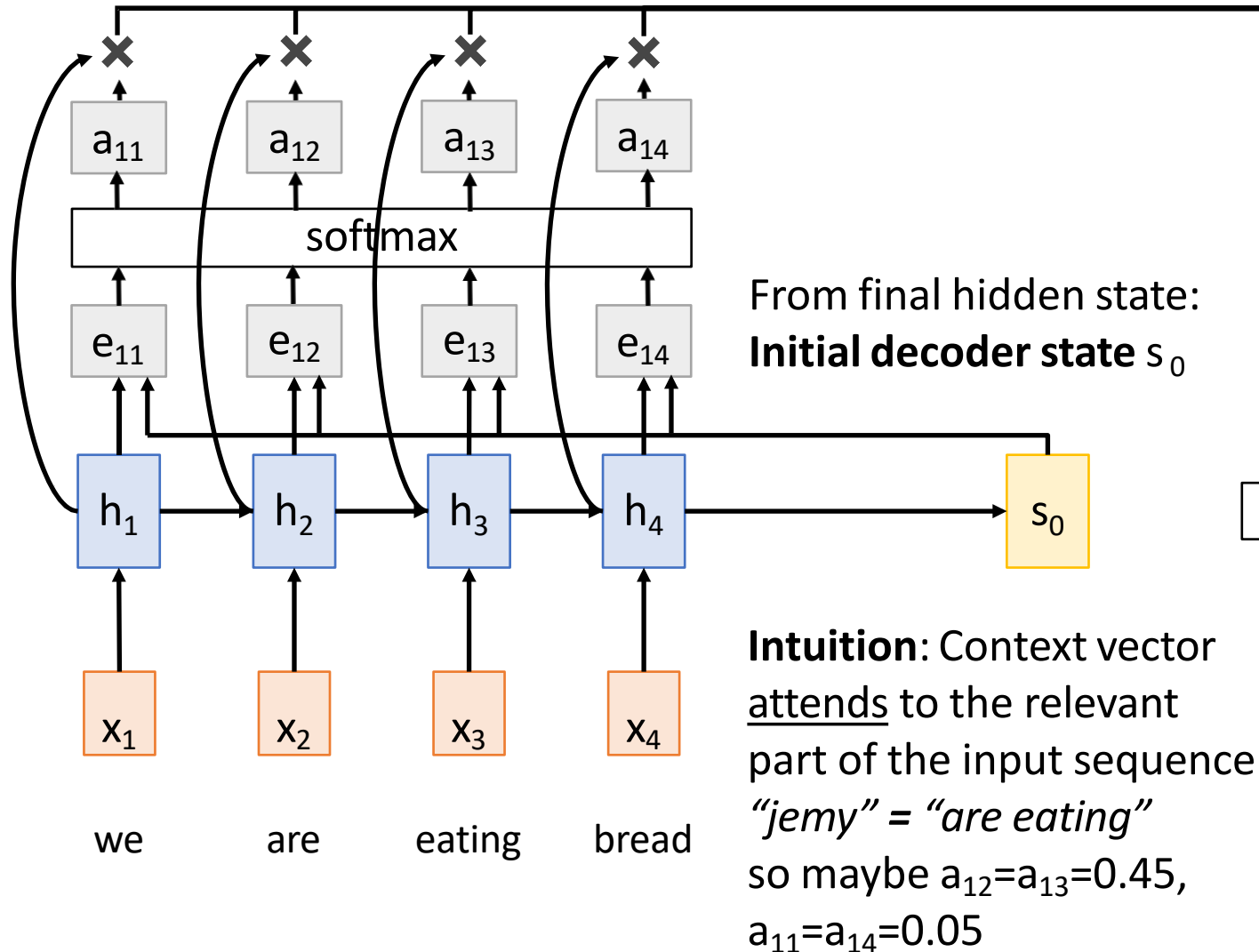
Compute context vector as linear  
combination of hidden states  
 $c_t = \sum_i a_{t,i} h_i$

Use context vector in  
decoder:  $s_t = g_U(y_{t-1}, s_{t-1}, c_t)$

**This is all differentiable.**



# Sequence-to-Sequence with RNNs and Attention



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

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

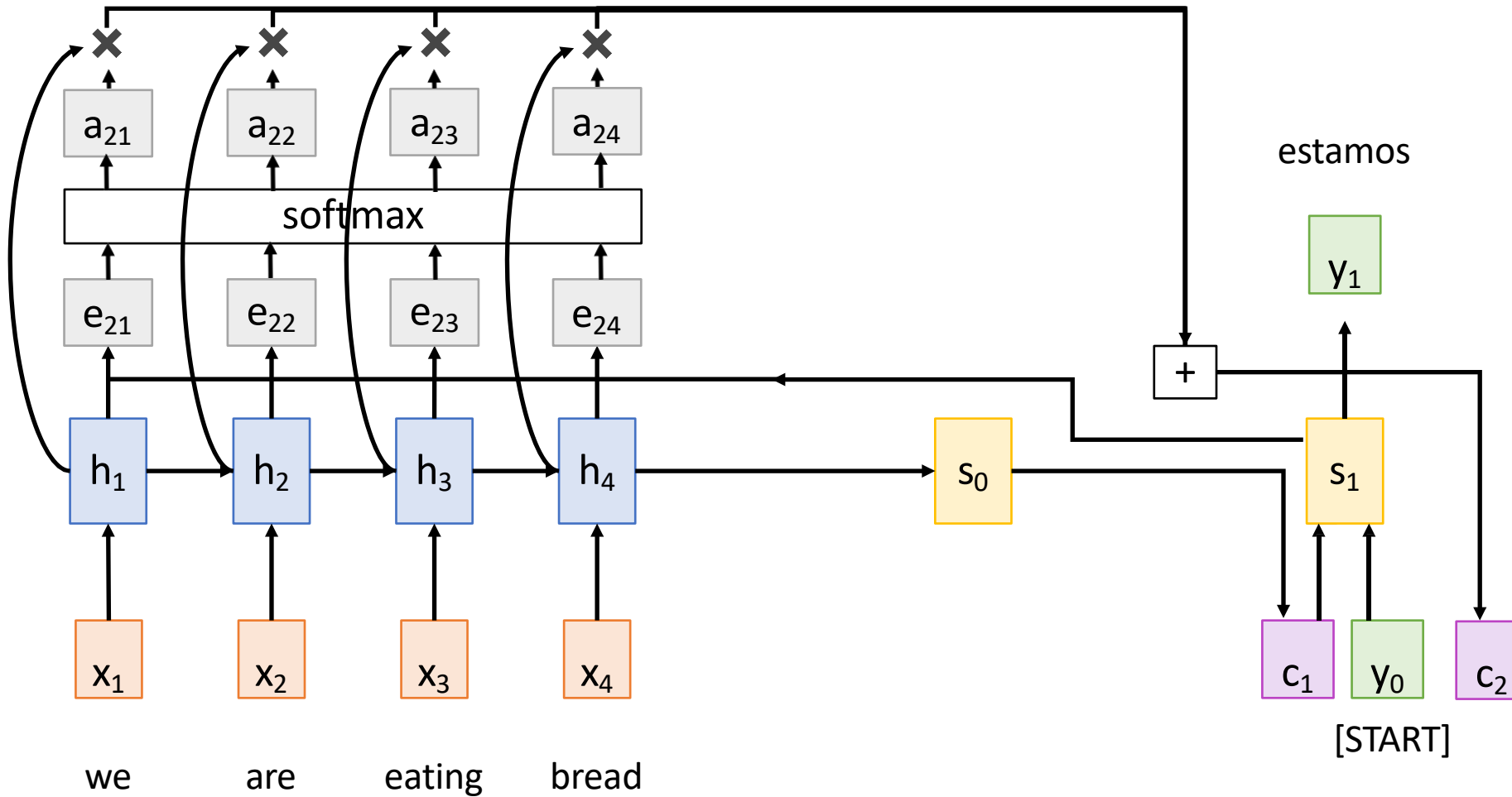
Compute context vector as linear  
combination of hidden states  
 $c_t = \sum_i a_{t,i} h_i$

Use context vector in  
decoder:  $s_t = g_U(y_{t-1}, s_{t-1}, c_t)$

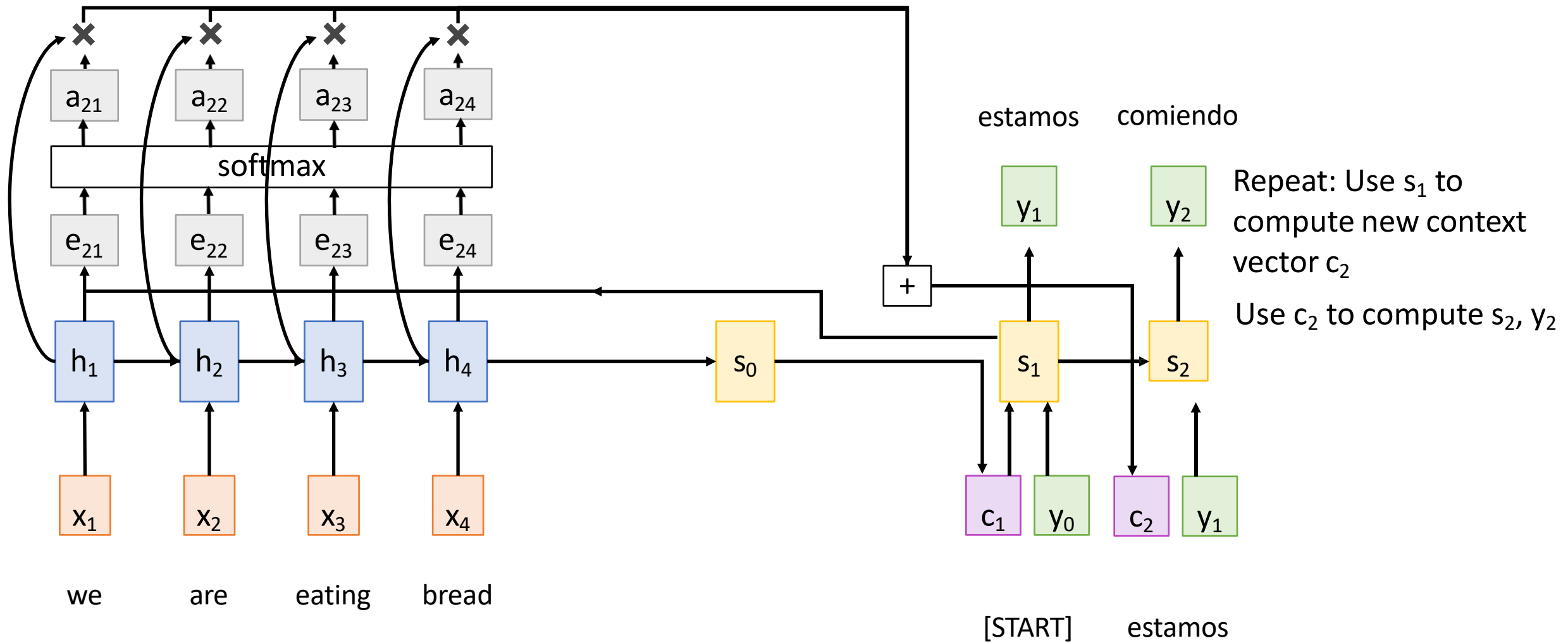
**This is all differentiable.**

# Sequence-to-Sequence with RNNs

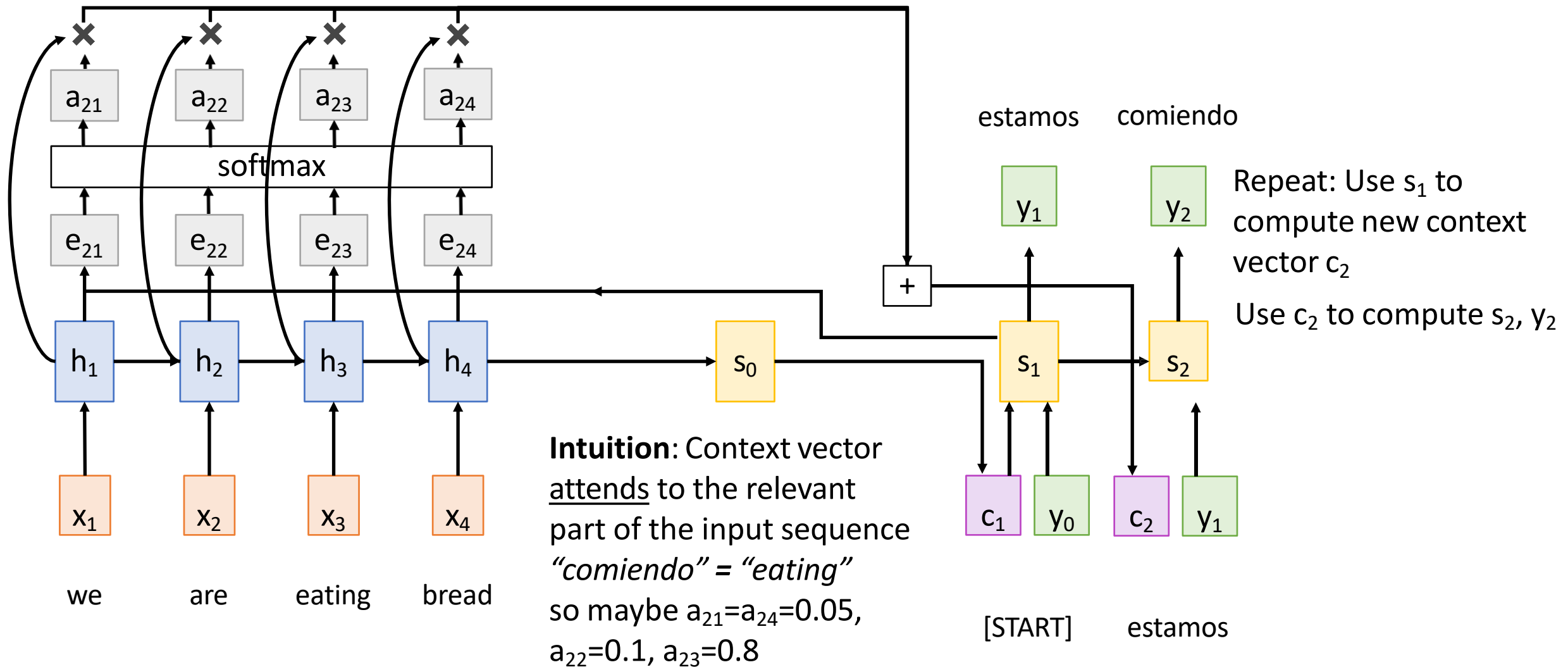
Repeat: Use  $s_1$  to compute new context vector  $c_2$



# Sequence-to-Sequence with RNNs and Attention



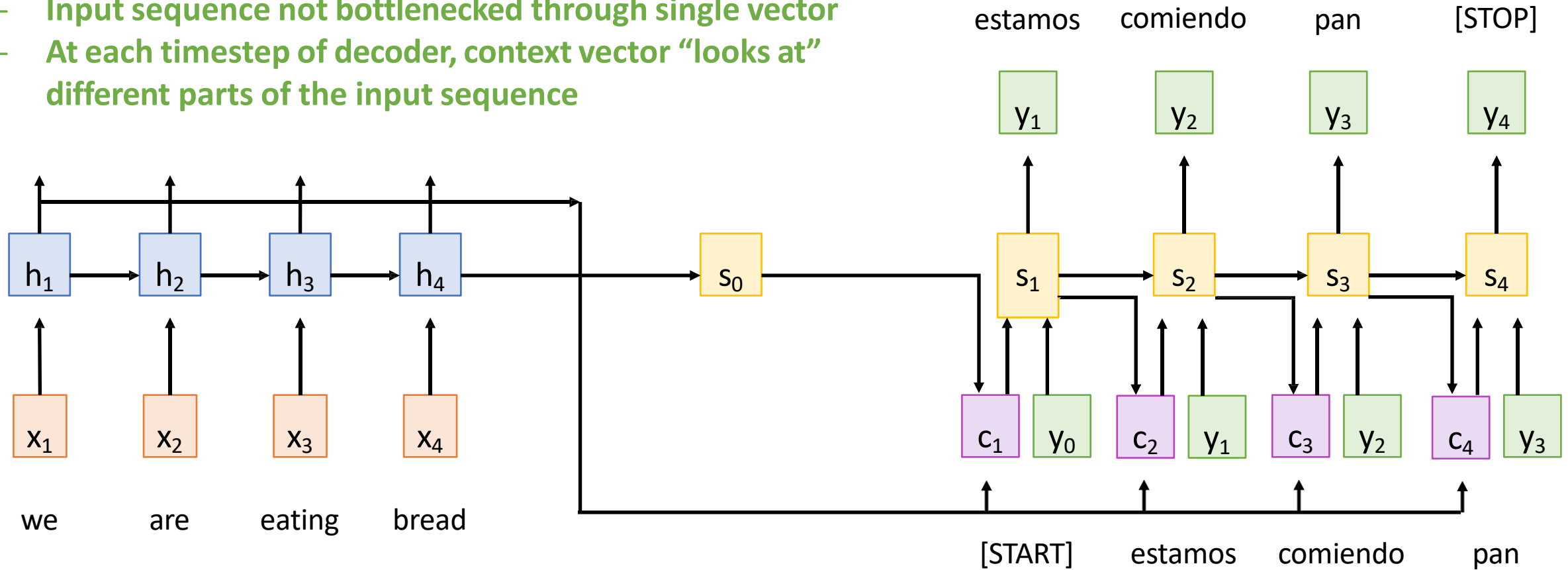
# Sequence-to-Sequence with RNNs and Attention



# Sequence-to-Sequence 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



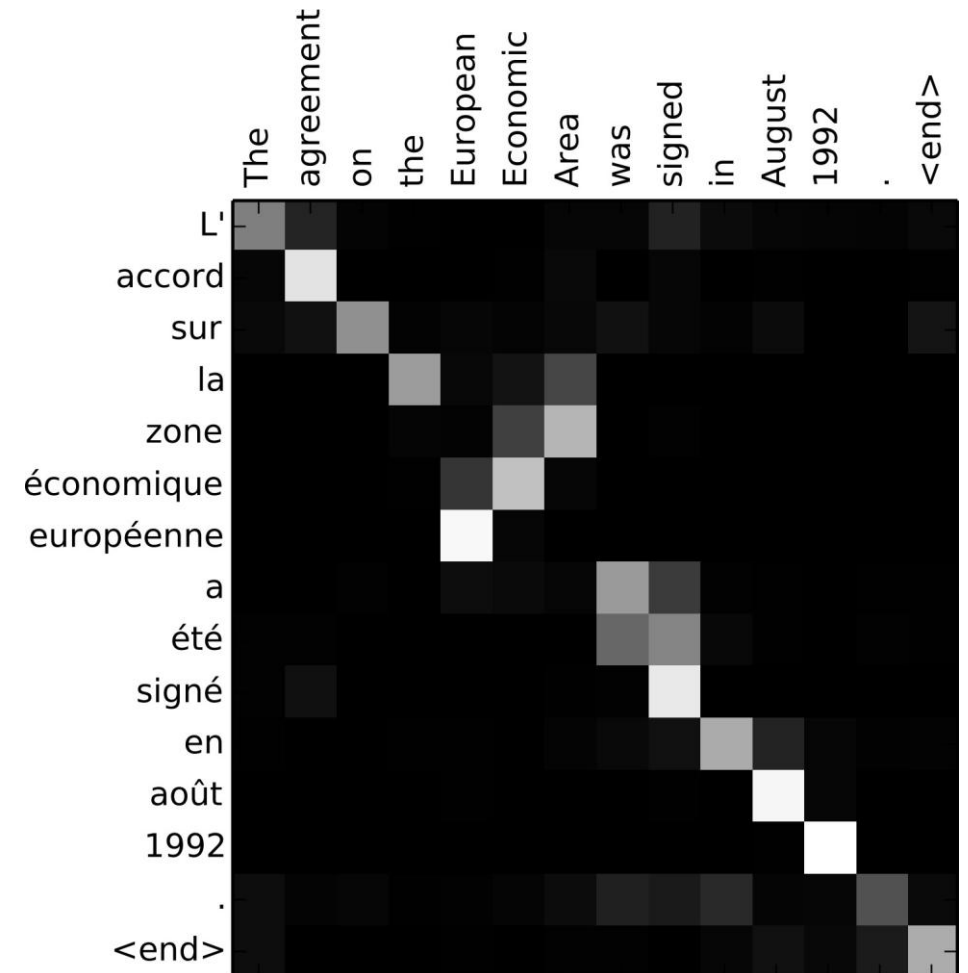
# Sequence-to-Sequence with RNNs and Attention

**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  $a_{t,i}$



# Sequence-to-Sequence with RNNs and Attention

**Example:** English to French translation

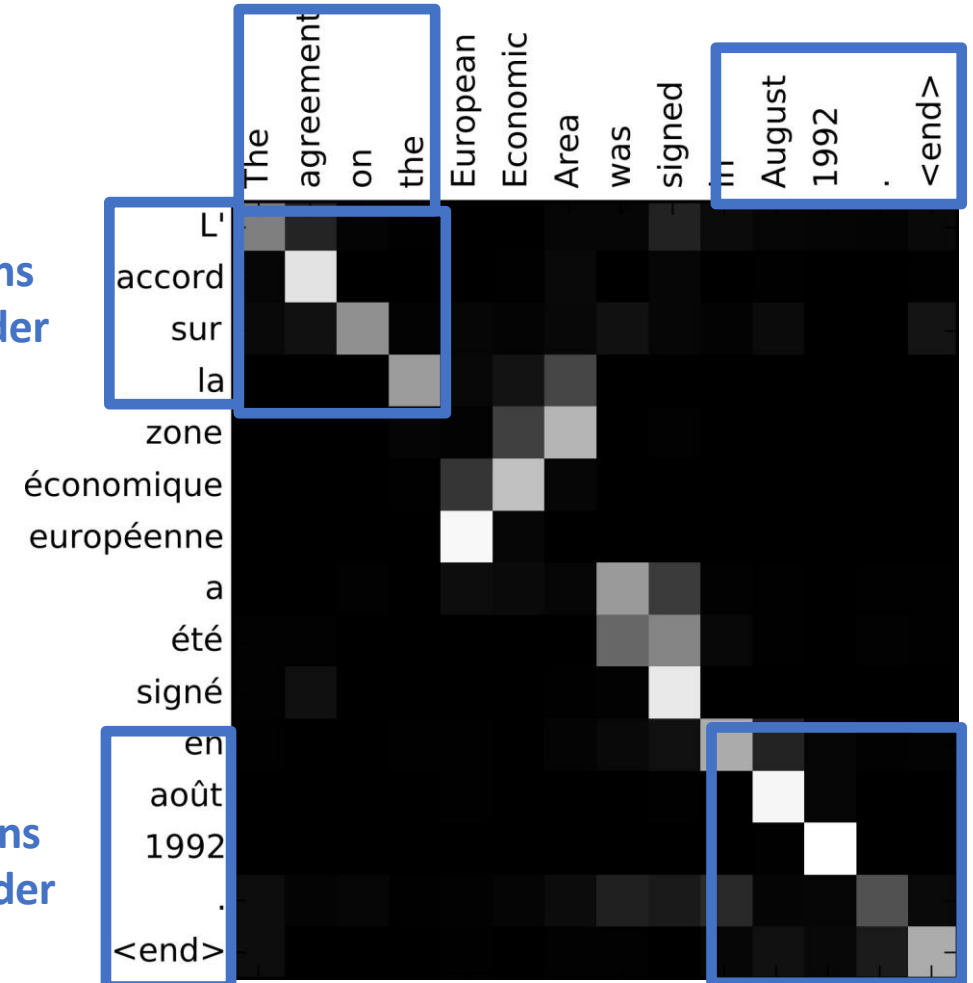
**Input:** “**The agreement on the** European Economic Area was signed **in August 1992.**”

**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  $a_{t,i}$



# Sequence-to-Sequence with RNNs and Attention

**Example:** English to French translation

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

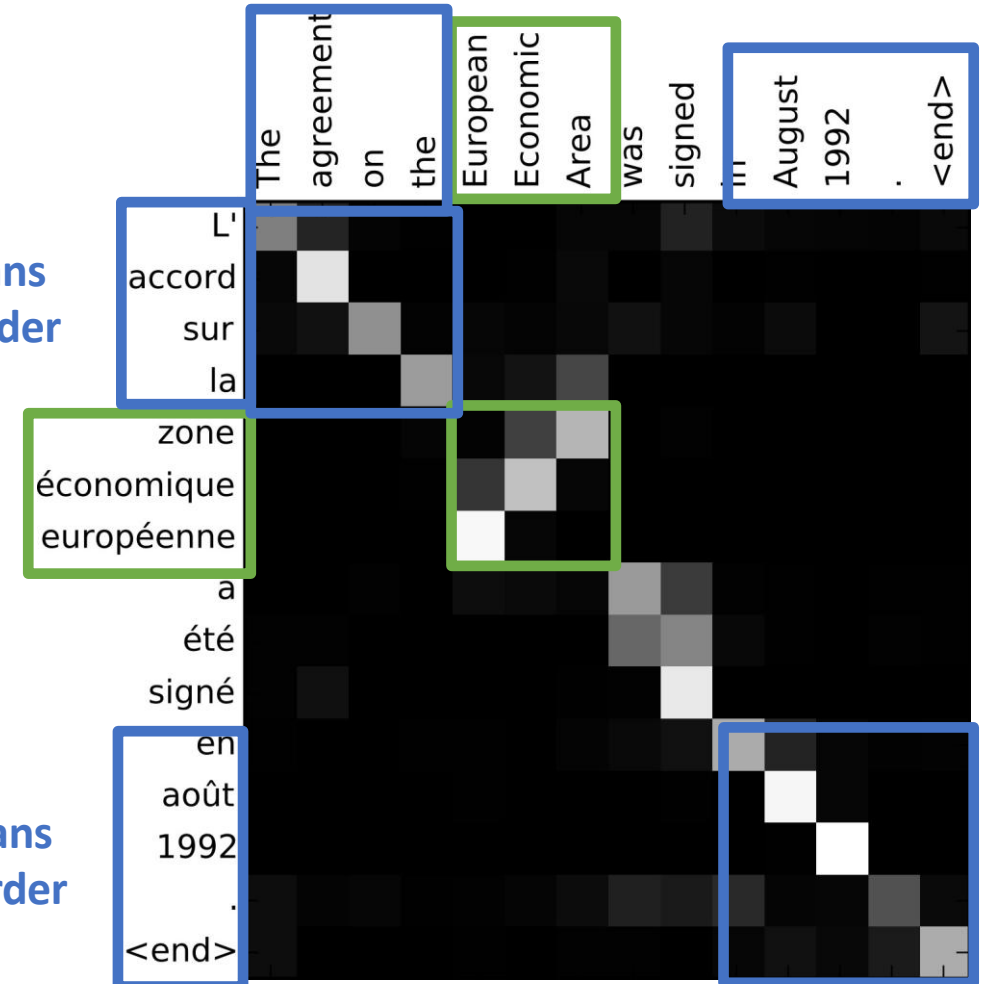
**Output:** “L’accord sur la zone économique européenne a été signé en août 1992.”

Diagonal attention means words correspond in order

Attention figures out different word orders

Diagonal attention means words correspond in order

Visualize attention weights  $a_{t,i}$





# Sequence-to-Sequence 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.”

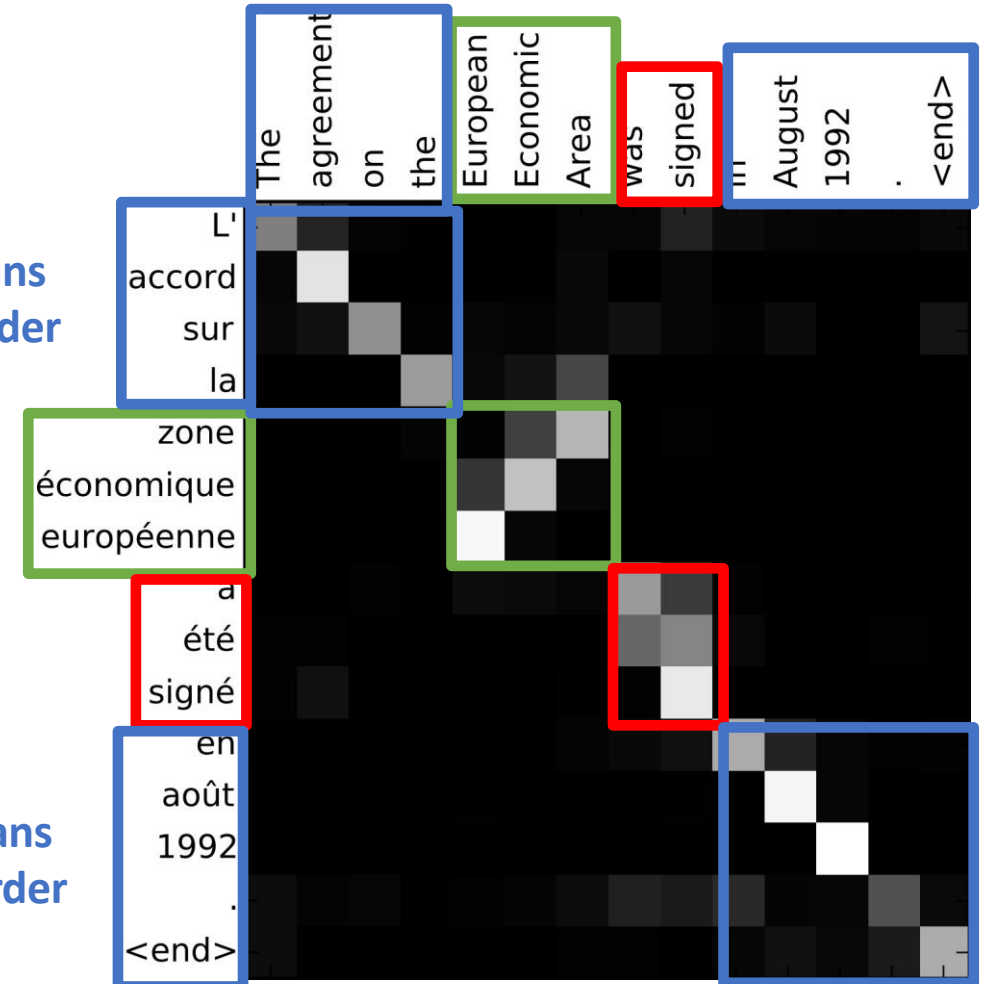
Diagonal attention means words correspond in order

Attention figures out different word orders

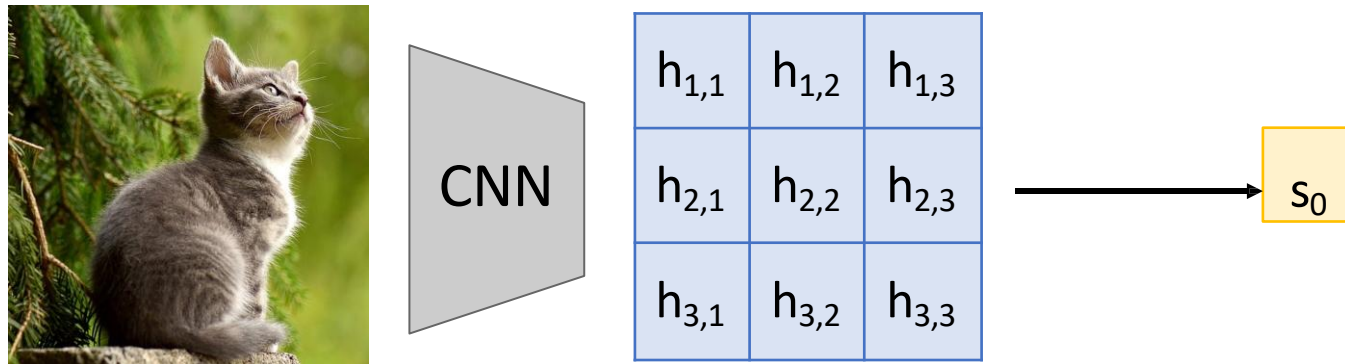
Verb conjugation

Diagonal attention means words correspond in order

Visualize attention weights  $a_{t,i}$



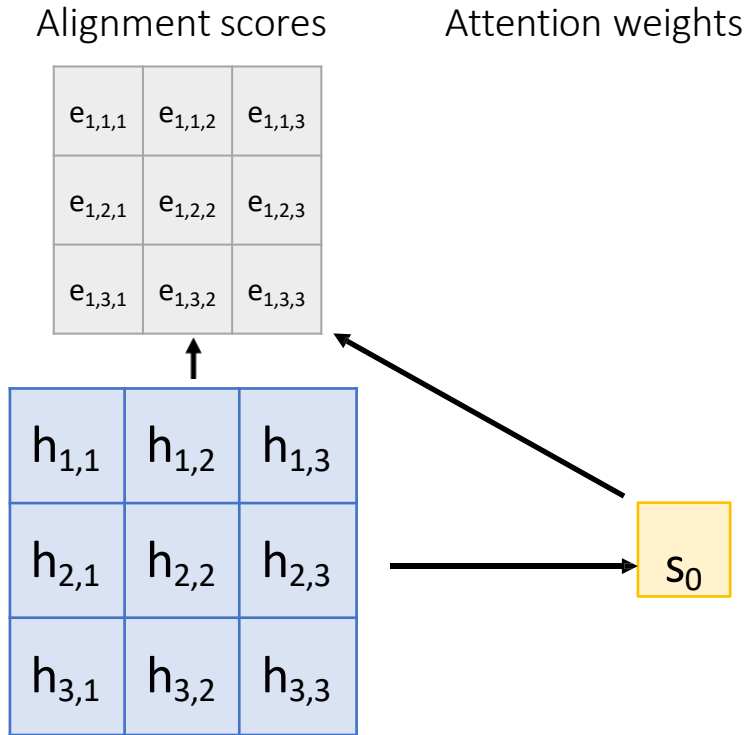
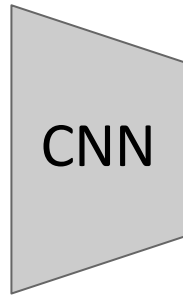
# Image Captioning with RNNs and Attention



Use a CNN to compute a  
grid of features for an image

# Image Captioning with RNNs and Attention

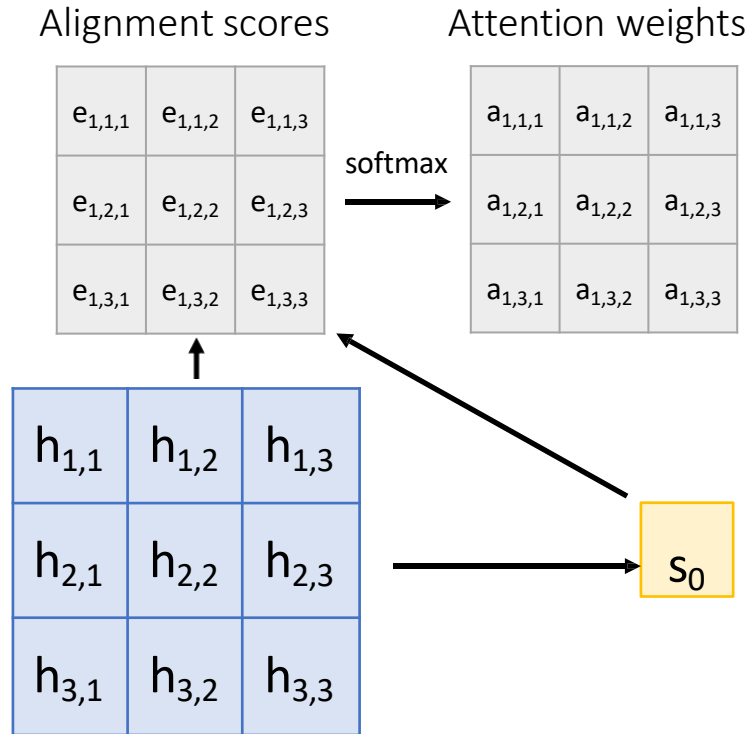
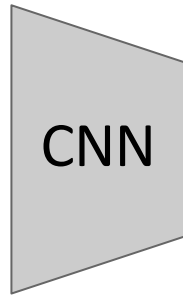
$$e_{t,i,j} = f_{\text{att}}(s_{t-1}, h_{i,j})$$



Use a CNN to compute a  
grid of features for an image

# Image Captioning with RNNs and Attention

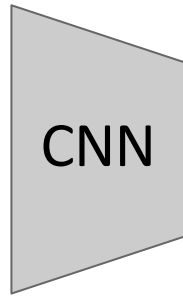
$$e_{t,i,j} = f_{\text{att}}(s_{t-1}, h_{i,j})$$
$$a_{t,:} = \text{softmax}(e_{t,:,:})$$



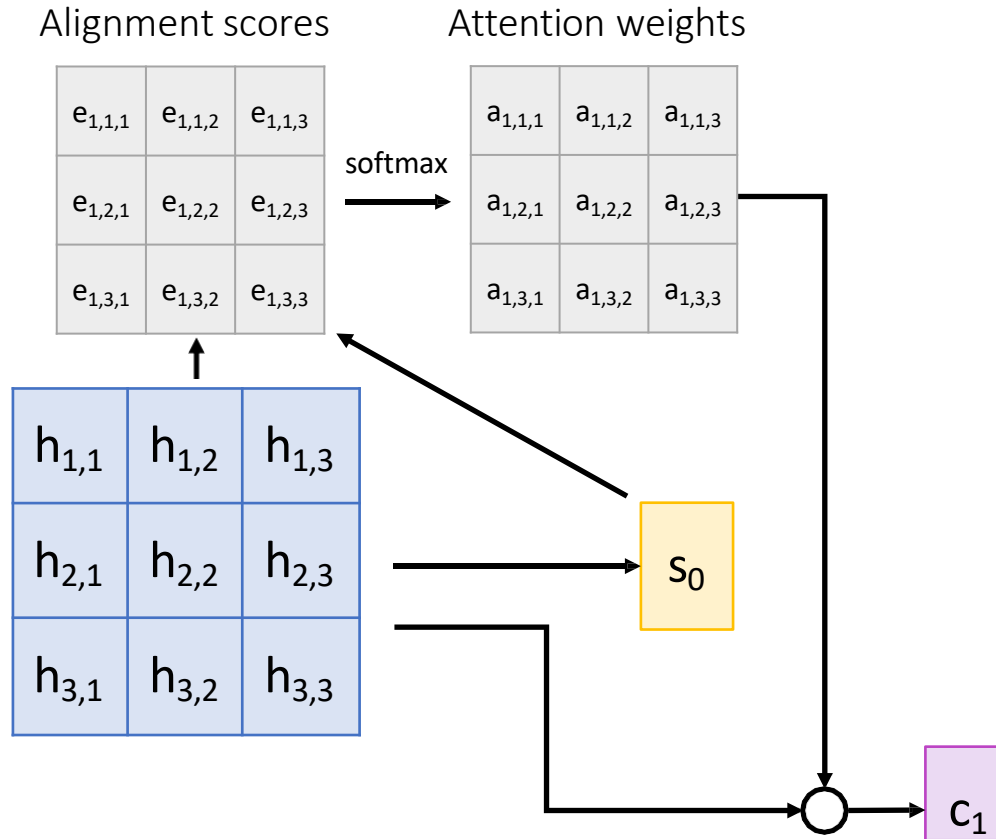
Use a CNN to compute a  
grid of features for an image

# Image Captioning with RNNs and Attention

$$e_{t,i,j} = f_{\text{att}}(s_{t-1}, h_{i,j})$$
$$a_{t,:} = \text{softmax}(e_{t,:})$$
$$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$$

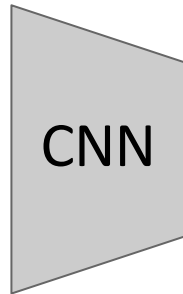


Use a CNN to compute a grid of features for an image

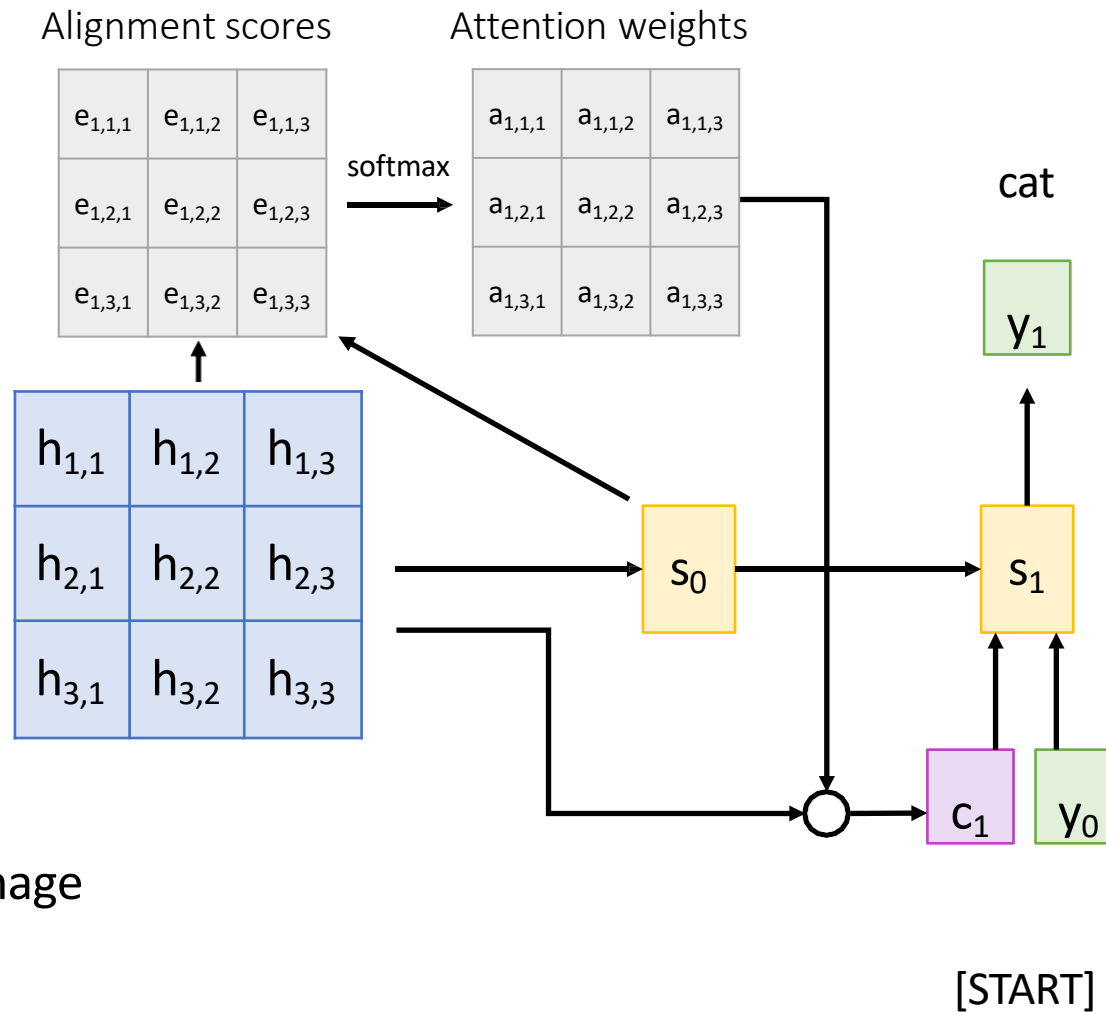


# Image Captioning with RNNs and Attention

$$e_{t,i,j} = f_{\text{att}}(s_{t-1}, h_{i,j})$$
$$a_{t,:} = \text{softmax}(e_{t,:})$$
$$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$$



Use a CNN to compute a grid of features for an image



# Image Captioning with RNNs and Attention

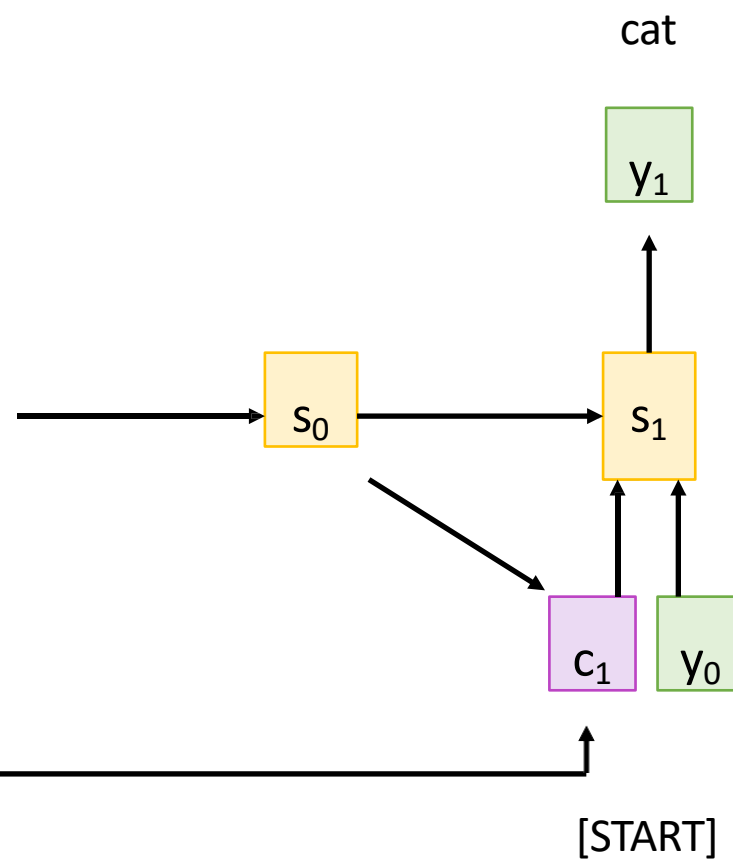
$$e_{t,i,j} = f_{\text{att}}(s_{t-1}, h_{i,j})$$
$$a_{t,:} = \text{softmax}(e_{t,:})$$
$$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$$



CNN

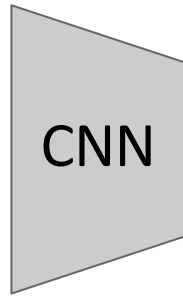
$h_{1,1}$	$h_{1,2}$	$h_{1,3}$
$h_{2,1}$	$h_{2,2}$	$h_{2,3}$
$h_{3,1}$	$h_{3,2}$	$h_{3,3}$

Use a CNN to compute a grid of features for an image

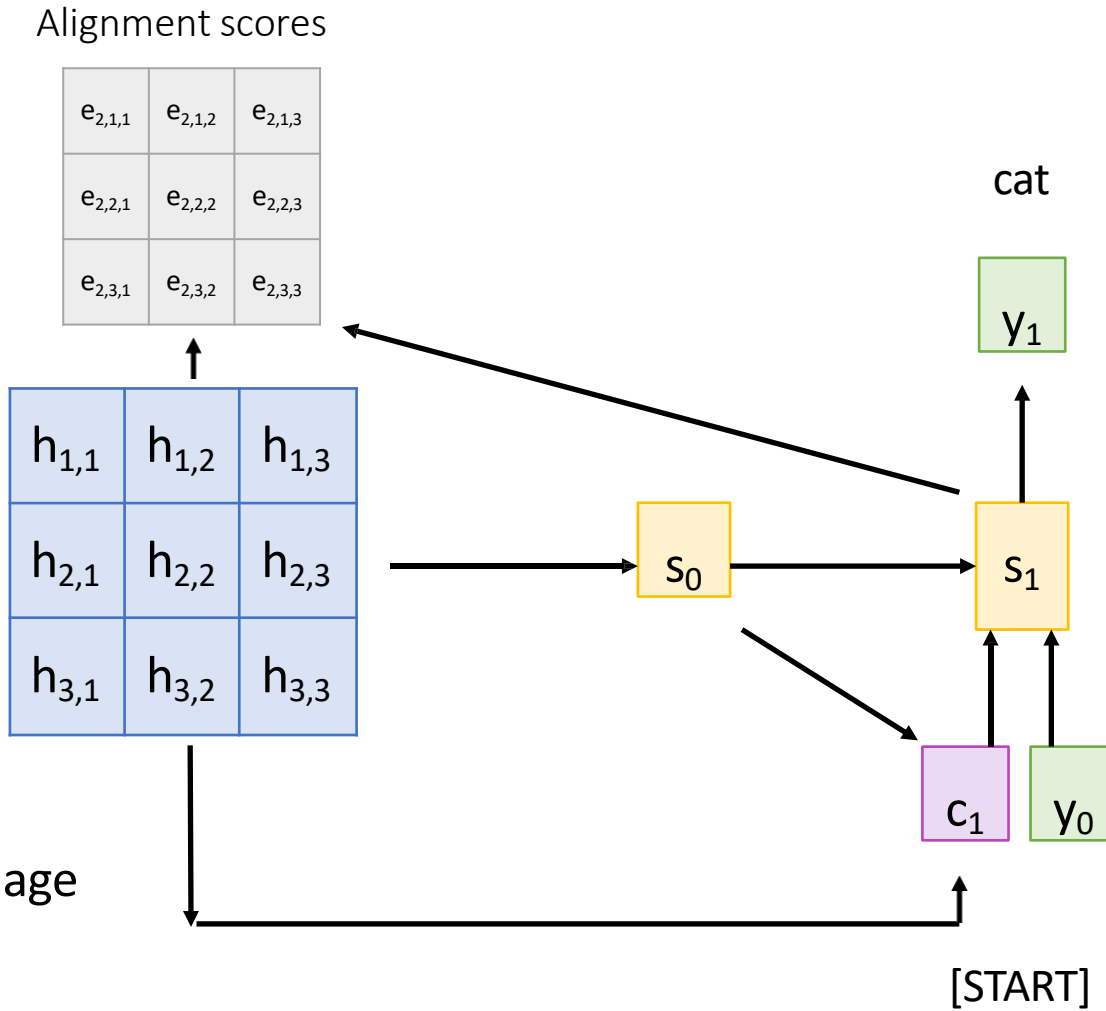


# Image Captioning with RNNs and Attention

$$e_{t,i,j} = f_{\text{att}}(s_{t-1}, h_{i,j})$$
$$a_{t,:} = \text{softmax}(e_{t,:})$$
$$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$$



Use a CNN to compute a grid of features for an image





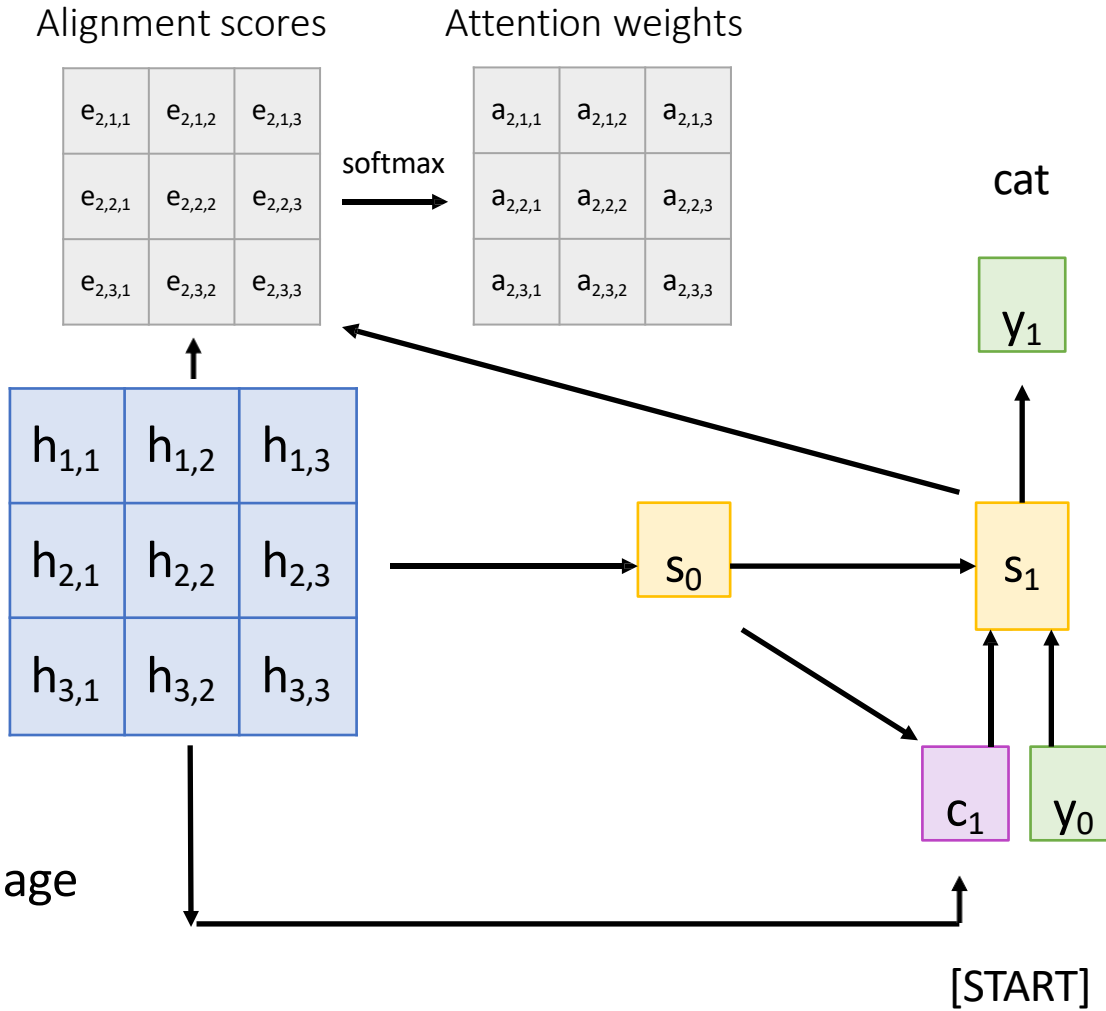
# Image Captioning with RNNs and Attention

$$e_{t,i,j} = f_{\text{att}}(s_{t-1}, h_{i,j})$$
$$a_{t,:} = \text{softmax}(e_{t,:})$$
$$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$$



CNN

Use a CNN to compute a grid of features for an image



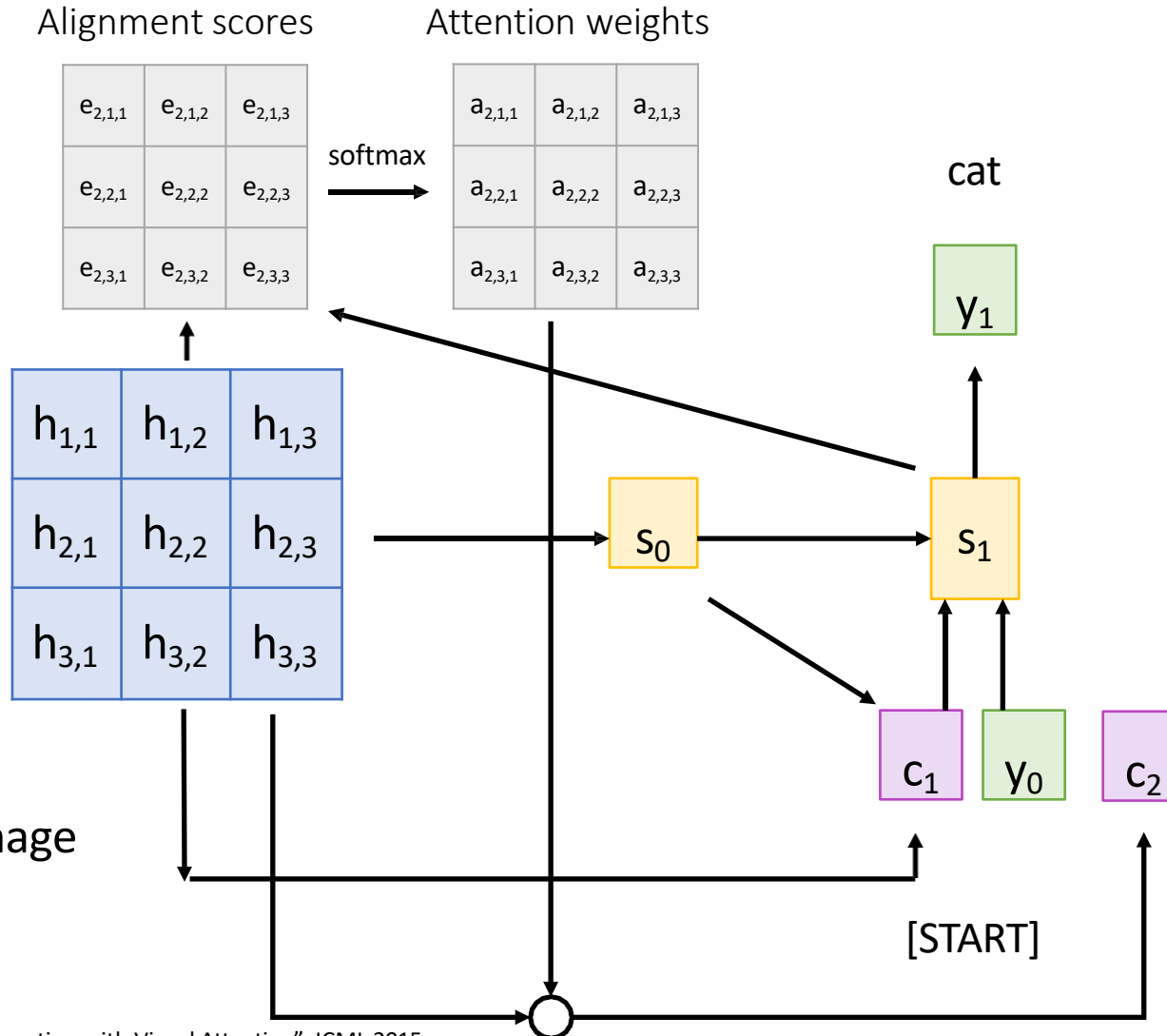
# Image Captioning with RNNs and Attention

$$e_{t,i,j} = f_{\text{att}}(s_{t-1}, h_{i,j})$$
$$a_{t,:} = \text{softmax}(e_{t,:})$$
$$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$$



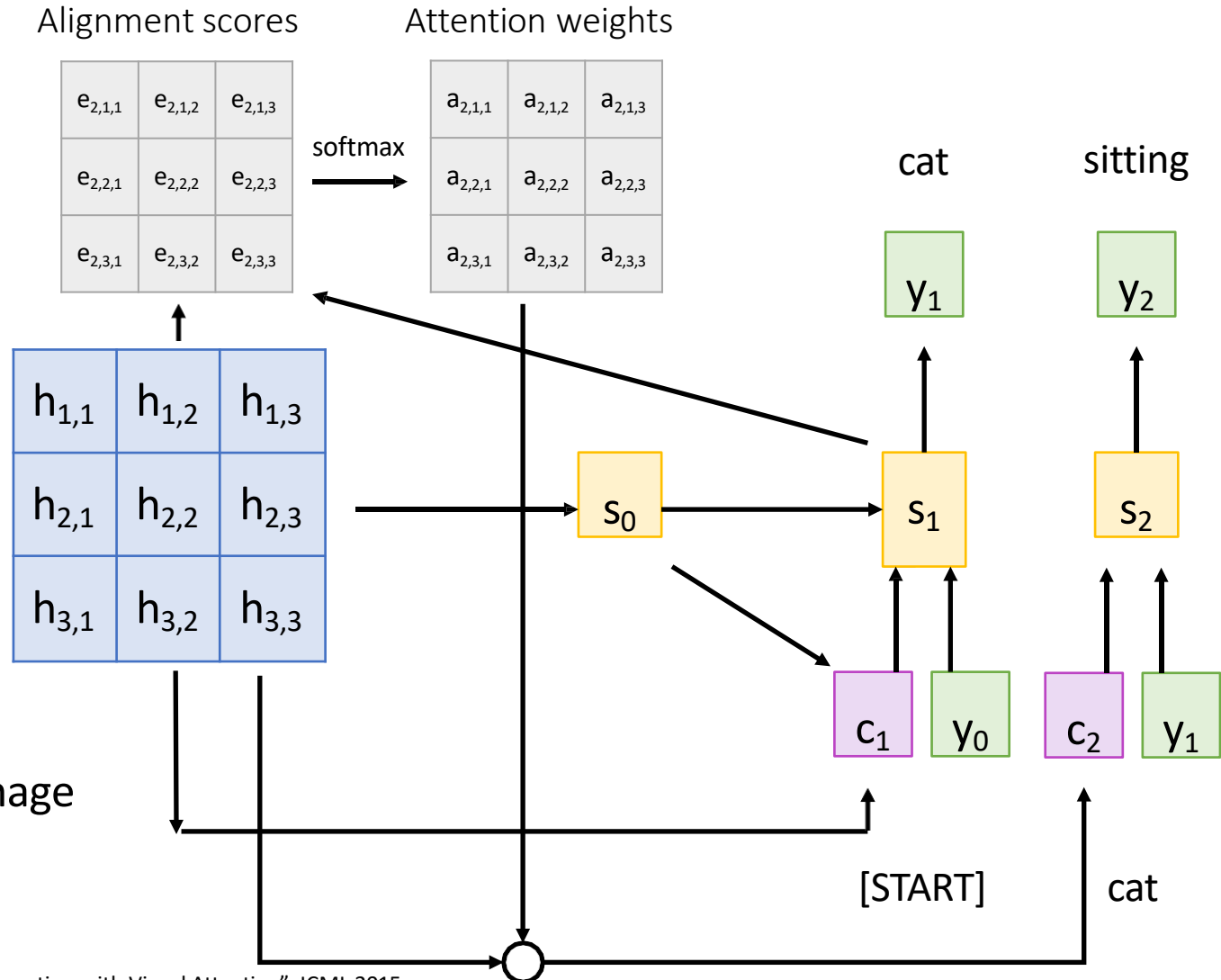
CNN

Use a CNN to compute a grid of features for an image



# Image Captioning with RNNs and Attention

$$\begin{aligned} e_{t,i,j} &= f_{\text{att}}(s_{t-1}, h_{i,j}) \\ a_{t,:,:} &= \text{softmax}(e_{t,:,:}) \\ c_t &= \sum_{i,j} a_{t,i,j} h_{i,j} \end{aligned}$$



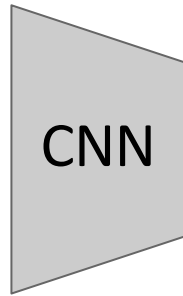
## Use a CNN to compute a grid of features for an image



# Image Captioning with RNNs and Attention

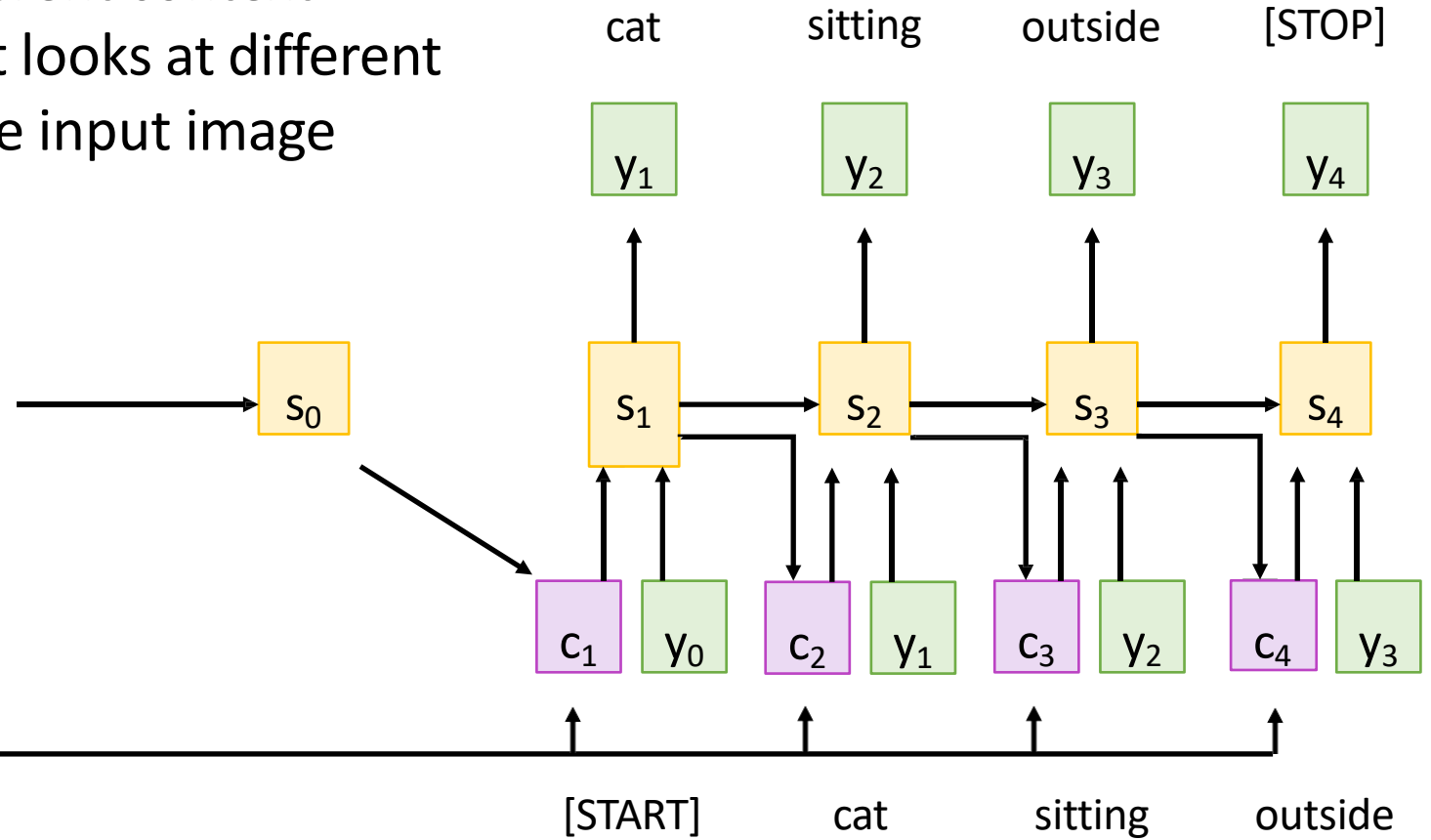
$$e_{t,i,j} = f_{\text{att}}(s_{t-1}, h_{i,j})$$
$$a_{t,:} = \text{softmax}(e_{t,:})$$
$$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$$

Each timestep of decoder  
uses a different context  
vector that looks at different  
parts of the input image

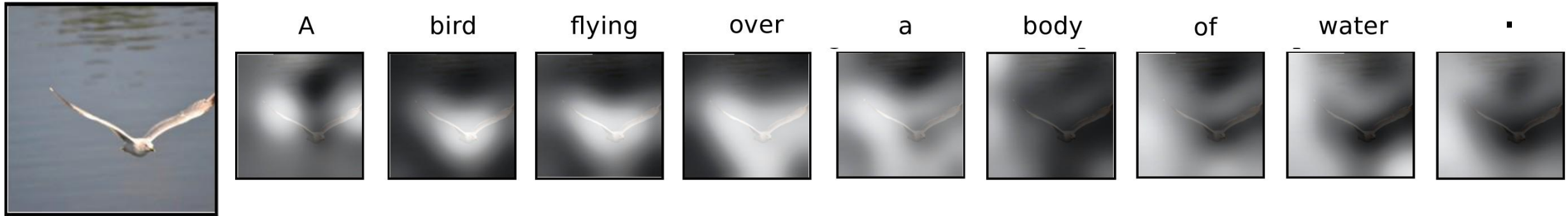


$h_{1,1}$	$h_{1,2}$	$h_{1,3}$
$h_{2,1}$	$h_{2,2}$	$h_{2,3}$
$h_{3,1}$	$h_{3,2}$	$h_{3,3}$

Use a CNN to compute a  
grid of features for an image



# Image Captioning with RNNs and Attention



# Image Captioning with RNNs and Attention



A dog is standing on a hardwood floor.



A stop sign is on a road with a mountain in the background.



A group of people sitting on a boat in the water.



A giraffe standing in a forest with trees in the background.

# X, Attend, and Y

**“Show, attend, and tell”** (*Xu et al, ICML 2015*)

Look at image, attend to image regions, produce question

**“Ask, attend, and answer”** (*Xu and Saenko, ECCV 2016*)

**“Show, ask, attend, and answer”** (*Kazemi and Elqursh, 2017*)

Read text of question, attend to image regions, produce answer

**“Listen, attend, and spell”** (*Chan et al, ICASSP 2016*)

Process raw audio, attend to audio regions while producing text

**“Listen, attend, and walk”** (*Mei et al, AAAI 2016*)

Process text, attend to text regions, output navigation commands

**“Show, attend, and interact”** (*Qureshi et al, ICRA 2017*)

Process image, attend to image regions, output robot control commands

**“Show, attend, and read”** (*Li et al, AAAI 2019*)

Process image, attend to image regions, output text

# Attention Layer

## Inputs:

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

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

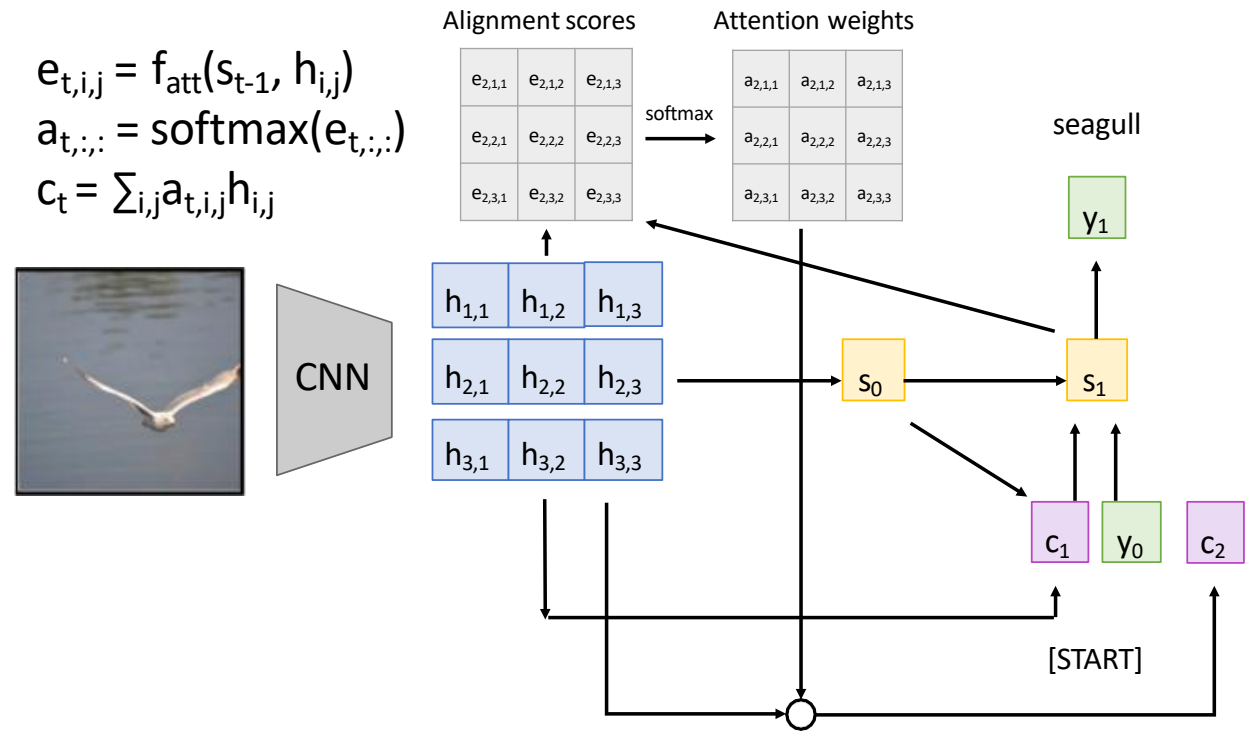
**Similarity function:**  $f_{\text{att}}$

## 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:  $N_X \times D_Q$ )

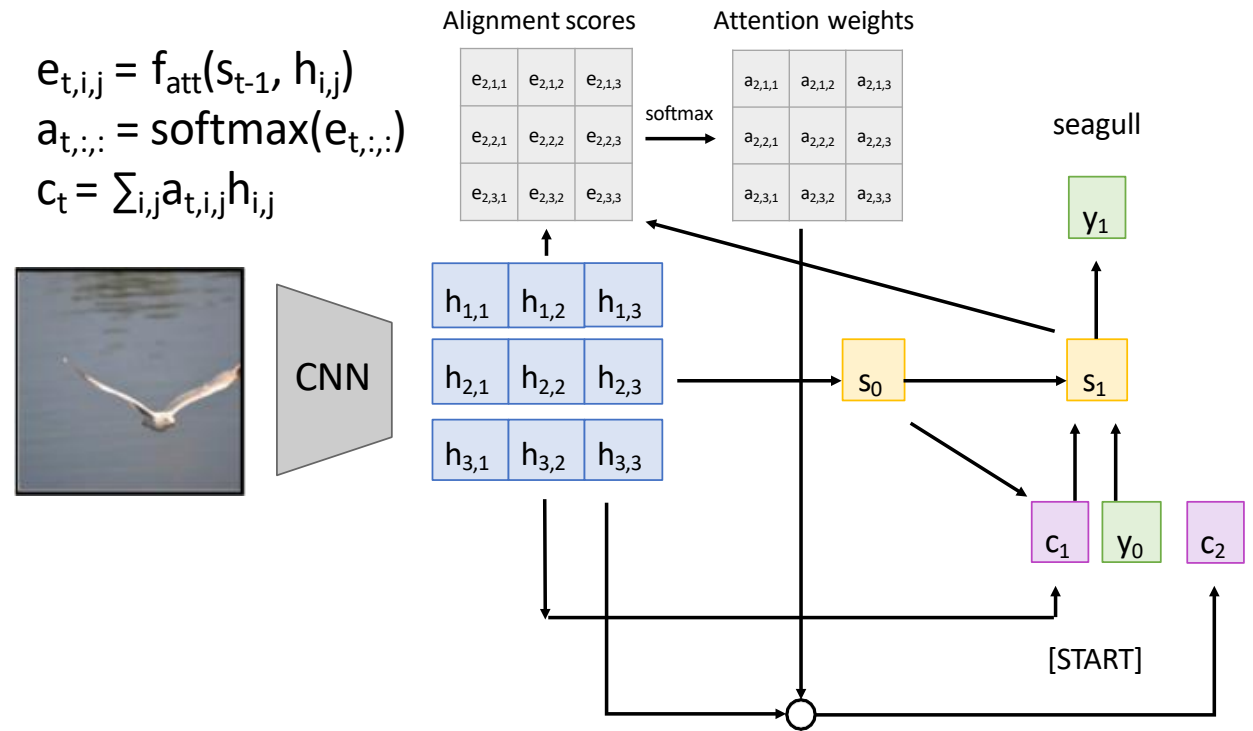
**Similarity function:** dot product

## Computation:

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

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

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



Changes:

- Use dot product for similarity

# Attention Layer

## Inputs:

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

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

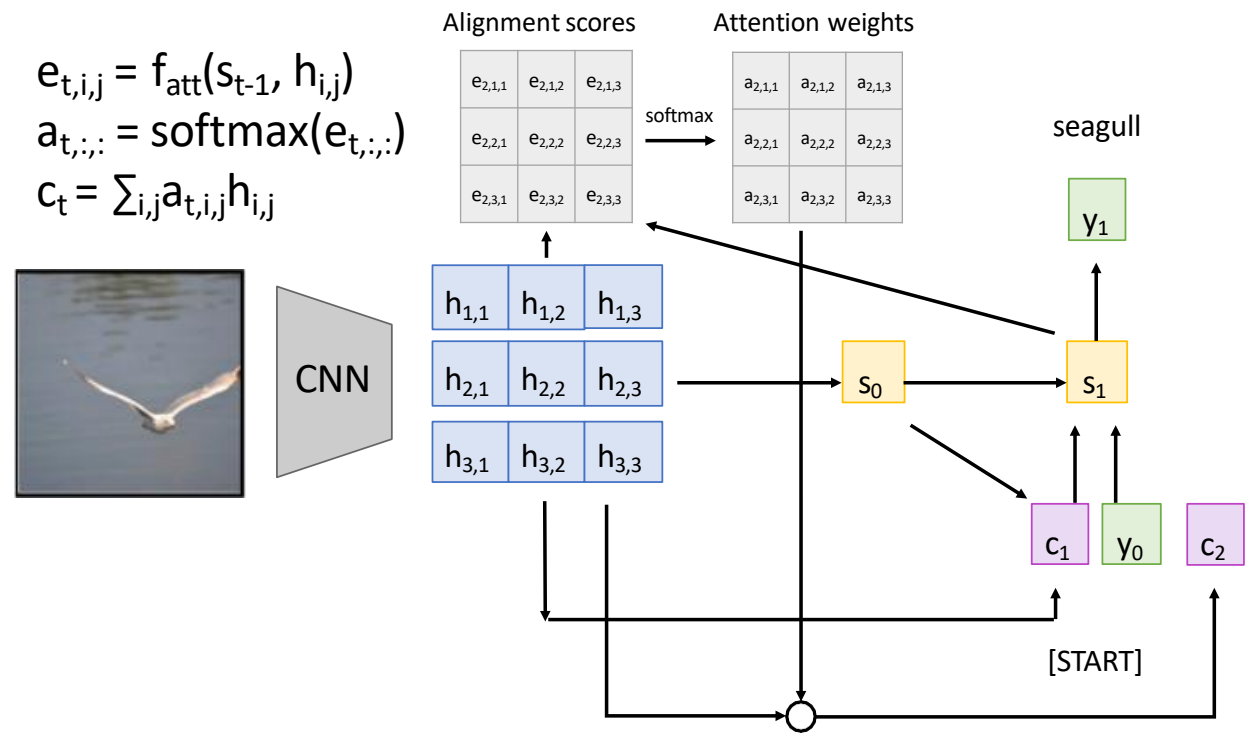
**Similarity function:** scaled dot product

## Computation:

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

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

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

**Similarity function:** scaled dot product

Large similarities will cause softmax to saturate and give vanishing gradients

Recall  $a \cdot b = |a| |b| \cos(\text{angle})$

Suppose that  $a$  and  $b$  are constant vectors of dimension  $D$

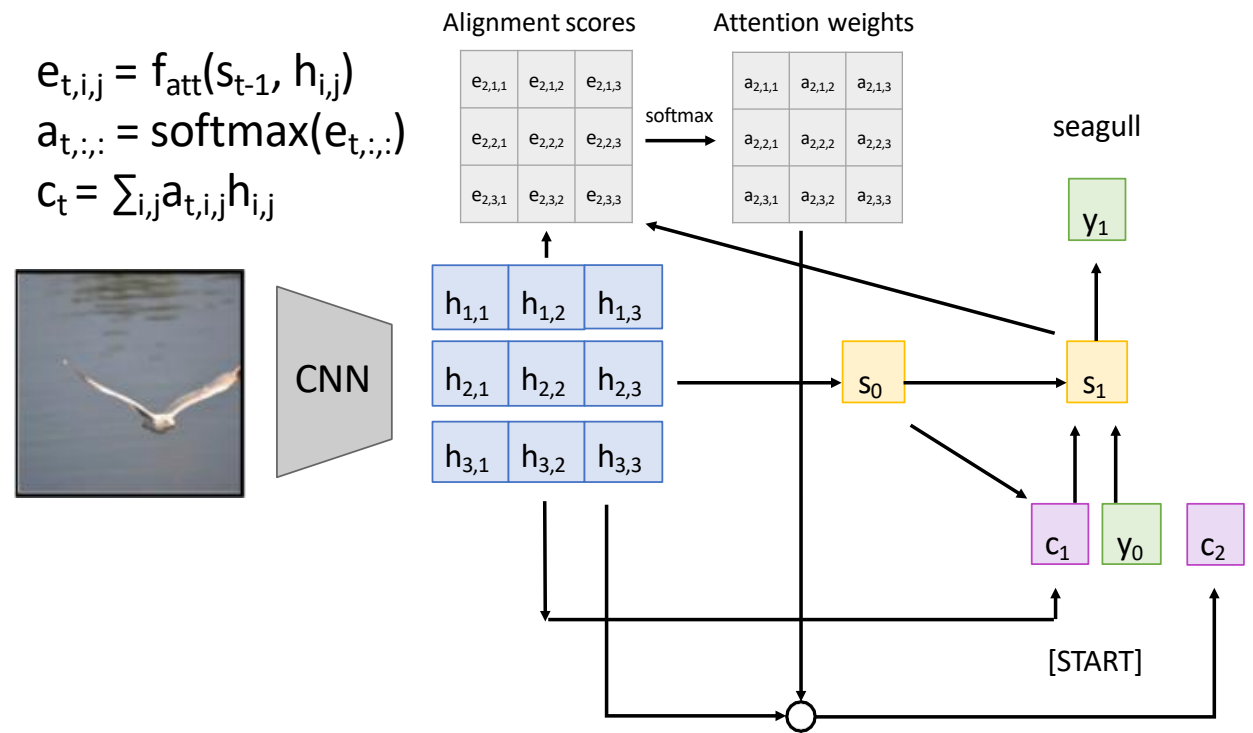
Then  $|a| = (\sum_i a_i^2)^{1/2} = a \text{ sqrt}(D)$

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



Changes:

- Use **scaled** dot product for similarity

# Attention Layer

## Inputs:

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

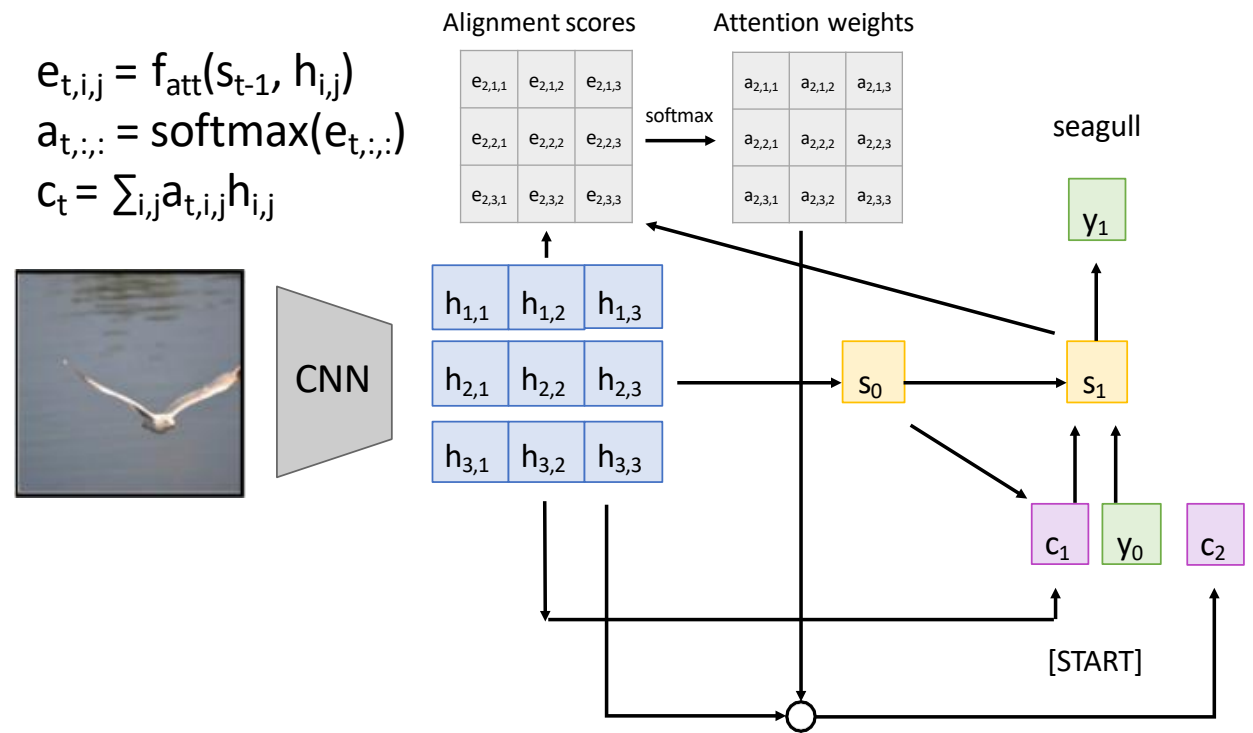
Input vectors: **X** (Shape:  $N_X \times D_X$ )

## Computation:

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

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:

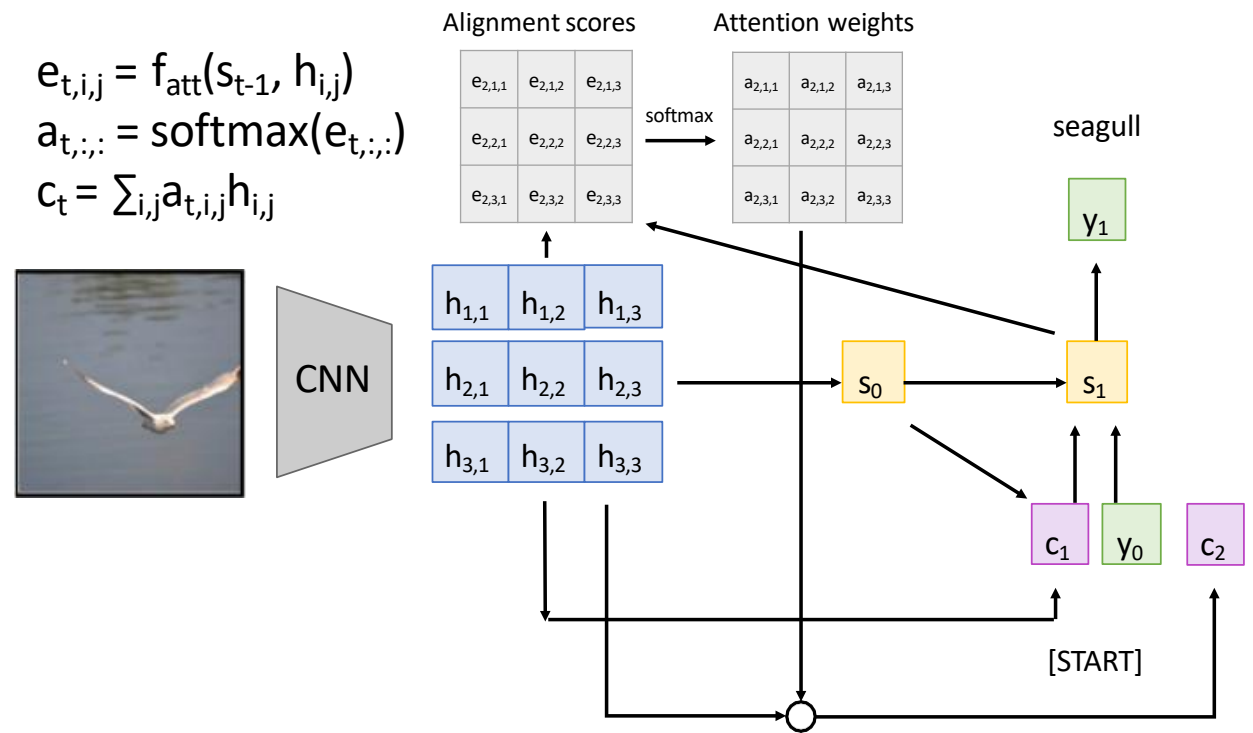
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

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

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

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

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



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

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

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

$X_1$

$X_2$

$X_3$

$Q_1$

$Q_2$

$Q_3$

$Q_4$

# 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:

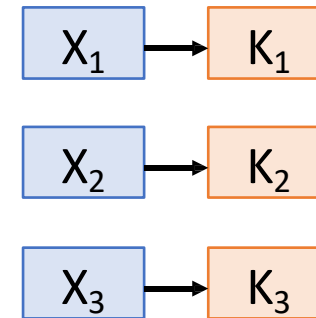
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

**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:

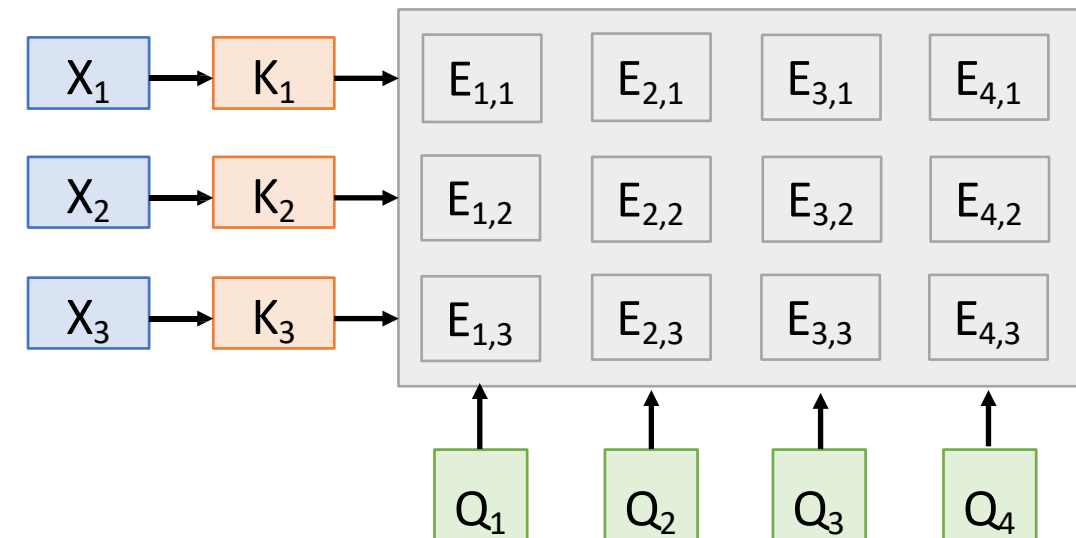
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

**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:

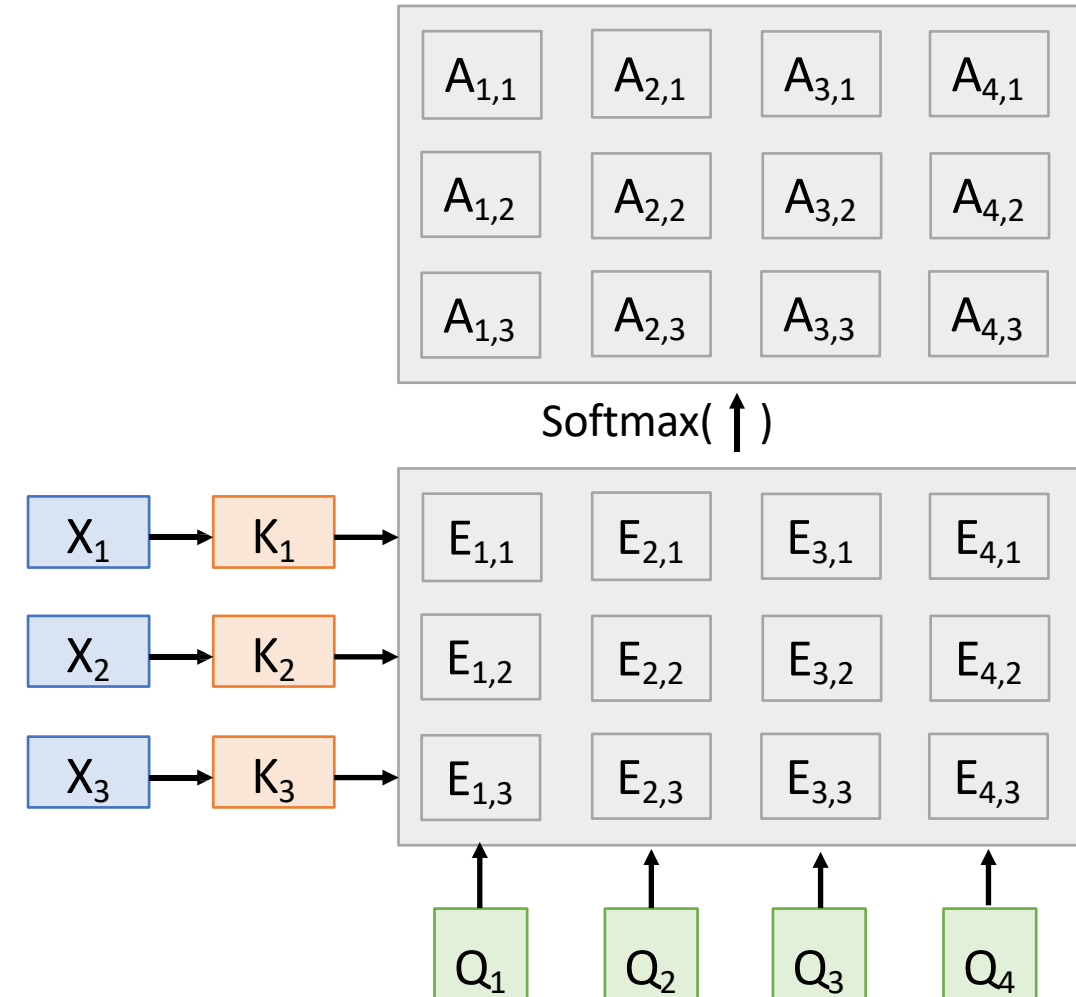
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

**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:

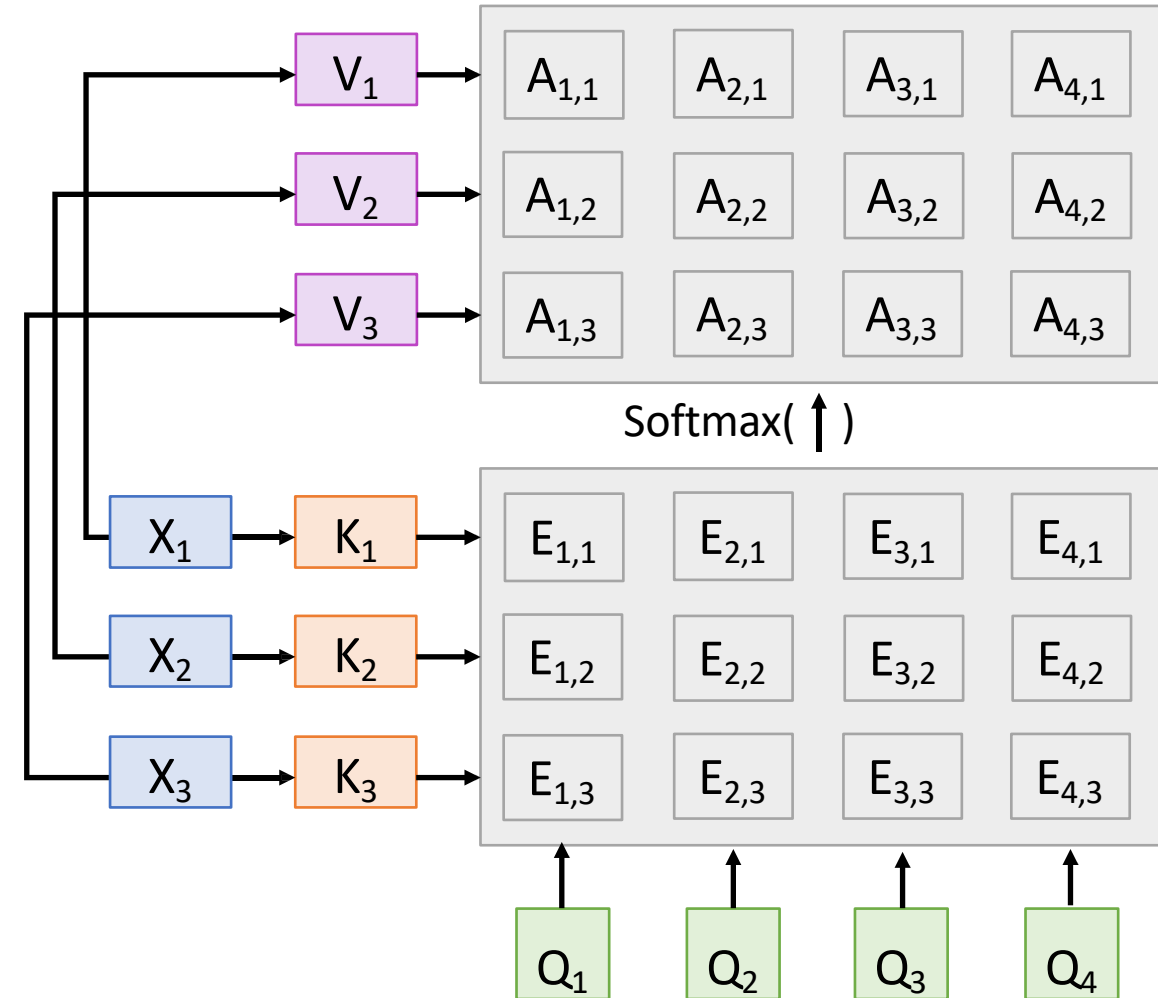
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

**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:

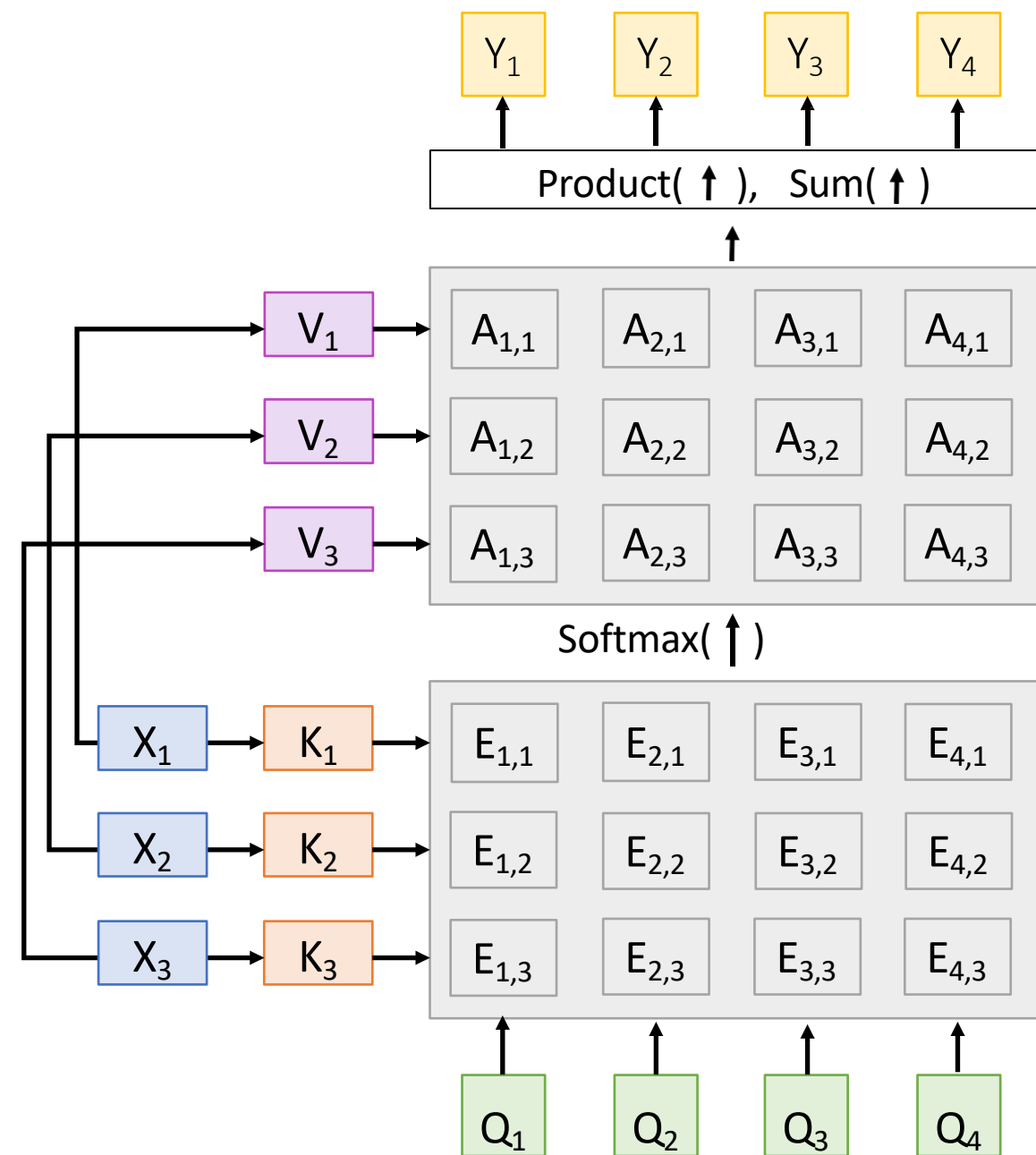
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

**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

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

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

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

$X_1$

$X_2$

$X_3$

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

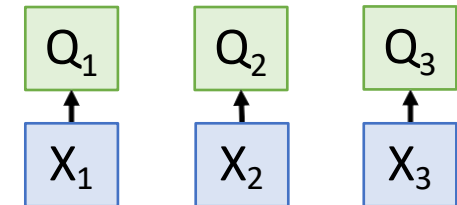
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

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

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

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

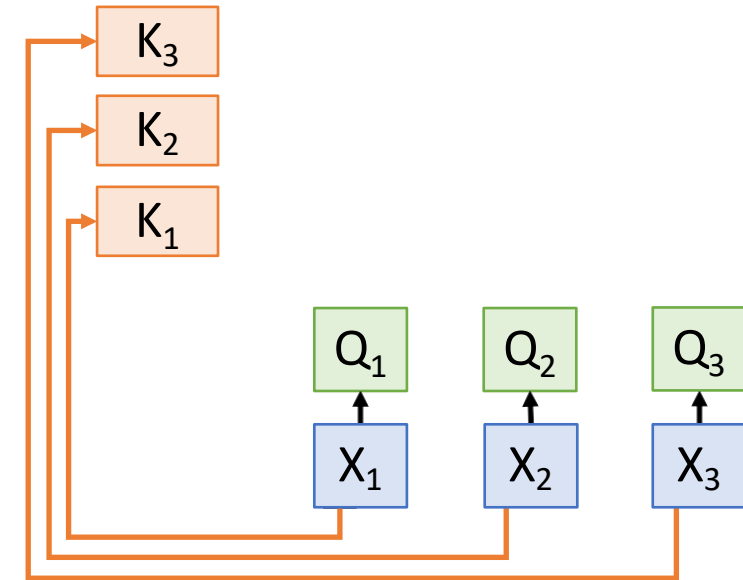
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

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



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

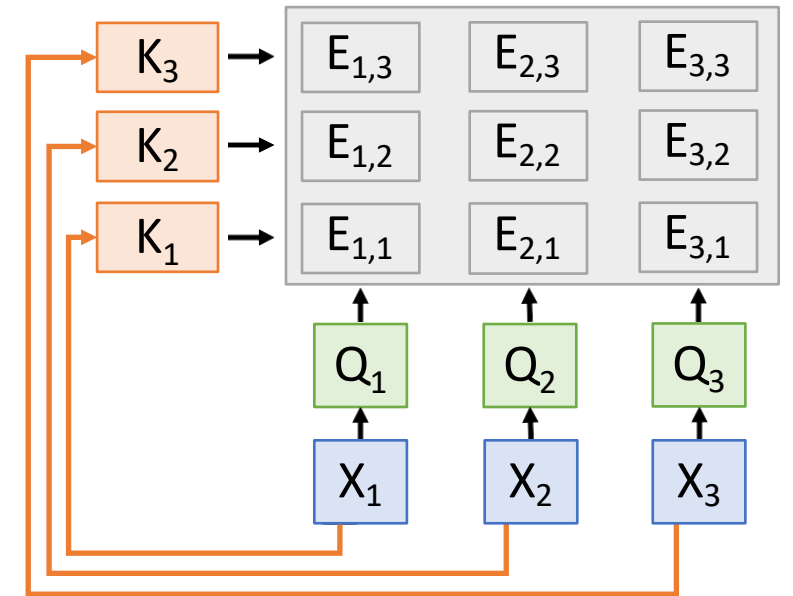
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

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

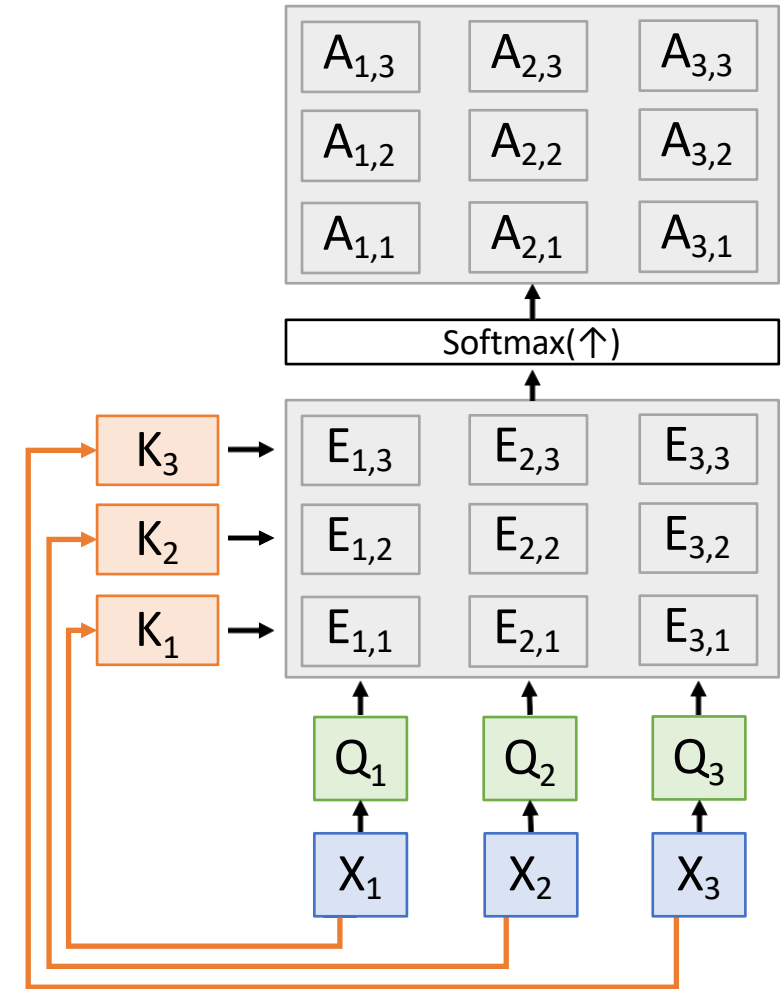
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

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

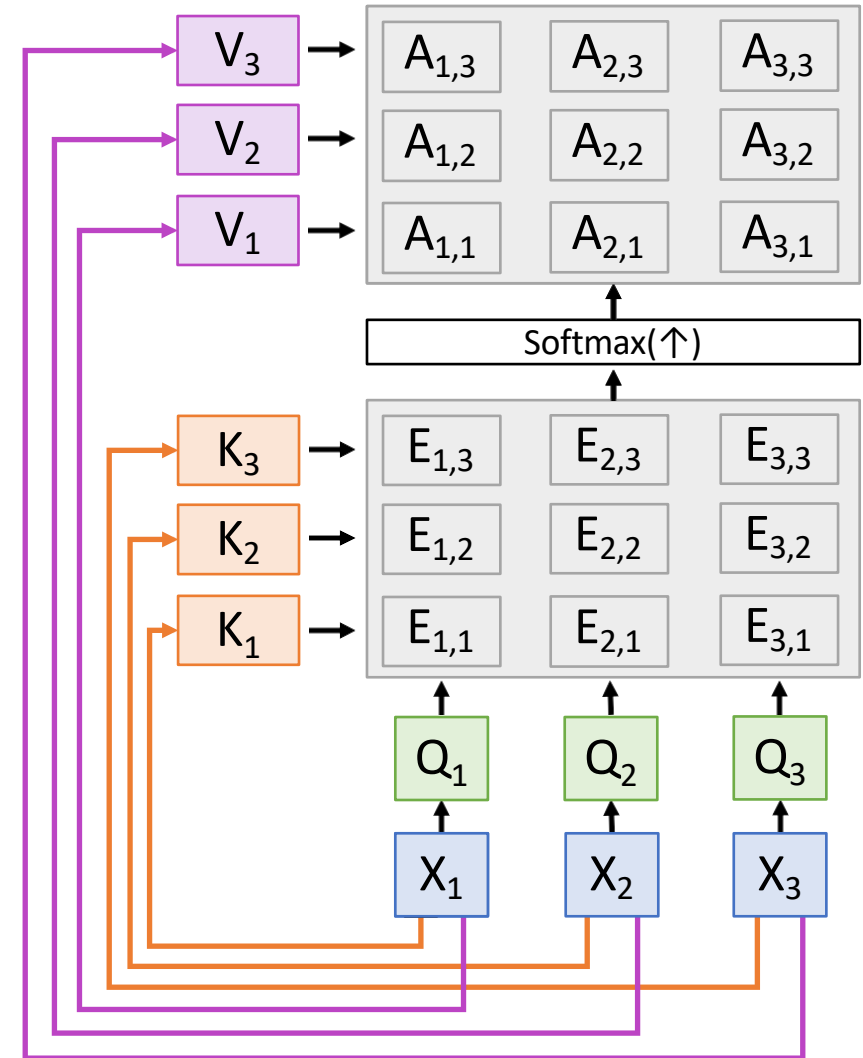
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

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

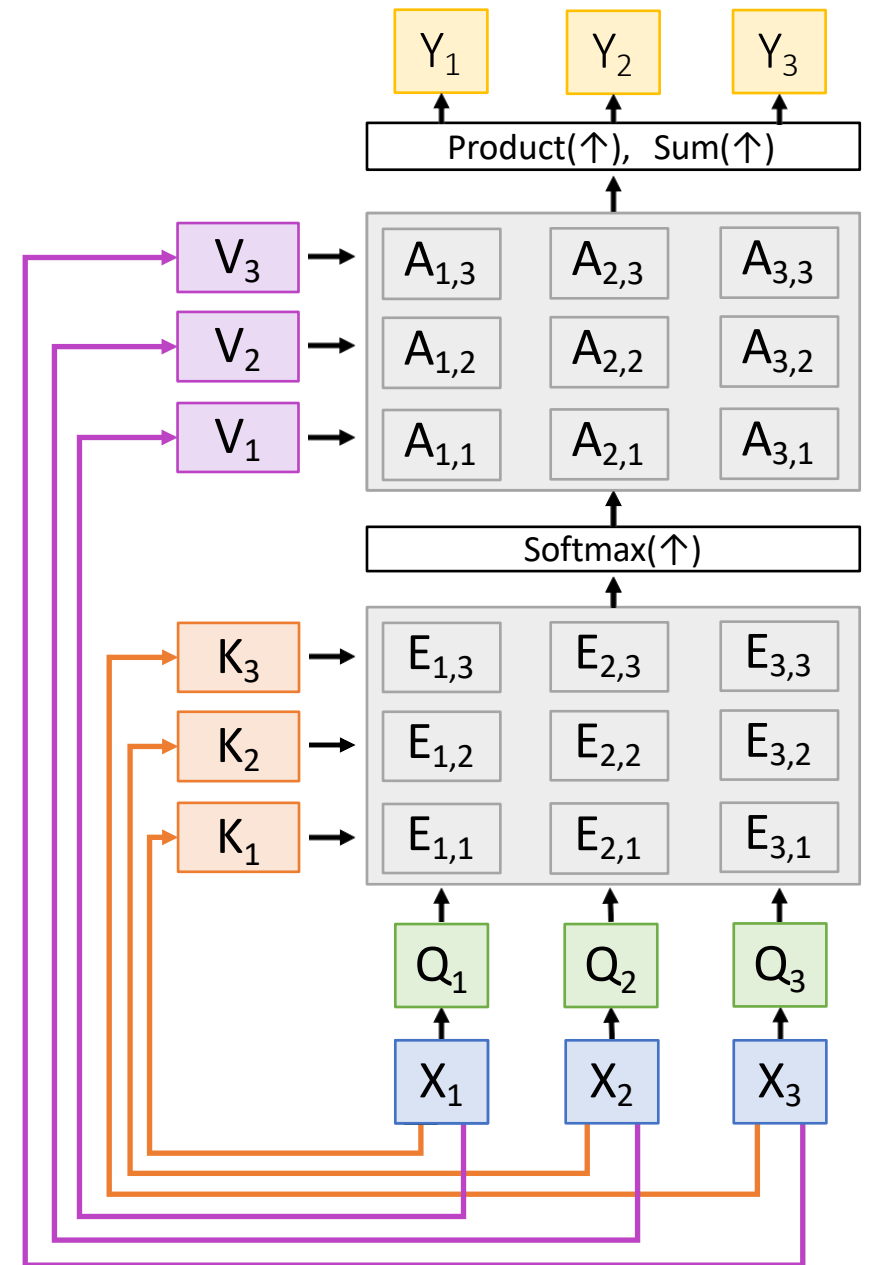
Key vectors:  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

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 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:  $N_X \times D_Q$ )

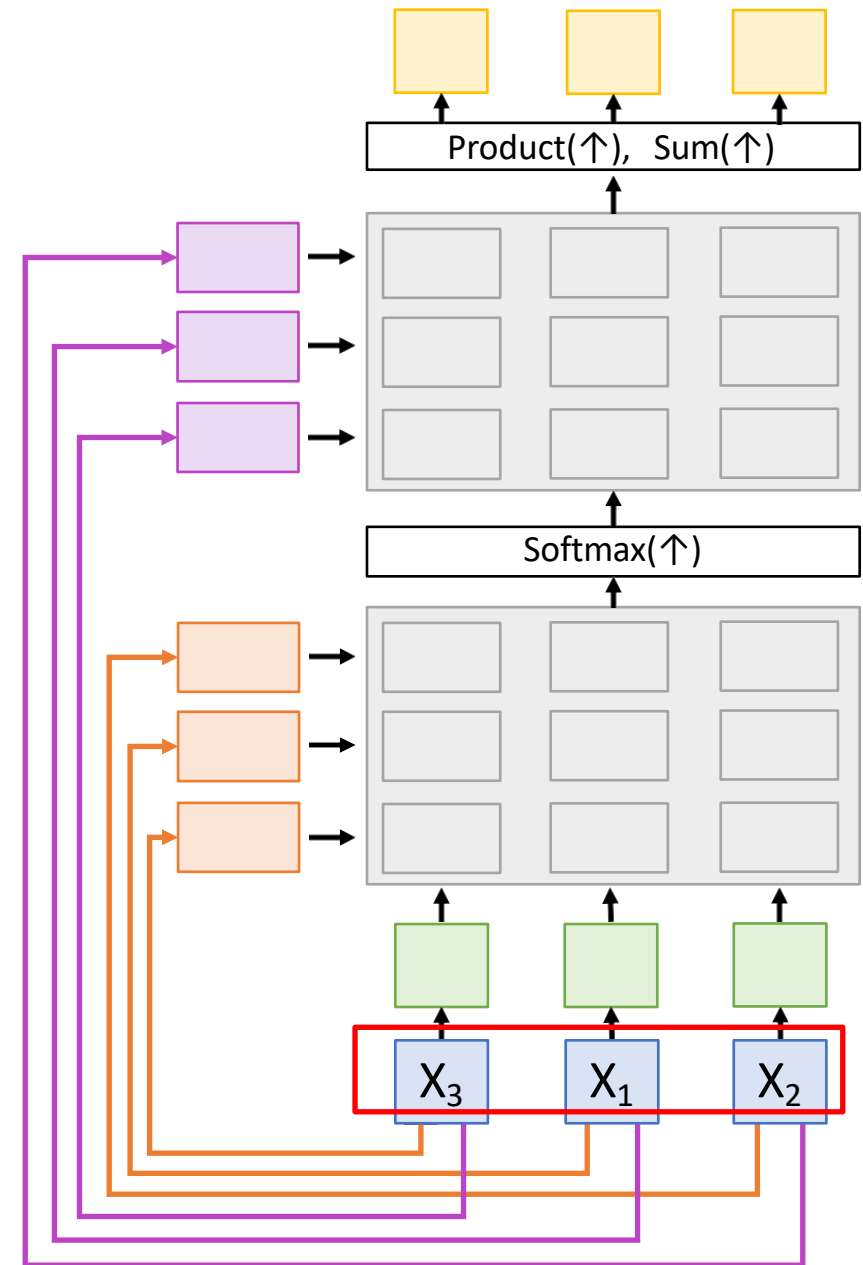
Value Vectors:  $\mathbf{V} = \mathbf{XW}_V$  (Shape:  $N_X \times D_V$ )

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:  $N_X \times D_V$ )

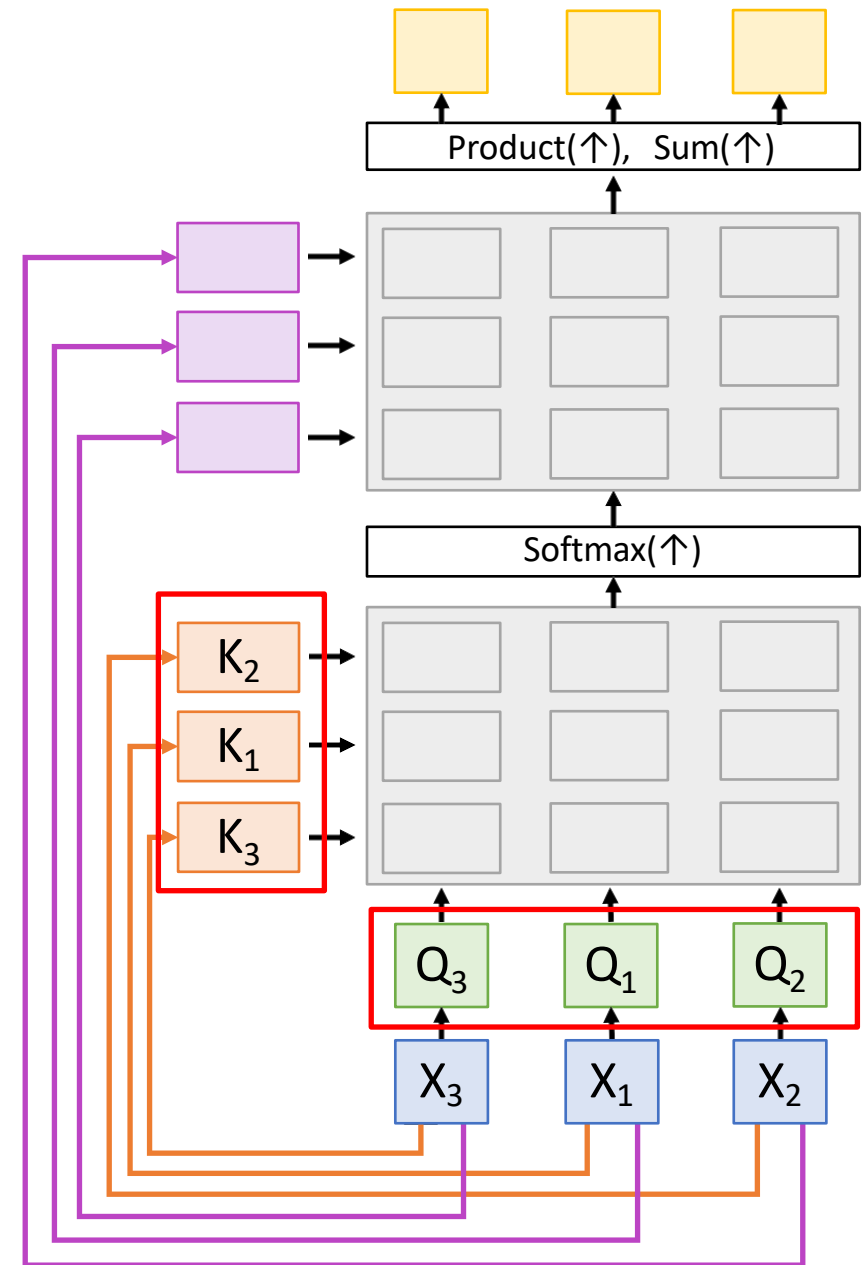
Similarities:  $\mathbf{E} = \mathbf{QK}^T$  (Shape:  $N_X \times N_X$ )  $E_{i,j} = \mathbf{Q}_i \cdot \mathbf{K}_j / \text{sqrt}(D_Q)$

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:  $N_X \times D_V$ )

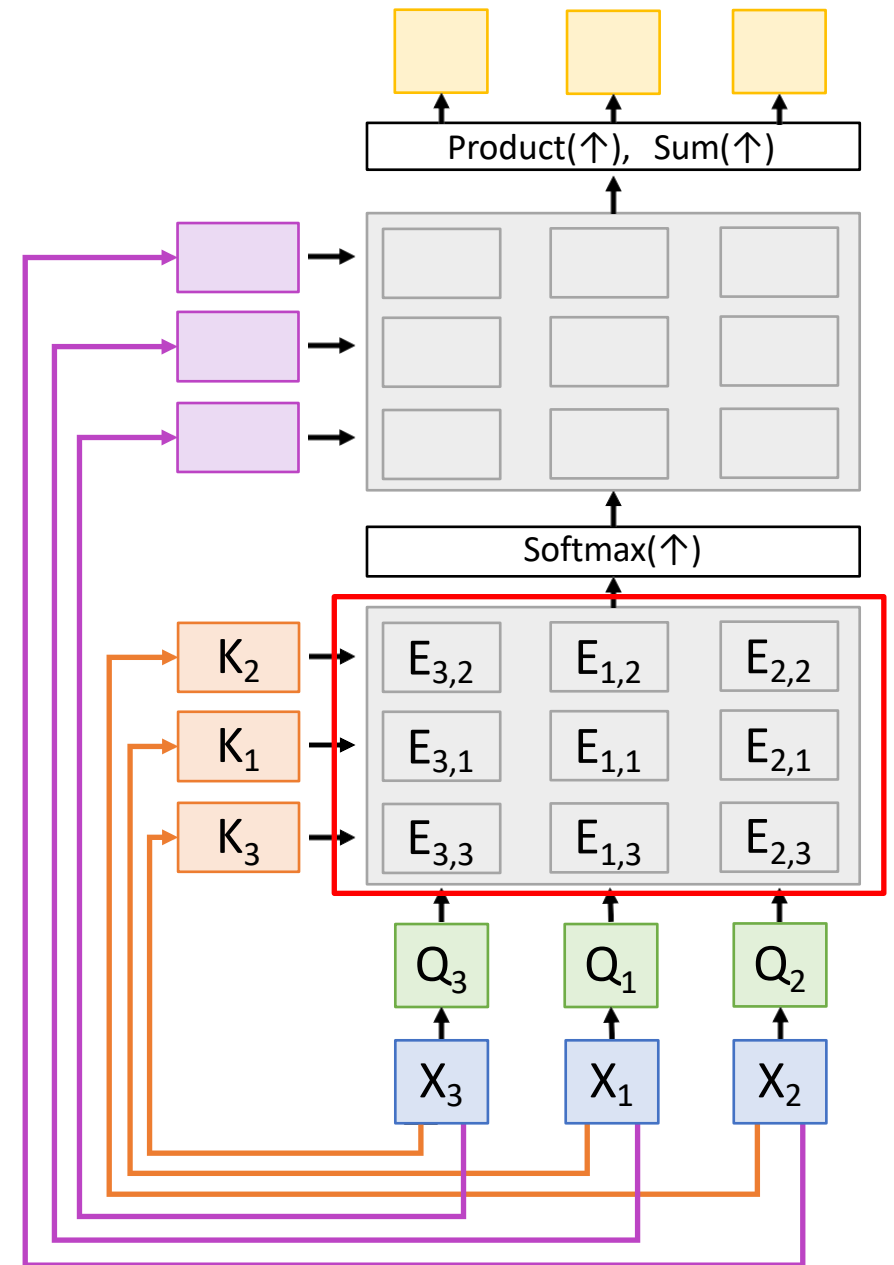
Similarities:  $\mathbf{E} = \mathbf{QK}^T$  (Shape:  $N_X \times N_X$ )  $E_{i,j} = \mathbf{Q}_i \cdot \mathbf{K}_j / \text{sqrt}(D_Q)$

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:  $N_X \times D_V$ )

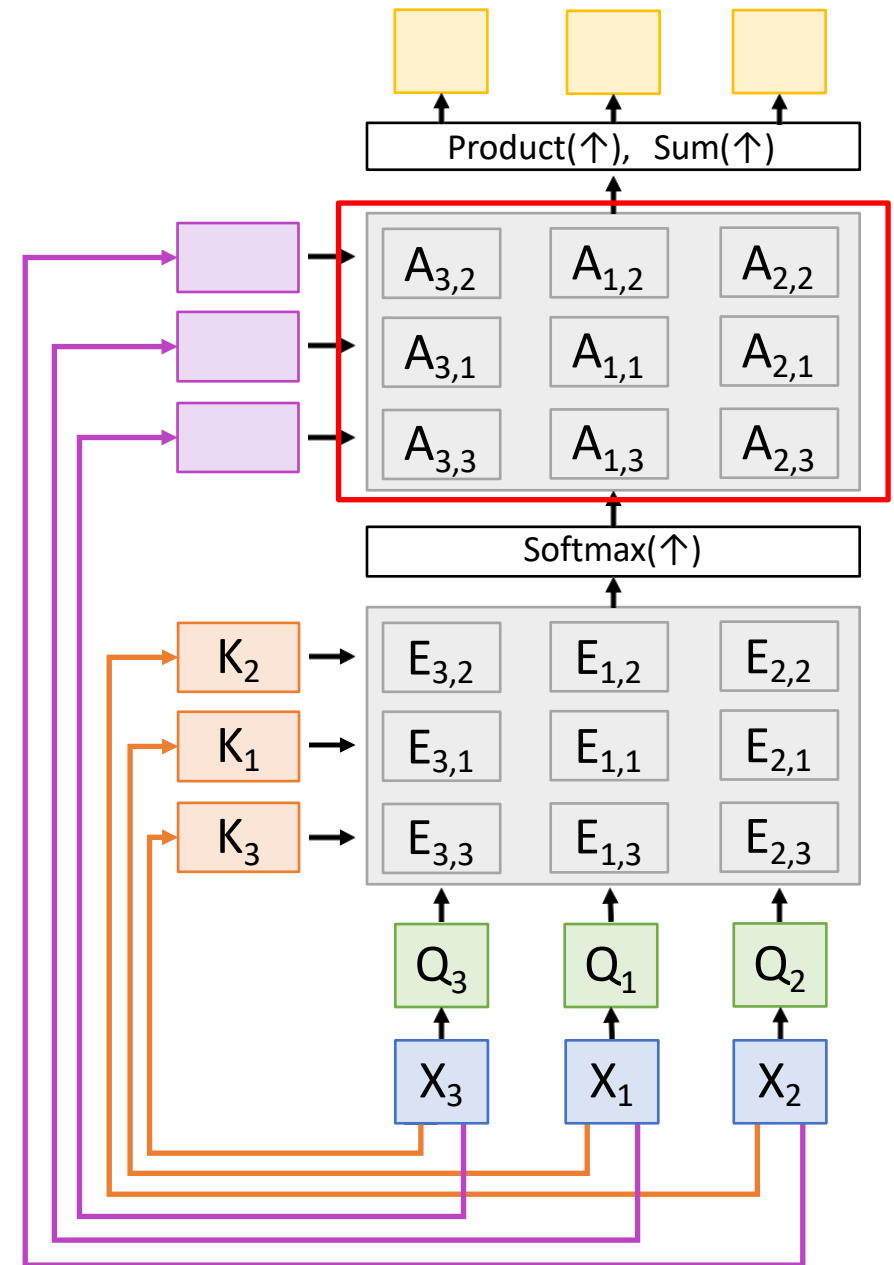
Similarities:  $\mathbf{E} = \mathbf{QK}^T$  (Shape:  $N_X \times N_X$ )  $E_{i,j} = \mathbf{Q}_i \cdot \mathbf{K}_j / \text{sqrt}(D_Q)$

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:  $\mathbf{X}$  (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:  $N_X \times D_V$ )

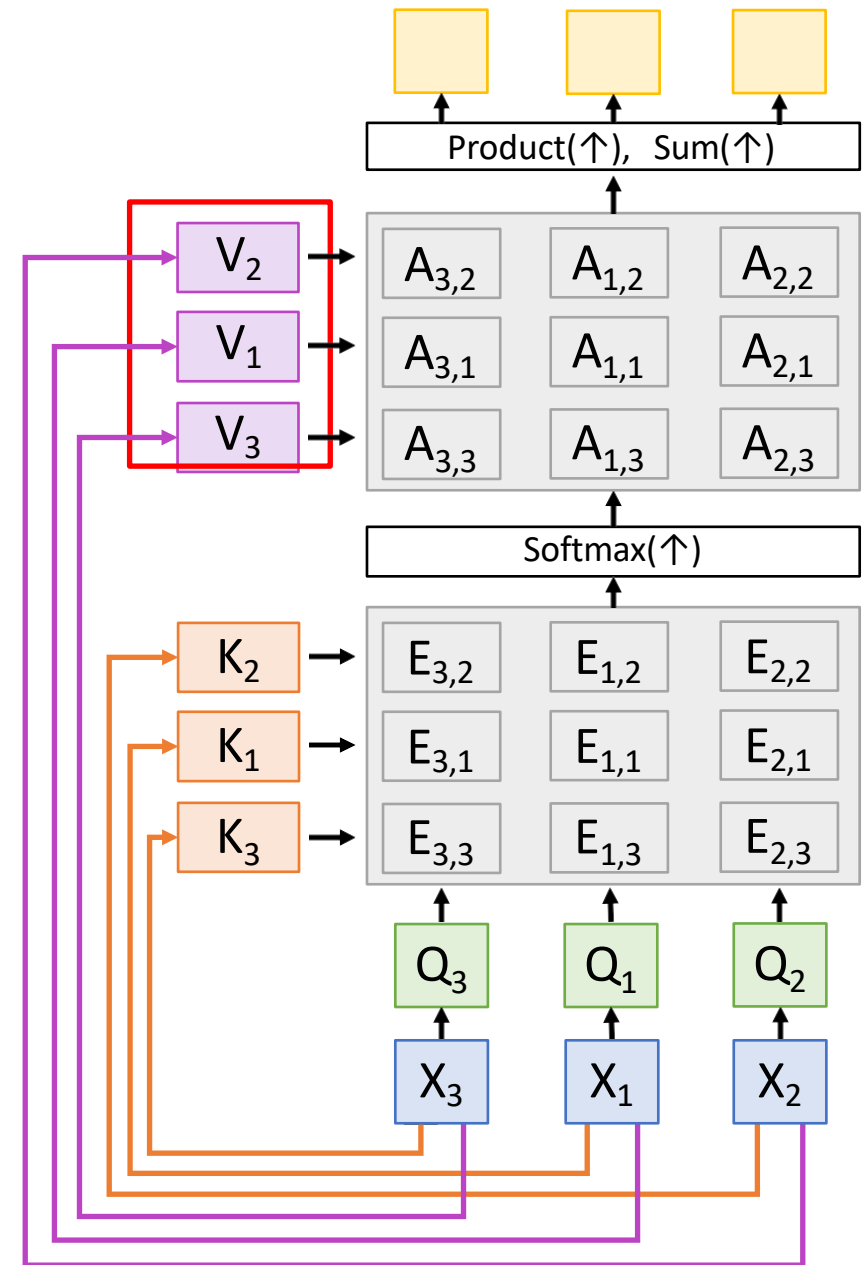
Similarities:  $\mathbf{E} = \mathbf{QK}^T$  (Shape:  $N_X \times N_X$ )  $E_{i,j} = \mathbf{Q}_i \cdot \mathbf{K}_j / \text{sqrt}(D_Q)$

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:

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:  $N_X \times D_V$ )

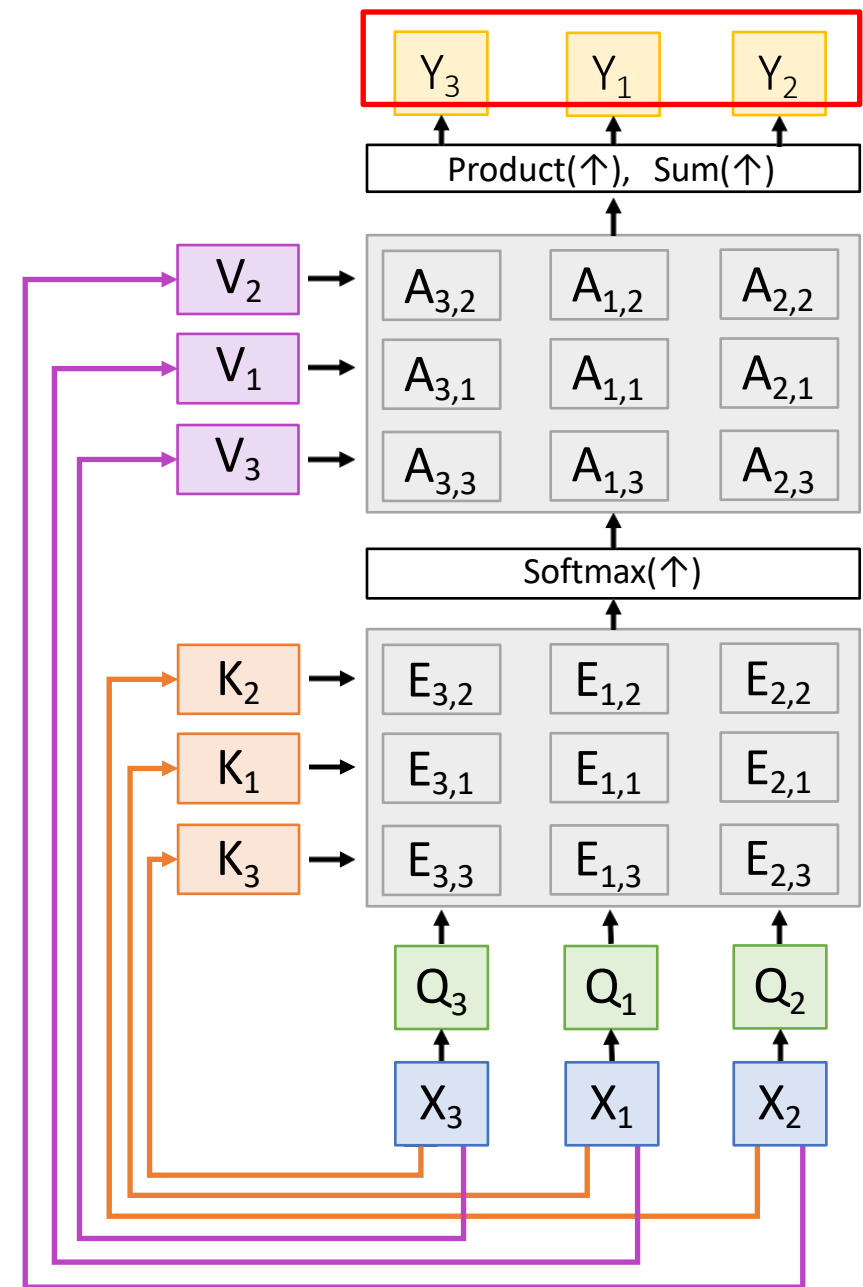
Similarities:  $\mathbf{E} = \mathbf{QK}^T$  (Shape:  $N_X \times N_X$ )  $E_{i,j} = \mathbf{Q}_i \cdot \mathbf{K}_j / \text{sqrt}(D_Q)$

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

## 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:  $N_X \times N_X$ )

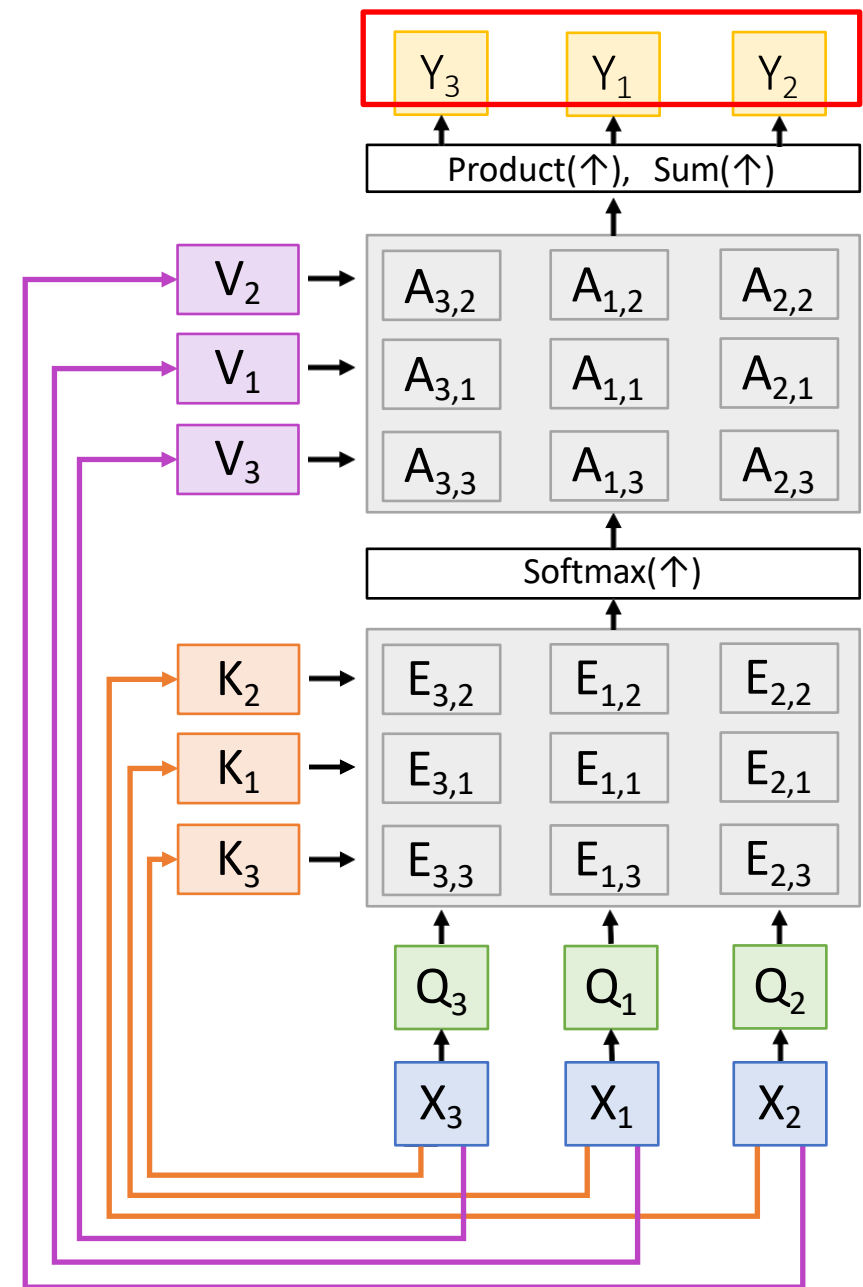
Output vectors:  $\mathbf{Y} = \mathbf{AV}$  (Shape:  $N_X \times D_V$ )  $Y_i = \sum_j A_{i,j} \mathbf{V}_j$

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 works  
on **sets** of 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:  $N_X \times N_X$ )  $E_{i,j} = \mathbf{Q}_i \cdot \mathbf{K}_j / \text{sqrt}(D_Q)$

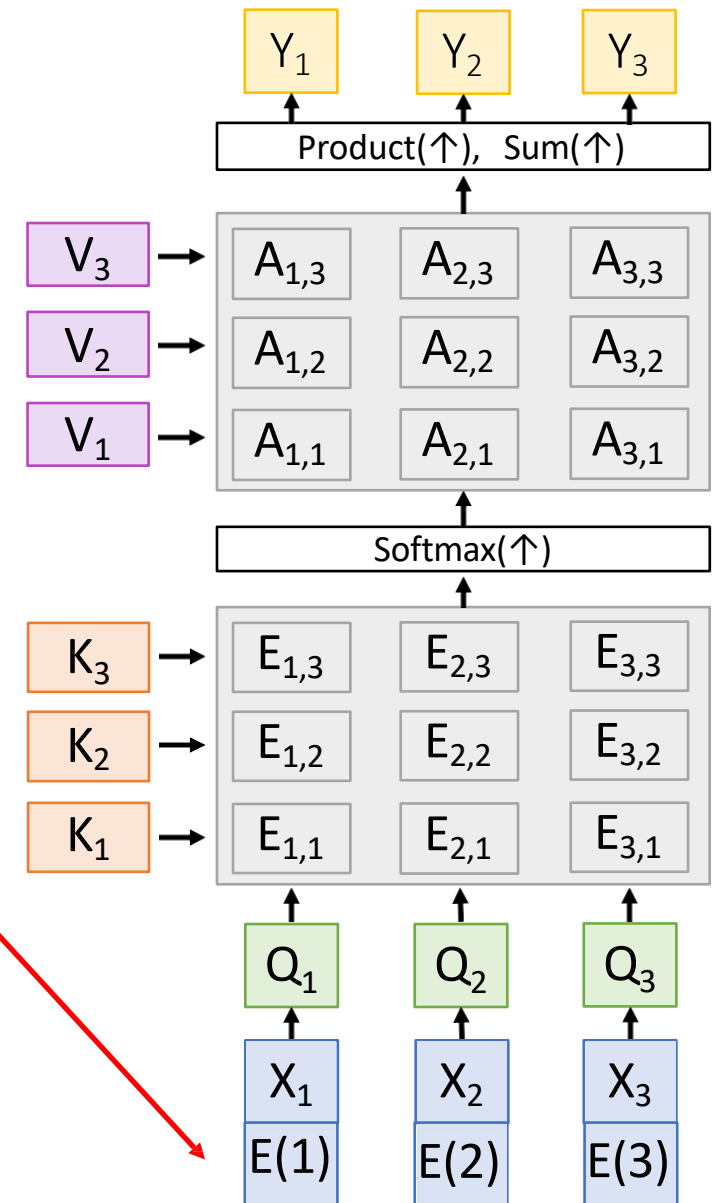
Attention weights:  $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$  (Shape:  $N_X \times N_X$ )

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

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

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

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

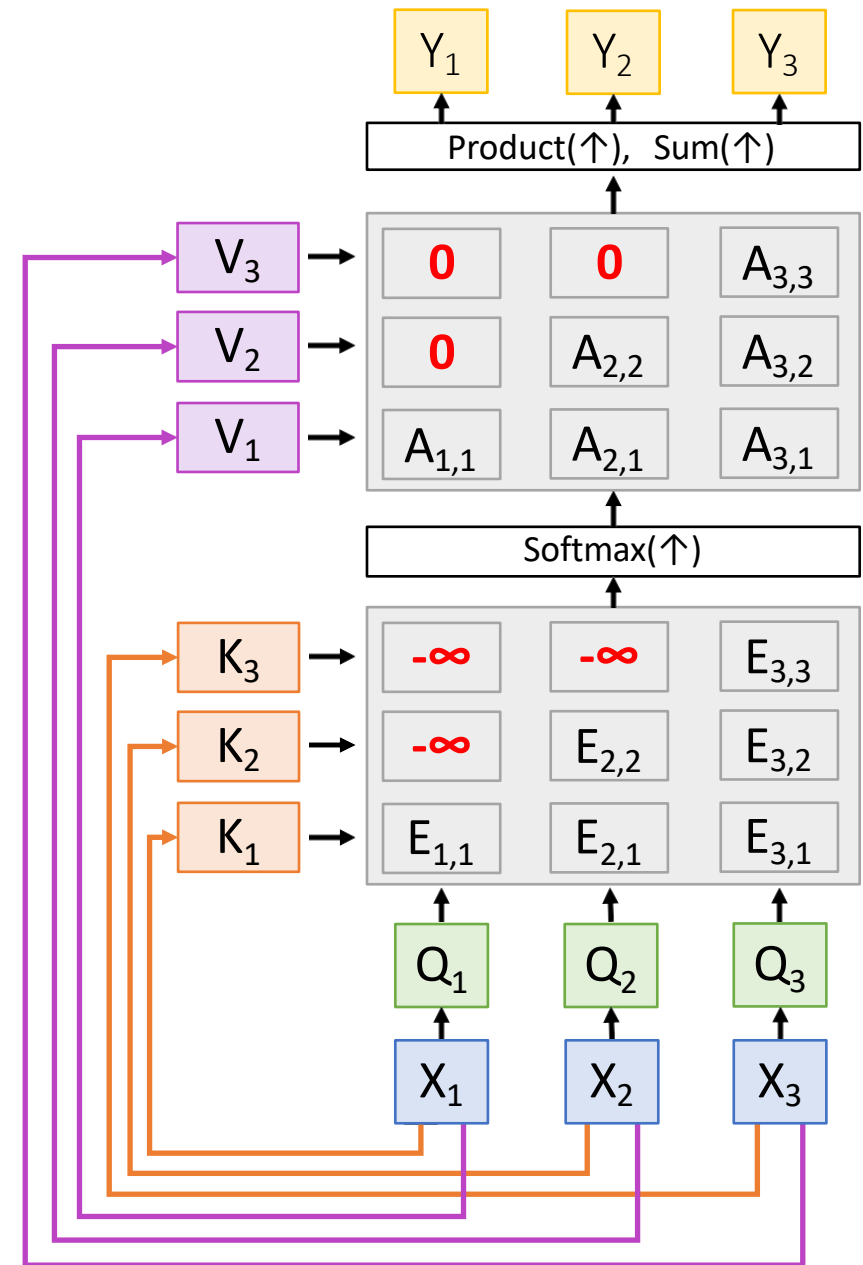
**Key vectors:**  $\mathbf{K} = \mathbf{XW}_K$  (Shape:  $N_X \times D_Q$ )

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



# Multihead Self-Attention Layer

Use H independent  
“Attention Heads” in parallel

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

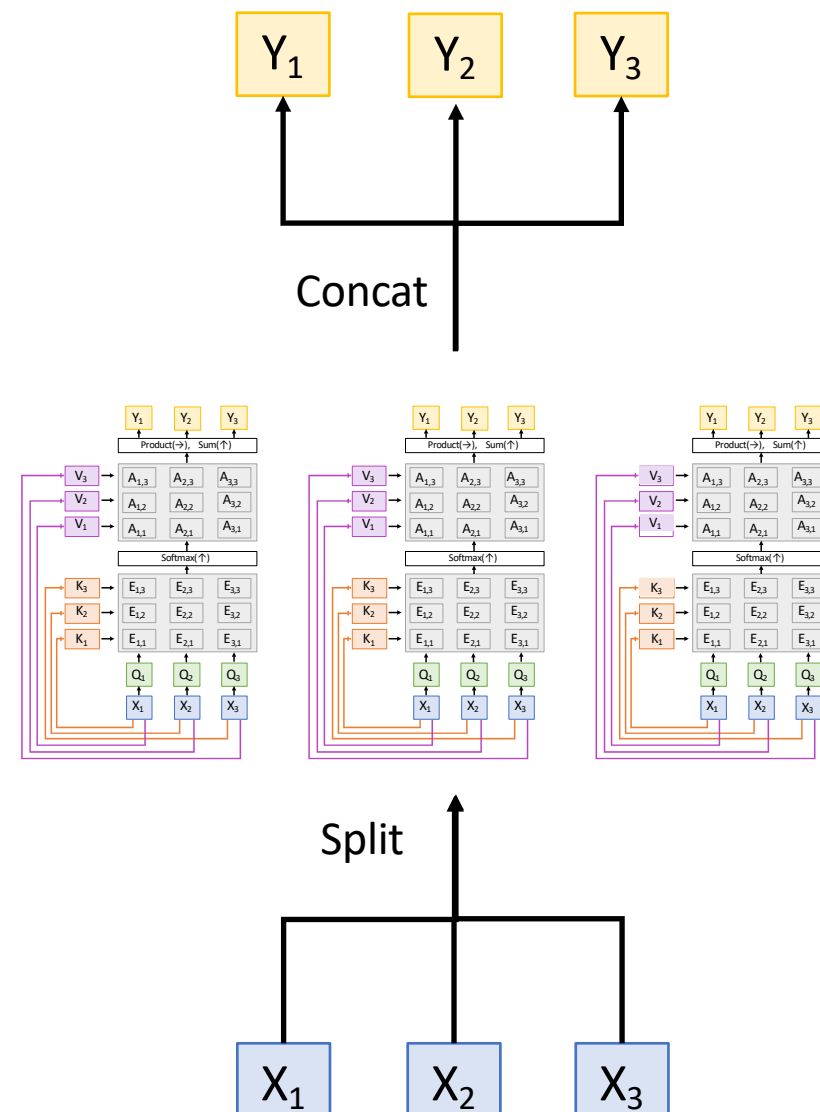
Attention weights:  $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$  (Shape:  $N_X \times N_X$ )

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

## Hyperparameters:

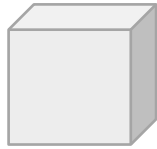
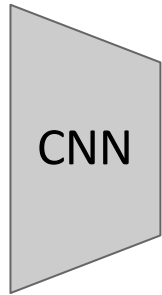
Query dimension  $D_Q$

Number of heads  $H$



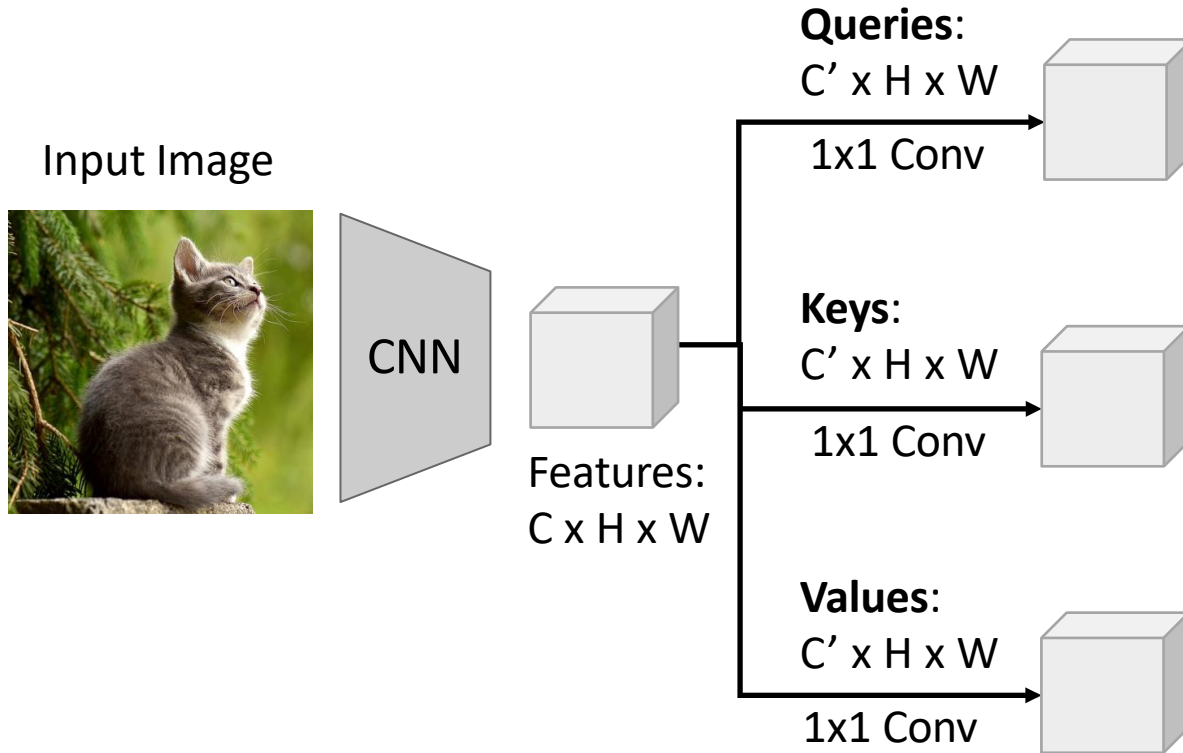
# Example: CNN with Self-Attention

Input Image

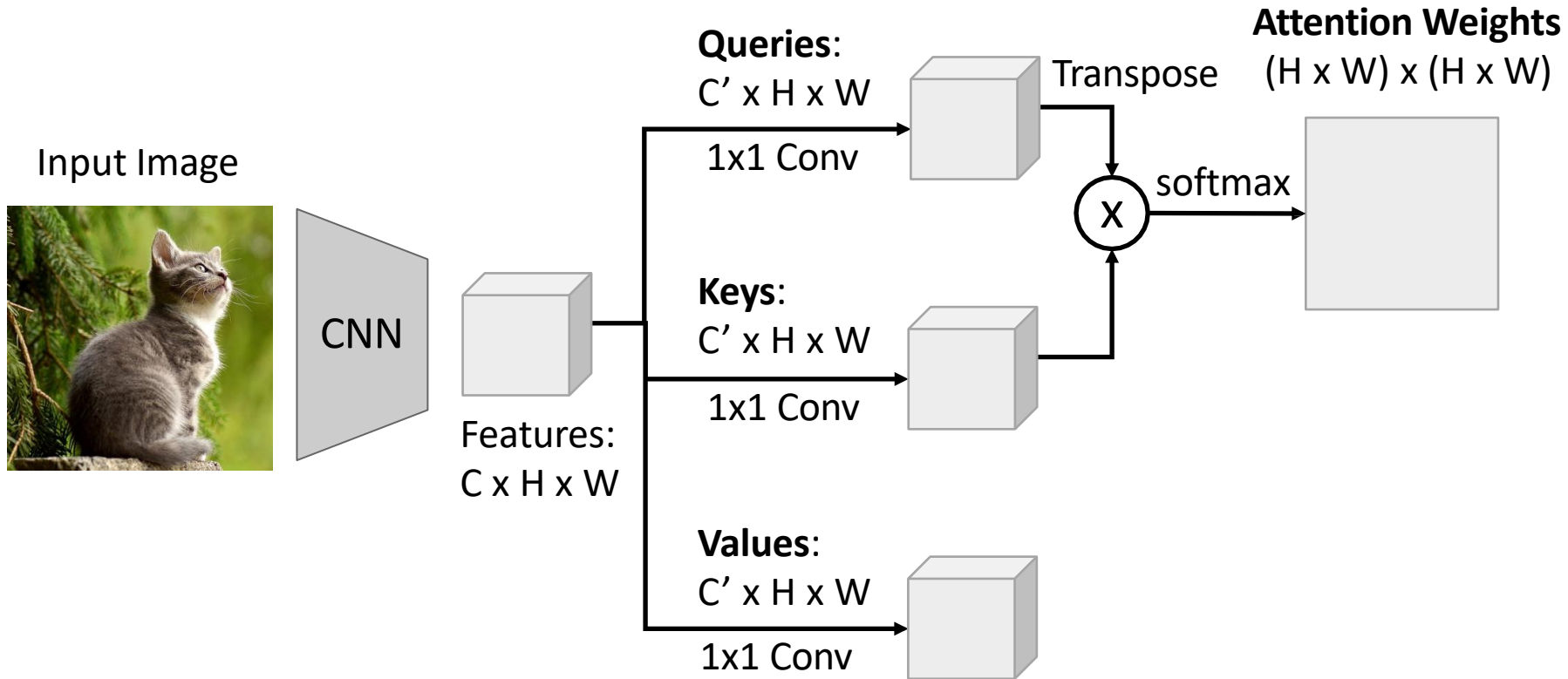


Features:  
 $C \times H \times W$

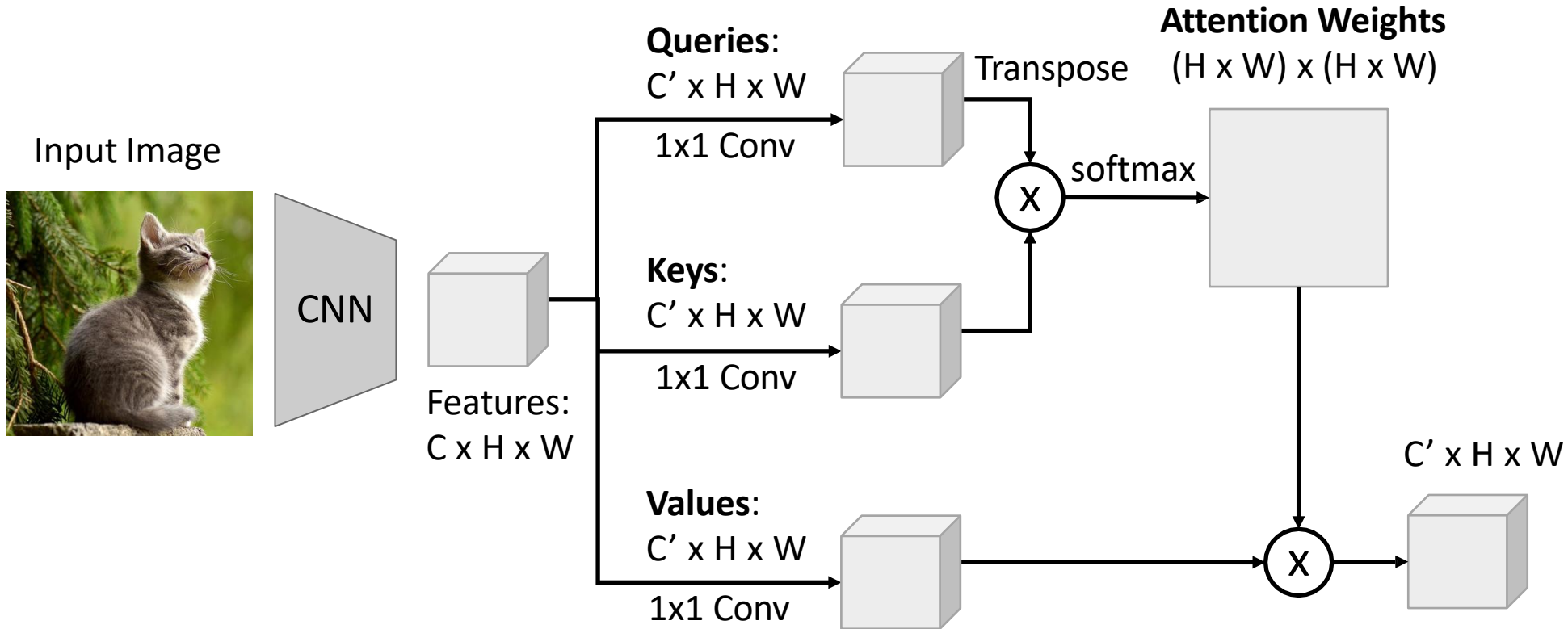
# Example: CNN with Self-Attention



# Example: CNN with Self-Attention

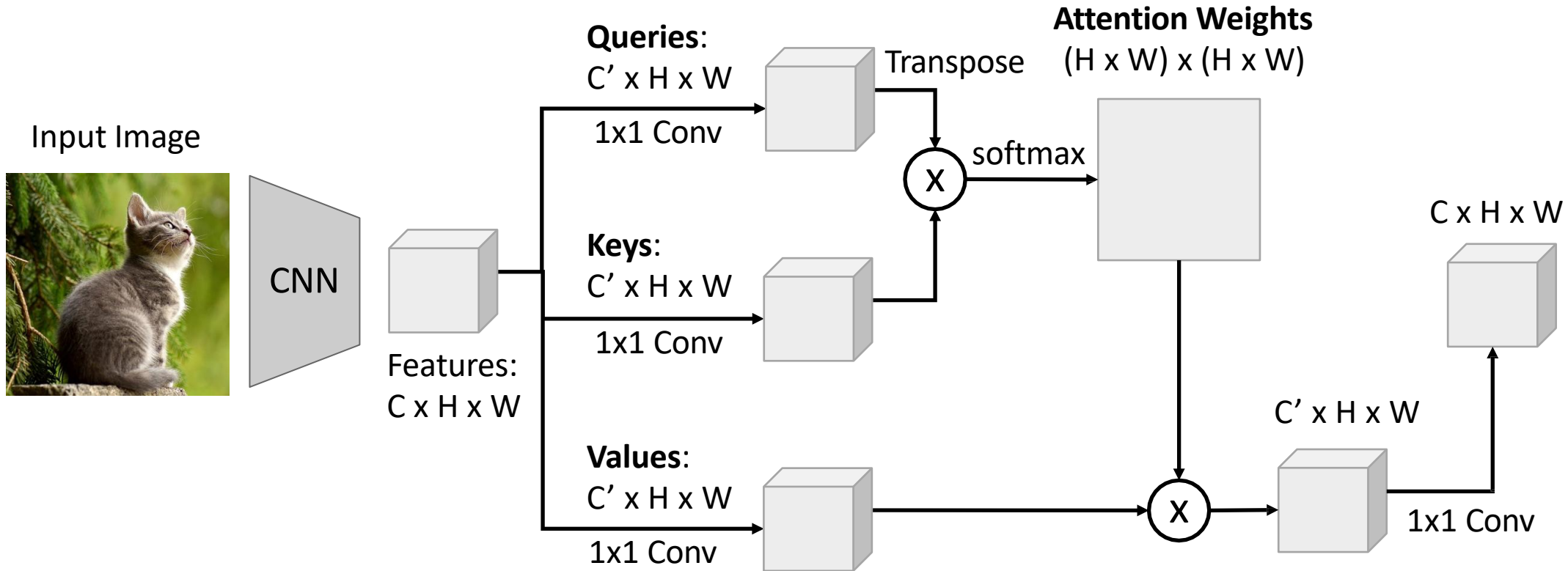


# Example: CNN with Self-Attention

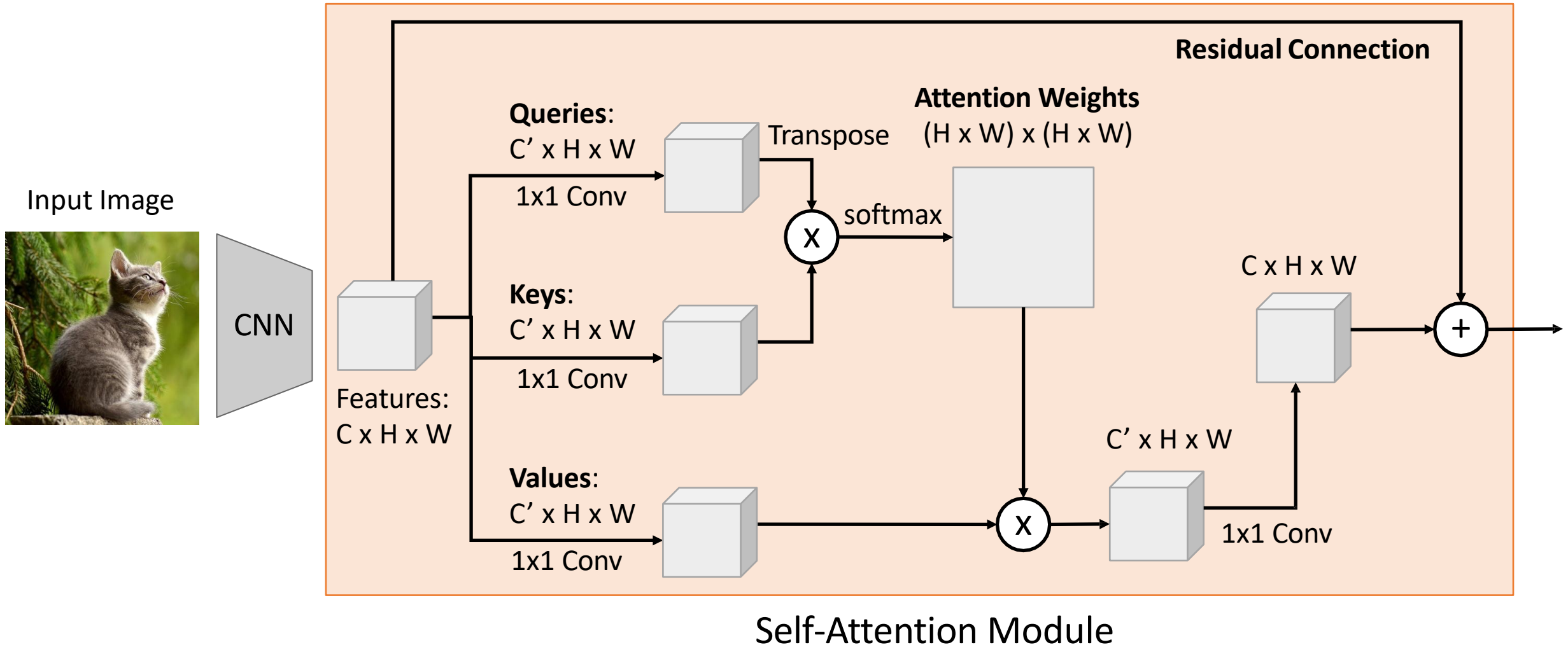




# Example: CNN with Self-Attention

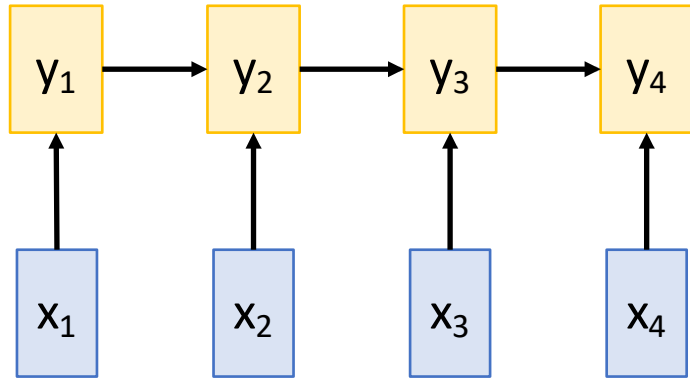


# Example: CNN with Self-Attention



# Three Ways of Processing Sequences

## Recurrent Neural Network



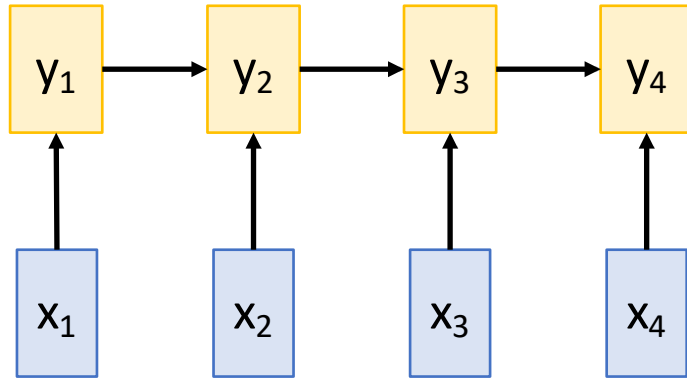
Works on **Ordered Sequences**

(+) **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

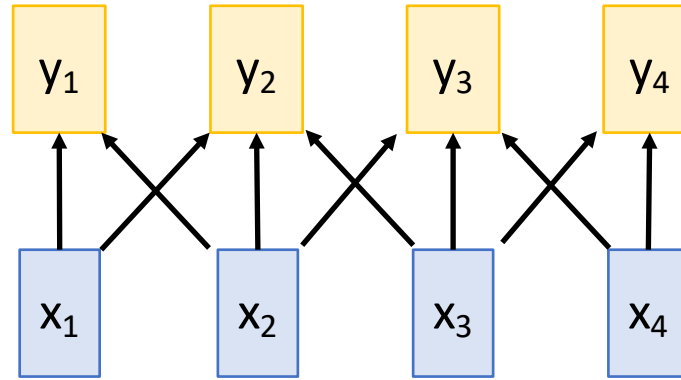


Works on **Ordered Sequences**

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

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

## 1D Convolution



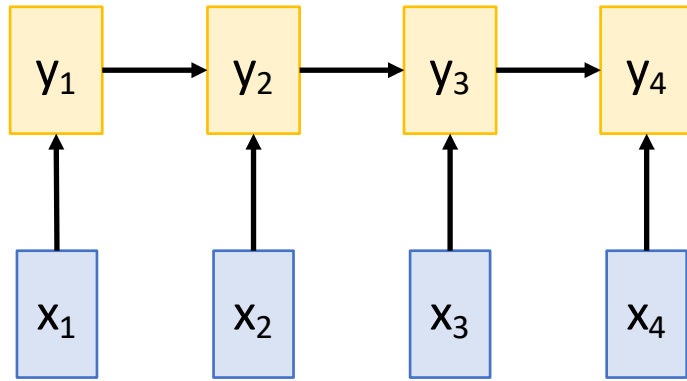
Works on **Multidimensional Grids**

(-) **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

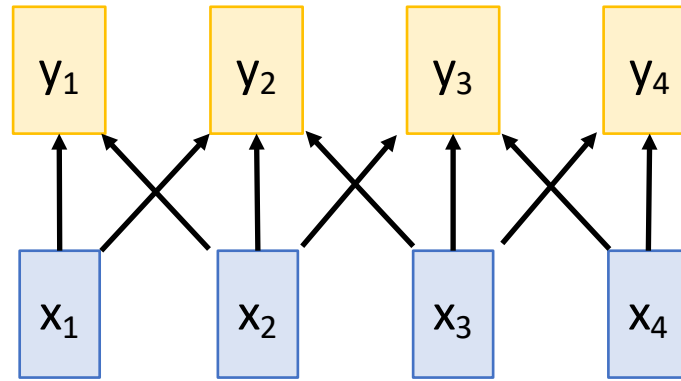


Works on **Ordered Sequences**

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

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

## 1D Convolution

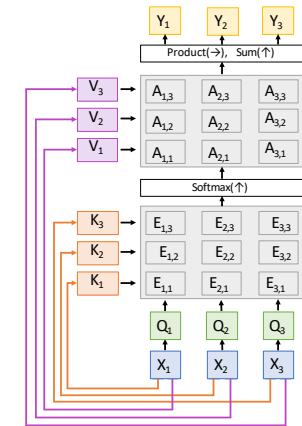


Works on **Multidimensional Grids**

(-) **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

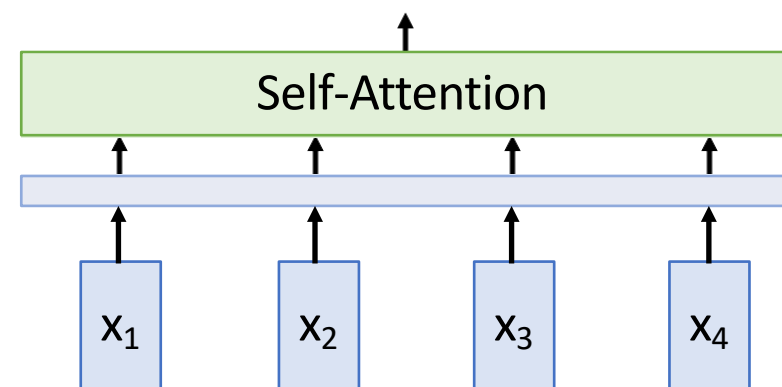
(-) **Very memory intensive**

# The Transformer



# The Transformer

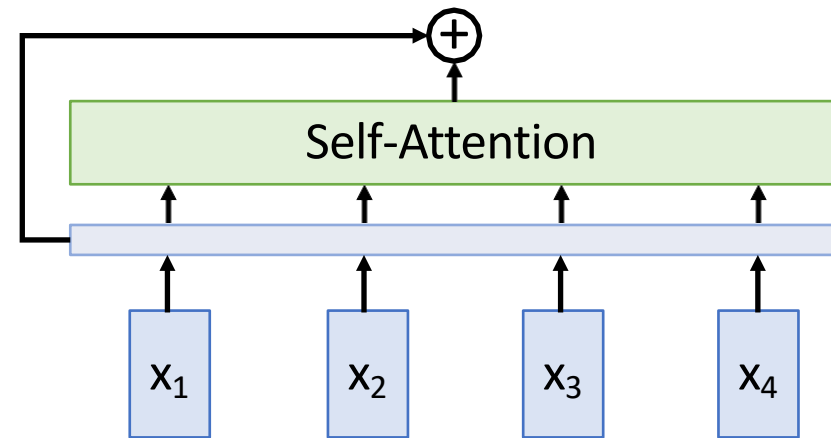
All vectors interact  
with each other





# The Transformer

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)

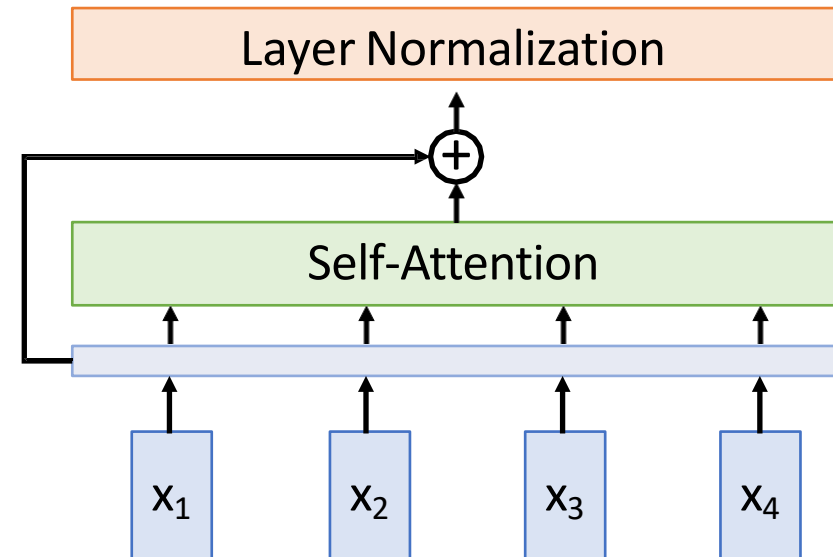
$\sigma_i = (\sum_j (h_{i,j} - \mu_i)^2)^{1/2}$  (scalar)

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

$y_i = \gamma^* z_i + \beta$

Ba et al, 2016

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)

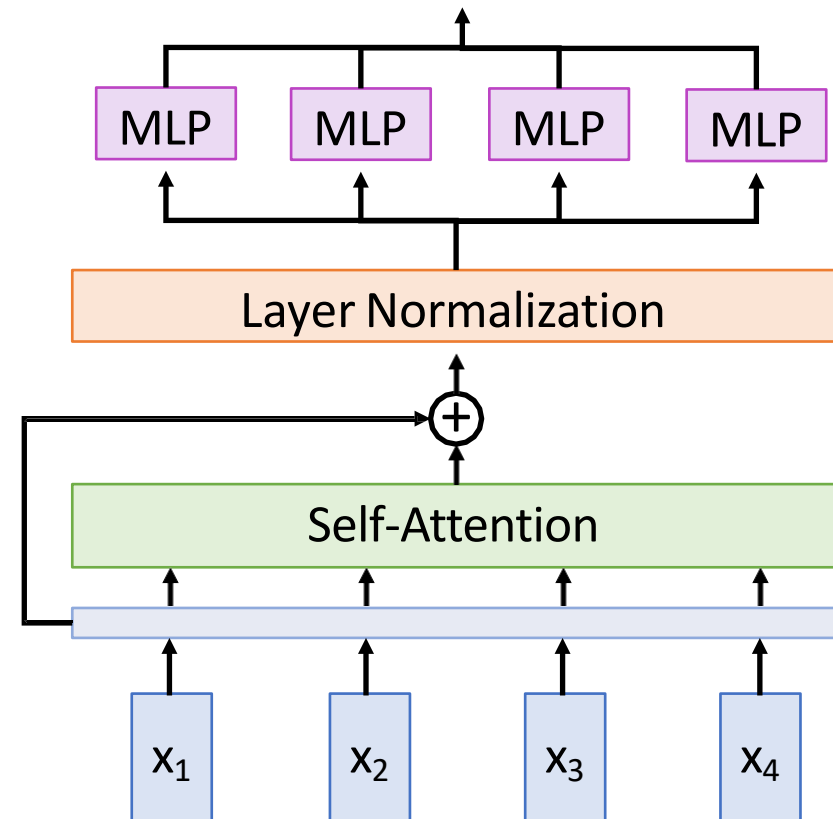
$z_i = (h_i - \mu_i) / \sigma_i$

$y_i = \gamma^* z_i + \beta$

Ba et al, 2016

MLP independently  
on each vector

Residual connection  
All vectors interact  
with each other



# The Transformer

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)

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

$y_i = \gamma^* z_i + \beta$

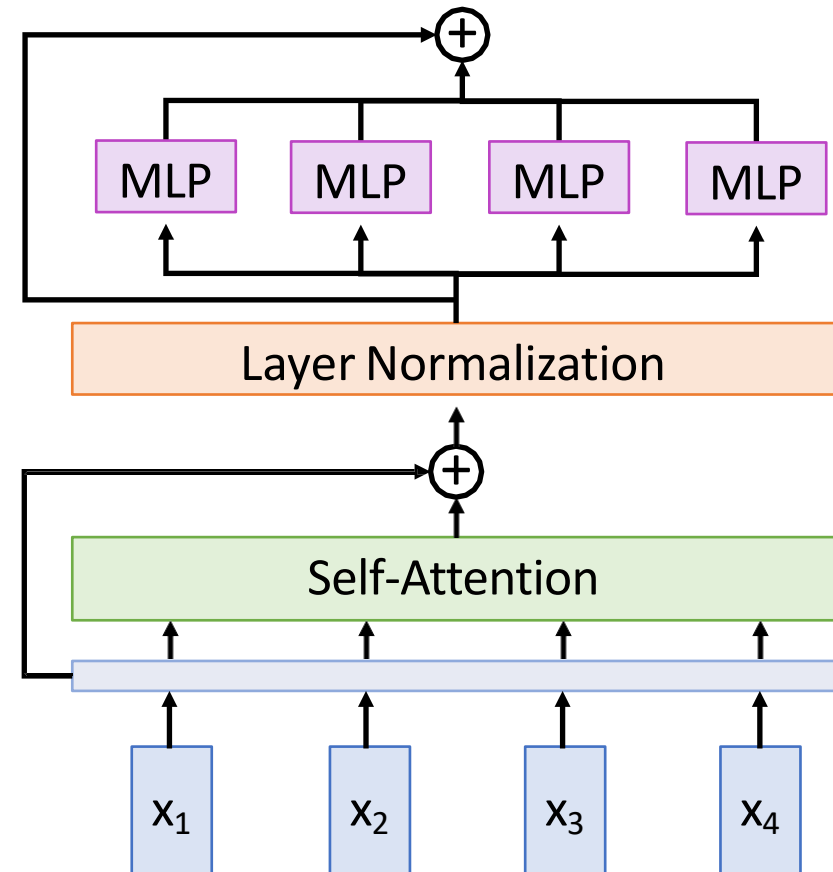
Ba et al, 2016

Residual connection

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)

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

$y_i = \gamma^* z_i + \beta$

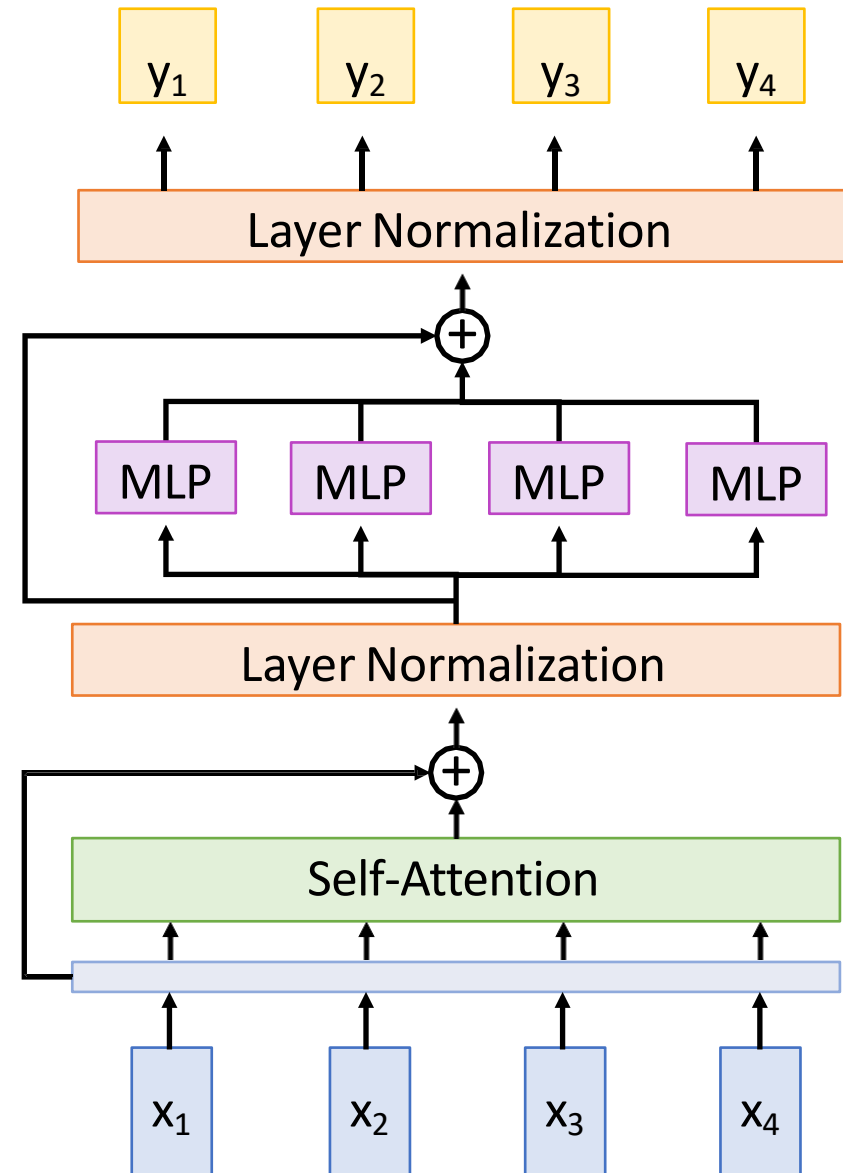
Ba et al, 2016

Residual connection

MLP independently  
on each vector

Residual connection

All vectors interact  
with each other



# The Transformer

## Transformer Block:

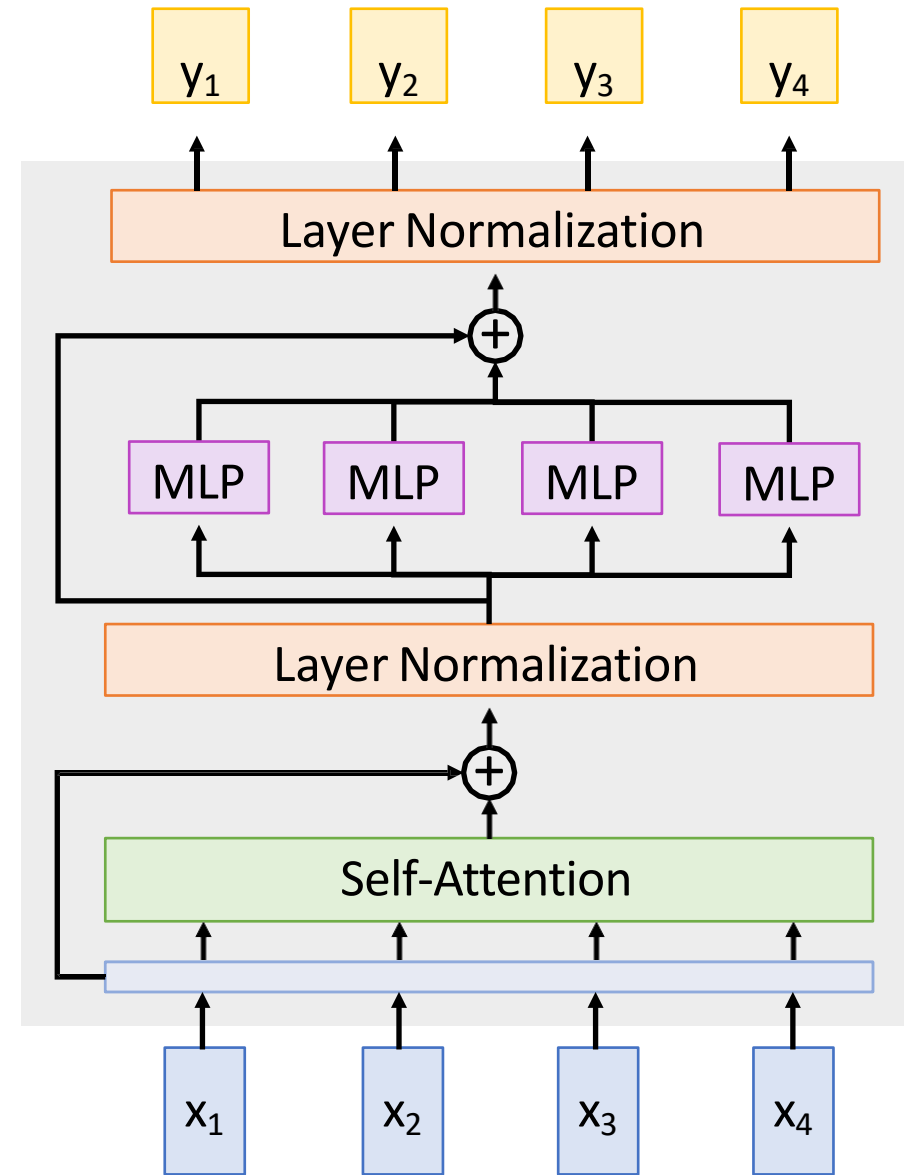
**Input:** Set of vectors  $x$

**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

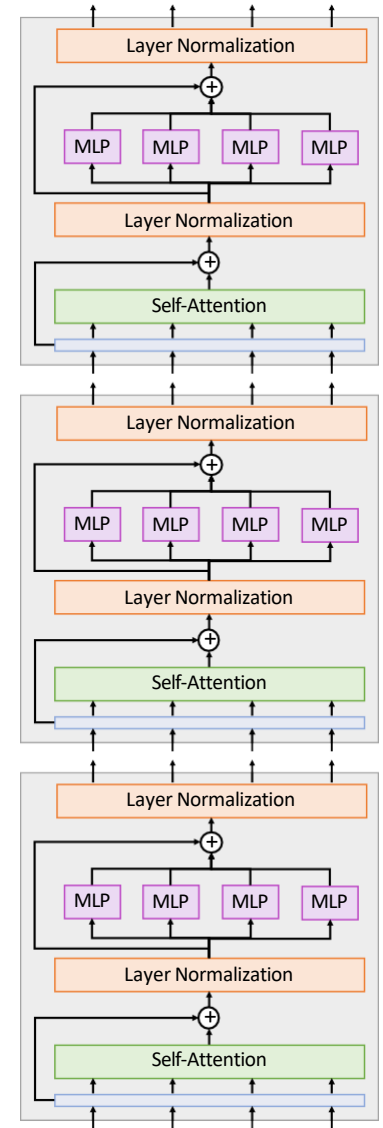
Layer norm and MLP work independently per vector

Highly scalable, highly parallelizable

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

Vaswani et al:

12 blocks,  $D_Q=512$ , 6 heads



# The Transformer: Transfer Learning

“ImageNet Moment for Natural Language Processing”

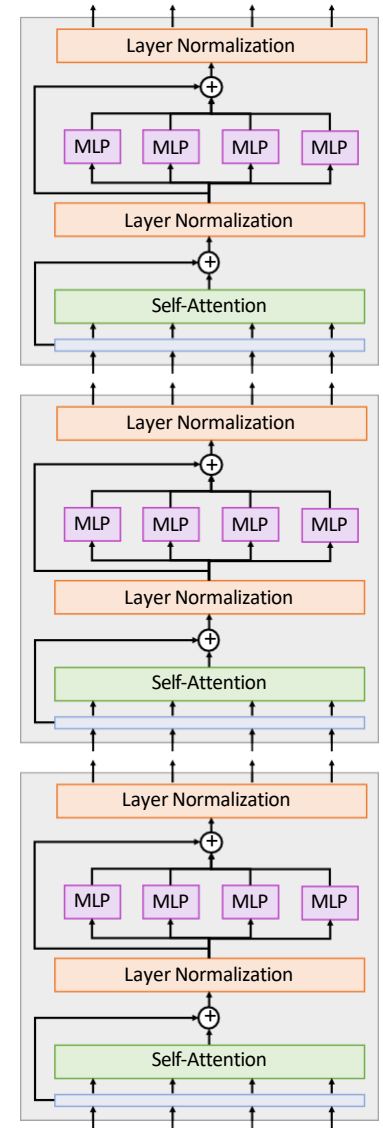
## Pretraining:

Download a lot of text from the internet

Train a giant Transformer model for language modeling

## Finetuning:

Fine-tune the Transformer on your own NLP task





# Scaling up Transformers

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)

# Scaling up Transformers

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)
BERT-Base	12	768	12	110M	13 GB	
BERT-Large	24	1024	16	340M	13 GB	

# Scaling up Transformers

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)
BERT-Base	12	768	12	110M	13 GB	
BERT-Large	24	1024	16	340M	13 GB	
XLNet-Large	24	1024	16	~340M	126 GB	512x TPU-v3 (2.5 days)
RoBERTa	24	1024	16	355M	160 GB	1024x V100 GPU (1 day)

# Scaling up Transformers

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)
BERT-Base	12	768	12	110M	13 GB	
BERT-Large	24	1024	16	340M	13 GB	
XLNet-Large	24	1024	16	~340M	126 GB	512x TPU-v3 (2.5 days)
RoBERTa	24	1024	16	355M	160 GB	1024x V100 GPU (1 day)
GPT-2	12	768	12	117M	40 GB	
GPT-2	24	1024	16	345M	40 GB	
GPT-2	36	1280	20	762M	40 GB	
GPT-2	48	1600	25	1.5B	40 GB	

# Scaling up Transformers

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)
BERT-Base	12	768	12	110M	13 GB	
BERT-Large	24	1024	16	340M	13 GB	
XLNet-Large	24	1024	16	~340M	126 GB	512x TPU-v3 (2.5 days)
RoBERTa	24	1024	16	355M	160 GB	1024x V100 GPU (1 day)
GPT-2	12	768	12	117M	40 GB	
GPT-2	24	1024	16	345M	40 GB	
GPT-2	36	1280	20	762M	40 GB	
GPT-2	48	1600	25	1.5B	40 GB	
Megatron-LM	40	1536	16	1.2B	174 GB	64x V100 GPU
Megatron-LM	54	1920	20	2.5B	174 GB	128x V100 GPU
Megatron-LM	64	2304	24	4.2B	174 GB	256x V100 GPU (10 days)
Megatron-LM	72	3072	32	8.3B	174 GB	512x V100 GPU (9 days)

Shoeybi et al, "Megatron-LM: Training Multi-Billion Parameter Language Models using Model Parallelism", 2019

# Scaling up Transformers

~\$430,000 on Amazon AWS

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)
BERT-Base	12	768	12	110M	13 GB	
BERT-Large	24	1024	16	340M	13 GB	
XLNet-Large	24	1024	16	~340M	126 GB	512x TPU-v3 (2.5 days)
RoBERTa	24	1024	16	355M	160 GB	1024x V100 GPU (1 day)
GPT-2	12	768	12	117M	40 GB	
GPT-2	24	1024	16	345M	40 GB	
GPT-2	36	1280	20	762M	40 GB	
GPT-2	48	1600	25	1.5B	40 GB	
Megatron-LM	40	1536	16	1.2B	174 GB	64x V100 GPU
Megatron-LM	54	1920	20	2.5B	174 GB	128x V100 GPU
Megatron-LM	64	2304	24	4.2B	174 GB	256x V100 GPU (10 days)
Megatron-LM	72	3072	32	8.3B	174 GB	512x V100 GPU (9 days)

# Scaling up Transformers

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)
BERT-Base	12	768	12	110M	13 GB	
BERT-Large	24	1024	16	340M	13 GB	
XLNet-Large	24	1024	16	~340M	126 GB	512x TPU-v3 (2.5 days)
RoBERTa	24	1024	16	355M	160 GB	1024x V100 GPU (1 day)
GPT-2	12	768	12	117M	40 GB	
GPT-2	24	1024	16	345M	40 GB	
GPT-2	36	1280	20	762M	40 GB	
GPT-2	48	1600	25	1.5B	40 GB	
Megatron-LM	40	1536	16	1.2B	174 GB	64x V100 GPU
Megatron-LM	54	1920	20	2.5B	174 GB	128x V100 GPU
Megatron-LM	64	2304	24	4.2B	174 GB	256x V100 GPU (10 days)
Megatron-LM	72	3072	32	8.3B	174 GB	512x V100 GPU (9 days)
GPT-3	96	12288	96	175B	570 GB	256xV100 (3 months)

# Scaling up Transformers

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)
BERT-Base	12	768	12	110M	13 GB	
BERT-Large	24	1024	16	340M	13 GB	
XLNet-Large	24	1024	16	~340M	126 GB	512x TPU-v3 (2.5 days)
RoBERTa	24	1024	16	355M	160 GB	1024x V100 GPU (1 day)
GPT-2	12	768	12	117M	40 GB	
GPT-2	24	1024	16	345M	40 GB	
GPT-2	36	1280	20	762M	40 GB	
GPT-2	48	1600	25	1.5B	40 GB	
Megatron-LM	40	1536	16	1.2B	174 GB	64x V100 GPU
Megatron-LM	54	1920	20	2.5B	174 GB	128x V100 GPU
Megatron-LM	64	2304	24	4.2B	174 GB	256x V100 GPU (10 days)
Megatron-LM	72	3072	32	8.3B	174 GB	512x V100 GPU (9 days)
GPT-3	96	12288	96	175B	570 GB	256xV100 (3 months)
Megatron-Turing NLG	105	20480	128	530B	>1TB	280xA100 (6 months)



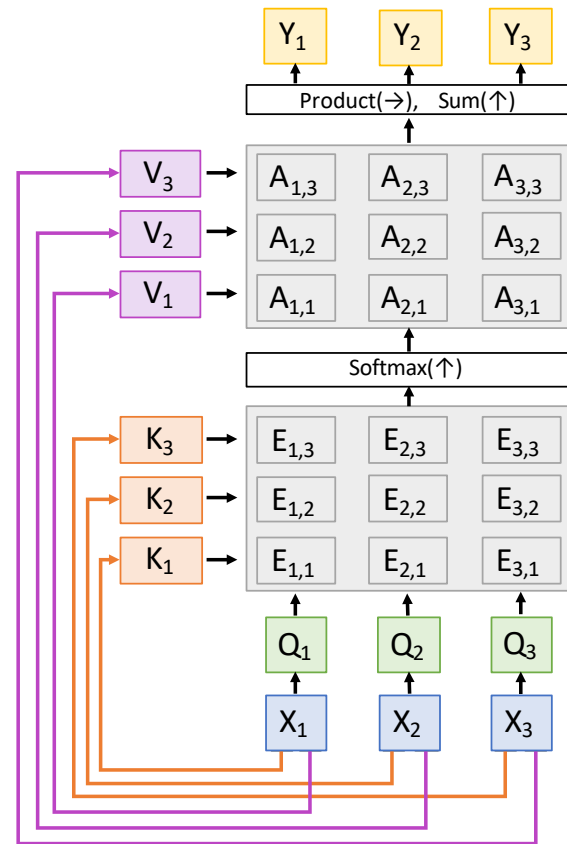
# Summary

Adding **Attention** to RNN models lets them look at different parts of the input at each timestep

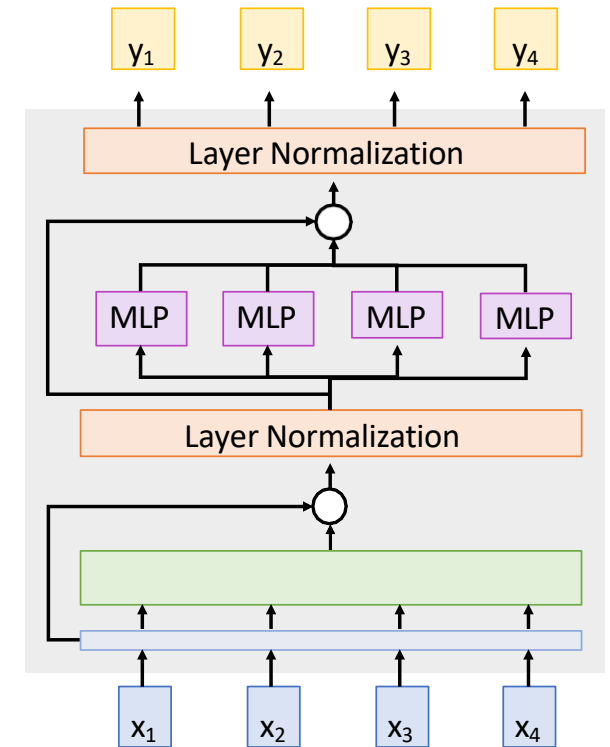


A dog is standing on a hardwood floor.

Generalized **Self-Attention** is a powerful neural network primitive



**Transformers** are a neural network model that only uses attention



# Scaling up Transformers

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)
BERT-Base	12	768	12	110M	13 GB	
BERT-Large	24	1024	16	340M	13 GB	
XLNet-Large	24	1024	16	~340M	126 GB	512x TPU-v3 (2.5 days)
RoBERTa	24	1024	16	355M	160 GB	1024x V100 GPU (1 day)
GPT-2	12	768	12	117M	40 GB	
GPT-2	24	1024	16	345M	40 GB	
GPT-2	36	1280	20	762M	40 GB	
GPT-2	48	1600	25	1.5B	40 GB	
Megatron-LM	40	1536	16	1.2B	174 GB	64x V100 GPU
Megatron-LM	54	1920	20	2.5B	174 GB	128x V100 GPU
Megatron-LM	64	2304	24	4.2B	174 GB	256x V100 GPU (10 days)
Megatron-LM	72	3072	32	8.3B	174 GB	512x V100 GPU (9 days)
GPT-3	96	12288	96	175B	570 GB	256xV100 (3 months)
GPT-4 (?)	150	20480	160	500B	5TB	1024xA100 (6 months)