

Lecture 13: Attention

Reminder: Assignment 4

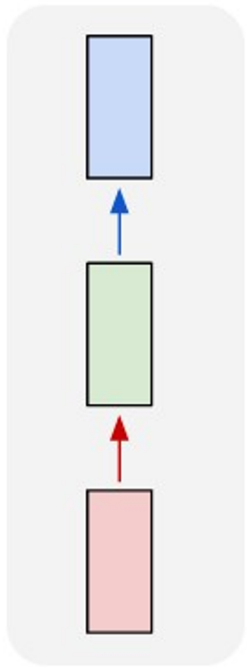
- Assignment 4 is released:
<https://web.eecs.umich.edu/~justincj/teaching/eecs498/FA2020/assignment4.html>
- Due Friday October 30, 11:59pm EDT
 - Two weeks from Friday! Feel free to start after midterm
- Lots of fun topics:
 - PyTorch Autograd
 - Recurrent networks
 - Attention
 - Network visualization: saliency maps, adversarial examples, feature inversion
 - Artistic style transfer

Reminder: Midterm

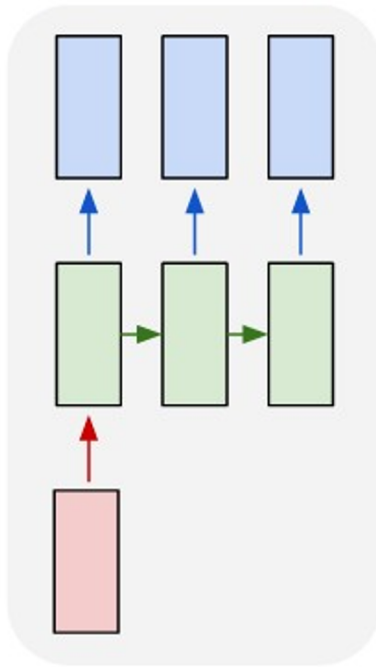
- Monday, October 19
- Will be online via <https://crabster.org/> Gradescope
- Exam is 90 minutes
- You can take it any time in a 24-hour window
- We will have 3-4 “on-call” periods during the 24-hour window where GSIs will answer questions within ~15 minutes
- Open note
- True / False, multiple choice, short answer
- For short answer questions requiring math, either write LaTeX or upload an image with handwritten math
- **Send SSD accommodations if you have them!**

Last Time: Recurrent Neural Networks

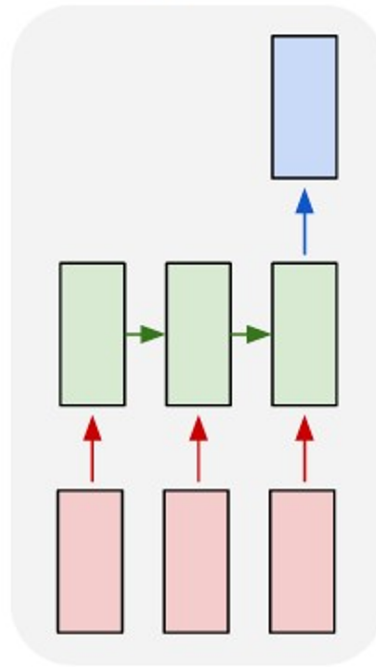
one to one



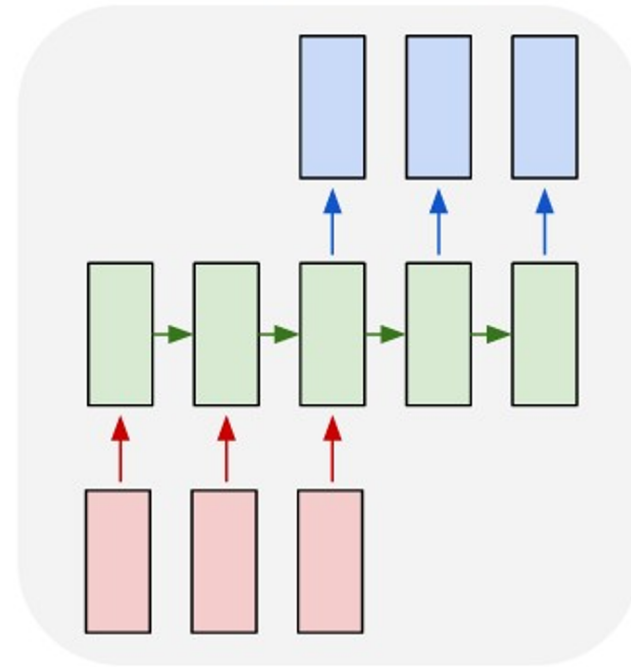
one to many



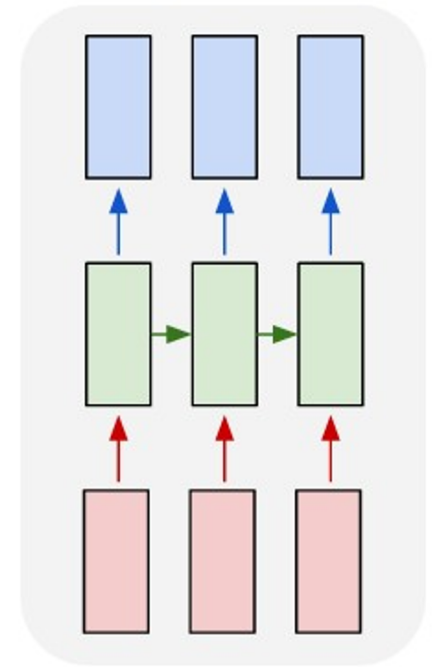
many to one



many to many



many to many

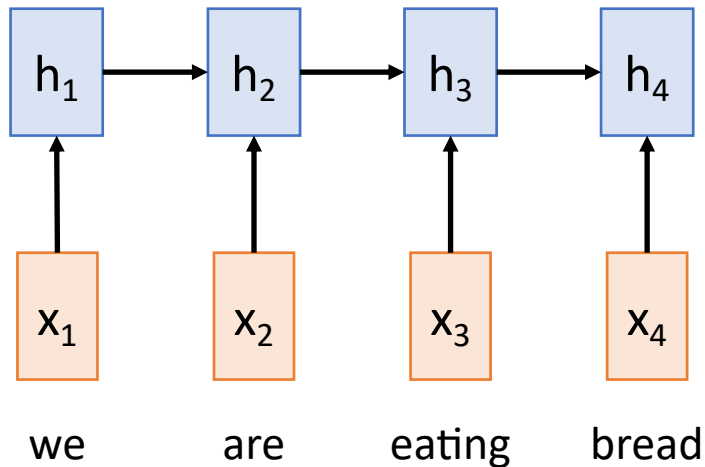


Sequence-to-Sequence with RNNs

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

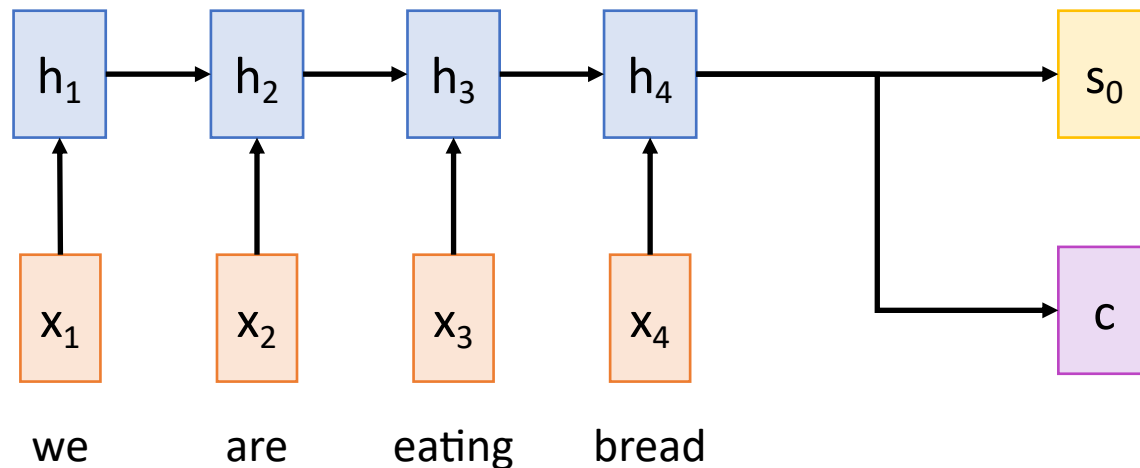
Output: Sequence $y_1, \dots, y_{T'}$

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

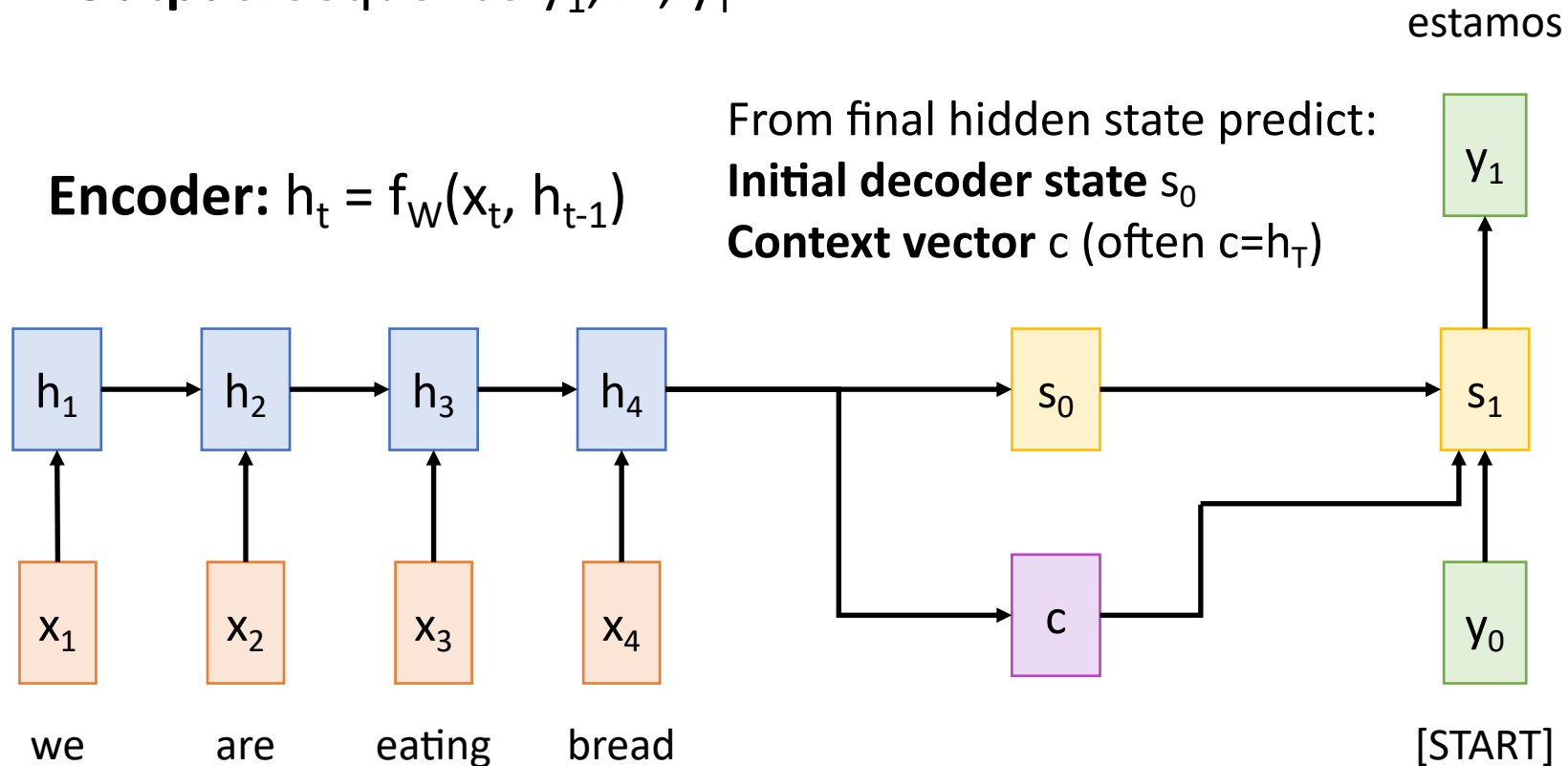
Input: Sequence x_1, \dots, x_T

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

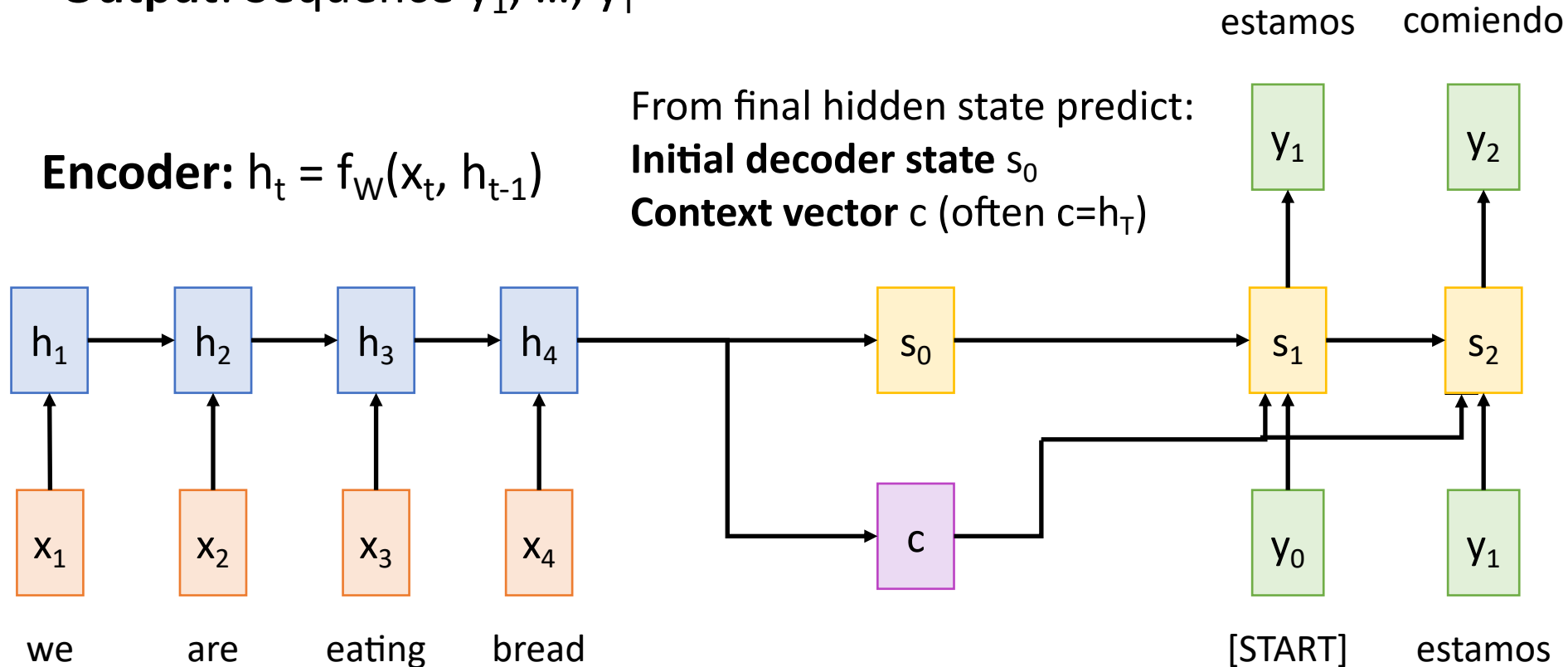
Input: Sequence x_1, \dots, x_T

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

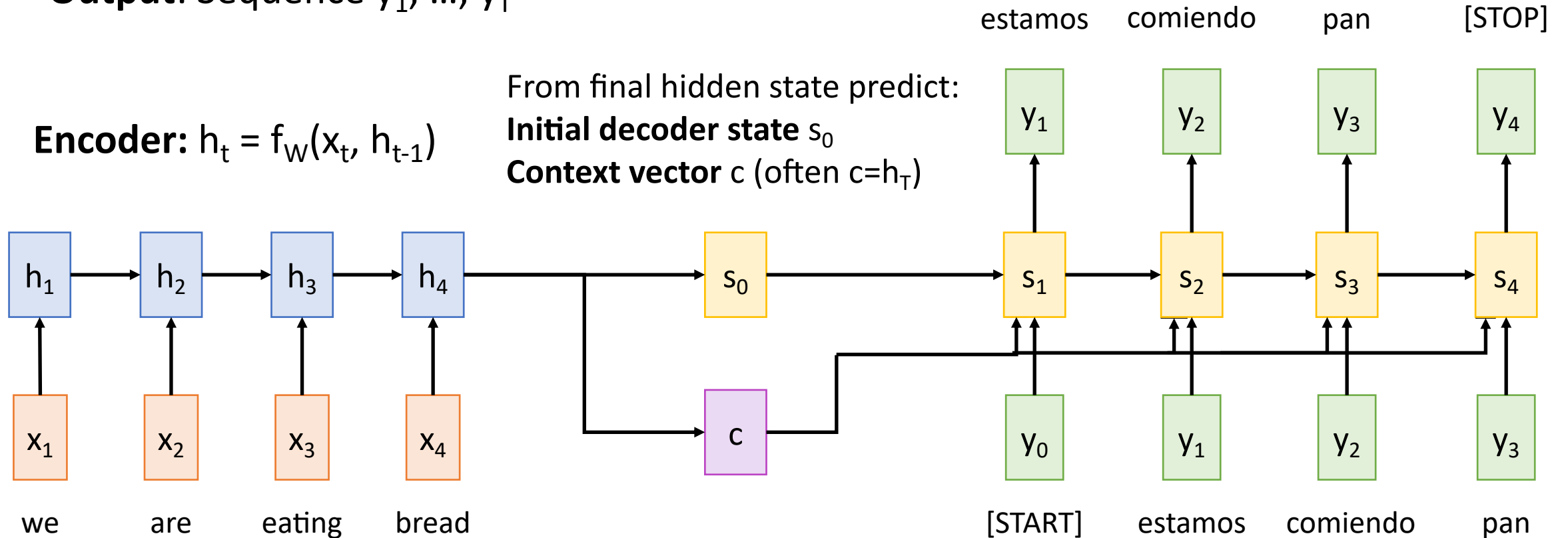
Input: Sequence x_1, \dots, x_T

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 $c=h_T$)



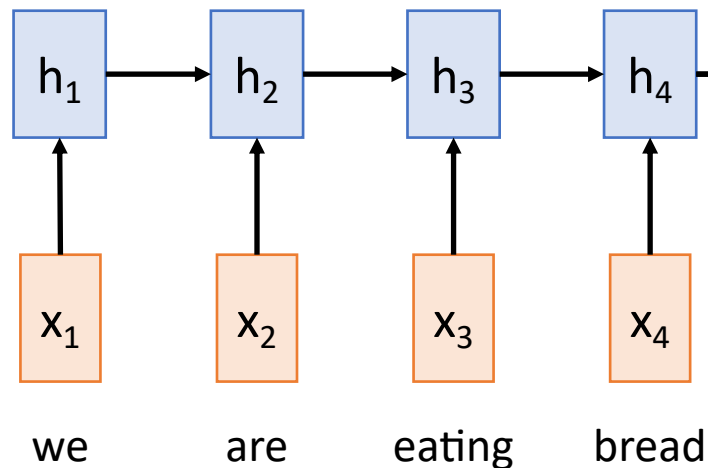
Sequence-to-Sequence with RNNs

Input: Sequence x_1, \dots, x_T

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

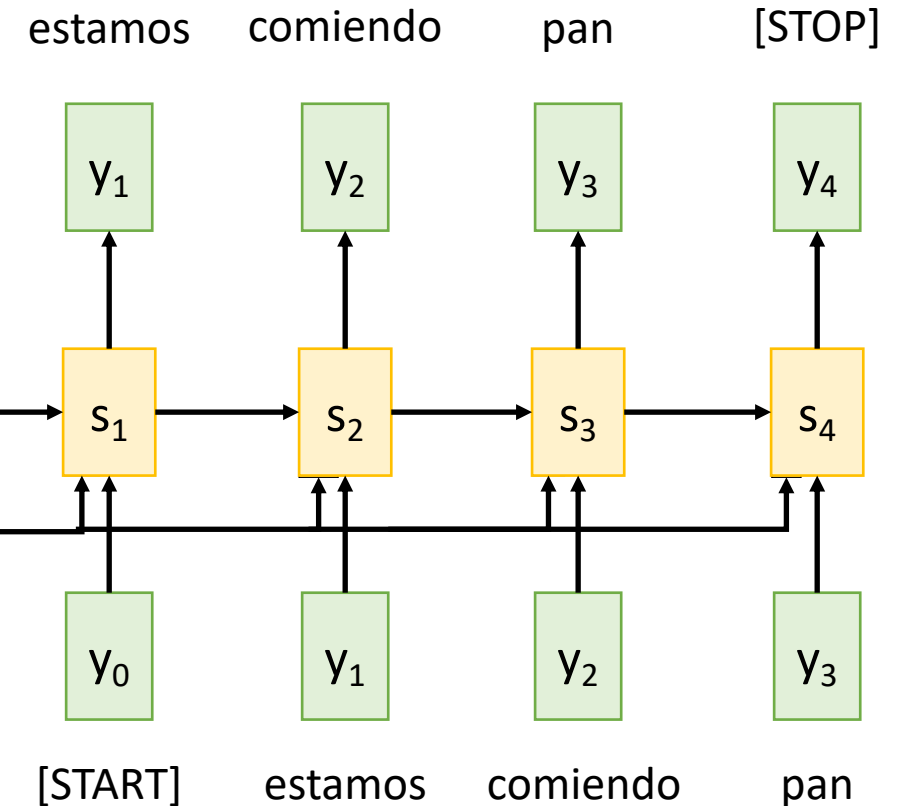
Decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c)$

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



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

Problem: Input sequence bottlenecked through fixed-sized vector. What if $T=1000$?



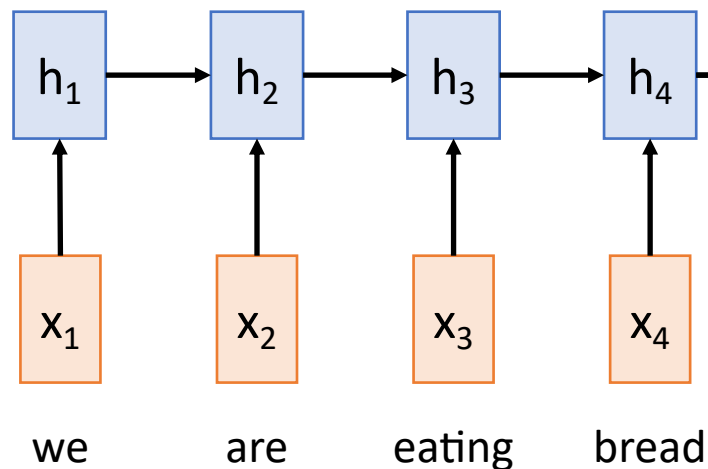
Sequence-to-Sequence with RNNs

Input: Sequence x_1, \dots, x_T

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

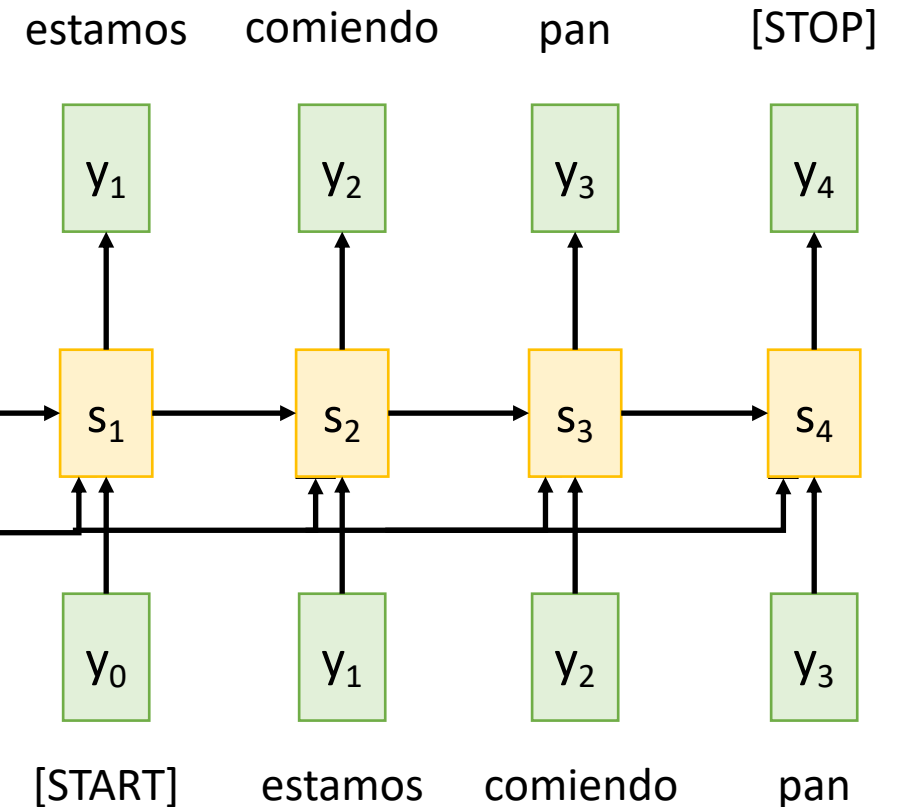
Decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c)$

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



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

Problem: Input sequence bottlenecked through fixed-sized vector. What if $T=1000$?



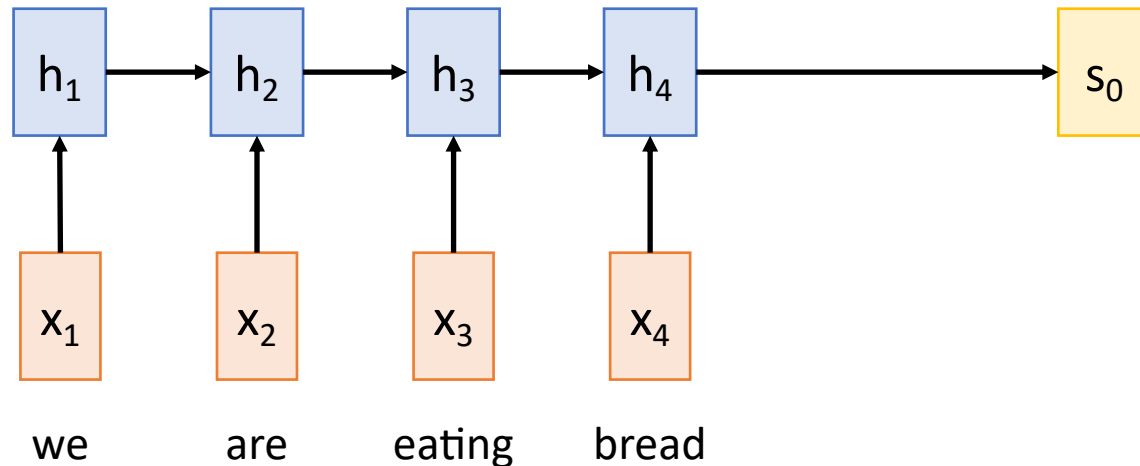
Idea: use new context vector at each step of decoder!

Sequence-to-Sequence with RNNs and Attention

Input: Sequence x_1, \dots, x_T

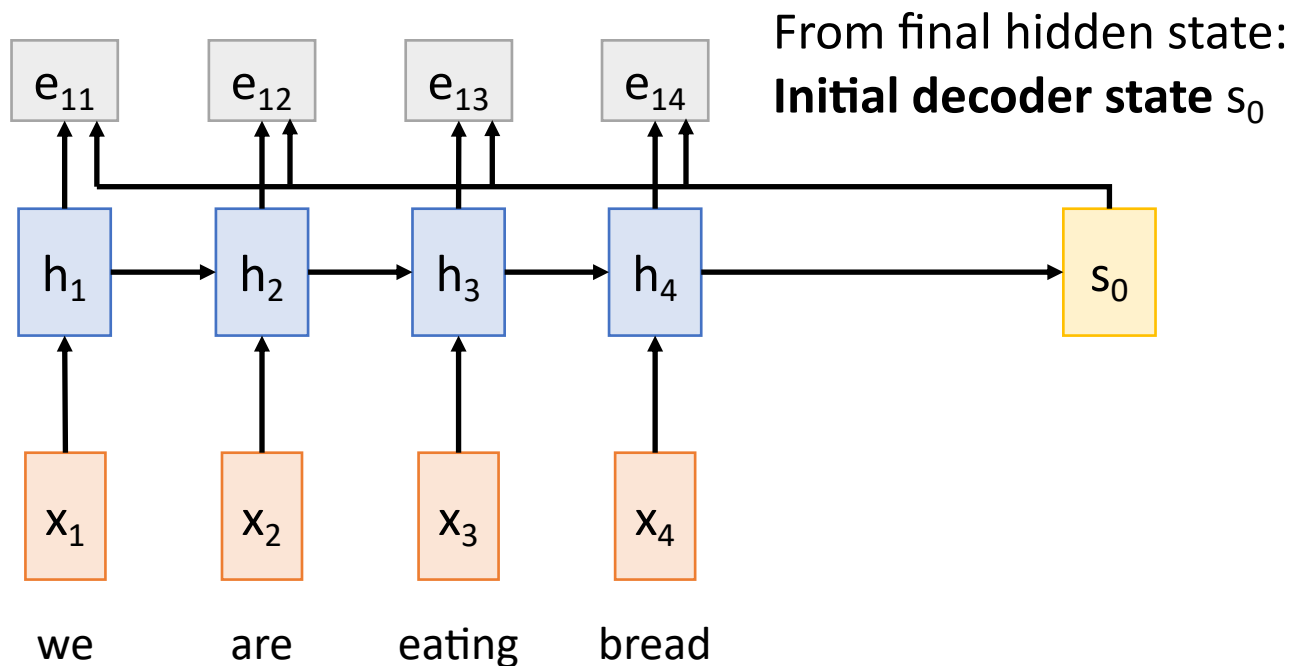
Output: Sequence $y_1, \dots, y_{T'}$

Encoder: $h_t = f_W(x_t, h_{t-1})$ From final hidden state:
Initial decoder state s_0

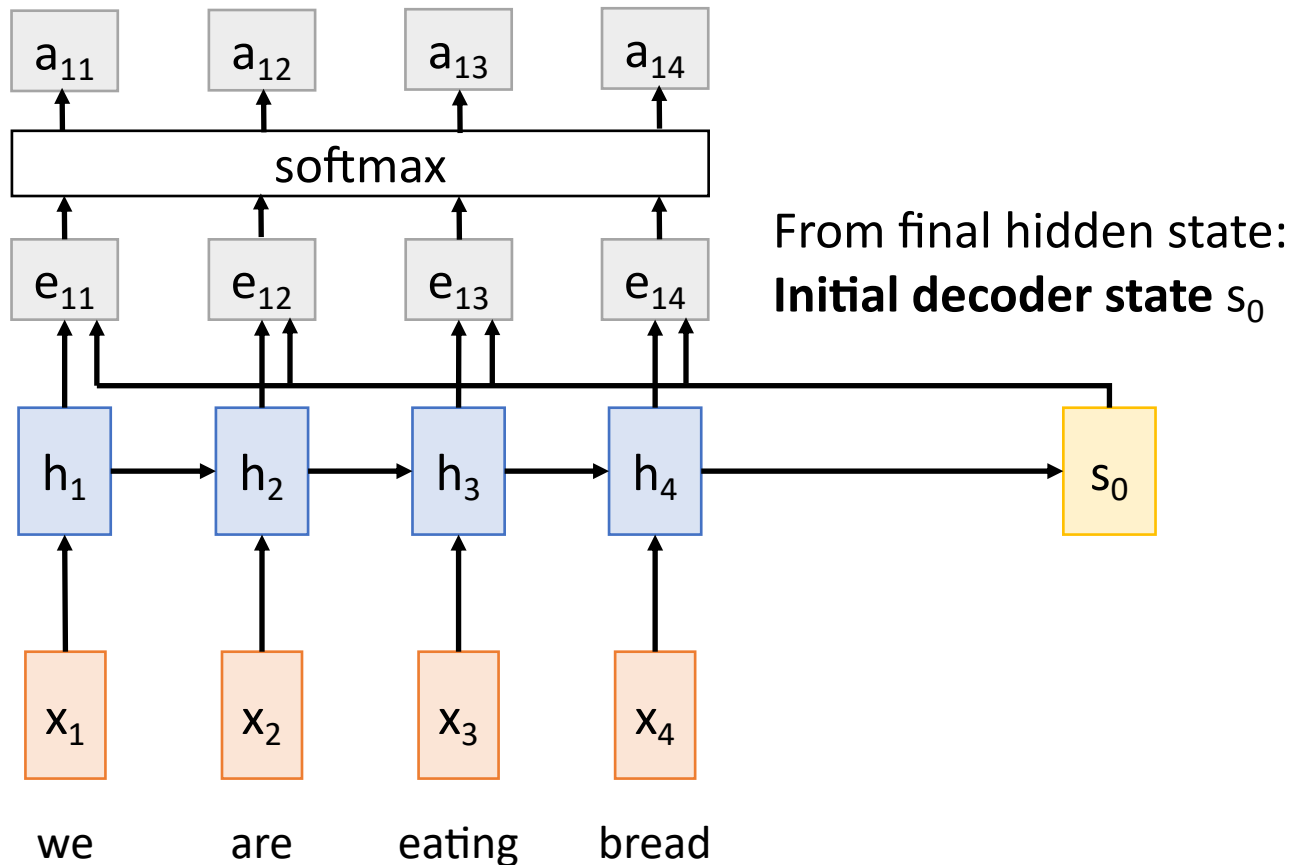


Sequence-to-Sequence with RNNs and Attention

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



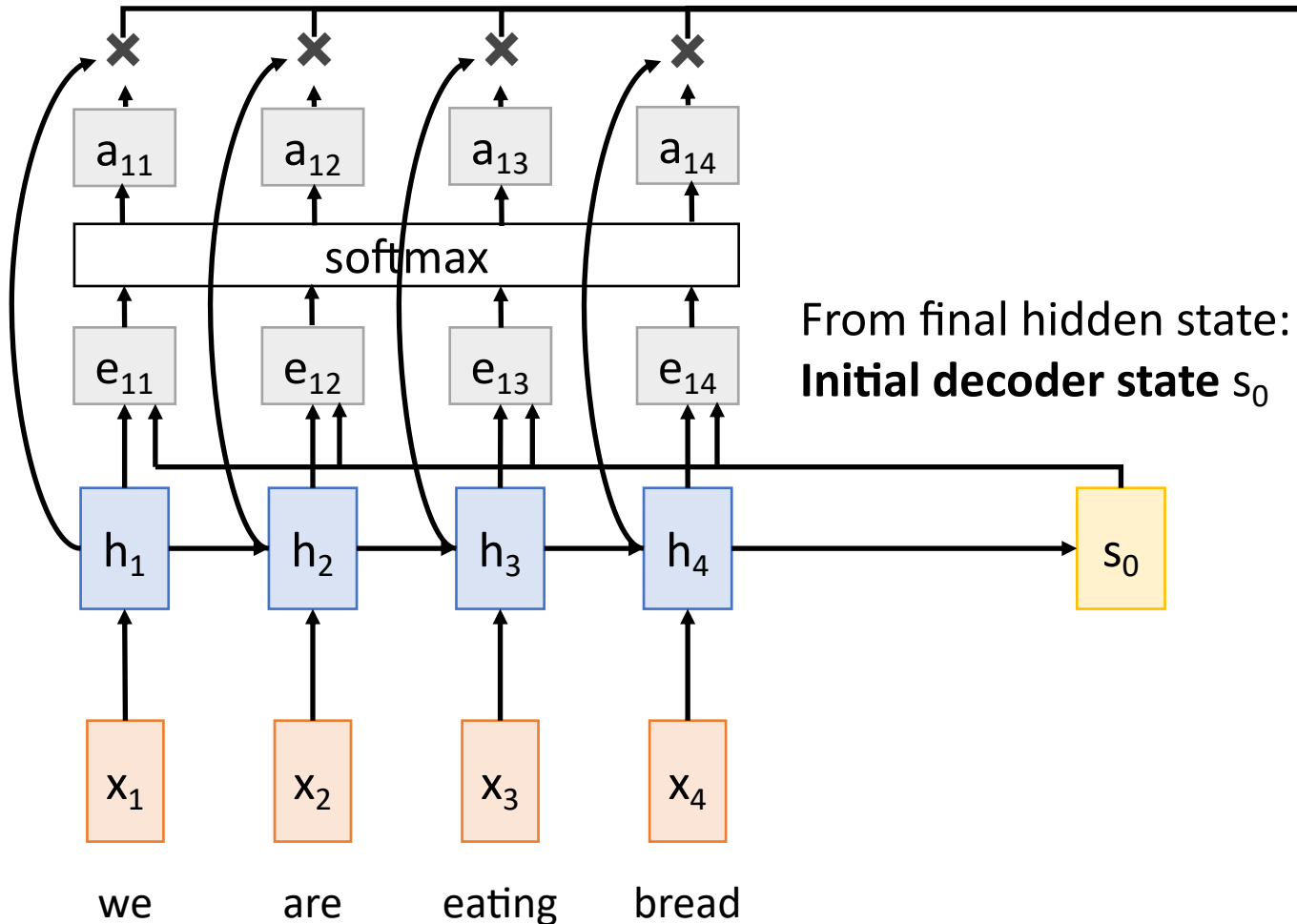
Sequence-to-Sequence with RNNs and Attention



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

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

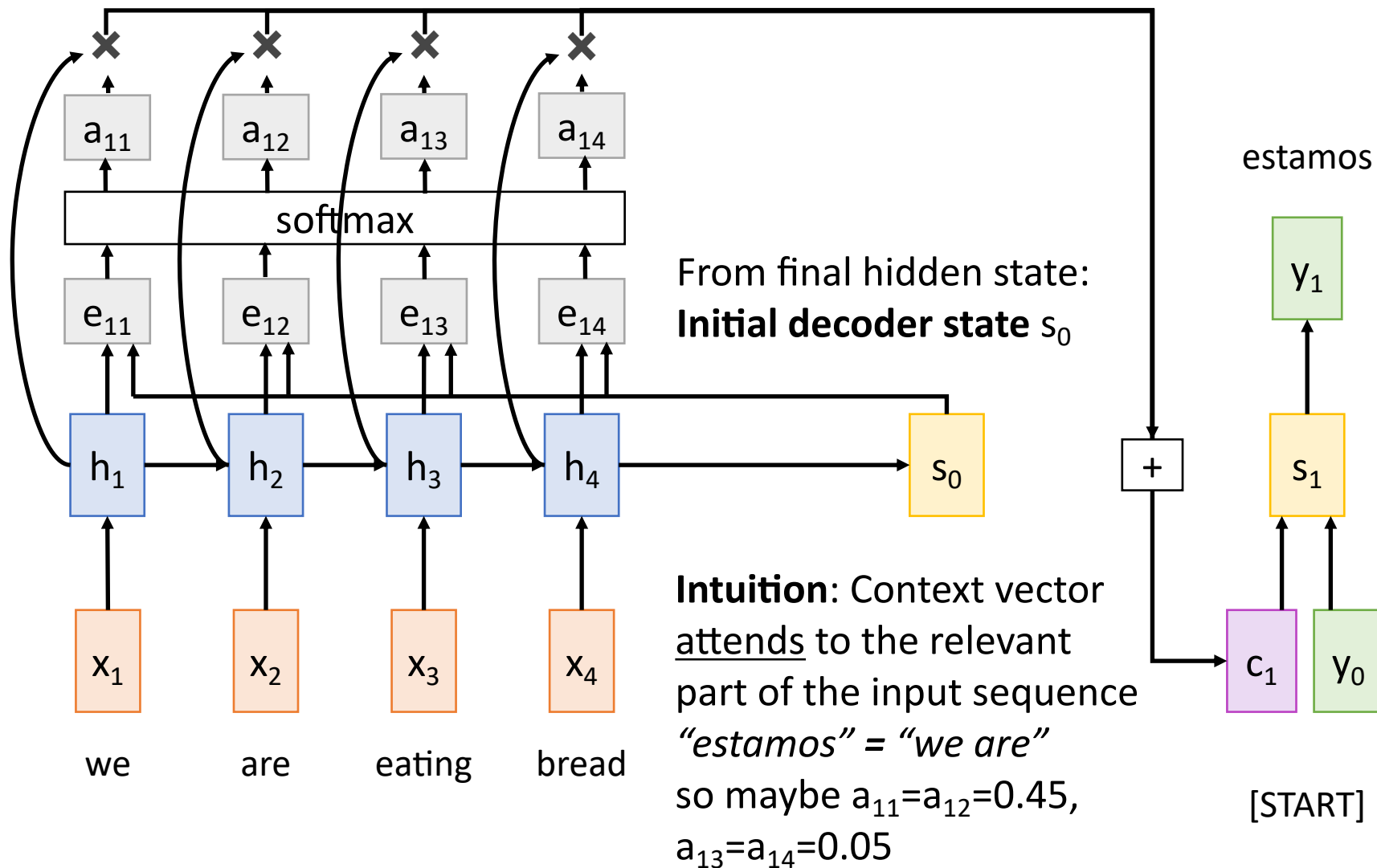
Compute context vector as linear
combination of hidden states
 $c_t = \sum_i a_{t,i} h_i$

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

[START]

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

Sequence-to-Sequence with RNNs and Attention



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

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

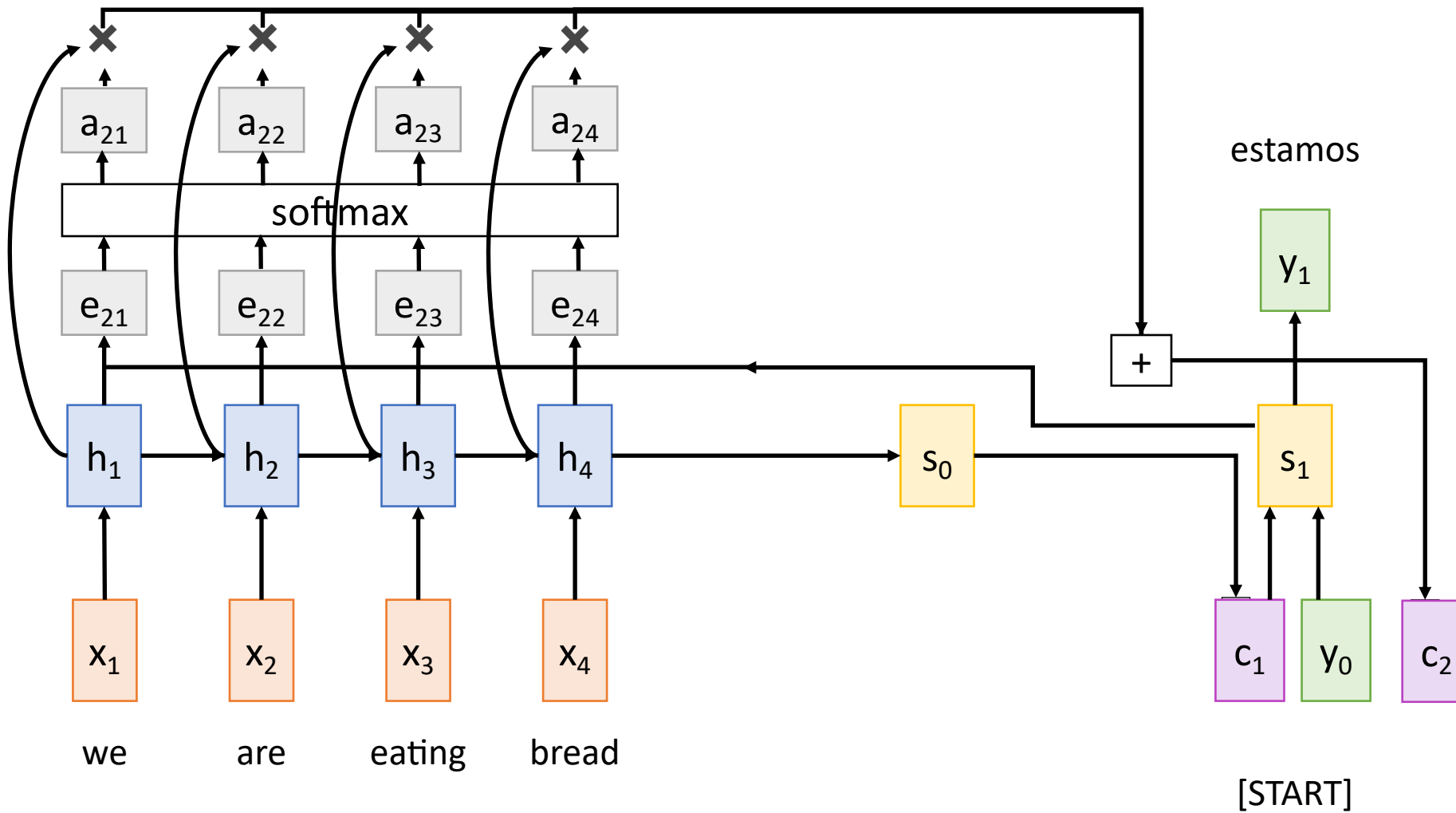
Compute context vector as linear
 combination of hidden states
 $c_t = \sum_i a_{t,i} h_i$

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

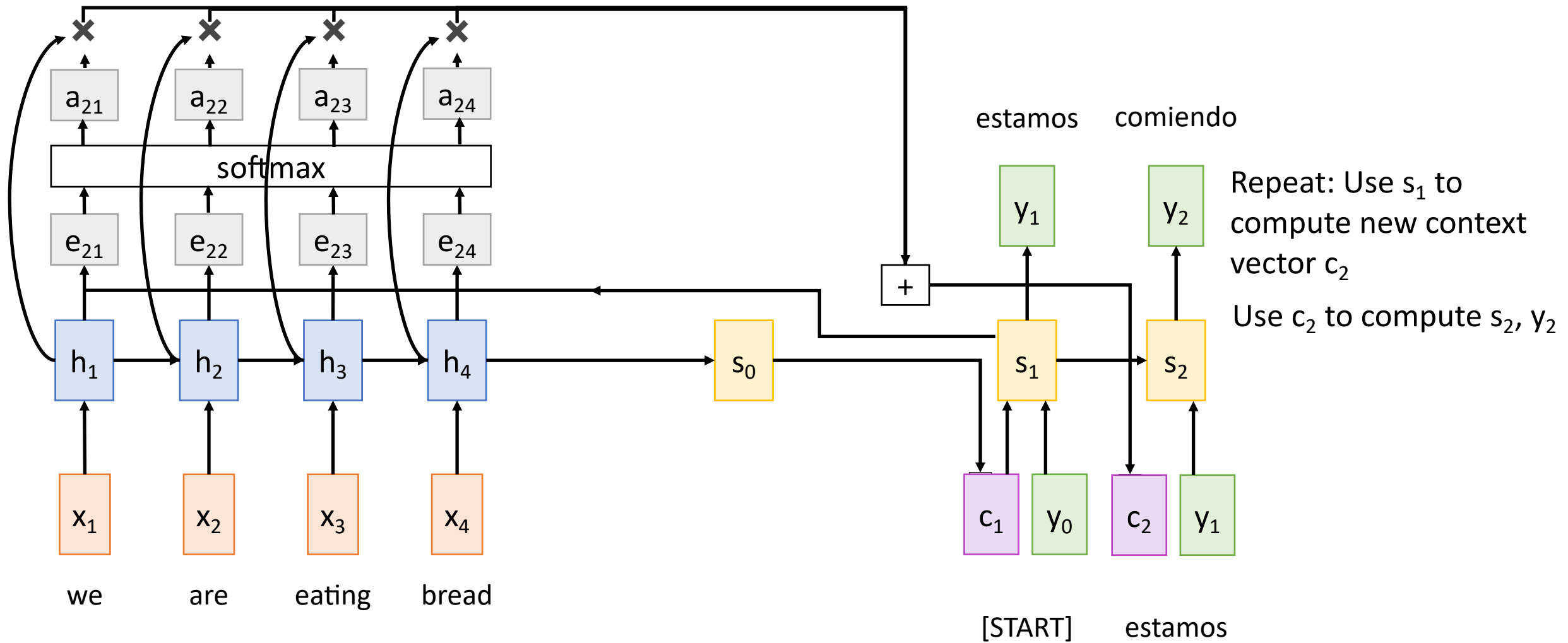
**This is all differentiable! Do not
 supervise attention weights –
 backprop through everything**

Sequence-to-Sequence with RNNs

Repeat: Use s_1 to compute new context vector c_2

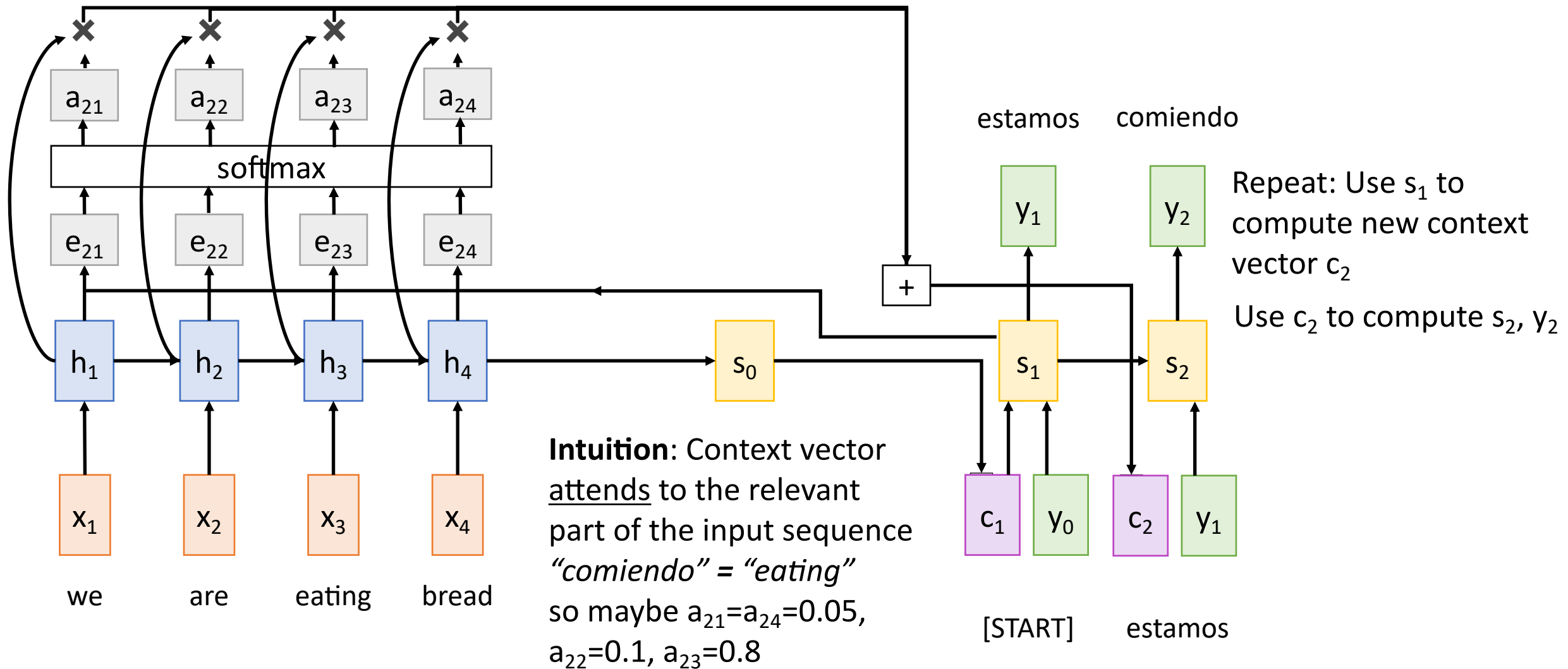


Sequence-to-Sequence with RNNs and Attention



Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Sequence-to-Sequence with RNNs and Attention

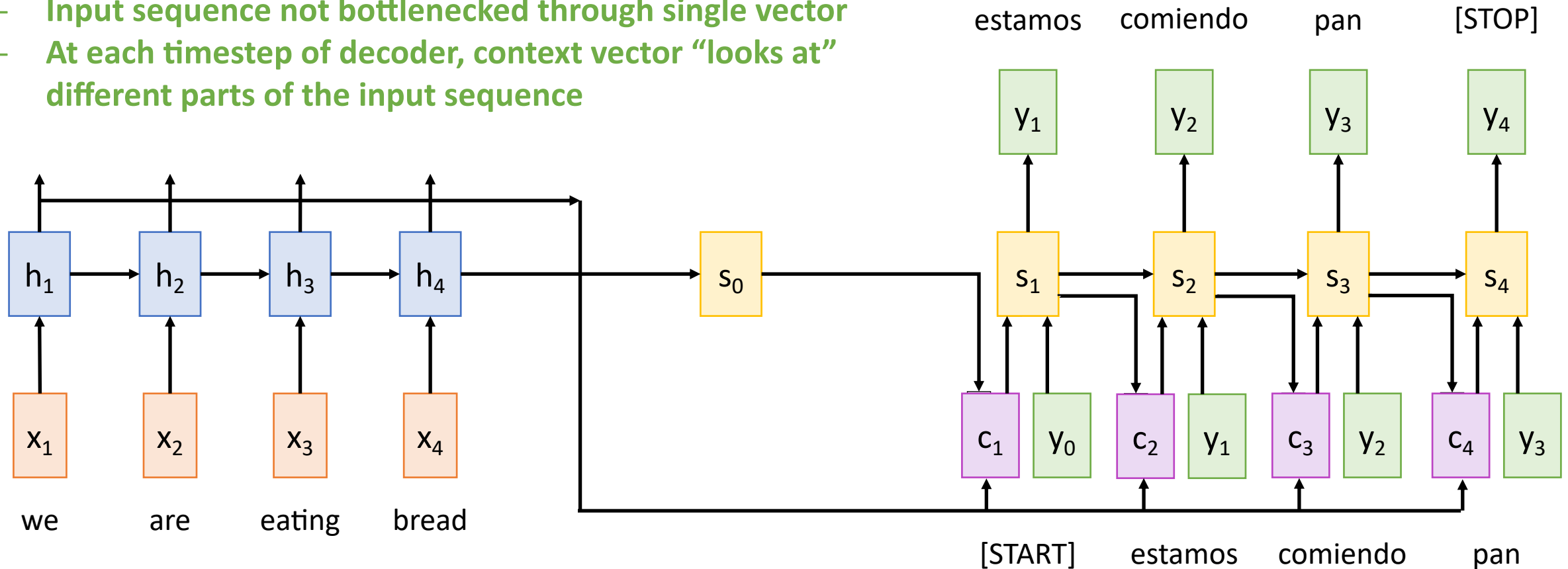


Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

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



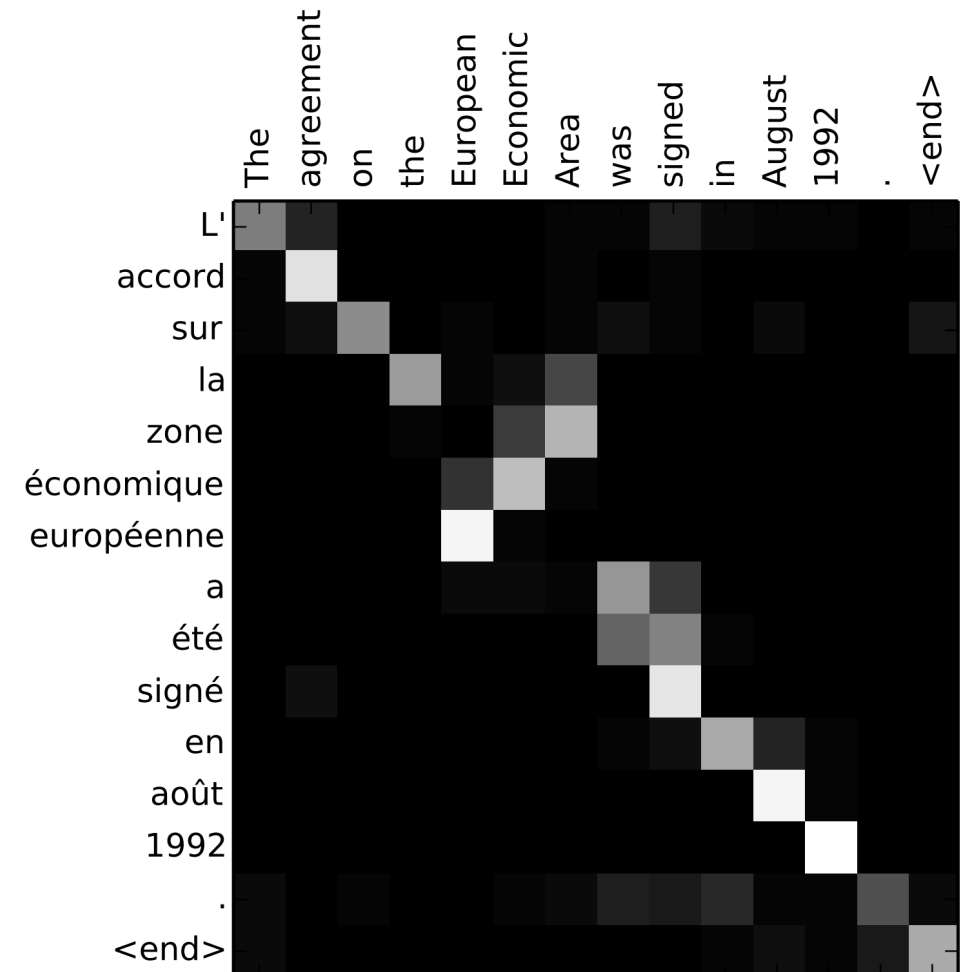
Sequence-to-Sequence with RNNs and Attention

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 $a_{t,i}$



Bahdanau et al, “Neural machine translation by jointly learning to align and translate”, ICLR 2015

Sequence-to-Sequence with RNNs and Attention

Example: English to French translation

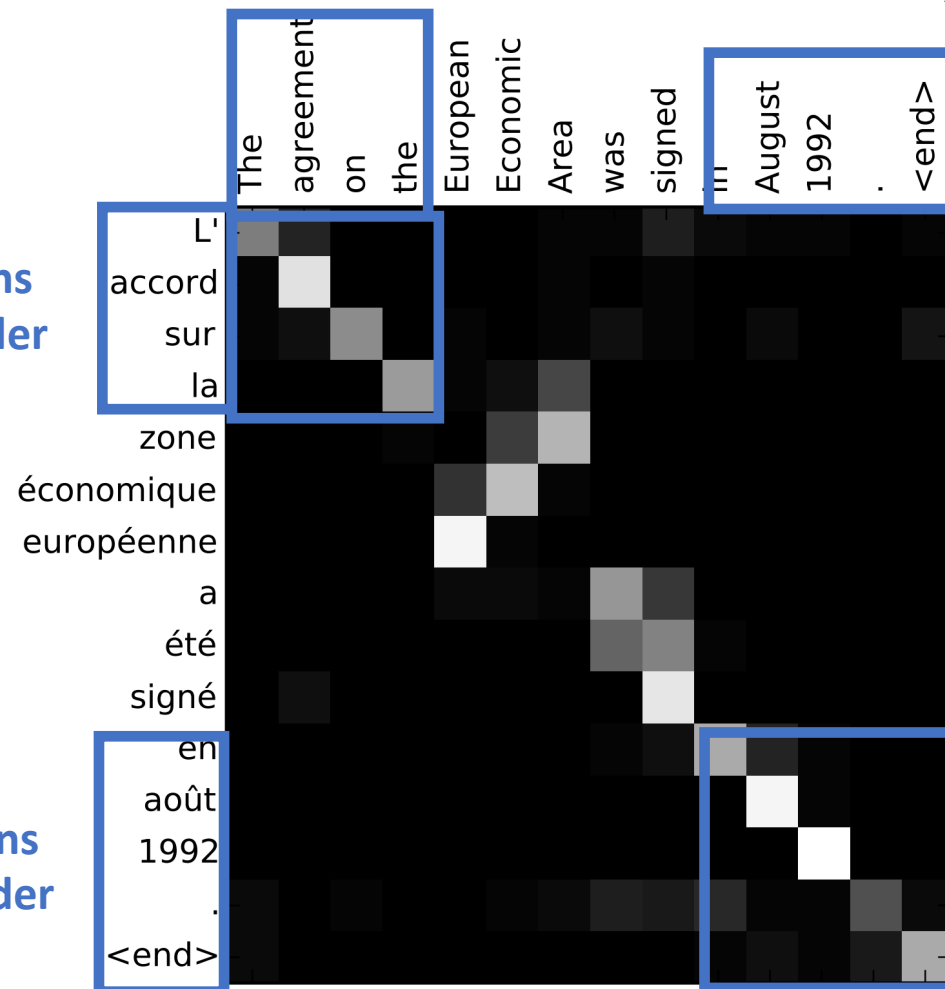
Input: “**The agreement on the** European Economic Area was signed **in August 1992.**”

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.”

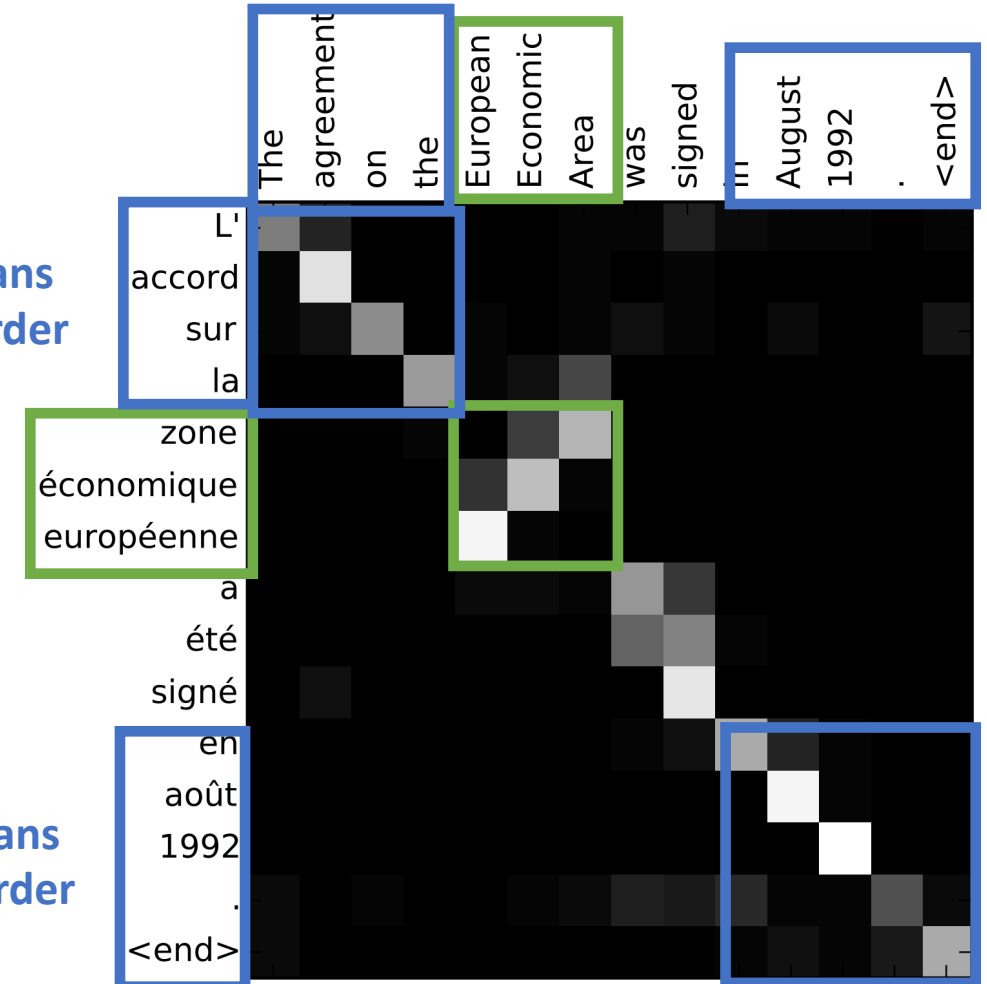
Output: “L'accord sur la zone économique européenne a été signé en août 1992.”

Visualize attention weights $a_{t,i}$

Diagonal attention means words correspond in order

Attention figures out different word orders

Diagonal attention means words correspond in order



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.”

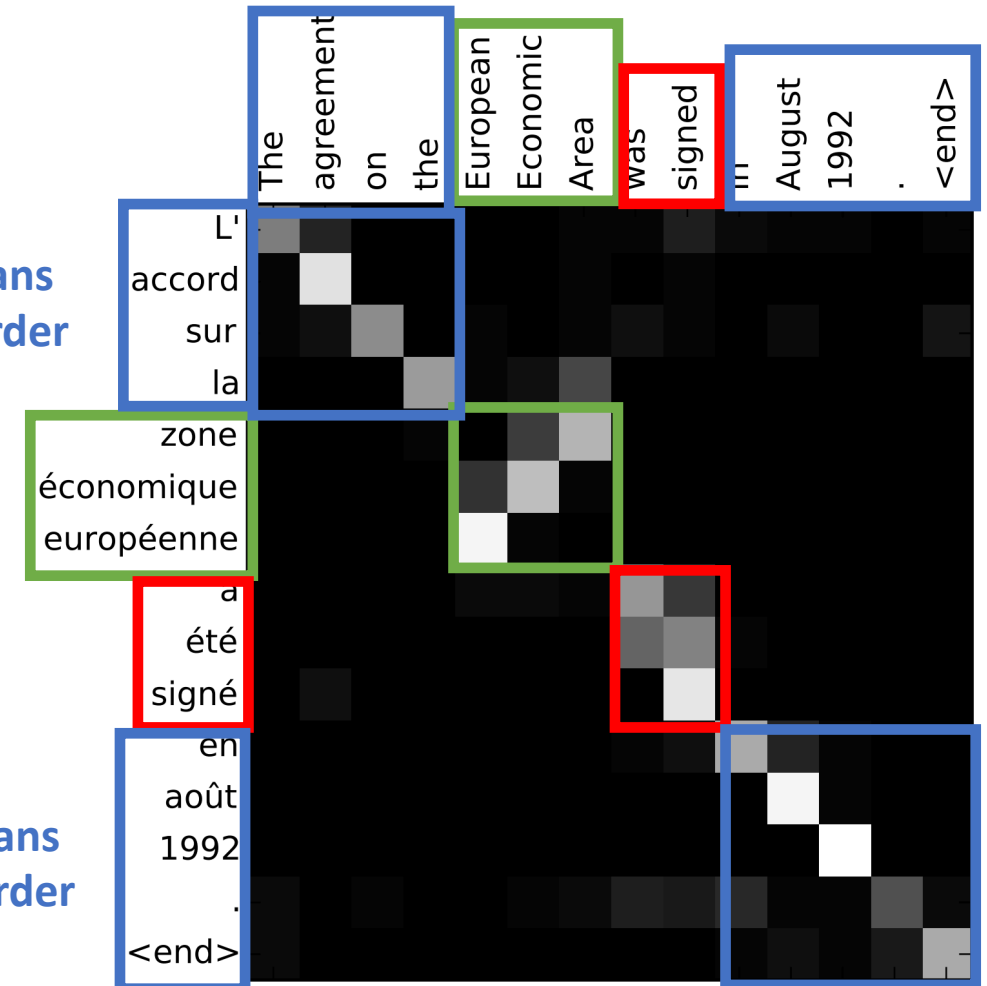
Diagonal attention means words correspond in order

Attention figures out different word orders

Verb conjugation

Diagonal attention means words correspond in order

Visualize attention weights $a_{t,i}$



Sequence-to-Sequence with RNNs and Attention

The decoder doesn't use the fact that h_i form an ordered sequence – it just treats them as an unordered set $\{h_i\}$

Can use similar architecture given any set of input hidden vectors $\{h_i\}$!

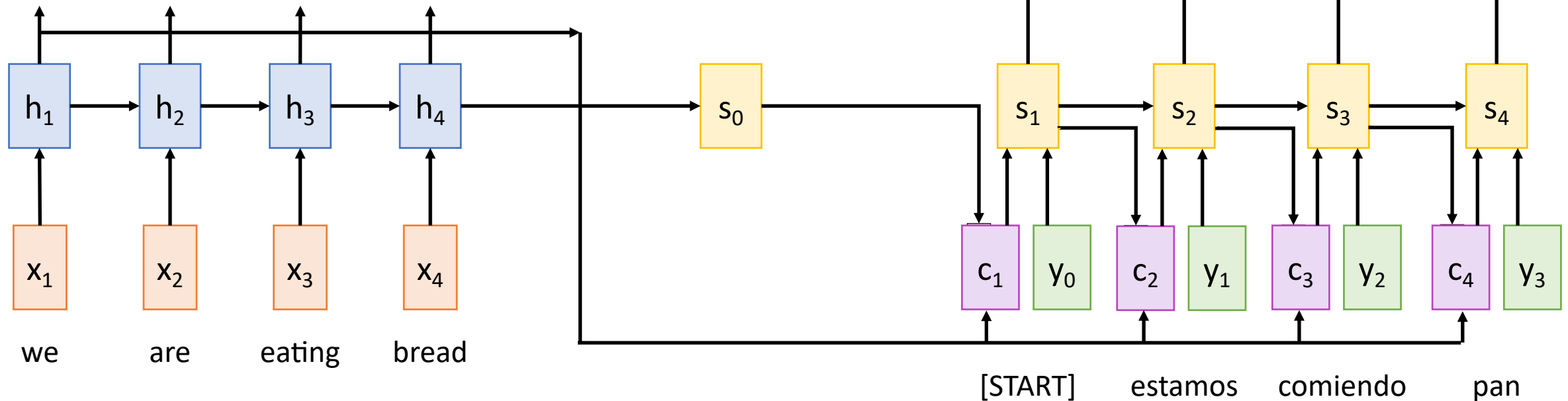
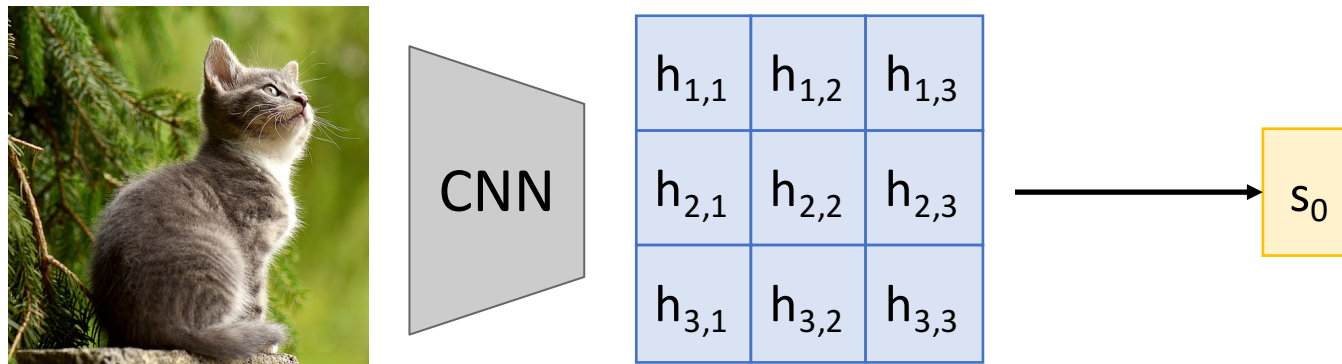


Image Captioning with RNNs and Attention



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

[Cat image](#) is free to use under the [Pixabay License](#)

Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with RNNs and Attention

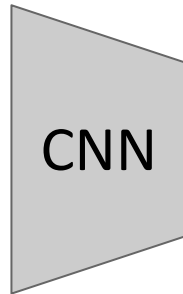
$$e_{t,i,j} = f_{\text{att}}(s_{t-1}, h_{i,j})$$

Alignment scores

$e_{1,1,1}$	$e_{1,1,2}$	$e_{1,1,3}$
$e_{1,2,1}$	$e_{1,2,2}$	$e_{1,2,3}$
$e_{1,3,1}$	$e_{1,3,2}$	$e_{1,3,3}$

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

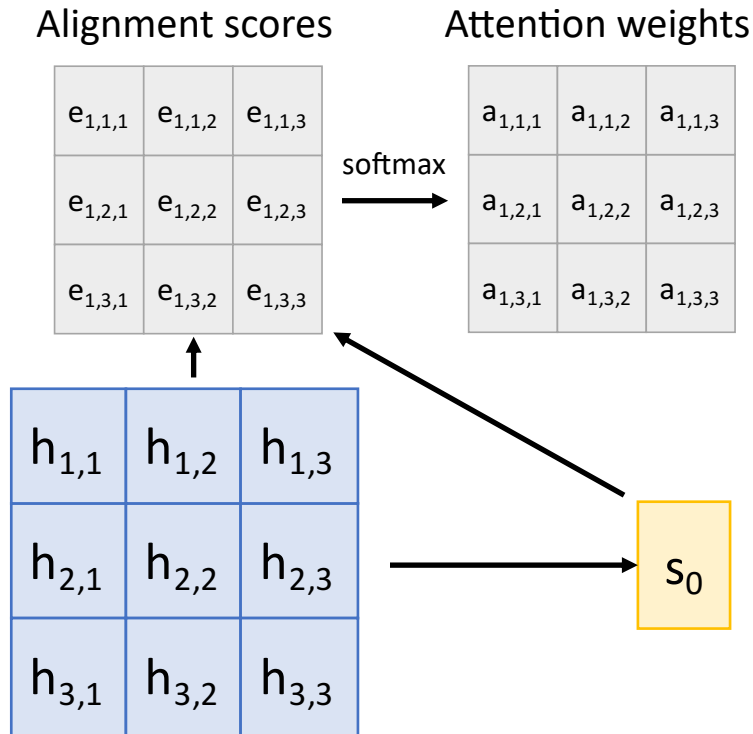
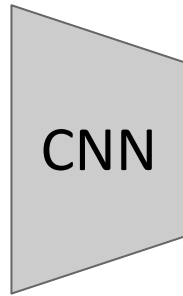
s_0



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

Image Captioning with RNNs and Attention

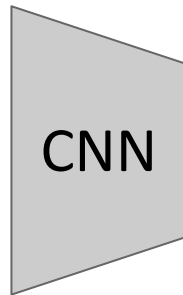
$$e_{t,i,j} = f_{\text{att}}(s_{t-1}, h_{i,j})$$
$$a_{t,:} = \text{softmax}(e_{t,:,:})$$



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

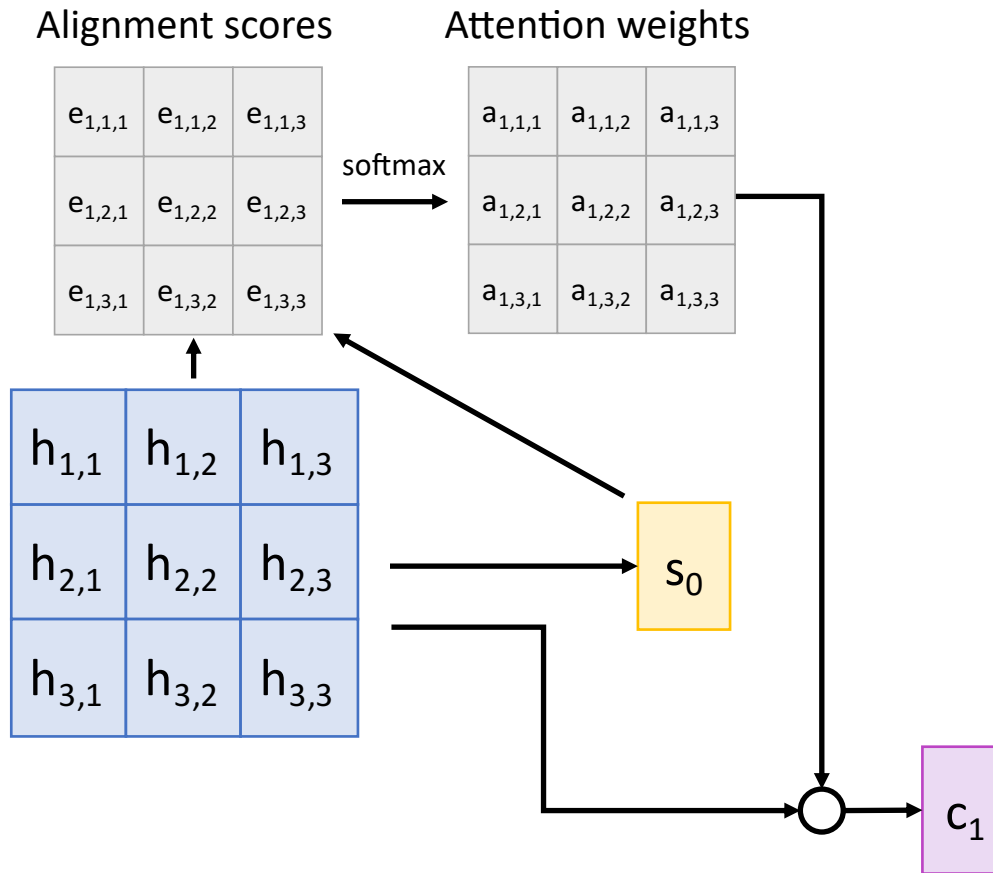
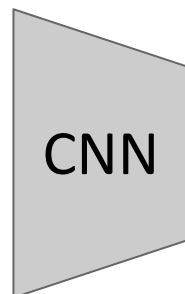
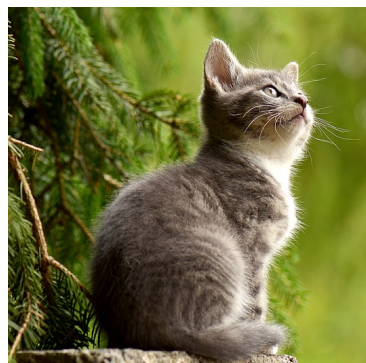


Image Captioning with RNNs and Attention

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

$$a_{t,:} = \text{softmax}(e_{t,:})$$

$$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$$



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

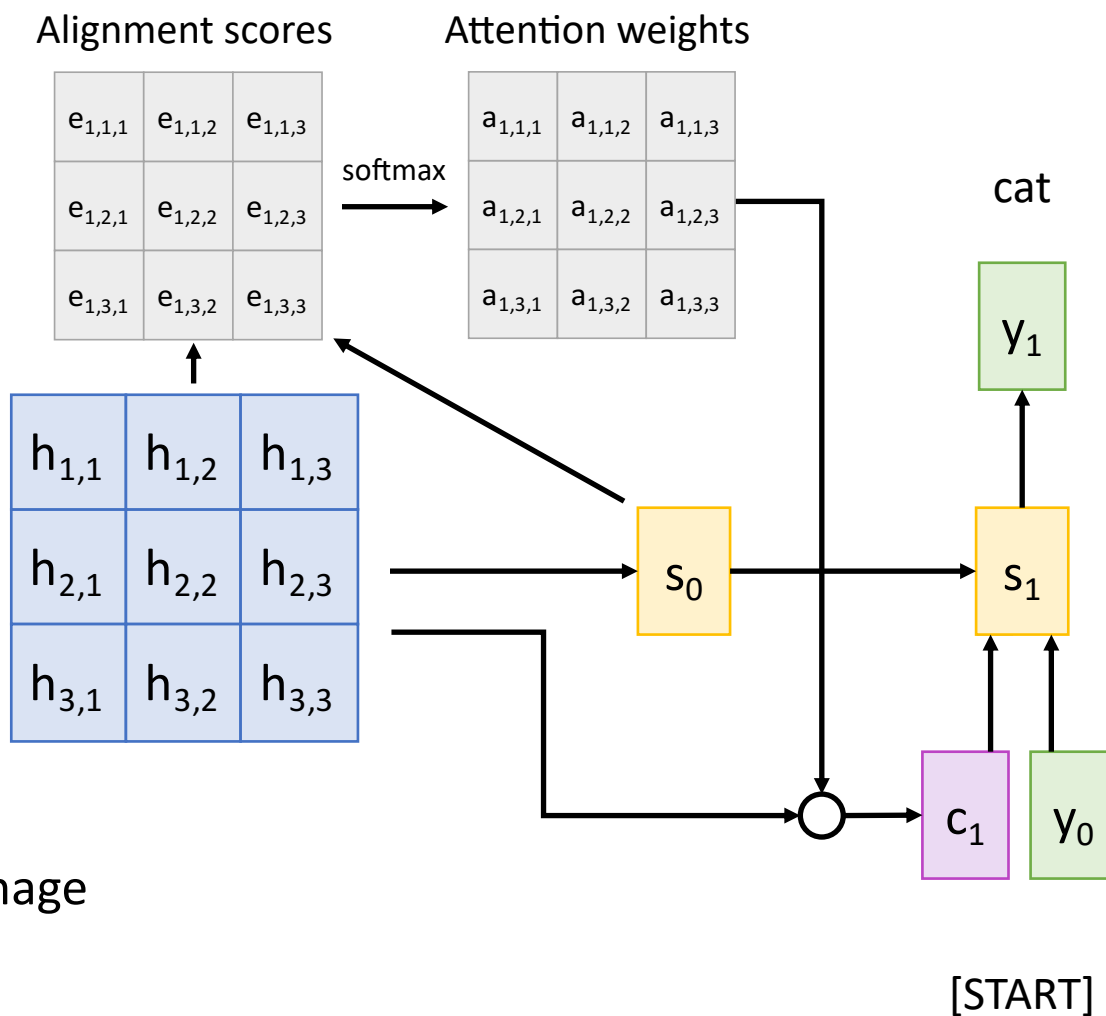


Image Captioning with RNNs and Attention

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



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

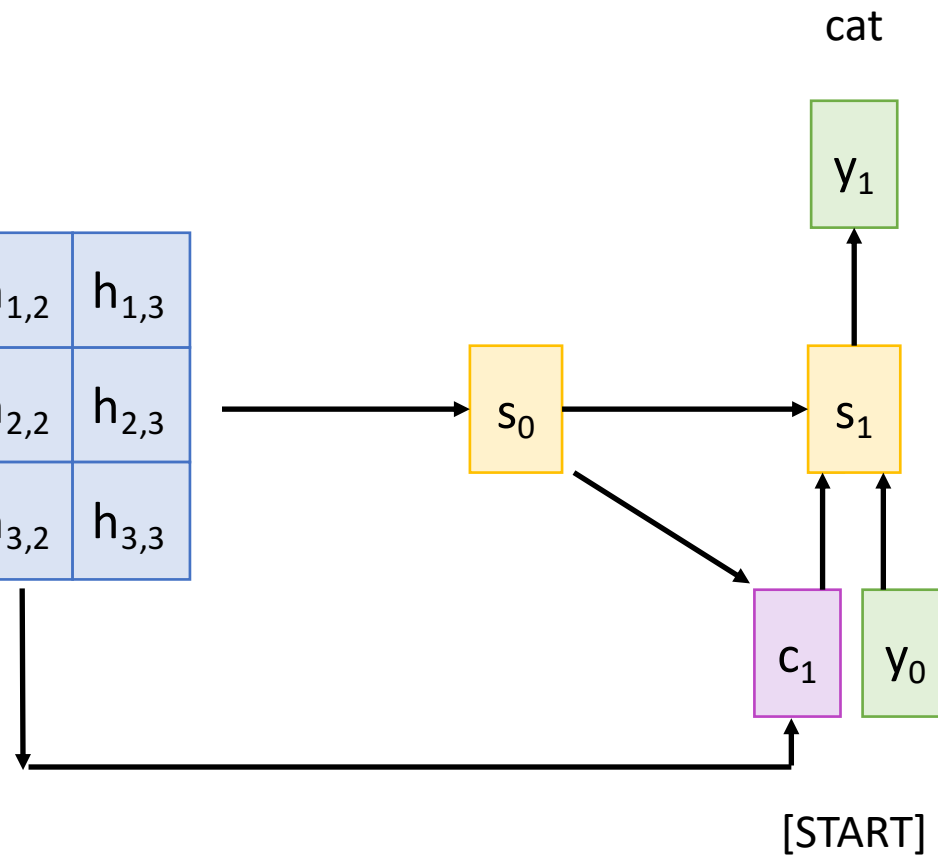


Image Captioning with RNNs and Attention

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



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

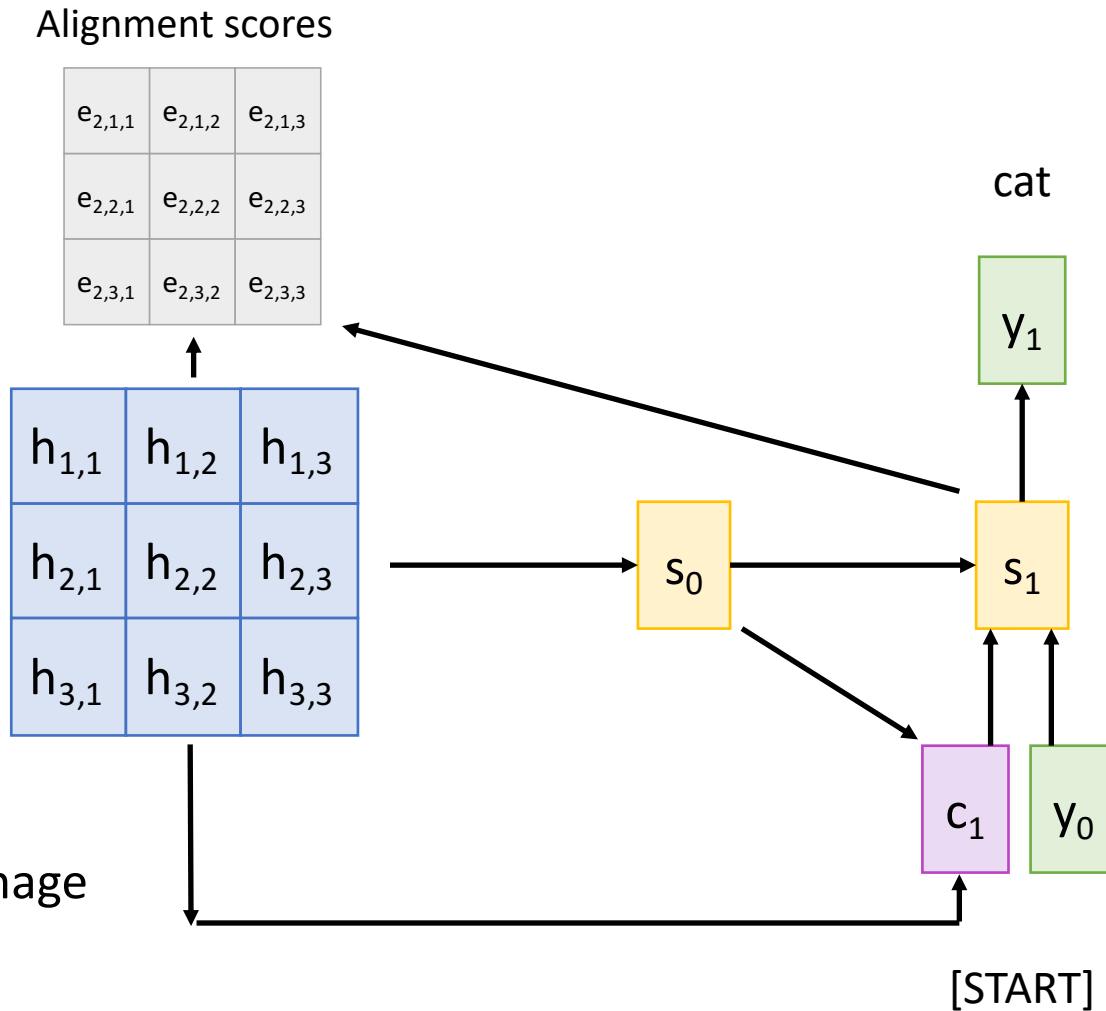
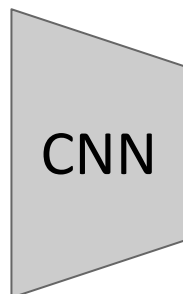


Image Captioning with RNNs and Attention

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

$$a_{t,:,:} = \text{softmax}(e_{t,:,:})$$

$$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$$



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

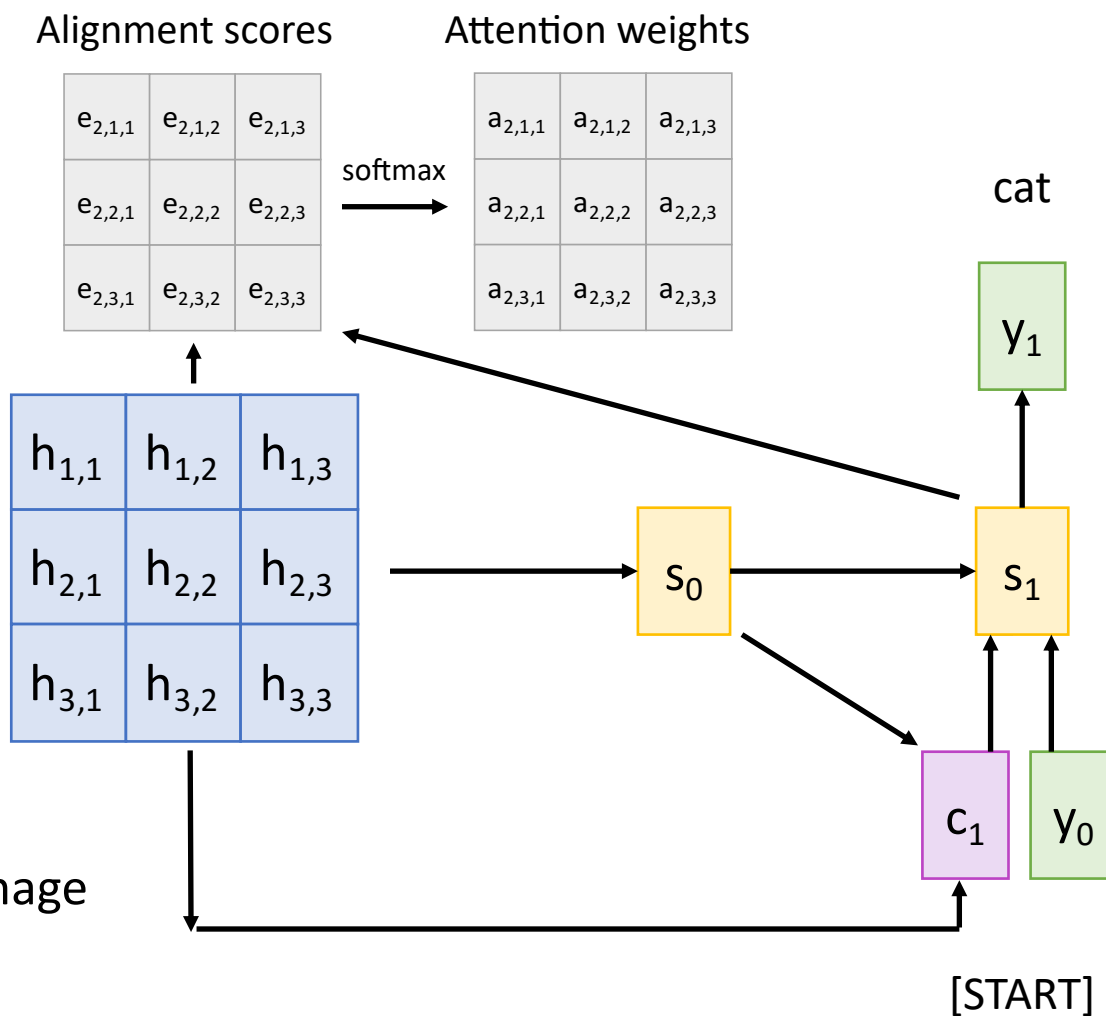
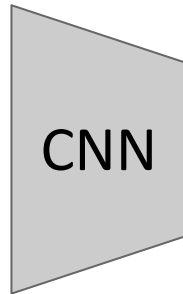


Image Captioning with RNNs and Attention

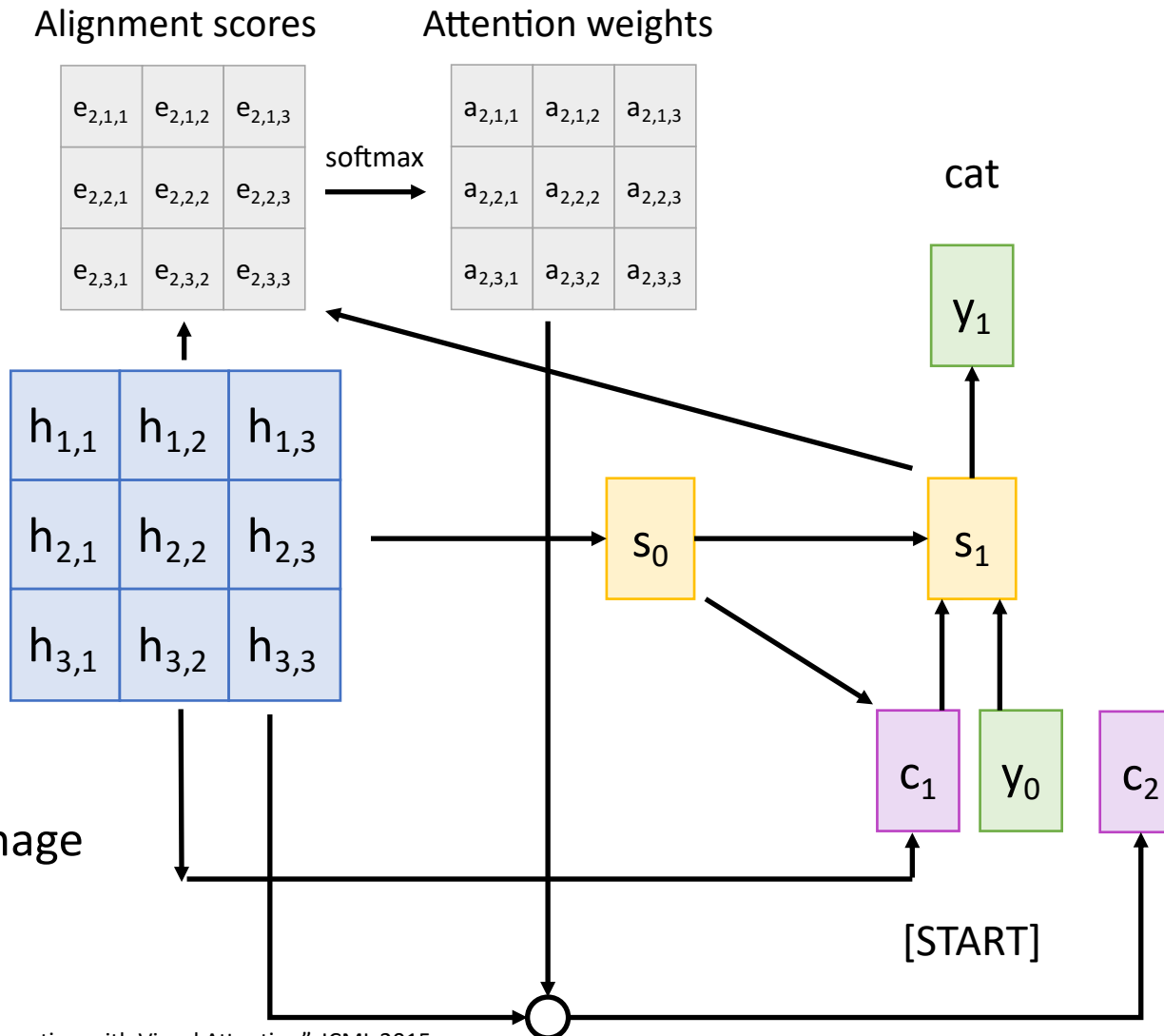
$$e_{t,i,j} = f_{\text{att}}(s_{t-1}, h_{i,j})$$

$$a_{t,:,:} = \text{softmax}(e_{t,:,:})$$

$$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$$



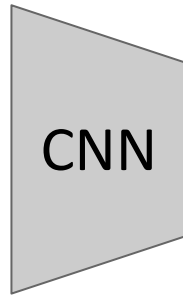
Use a CNN to compute a grid of features for an image



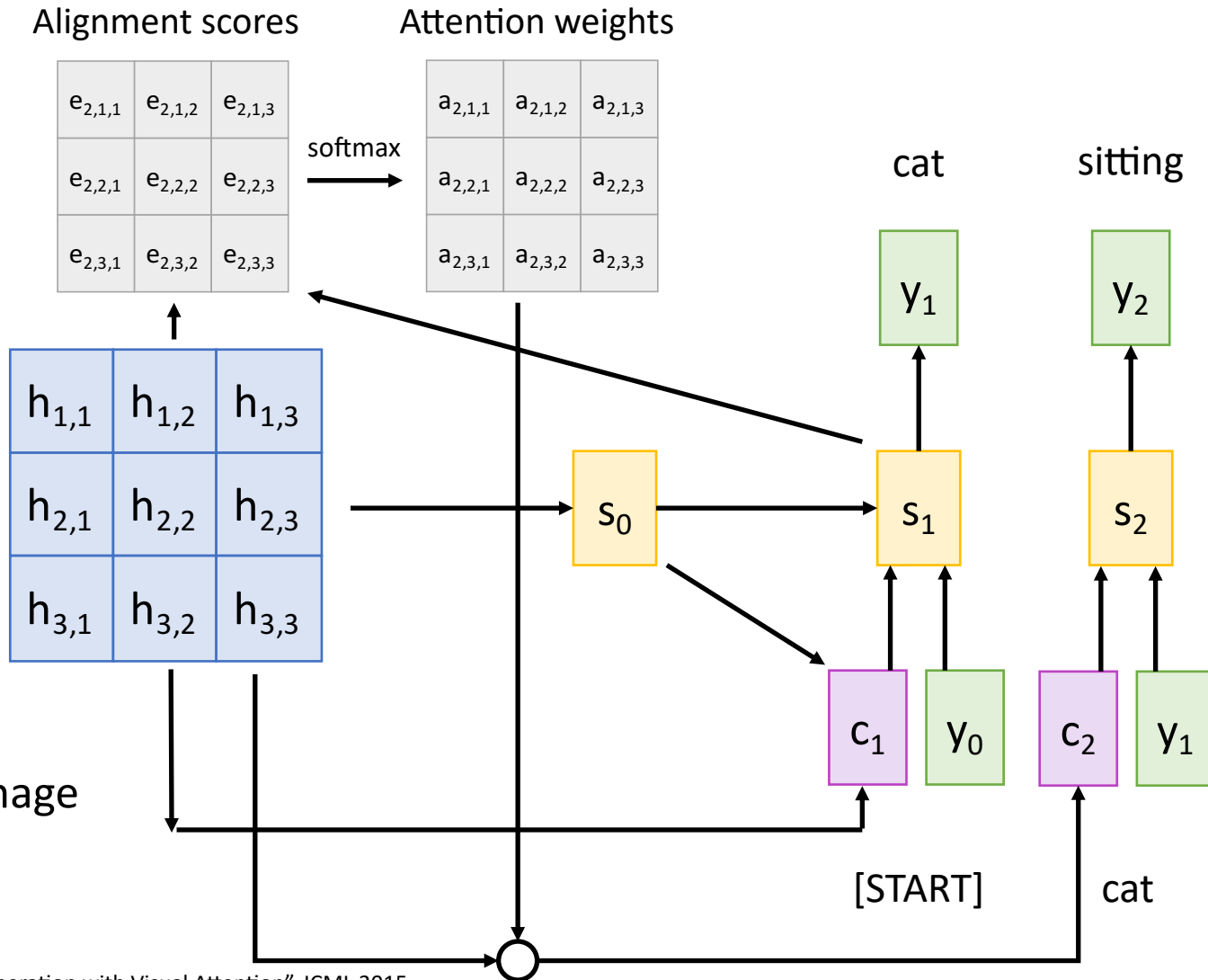
Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

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

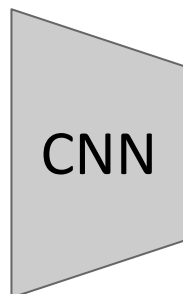
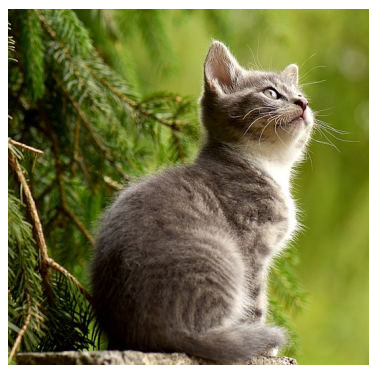


Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

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

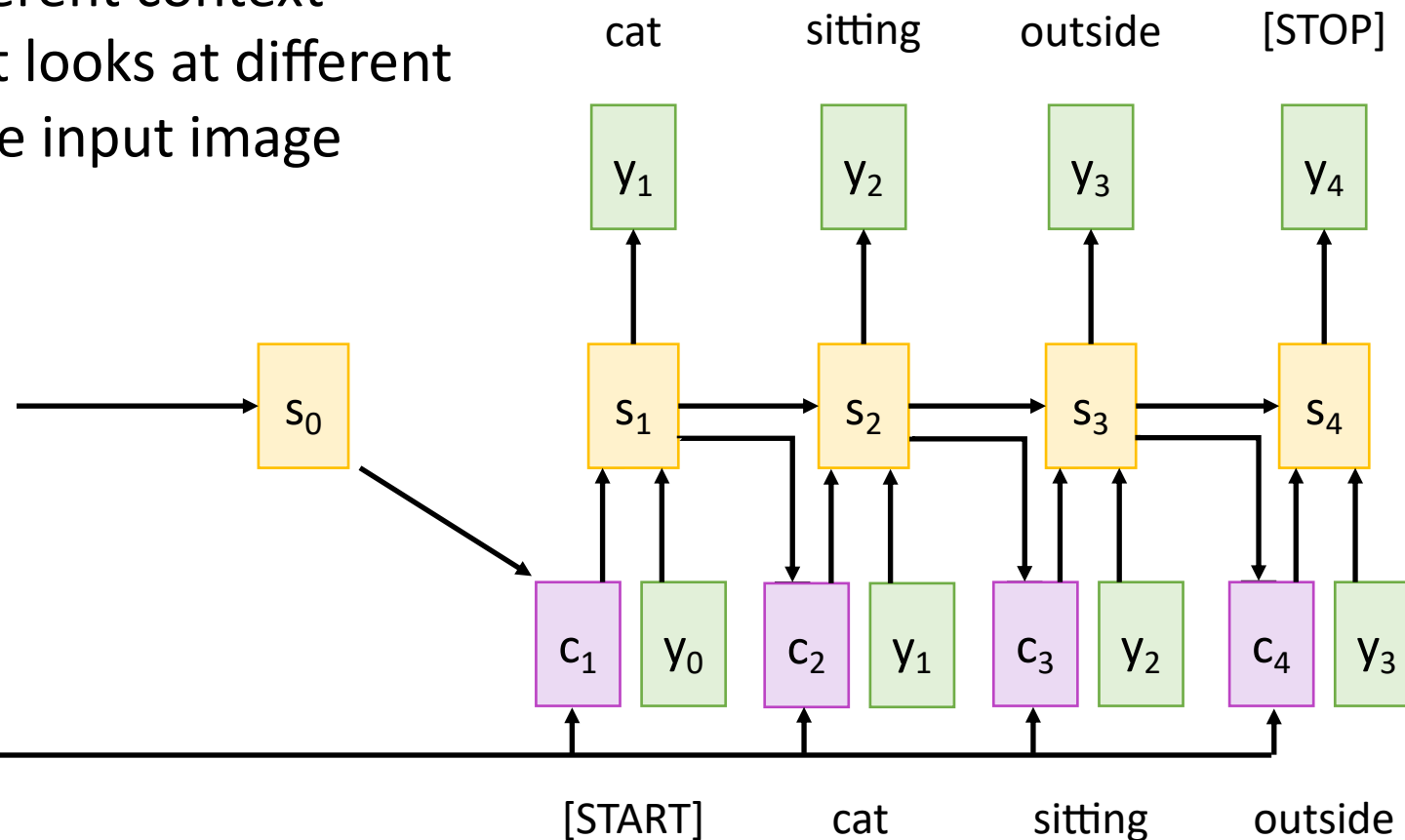
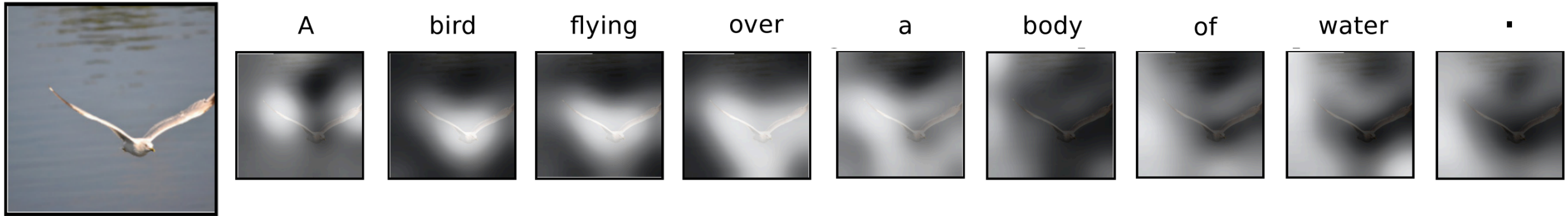
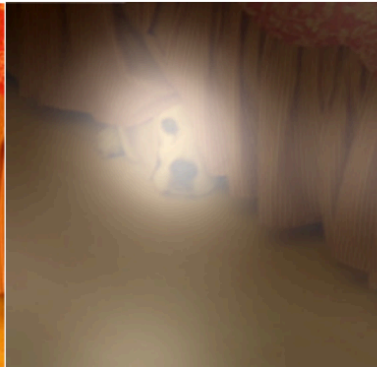


Image Captioning with RNNs and Attention



Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with RNNs and Attention



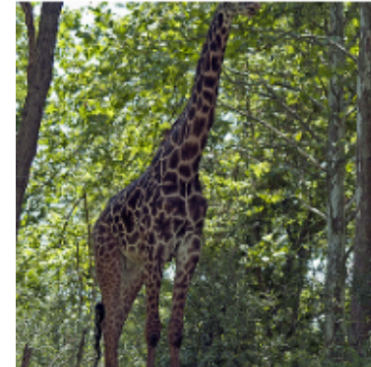
A dog is standing on a hardwood floor.



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

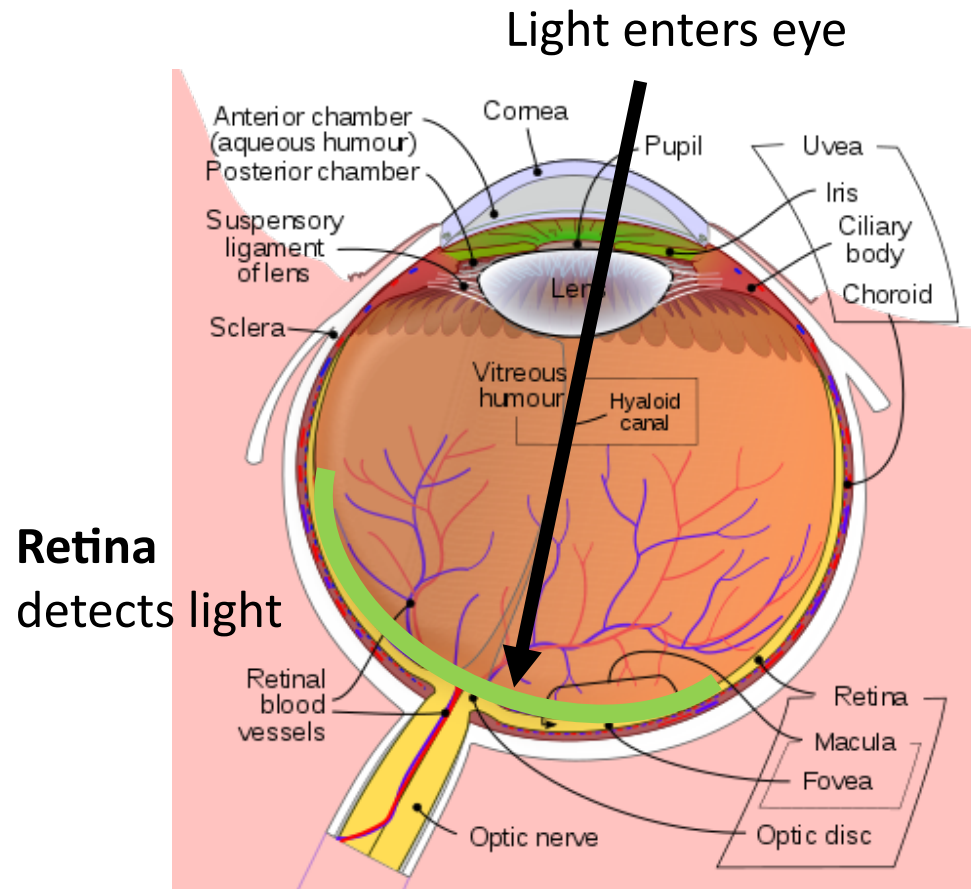


A group of people sitting on a boat in the water.

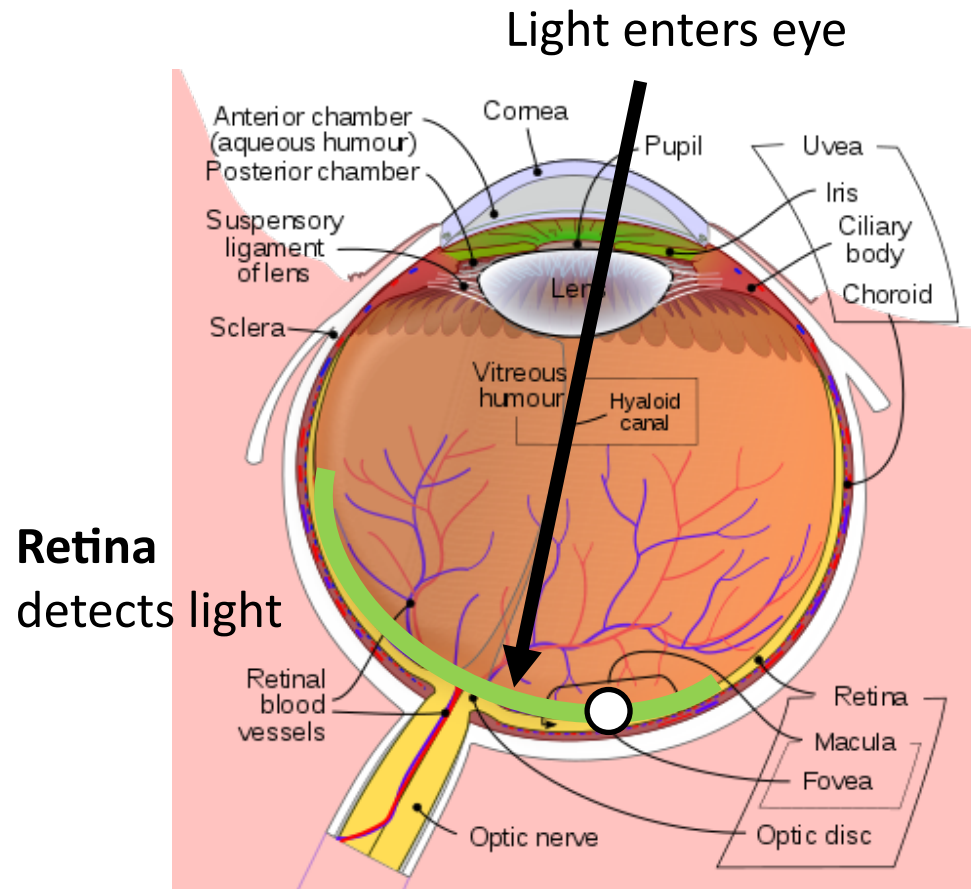


A giraffe standing in a forest with trees in the background.

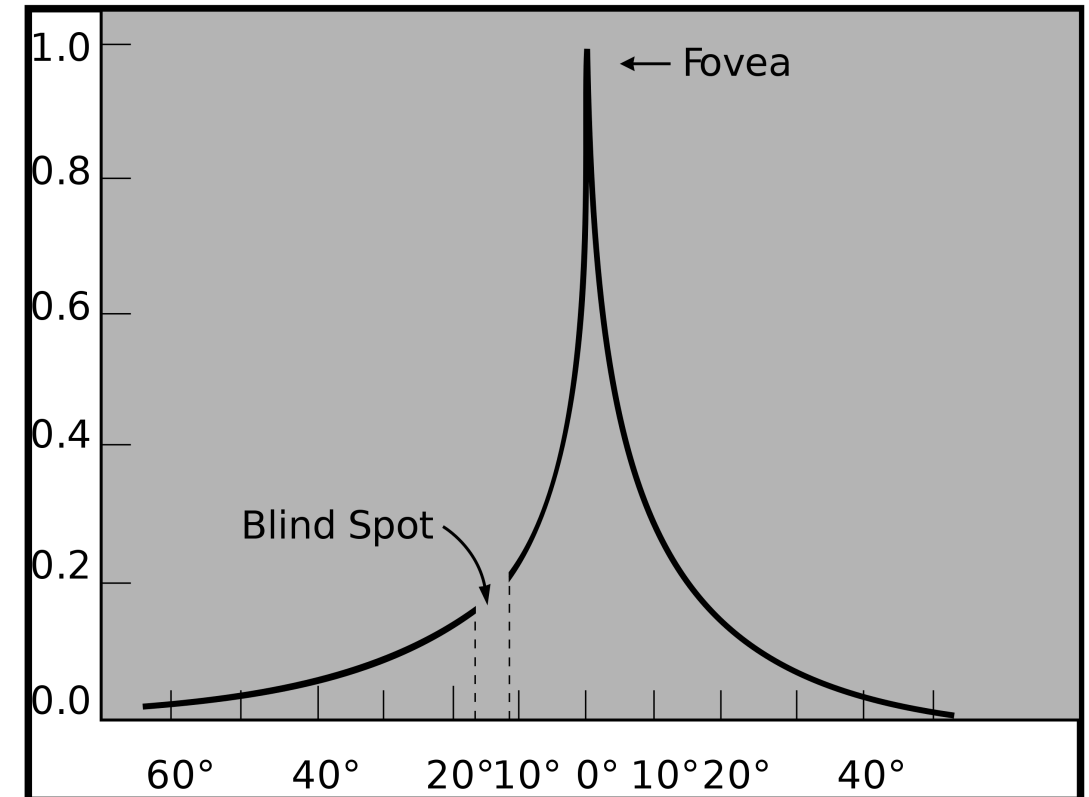
Human Vision: Fovea



Human Vision: Fovea



The **fovea** is a tiny region of the retina that can see with high acuity



Human Vision: Saccades

Human eyes are constantly moving so we don't notice



The **fovea** is a tiny region of the retina that can see with high acuity

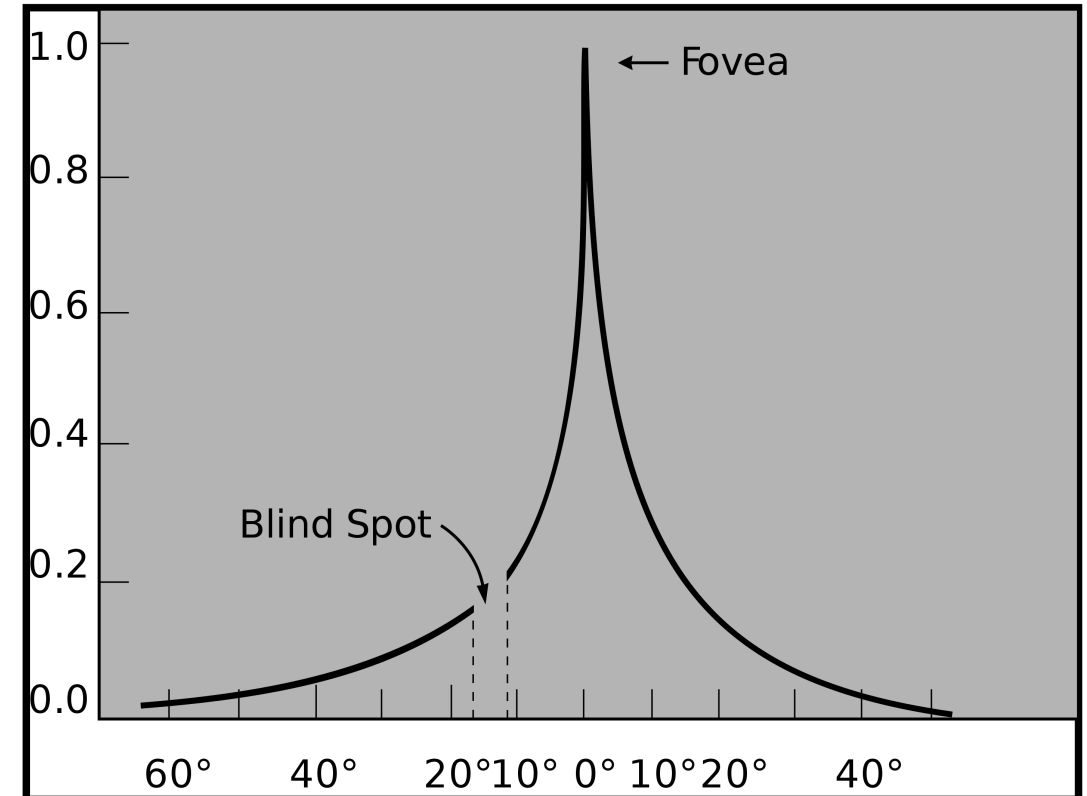
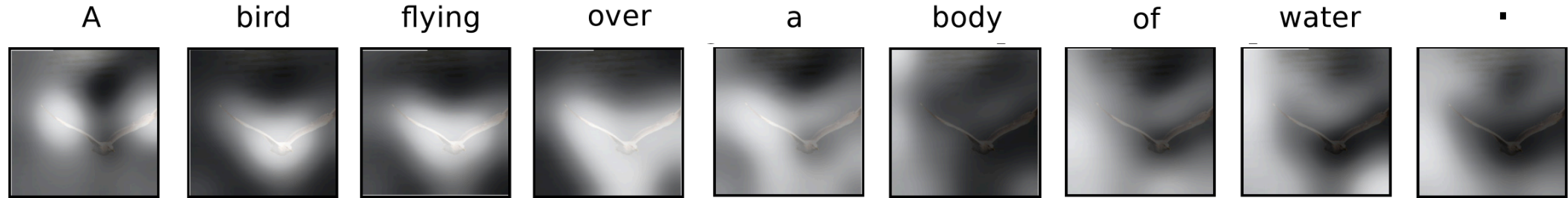
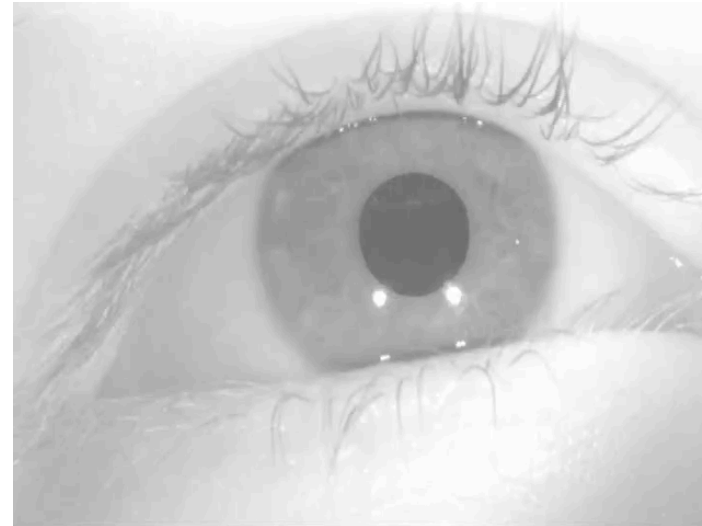


Image Captioning with RNNs and Attention



Attention weights at each timestep kind of like saccades of human eye



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: $N_X \times D_X$)

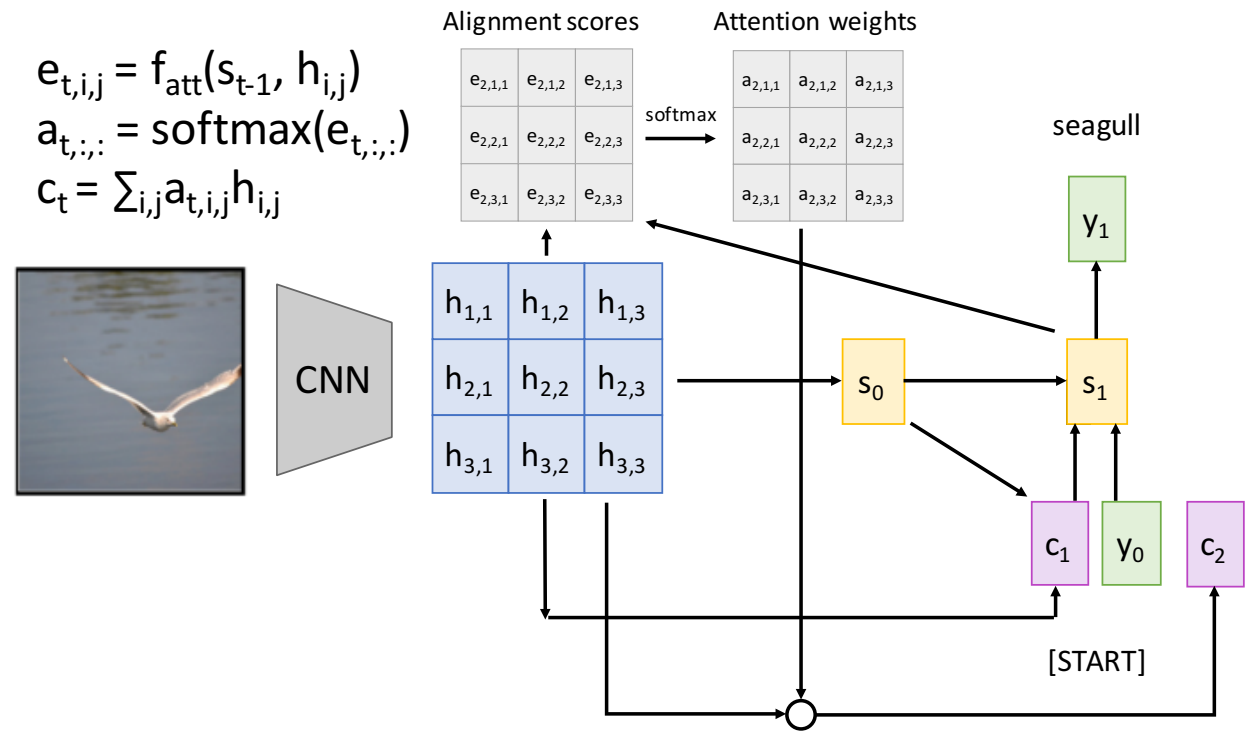
Similarity function: f_{att}

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: $N_X \times D_Q$)

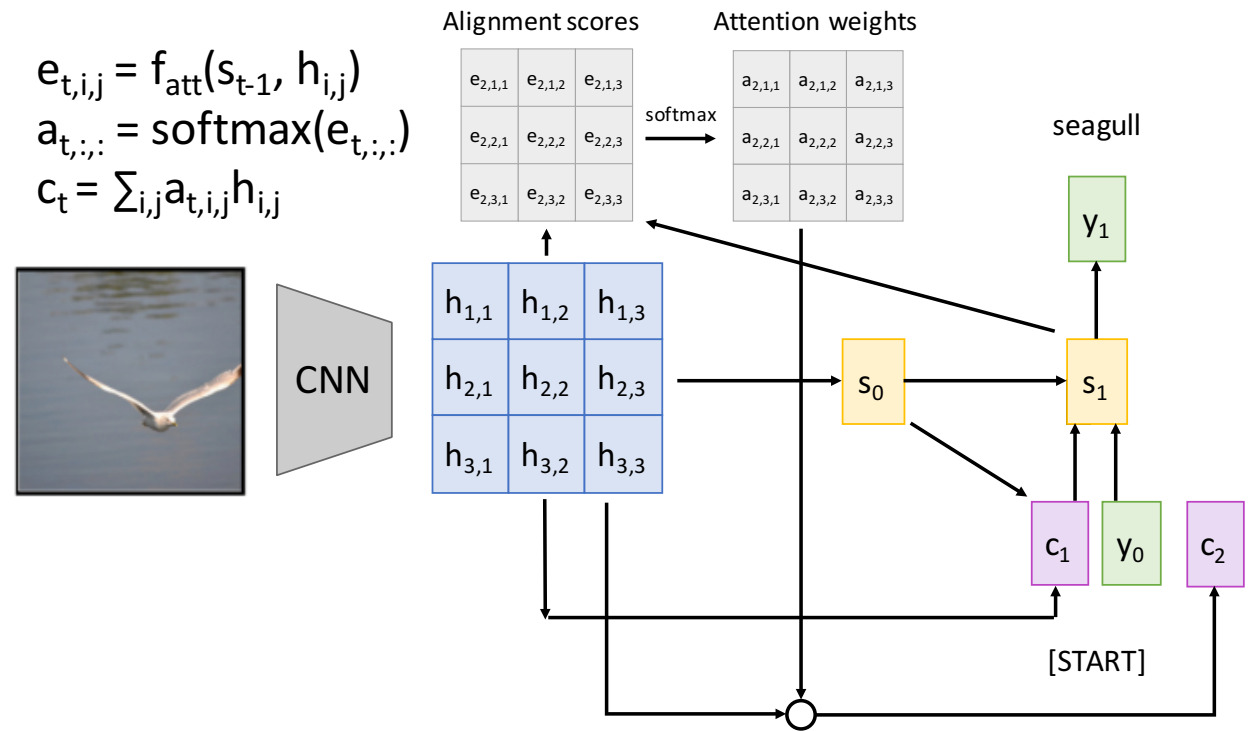
Similarity function: dot product

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: $N_X \times D_Q$)

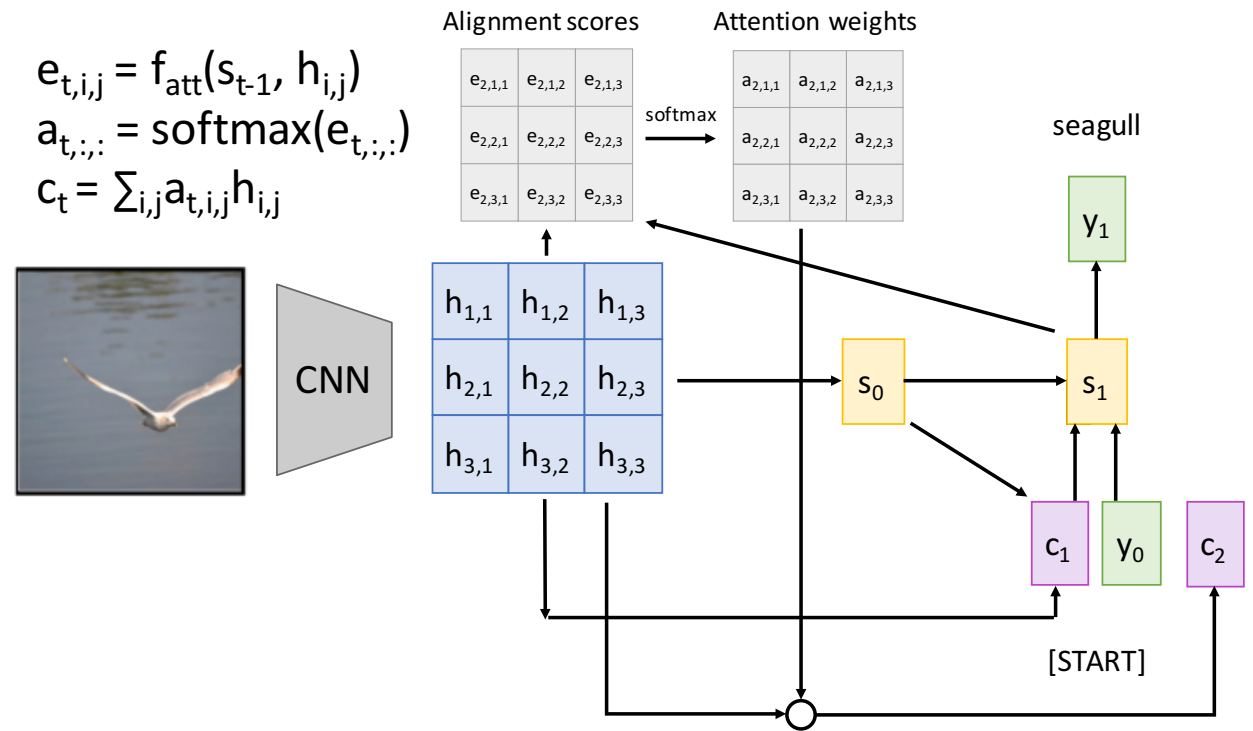
Similarity function: scaled dot product

Computation:

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

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 **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

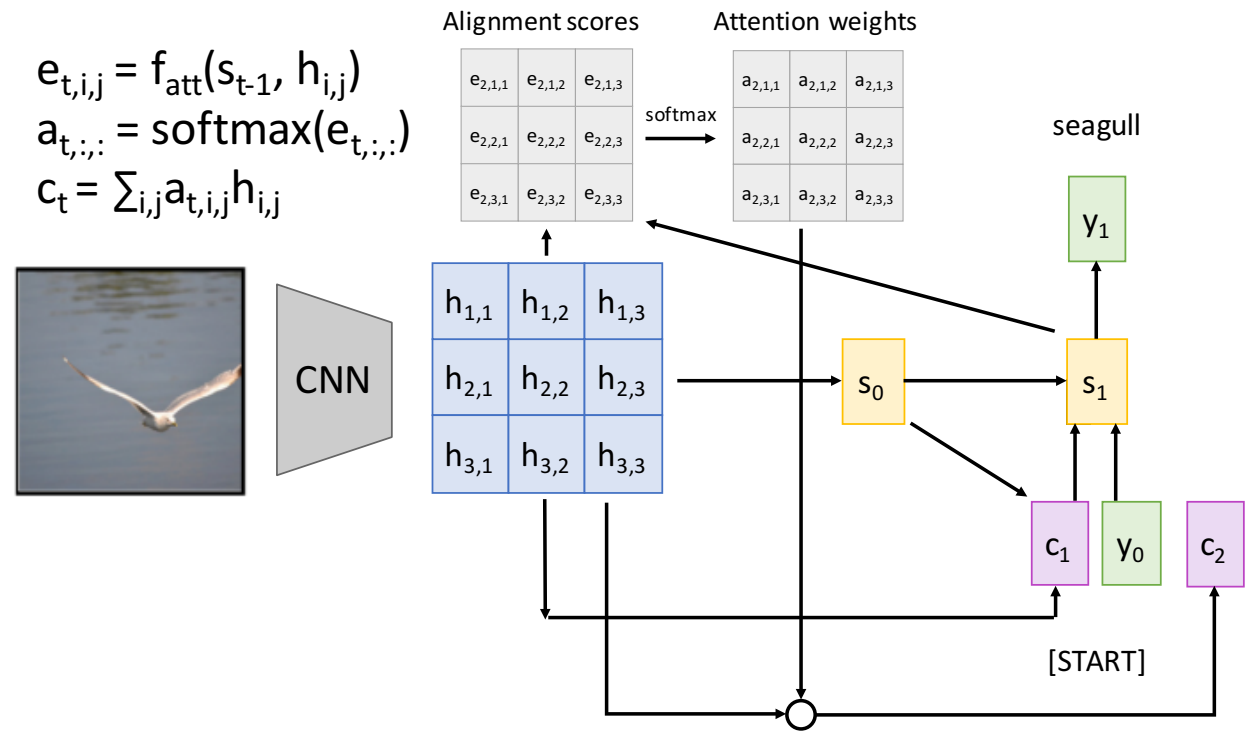
Then $|a| = (\sum_i a^2)^{1/2} = a \sqrt{D}$

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

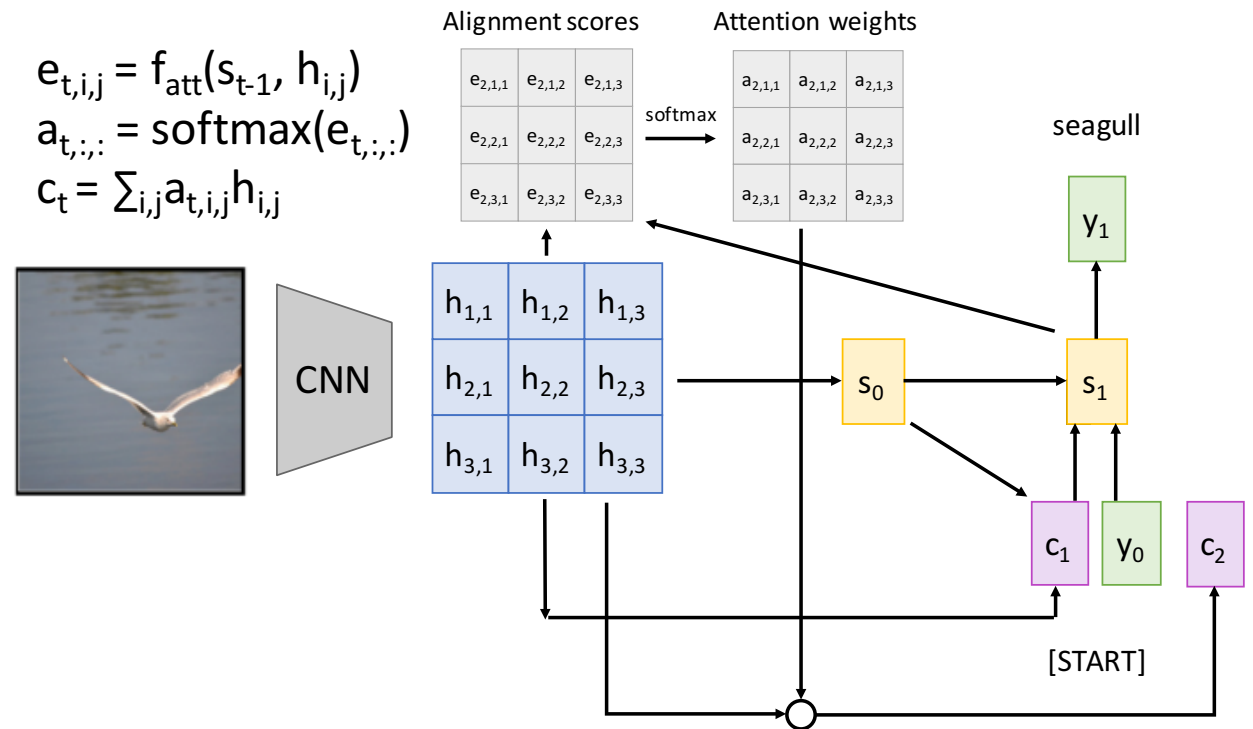
Input vectors: **X** (Shape: $N_X \times D_X$)

Computation:

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

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:

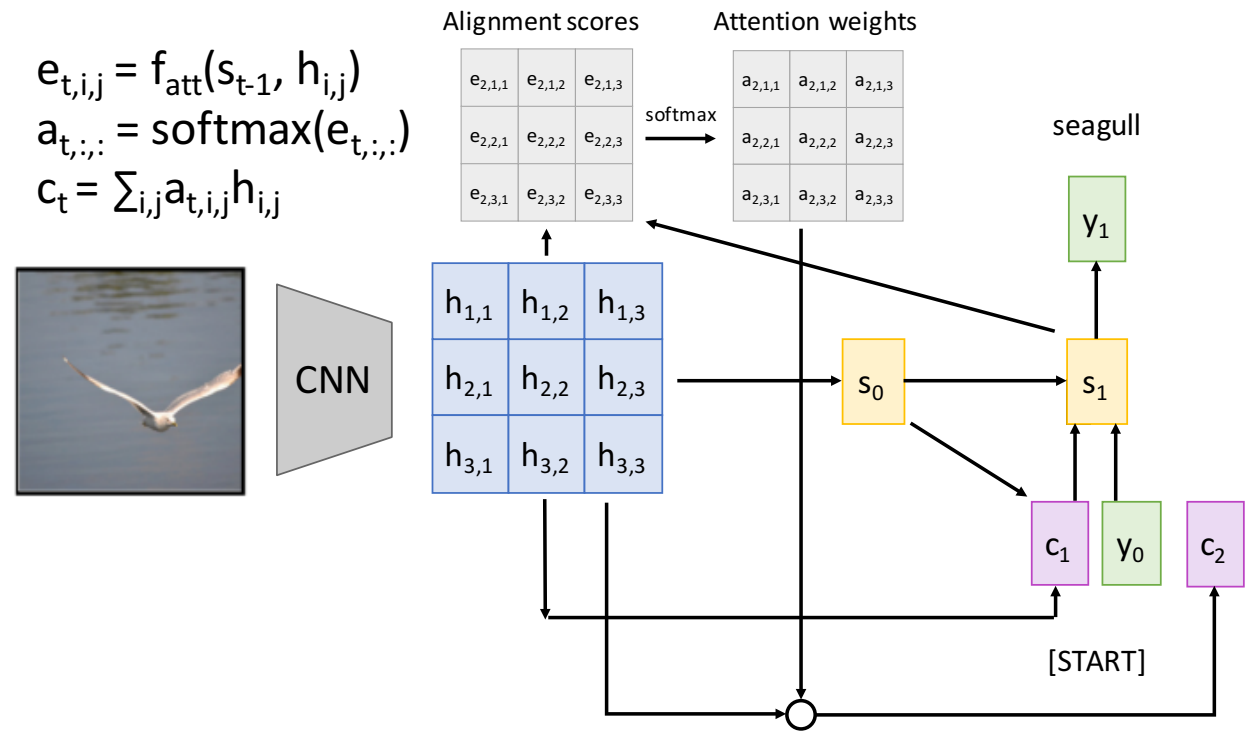
Key vectors: $\mathbf{K} = \mathbf{XW}_K$ (Shape: $N_X \times D_Q$)

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

Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (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: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (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: $N_Q \times D_V$) $Y_i = \sum_j A_{i,j} \mathbf{V}_j$

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:

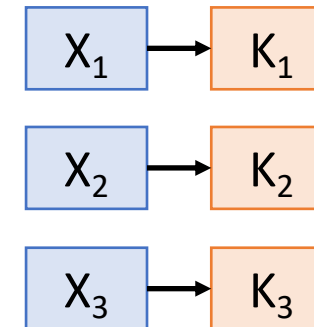
Key vectors: $\mathbf{K} = \mathbf{XW}_K$ (Shape: $N_X \times D_Q$)

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

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

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

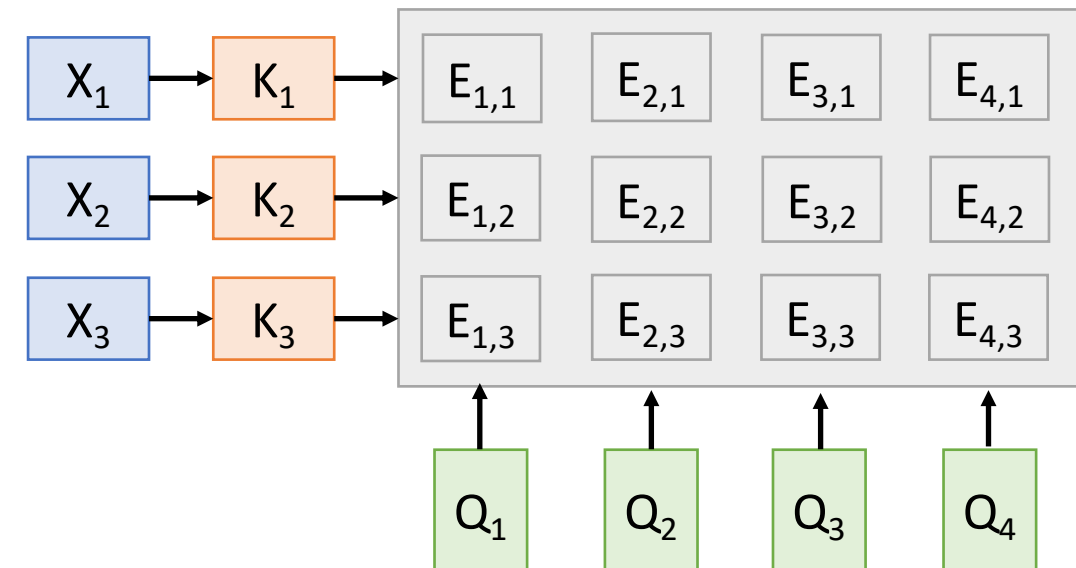
Key vectors: $\mathbf{K} = \mathbf{XW}_K$ (Shape: $N_X \times D_Q$)

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

Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (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: $N_Q \times D_V$) $Y_i = \sum_j A_{i,j} \mathbf{V}_j$



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:

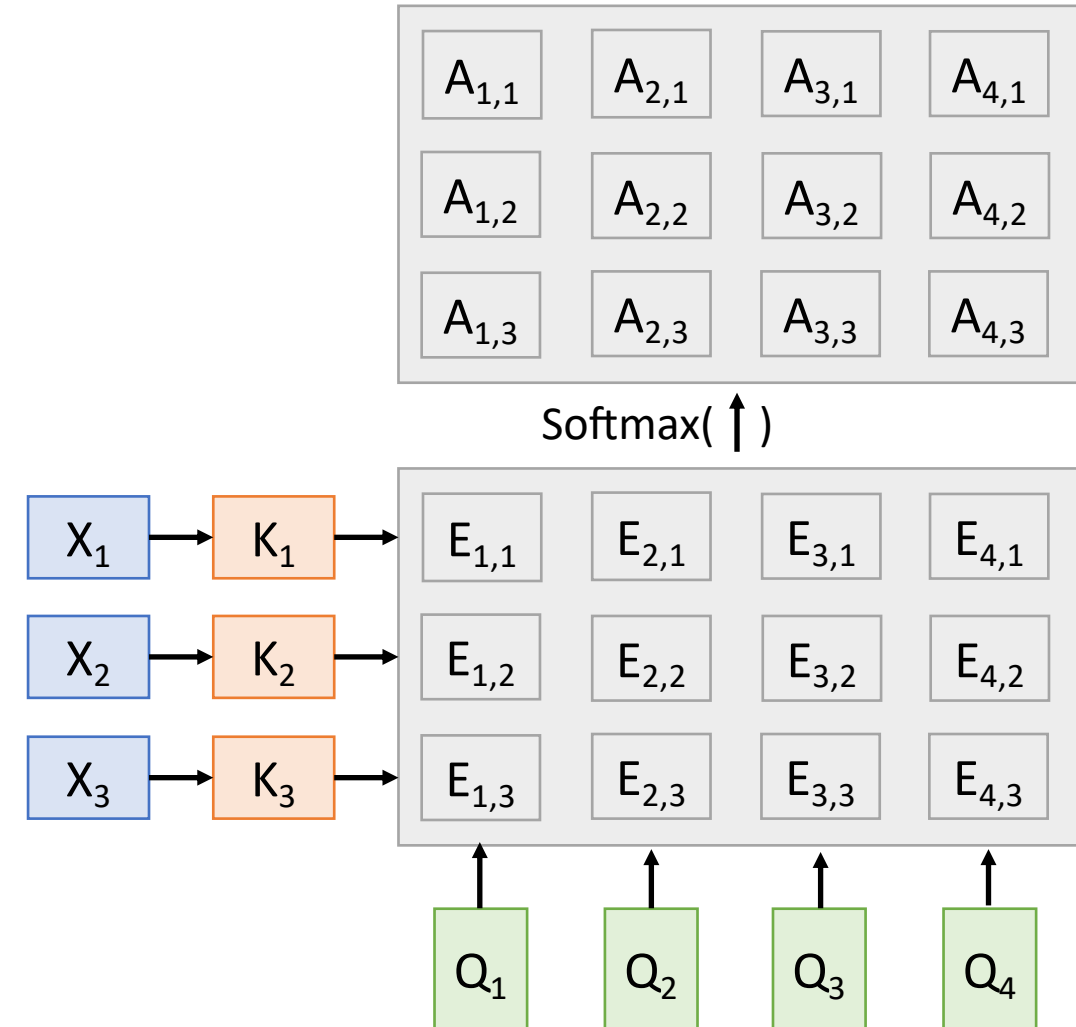
Key vectors: $\mathbf{K} = \mathbf{XW}_K$ (Shape: $N_X \times D_Q$)

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

Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (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: $N_Q \times D_V$) $Y_i = \sum_j A_{i,j} \mathbf{V}_j$



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:

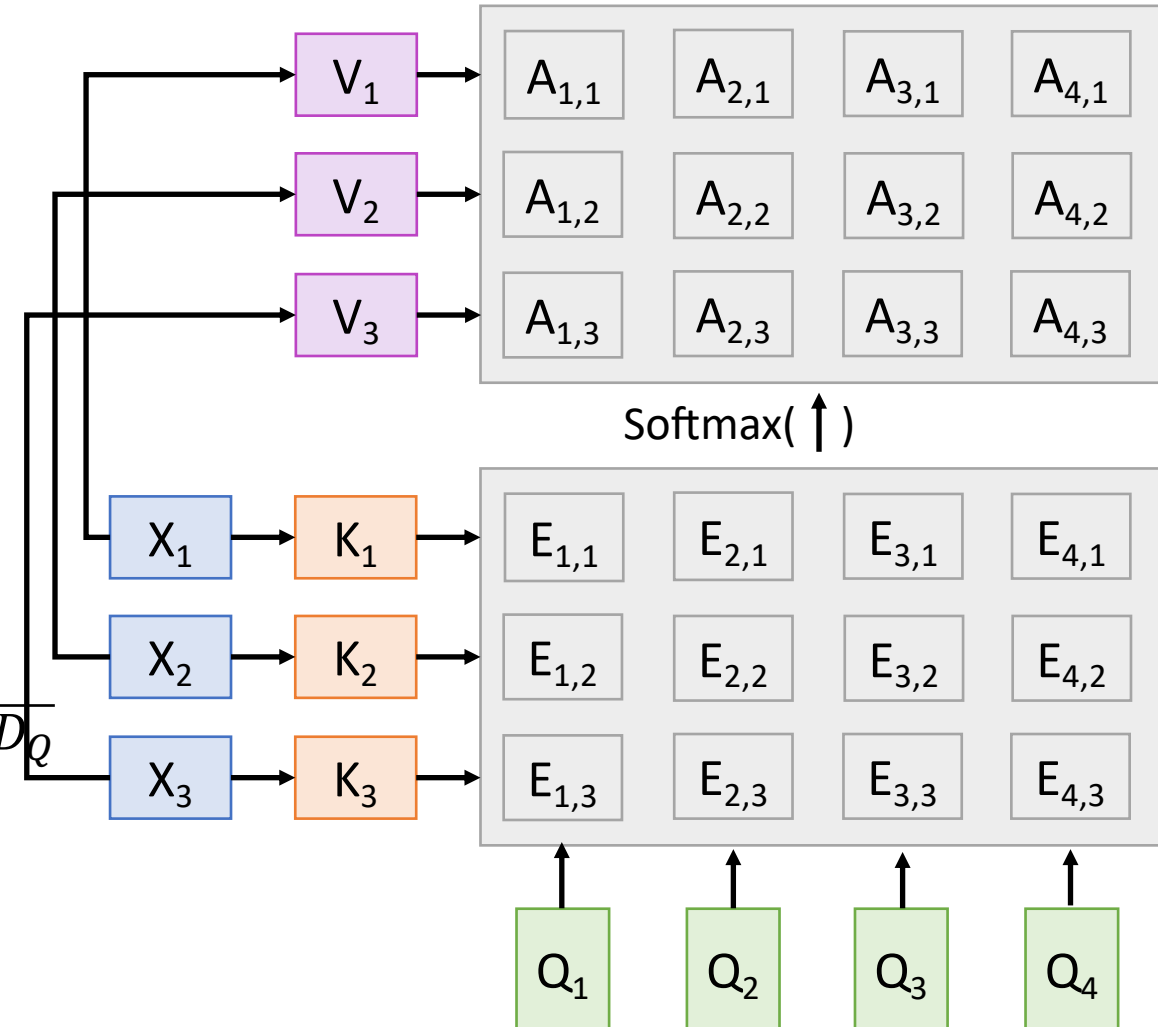
Key vectors: $\mathbf{K} = \mathbf{XW}_K$ (Shape: $N_X \times D_Q$)

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

Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (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: $N_Q \times D_V$) $Y_i = \sum_j A_{i,j} \mathbf{V}_j$



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:

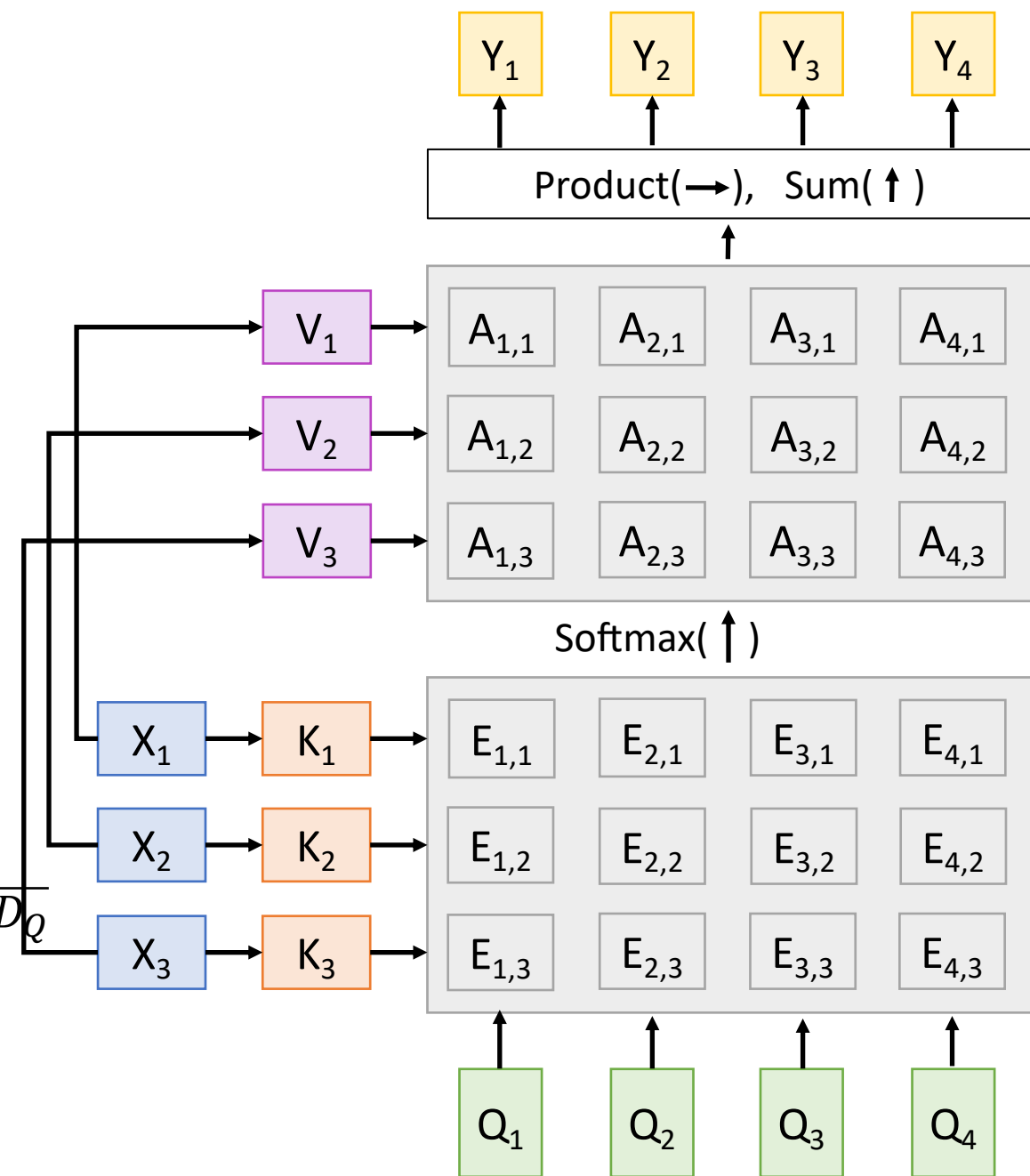
Key vectors: $\mathbf{K} = \mathbf{XW}_K$ (Shape: $N_X \times D_Q$)

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

Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (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: $N_Q \times D_V$) $Y_i = \sum_j A_{i,j} \mathbf{V}_j$



Self-Attention Layer

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

Inputs:

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

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

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

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

Computation:

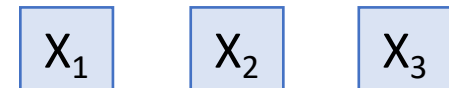
Key vectors: **K** = **XW_K** (Shape: $N_X \times D_Q$)

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

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

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

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



Self-Attention Layer

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

Inputs:

Input vectors: \mathbf{X} (Shape: $N_x \times D_x$)

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

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

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

Computation:

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

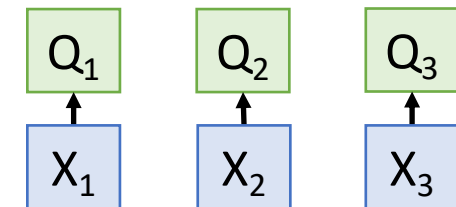
Key vectors: $\mathbf{K} = \mathbf{XW}_K$ (Shape: $N_x \times D_Q$)

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

Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (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

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

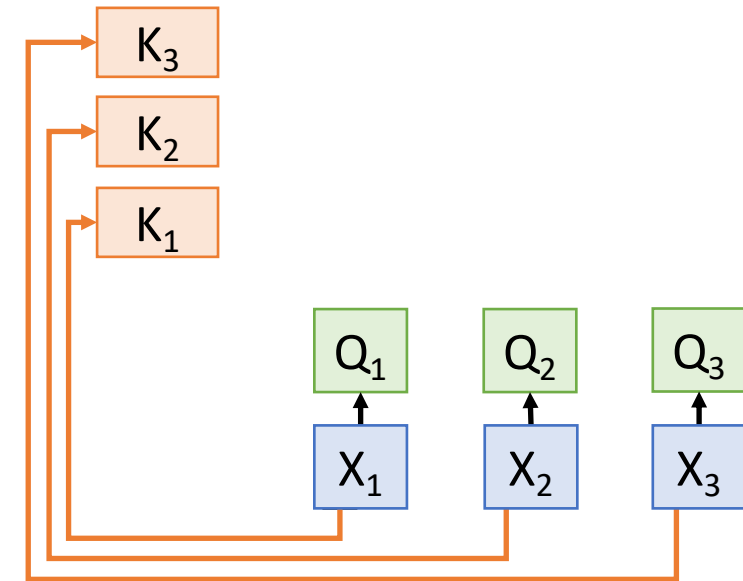
Key vectors: $\mathbf{K} = \mathbf{XW}_K$ (Shape: $N_x \times D_Q$)

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

Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (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

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

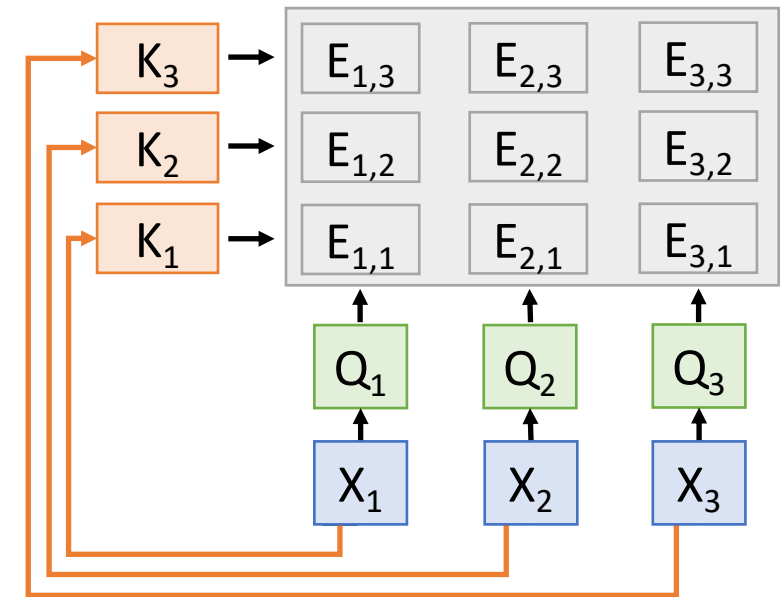
Key vectors: $\mathbf{K} = \mathbf{XW}_K$ (Shape: $N_x \times D_Q$)

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

Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (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

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

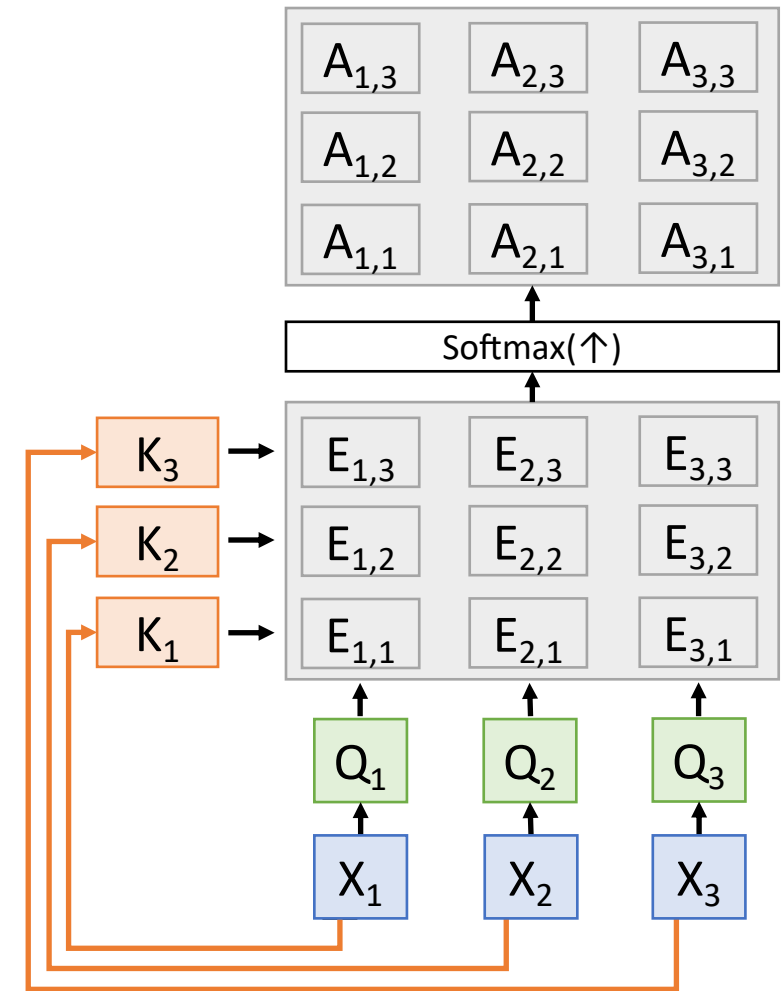
Key vectors: $\mathbf{K} = \mathbf{XW}_K$ (Shape: $N_x \times D_Q$)

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

Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (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

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

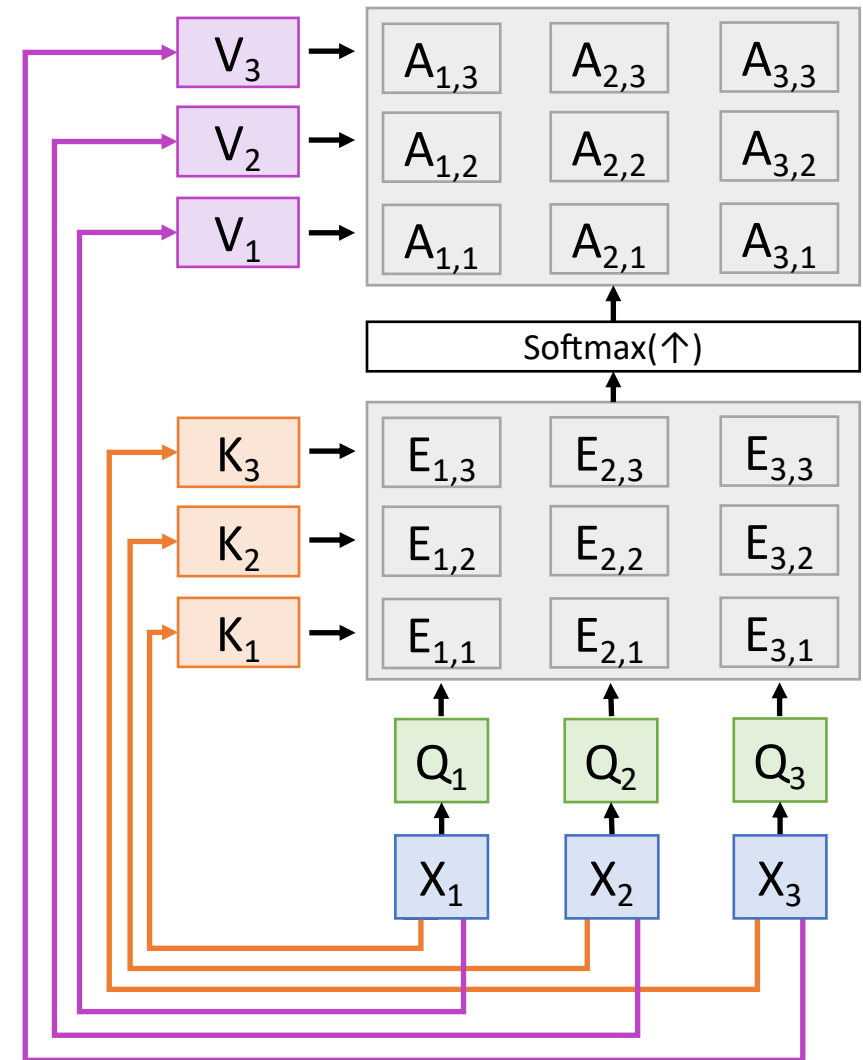
Key vectors: $\mathbf{K} = \mathbf{XW}_K$ (Shape: $N_x \times D_Q$)

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

Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (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

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

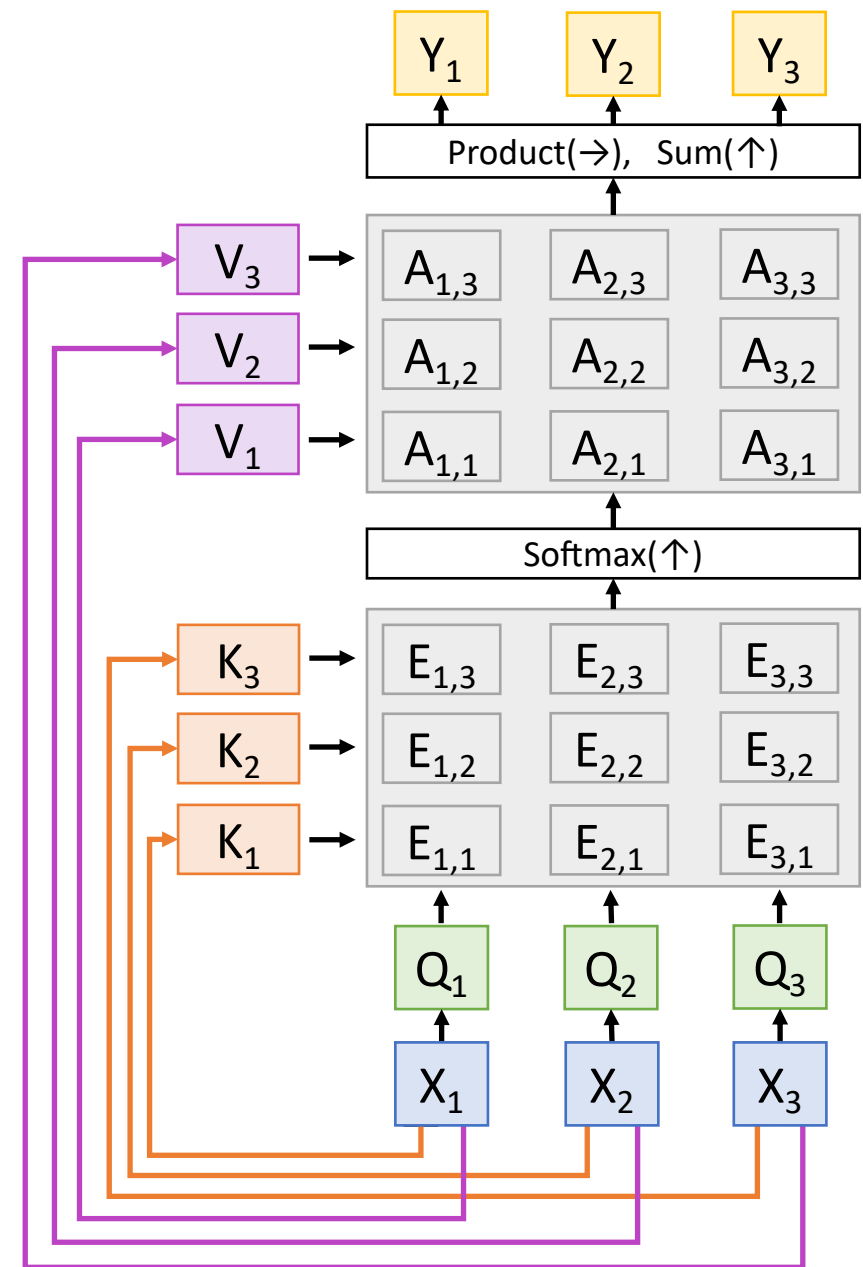
Key vectors: $\mathbf{K} = \mathbf{XW}_K$ (Shape: $N_x \times D_Q$)

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

Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (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: $N_x \times D_Q$)

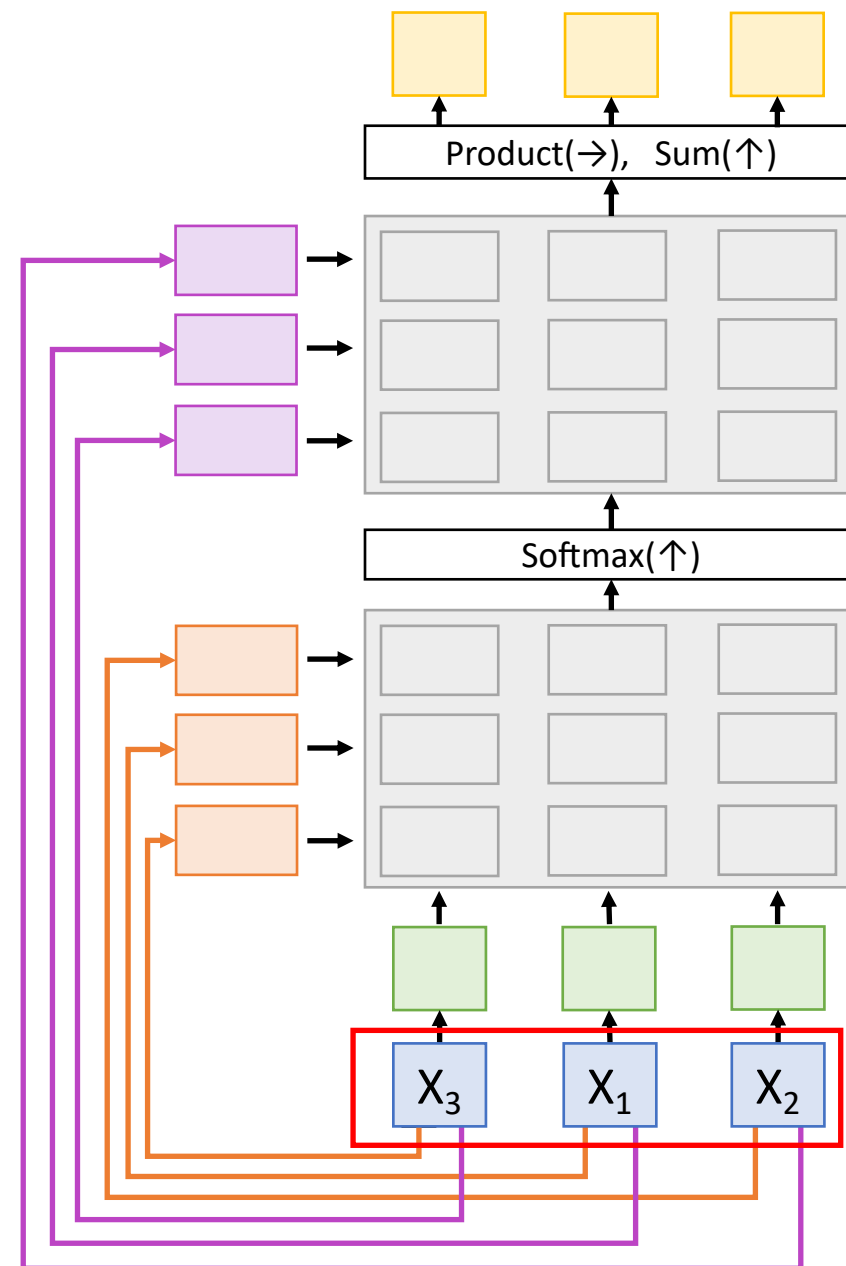
Value Vectors: $\mathbf{V} = \mathbf{XW}_V$ (Shape: $N_x \times D_V$)

Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (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: $N_x \times D_V$) $Y_i = \sum_j A_{i,j} \mathbf{V}_j$

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: $N_x \times D_V$)

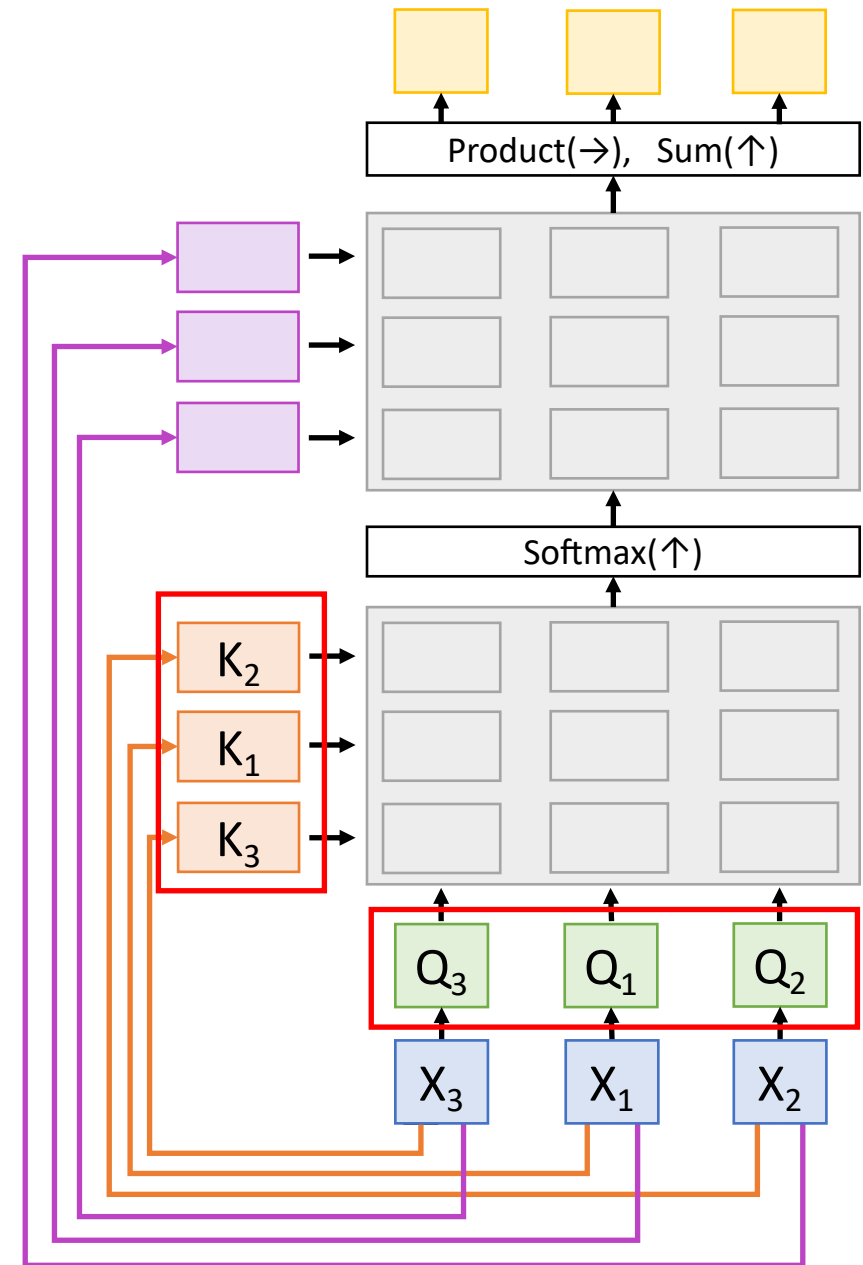
Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: $N_x \times N_x$) $E_{i,j} = (\mathbf{Q}_i \cdot \mathbf{K}_j) / \sqrt{D_Q}$

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

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

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: $N_x \times D_V$)

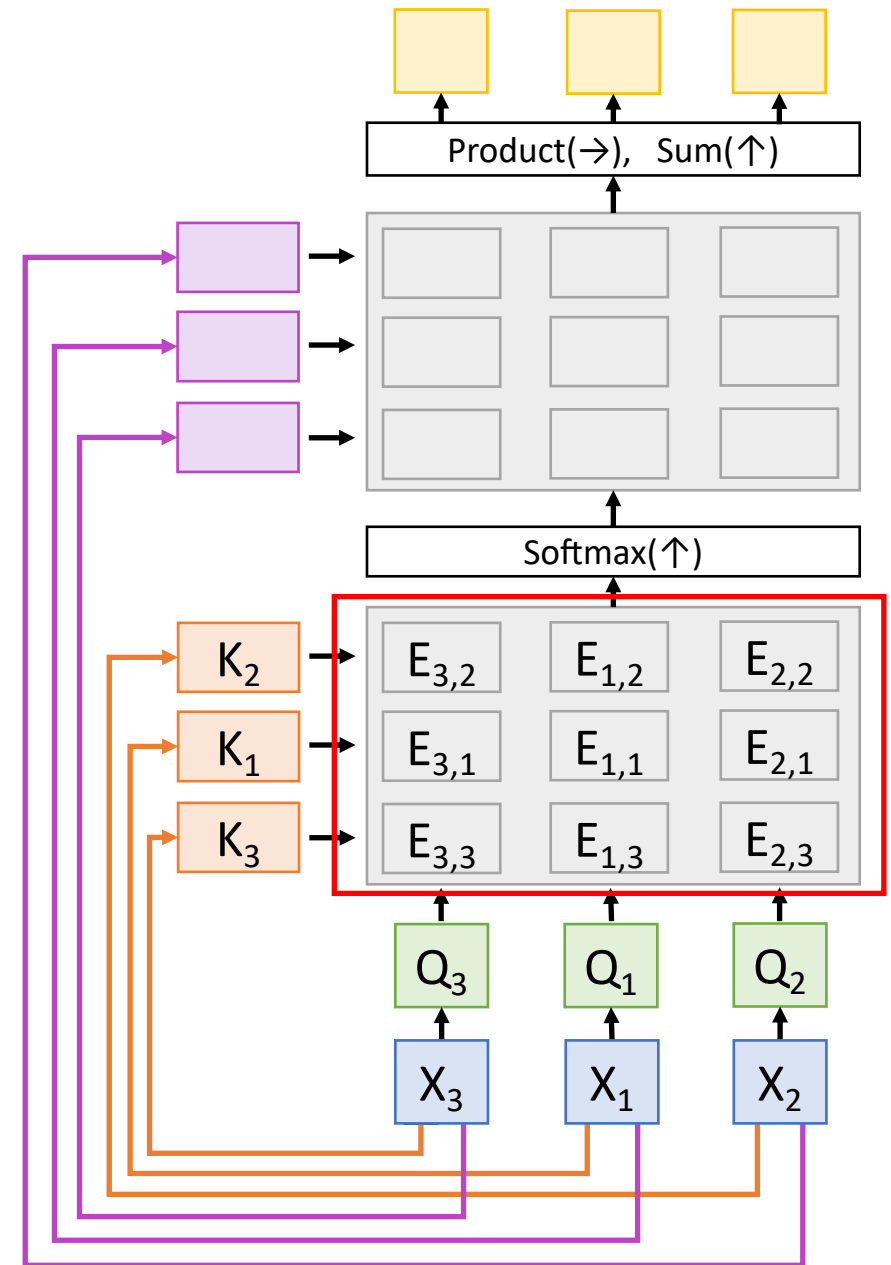
Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: $N_x \times N_x$) $E_{i,j} = (\mathbf{Q}_i \cdot \mathbf{K}_j) / \sqrt{D_Q}$

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

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

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: $N_x \times D_V$)

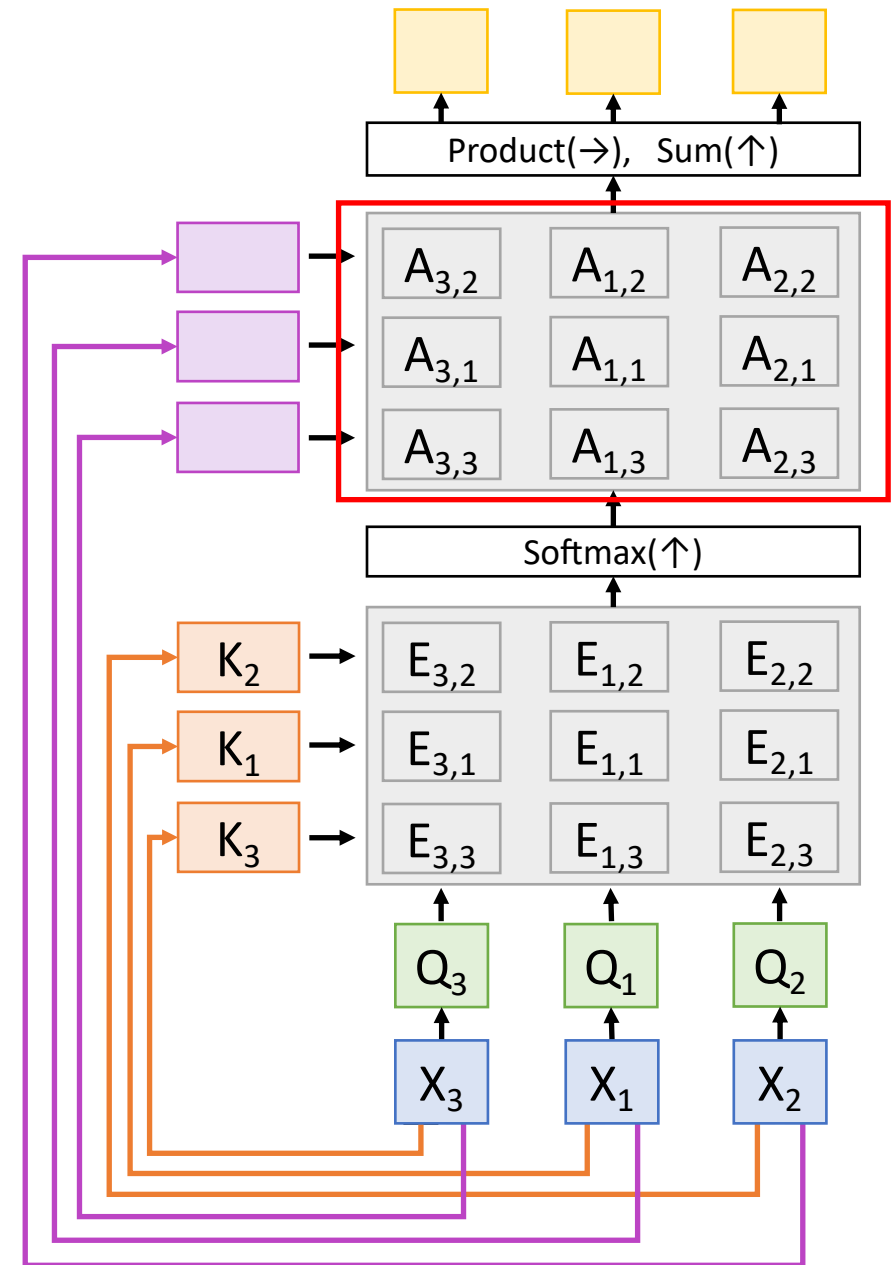
Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: $N_x \times N_x$) $E_{i,j} = (\mathbf{Q}_i \cdot \mathbf{K}_j) / \sqrt{D_Q}$

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

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

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: $D_x \times D_K$)

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: $N_x \times D_K$)

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

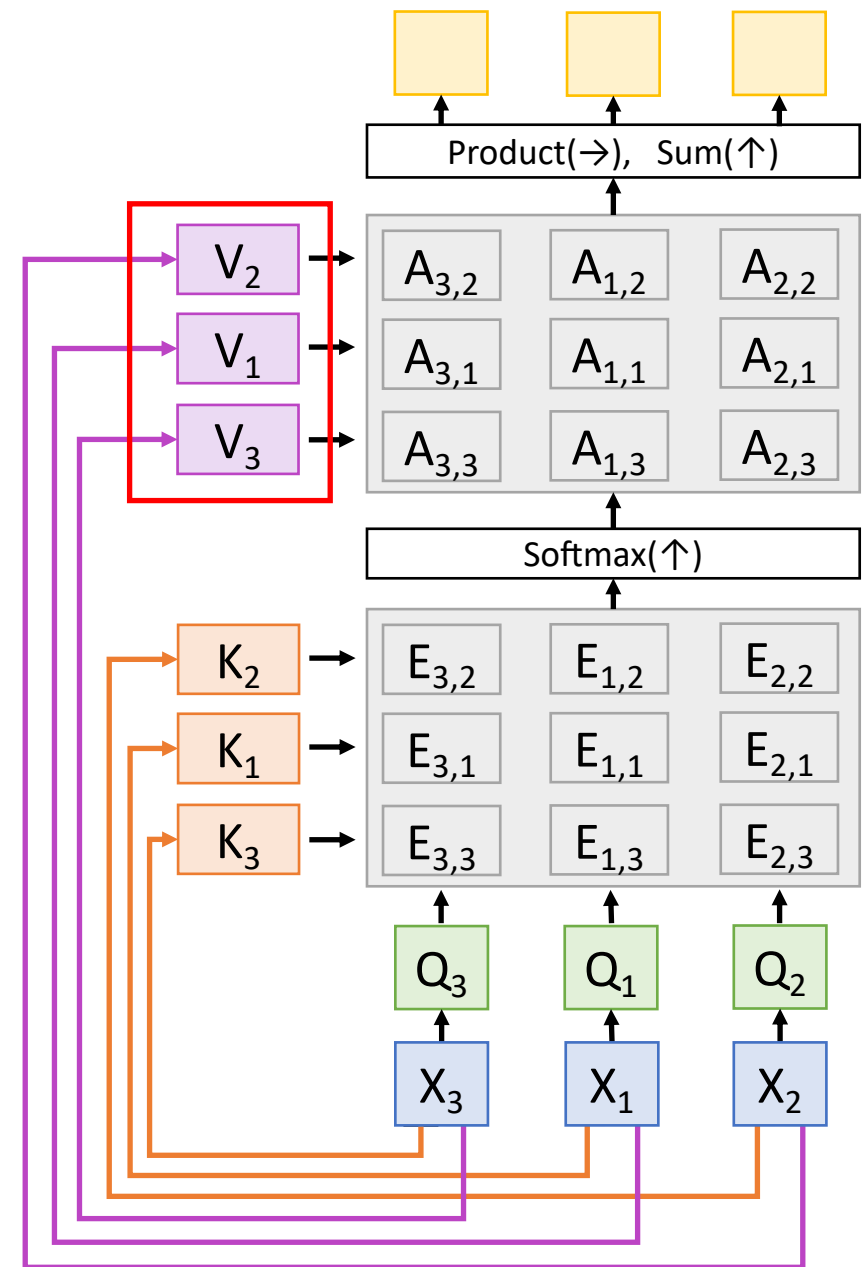
Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: $N_x \times N_x$) $E_{i,j} = (\mathbf{Q}_i \cdot \mathbf{K}_j) / \sqrt{D_Q}$

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

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

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: $N_x \times D_V$)

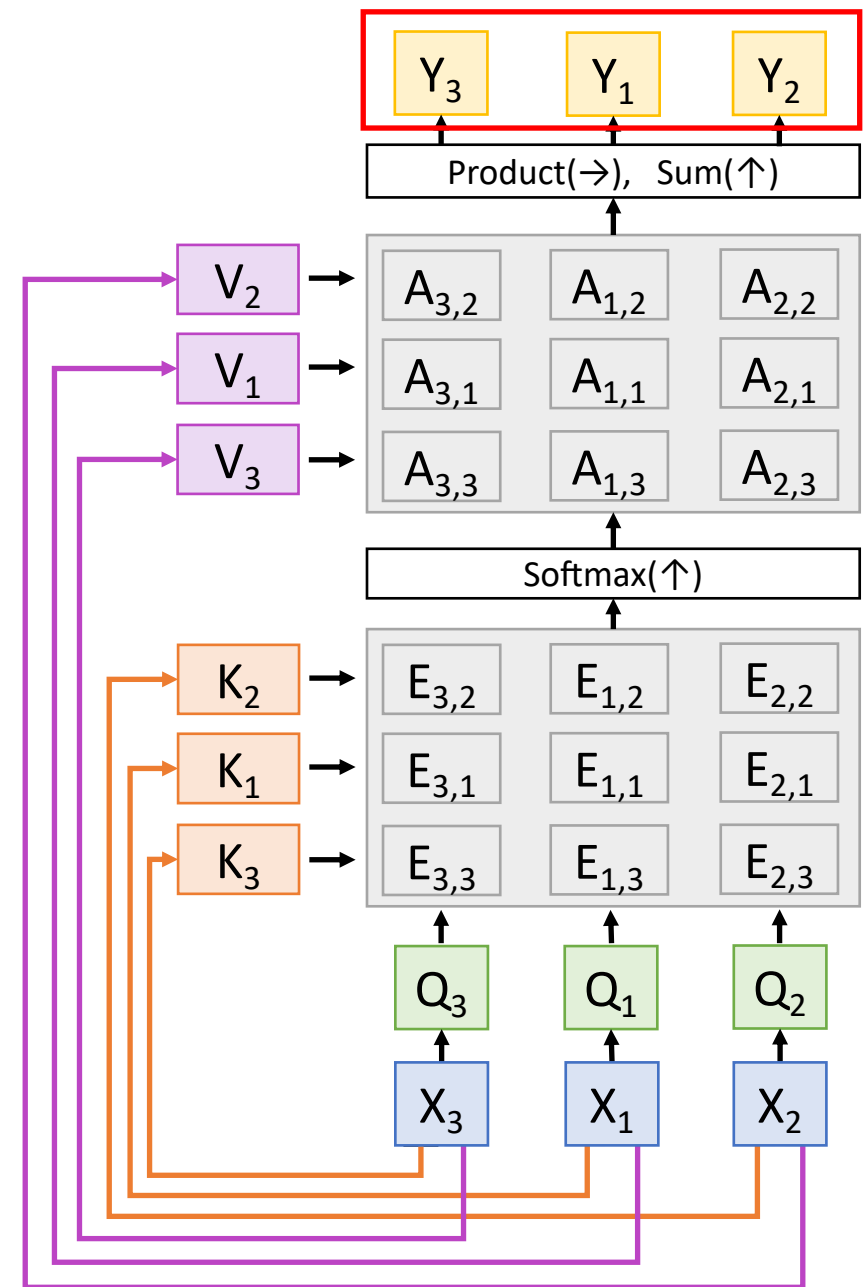
Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: $N_x \times N_x$) $E_{i,j} = (\mathbf{Q}_i \cdot \mathbf{K}_j) / \sqrt{D_Q}$

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

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

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: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (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: $N_x \times N_x$)

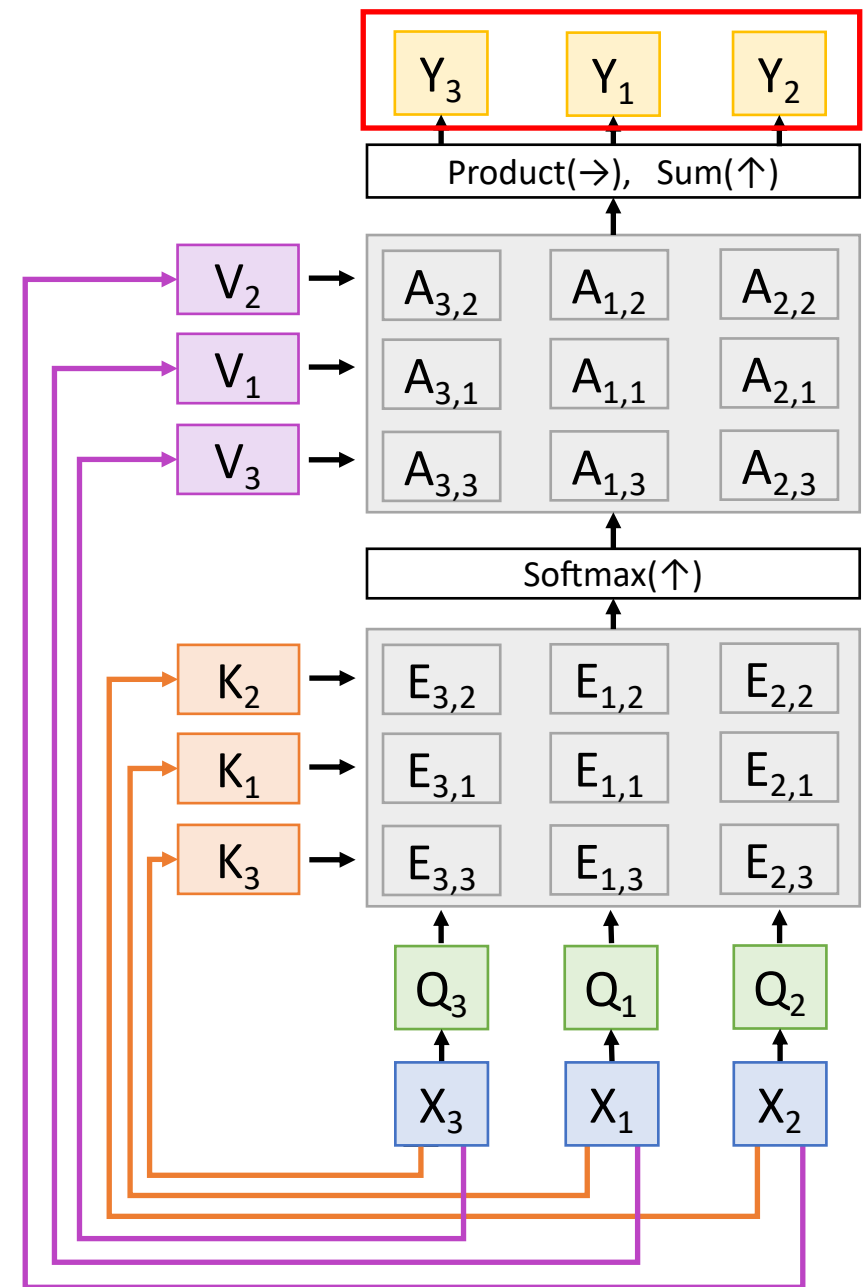
Output vectors: $\mathbf{Y} = \mathbf{AV}$ (Shape: $N_x \times D_V$) $Y_i = \sum_j A_{i,j} \mathbf{V}_j$

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: $N_x \times D_Q$)

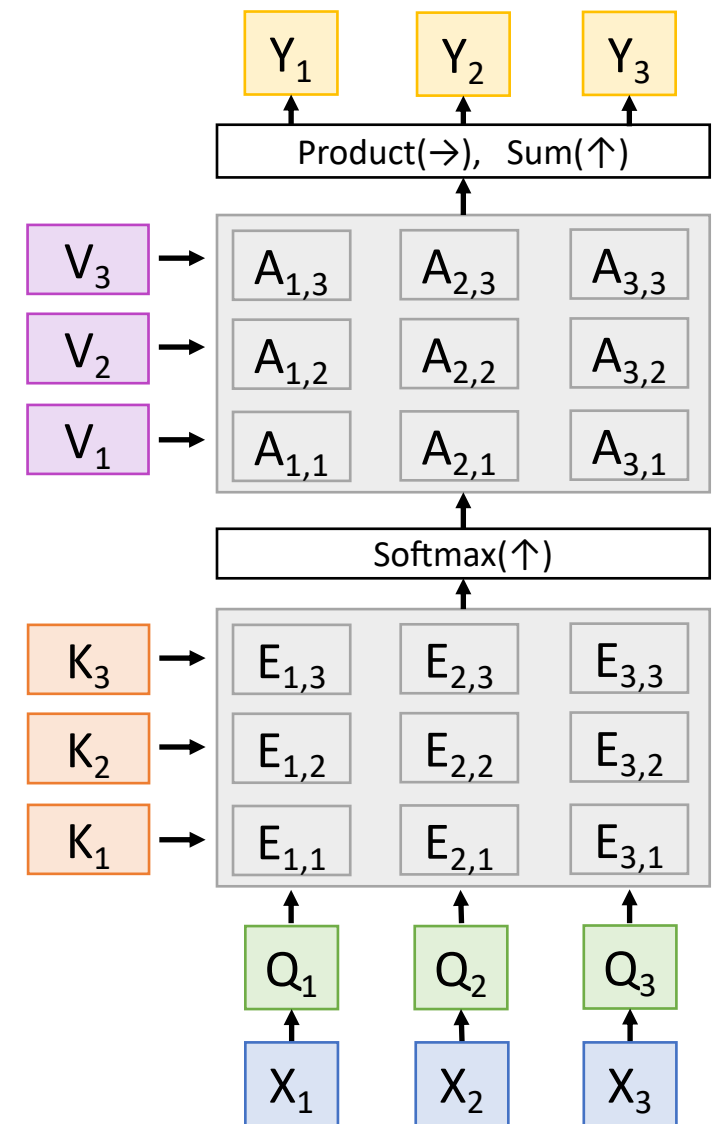
Value Vectors: $\mathbf{V} = \mathbf{XW}_V$ (Shape: $N_x \times D_V$)

Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (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 doesn't
"know" the order of the
vectors it is processing!



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

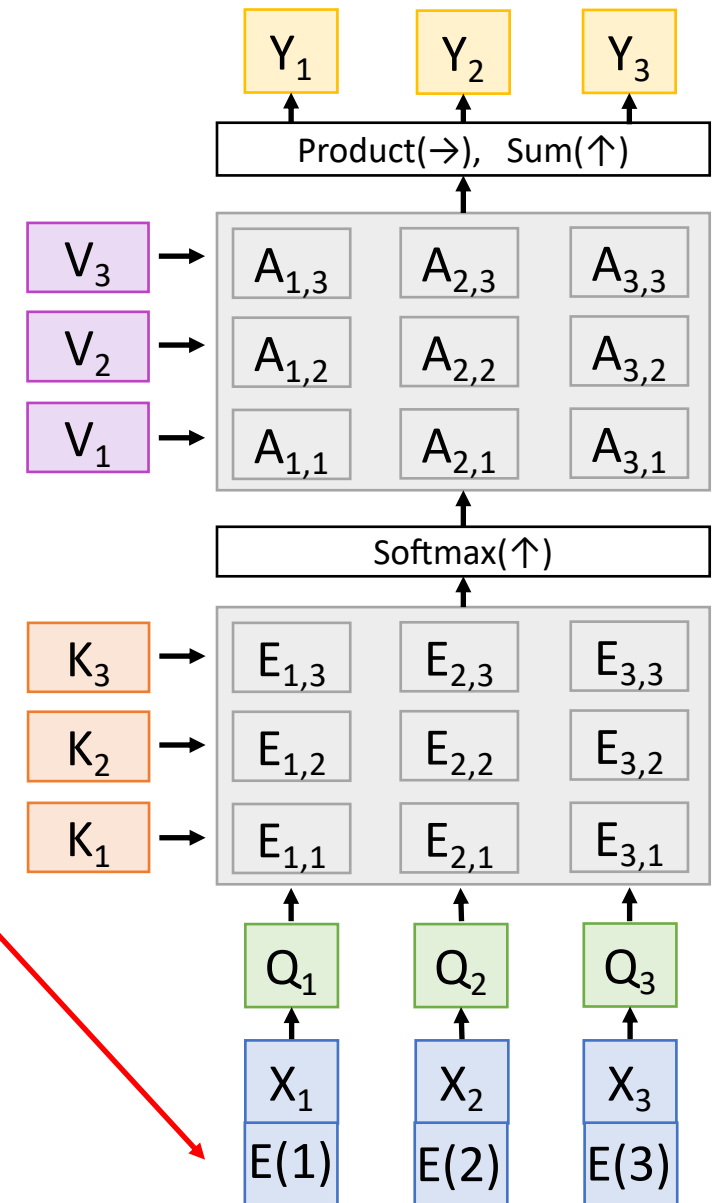
Attention weights: $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$ (Shape: $N_x \times N_x$)

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

Self attention 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

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

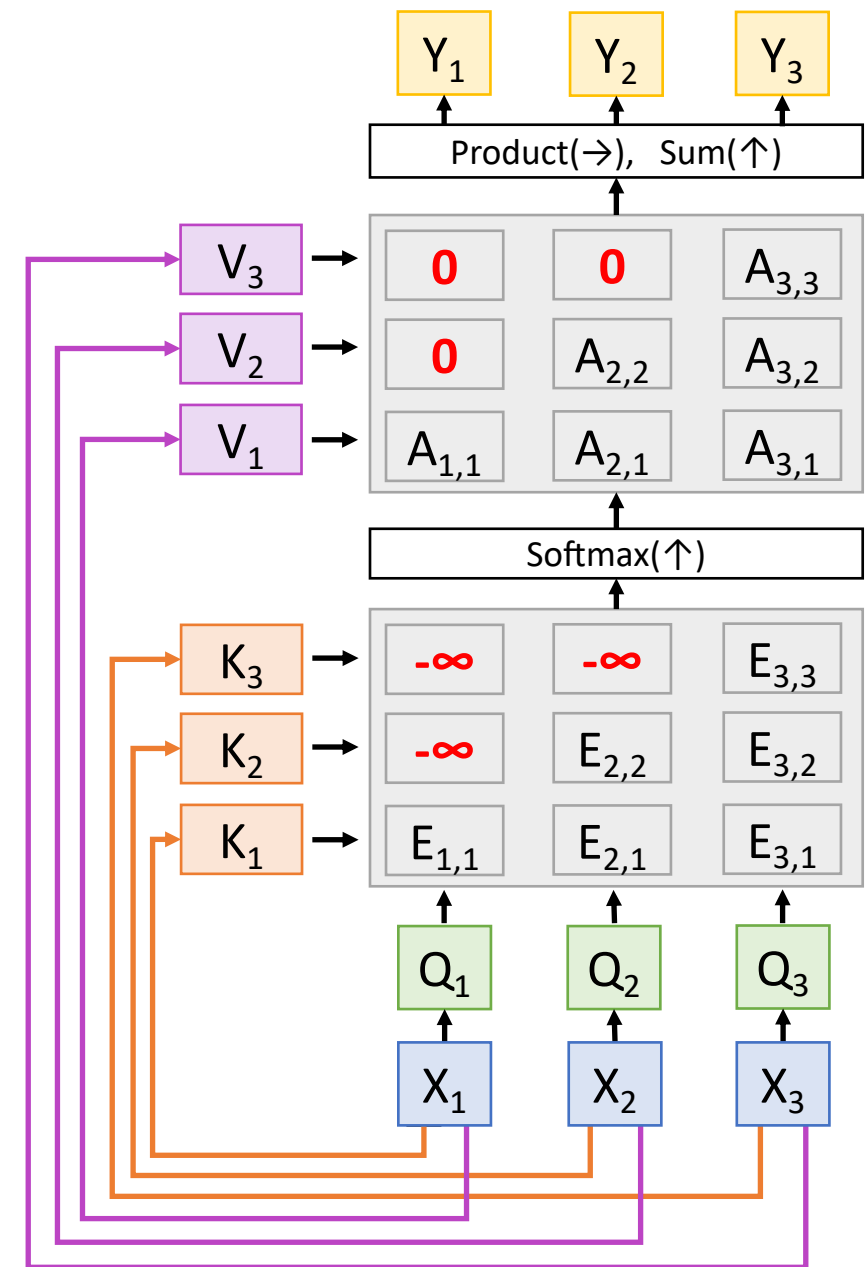
Key vectors: $\mathbf{K} = \mathbf{XW}_K$ (Shape: $N_x \times D_Q$)

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

Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (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: $N_x \times D_V$) $Y_i = \sum_j A_{i,j} \mathbf{V}_j$



Masked Self-Attention Layer

Don't let vectors "look ahead" in the sequence
Used for language modeling (predict next word)

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

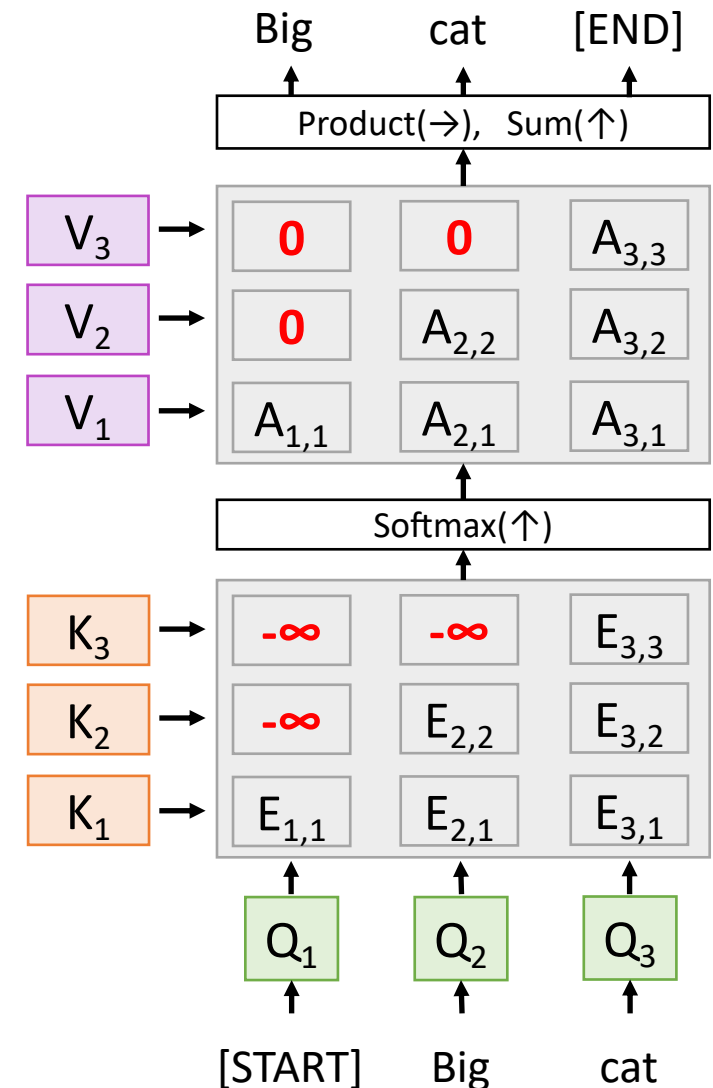
Key vectors: $\mathbf{K} = \mathbf{XW}_K$ (Shape: $N_x \times D_Q$)

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

Similarities: $\mathbf{E} = \mathbf{QK}^T / \sqrt{D_Q}$ (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: $N_x \times D_V$) $Y_i = \sum_j A_{i,j} \mathbf{V}_j$



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: $D_x \times D_K$)

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: $N_x \times D_K$)

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

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

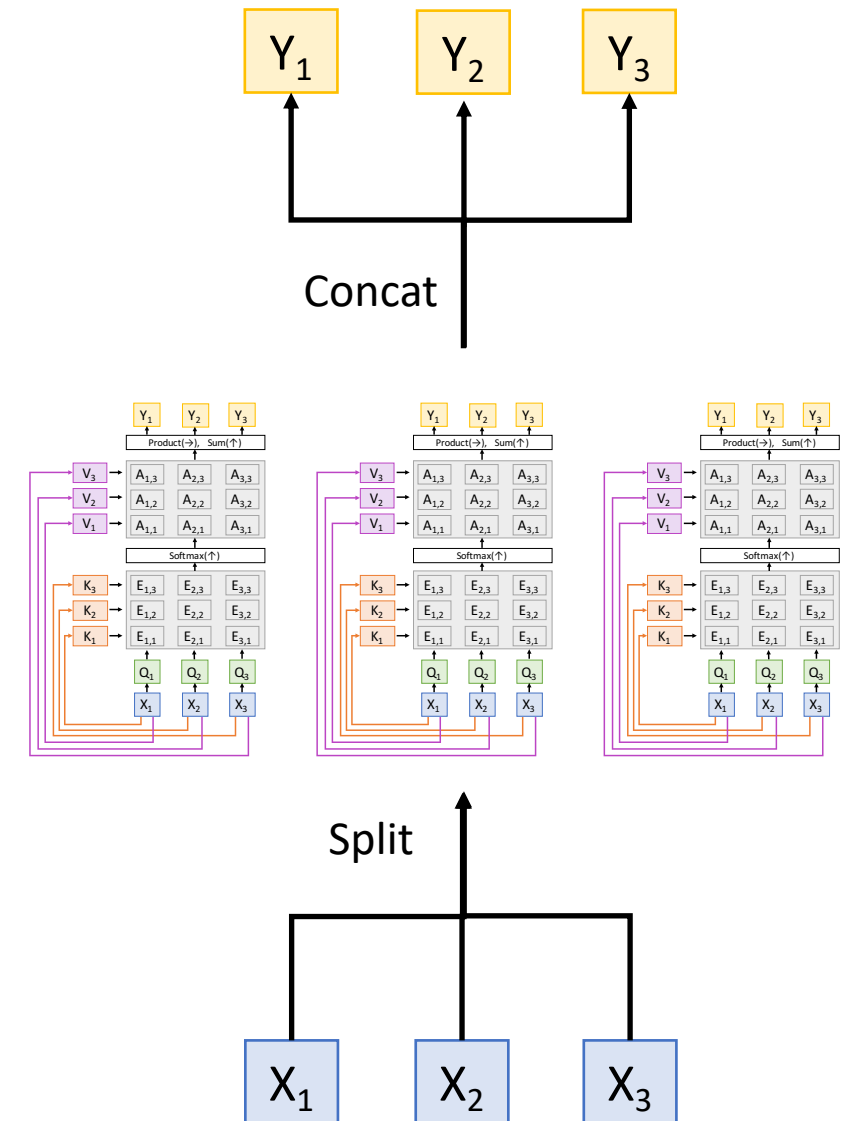
Attention weights: $\mathbf{A} = \text{softmax}(\mathbf{E}, \text{dim}=1)$ (Shape: $N_x \times N_x$)

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

Hyperparameters:

Query dimension D_Q

Number of heads H

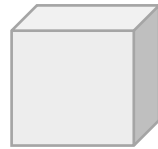
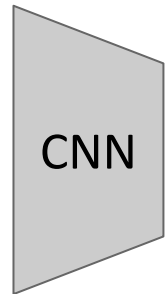


Example: CNN with Self-Attention

Input Image

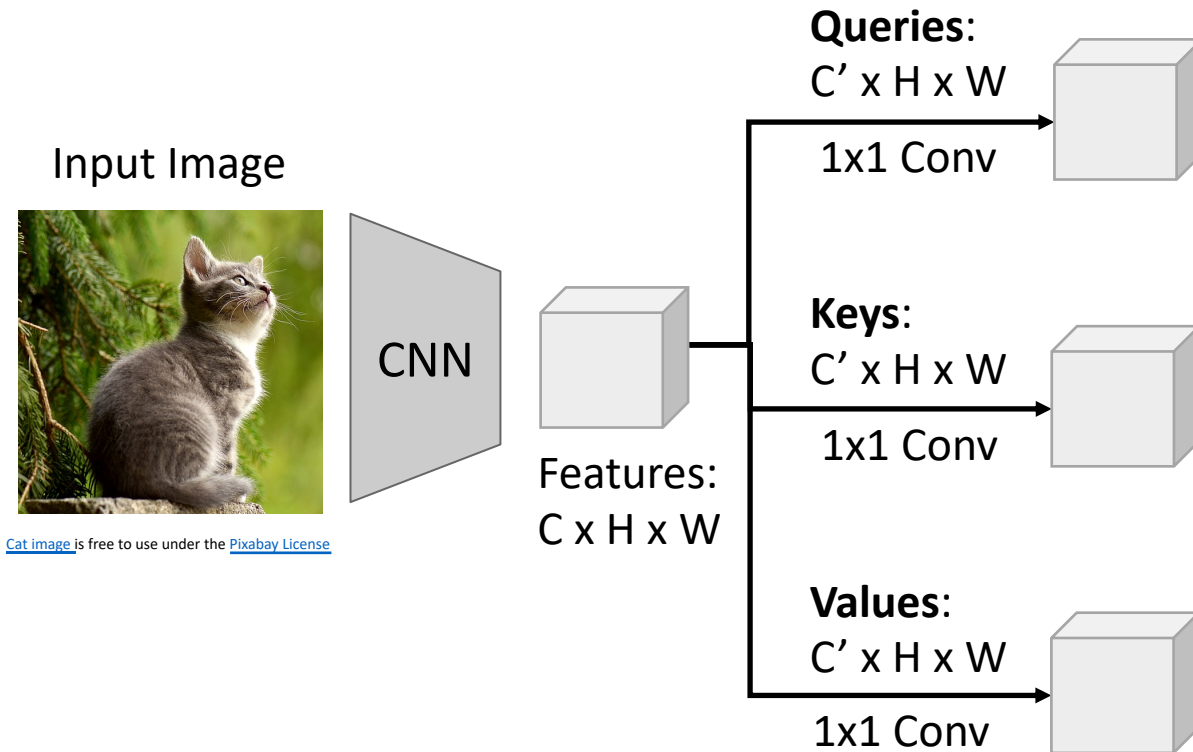


[Cat image](#) is free to use under the [Pixabay License](#)

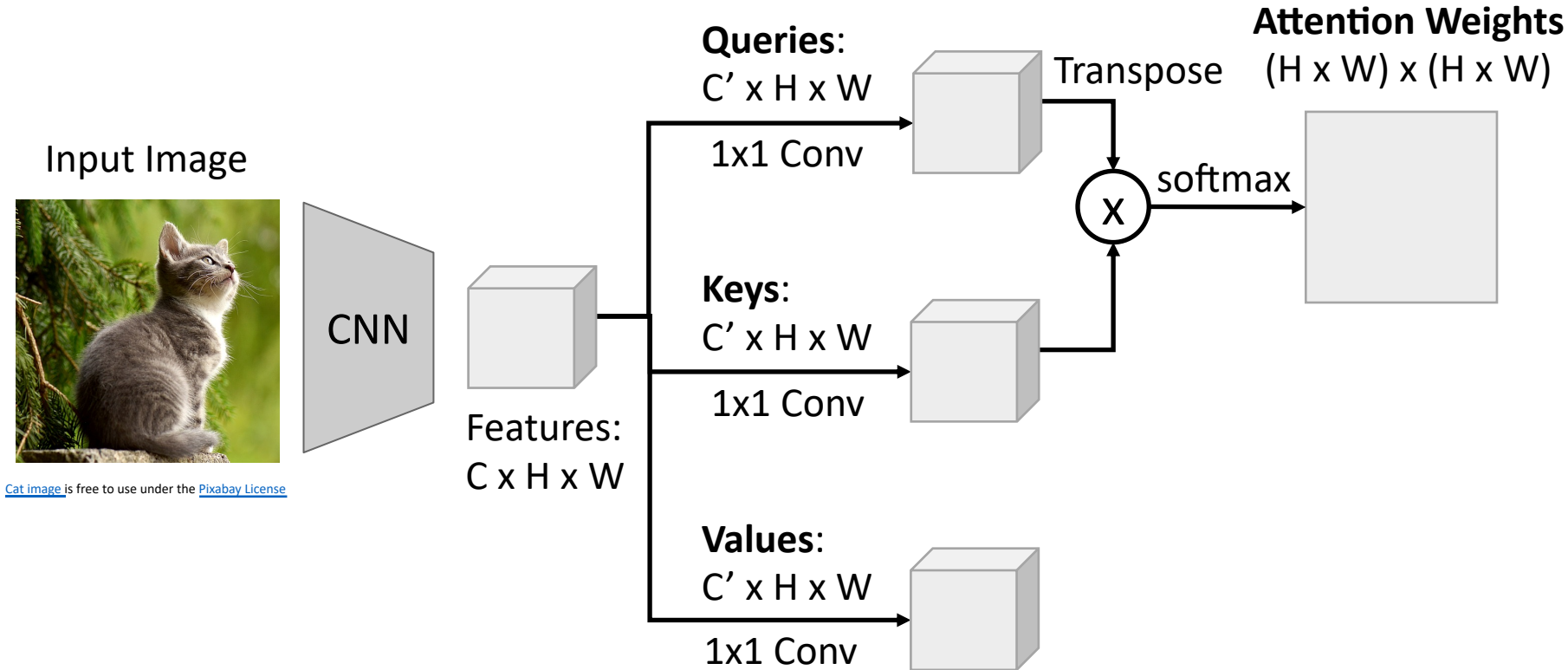


Features:
 $C \times H \times W$

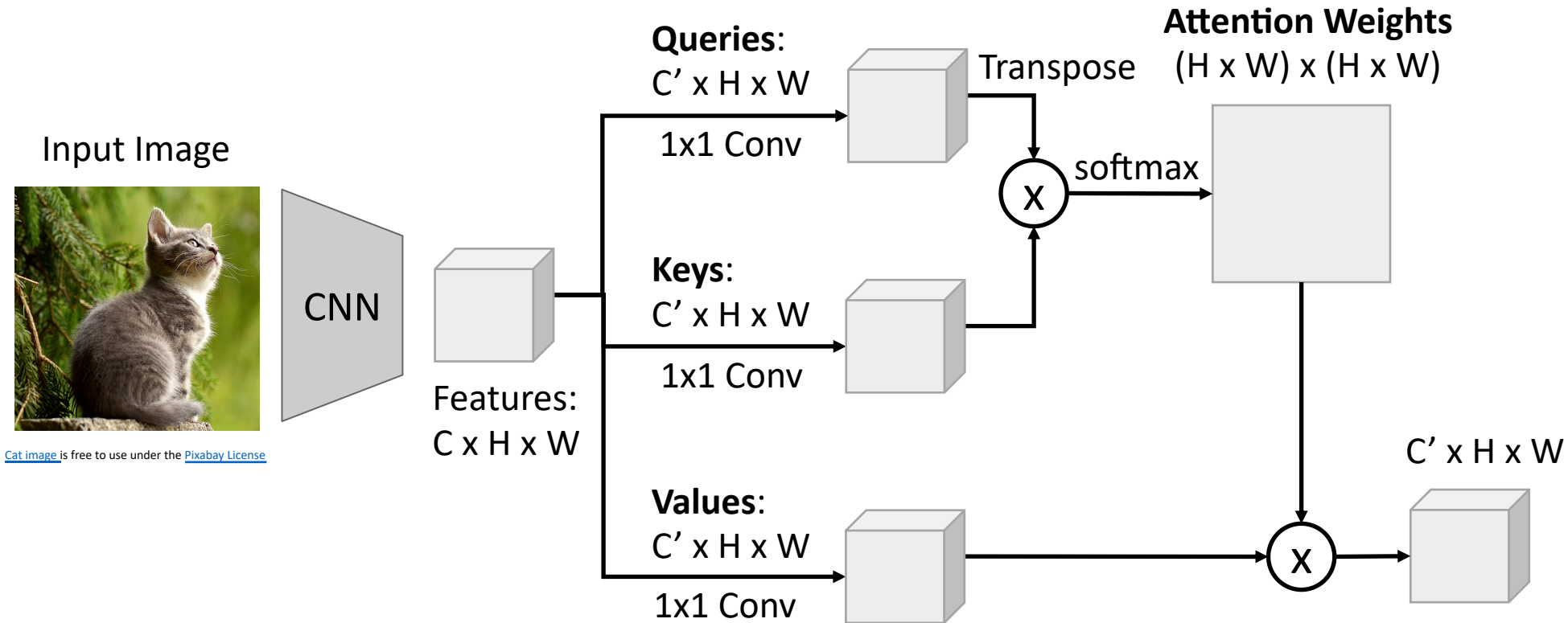
Example: CNN with Self-Attention



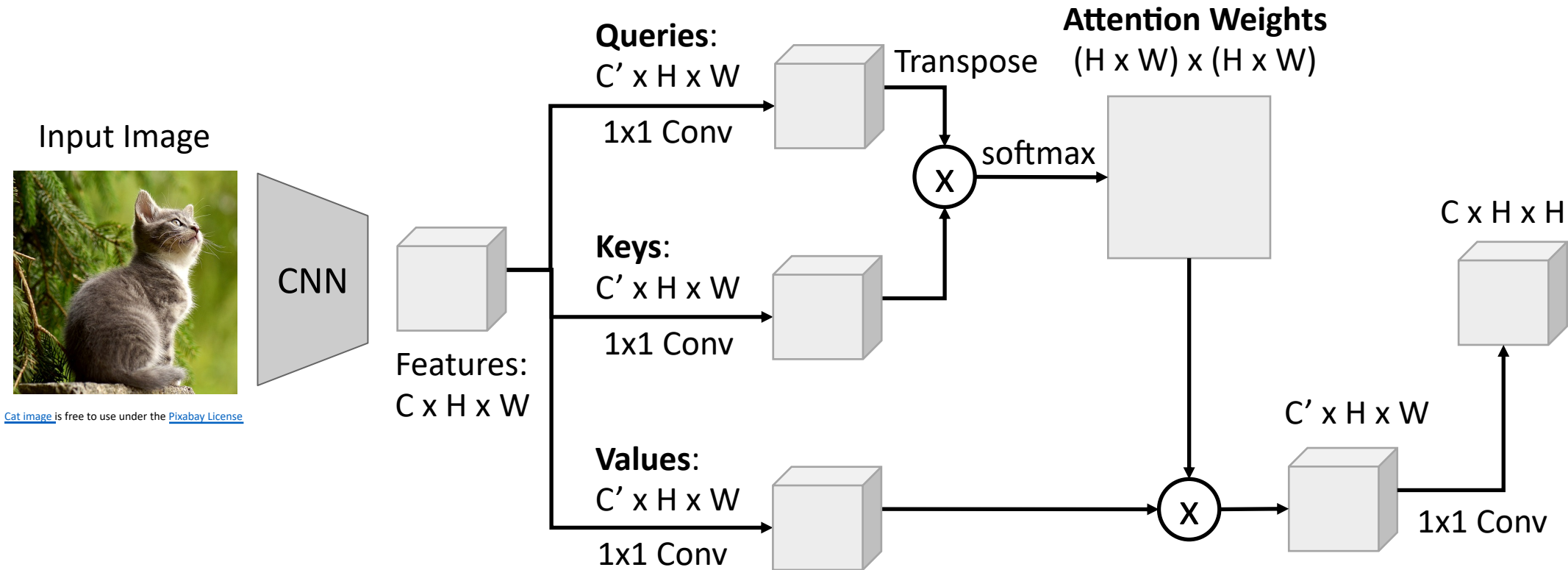
Example: CNN with Self-Attention



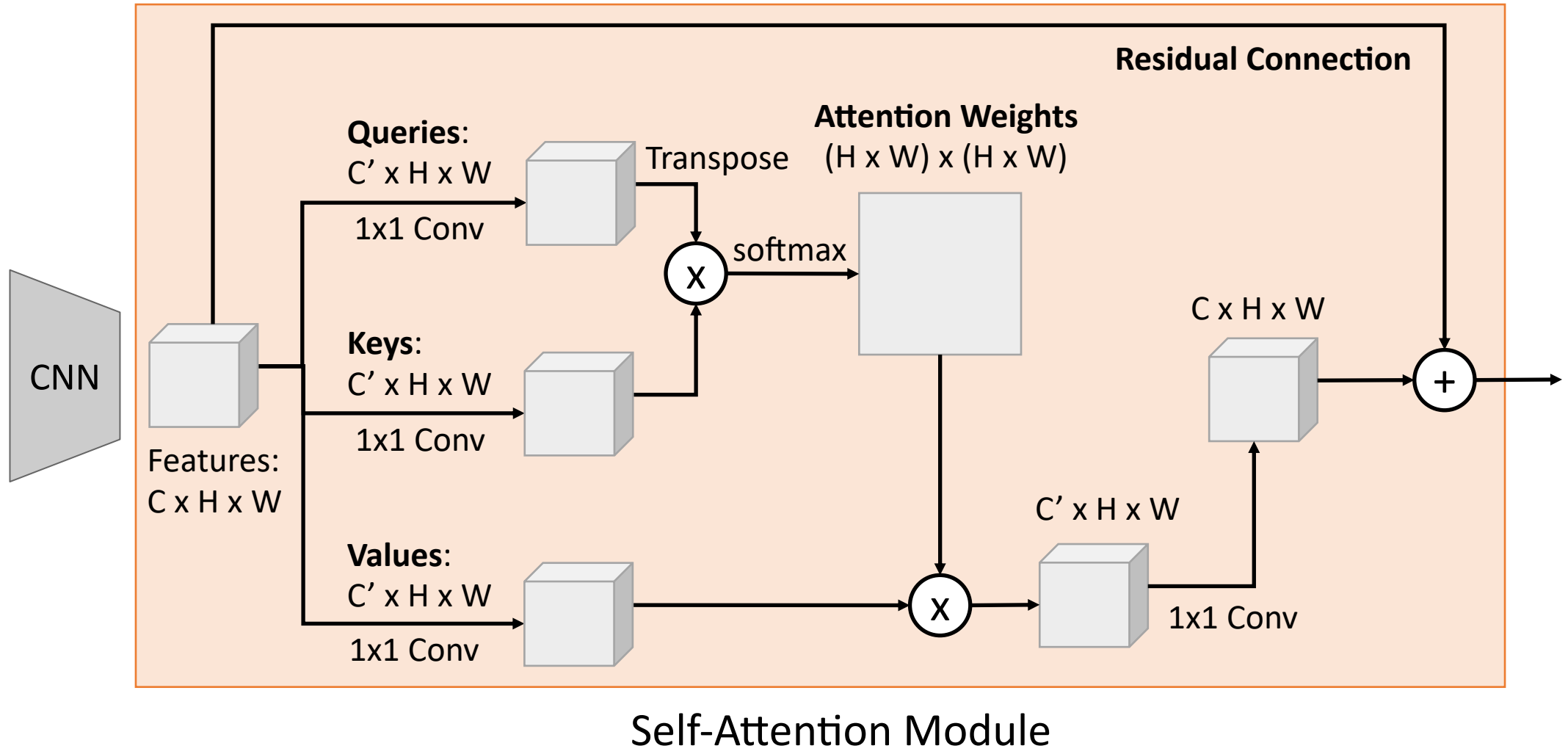
Example: CNN with Self-Attention



Example: CNN with Self-Attention



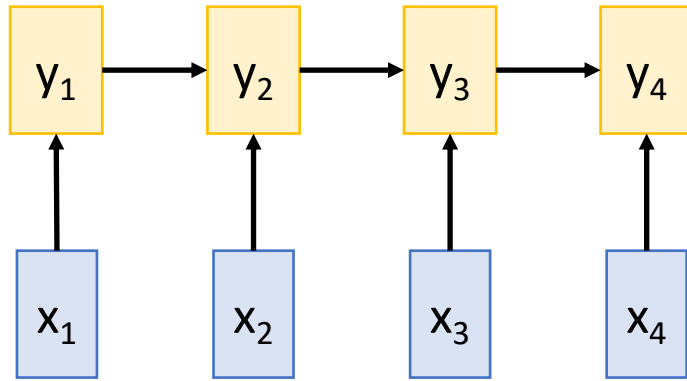
Example: CNN with Self-Attention



Zhang et al, "Self-Attention Generative Adversarial Networks", ICML 2018

Three Ways of Processing Sequences

Recurrent Neural Network



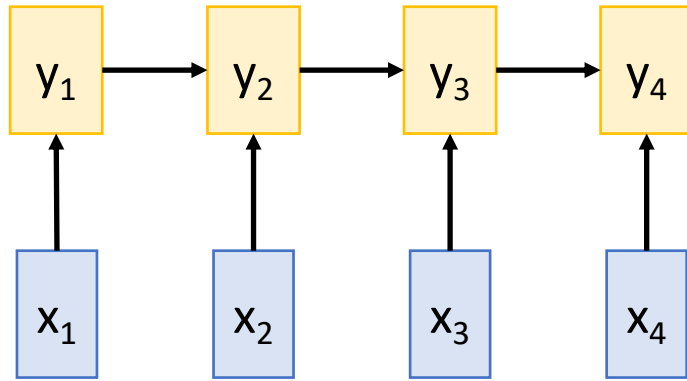
Works on **Ordered Sequences**

(+) **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

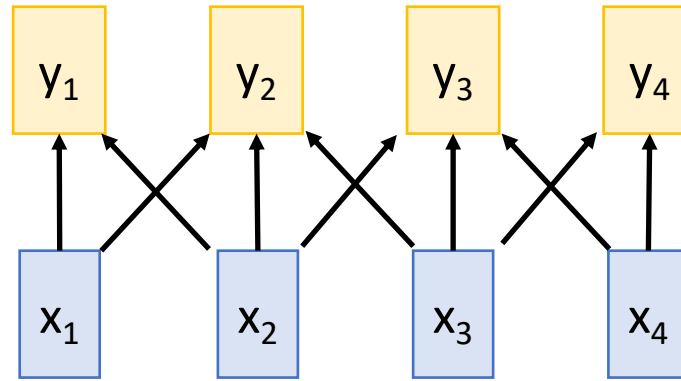


Works on **Ordered Sequences**

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

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

1D Convolution



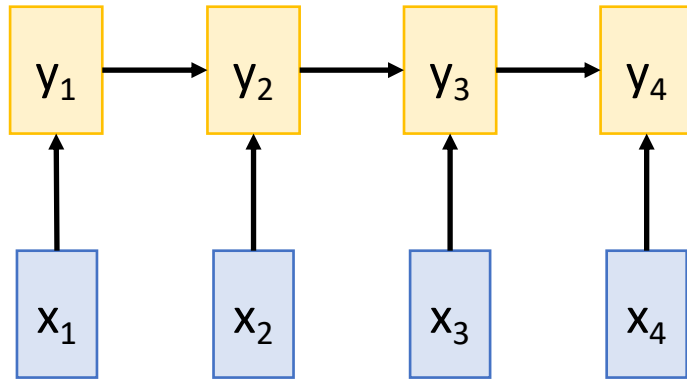
Works on **Multidimensional Grids**

(-) **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

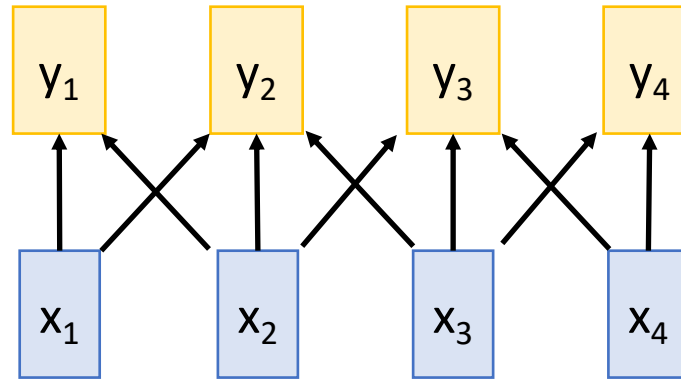
Three Ways of Processing Sequences

Recurrent Neural Network



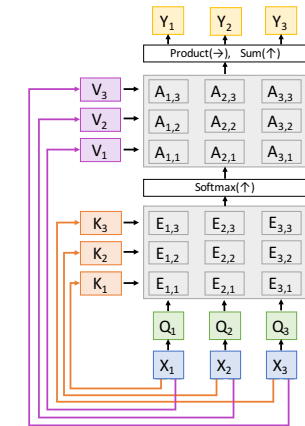
Works on **Ordered Sequences**
(+) **Good at long sequences:** After one RNN layer, h_T "sees" the whole sequence
(-) **Not parallelizable:** need to compute hidden states sequentially

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**

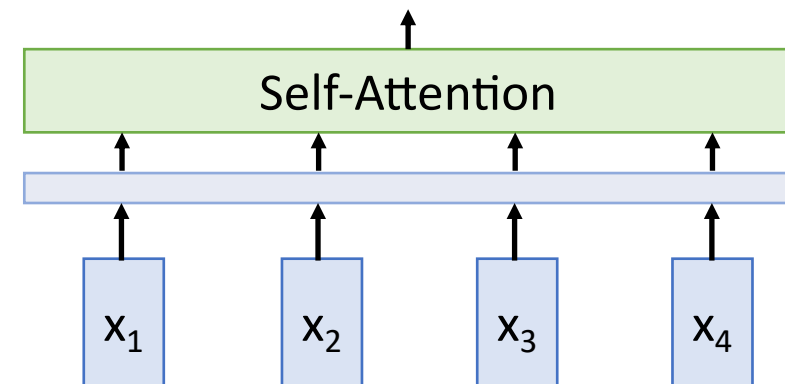
The Transformer



Vaswani et al, "Attention is all you need", NeurIPS 2017

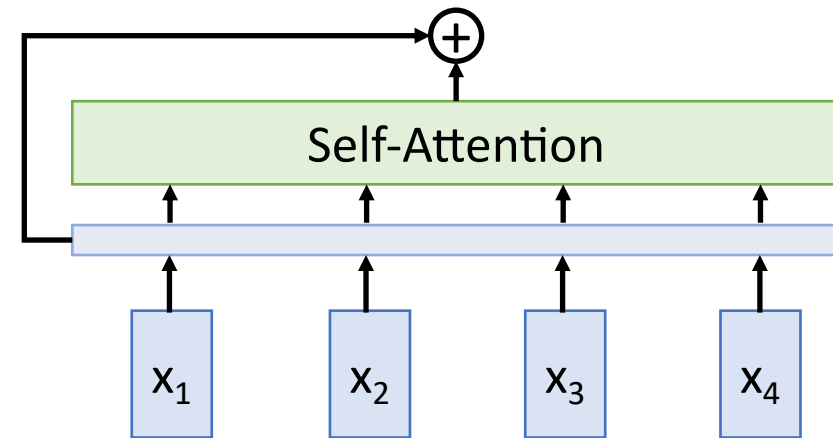
The Transformer

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: γ (Shape: D)

shift: β (Shape: D)

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

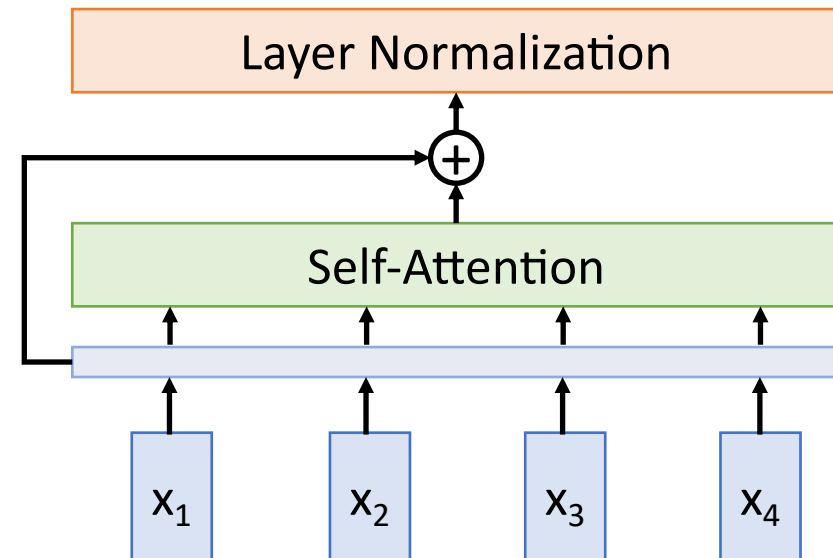
$\sigma_i = (\sum_j (h_{i,j} - \mu_i)^2/D)^{1/2}$ (scalar)

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

$y_i = \gamma * z_i + \beta$

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: γ (Shape: D)

shift: β (Shape: D)

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

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

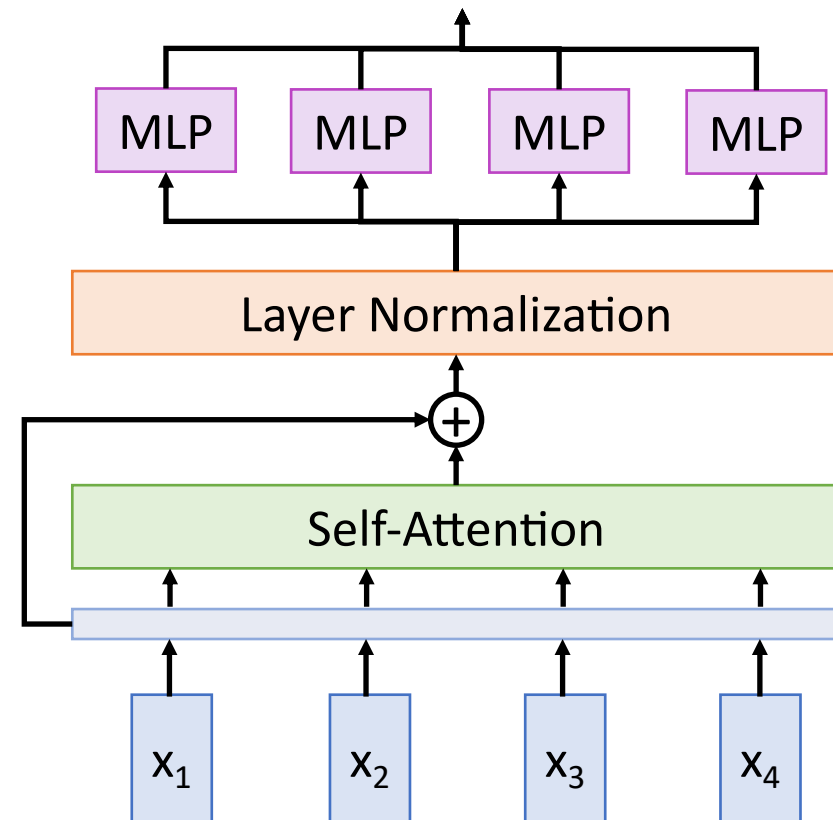
$z_i = (h_i - \mu_i) / \sigma_i$

$y_i = \gamma * z_i + \beta$

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: γ (Shape: D)

shift: β (Shape: D)

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

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

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

$y_i = \gamma * z_i + \beta$

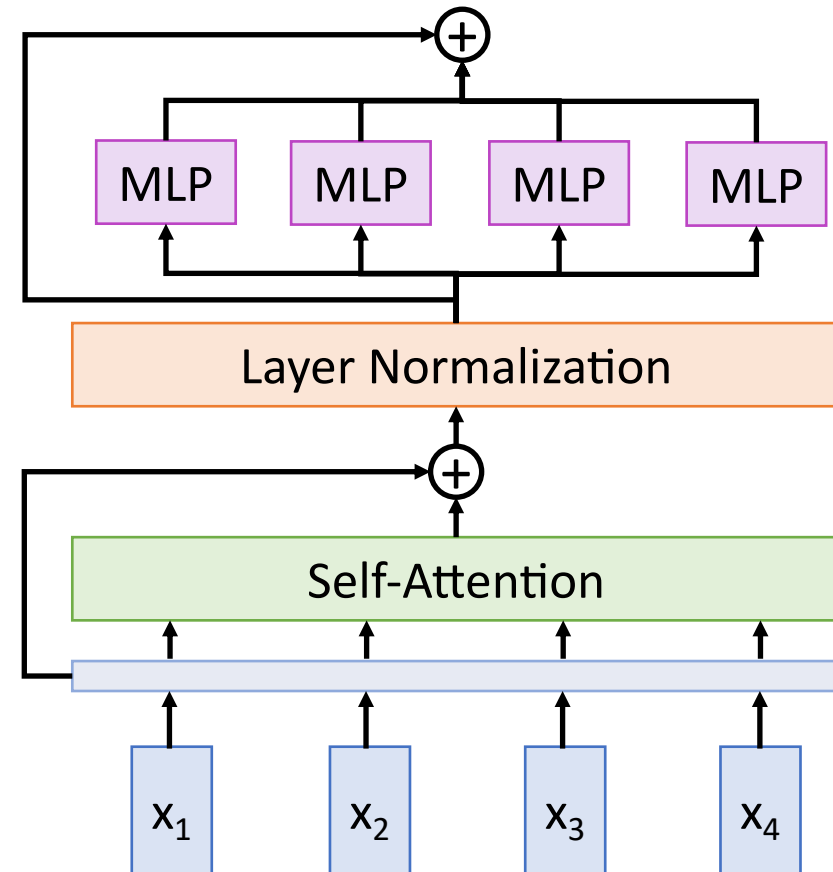
Ba et al, 2016

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: γ (Shape: D)

shift: β (Shape: D)

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

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

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

$y_i = \gamma * z_i + \beta$

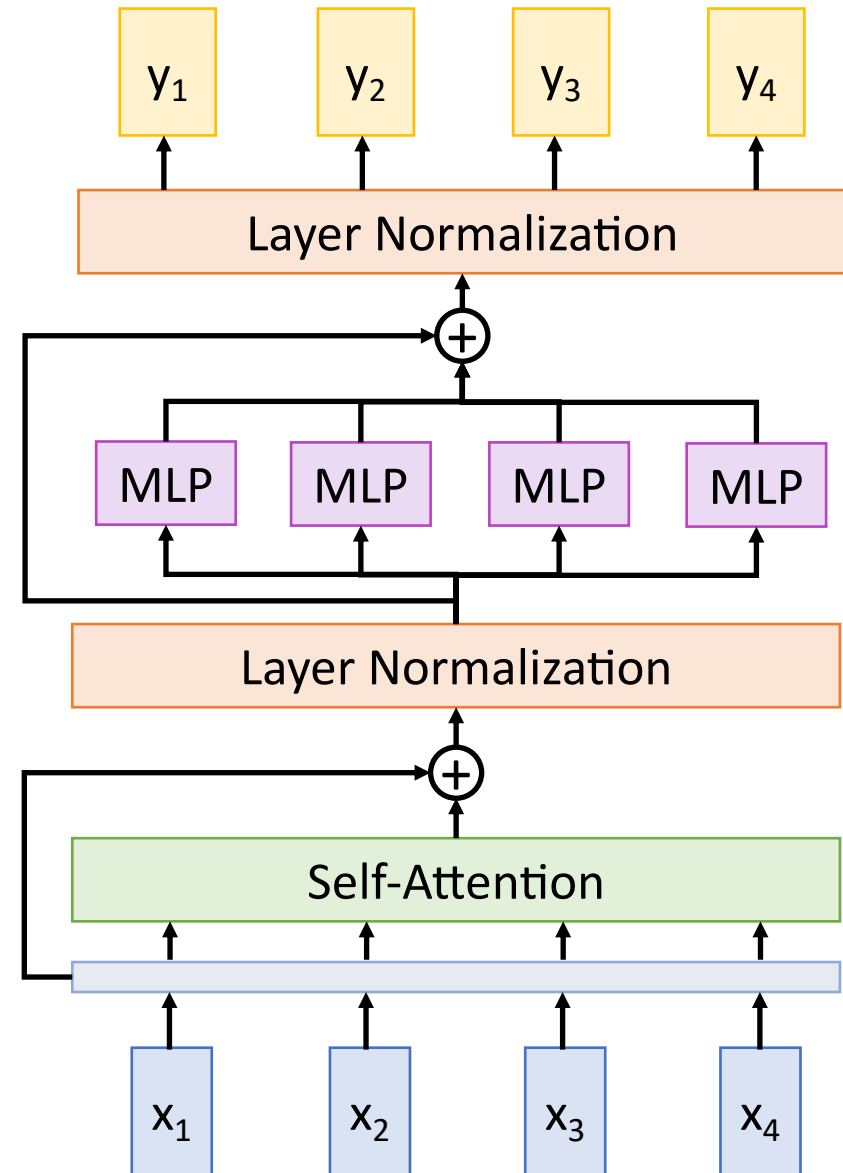
Ba et al, 2016

Residual connection

MLP independently
on each vector

Residual connection

All vectors interact
with each other



The Transformer

Transformer Block:

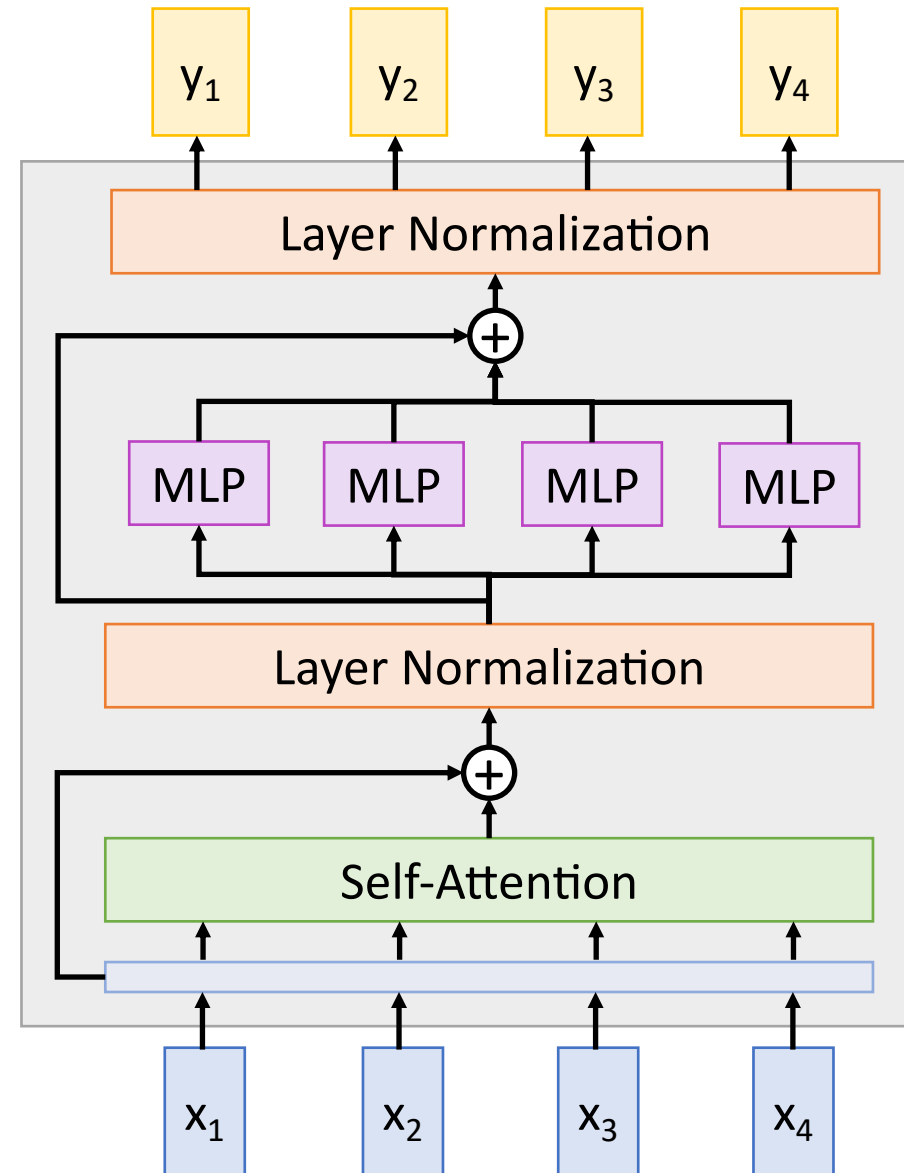
Input: Set of vectors x

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

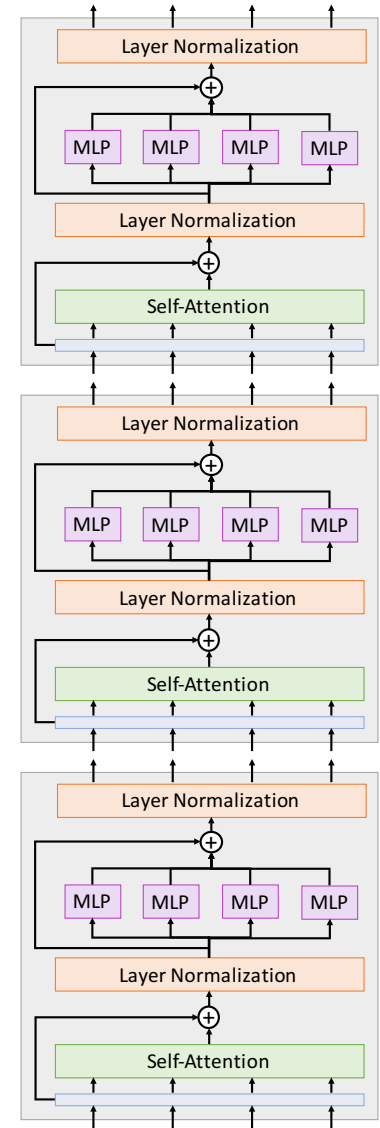
Self-attention is the only interaction between vectors!

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”

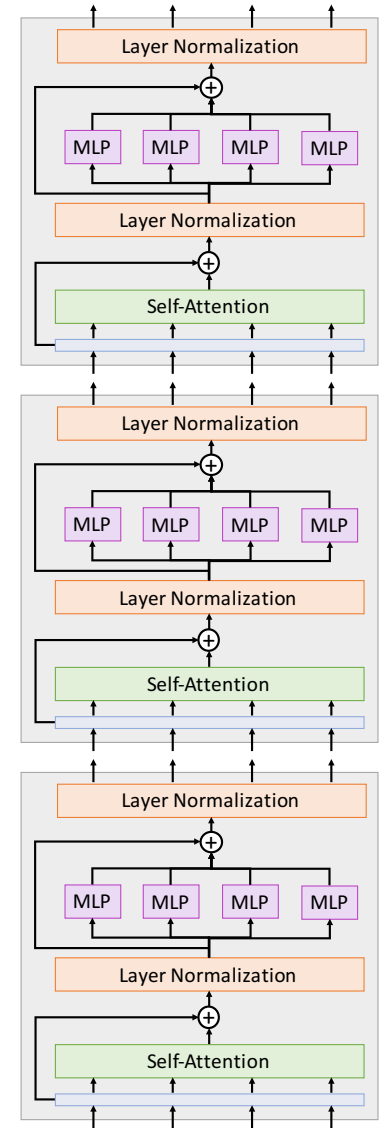
Pretraining:

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)

Vaswani et al, "Attention is all you need", NeurIPS 2017

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	

Devlin et al, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", EMNLP 2018

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)

Yang et al, XLNet: Generalized Autoregressive Pretraining for Language Understanding", 2019
Liu et al, "RoBERTa: A Robustly Optimized BERT Pretraining Approach", 2019

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	48	1600	?	1.5B	40 GB	

Radford et al, "Language models are unsupervised multitask learners", 2019

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	48	1600	?	1.5B	40 GB	
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	48	1600	?	1.5B	40 GB	
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

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	48	1600	?	1.5B	40 GB	
Megatron-LM	72	3072	32	8.3B	174 GB	512x V100 GPU (9 days)
Turing-NLG	78	4256	28	17B	?	256x V100 GPU

Microsoft, "Turing-NLG: A 17-billion parameter language model by Microsoft", 2020

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	48	1600	?	1.5B	40 GB	
Megatron-LM	72	3072	32	8.3B	174 GB	512x V100 GPU (9 days)
Turing-NLG	78	4256	28	17B	?	256x V100 GPU
GPT-3	96	12288	96	175B	694GB	?

Brown et al, "Language Models are Few-Shot Learners", arXiv 2020

Generated Text from GPT-3

Bold text: Input prompt written by humans
Italics: Completion by GPT-3

Title: United Methodists Agree to Historic Split

Subtitle: Those who oppose gay marriage will form their own denomination

Article:

Generated Text from GPT-3

Bold text: Input prompt written by humans
Italics: Completion by GPT-3

Title: United Methodists Agree to Historic Split

Subtitle: Those who oppose gay marriage will form their own denomination

Article: After two days of intense debate, the United Methodist Church has agreed to a historic split - one that is expected to end in the creation of a new denomination, one that will be "theologically and socially conservative," according to The Washington Post. The majority of delegates attending the church's annual General Conference in May voted to strengthen a ban on the ordination of LGBTQ clergy and to write new rules that will "discipline" clergy who officiate at same-sex weddings. But those who opposed these measures have a new plan: They say they will form a separate denomination by 2020, calling their church the Christian Methodist denomination. The Post notes that the denomination, which claims 12.5 million members, was in the early 20th century the "largest Protestant denomination in the U.S.," but that it has been shrinking in recent decades. The new split will be the second in the church's history. The first occurred in 1968, when roughly 10 percent of the denomination left to form the Evangelical United Brethren Church. The Post notes that the proposed split "comes at a critical time for the church, which has been losing members for years," which has been "pushed toward the brink of a schism over the role of LGBTQ people in the church." Gay marriage is not the only issue that has divided the church. In 2016, the denomination was split over ordination of transgender clergy, with the North Pacific regional conference voting to ban them from serving as clergy, and the South Pacific regional conference voting to allow them.

GPT-3: Programming by prompt

Bold text: Input prompt written by humans

Italics: Completion by GPT-3

Poor English input: I eated the purple berries.

Good English output: I ate the purple berries.

Poor English input: Thank you for picking me as your designer. I'd appreciate it.

Good English output: Thank you for choosing me as your designer. I appreciate it.

Poor English input: The mentioned changes have done. or I did the alteration that you requested. or I changed things you wanted and did the modifications.

Good English output: The requested changes have been made. or I made the alteration that you requested. or I changed things you wanted and made the modifications.

Poor English input: I'd be more than happy to work with you in another project.

Good English output:

GPT-3: Programming by prompt

Bold text: Input prompt written by humans

Italics: Completion by GPT-3

Poor English input: I eated the purple berries.

Good English output: I ate the purple berries.

Poor English input: Thank you for picking me as your designer. I'd appreciate it.

Good English output: Thank you for choosing me as your designer. I appreciate it.

Poor English input: The mentioned changes have done. or I did the alteration that you requested. or I changed things you wanted and did the modifications.

Good English output: The requested changes have been made. or I made the alteration that you requested. or I changed things you wanted and made the modifications.

Poor English input: I'd be more than happy to work with you in another project.

Good English output: *I'd be more than happy to work with you on another project.*

GPT-3: Literary Style Transfer

Topic: humor: literary parodies and homages, pastiches, style parodies.

Parodies of the fantasy novel series *Harry Potter* in the style of various famous authors:

By Ernest Hemingway:

"It was a cold day on Privet Drive. A child cried. Harry felt nothing.

GPT-3: Literary Style Transfer

Topic: humor: literary parodies and homages, pastiches, style parodies.

Parodies of the fantasy novel series *Harry Potter* in the style of various famous authors:

By Ernest Hemingway:

"It was a cold day on Privet Drive. A child cried. Harry felt nothing. He was dryer than dust. He had been silent too long. He had not felt love. He had scarcely felt hate. Yet the Dementor's Kiss killed nothing. Death didn't leave him less dead than he had been a second before. It wasn't about living or dying really. It wasn't about death. It was about the Dark Lord, his creator, his engenderer. He was not resurrected. He was created again. He was whole again in the courtyard of Malfoy Manor."

GPT-3: Literary Style Transfer

Topic: humor: literary parodies and homages, pastiches, style parodies.

Parodies of the fantasy novel series *Harry Potter* in the style of various famous authors:

By Ernest Hemingway:

"It was a cold day on Privet Drive. A child cried. Harry felt nothing. He was dryer than dust. He had been silent too long. He had not felt love. He had scarcely felt hate. Yet the Dementor's Kiss killed nothing. Death didn't leave him less dead than he had been a second before. It wasn't about living or dying really. It wasn't about death. It was about the Dark Lord, his creator, his engenderer. He was not resurrected. He was created again. He was whole again in the courtyard of Malfoy Manor."

By Jane Austen:

"It is a truth universally acknowledged, that a broken Harry is in want of a book. This he knows to his cost; pressing that wretched nose against the window of a bookstore in Diagon Alley the last thing before he goes into hiding for several years, where upon, he goes straight to Azkaban."

GPT-3: Literary Style Transfer

Topic: humor: literary parodies and homages, pastiches, style parodies.

Parodies of the fantasy novel series *Harry Potter* in the style of various famous authors:

By Ernest Hemingway:

"It was a cold day on Privet Drive. A child cried. Harry felt nothing. He was dryer than dust. He had been silent too long. He had not felt love. He had scarcely felt hate. Yet the Dementor's Kiss killed nothing. Death didn't leave him less dead than he had been a second before. It wasn't about living or dying really. It wasn't about death. It was about the Dark Lord, his creator, his engenderer. He was not resurrected. He was created again. He was whole again in the courtyard of Malfoy Manor."

By Jane Austen:

"It is a truth universally acknowledged, that a broken Harry is in want of a book. This he knows to his cost; pressing that wretched nose against the window of a bookstore in Diagon Alley the last thing before he goes into hiding for several years, where upon, he goes straight to Azkaban."

By Arthur Conan Doyle:

"Harry pushed at the swinging doors of the bookshop hard, and nearly knocked himself unconscious. He staggered in with his ungainly package, his cheeks scarlet with cold and the shame of having chosen the wrong month to go Christmas shopping. The proprietor of the store, however, didn't cast even a cursory glance at him, being far more interested in an enormous hunk of wood lying in the middle of the floor, which certainly looked like a gallows. Yes, the proprietor said to a reedy old man wearing a bowler hat and a forlorn expression that made Harry want to kick him, I can rent you such a gallows for a small fee."

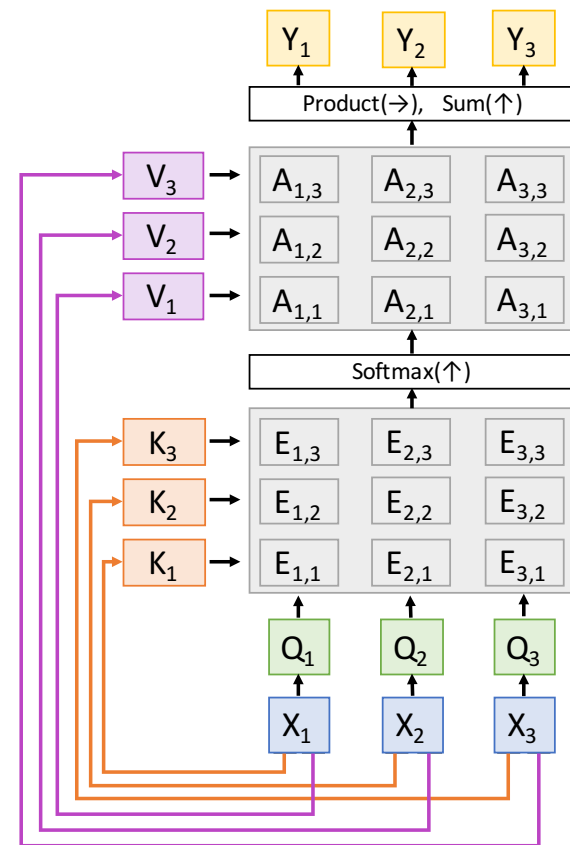
Summary

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

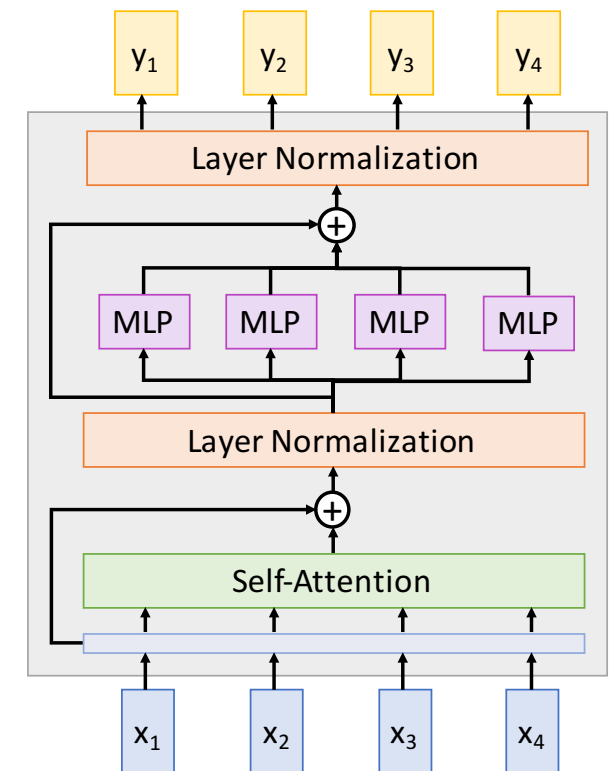


A dog is standing on a hardwood floor.

Generalized **Self-Attention** is new, powerful neural network primitive



Transformers are a new neural network model that only uses attention



Next Time: Midterm!