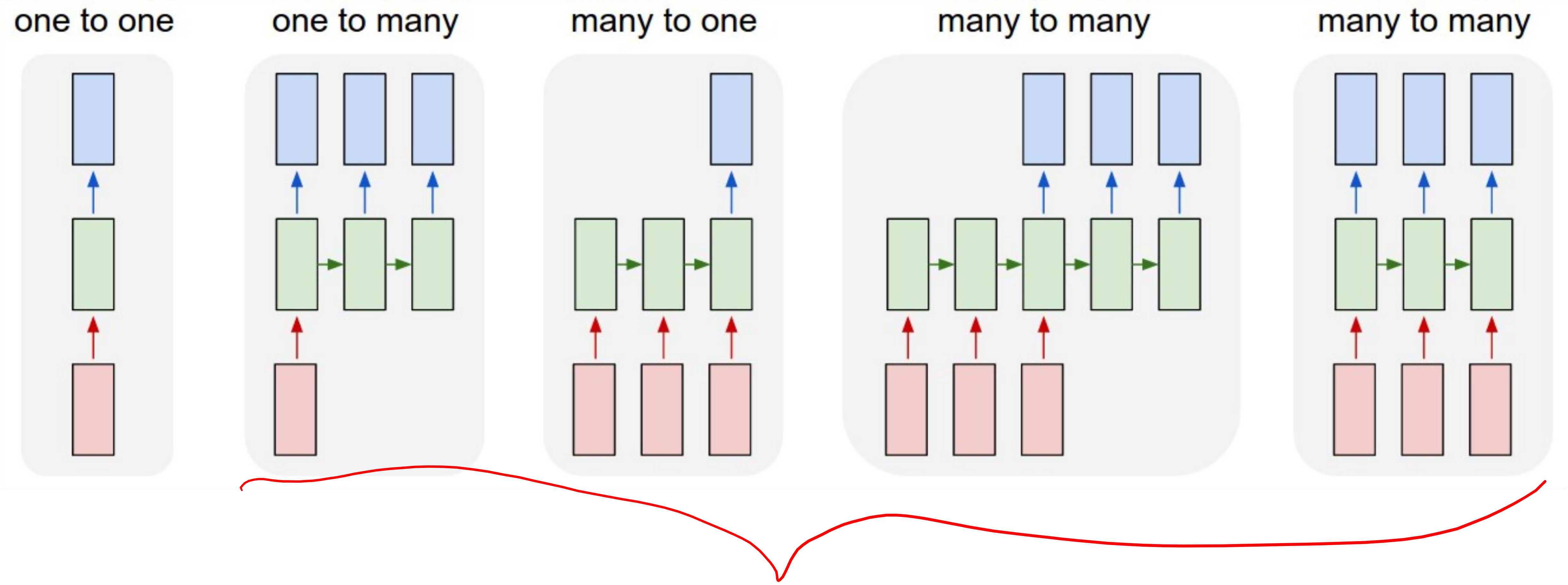
Lecture 7: RNN-II

Last Time: Recurrent Neural Networks



Sequence-to-Sequence with RNNs

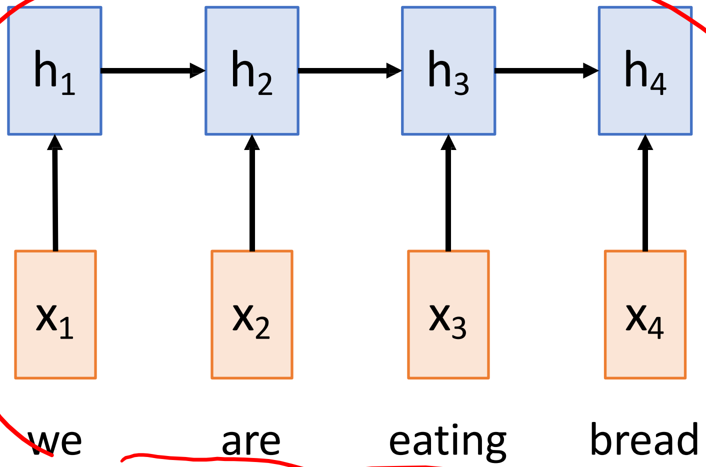
Input: Sequence $x_1, \ldots x_T$

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

T

T'

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



h_0

Sutskever et al, "Sequence to sequence learning with neural networks", NeurIPS 2014

Sequence-to-Sequence with RNNs

Input: Sequence $x_1, \ldots x_T$

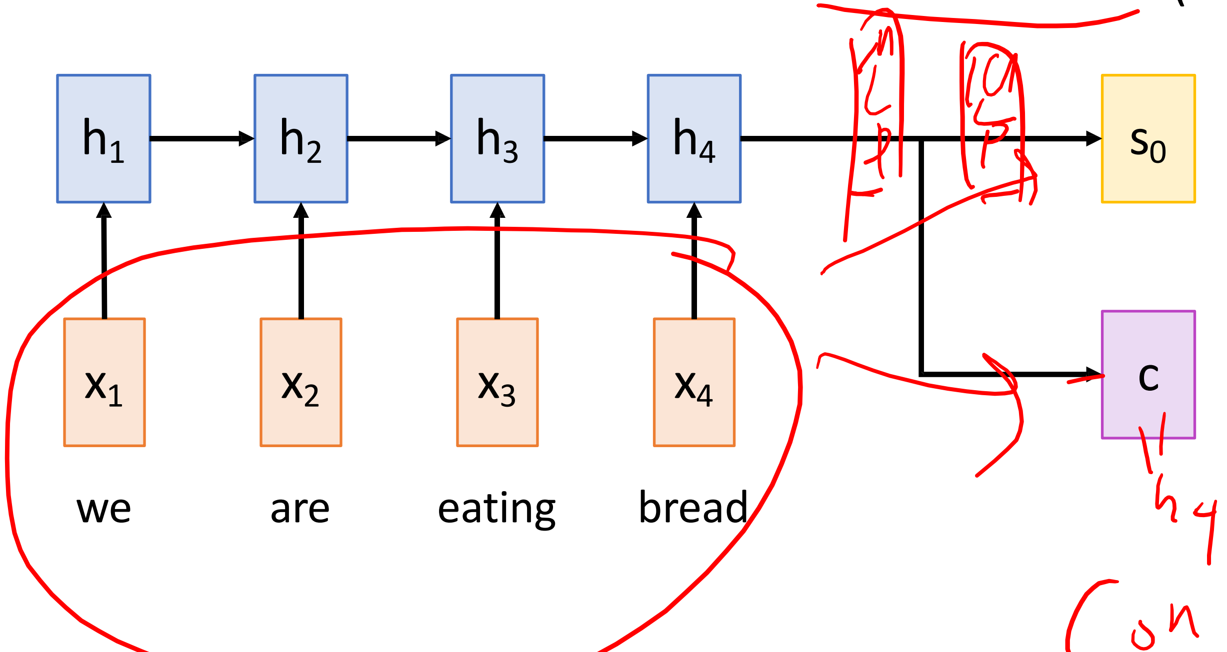
Output: Sequence $y_1, \ldots, y_{T'}$

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

From final hidden state predict:

Initial decoder state s_0

Context vector c (often $c=h_T$)



Sutskever et al, "Sequence to sequence learning with neural networks", NeurIPS 2014

Sequence-to-Sequence with RNNs

Input: Sequence $x_1, \ldots x_T$

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

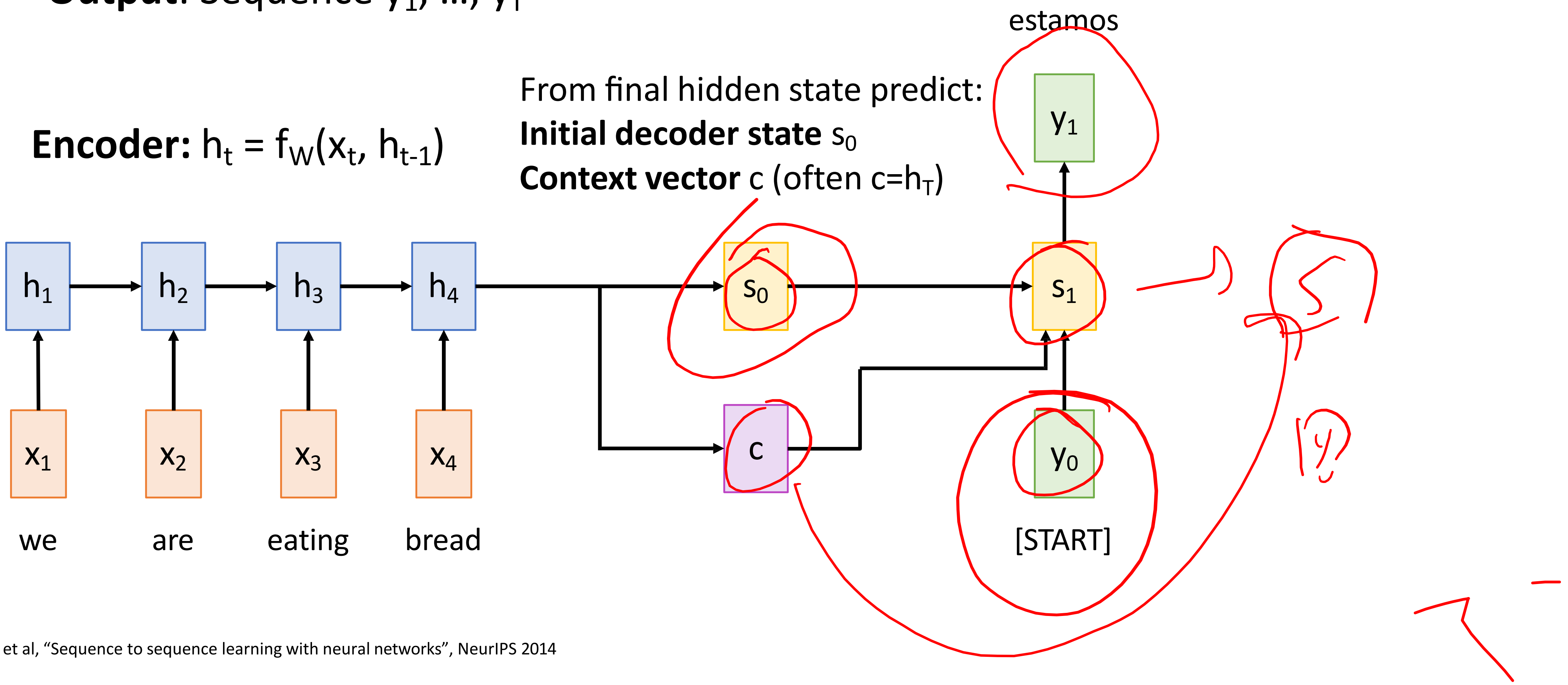
Decoder: $s_t = g_U(y_{t-1}, h_{t-1}, c)$

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

From final hidden state predict:

Initial decoder state s_0

Context vector c (often $c=h_T$)



Sutskever et al, "Sequence to sequence learning with neural networks", NeurIPS 2014

Sequence-to-Sequence with RNNs

Input: Sequence $x_1, \ldots x_T$

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

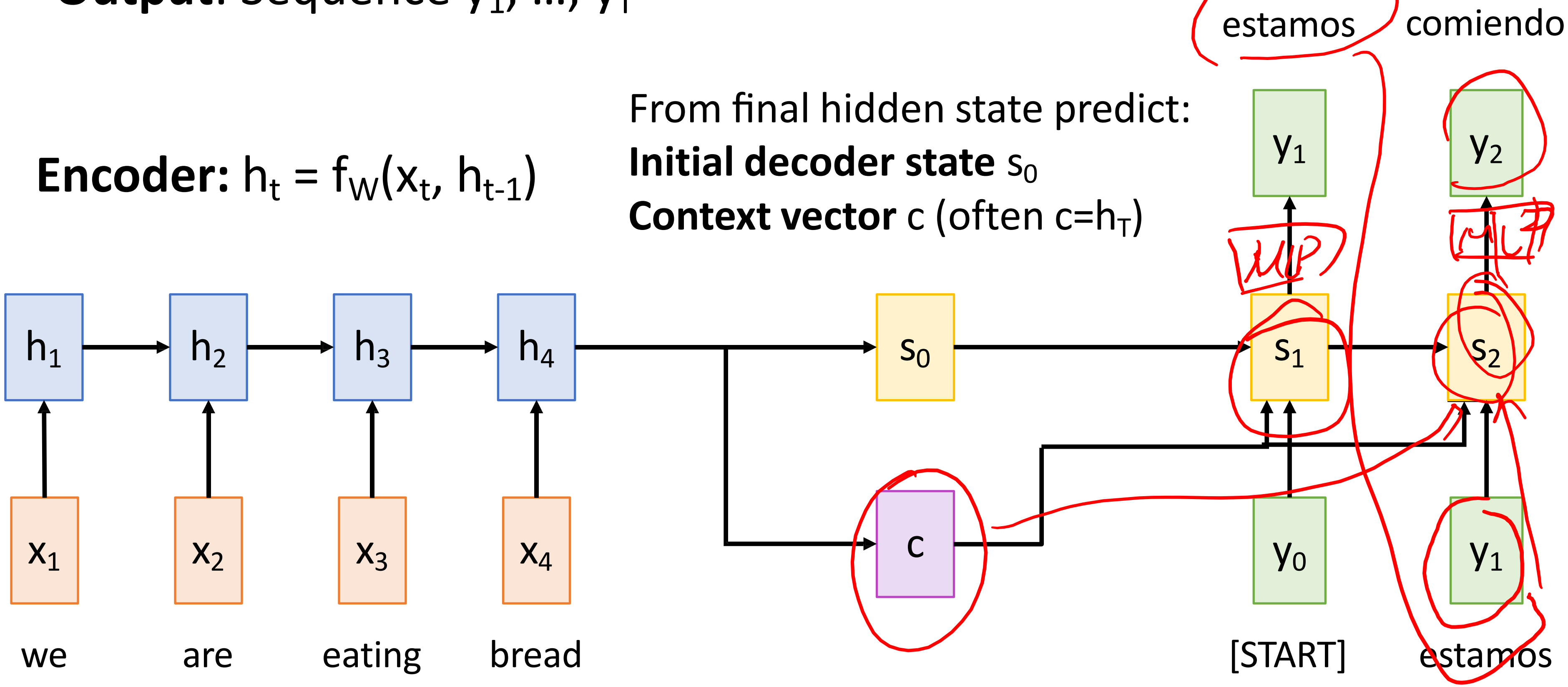
Decoder: $s_t = g_U(y_{t-1}, h_{t-1}, c)$

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

From final hidden state predict:

Initial decoder state s_0

Context vector c (often $c=h_T$)



Sutskever et al, “Sequence to sequence learning with neural networks”, NeurIPS 2014

Sequence-to-Sequence with RNNs

Input: Sequence $x_1, \ldots x_T$

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

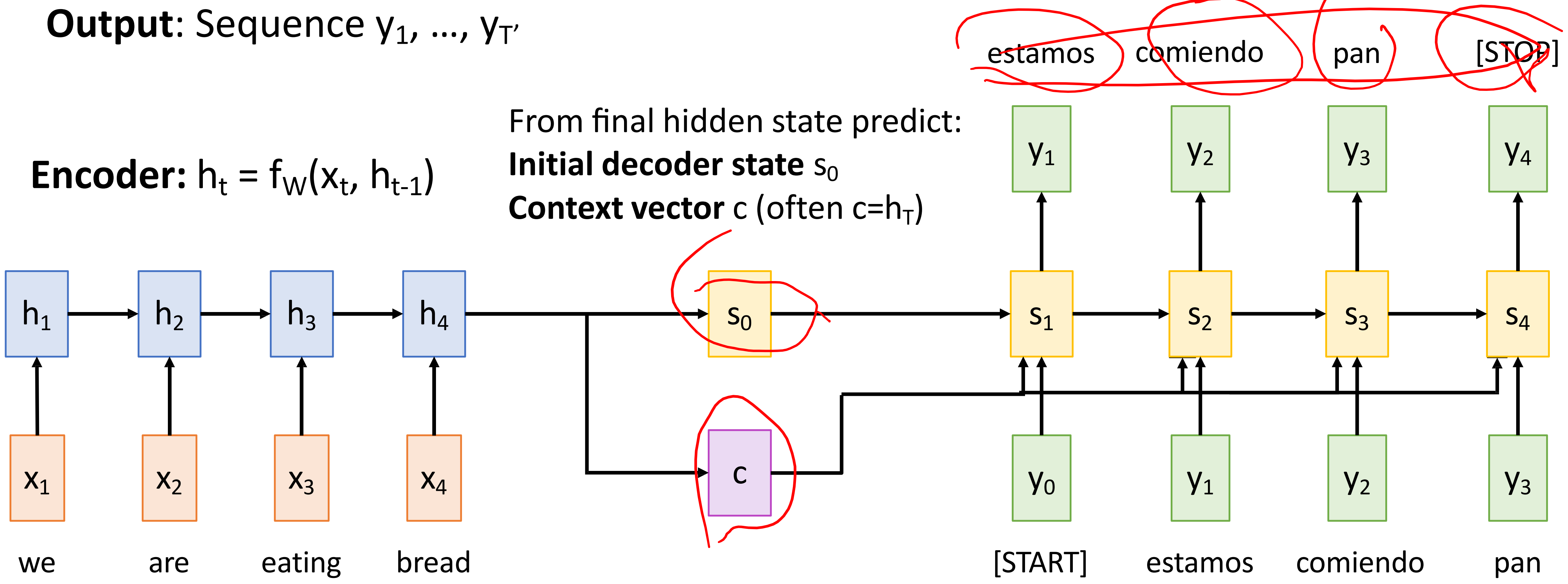
Decoder: $s_t = g_U(y_{t-1}, h_{t-1}, c)$

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

From final hidden state predict:

Initial decoder state s_0

Context vector c (often $c=h_T$)



Sutskever et al, "Sequence to sequence learning with neural networks", NeurIPS 2014

Sequence-to-Sequence with RNNs

Input: Sequence $x_1, \ldots x_T$

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

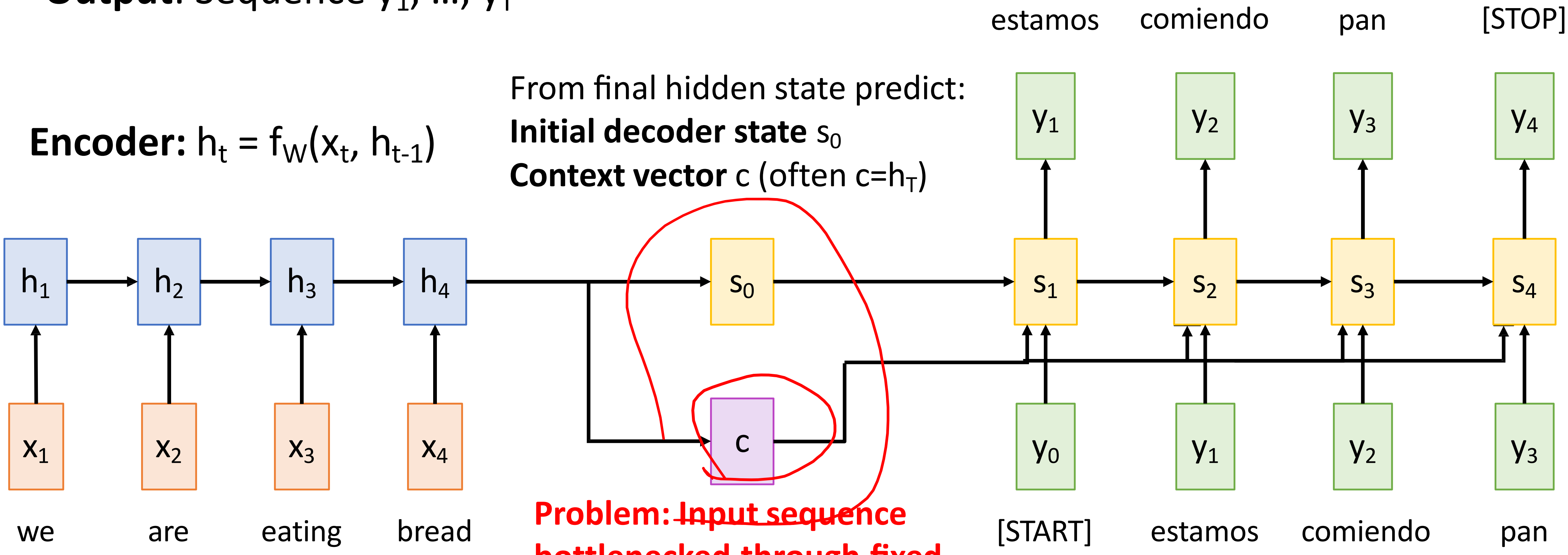
Decoder: $s_t = g_U(y_{t-1}, h_{t-1}, c)$

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?

Sutskever et al, “Sequence to sequence learning with neural networks”, NeurIPS 2014

Sequence-to-Sequence with RNNs

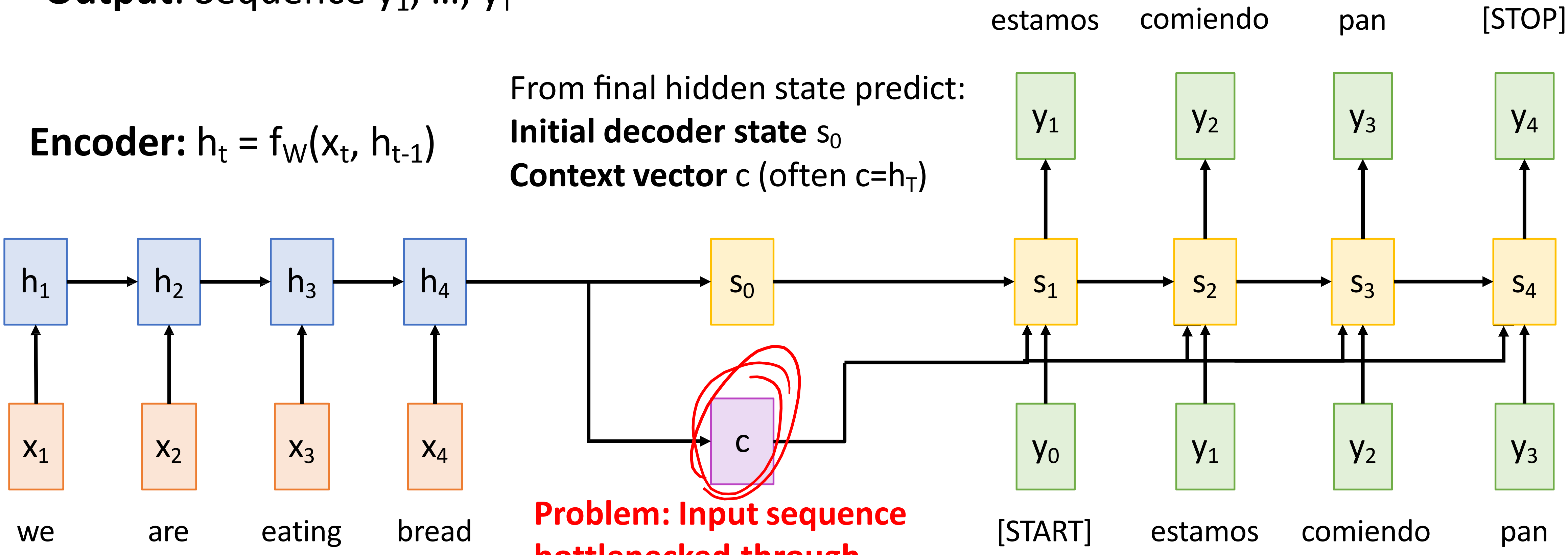
Input: Sequence $x_1, \ldots x_T$

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

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

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-vector. What if T=1000?

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

Sutskever et al, "Sequence to sequence learning with neural networks", NeurIPS 2014

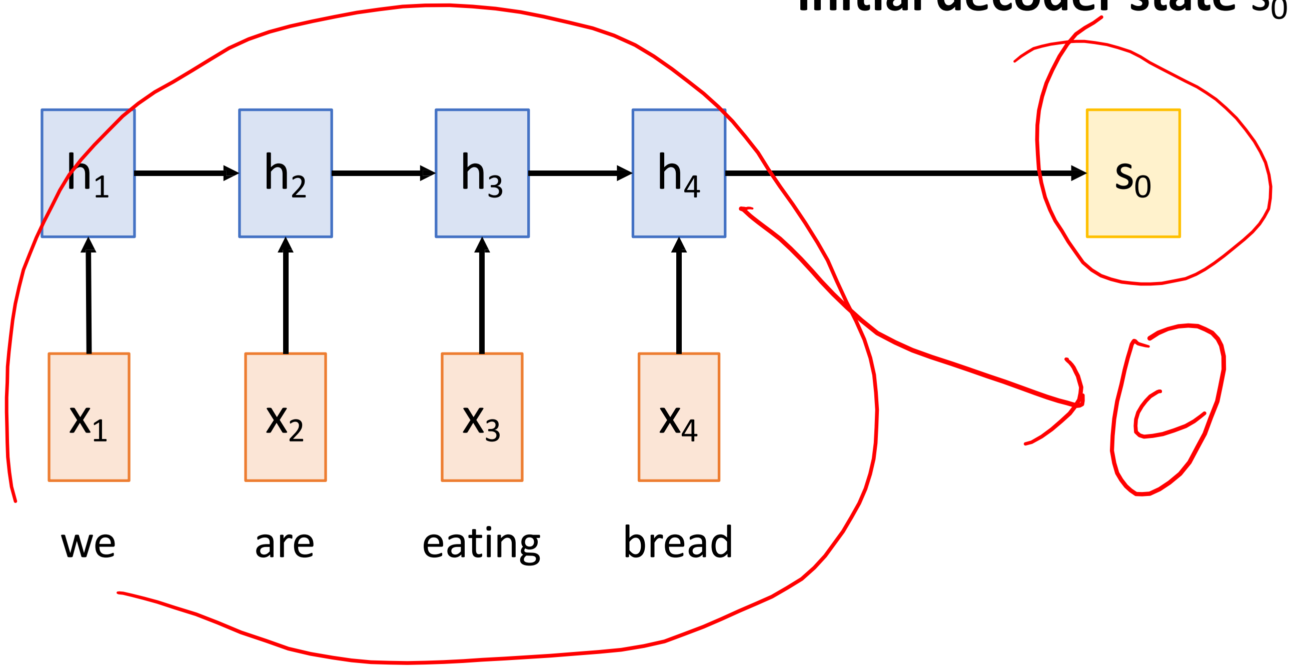
Sequence-to-Sequence with RNNs and attention

Input: Sequence $x_1, \ldots x_T$

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

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

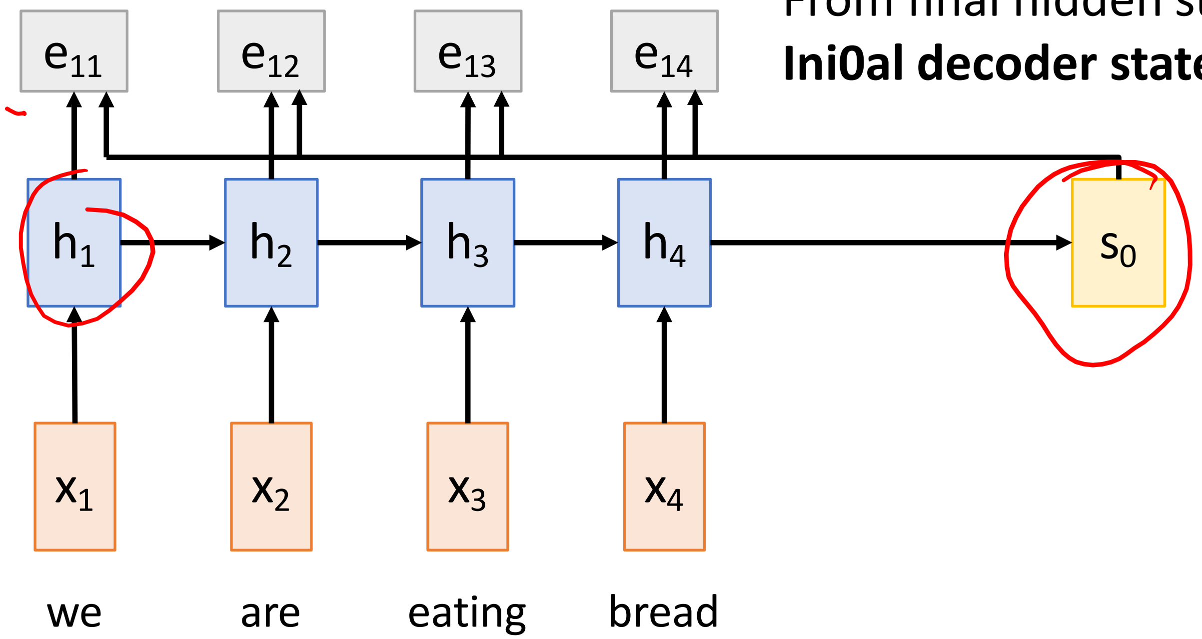
From final hidden state:
Initial decoder state s_0



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

Sequence-to-Sequence with RNNs and attention

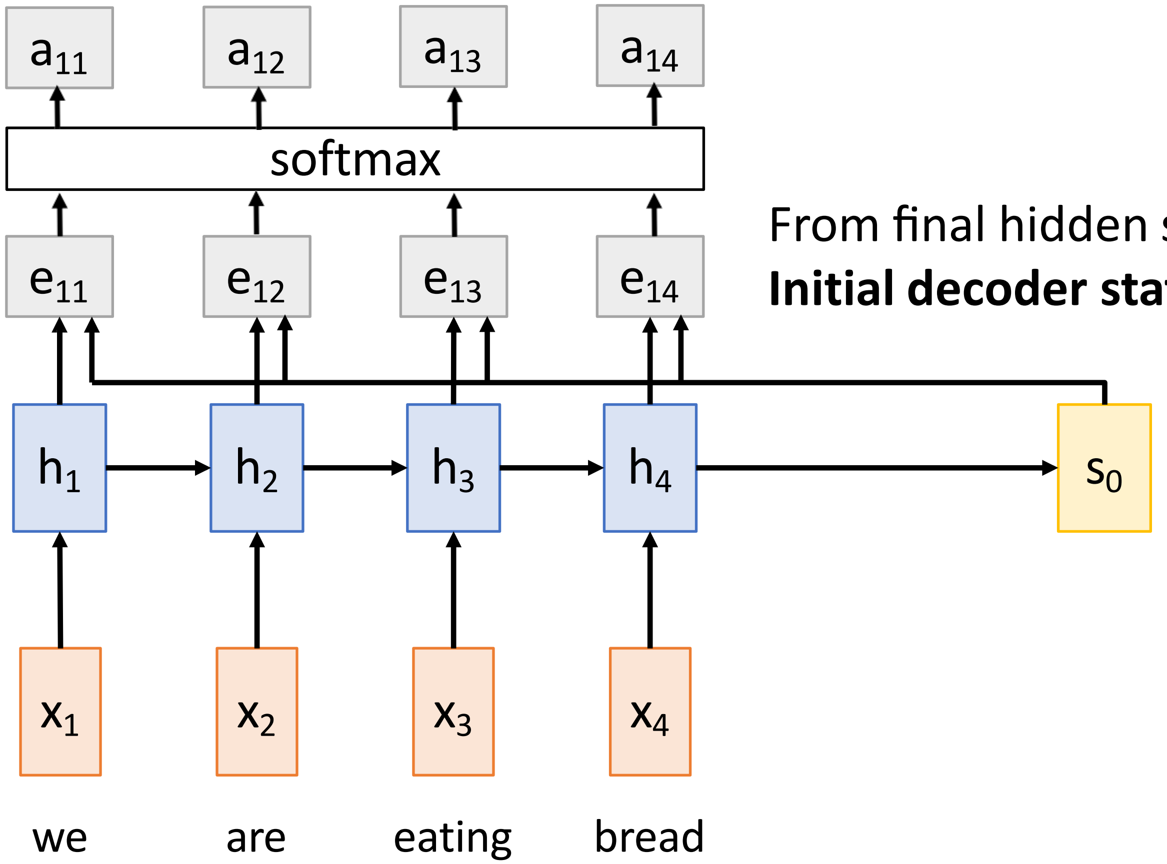
Compute (scalar) **alignment scores**
$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)



From final hidden state:
Ini0al decoder state s_0

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

Sequence-to-Sequence with RNNs and Attention



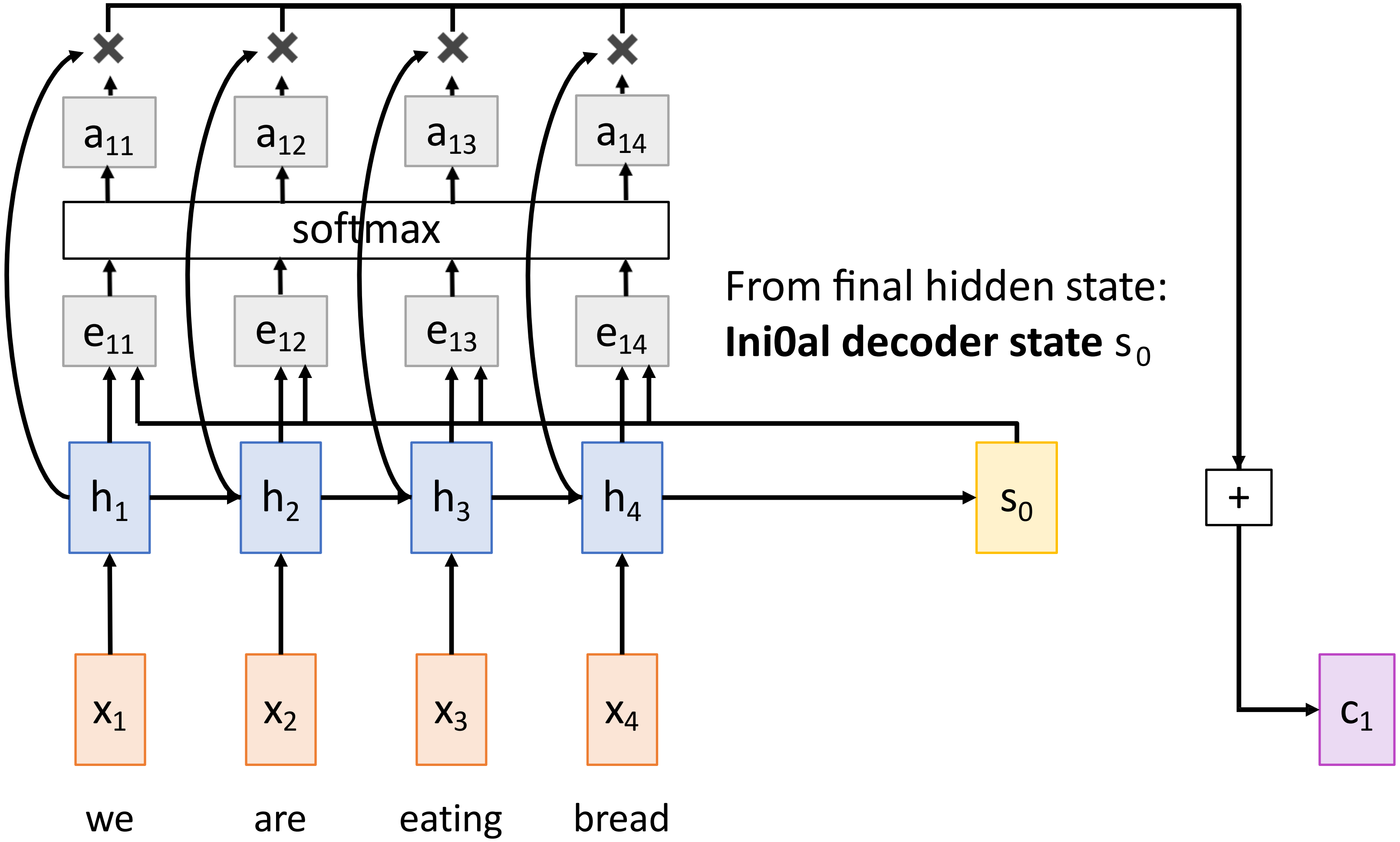
From final hidden state:
Initial decoder state s_0

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

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

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

Sequence-to-Sequence with RNNs and Attention



From final hidden state:
Ini0al decoder state s_0

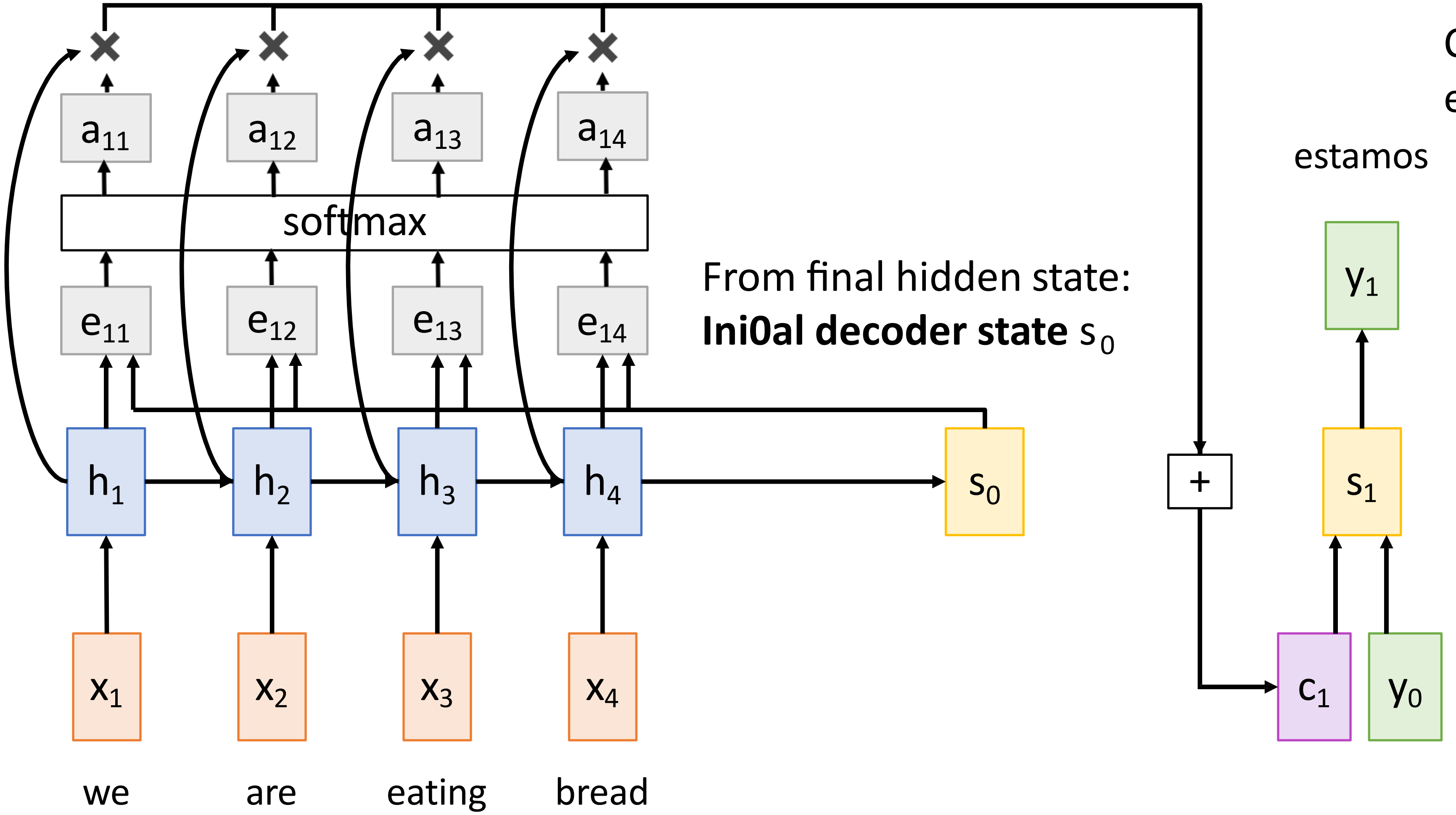
Compute (scalar) **alignment scores**
$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)

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

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

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

Sequence-to-Sequence with RNNs and Attention



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

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

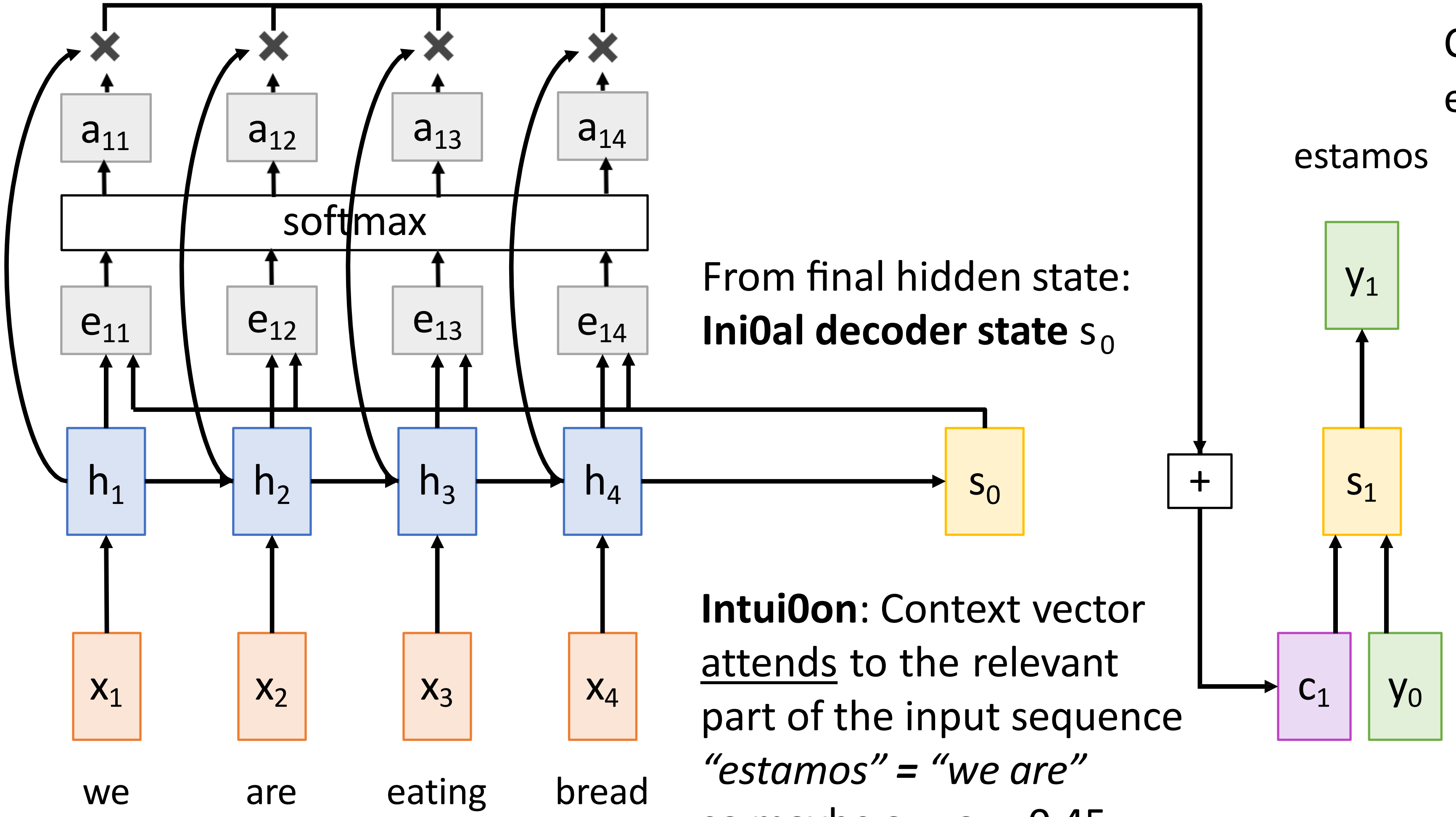
Compute context vector as linear combina$on of hidden states
$c_t = \sum_i a_{t,i} h_i$

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

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

Sequence-to-Sequence with RNNs and Attention



From final hidden state: **Ini0al decoder state** s_0

Intui0on: Context vector attends to the relevant part of the input sequence
"estamos" = "we are"
so maybe $a_{11}=a_{12}=0.45$, $a_{13}=a_{14}=0.05$

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

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

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

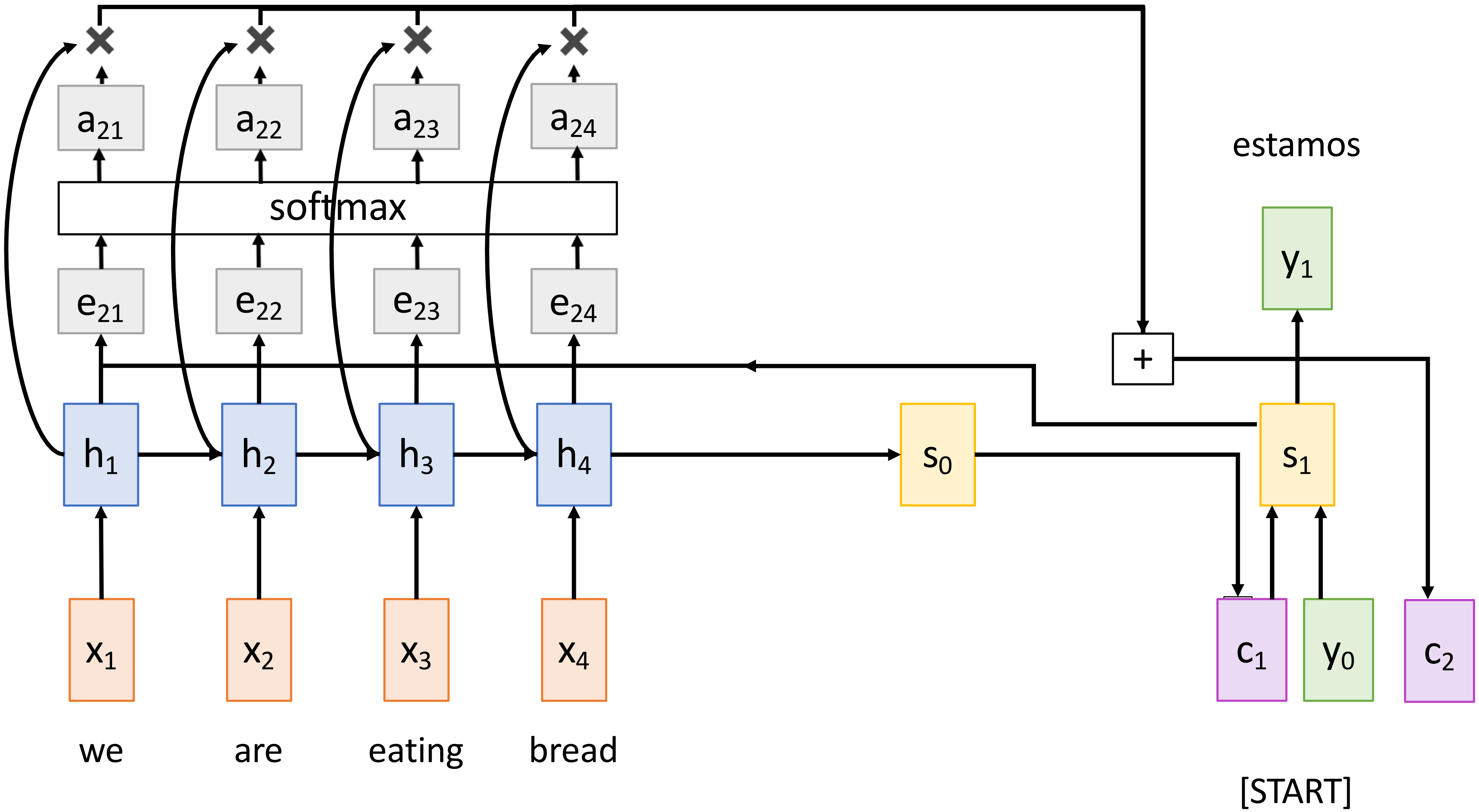
Use context vector in decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c_t)$

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

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

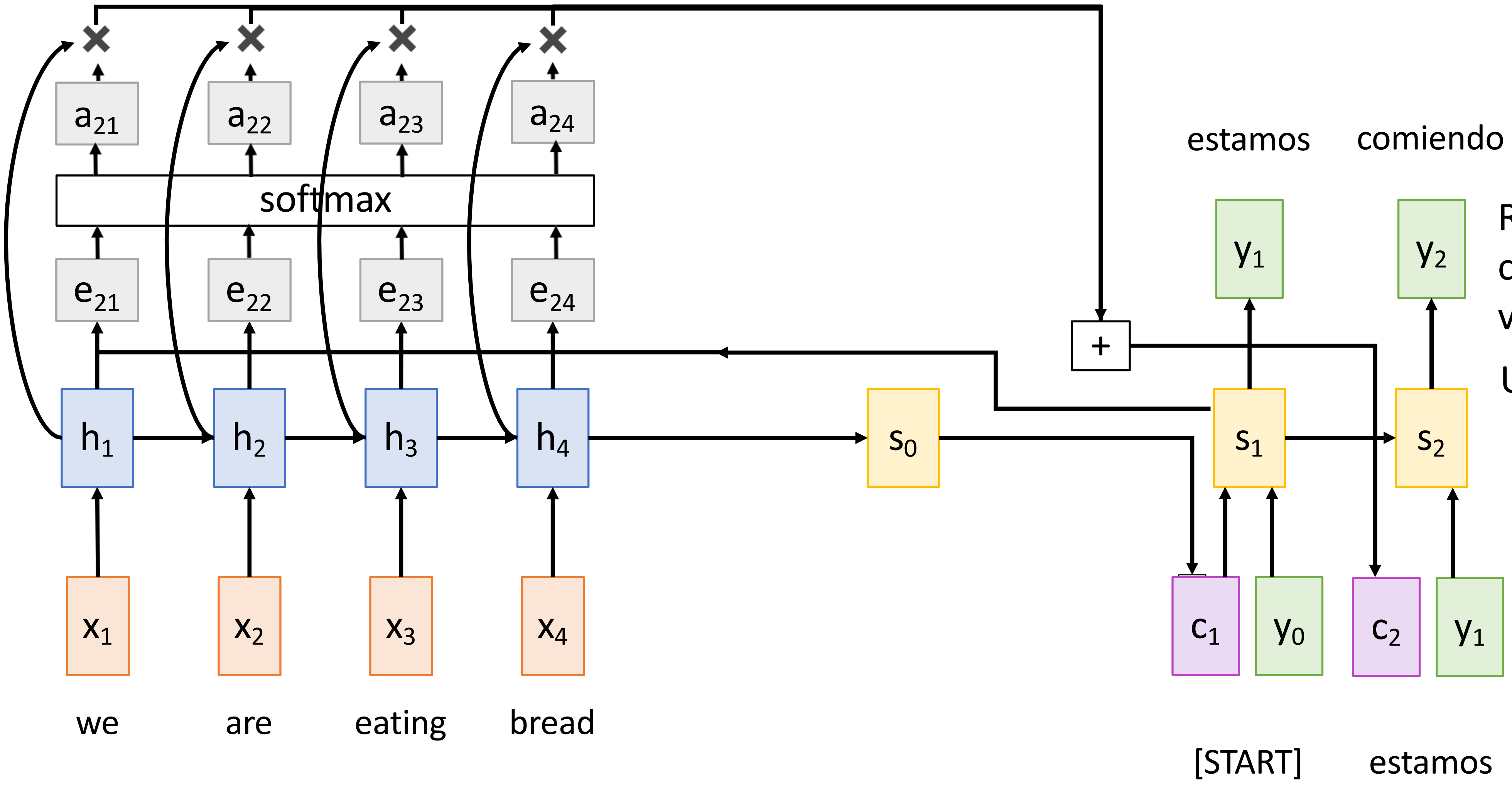
Sequence-to-Sequence with RNNs

Repeat: Use s_1 to compute new context vector c_2



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

Sequence-to-Sequence with RNNs and Attention

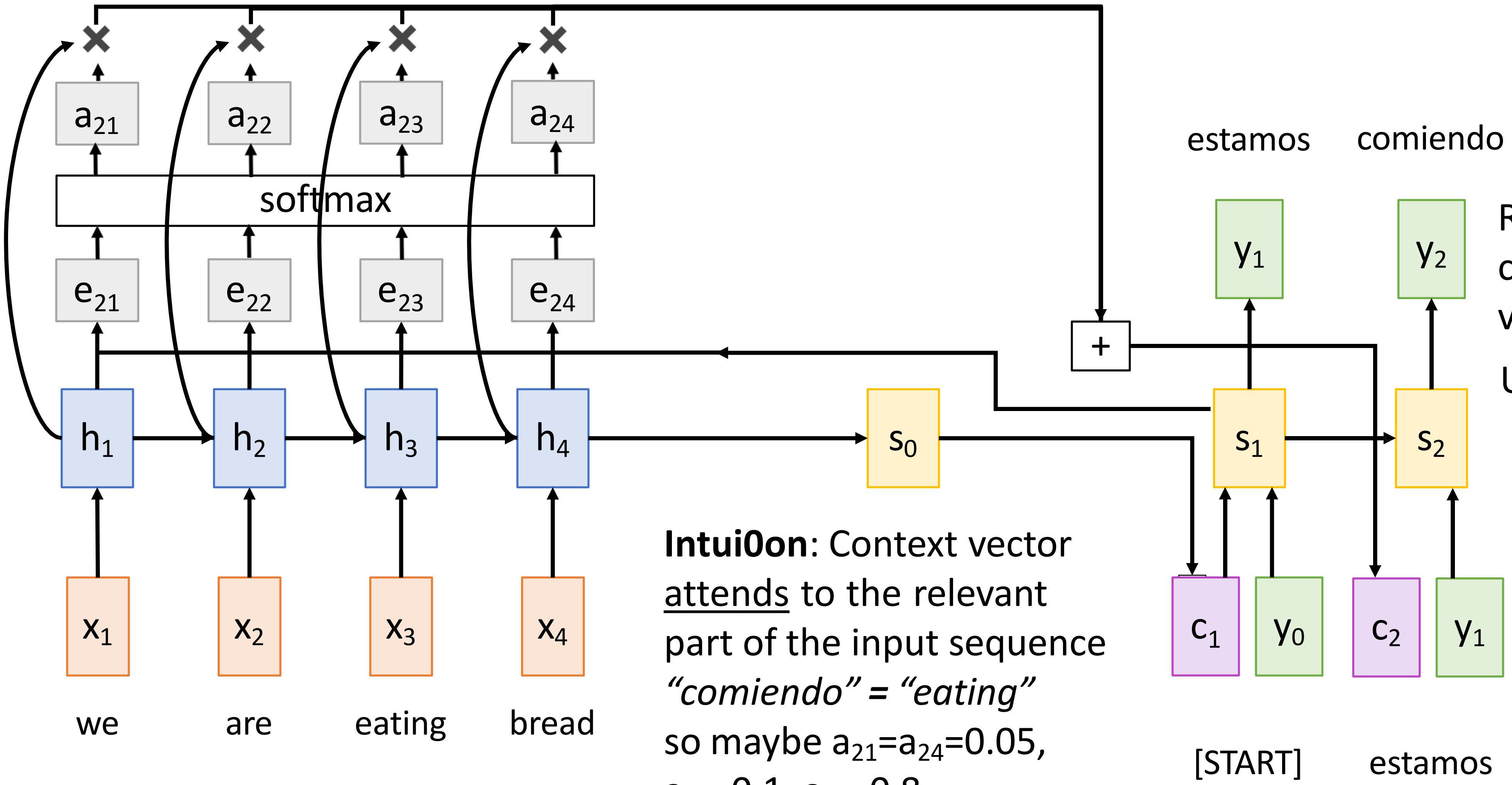


Repeat: Use s_1 to compute new context vector c_2

Use c_2 to compute s_2, y_2

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

Sequence-to-Sequence with RNNs and Attention



Repeat: Use s_1 to compute new context vector c_2

Use c_2 to compute s_2, y_2

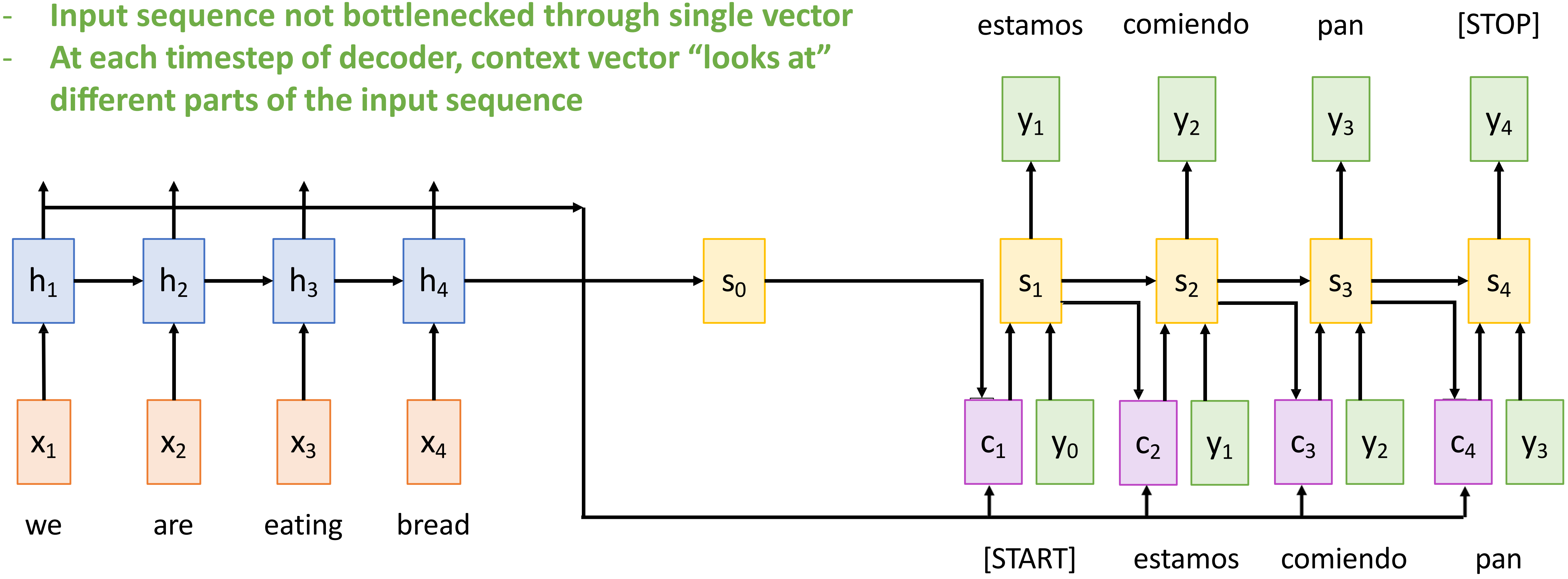
Intui0on: Context vector attends to the relevant part of the input sequence *"comiendo" = "eating"* so maybe $a_{21}=a_{24}=0.05$, $a_{22}=0.1$, $a_{23}=0.8$

Bahdanau et al, "Neural machine transla$on 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**



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

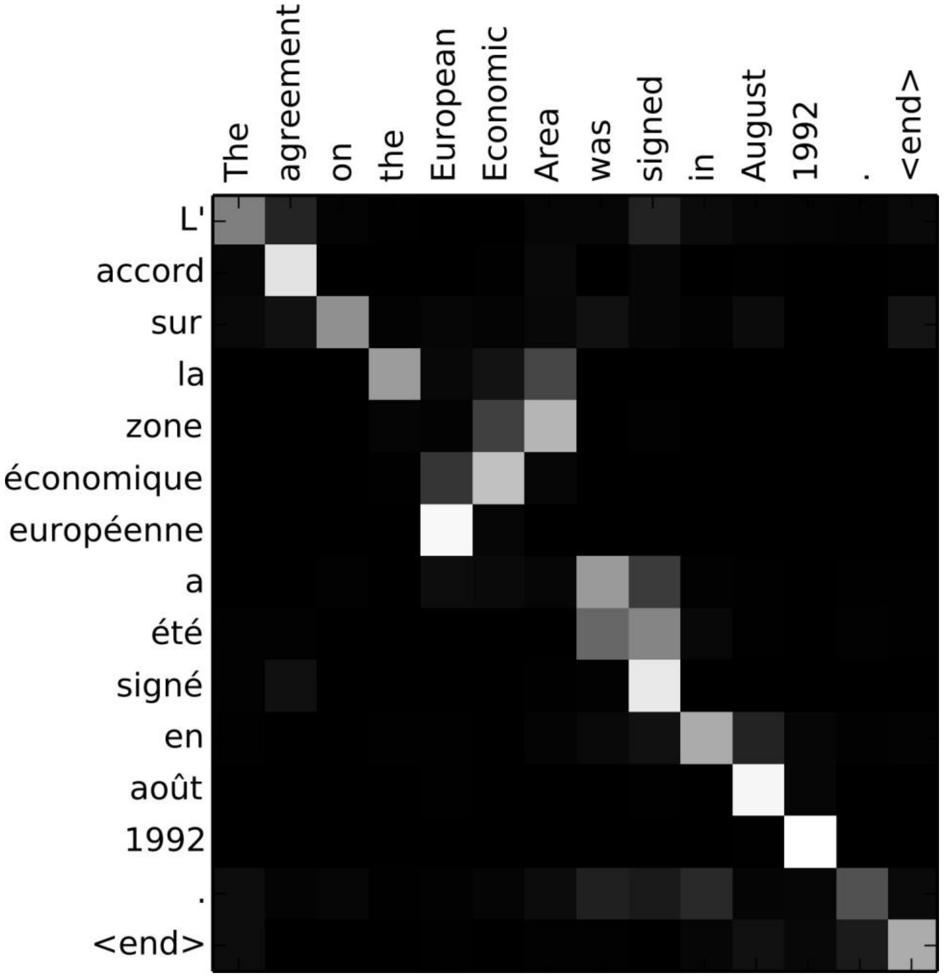
Sequence-to-Sequence with RNNs and Attention

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.”

Visualize attition weights $a_{t,i}$



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

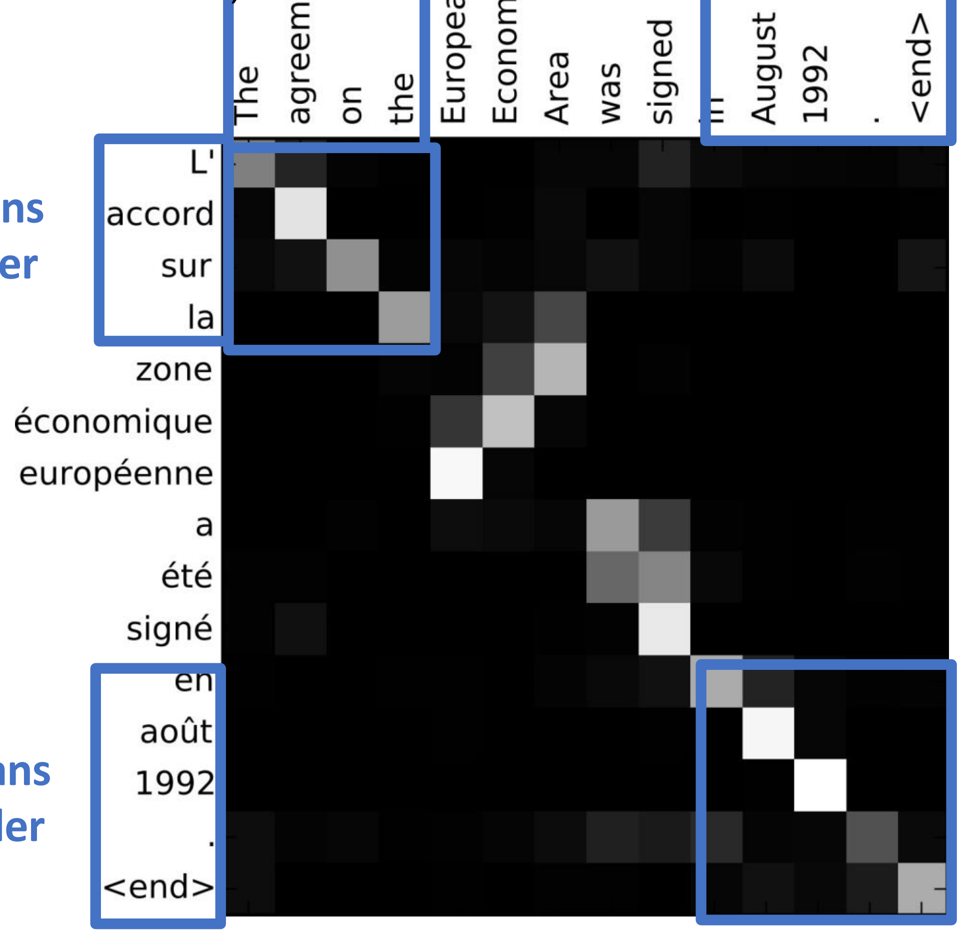
Sequence-to-Sequence with RNNs and A,en.on

Example: English to French transla$on

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

Diagonal Attention means words correspond in order

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

Sequence-to-Sequence with RNNs and A,en.on

Example: English to French transla$on

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

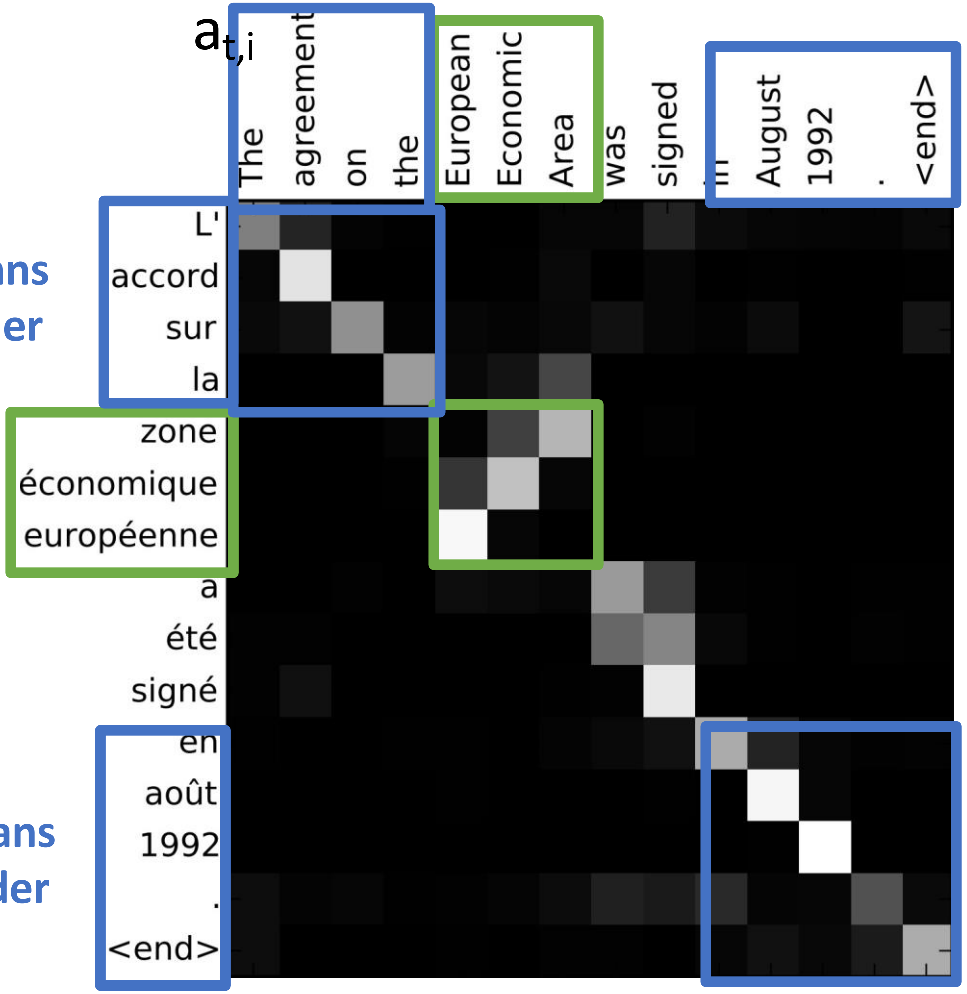
Output: “L’accord sur la zone économique européenne a été signé en août 1992.”

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

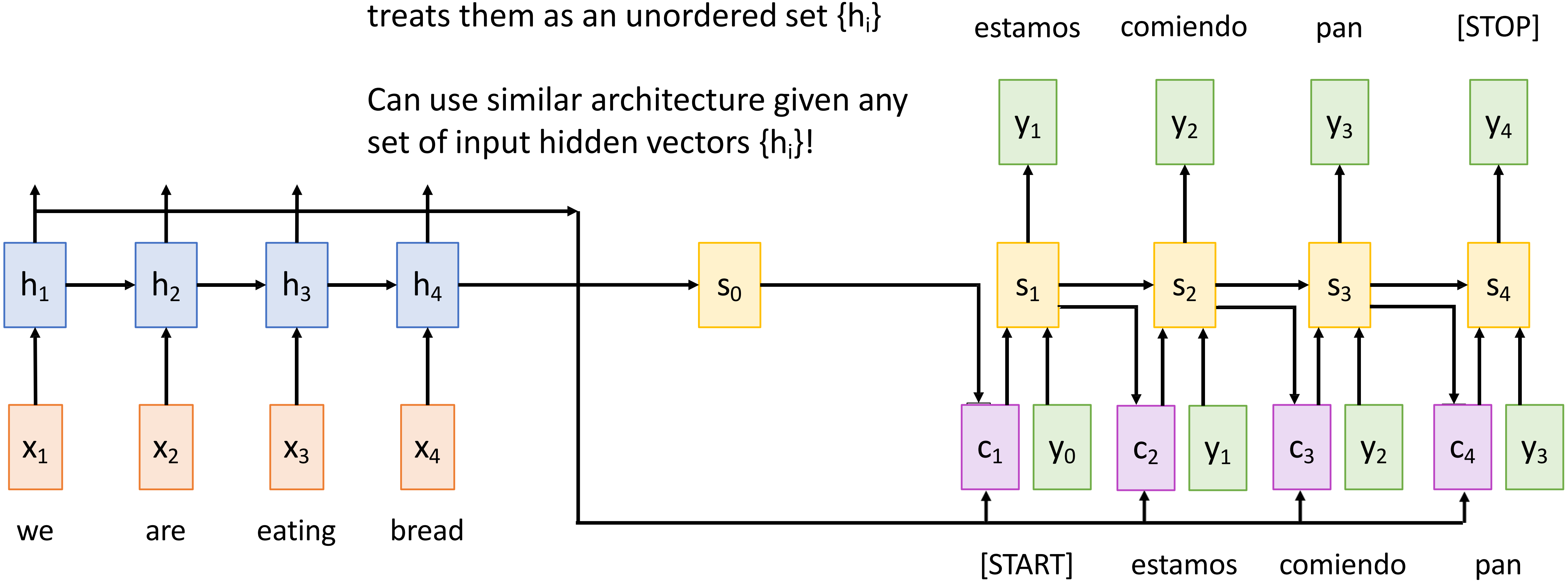


Bahdanau et al, “Neural machine transla$on by jointly learning to align and translate”, ICLR 2015

Sequence-to-Sequence with RNNs and Attention

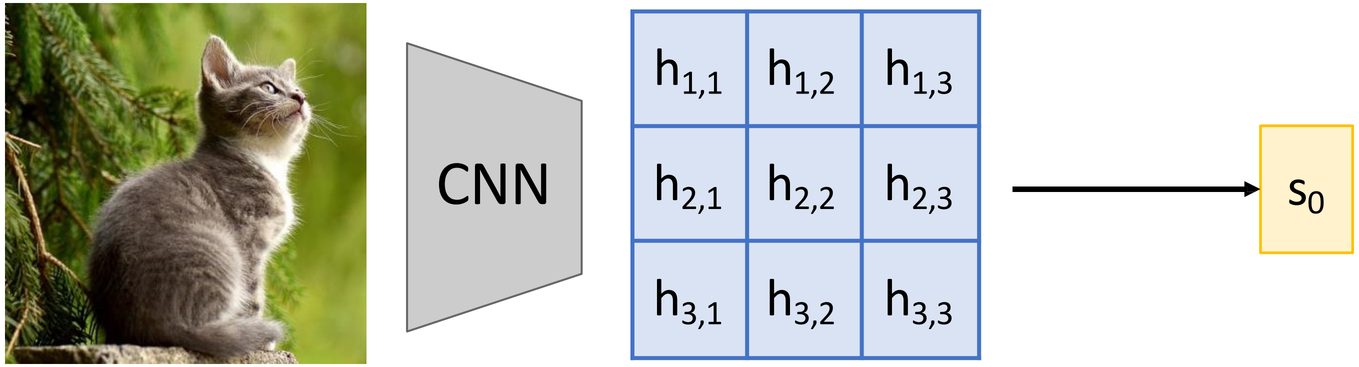
The decoder doesn't use the fact that h_i form an ordered sequence – it just treats them as an unordered set $\{h_i\}$

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



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

Image Cap.oning with RNNs and Attention

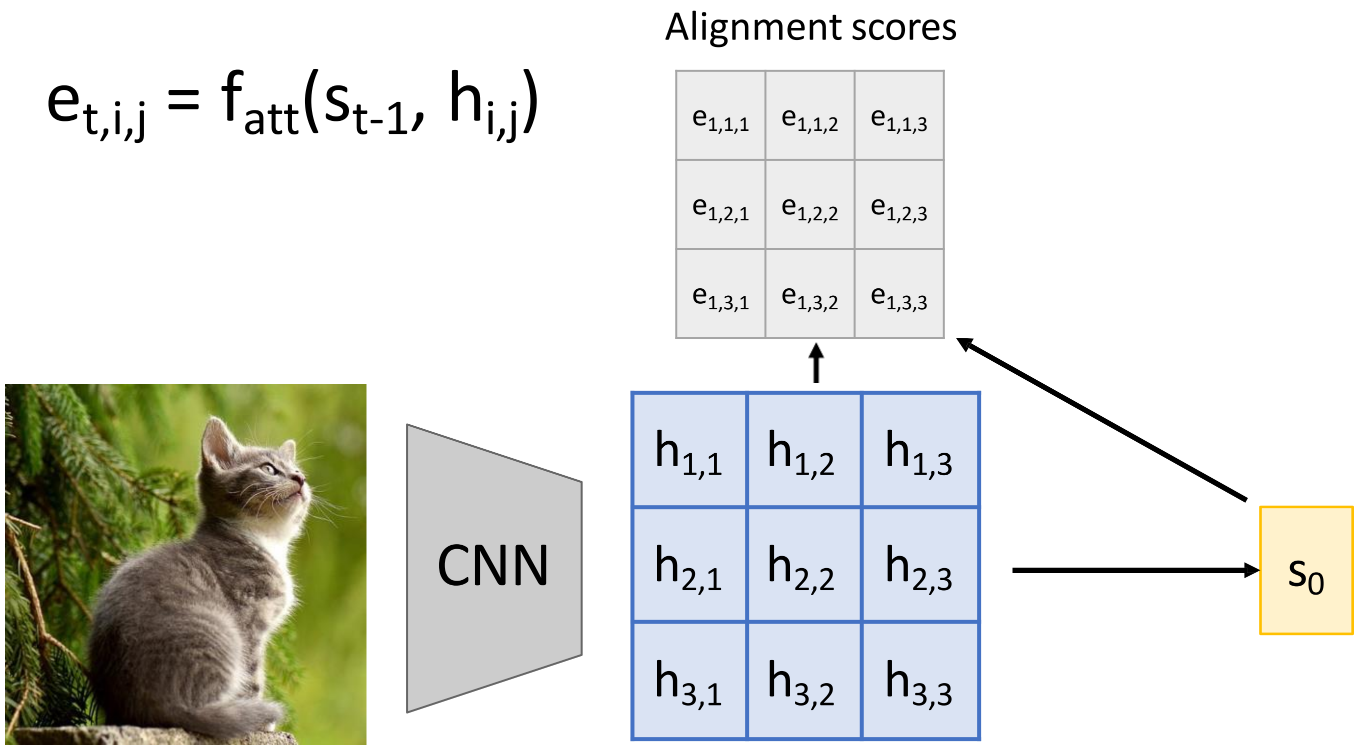


Use a CNN to compute a
grid of features for an image

Xu et al, “Show, attend, and Tell: Neural Image Cap$on Genera$on with Visual Attention”, ICML 2015

Image Captioning with RNNs and Attention

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



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

Xu et al, “Show, attend, and Tell: Neural Image Cap$on Genera$on with Visual Attention”, ICML 2015

Image Captioning with RNNs and Attention

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

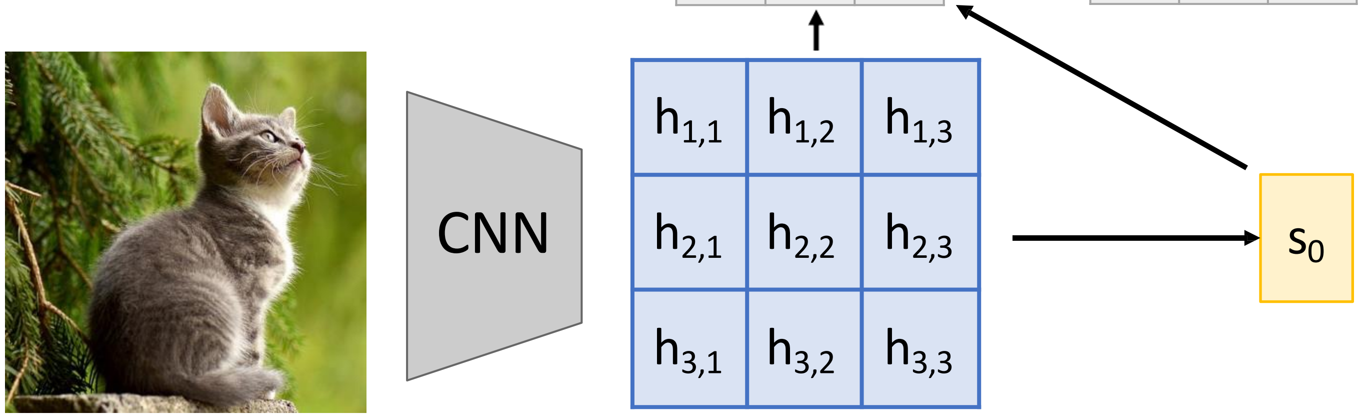
Alignment scores

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

softmax

AEention weights

$a_{1,1,1}$	$a_{1,1,2}$	$a_{1,1,3}$
$a_{1,2,1}$	$a_{1,2,2}$	$a_{1,2,3}$
$a_{1,3,1}$	$a_{1,3,2}$	$a_{1,3,3}$

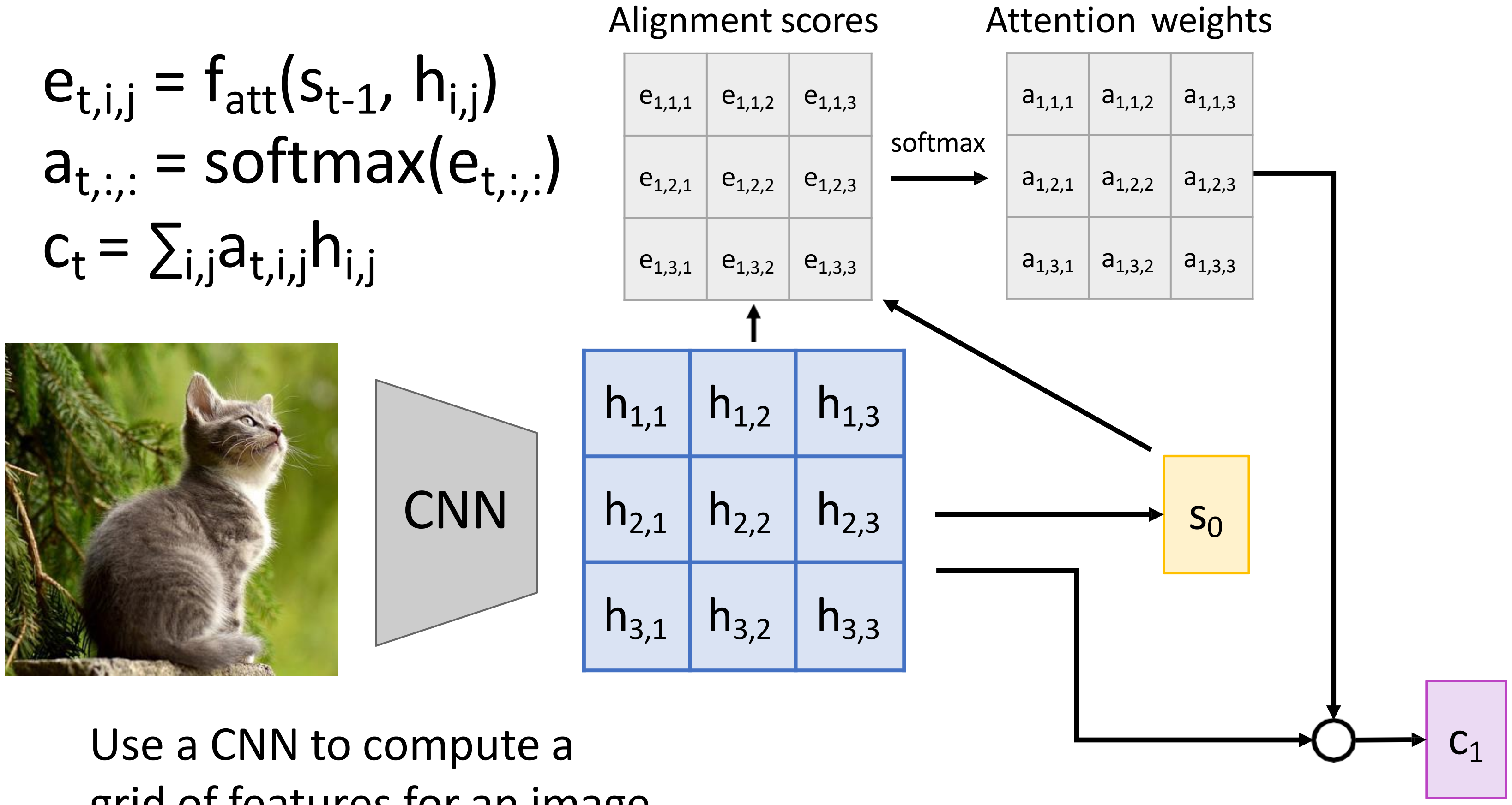


Use a CNN to compute a
grid of features for an image

Xu et al, "Show, attend, and Tell: Neural Image Cap$on Genera$on with Visual Attention", ICML 2015

Image Captioning with RNNs and Attention

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

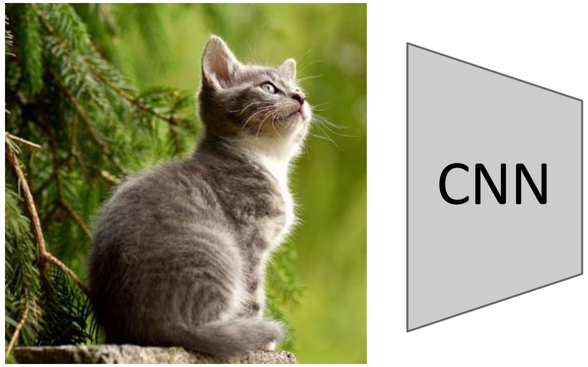


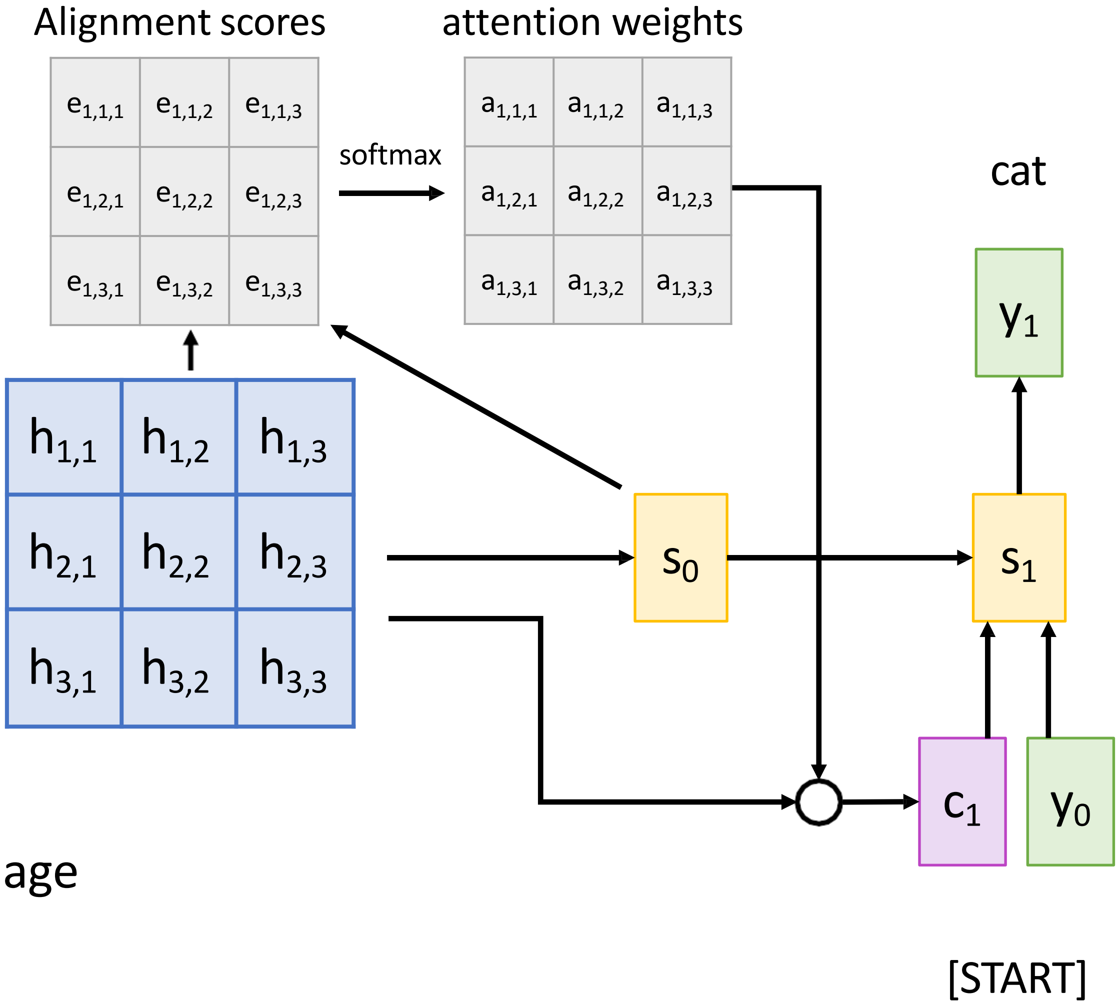
Use a CNN to compute a
grid of features for an image

Xu et al, “Show, attend, and Tell: Neural Image Cap$on Genera$on with Visual Attention”, ICML 2015

Image Captioning with RNNs and Attention

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



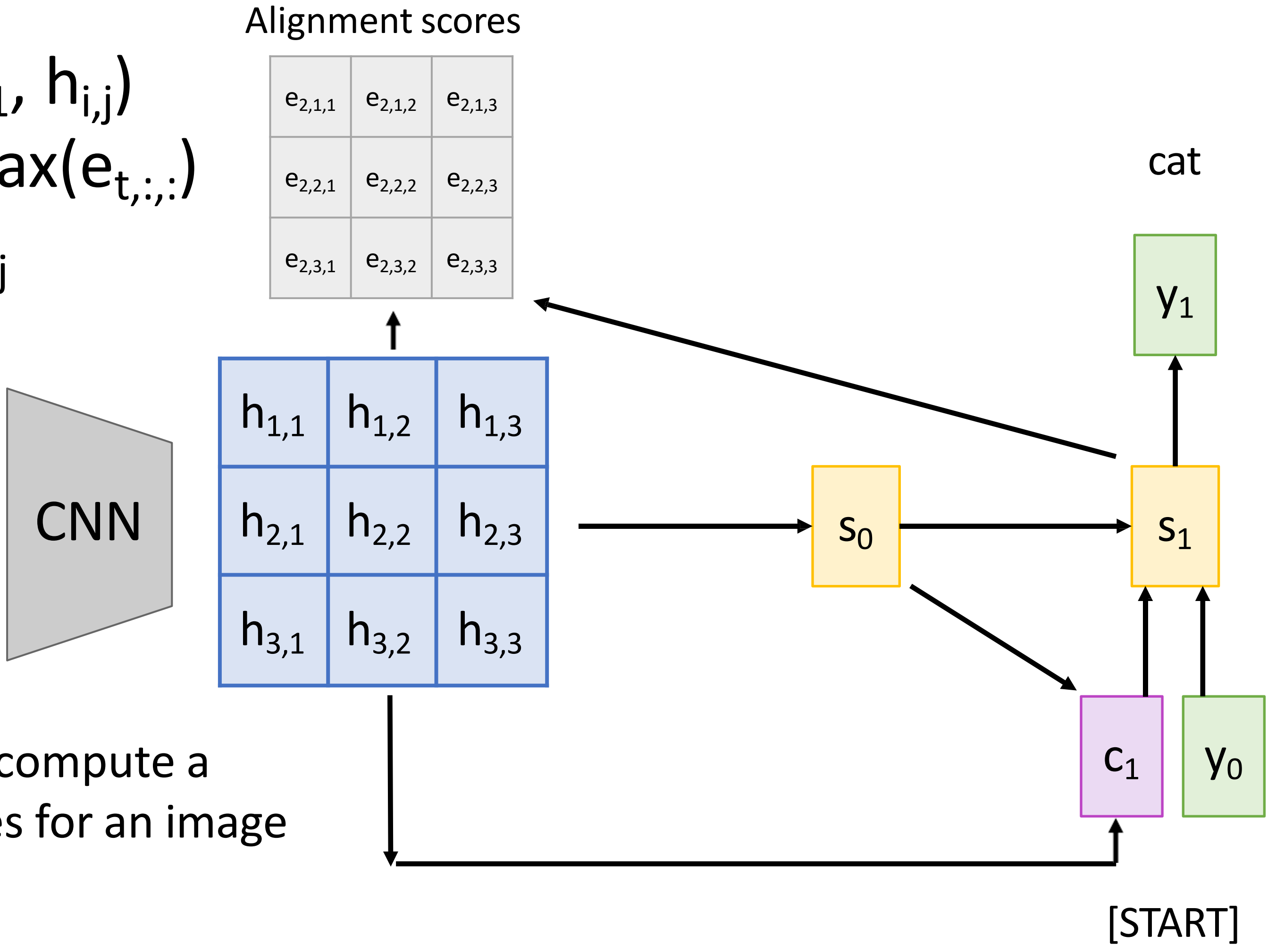


Use a CNN to compute a
grid of features for an image

Xu et al, “Show, attend, and Tell: Neural Image Cap$on Genera$on with Visual Attention”, ICML 2015

Image Captioning with RNNs and Attention

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

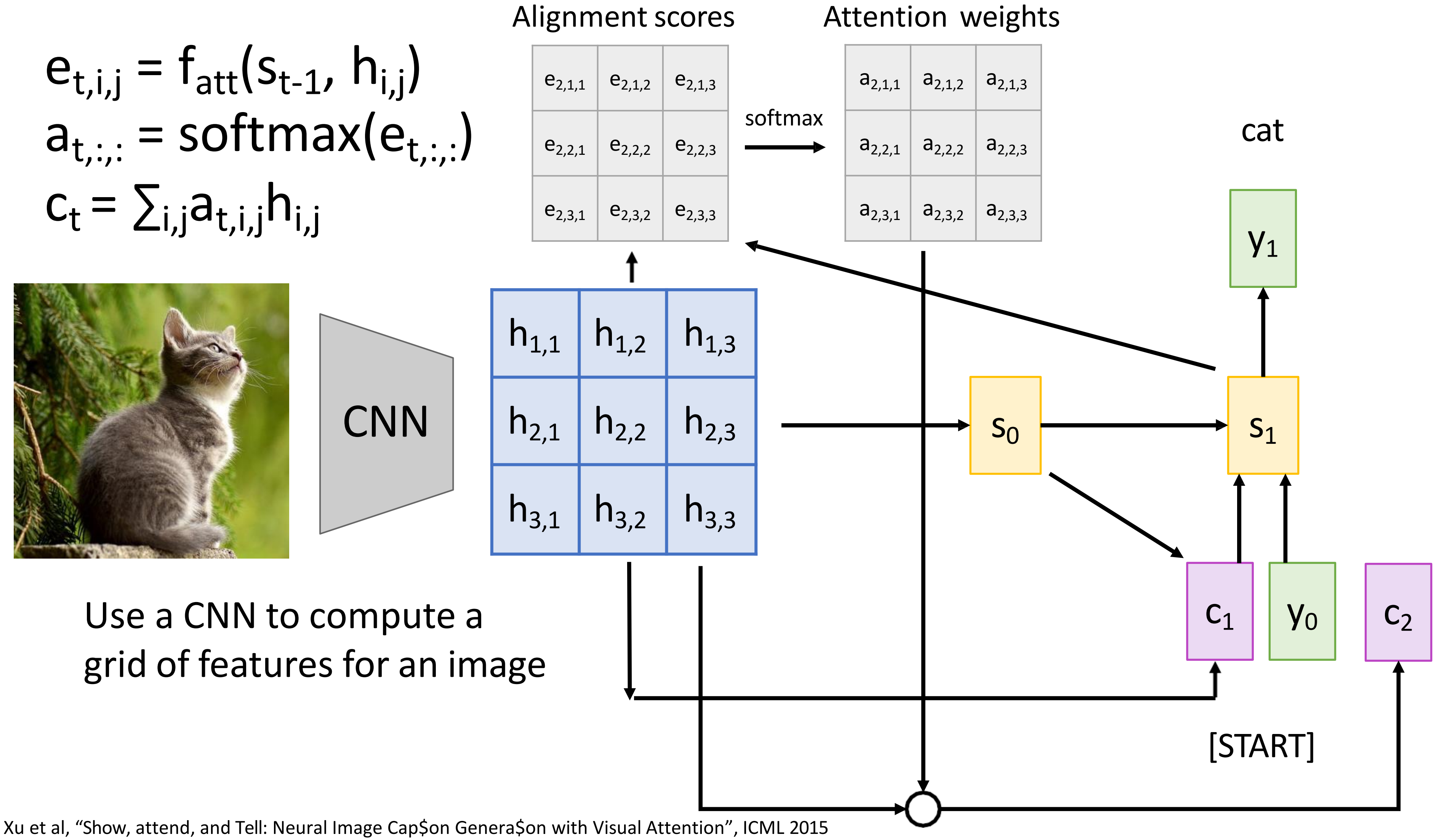


Use a CNN to compute a grid of features for an image

Xu et al, “Show, attend, and Tell: Neural Image Cap$on Genera$on with Visual Attention”, ICML 2015

Image Captioning with RNNs and Attention

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



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

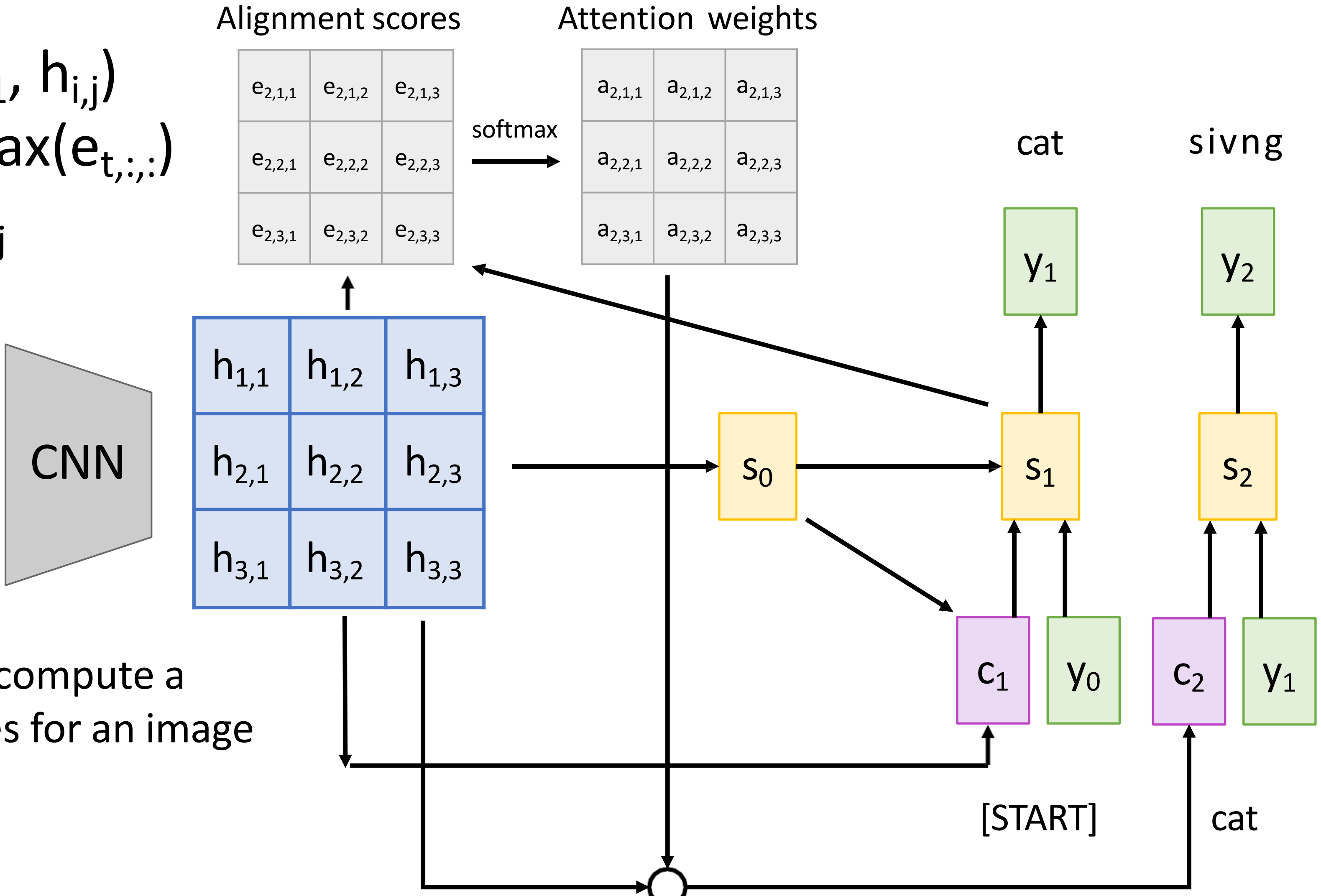
Xu et al, "Show, attend, and Tell: Neural Image Cap$on Genera$on with Visual Attention", ICML 2015

Image Captioning with RNNs and Attention

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

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

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



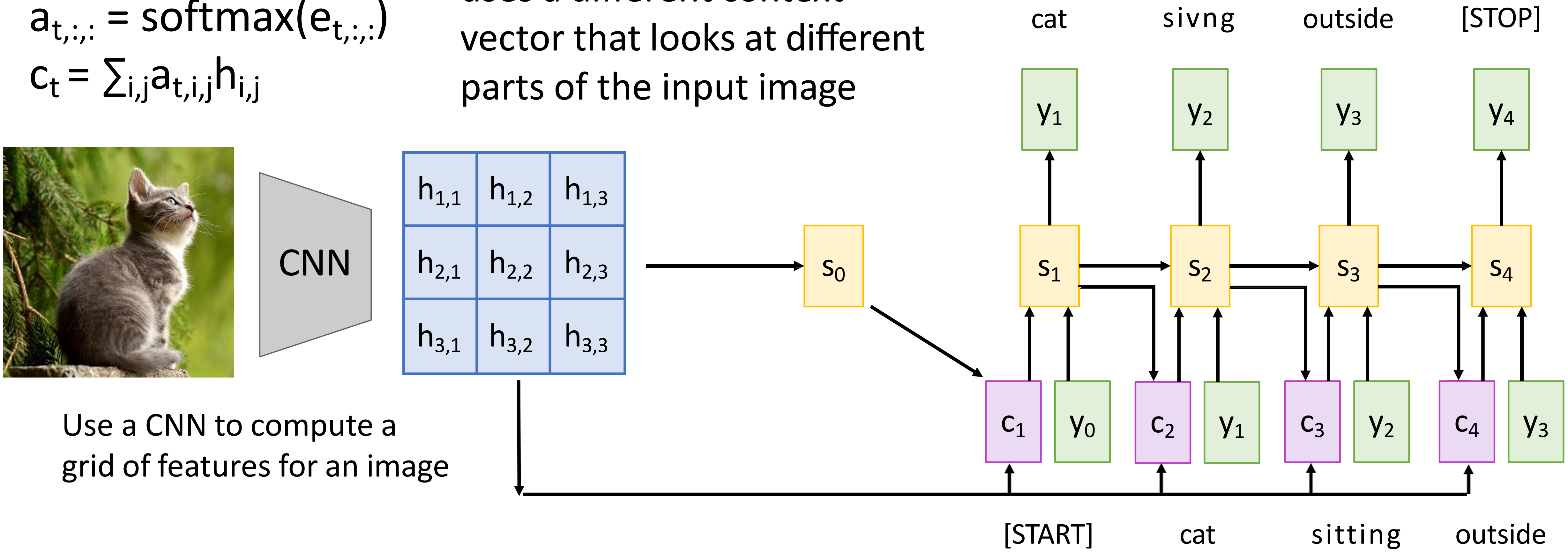
Use a CNN to compute a grid of features for an image

Xu et al, “Show, attend, and Tell: Neural Image Cap$on Genera$on with Visual Attention”, ICML 2015

Image Captioning with RNNs and Attention

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

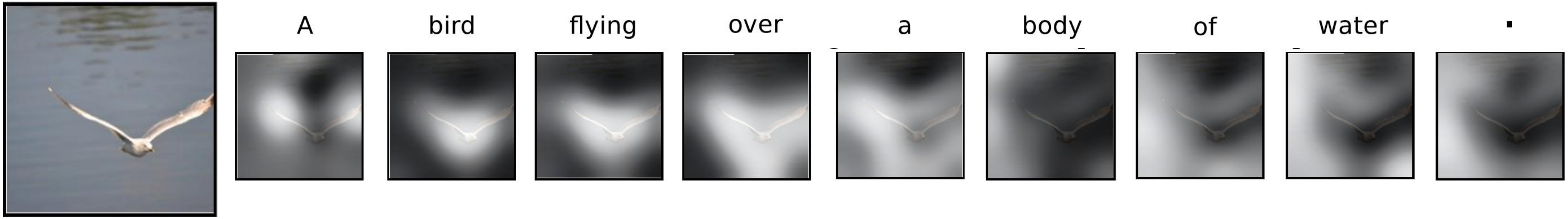
Each timestep of decoder uses a different context vector that looks at different parts of the input image



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

Xu et al, “Show, attend, and Tell: Neural Image Cap$on Genera$on with Visual Attention”, ICML 2015

Image Captioning with RNNs and Attention



Xu et al, "Show, attend, and Tell: Neural Image Cap$on Genera$on 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.

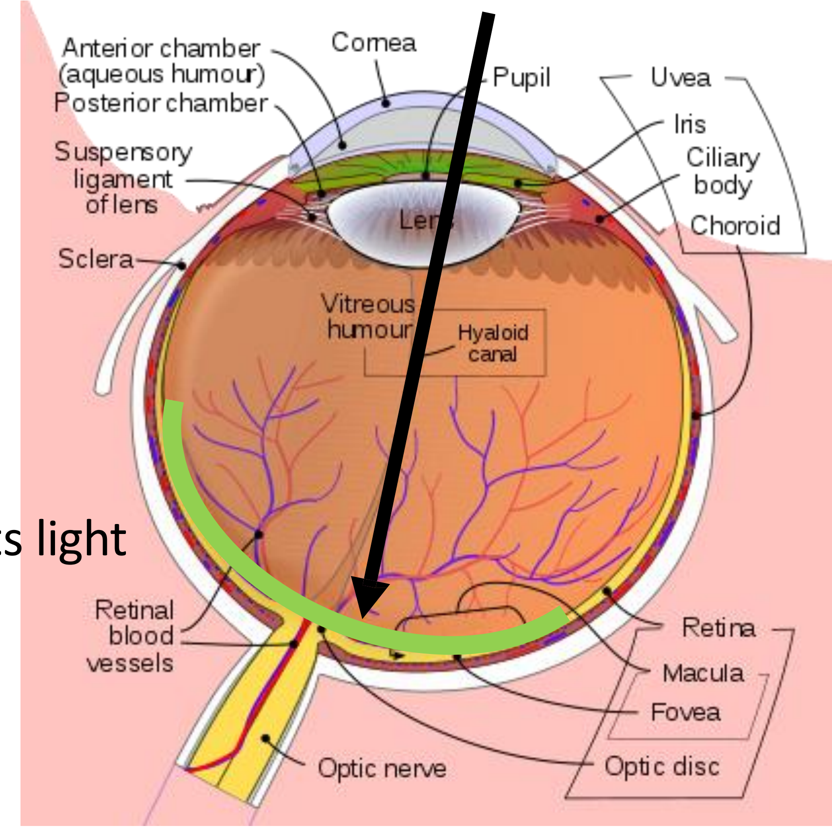
A group of people sitting on a boat in the water.

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

Xu et al, “Show, attend, and Tell: Neural Image Cap$on Genera$on with Visual Attention”, ICML 2015

Human Vision: Fovea

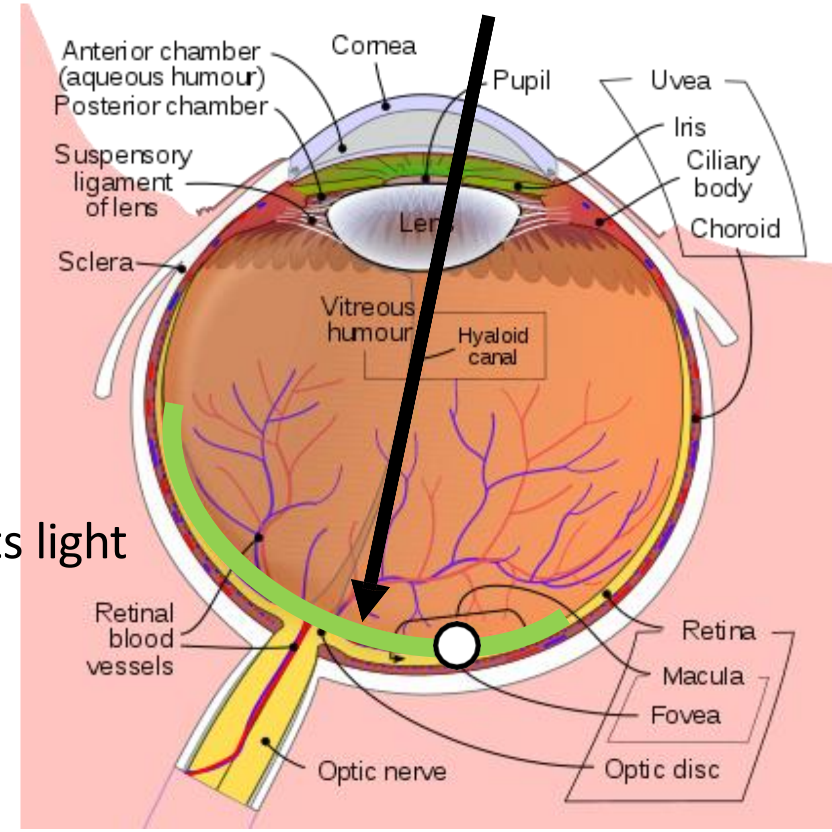
Light enters eye



Re0na
detects light

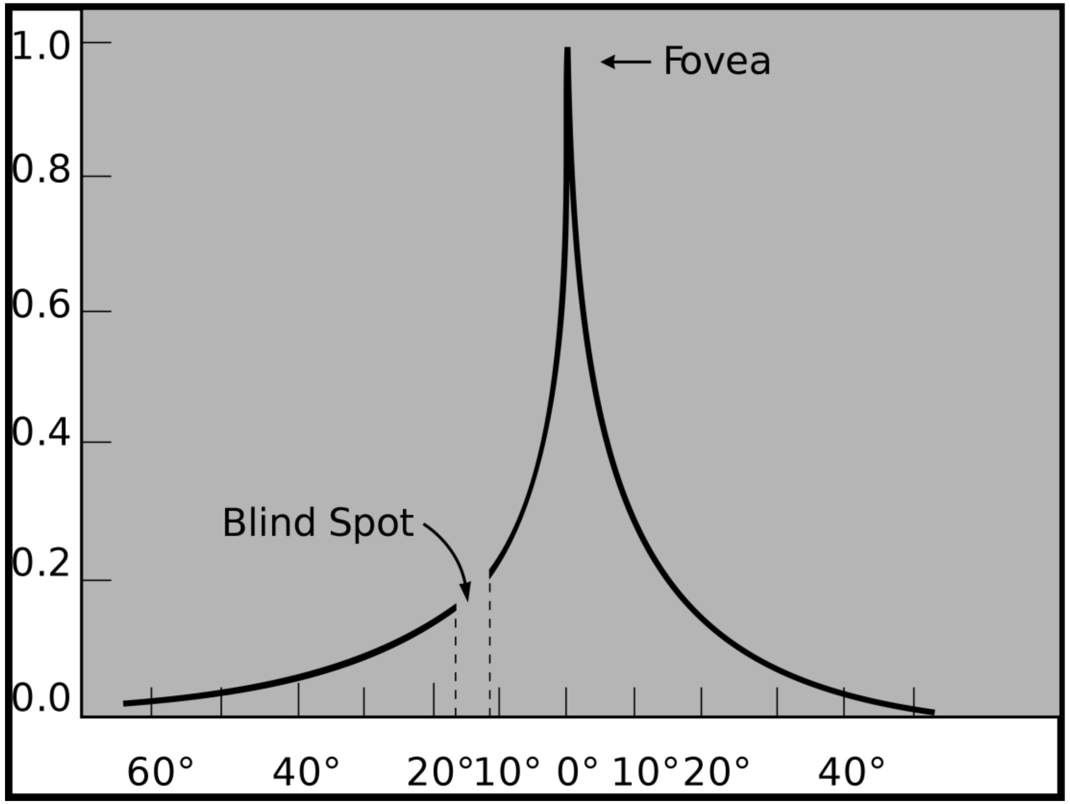
Human Vision: Fovea

Light enters eye



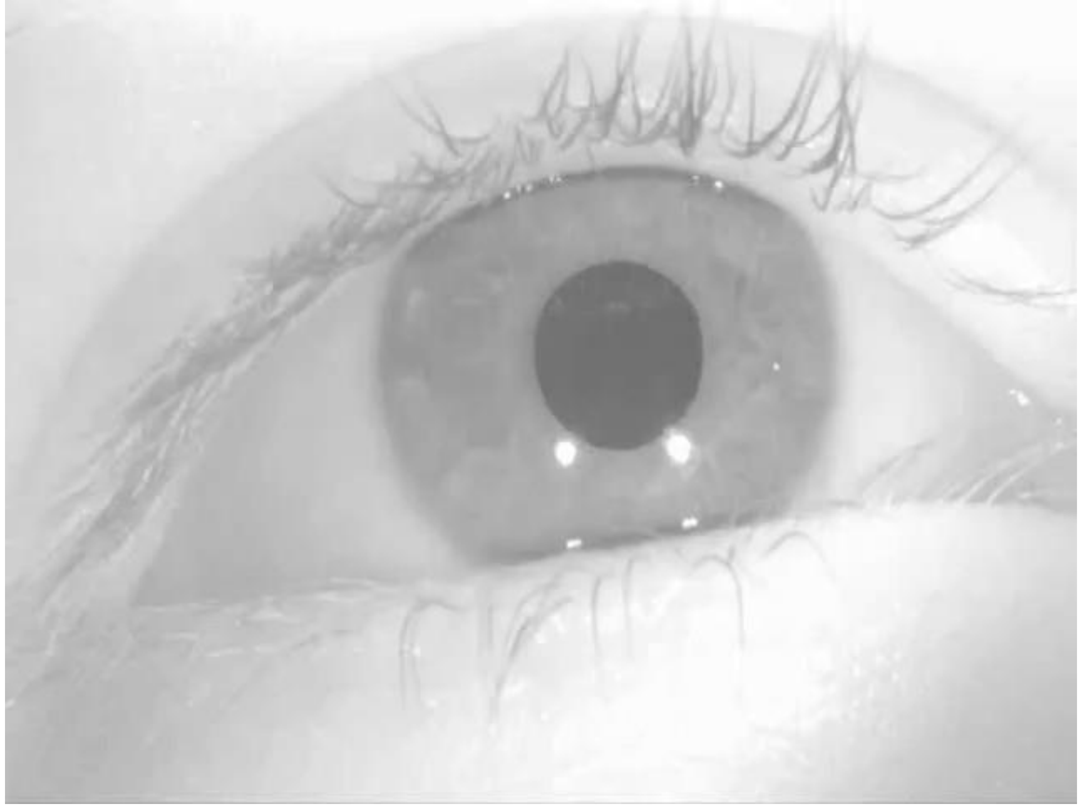
Re0na detects light

The **fovea** is a $ny region of the re$na 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 re$na that can see with high acuity

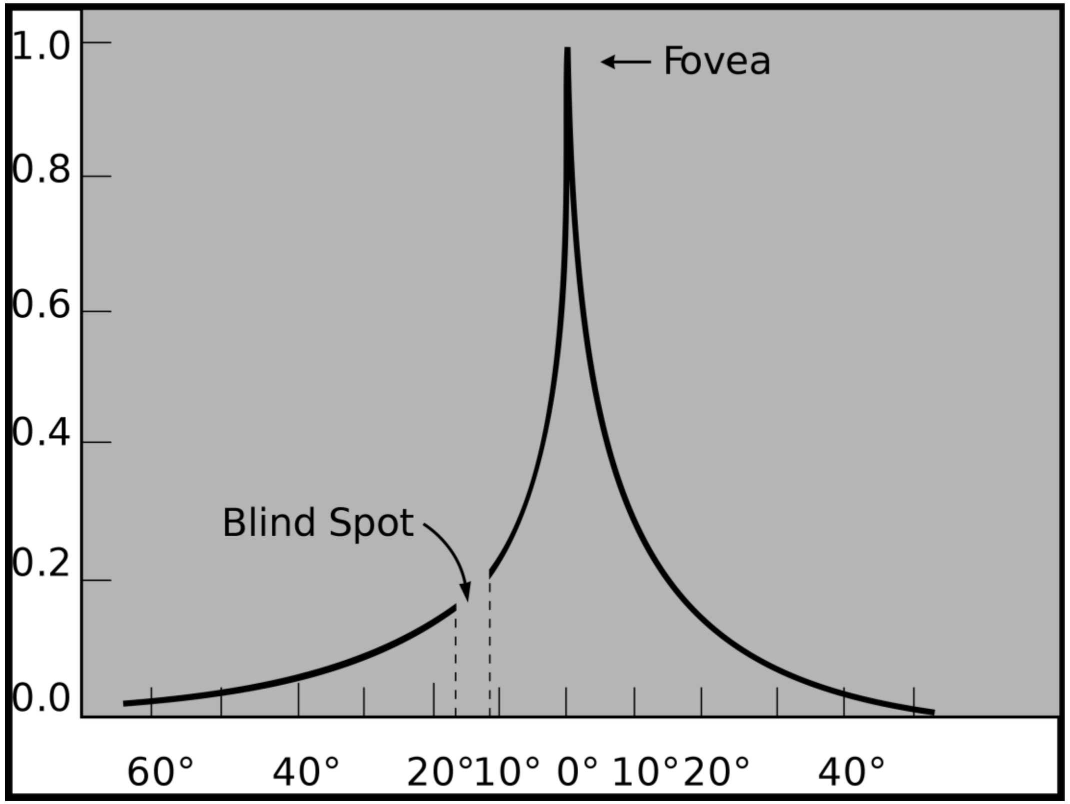
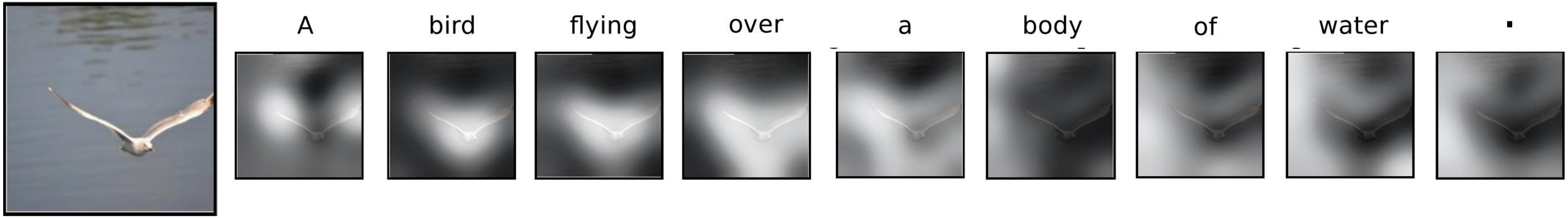
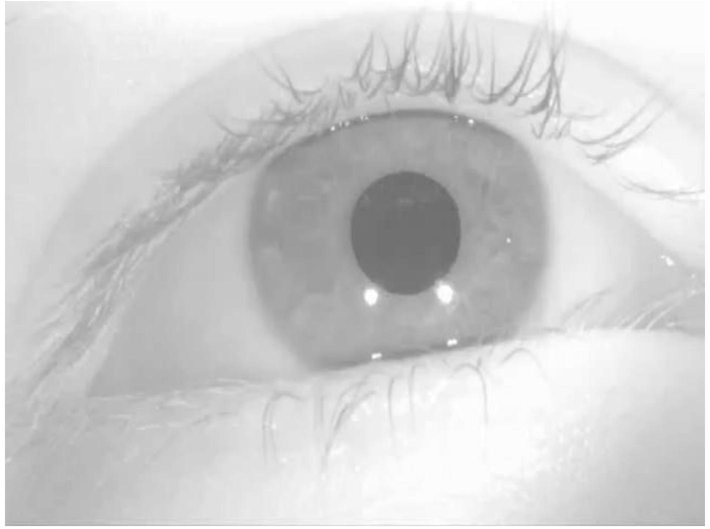


Image Captioning with RNNs and Attention



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



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

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

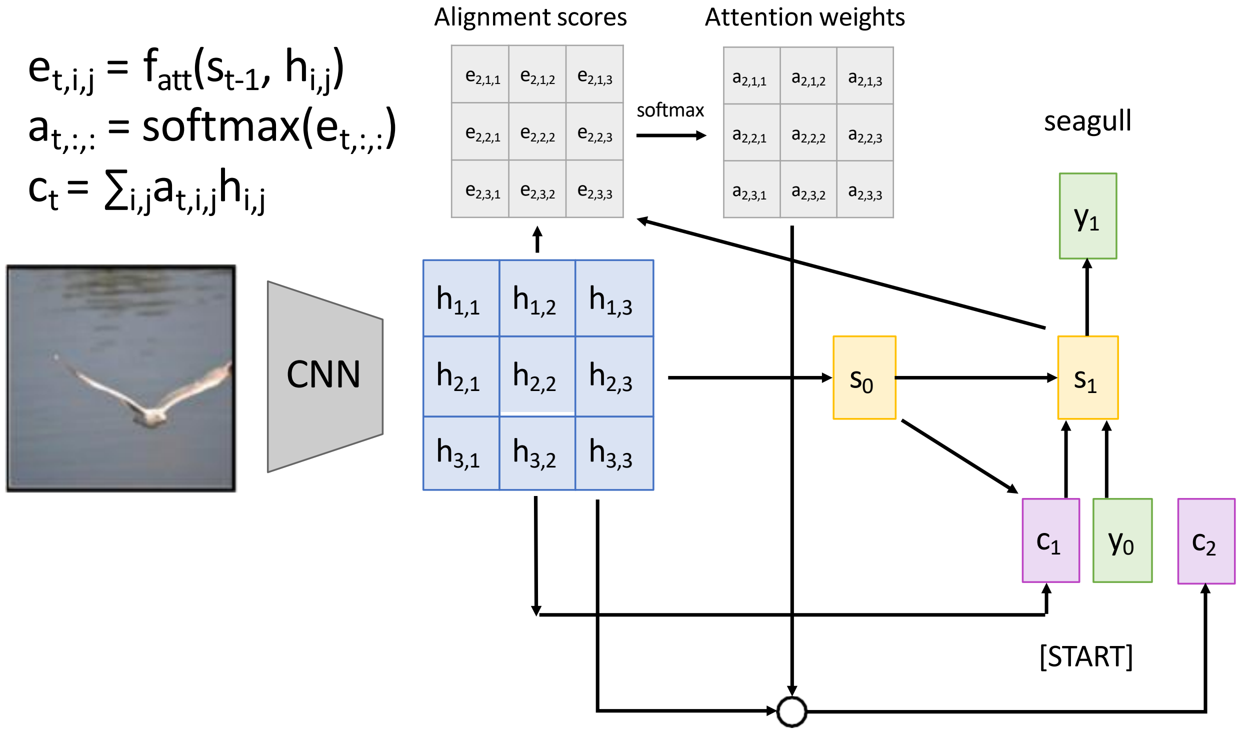
"**Listen, attend, and walk**" *(Mei et al, AAAI 2016)*
Process text, attend to text regions, output navigation commands

"**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

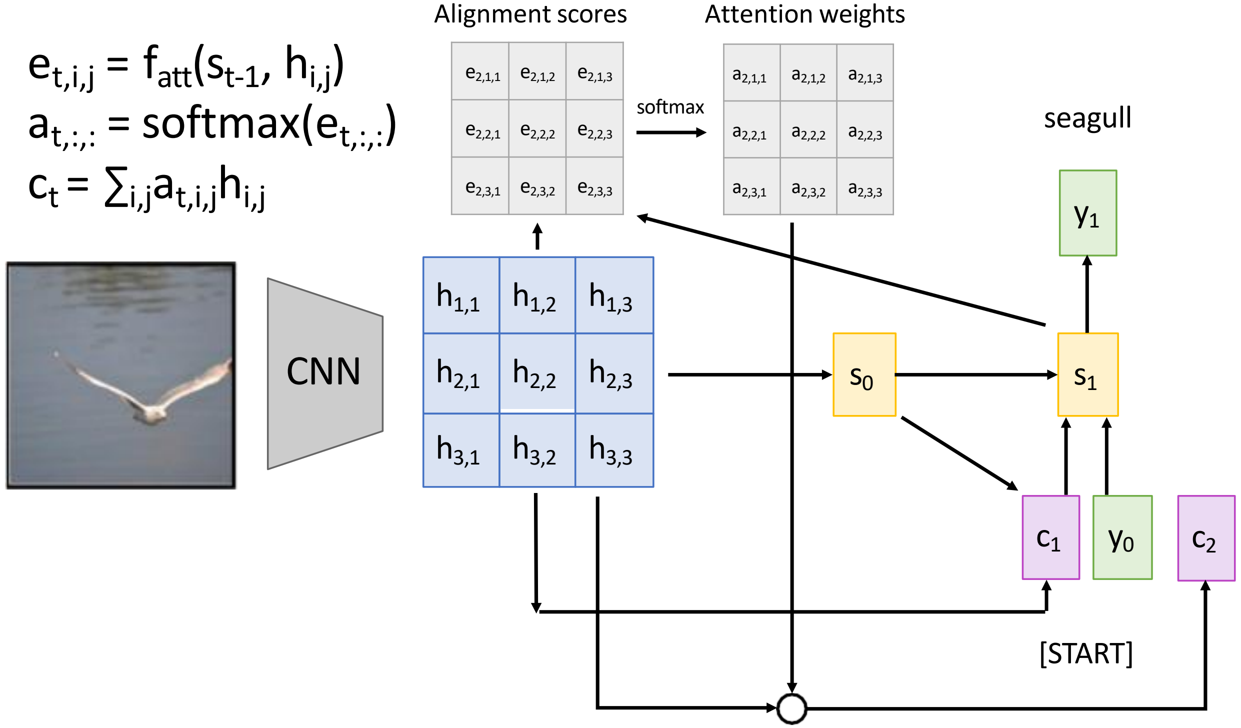
Inputs:
Query vector: **q** (Shape: D_Q)
Input vectors: **X** (Shape: N_X x D_X)
Similarity function: f_{att}



Computation:
Similarities: e (Shape: N_X) $e_i = f_{att}(\mathbf{q}, \mathbf{X_i})$
Attention weights: a = softmax(e) (Shape: N_X)
Output vector: $y = \sum_i a_i \mathbf{X_i}$ (Shape: D_X)

Attention Layer

Inputs:
Query vector: $\mathbf{q}$ (Shape: D_Q)
Input vectors: $\mathbf{X}$ (Shape: N_X x D_Q)
Similarity function: $\boxed{\text{dot product}}$

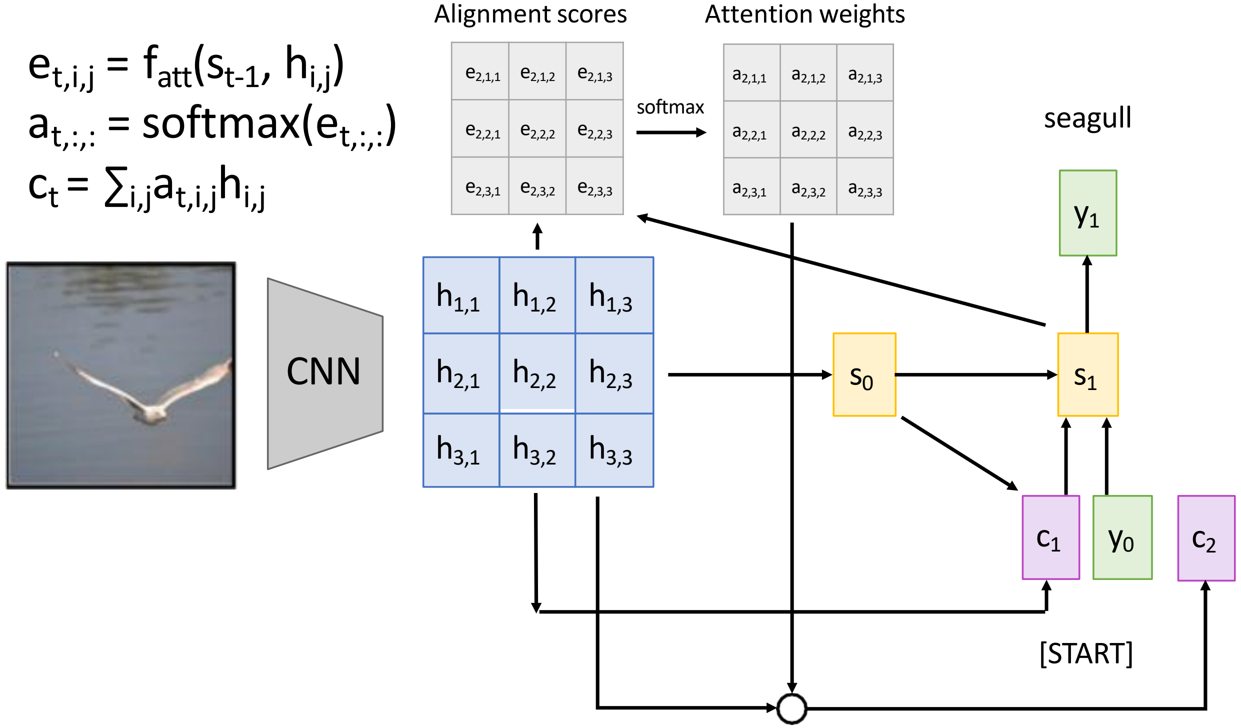


Computation:
Similarities: e (Shape: N_X) $\boxed{e_i = \mathbf{q} \cdot \mathbf{X}_i}$
Attention weights: $a = \text{softmax}(e)$ (Shape: N_X)
Output vector: $y = \sum_i a_i \mathbf{X}_i$ (Shape: D_X)

Changes:
- Use dot product for similarity

Attention Layer

Inputs:
Query vector: **q** (Shape: D_Q)
Input vectors: **X** (Shape: N_X x D_Q)
Similarity function: $\boxed{\text{scaled dot product}}$



Computation:
Similarities: e (Shape: N_X) $e_i = \mathbf{q} \cdot \mathbf{X}_i \boxed{/ \text{sqrt}(D_Q)}$
Attention weights: a = softmax(e) (Shape: N_X)
Output vector: $y = \sum_i a_i \mathbf{X}_i$ (Shape: D_X)

Changes:
- Use **scaled** dot product for similarity

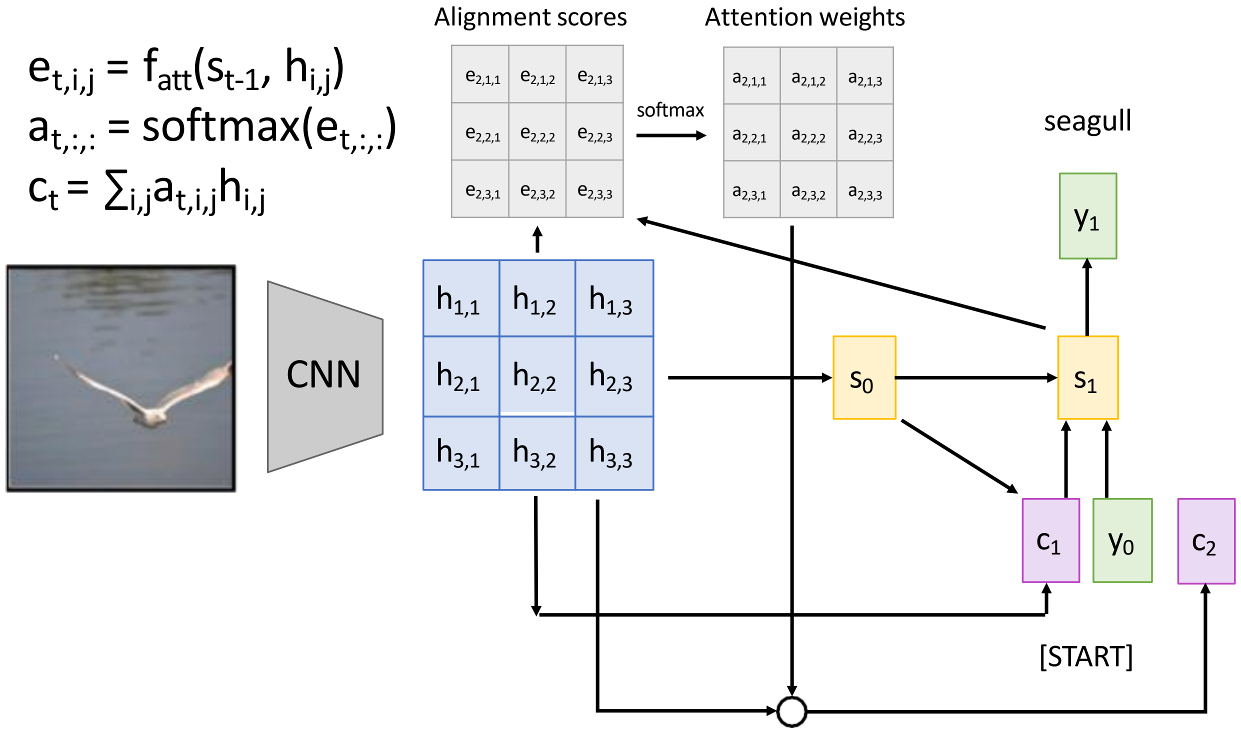
Attention Layer

Inputs:
Query vector: **q** (Shape: D_Q)
Input vectors: **X** (Shape: N_X x D_Q)
Similarity function: $\boxed{\text{scaled dot product}}$

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

Computation:
Similarities: e (Shape: N_X) $e_i = \mathbf{q} \cdot \mathbf{X}_i \boxed{/ \text{ sqrt}(D_Q)}$
Attention weights: a = softmax(e) (Shape: N_X)
Output vector: $y = \sum_i a_i \mathbf{X}_i$ (Shape: D_X)

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

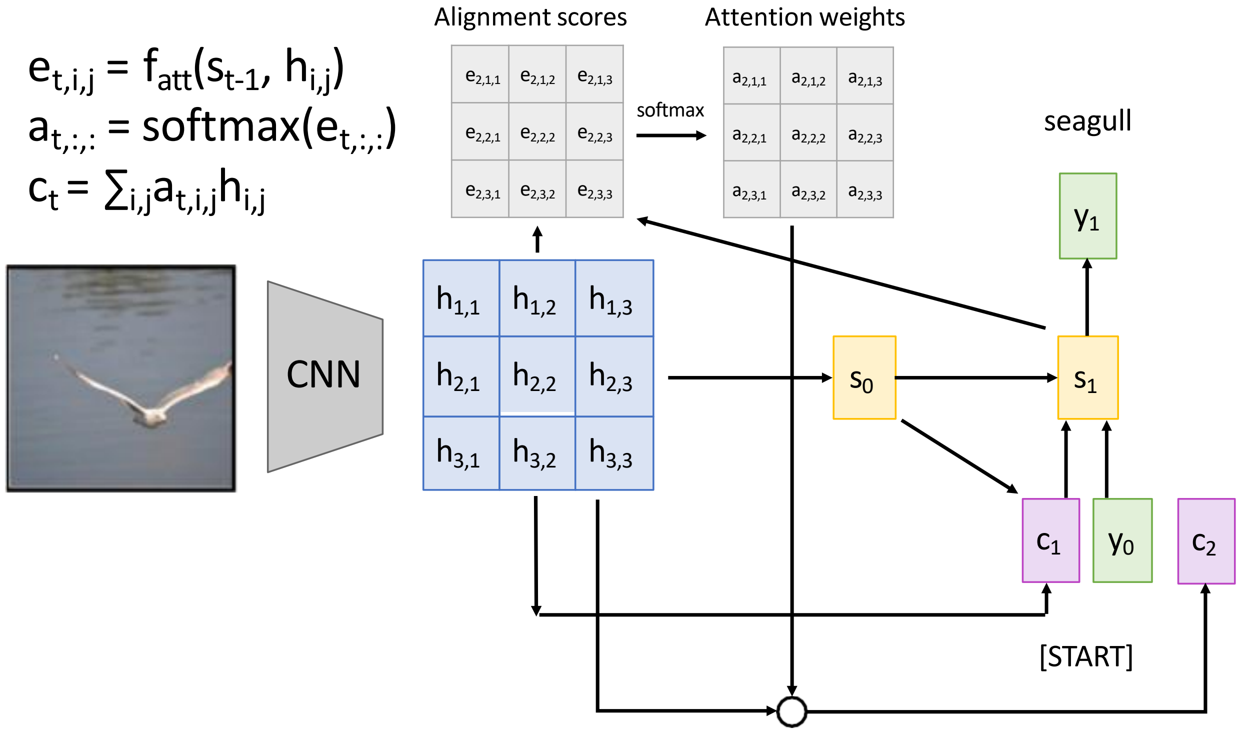


Changes:
- Use **scaled** dot product for similarity

Attention Layer

Inputs:
Query vectors: $\boxed{\mathbf{Q} \text{ (Shape: } N_Q \times D_Q)}$
Input vectors: $\mathbf{X}$ (Shape: $N_X \times D_Q$)

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



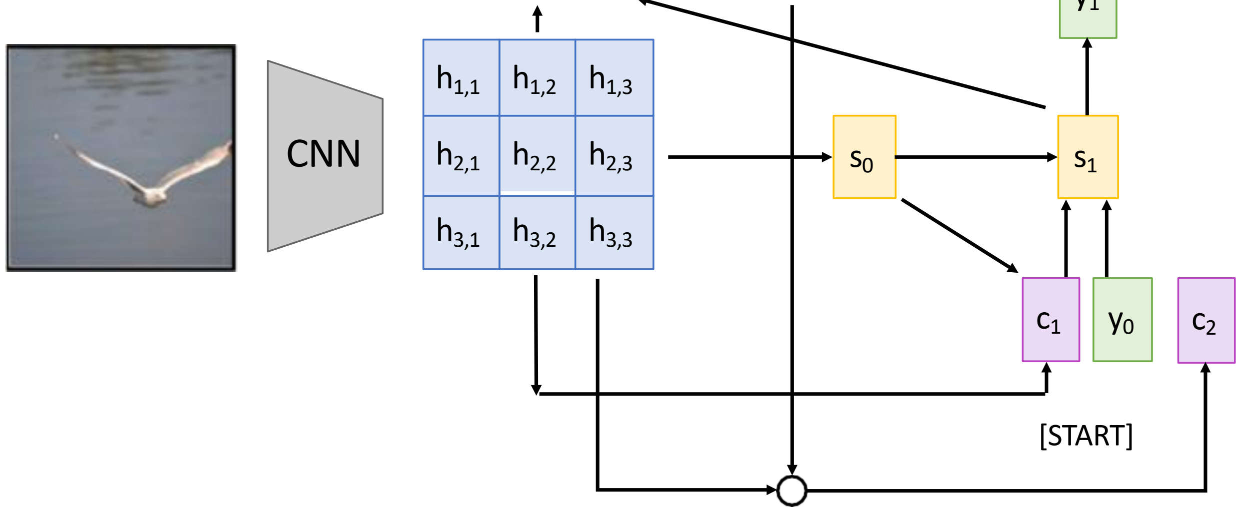
Computation:
Similarities: $E = \mathbf{Q}\mathbf{X}^T$ (Shape: $N_Q \times N_X$) $E_{i,j} = \mathbf{Q_i} \cdot \mathbf{X_j} / \text{sqrt}(D_Q)$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_Q \times N_X$)
Output vectors: $Y = A\mathbf{X}$ (Shape: $N_Q \times D_X$) $Y_i = \sum_j A_{i,j} X_j$

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

Attention Layer

Inputs:
Query vectors: $\color{green}{Q}$ (Shape: $N_Q \times D_Q$)
Input vectors: $\color{blue}{X}$ (Shape: $N_X \times D_X$)
Key matrix: $\color{orange}{W_K}$ (Shape: $D_X \times D_Q$)
Value matrix: $\color{purple}{W_V}$ (Shape: $D_X \times D_V$)

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



Computation:
Key vectors: $\color{orange}{K} = \color{blue}{X}\color{orange}{W_K}$ (Shape: $N_X \times D_Q$)
Value Vectors: $\color{purple}{V} = \color{blue}{X}\color{purple}{W_V}$ (Shape: $N_X \times D_V$)
Similarities: $E = \color{green}{Q}\color{orange}{K}^T$ (Shape: $N_Q \times N_X$) $E_{i,j} = \color{green}{Q_i} \cdot \color{orange}{K_j} / \text{sqrt}(D_Q)$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_Q \times N_X$)
Output vectors: $Y = A\color{purple}{V}$ (Shape: $N_Q \times D_V$) $Y_i = \sum_j A_{i,j} \color{purple}{V_j}$

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:

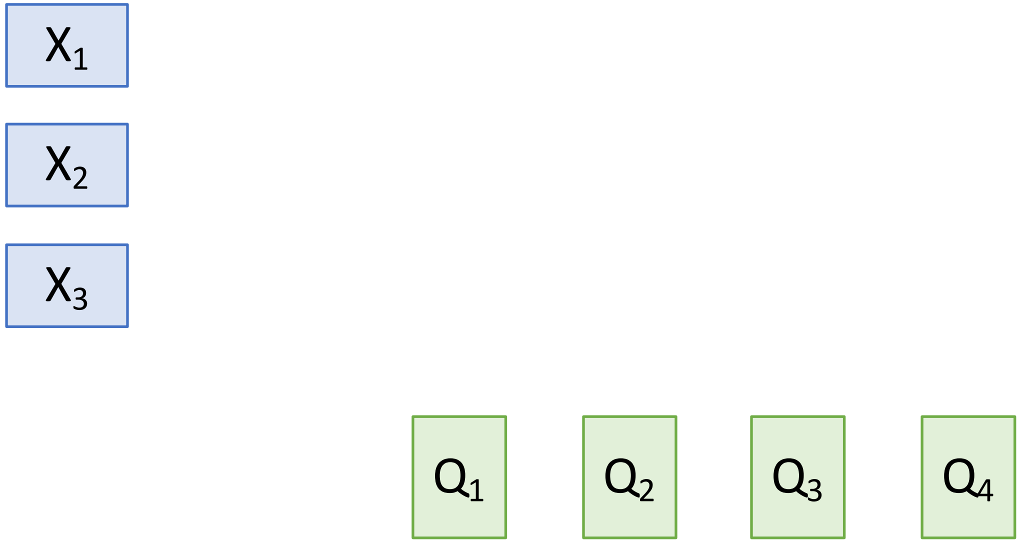
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: $N_X \times D_Q$)

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

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

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

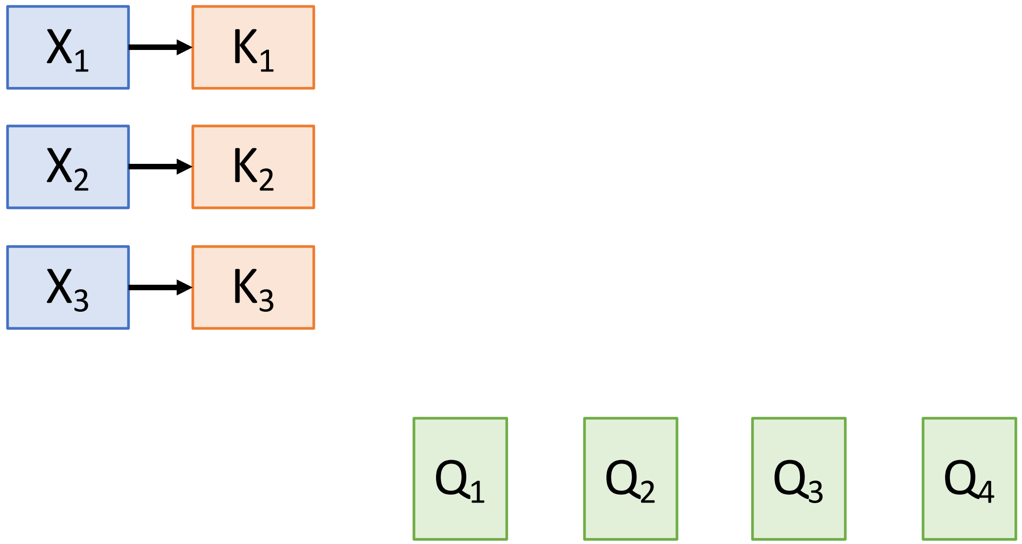
Output vectors: $Y = A\mathbf{V}$ (Shape: $N_Q \times D_V$) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$



Attention Layer

Inputs:
Query vectors: **Q** (Shape: N_Q x D_Q)
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)

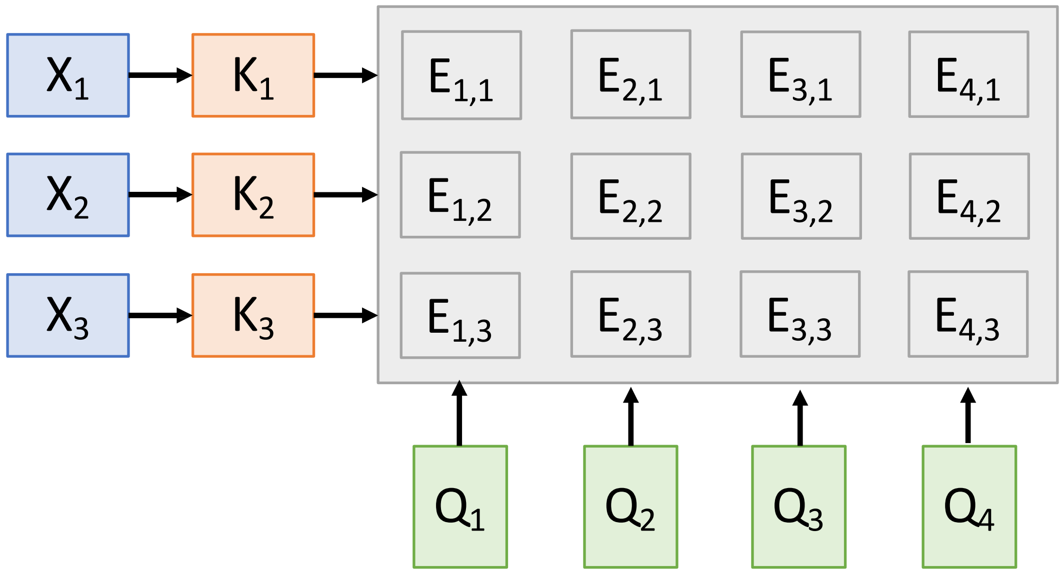
Computation:
Key vectors: **K** = $\mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: **V** = $\mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: E = $\mathbf{QK^T}$ (Shape: N_Q x N_X) $E_{i,j} = \mathbf{Q_i} \cdot \mathbf{K_j} / \mathrm{sqrt}(D_Q)$
Attention weights: A = softmax(E, dim=1) (Shape: N_Q x N_X)
Output vectors: Y = A**V** (Shape: N_Q x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$



Attention Layer

Inputs:
Query vectors: **Q** (Shape: N_Q x D_Q)
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)

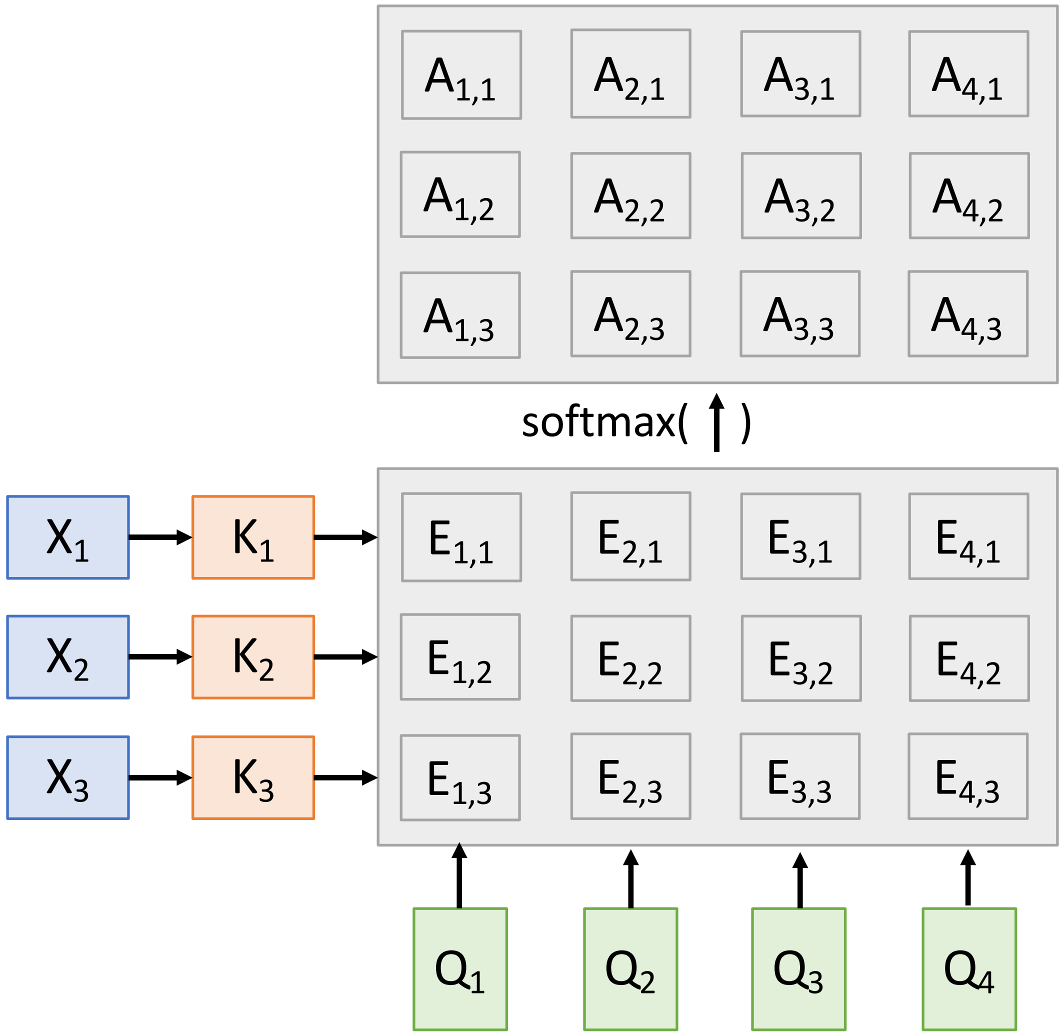
Computation:
Key vectors: **K** = $\mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: **V** = $\mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: E = $\mathbf{QK^T}$ (Shape: N_Q x N_X) $E_{i,j} = \mathbf{Q_i} \cdot \mathbf{K_j} / \mathrm{sqrt}(D_Q)$
Attention weights: A = softmax(E, dim=1) (Shape: N_Q x N_X)
Output vectors: Y = A**V** (Shape: N_Q x D_V) $Y_i = \sum_j A_{i,j} \mathbf{V_j}$



Attention Layer

Inputs:
Query vectors: **Q** (Shape: $N_Q \times D_Q$)
Input vectors: **X** (Shape: $N_X \times D_X$)
Key matrix: $\mathbf{W_K}$ (Shape: $D_X \times D_Q$)
Value matrix: $\mathbf{W_V}$ (Shape: $D_X \times D_V$)

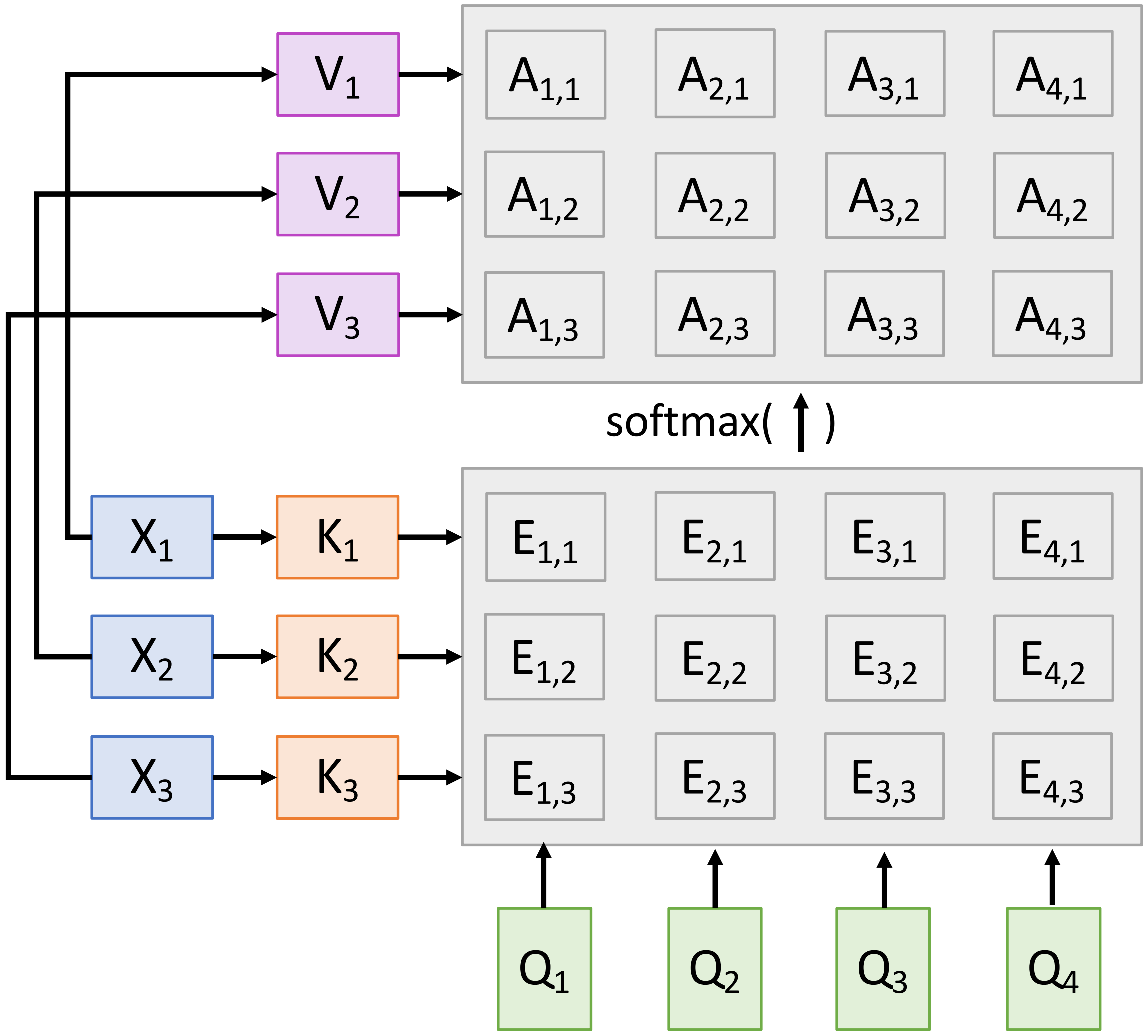
Computation:
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: $N_X \times D_Q$)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: $N_X \times D_V$)
Similarities: $E = \mathbf{QK^T}$ (Shape: $N_Q \times N_X$) $E_{i,j} = \mathbf{Q_i} \cdot \mathbf{K_j} / \text{sqrt}(D_Q)$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_Q \times N_X$)
Output vectors: $Y = A\mathbf{V}$ (Shape: $N_Q \times D_V$) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$



Attention Layer

Inputs:
Query vectors: Q (Shape: N_Q x D_Q)
Input vectors: X (Shape: N_X x D_X)
Key matrix: W_K (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)

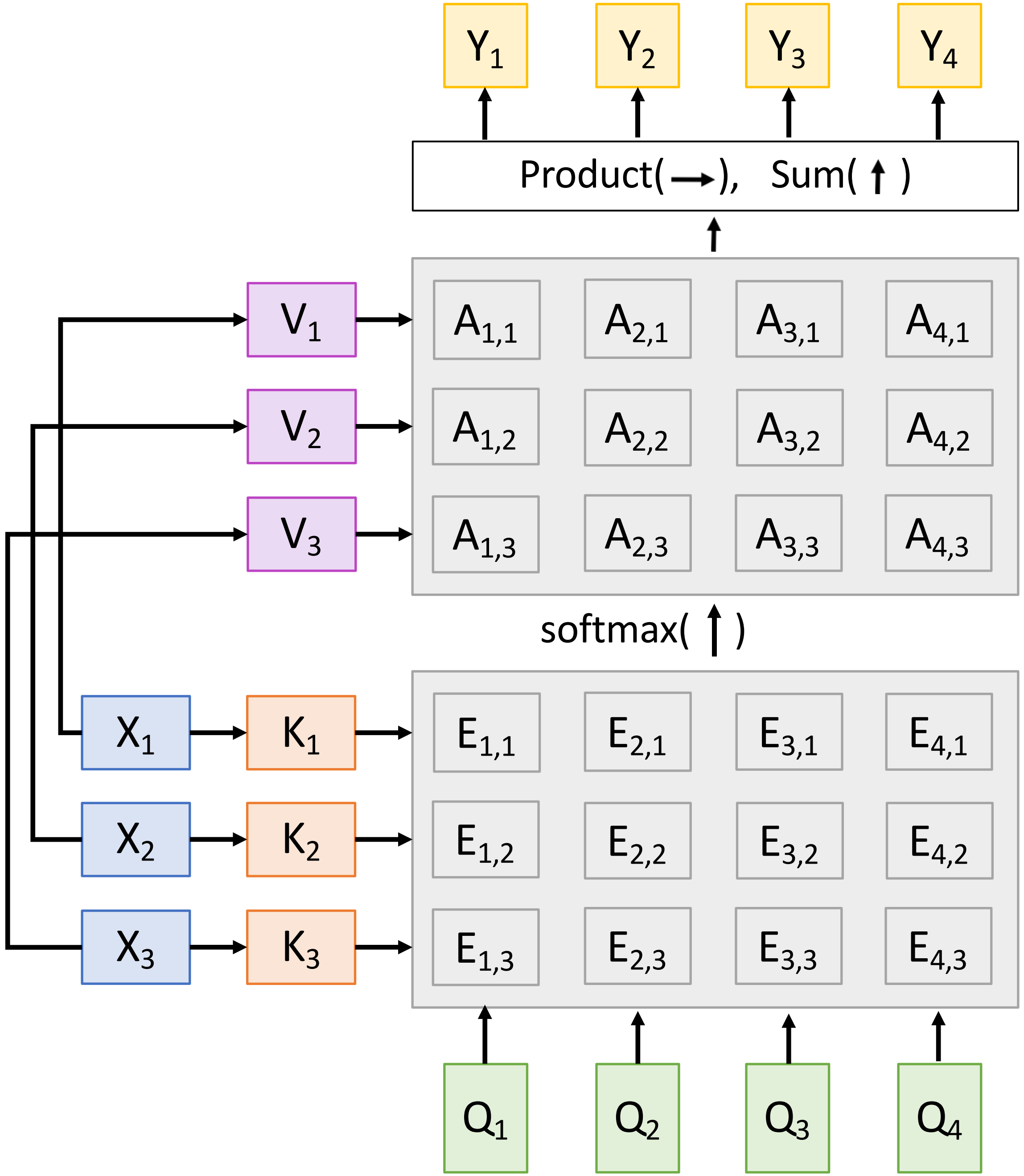
Computation:
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value Vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T$ (Shape: N_Q x N_X) $E_{i,j} = Q_i \cdot K_j / \text{sqrt}(D_Q)$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: N_Q x N_X)
Output vectors: $Y = AV$ (Shape: N_Q x D_V) $Y_i = \sum_j A_{i,j} V_j$



Attention Layer

Inputs:
Query vectors: $\mathbf{Q}$ (Shape: N_Q x D_Q)
Input vectors: $\mathbf{X}$ (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)

Computation:
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{QK}^T$ (Shape: N_Q x N_X) $E_{i,j} = \mathbf{Q_i} \cdot \mathbf{K_j} / \mathrm{sqrt}(D_Q)$
Attention weights: A = softmax(E, dim=1) (Shape: N_Q x N_X)
Output vectors: $Y = A\mathbf{V}$ (Shape: N_Q x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$

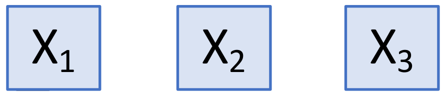


Self-Attention Layer

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

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: **W_K** (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)
Query matrix: **W_Q** (Shape: D_X x D_Q)

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value Vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T$ (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j / \text{sqrt}(D_Q)$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: N_X x N_X)
Output vectors: $Y = AV$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$

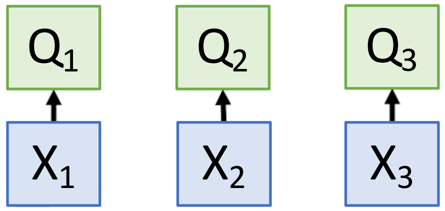


Self-Attention Layer

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

Inputs:
Input vectors: **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: $E = \mathbf{QK}^T$ (Shape: $N_X \times N_X$) $E_{i,j} = \mathbf{Q_i} \cdot \mathbf{K_j} / \mathrm{sqrt}(D_Q)$
Attention weights: $A = \mathrm{softmax}(E, \mathrm{dim}=1)$ (Shape: $N_X \times N_X$)
Output vectors: $Y = A\mathbf{V}$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$

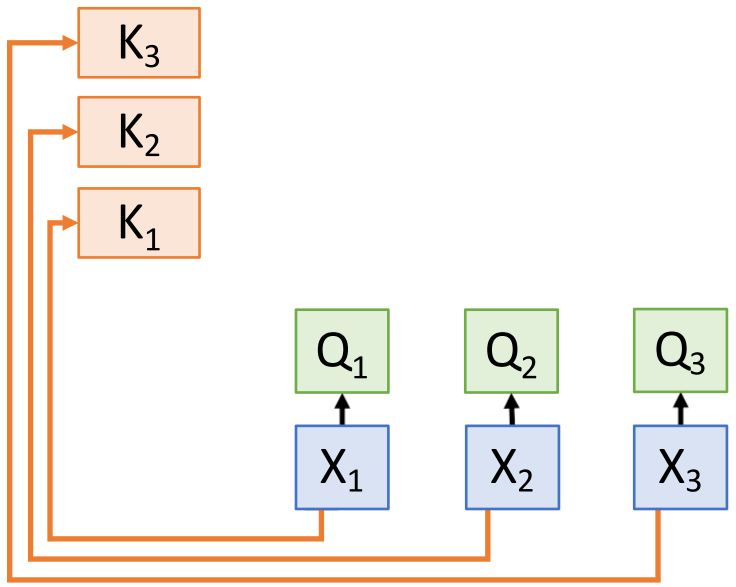


Self-Attention Layer

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: $E = \mathbf{QK}^T$ (Shape: $N_X \times N_X$) $E_{i,j} = \mathbf{Q_i} \cdot \mathbf{K_j} / \mathrm{sqrt}(D_Q)$
Attention weights: $A = \mathrm{softmax}(E, \mathrm{dim}=1)$ (Shape: $N_X \times N_X$)
Output vectors: $Y = A\mathbf{V}$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$

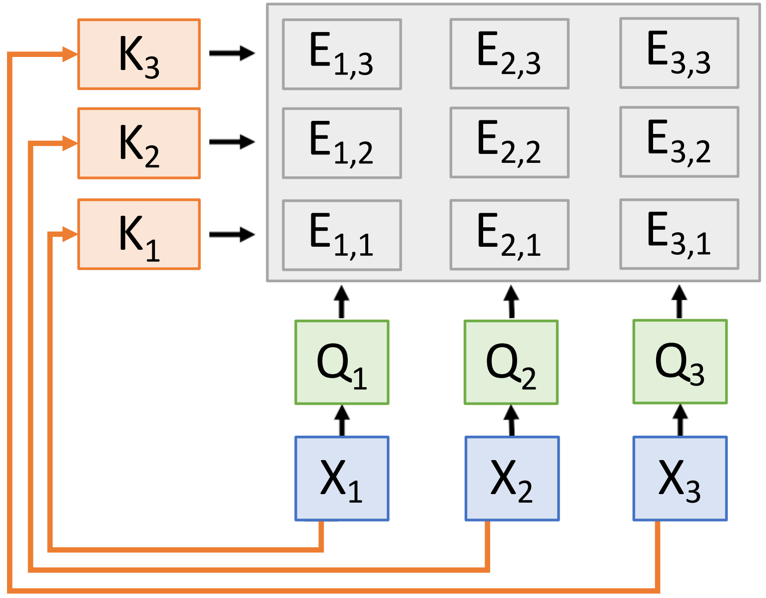


Self-Attention Layer

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

Inputs:
Input vectors: $\mathbf{X}$ (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

Computation:
Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{QK}^T$ (Shape: N_X x N_X) $E_{i,j} = \mathbf{Q_i} \cdot \mathbf{K_j} / \mathrm{sqrt}(D_Q)$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X) **Output vectors**: $Y = A\mathbf{V}$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$

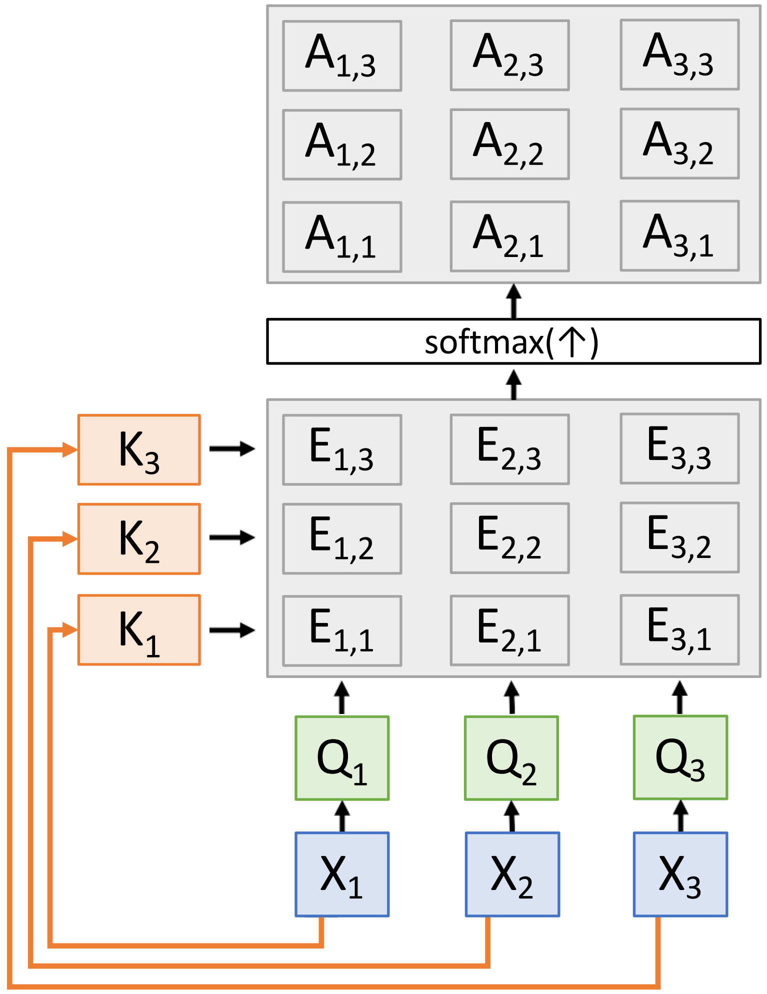


Self-Attention Layer

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

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

Computation:
Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{QK}^T$ (Shape: N_X x N_X) $E_{i,j} = \mathbf{Q_i} \cdot \mathbf{K_j} / \mathrm{sqrt}(D_Q)$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: $Y = A\mathbf{V}$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$

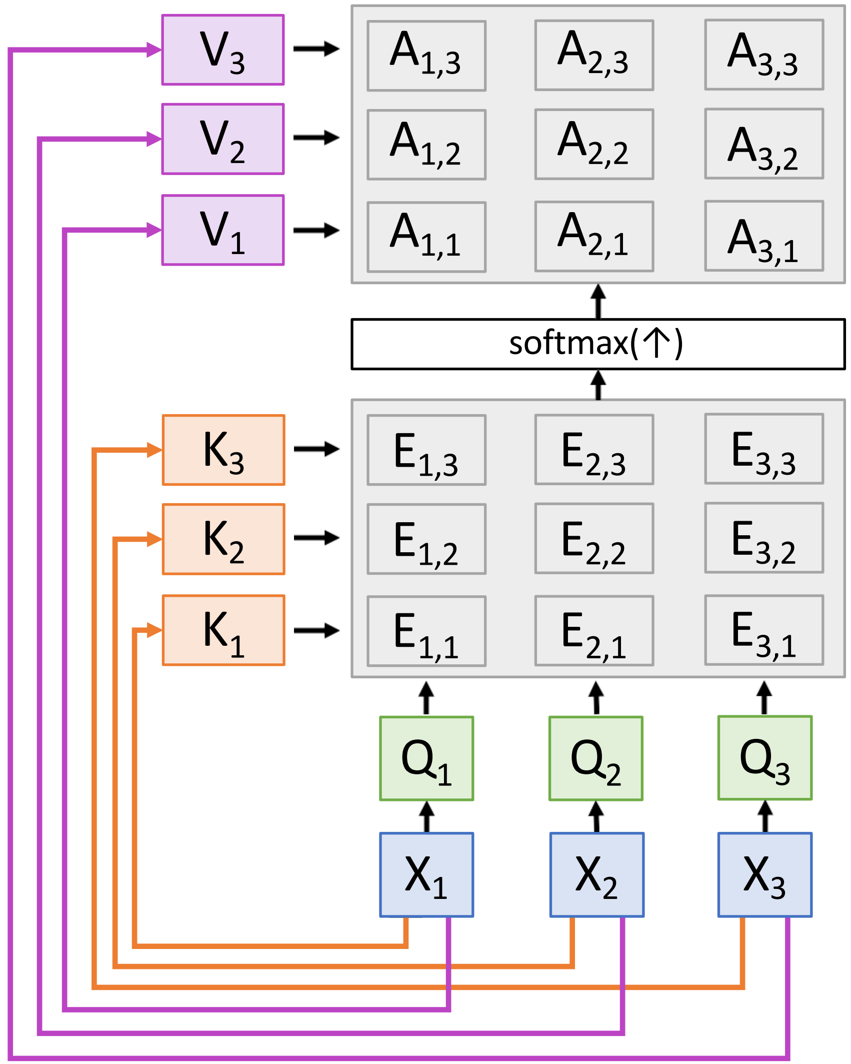


Self-Attention Layer

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

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

Computation:
Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{QK}^T$ (Shape: N_X x N_X) $E_{i,j} = \mathbf{Q_i} \cdot \mathbf{K_j} / \mathrm{sqrt}(D_Q)$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X) **Output vectors**: $Y = A\mathbf{V}$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$

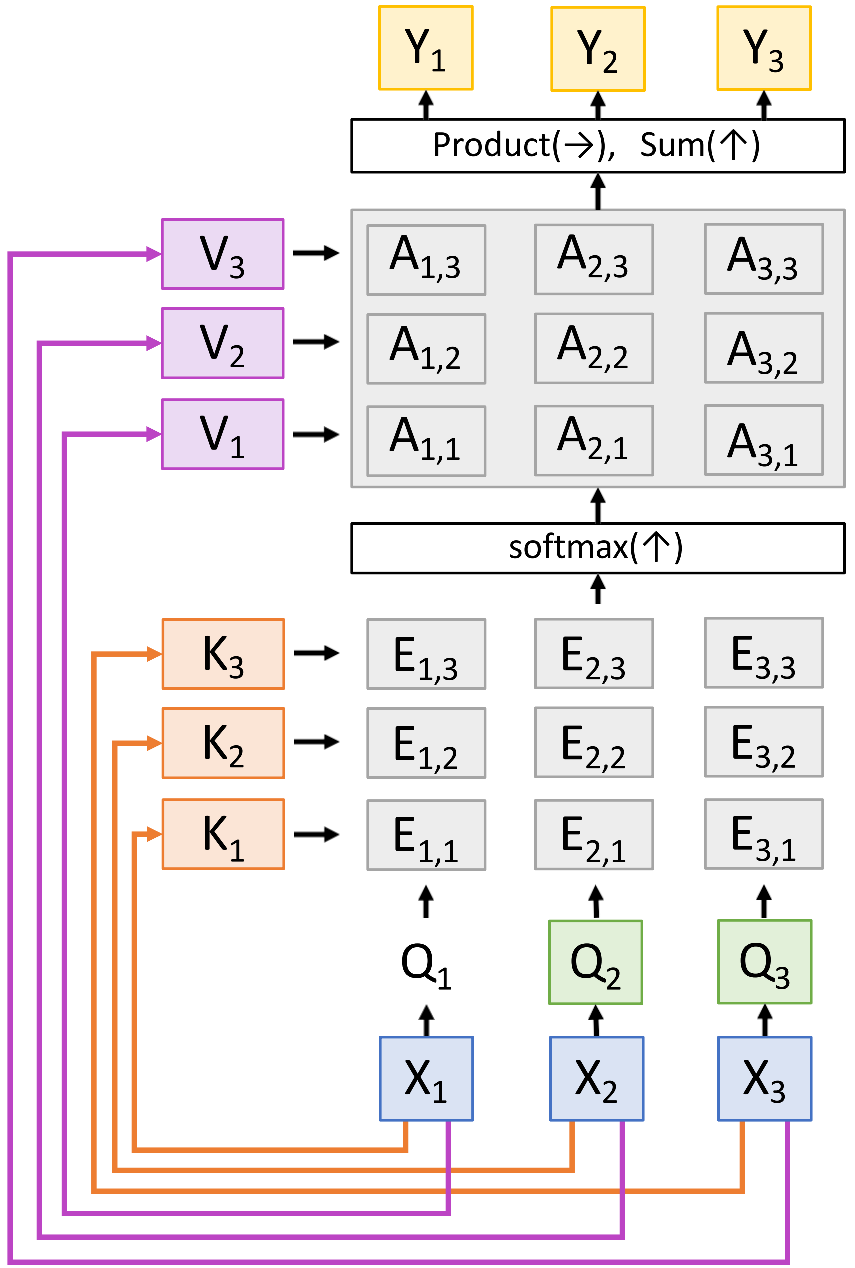


Self-Attention Layer

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

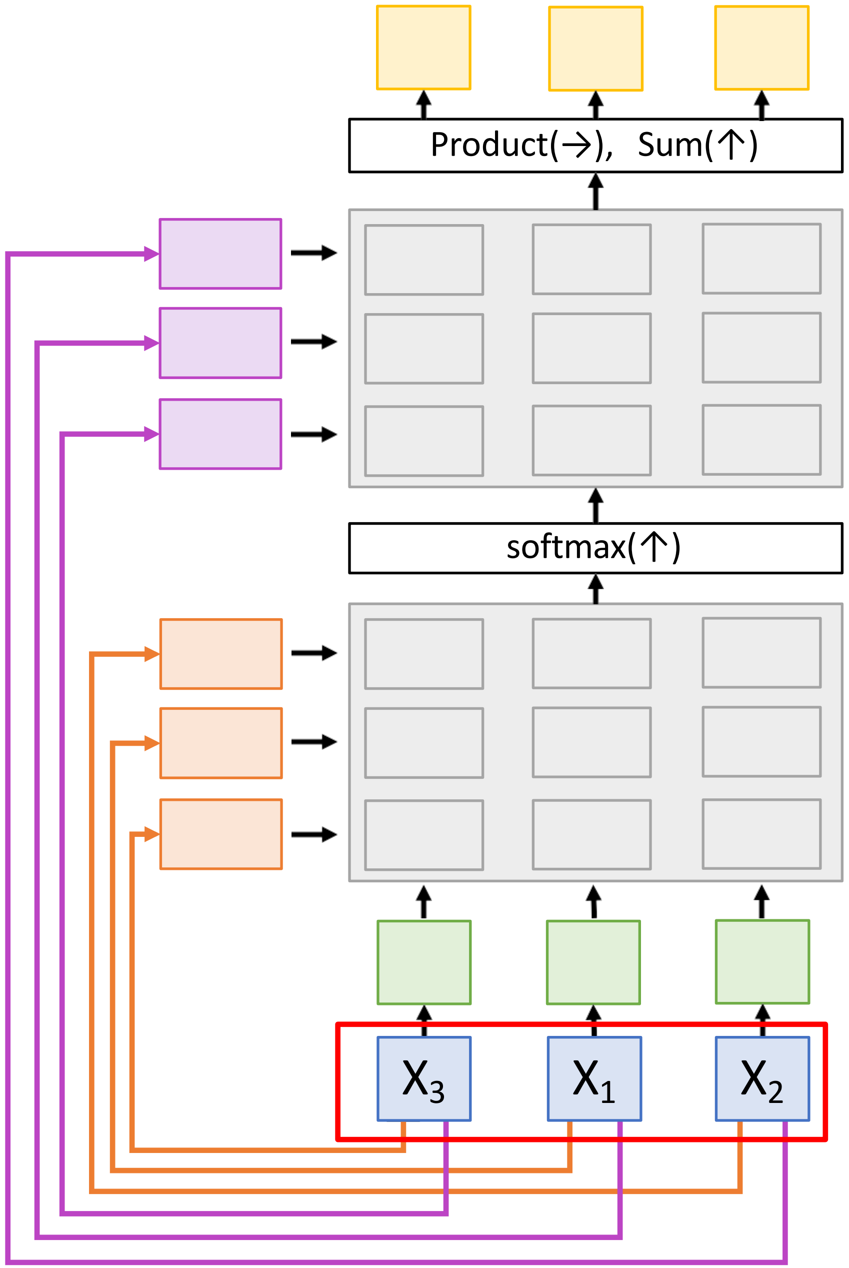
Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

Computation:
Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{QK}^T$ (Shape: N_X x N_X) $E_{i,j} = \mathbf{Q_i} \cdot \mathbf{K_j} / \mathrm{sqrt}(D_Q)$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: $Y = A\mathbf{V}$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$



Self-Attention Layer

Consider **permuting** the input vectors:



Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: **W_K** (Shape: D_X x D_Q)
Value matrix: **W_V** (Shape: D_X x D_V)
Query matrix: **W_Q** (Shape: D_X x D_Q)

Computation:
Query vectors: **Q** = **XW_Q**
Key vectors: **K** = **XW_K** (Shape: N_X x D_Q)
Value Vectors: **V** = **XW_V** (Shape: N_X x D_V)
Similarities: E = **QK^T** (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j / \text{sqrt}(D_Q)$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: Y = A**V** (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$

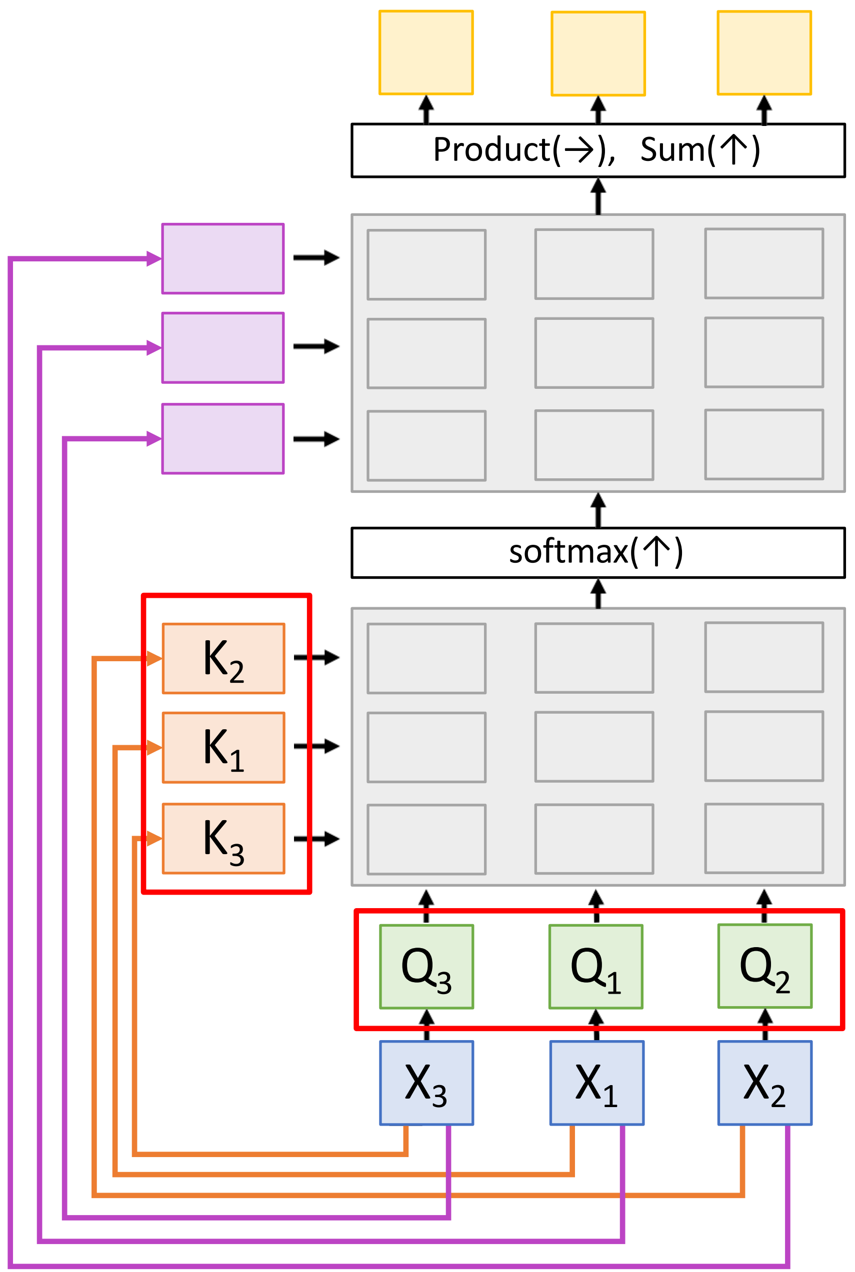
Self-Attention Layer

Consider **permuting** the input vectors:

Queries and Keys will be the same, but permuted

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

Computation:
Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{QK}^T$ (Shape: N_X x N_X) $E_{i,j} = \mathbf{Q_i} \cdot \mathbf{K_j} / \mathrm{sqrt}(D_Q)$
Attention weights: $A = \mathrm{softmax}(E, \mathrm{dim}=1)$ (Shape: N_X x N_X)
Output vectors: $Y = A\mathbf{V}$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$



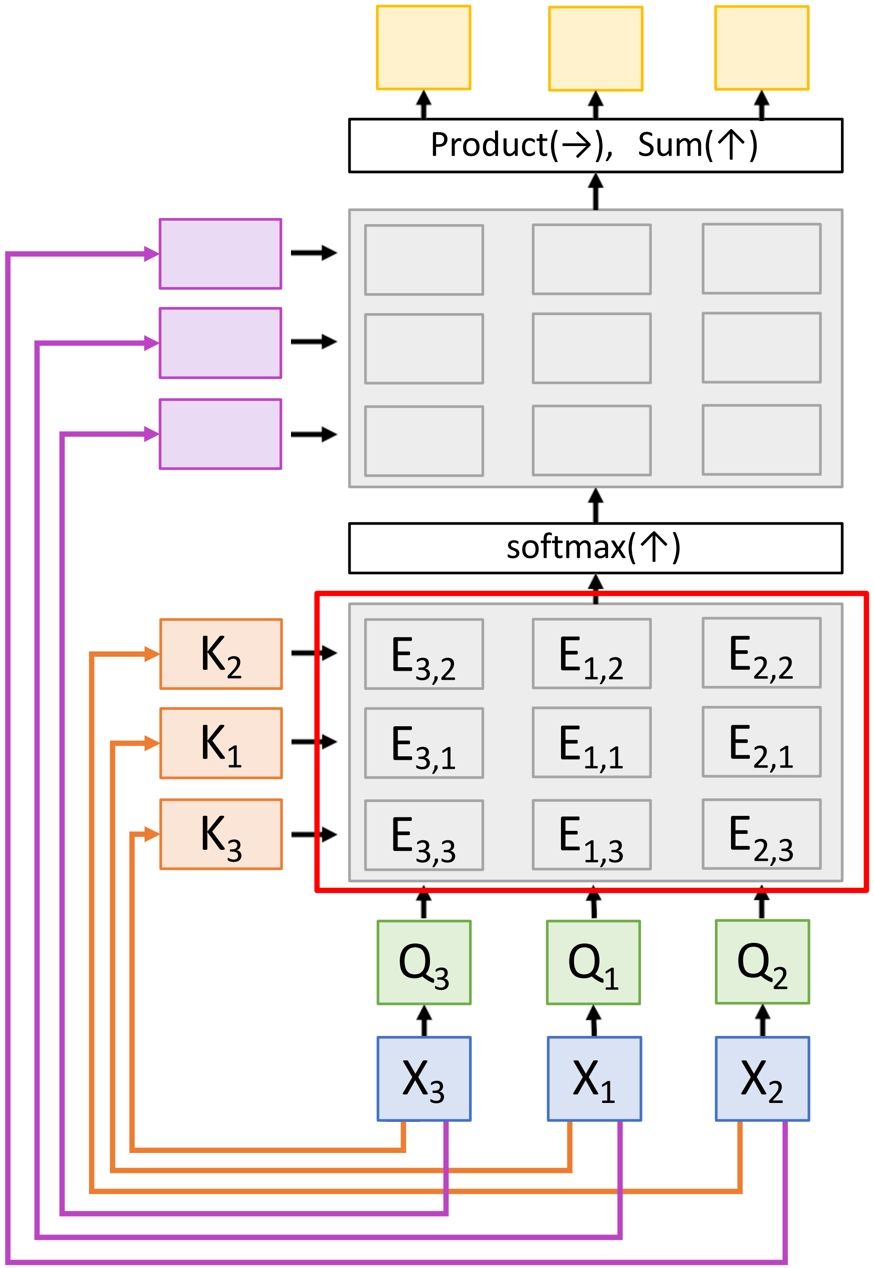
Self-Attention Layer

Consider **permuting** the input vectors:

Similarities will be the same, but permuted

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: W_K (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)
Query matrix: W_Q (Shape: D_X x D_Q)

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value Vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T$ (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j / \mathrm{sqrt}(D_Q)$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: $Y = AV$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$



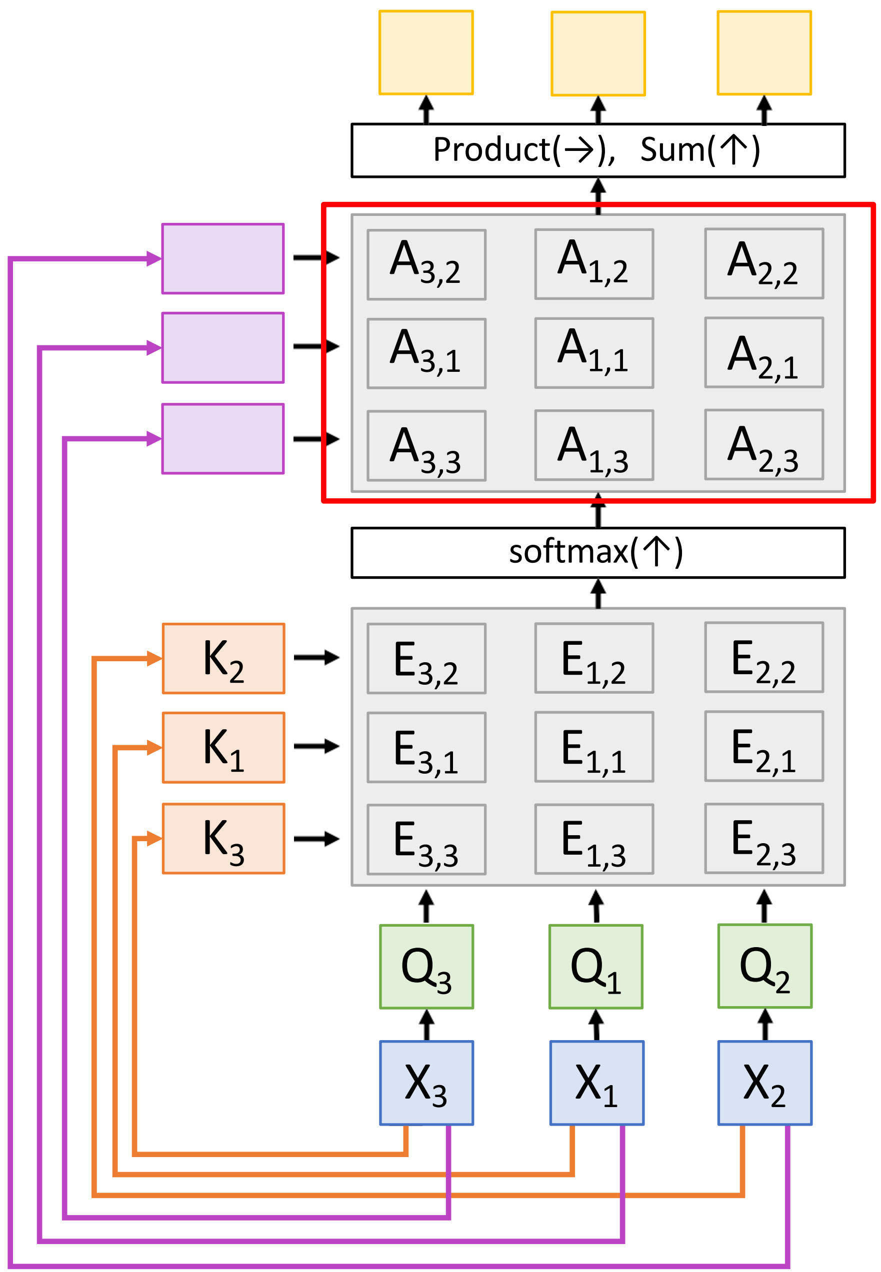
Self-Attention Layer

Consider **permuting** the input vectors:

Attention weights will be the same, but permuted

<u>**Inputs**</u>:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: **W_K** (Shape: D_X x D_Q)
Value matrix: **W_V** (Shape: D_X x D_V)
Query matrix: **W_Q** (Shape: D_X x D_Q)

<u>**Computation**</u>:
Query vectors: **Q** = **XW_Q**
Key vectors: **K** = **XW_K** (Shape: N_X x D_Q)
Value Vectors: **V** = **XW_V** (Shape: N_X x D_V)
Similarities: E = QK^T (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j / \text{sqrt}(D_Q)$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: Y = A**V** (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$



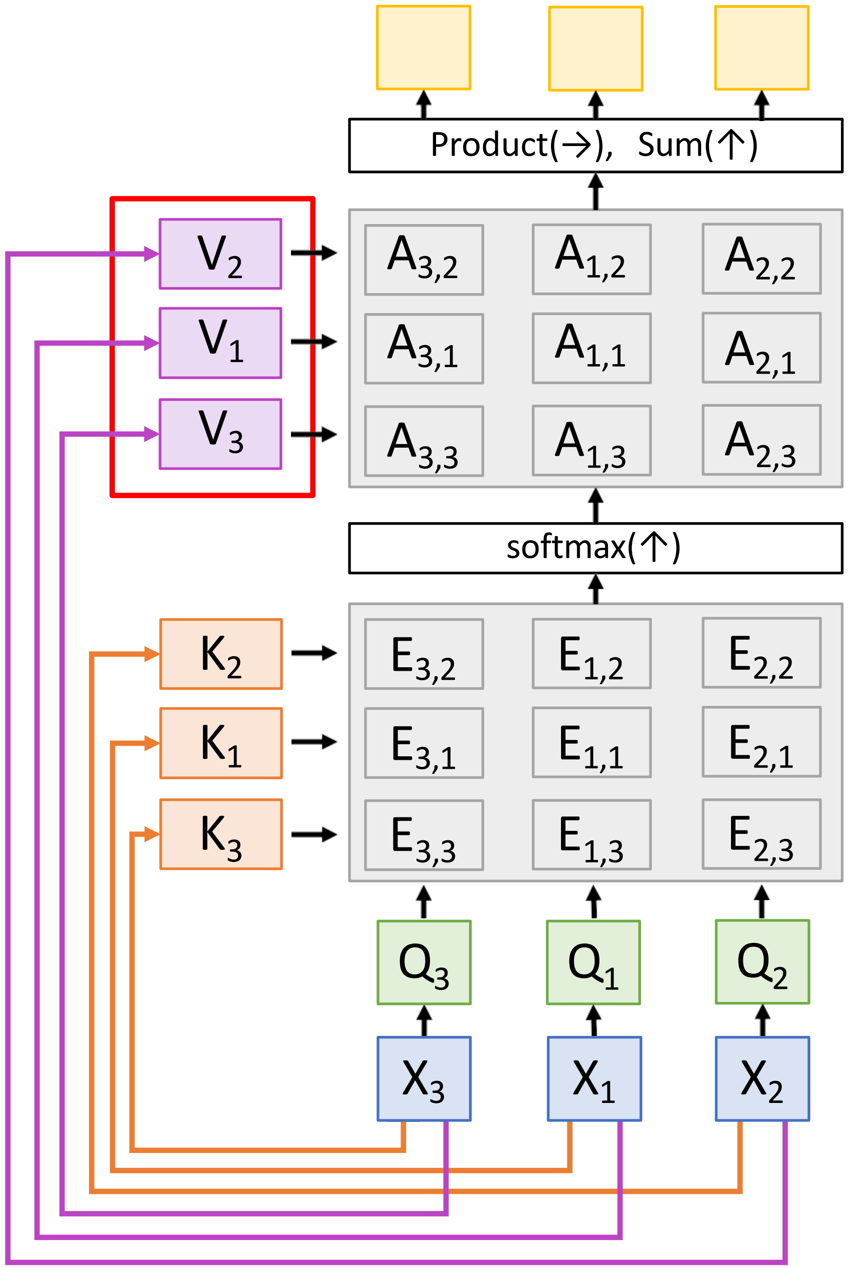
Self-Attention Layer

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: **W_K** (Shape: D_X x D_Q)
Value matrix: **W_V** (Shape: D_X x D_V)
Query matrix: **W_Q** (Shape: D_X x D_Q)

Computation:
Query vectors: **Q** = **XW_Q**
Key vectors: **K** = **XW_K** (Shape: N_X x D_Q)
Value Vectors: **V** = **XW_V** (Shape: N_X x D_V)
Similarities: E = **QK^T** (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j / \text{sqrt}(D_Q)$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: Y = A**V** (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$

Consider **permuting** the input vectors:

Values will be the same, but permuted



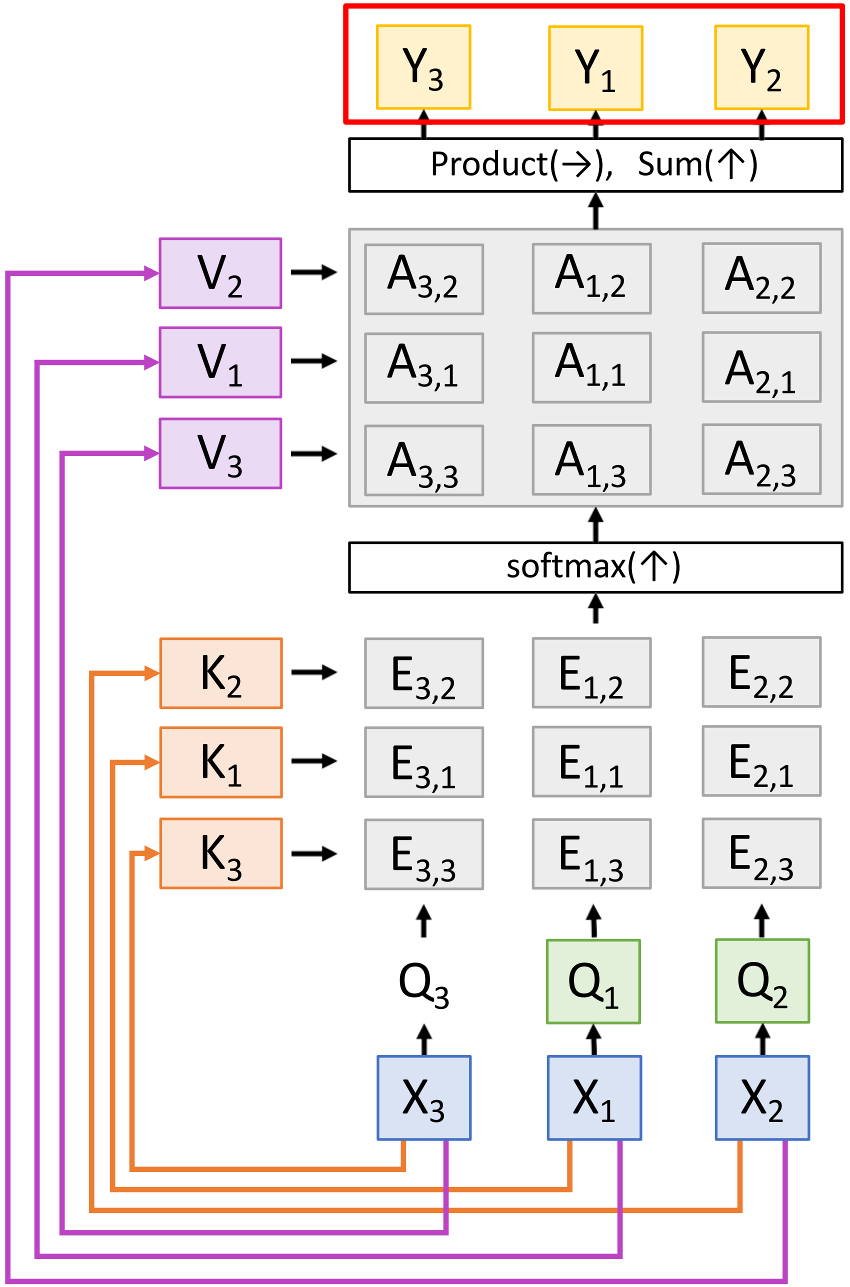
Self-Attention Layer

Consider **permuting** the input vectors:

Outputs will be the same, but permuted

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: $E = \mathbf{QK}^T$ (Shape: $N_X \times N_X$) $E_{i,j} = \mathbf{Q_i} \cdot \mathbf{K_j} / \mathrm{sqrt}(D_Q)$
Attention weights: $A = \mathrm{softmax}(E, \mathrm{dim}=1)$ (Shape: $N_X \times N_X$)
Output vectors: $Y = A\mathbf{V}$ (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 x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

Computation:

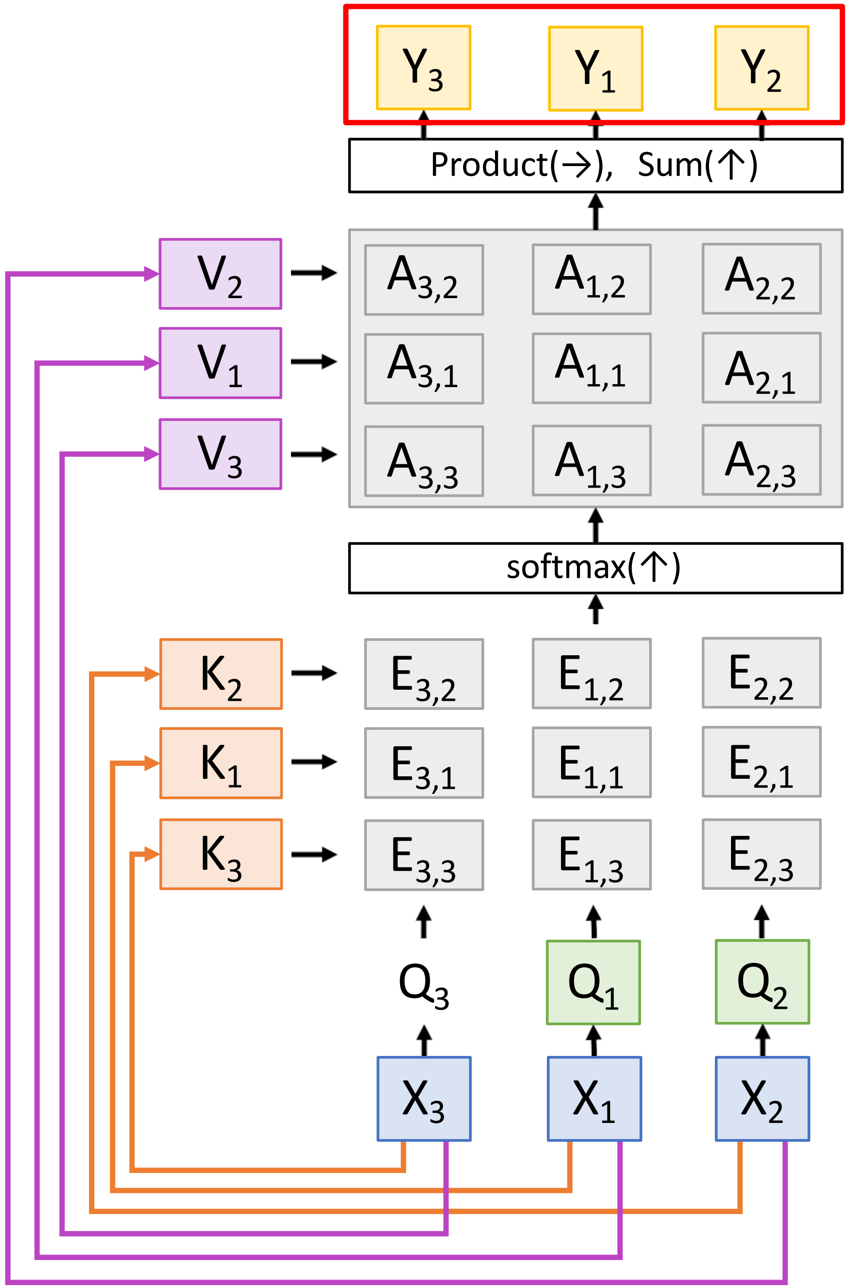
Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{QK}^T$ (Shape: N_X x N_X) $E_{i,j} = \mathbf{Q_i} \cdot \mathbf{K_j} / \text{sqrt}(D_Q)$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: N_X x N_X)
Output vectors: $Y = A\mathbf{V}$ (Shape: N_X x 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

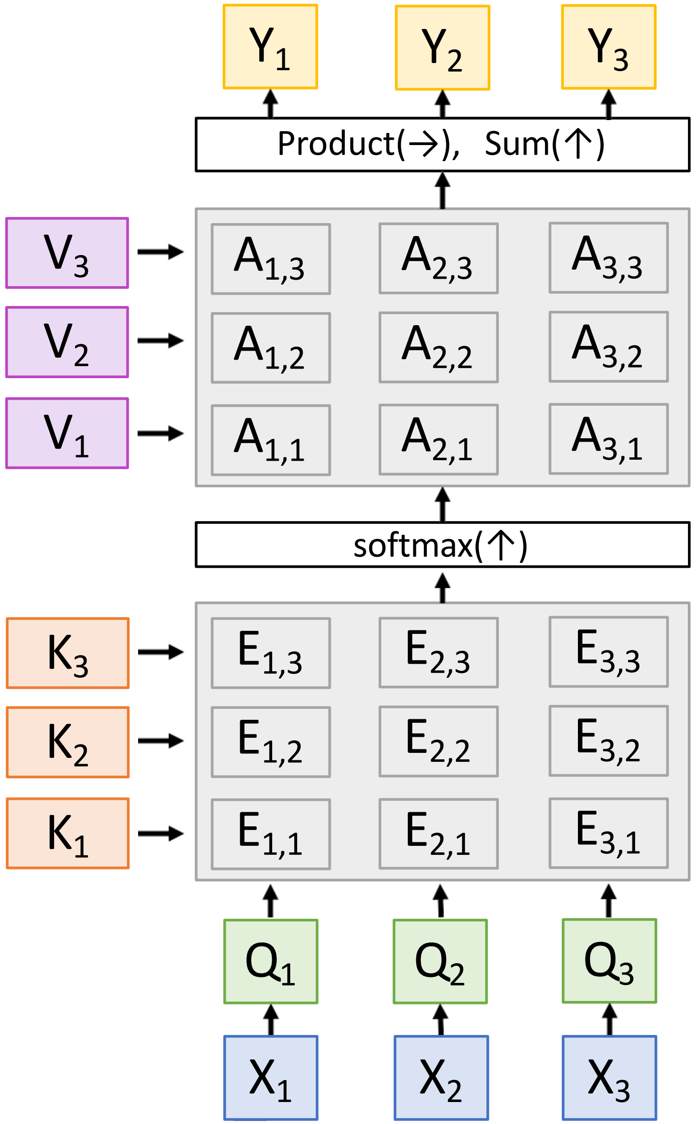


Self-Attention Layer

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

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: W_K (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)
Query matrix: W_Q (Shape: D_X x D_Q)

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value Vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T$ (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j / \mathrm{sqrt}(D_Q)$
Attention weights: $A = \mathrm{softmax}(E, dim=1)$ (Shape: N_X x N_X)
Output vectors: $Y = AV$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$



Self-Attention Layer

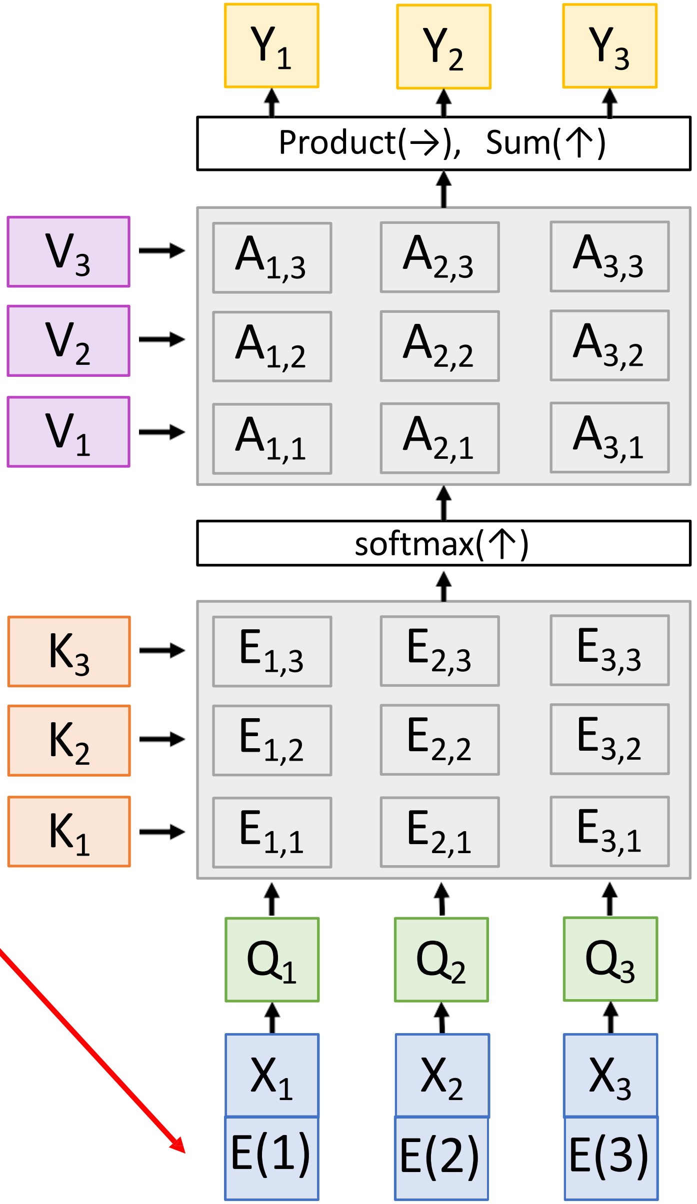
Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: **W_K** (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)
Query matrix: **W_Q** (Shape: D_X x D_Q)

Computation:
Query vectors: **Q** = **XW_Q**
Key vectors: **K** = **XW_K** (Shape: N_X x D_Q)
Value Vectors: **V** = **XW_V** (Shape: N_X x D_V)
Similarities: E = **QK^T** (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j / \mathrm{sqrt}(D_Q)$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: Y = A**V** (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$

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

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

E can be learned lookup table, or fixed function

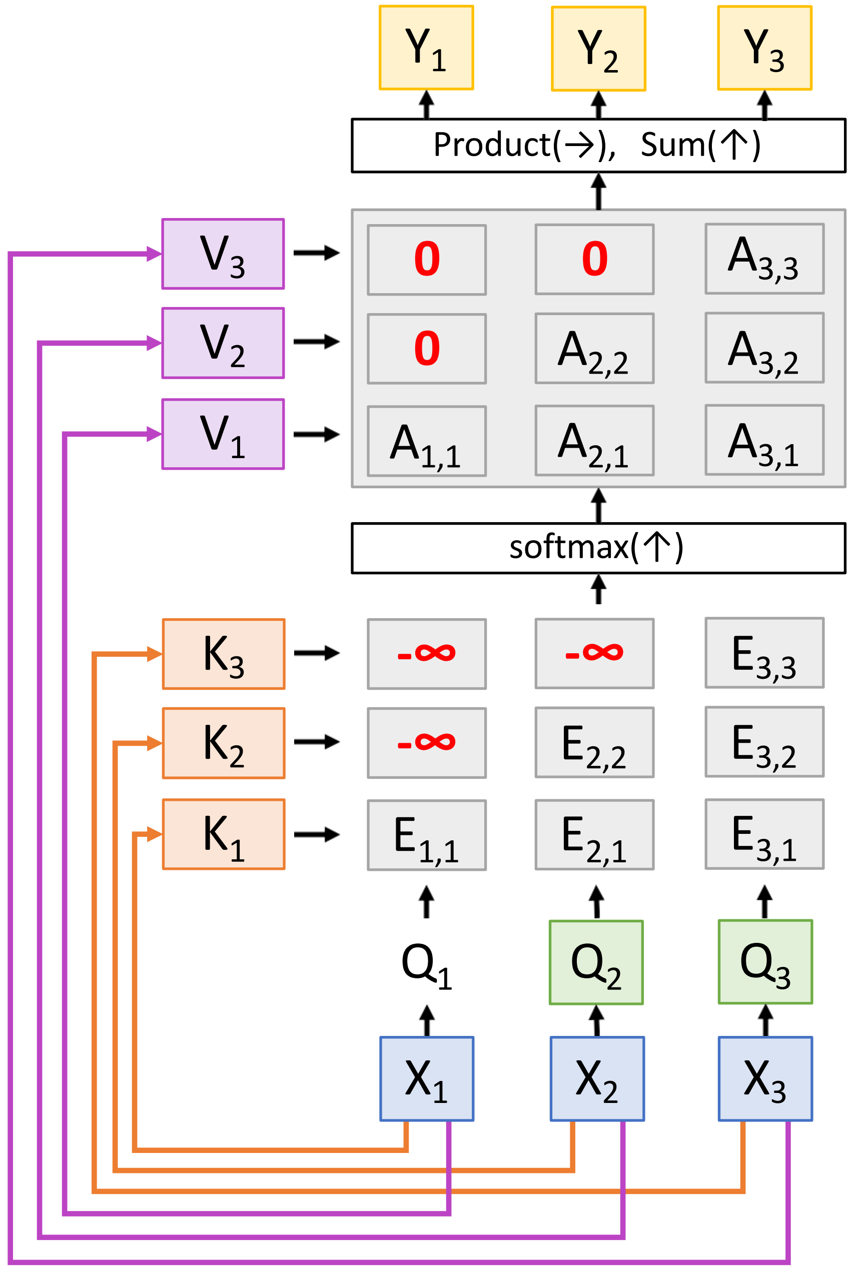


Masked Self-Attention Layer

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

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

Computation:
Query vectors: **Q** = $\mathbf{XW_Q}$
Key vectors: **K** = $\mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: **V** = $\mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: E = $\mathbf{QK}^T$ (Shape: N_X x N_X) $E_{i,j} = \mathbf{Q_i} \cdot \mathbf{K_j} / \text{sqrt}(D_Q)$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X) **Output vectors**: Y = A**V** (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} \mathbf{V_j}$

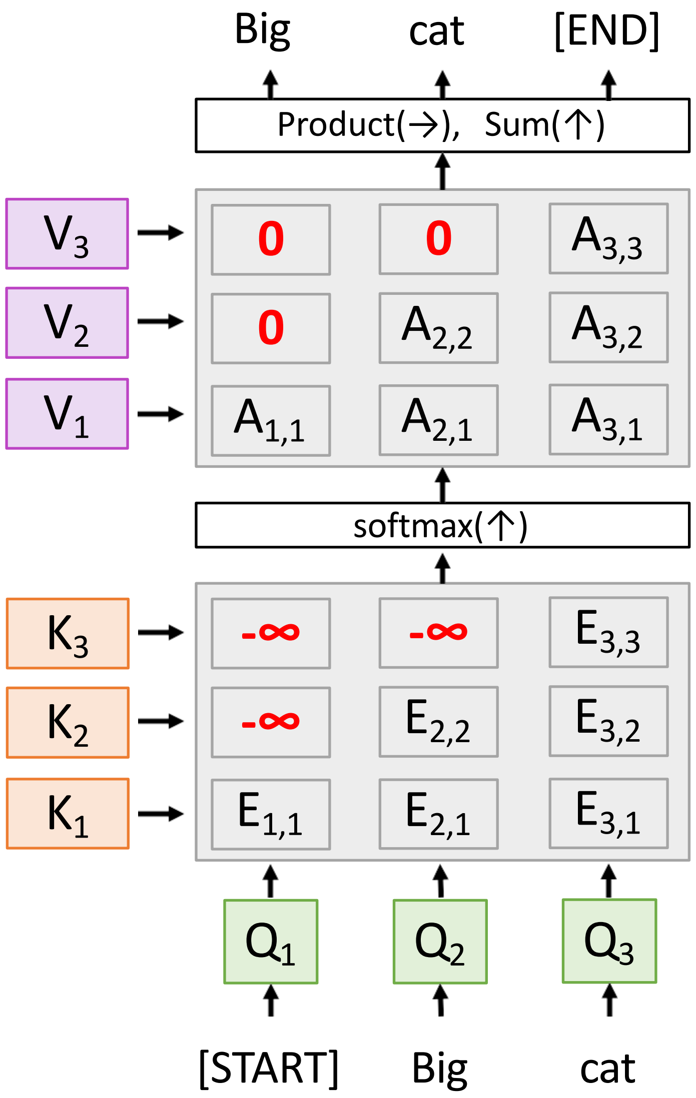


Masked Self-Attention Layer

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

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

Computation:
Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{QK}^T$ (Shape: N_X x N_X) $E_{i,j} = \mathbf{Q_i} \cdot \mathbf{K_j} / \mathrm{sqrt}(D_Q)$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X) **Output vectors**: $Y = A\mathbf{V}$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$



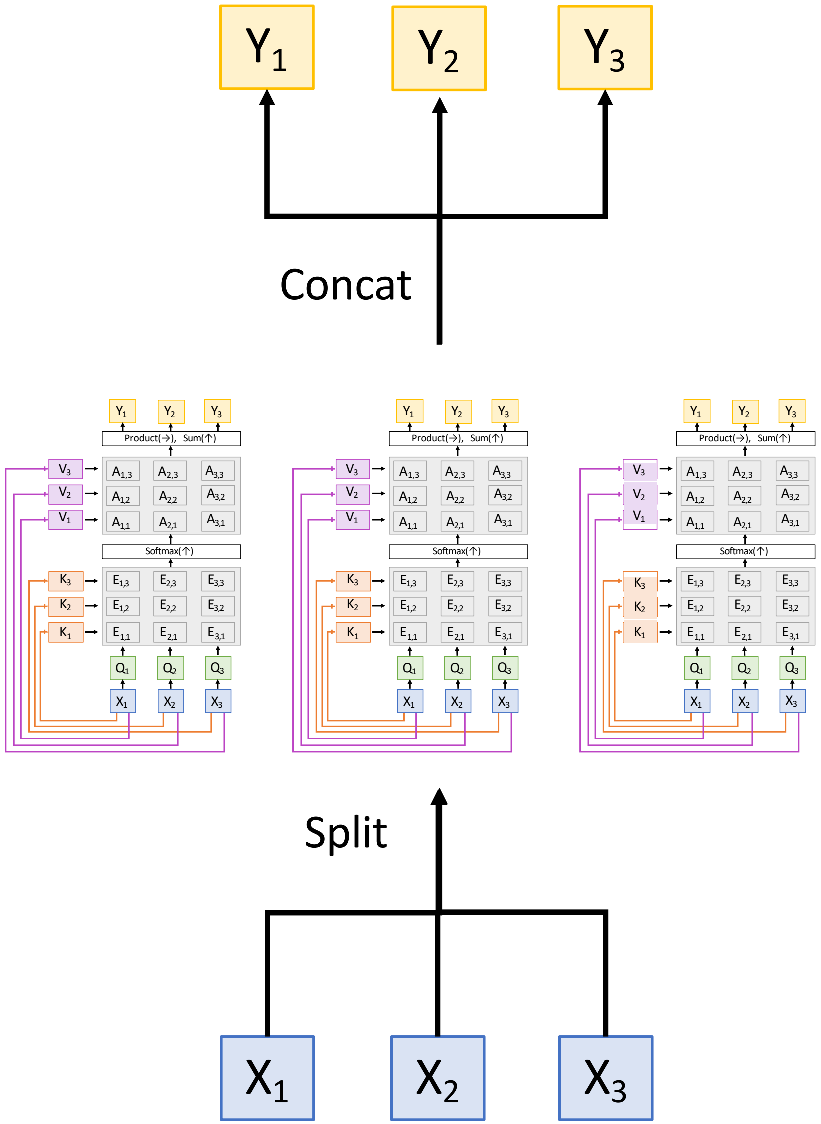
Multihead Self-Attention Layer

Use H independent
“attention Heads” in parallel

Key matrix: W_K (Shape: D_X x D_Q)
Input vectors: X (Shape: N_X x D_X)
Query matrix: W_Q (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)

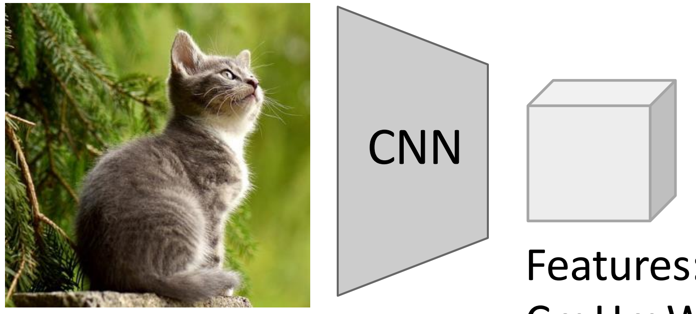
Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value Vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T$ (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j / \text{sqrt}(D_Q)$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: N_X x N_X)
Output vectors: $Y = AV$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$

Hyperparameters:
Query dimension D_Q
Number of heads H



Example: CNN with Self-A,en.on

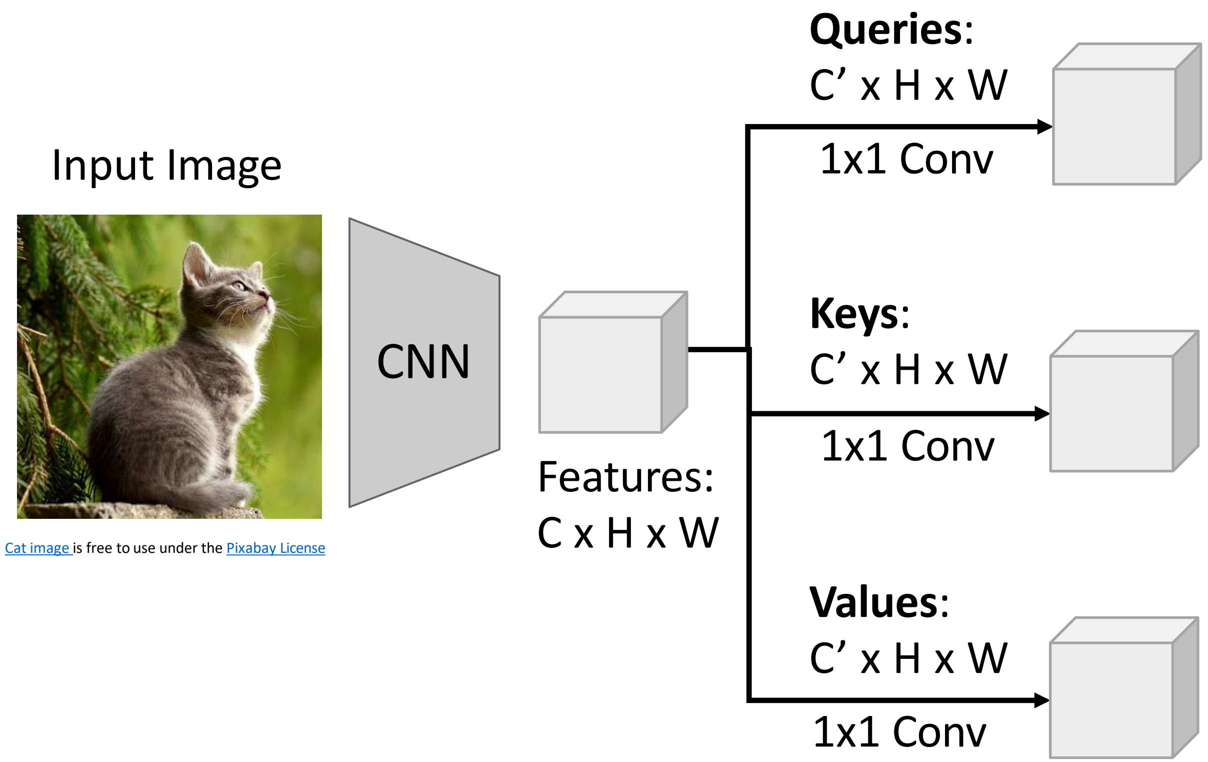
Input Image



Features:
C x H x W

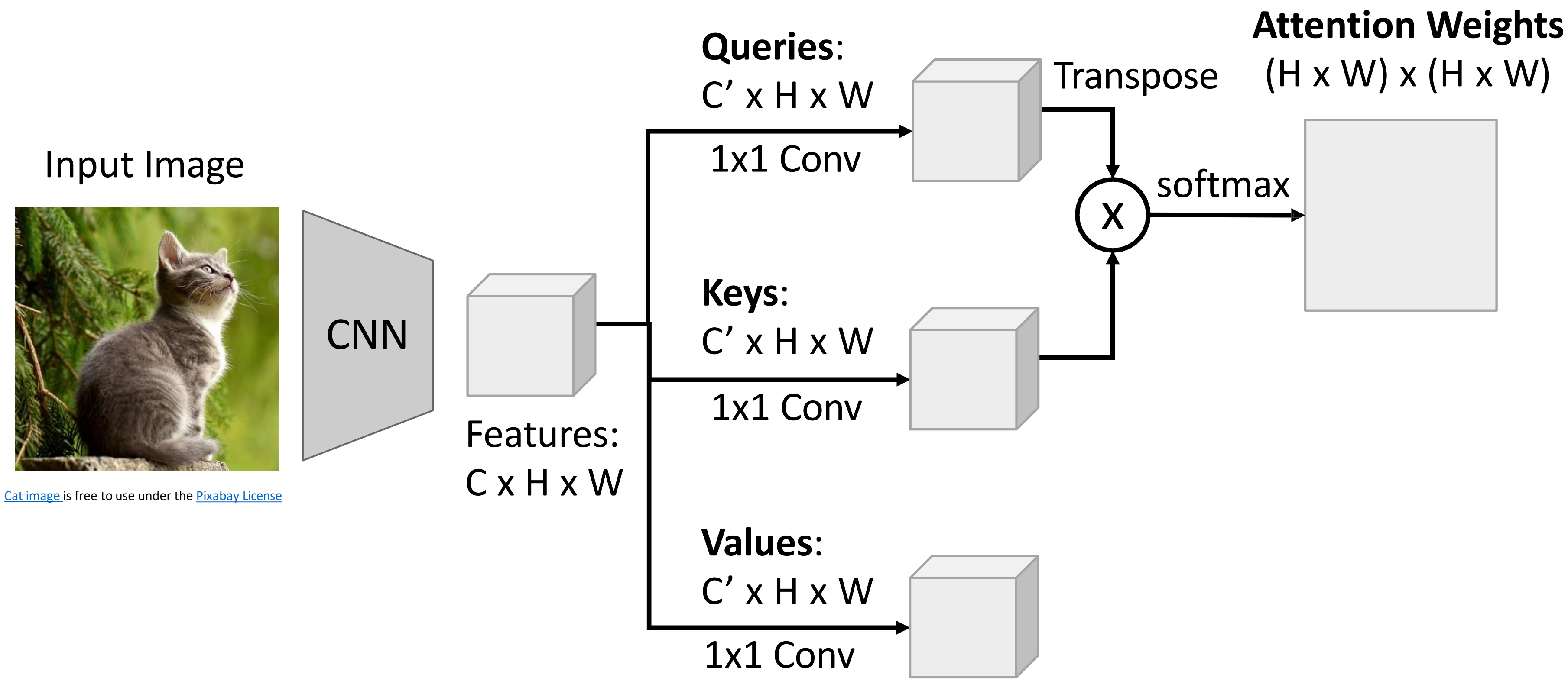
Zhang et al, "Self-attention Generative Adversarial Networks", ICML 2018

Example: CNN with Self-Attention



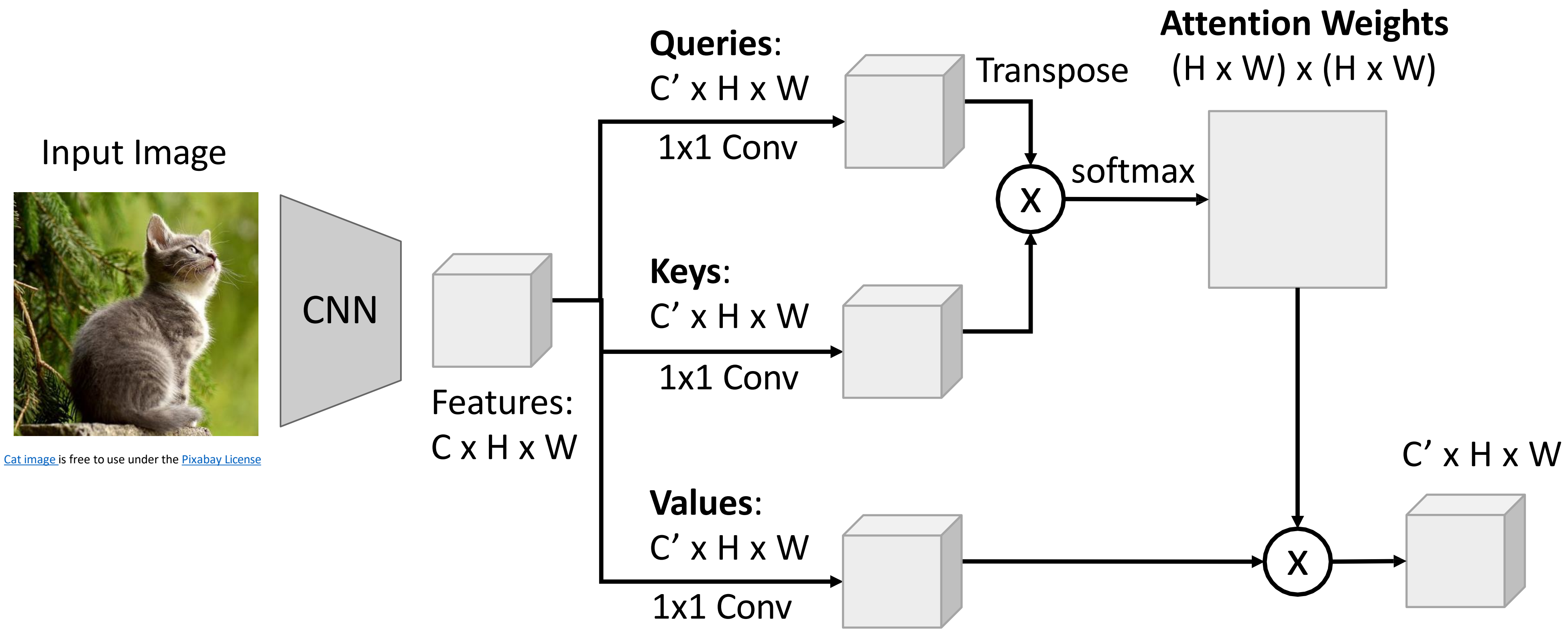
Zhang et al, "Self-attention Genera$ve Adversarial Networks", ICML 2018

Example: CNN with Self-A,en.on



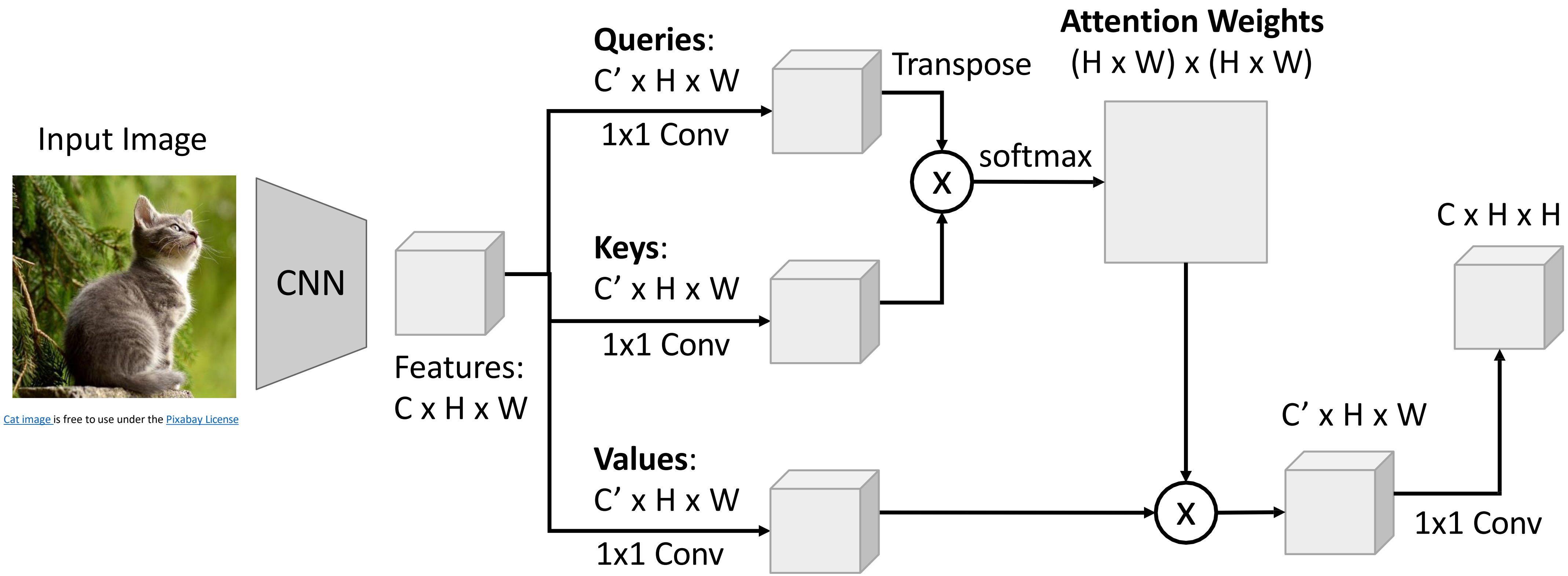
Zhang et al, "Self-attention Genera$ve Adversarial Networks", ICML 2018

Example: CNN with Self-A,en.on



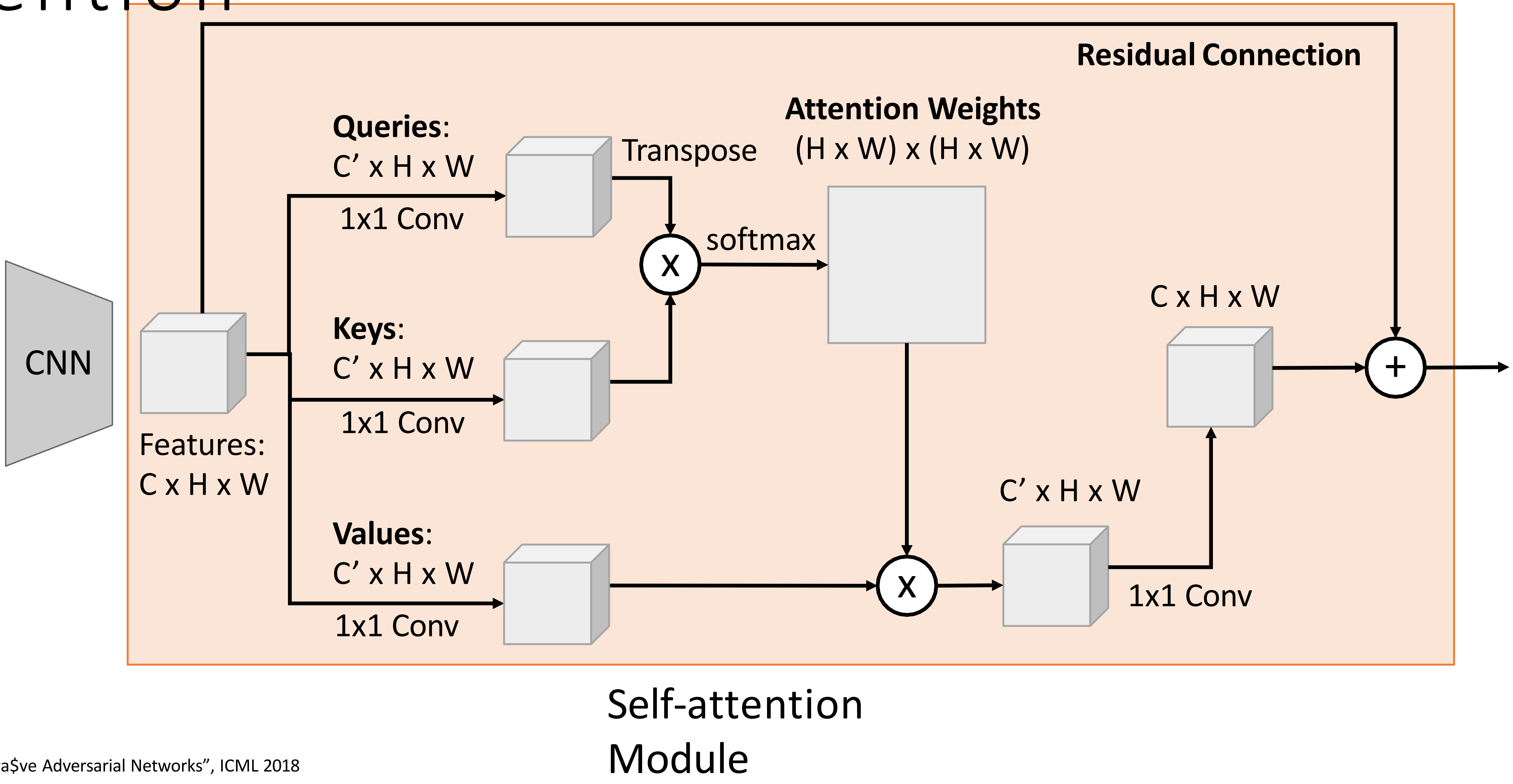
Zhang et al, "Self-attention Genera$ve Adversarial Networks", ICML 2018

Example: CNN with Self-A,en.on



Zhang et al, “Self-attention Genera$ve Adversarial Networks”, ICML 2018

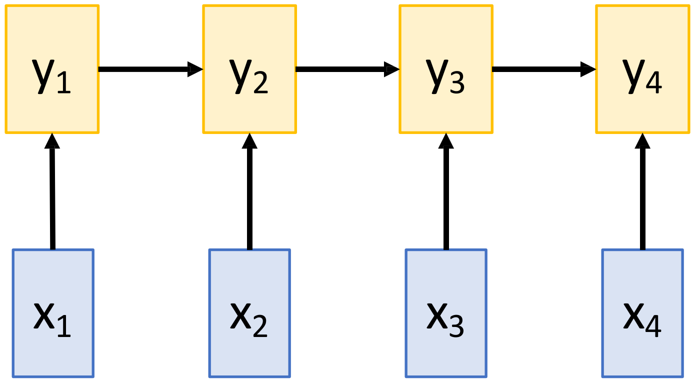
Example: CNN with Self-Attention



Zhang et al, "Self-attention Genera$ve Adversarial Networks", ICML 2018

Three Ways of Processing Sequences

Recurrent Neural Network



Works on **Ordered Sequences**

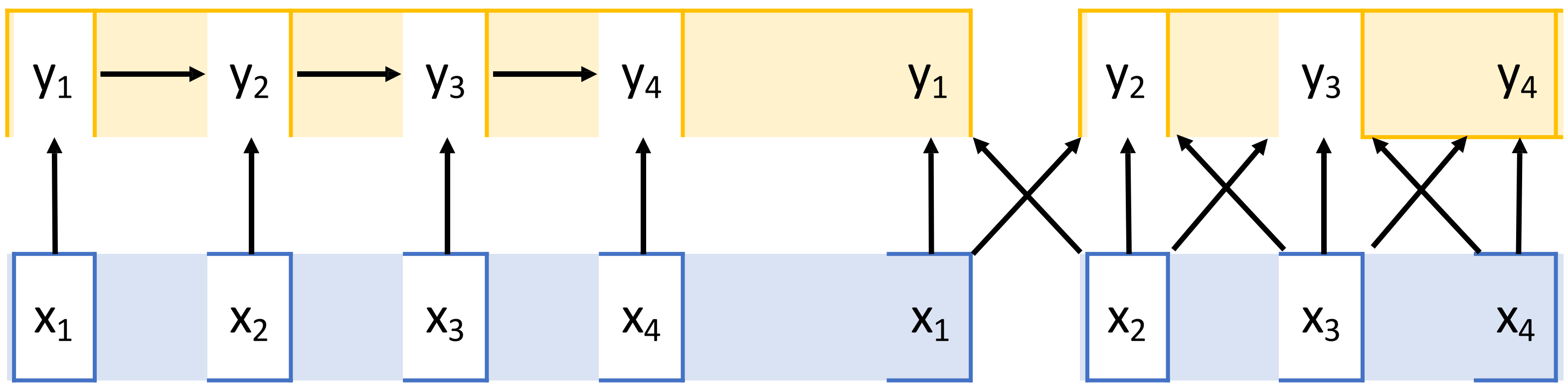
(+) Good at long sequences: Aher one RNN layer, h_T "sees" the whole sequence

(-) Not parallelizable: need to compute hidden states sequen0ally

Three Ways of Processing Sequences

Recurrent Neural Network

1D Convolution



Works on **Ordered Sequences**

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

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

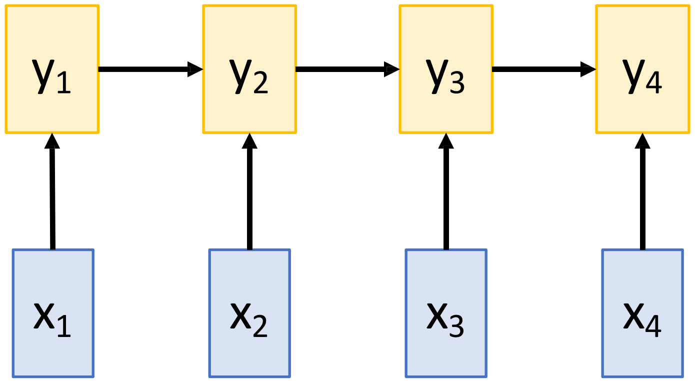
Works on **Multidimensional Grids**

(-) 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

Three Ways of Processing Sequences

Recurrent Neural Network

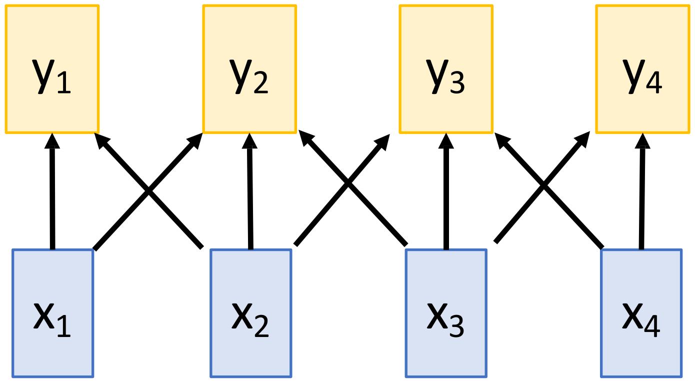


Works on **Ordered Sequences**

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

(-) Not parallelizable: need to compute hidden states sequen0ally

1D Convolu$on

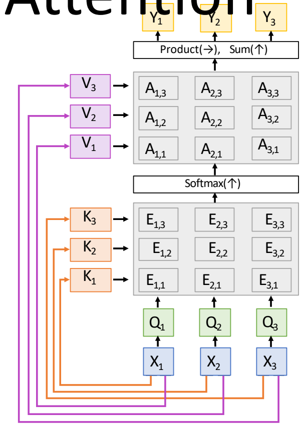


Works on **Mul0dimensional Grids**

(-) 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: aher 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: Aher one RNN layer, h_T "sees" the whole sequence

(-) Not parallelizable: need to compute hidden states sequen0ally

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

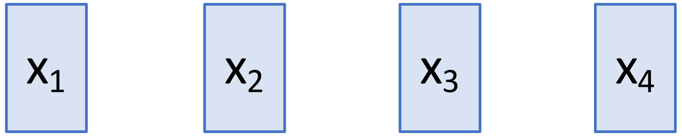
Works on **Sets of Vectors**

(-) Good at long sequences: aher one self-Attention layer, each output "sees" all inputs!

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

(-) Very memory intensive

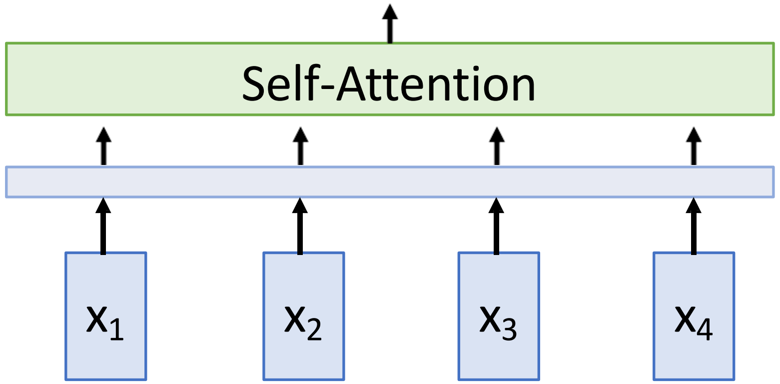
The Transformer



Vaswani et al, “attention is all you need”, NeurIPS 2017

The Transformer

All vectors interact with each other

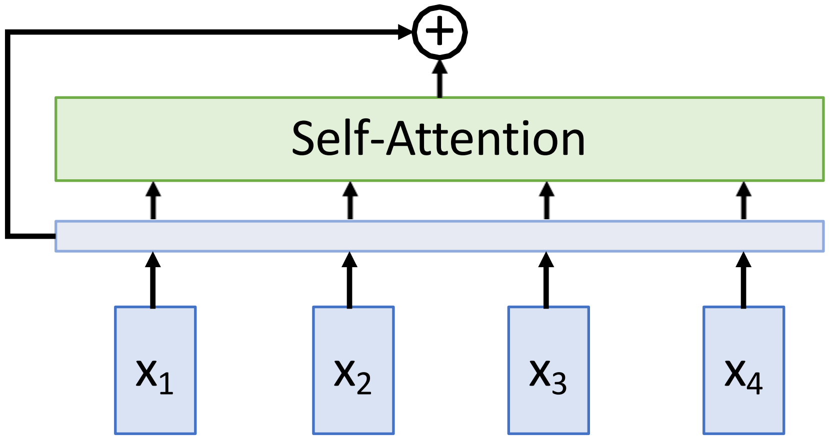


Vaswani et al, "attention is all you need", NeurIPS 2017

The Transformer

Residual connection

All vectors interact with each other



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

The Transformer

Recall **Layer Normalization**:

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

scale: γ (Shape: D)

shift: β (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)

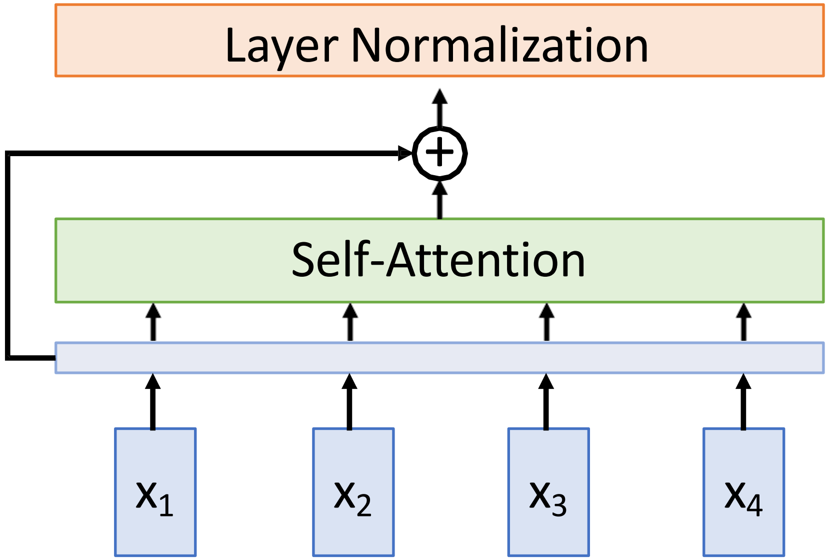
$z_i = (h_i - \mu_i) / \sigma_i$

$y_i = \gamma * z_i + \beta$

Ba et al, 2016

Residual connection

All vectors interact with each other



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

The Transformer

Recall **Layer Normalization**:

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

scale: γ (Shape: D)

shift: β (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)

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

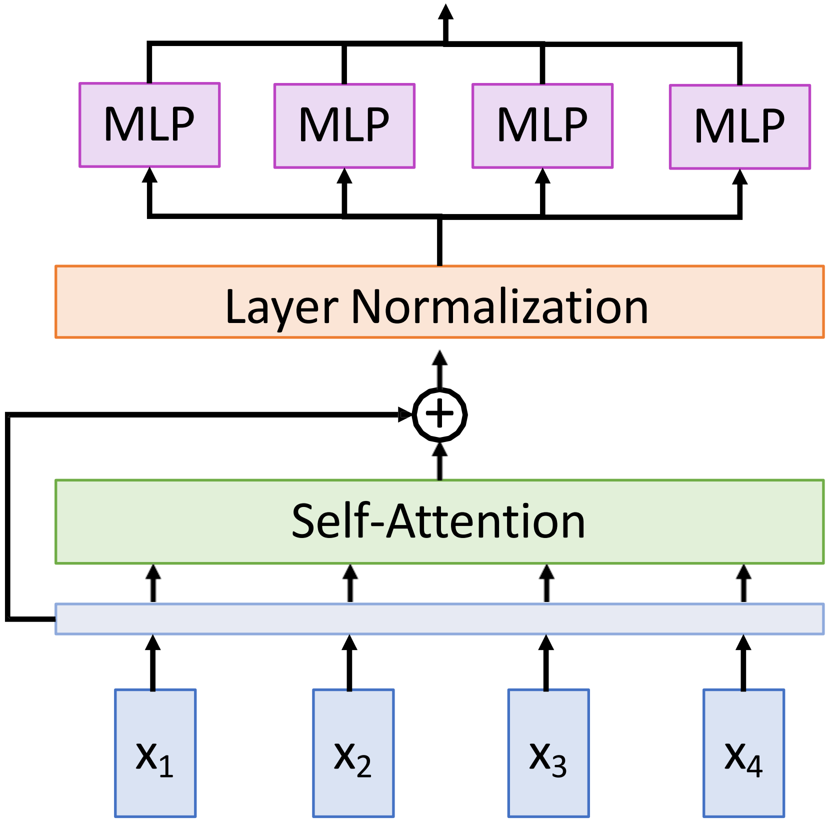
$y_i = \gamma * z_i + \beta$

Ba et al, 2016

MLP independently on each vector

Residual connection

All vectors interact with each other



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

The Transformer

Recall **Layer Normalization**:

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

scale: γ (Shape: D)

shift: β (Shape: D)

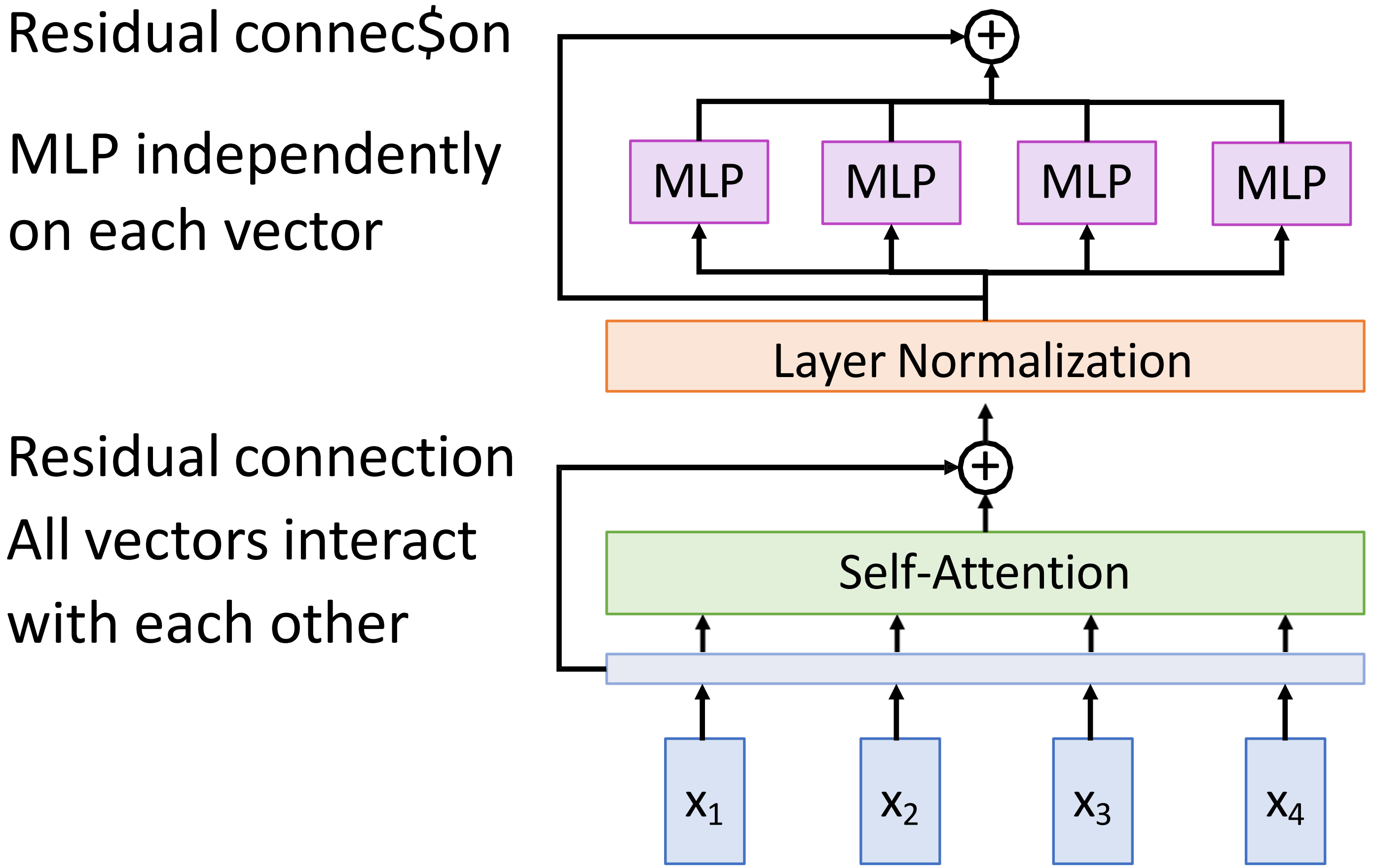
$\mu_i = (1/D)\sum_j h_{i,j}$ (scalar)

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

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

$y_i = \gamma * z_i + \beta$

Ba et al, 2016



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

The Transformer

Recall **Layer Normalization**:

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

scale: γ (Shape: D)

shift: β (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)

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

$y_i = \gamma * z_i + \beta$

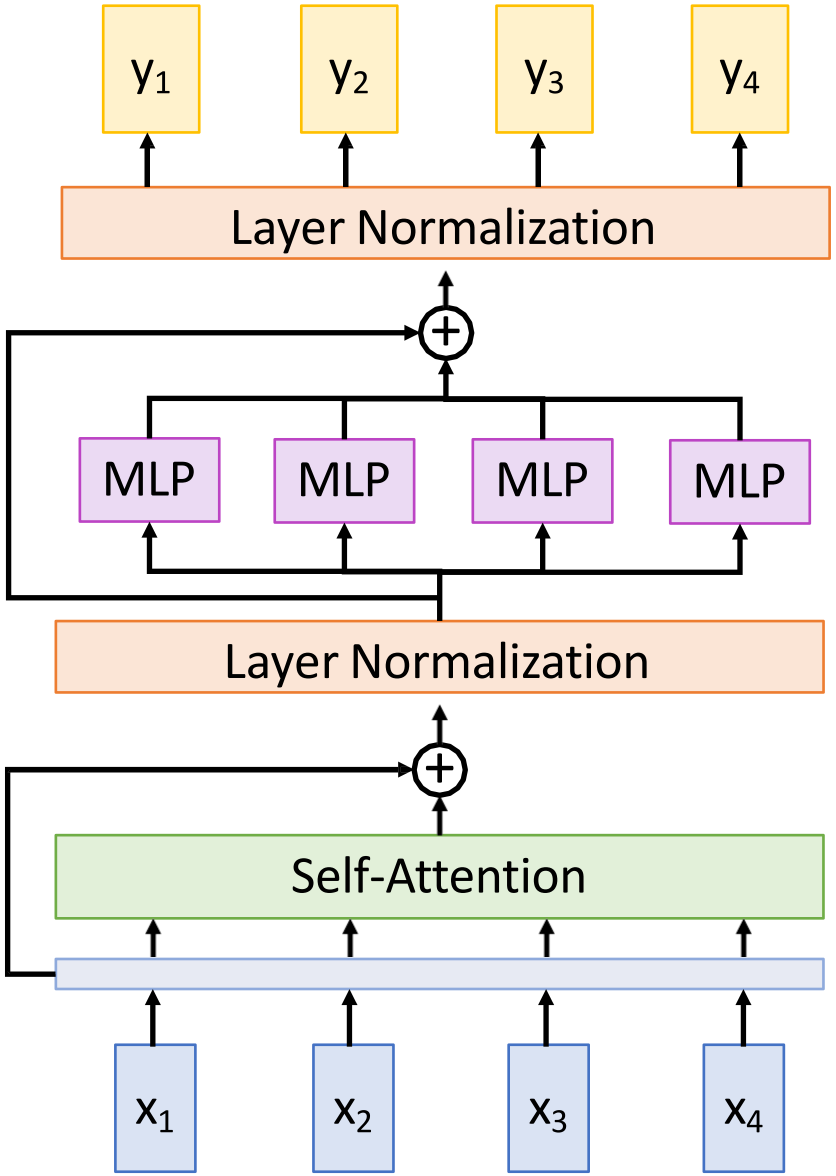
Ba et al, 2016

Residual connec$on

MLP independently on each vector

Residual connection

All vectors interact with each other



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

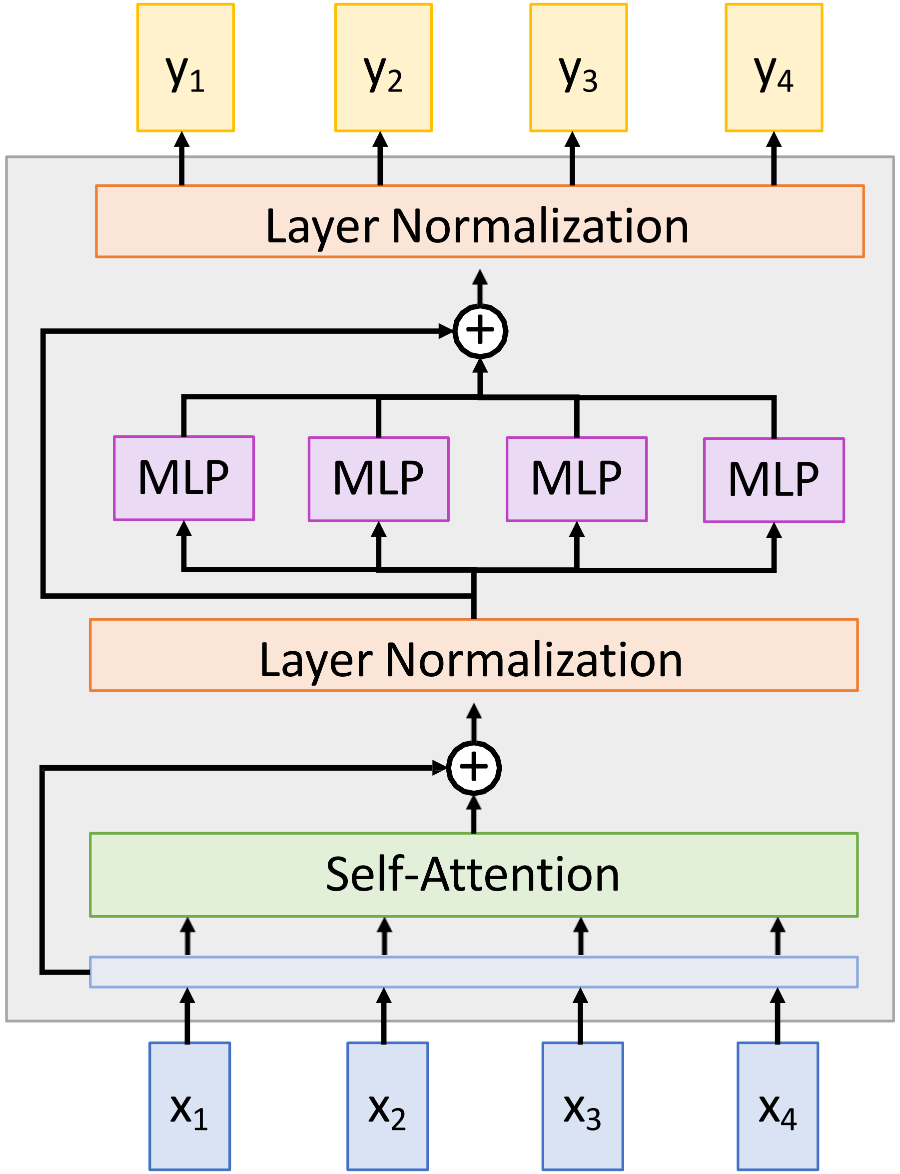
The Transformer

Transformer Block:
Input: Set of vectors x
Output: Set of vectors y

Self-attention is the only interac$on between vectors!

Layer norm and MLP work independently per vector

Highly scalable, highly parallelizable



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

The Transformer

- **<u>Transformer Block</u>:**
- **Input**: Set of vectors x
- **Output**: Set of vectors y

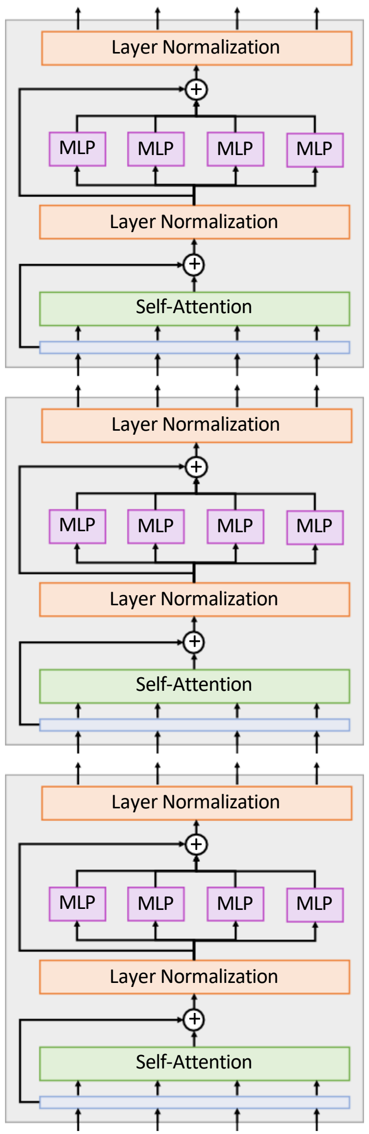
- Self-attention is the only interaction between vectors!

- 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



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

The Transformer: Transfer Learning

"ImageNet Moment for Natural Language Processing"

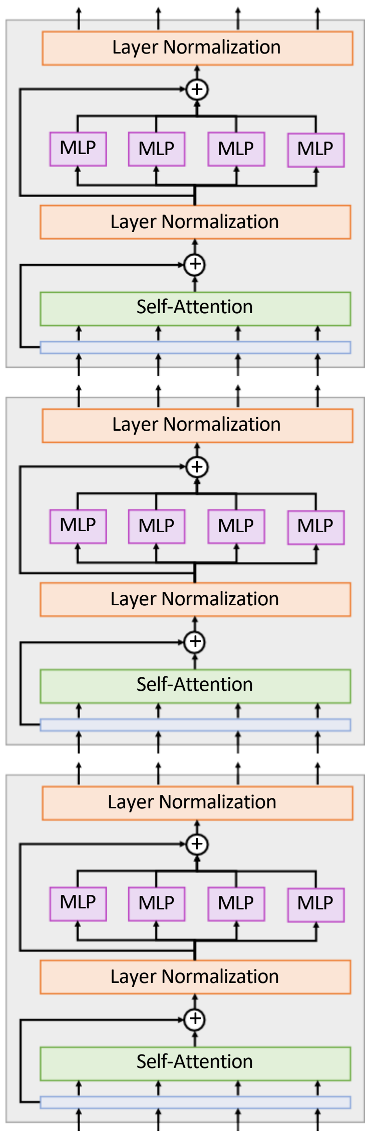
<u>**Pretraining**</u>:

Download a lot of text from the internet

Train a giant Transformer model for language modeling

<u>**Finetuning:**</u>

Fine-tune the Transformer on your own NLP task



Devlin et al, "BERT: Pre-training of Deep Bidirec$onal 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)

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	12	768	?	117M	40 GB	
GPT-2	24	1024	?	345M	40 GB	
GPT-2	36	1280	?	762M	40 GB	
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	12	768	?	117M	40 GB	
GPT-2	24	1024	?	345M	40 GB	
GPT-2	36	1280	?	762M	40 GB	
GPT-2	48	1600	?	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 Languge 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	?	117M	40 GB	
GPT-2	24	1024	?	345M	40 GB	
GPT-2	36	1280	?	762M	40 GB	
GPT-2	48	1600	?	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	$\boxed{\text{512x V100 GPU (9 days)}}$

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

PROMPT (Human-written): *In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.*

OpenAI, "Better Language Models and their Implications", 2019, https://openai.com/blog/better-language-models/

PROMPT (Human-written): *In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.*

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

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

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

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

Pérez and his friends were astonished to see the unicorn herd. These creatures could be seen from the air without having to move too much to see them – they were so close they could touch their horns.

While examining these bizarre creatures the scientists discovered that the creatures also spoke some fairly regular English. Pérez stated, "We can see, for example, that they have a common 'language,' something like a dialect or dialectic."

Dr. Pérez believes that the unicorns may have originated in Argentina, where the animals were believed to be descendants of a lost race of people who lived there before the arrival of humans in those parts of South America.

While their origins are still unclear, some believe that perhaps the creatures were created when a human and a unicorn met each other in a time before human civilization. According to Pérez, "In South America, such incidents seem to be quite common."

However, Pérez also pointed out that it is likely that the only way of knowing for sure if unicorns are indeed the descendants of a lost alien race is through DNA. "But they seem to be able to communicate in English quite well, which I believe is a sign of evolution, or at least a change in social organization," said the scientist.

OpenAI, "Better Language Models and their Implications", 2019, https://openai.com/blog/better-language-models/

PROMPT (Human-written): *In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.*
COMPLETION (Transformer-written): The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

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

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes

- Try it yourself: https://talktotransformer.com

Dr. Pérez believes that the unicorns may have originated in Argentina, where the animals were believed to be descendants of a lost race of people who lived there before the arrival of humans in those parts of South America.

While their origins are still unclear, some believe that perhaps the creatures were created when a human and a unicorn met each other in a time before human civilization. According to Pérez, "In South America, such incidents seem to be quite common."

However, Pérez also pointed out that it is likely that the only way of knowing for sure if unicorns are indeed the descendants of a lost alien race is through DNA. "But they seem to be able to communicate in English quite well, which I believe is a sign of evolution, or at least a change in social organization," said the scientist.

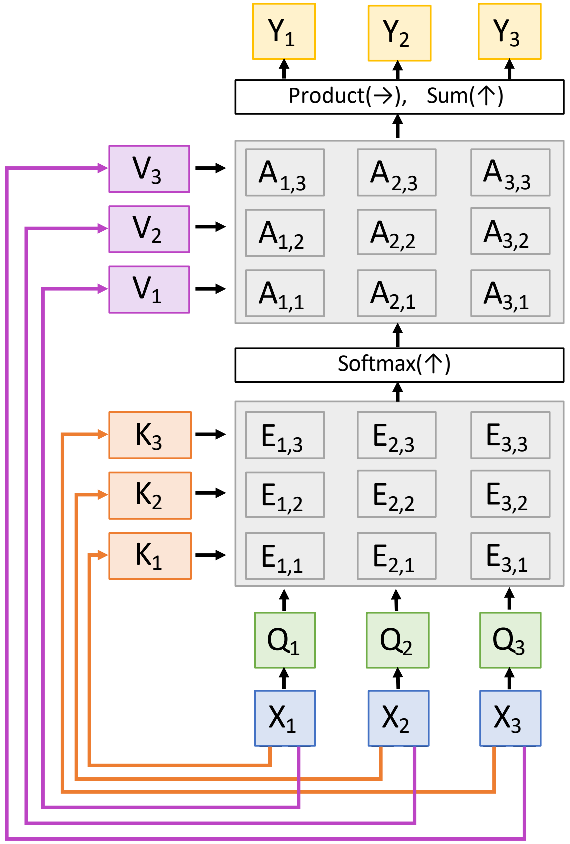
OpenAI, "Better Language Models and their Implications", 2019, https://openai.com/blog/better-language-models/

Summary

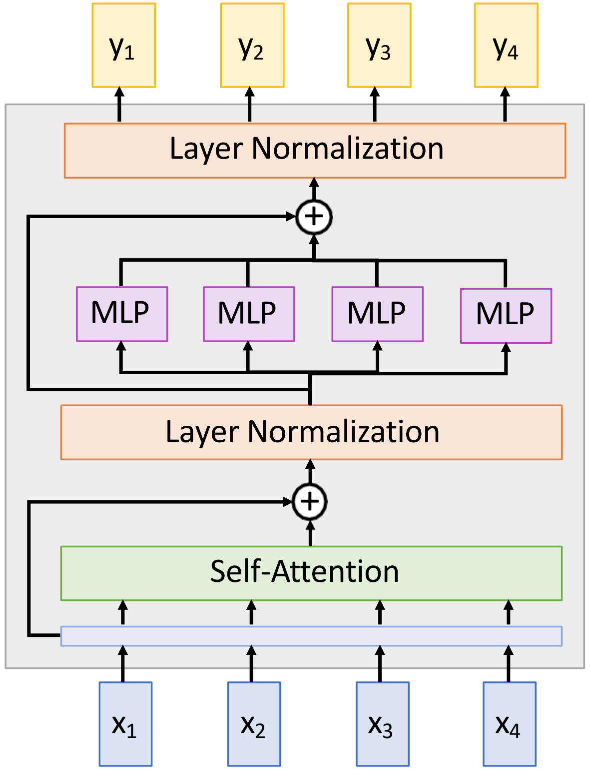
Adding **Attention** to RNN models lets them look at different parts of the input at each timestep

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



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