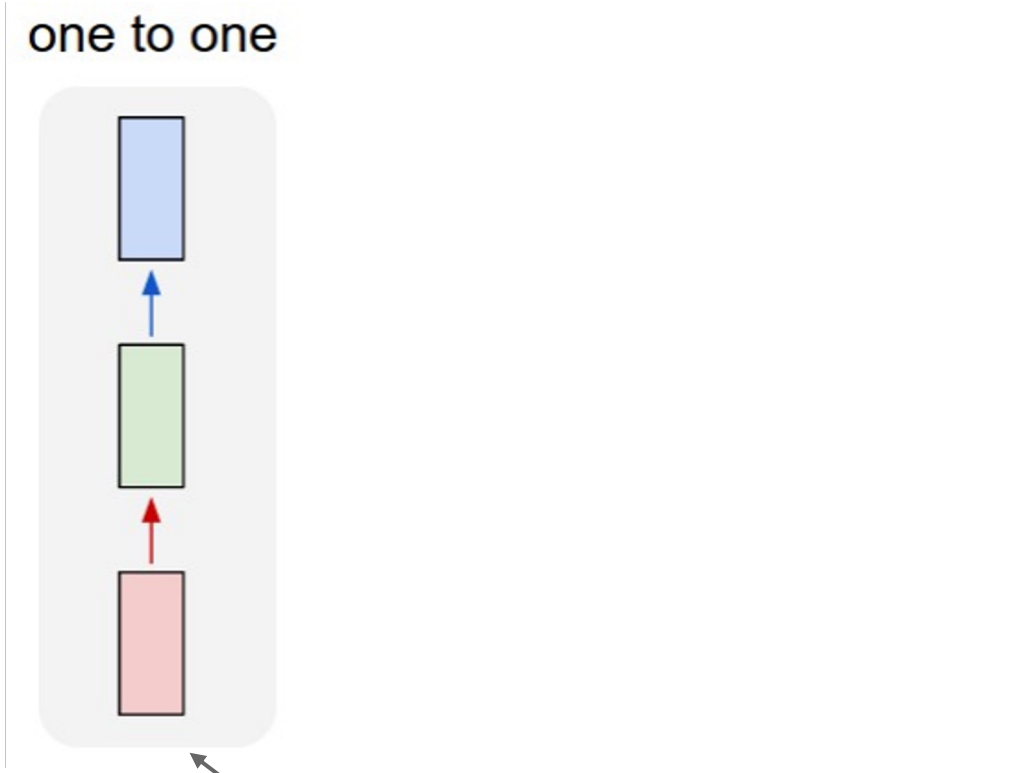
CS163: Deep Learning for Computer Vision

Lecture 11:
Recursive Neural Network (RNN)
and Attention

Announcement

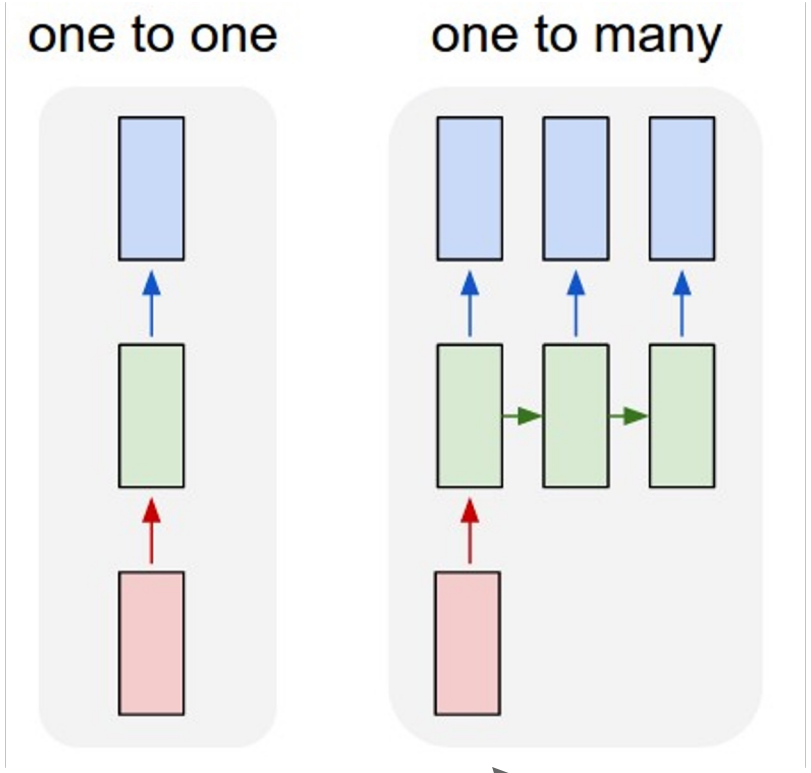
- Assignment 3 is released at https://github.com/UCLAdeepvision/CS163-Assignments-2024Fall/tree/main/Assignment3
  - Due: Nov. 24, 2024 (three weeks to go)
  - Build a Vision Transformer (ViT) from scratch.
  - Build a Semantic Segmentation model using a ViT encoder.
- Midway presentation: Please follow the 20min time limits when preparing your midway presentations

So far: “Feedforward” Neural Networks



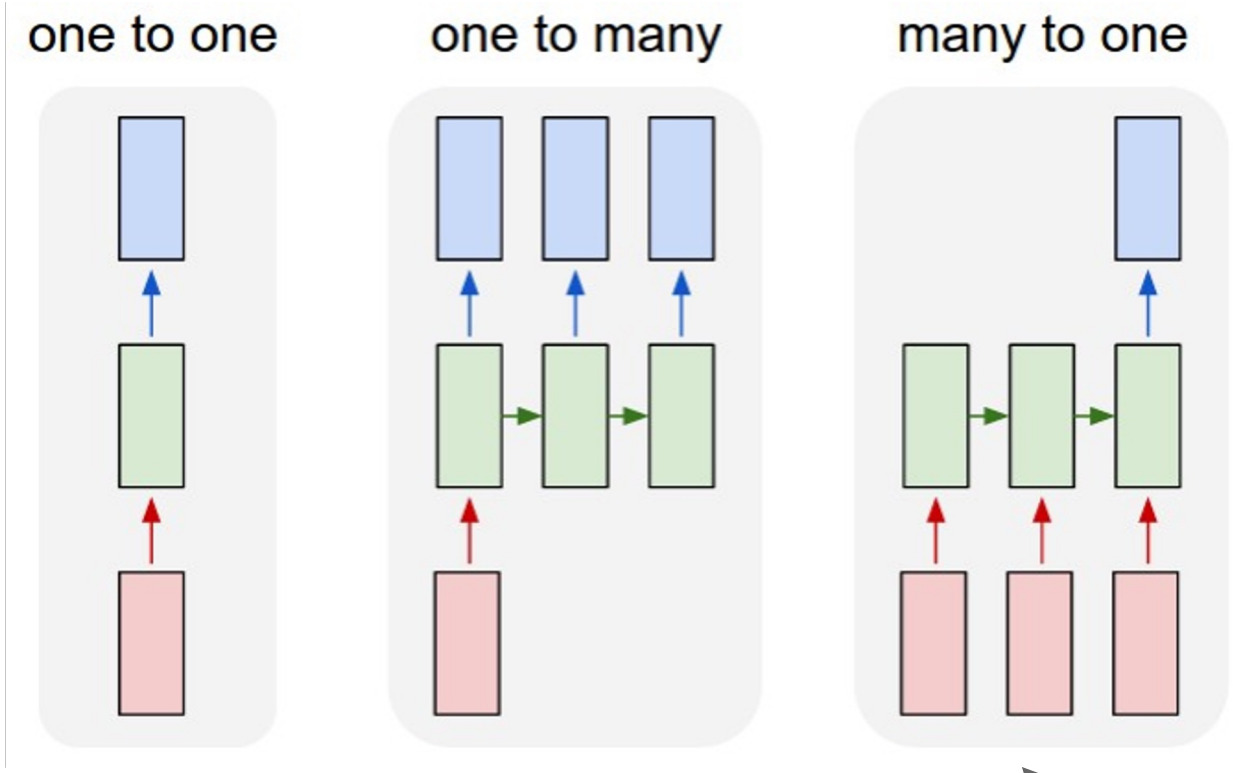
e.g. **Image classification**
Image -> Label

Recurrent Neural Networks: Process Sequences



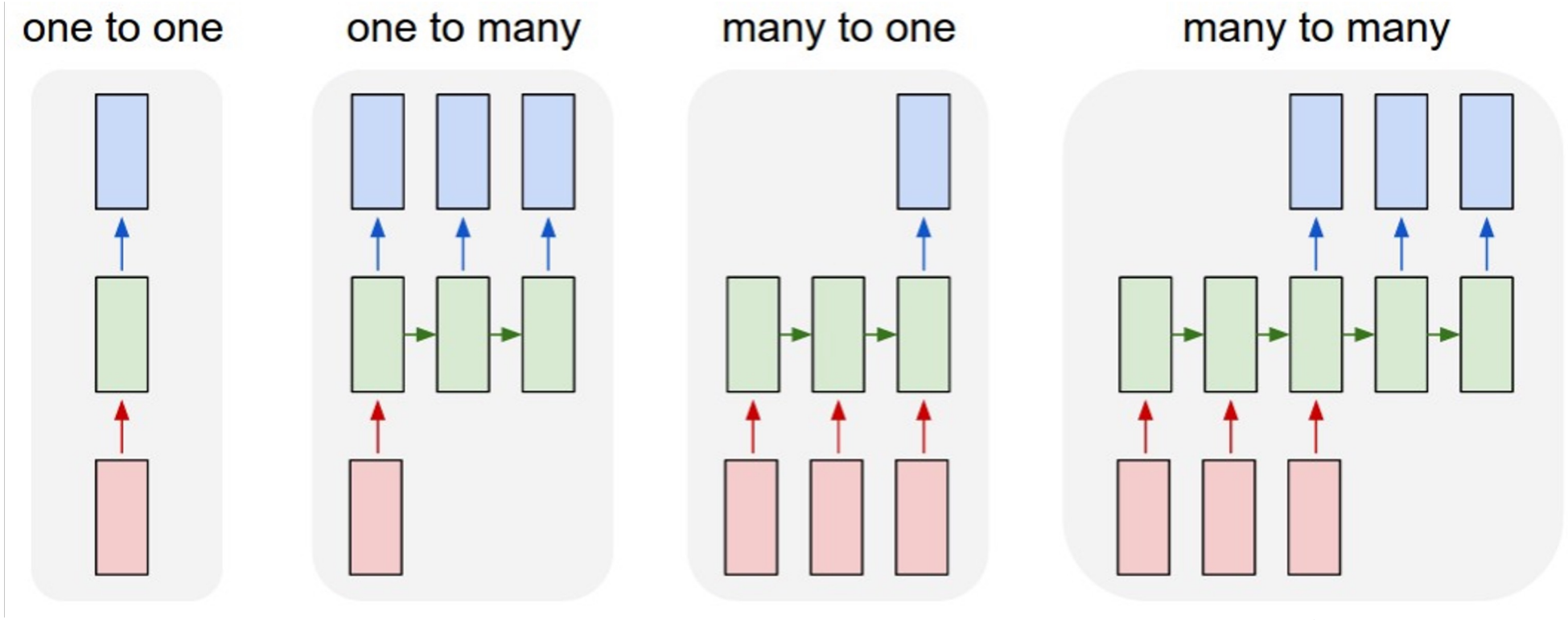
e.g. **Image Captioning**:
Image -> sequence of words

Recurrent Neural Networks: Process Sequences



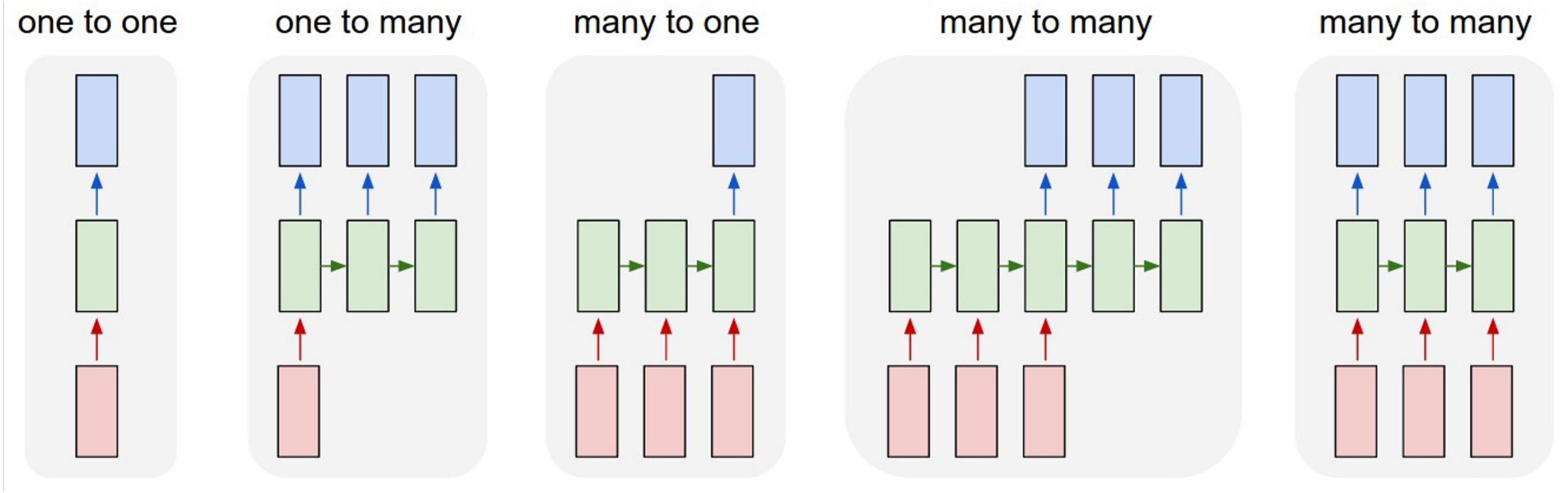
e.g. **Video classification:**
Sequence of images -> label

Recurrent Neural Networks: Process Sequences



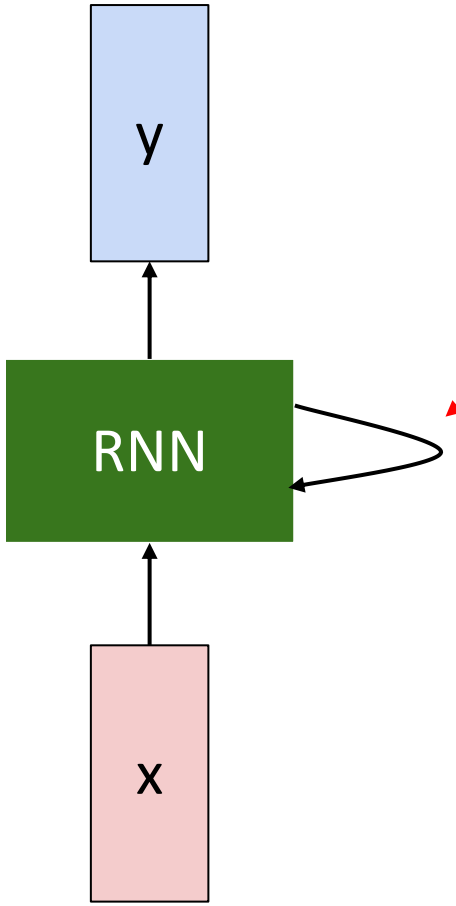
e.g. **Machine Translation:**
Sequence of words -> Sequence of words

Recurrent Neural Networks: Process Sequences



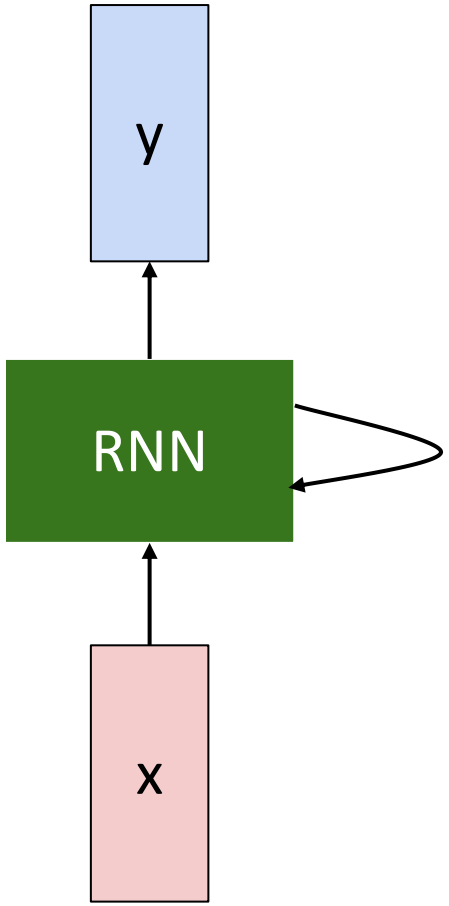
e.g. **Per-frame video classification:**
Sequence of images -> Sequence of labels

Recurrent Neural Networks



Key idea: RNNs have an "internal state" that is updated as a sequence is processed

Recurrent Neural Networks



We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

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

new state

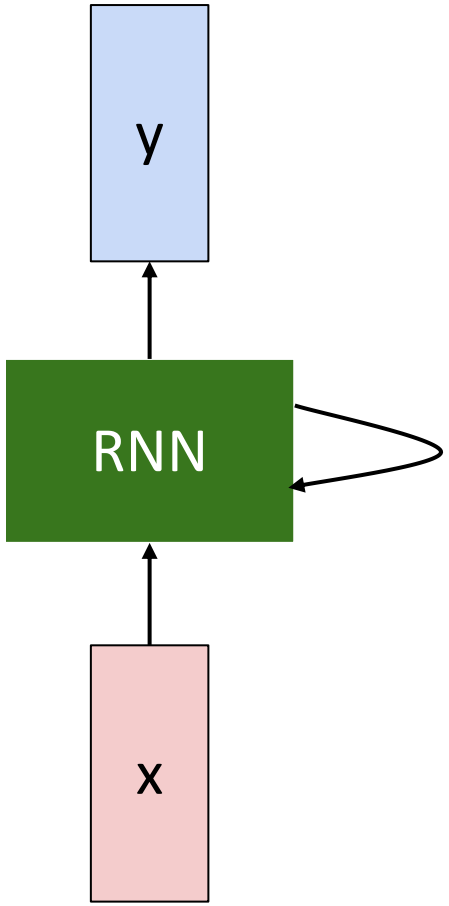
some function with parameters W

old state

input vector at some time step

Recurrent Neural Networks

Notice: the same function and the same set of parameters are used at every time step.



We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

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

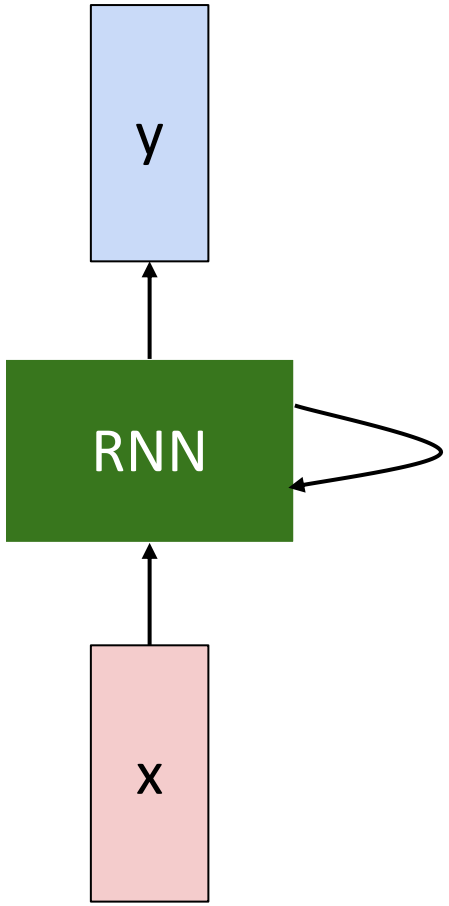
new state

some function with parameters W

old state

input vector at some time step

(Vanilla) Recurrent Neural Networks



The state consists of a single *"hidden"* vector **h**:

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

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

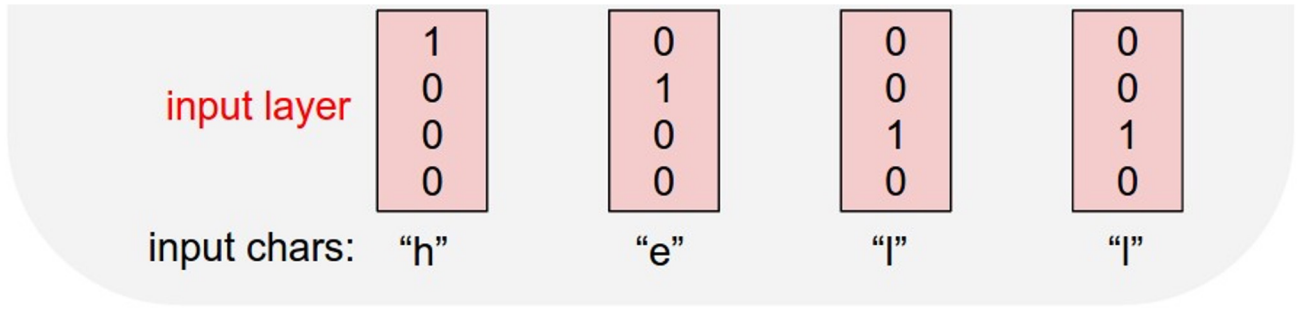
$$y_t = W_{hy}h_t + b_y$$

Example: Language Modeling

Given characters 1, 2, …, t-1,
model predicts character t

Training sequence: ”hello”

Vocabulary: [h, e, l, o]



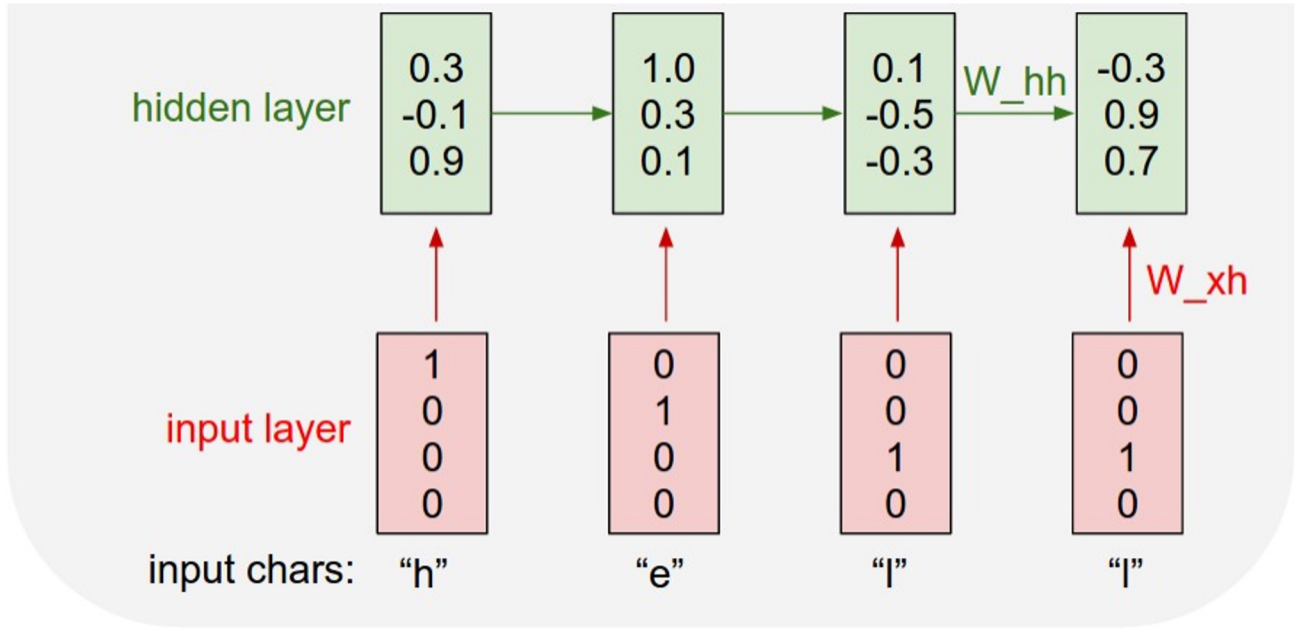
Example: Language Modeling

Given characters 1, 2, …, t-1,
model predicts character t

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

Training sequence: ”hello”

Vocabulary: [h, e, l, o]



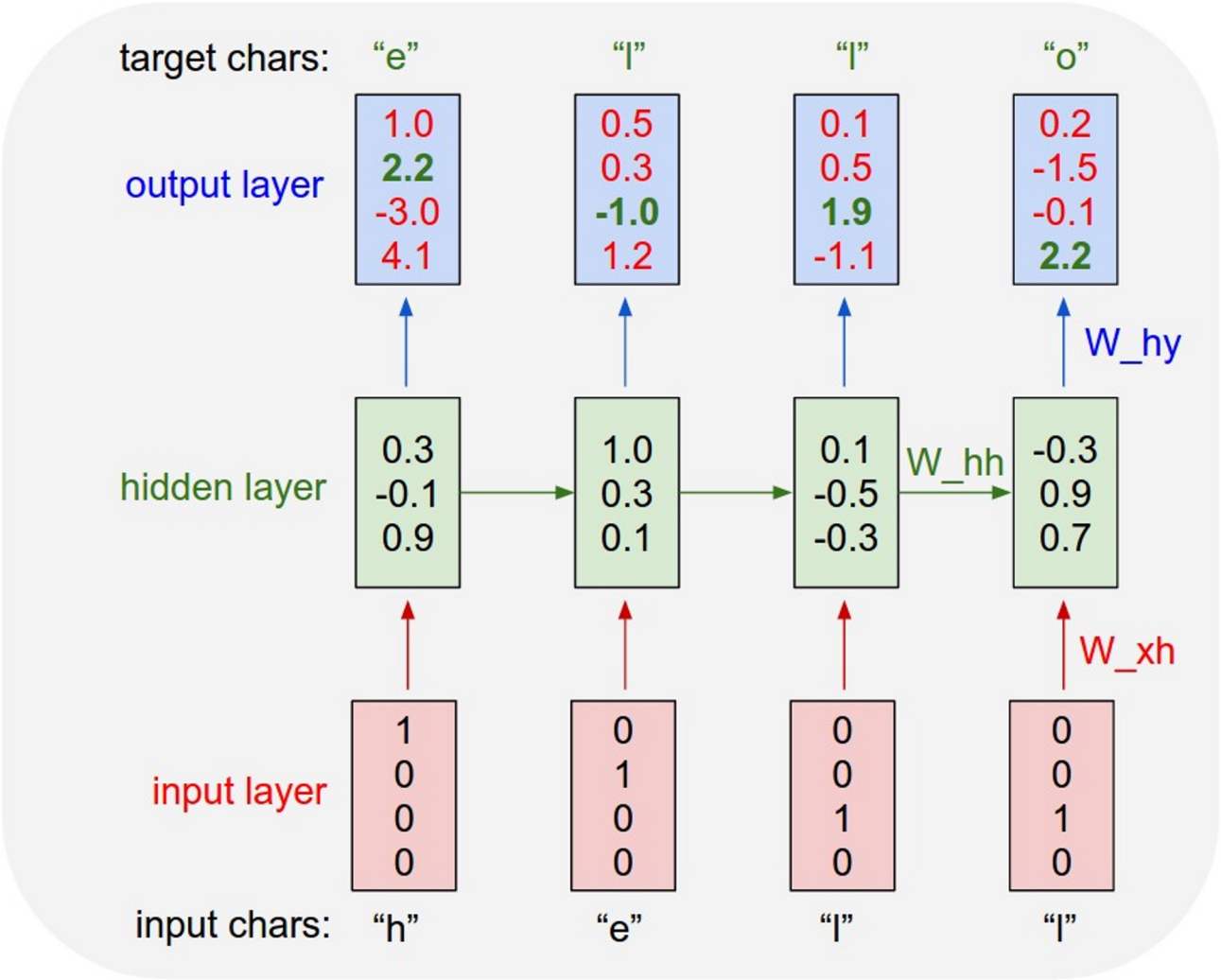
Example: Language Modeling

Given characters 1, 2, …, t-1, model predicts character t

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

Training sequence: ”hello”

Vocabulary: [h, e, l, o]



Example: Language Modeling

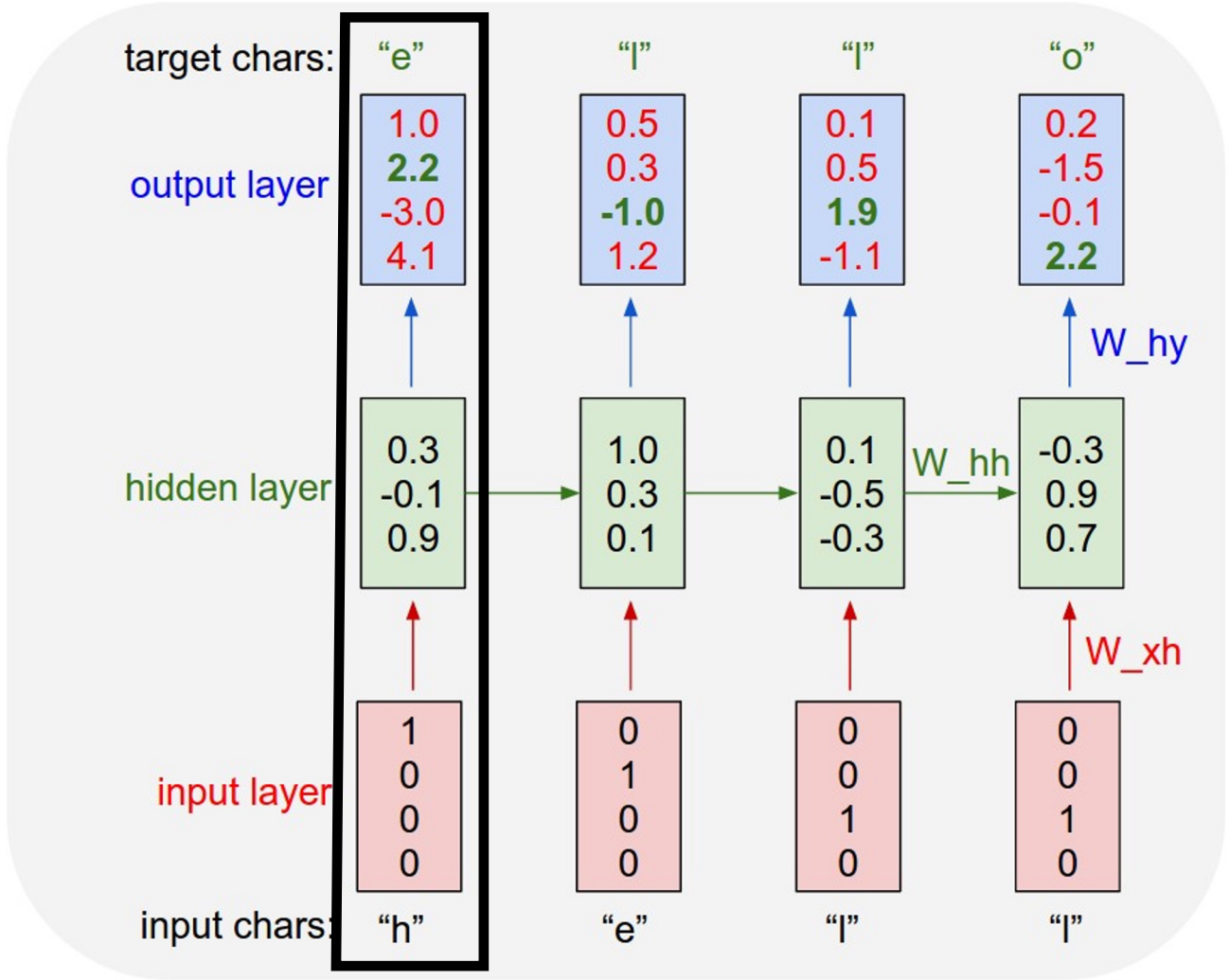
Given "h", predict "e"

Given characters 1, 2, …, t-1, model predicts character t

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

Training sequence: "hello"

Vocabulary: [h, e, l, o]



Example: Language Modeling

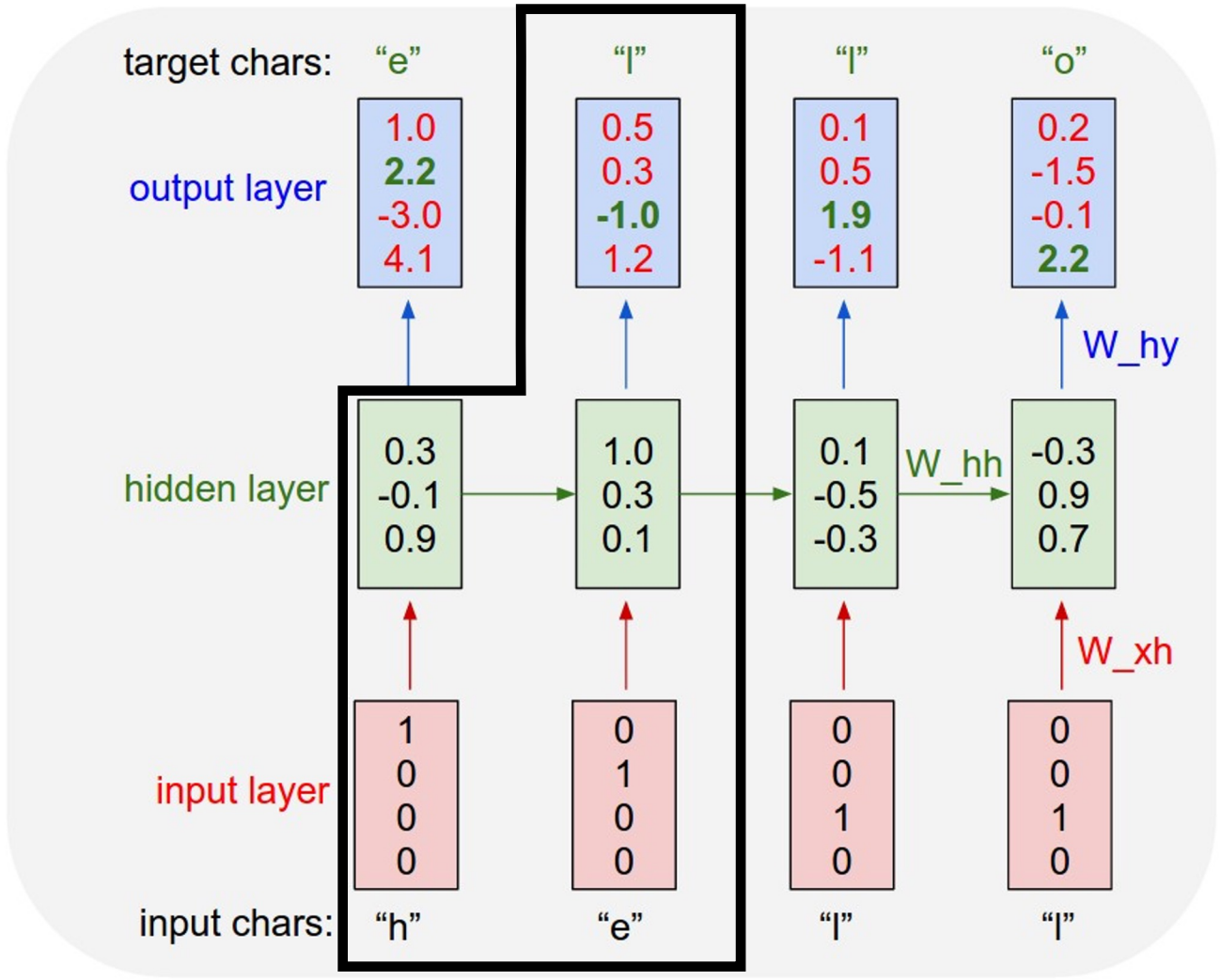
Given "he", predict "l"

Given characters 1, 2, …, t-1, model predicts character t

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

Training sequence: "hello"

Vocabulary: [h, e, l, o]



Example: Language Modeling

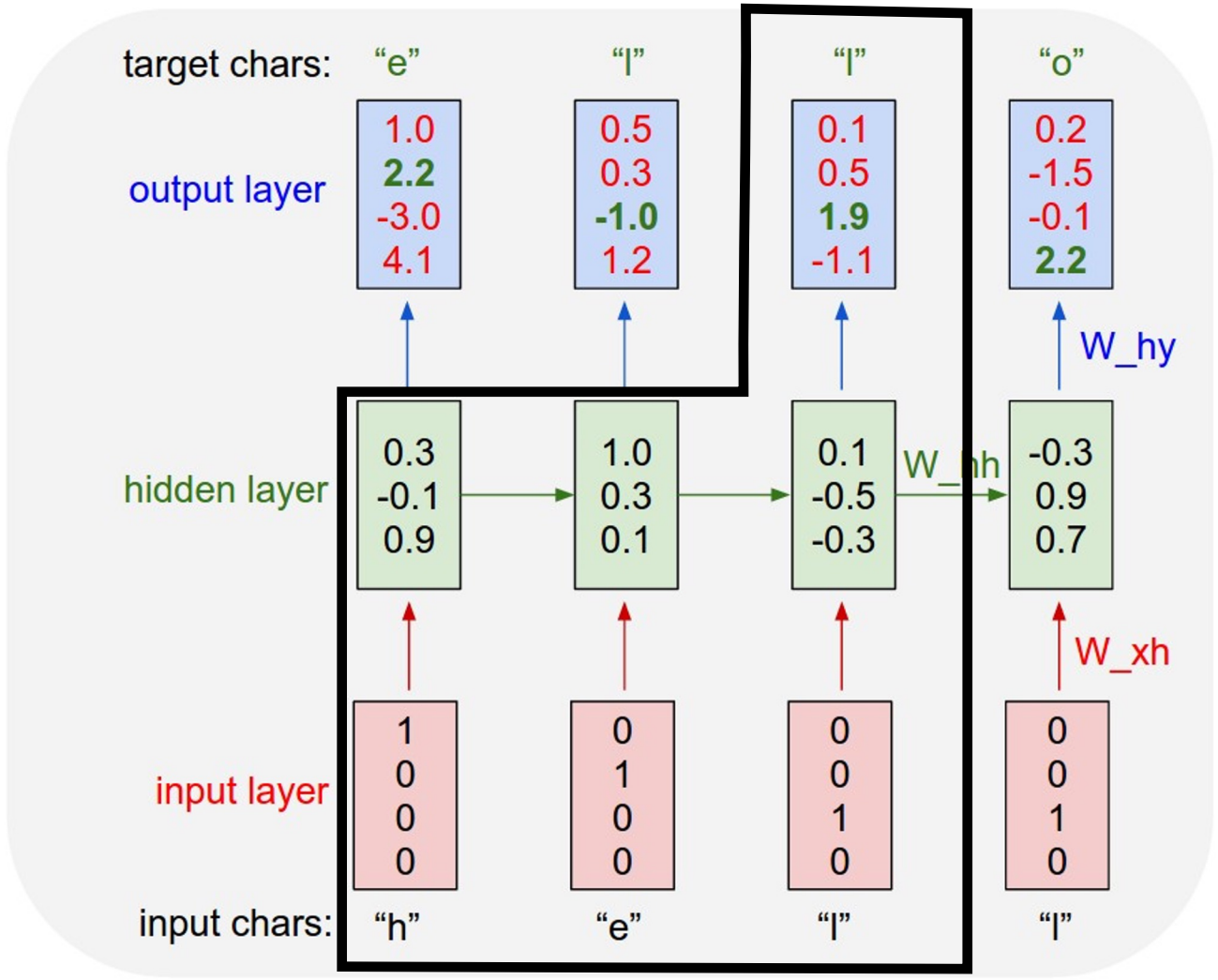
Given "hel", predict "l"

Given characters 1, 2, …, t-1, model predicts character t

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

Training sequence: "hello"

Vocabulary: [h, e, l, o]



Example: Language Modeling

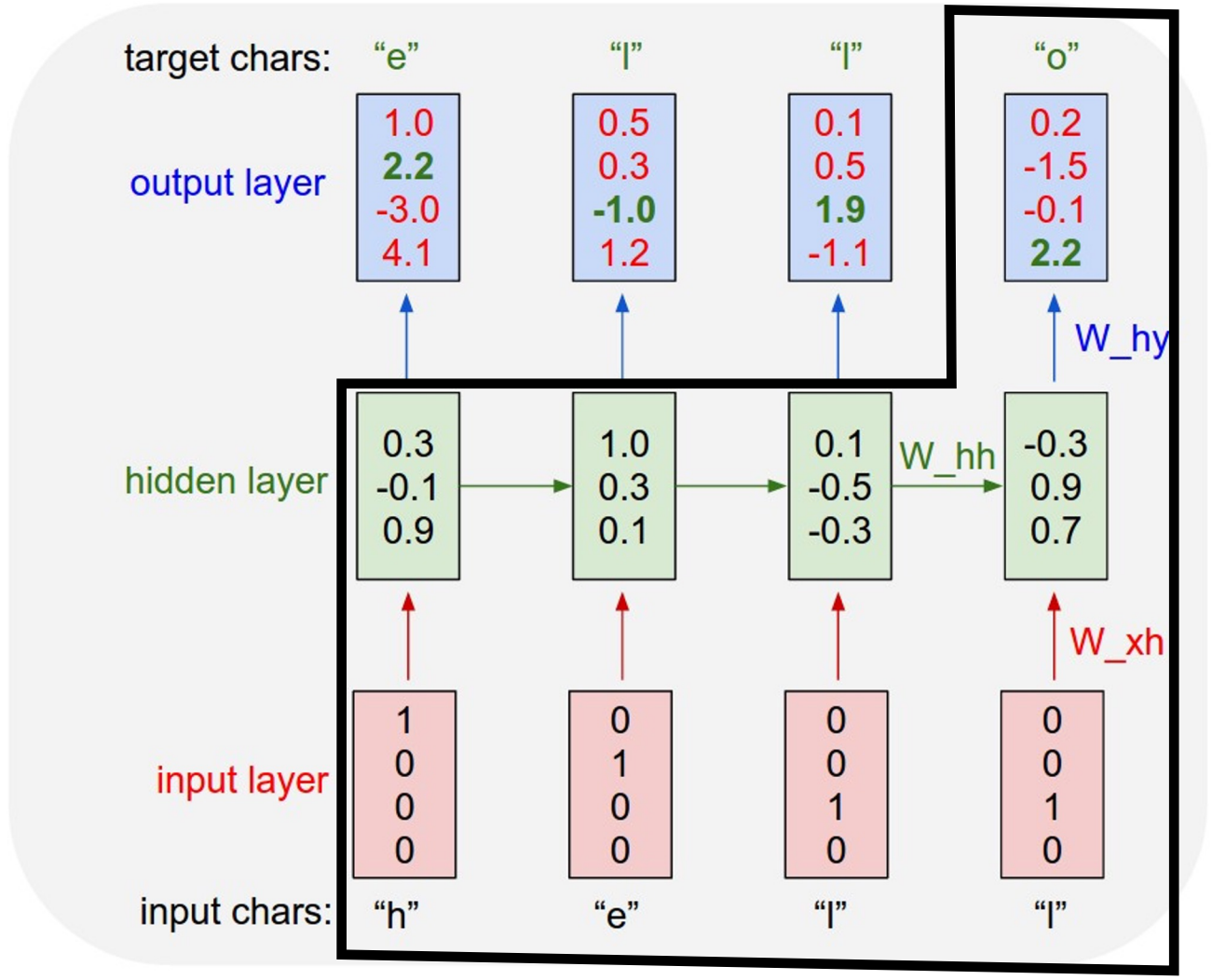
Given "hell", predict "o"

Given characters 1, 2, …, t-1, model predicts character t

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

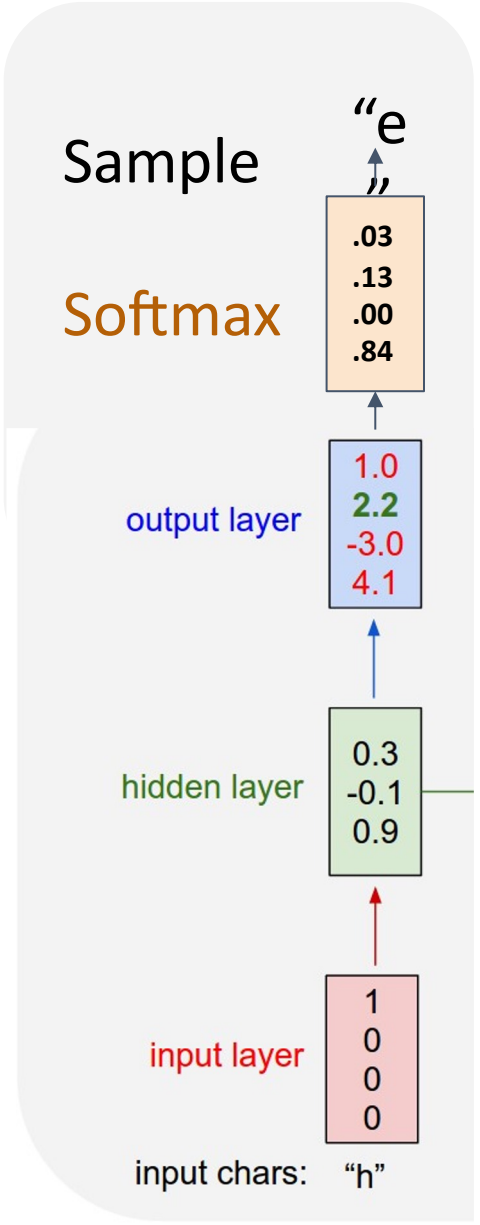
Training sequence: "hello"

Vocabulary: [h, e, l, o]



Example: Language Modeling

At test-time, **generate** new text: sample characters one at a time, feed back to model



Training sequence: ”hello”

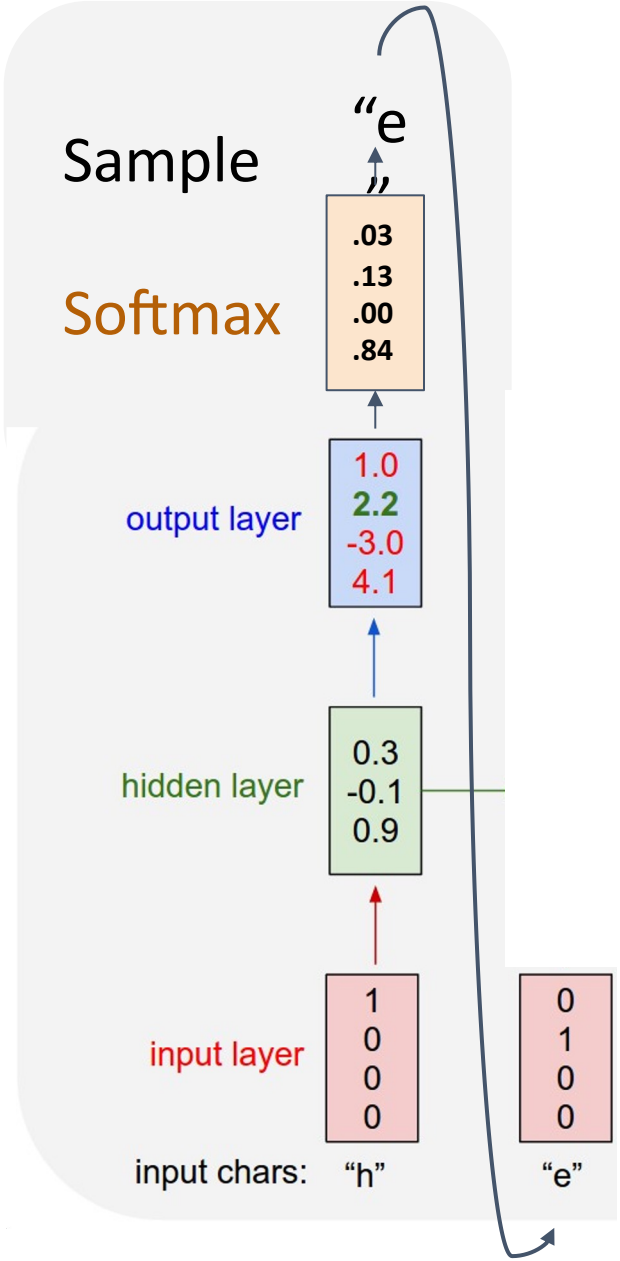
Vocabulary: [h, e, l, o]

Example: Language Modeling

At test-time, **generate** new text: sample characters one at a time, feed back to model

Training sequence: ”hello”

Vocabulary: [h, e, l, o]

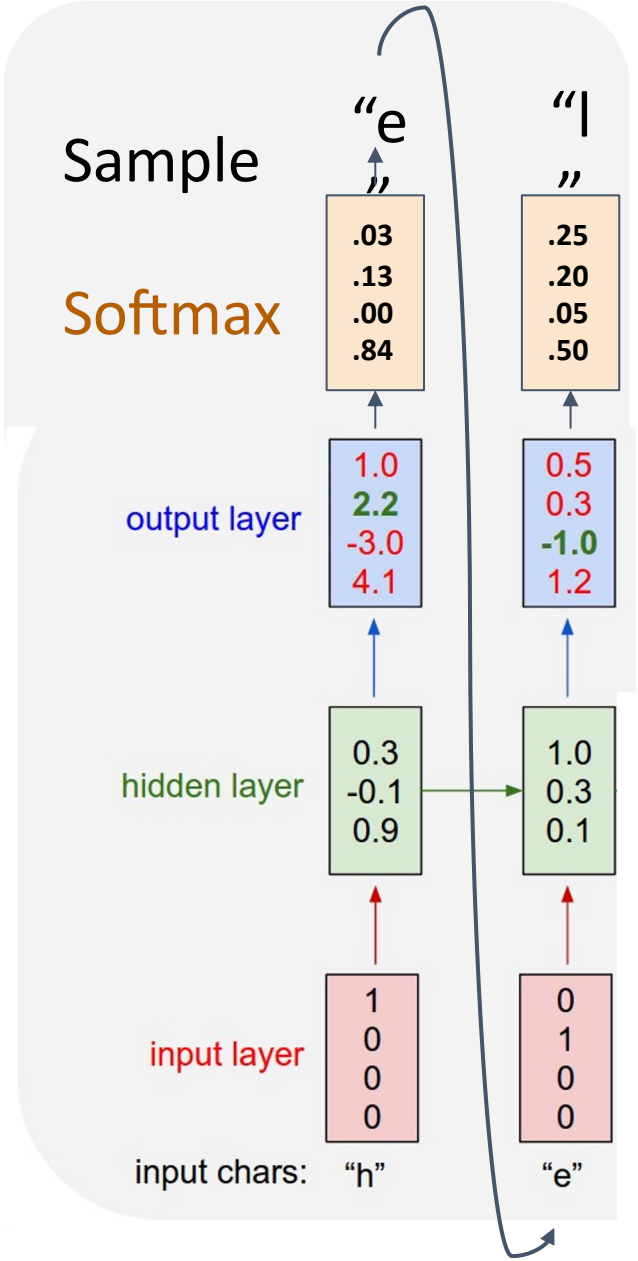


Example: Language Modeling

At test-time, **generate** new text: sample characters one at a time, feed back to model

Training sequence: ”hello”

Vocabulary: [h, e, l, o]

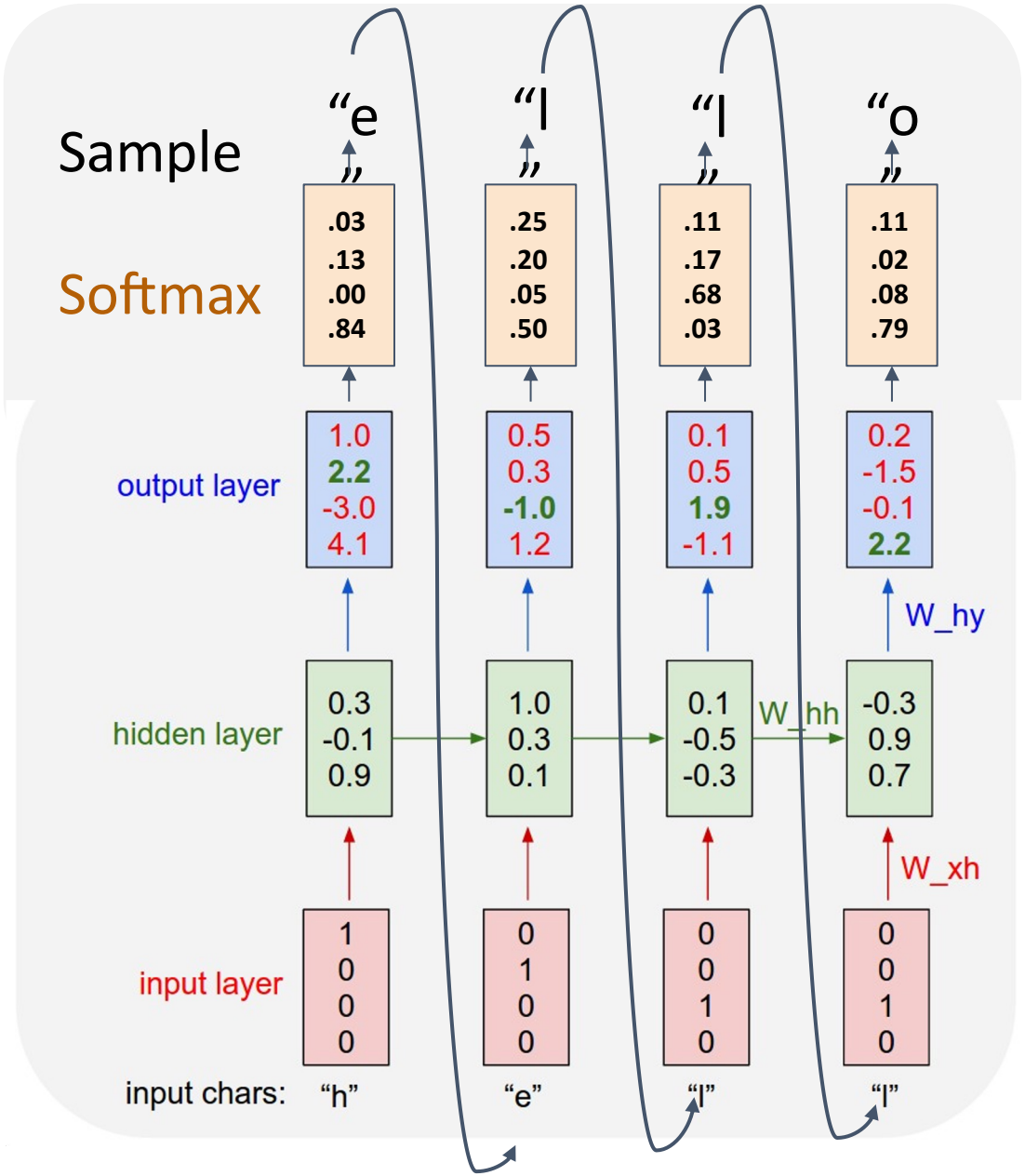


Example: Language Modeling

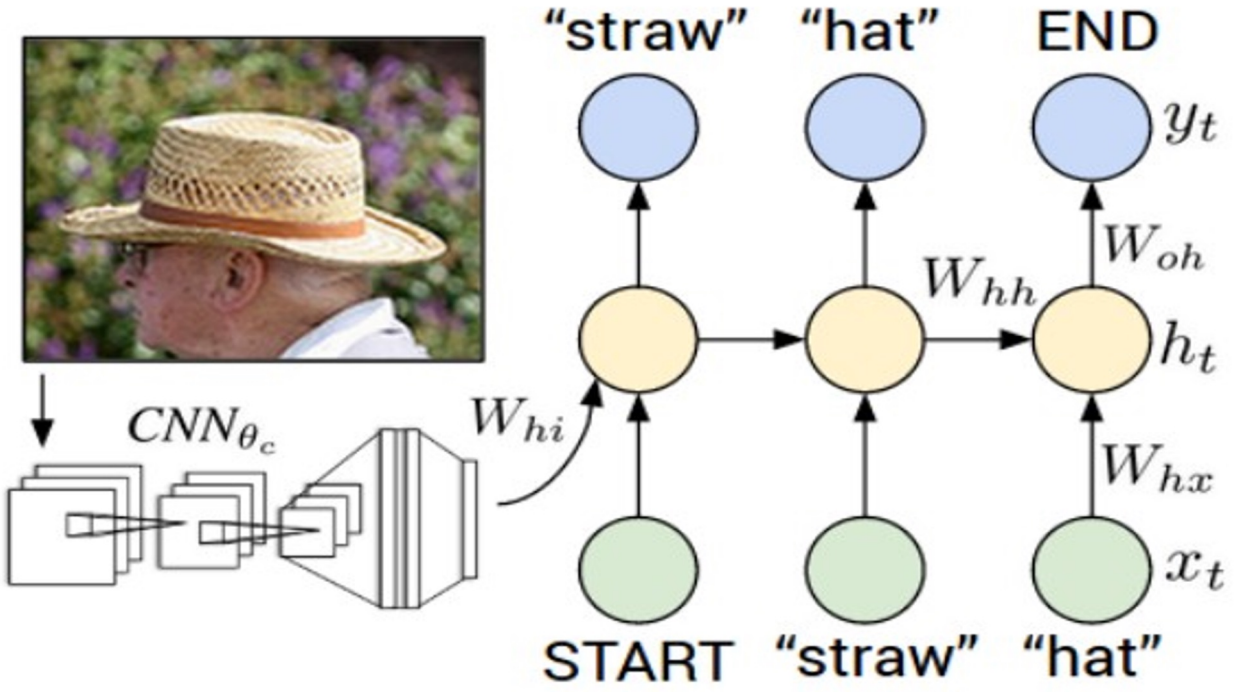
At test-time, **generate** new text: sample characters one at a time, feed back to model

Training sequence: ”hello”

Vocabulary: [h, e, l, o]



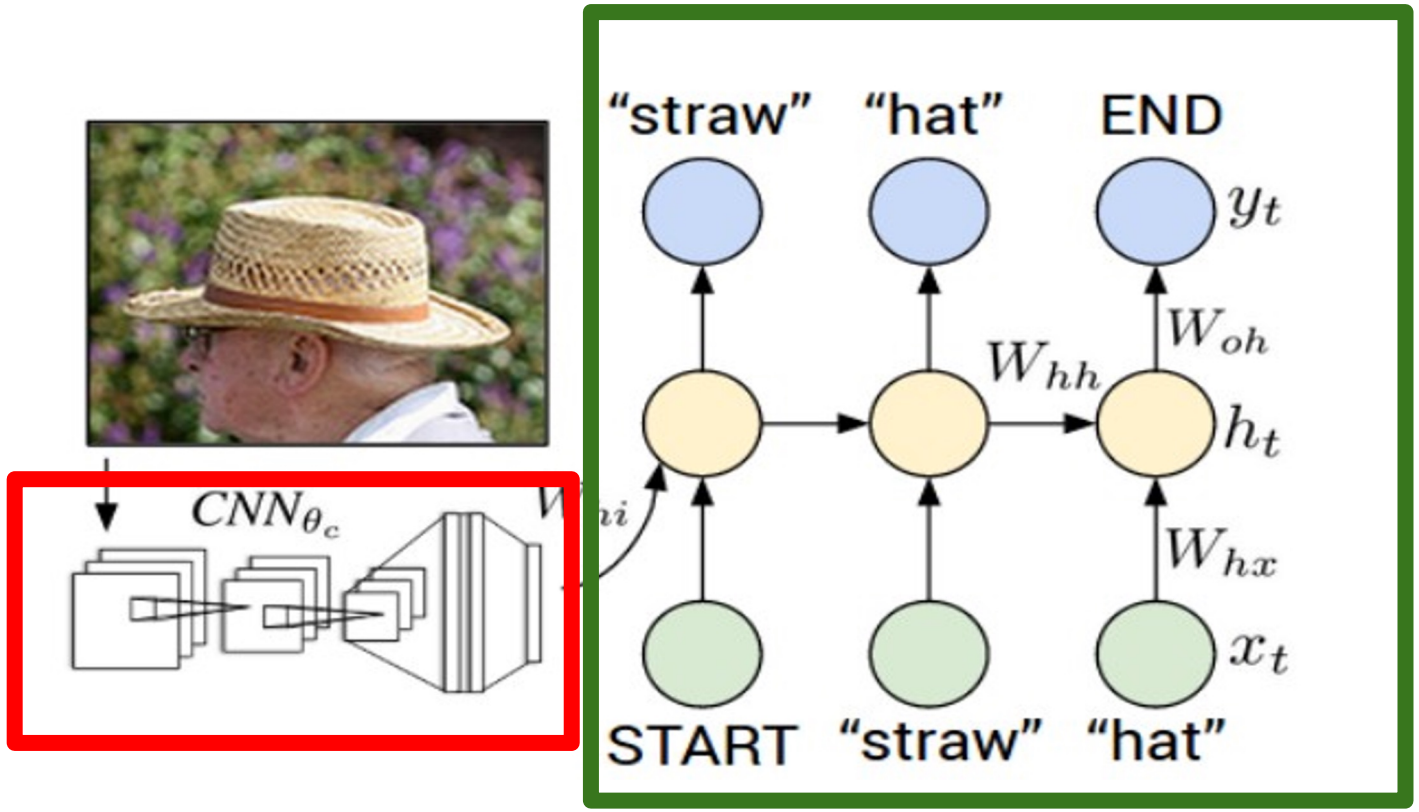
Example: Image Captioning



Mao et al, "Explain Images with Multimodal Recurrent Neural Networks", NeurIPS 2014 Deep Learning and Representation Workshop
Karpathy and Fei-Fei, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015
Vinyals et al, "Show and Tell: A Neural Image Caption Generator", CVPR 2015
Donahue et al, "Long-term Recurrent Convolutional Networks for Visual Recognition and Description", CVPR 2015
Chen and Zitnick, "Learning a Recurrent Visual Representation for Image Caption Generation", CVPR 2015

Figure from Karpathy et a, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015

Example: Image Captioning



Recurrent Neural Network

Convolutional Neural Network

Figure from Karpathy et a, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015

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

image

conv-64
conv-64
maxpool

conv-128
conv-128
maxpool

conv-256
conv-256
maxpool

conv-512
conv-512
maxpool

conv-512
conv-512
maxpool

FC-4096
FC-4096

x0

<START>

image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096

Before:

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

Now:

$$\tanh(\boldsymbol{W_{hh}h_{t-1}} + \boldsymbol{W_{xh}x_t} + \boldsymbol{W_{ih}v} + b_h)$$

image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096

Before:

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

Now:

$$\tanh(\boldsymbol{W_{hh}h_{t-1}} + \boldsymbol{W_{xh}x_t} + \boldsymbol{W_{ih}v} + b_h)$$

image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096

Before:

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

Now:

$$\tanh(\boldsymbol{W_{hh}h_{t-1}} + \boldsymbol{W_{xh}x_t} + \boldsymbol{W_{ih}v} + b_h)$$

W_{ih}

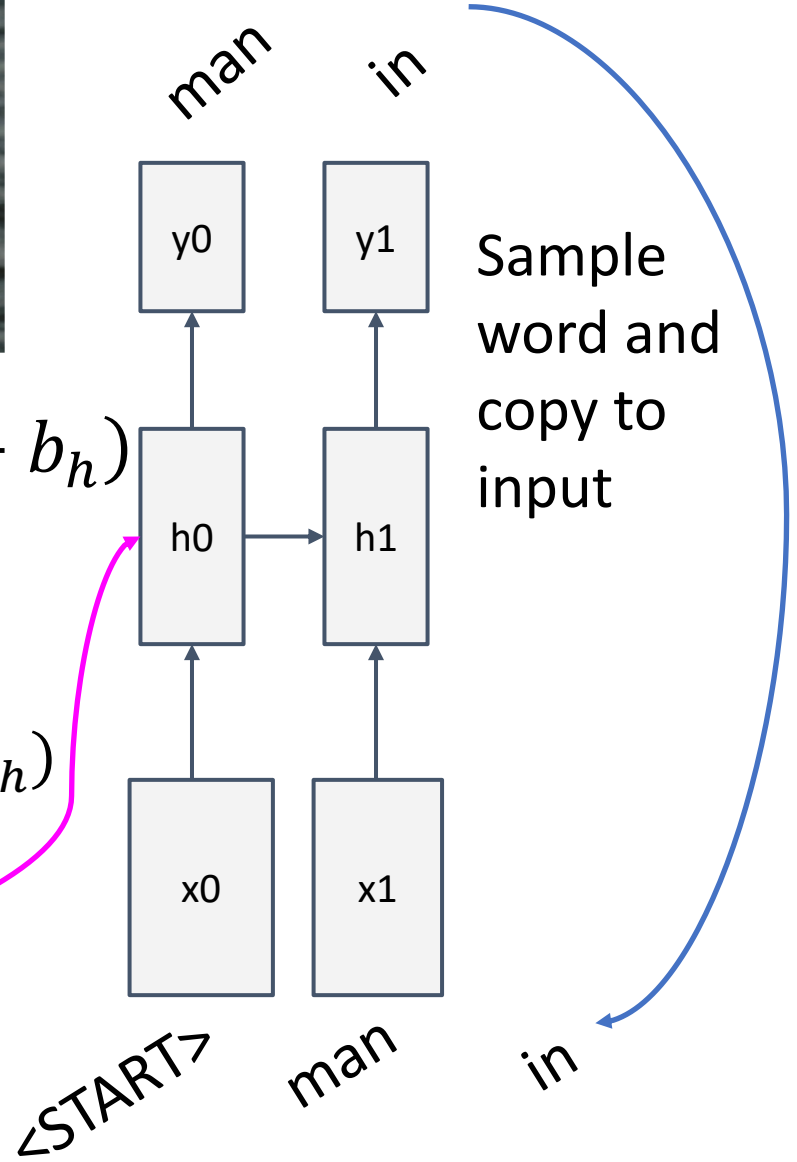


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096

Before:

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

Now:

$$\tanh(\boldsymbol{W_{hh}h_{t-1}} + \boldsymbol{W_{xh}x_t} + \boldsymbol{W_{ih}v} + b_h)$$

$\mathbf{W_{ih}}$

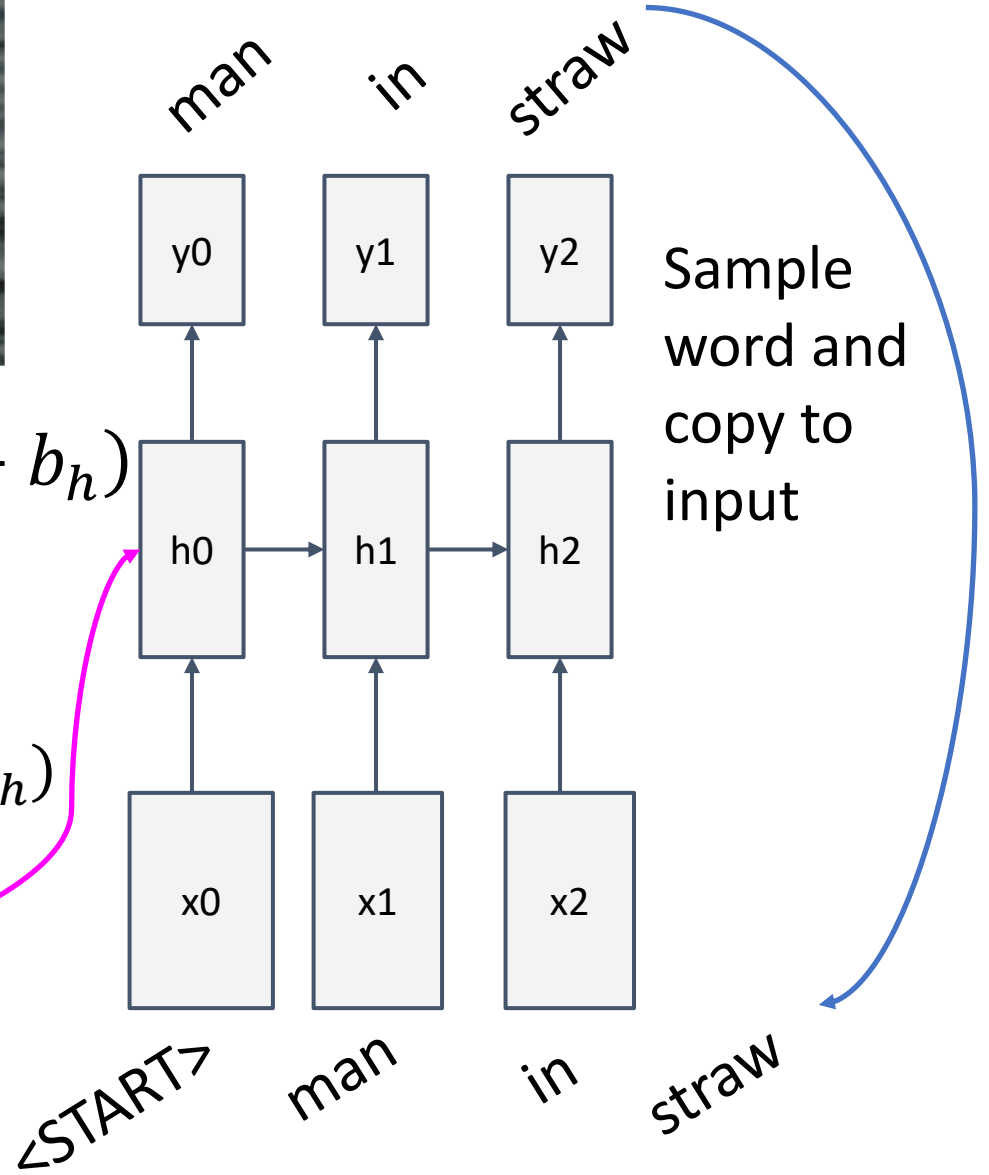


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096

Before:

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

Now:

$$\tanh(\boldsymbol{W_{hh}h_{t-1}} + \boldsymbol{W_{xh}x_t} + \boldsymbol{W_{ih}v} + b_h)$$

$\boldsymbol{W_{ih}}$

man in straw hat

y0 y1 y2 y3

h0 h1 h2 h3

x0 x1 x2 x3

<START> man in straw hat

Sample word and copy to input

Stop after sampling <END> token

image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096

Before:

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

Now:

$$\tanh(\boldsymbol{W_{hh}h_{t-1}} + \boldsymbol{W_{xh}x_t} + \boldsymbol{W_{ih}v} + b_h)$$

$\boldsymbol{W_{ih}}$

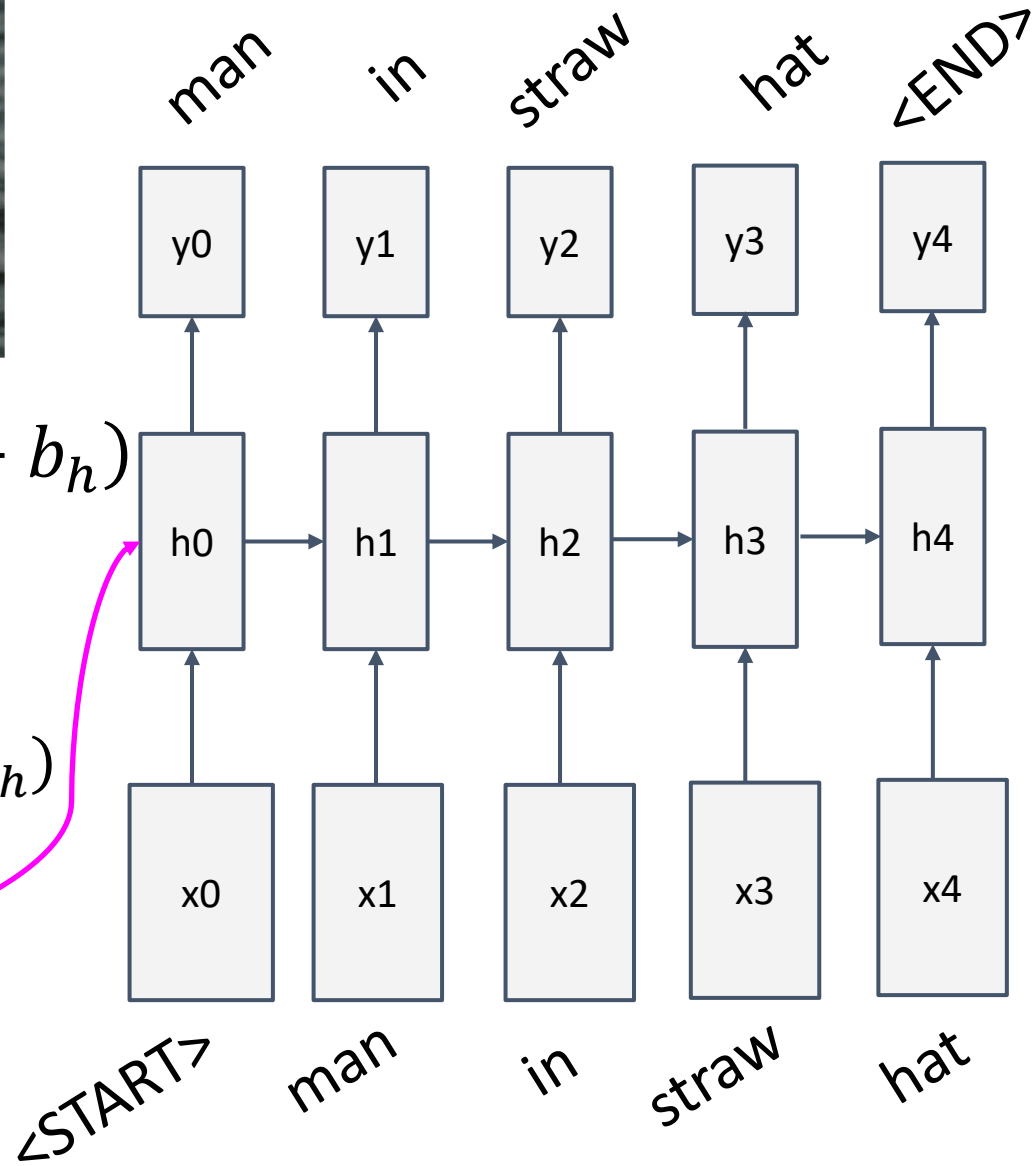


Image Captioning: Example Results

Captions generated using neuraltalk2
All images are CC0 Public domain: cat suitcase, cat tree, dog, bear, surfers, tennis, giraffe, motorcycle

A cat sitting on a suitcase on the floor

A cat is sitting on a tree branch

A dog is running in the grass with a frisbee

A white teddy bear sitting in the grass

Two people walking on the beach with surfboards

A tennis player in action on the court

Two giraffes standing in a grassy field

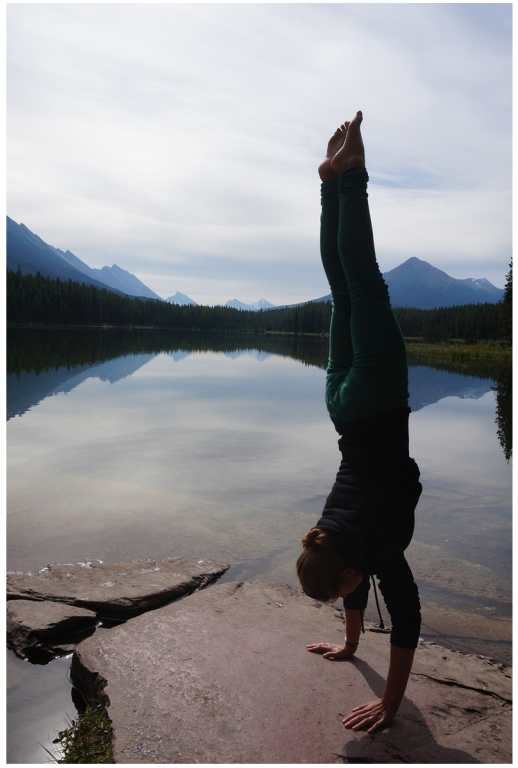
A man riding a dirt bike on a dirt track

Image Captioning: Failure Cases

Captions generated using neuraltalk2
All images are CC0 Public domain: fur coat, handstand, spider web, baseball

A woman is holding a cat in her hand

A person holding a computer mouse on a desk



A woman standing on a beach holding a surfboard

A bird is perched on a tree branch

A man in a baseball uniform throwing a ball

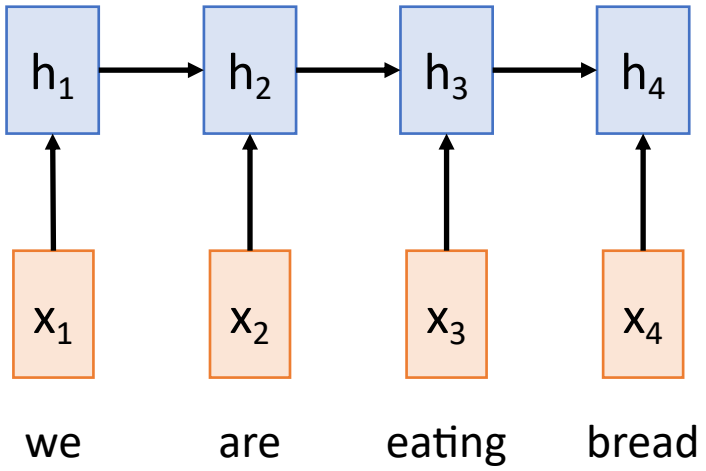
RNN with Attention

Sequence-to-Sequence with RNNs

Input: Sequence $x_1, \ldots x_T$

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

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



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

Sequence-to-Sequence with RNNs

Input: Sequence $x_1, \ldots x_T$

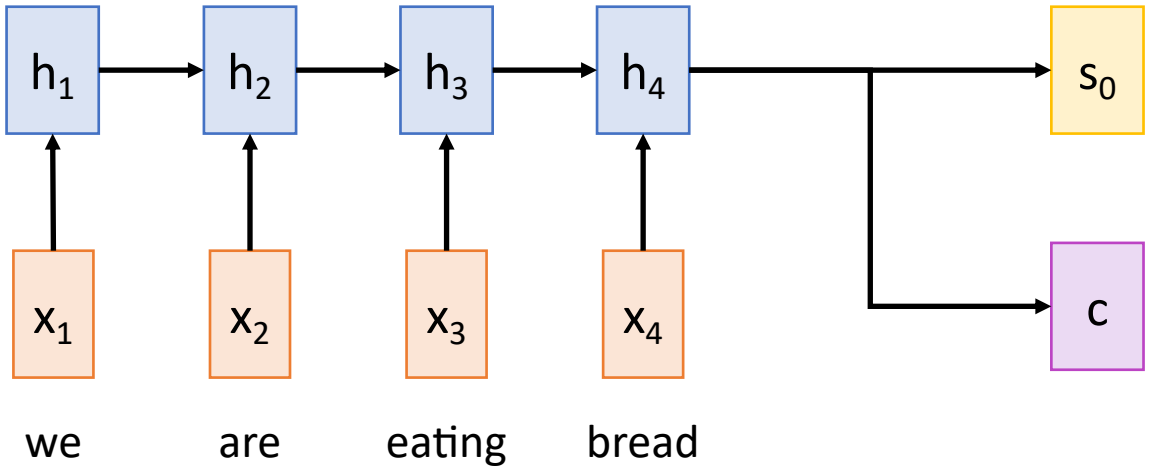
Output: Sequence $y_1, \ldots, y_{T'}$

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

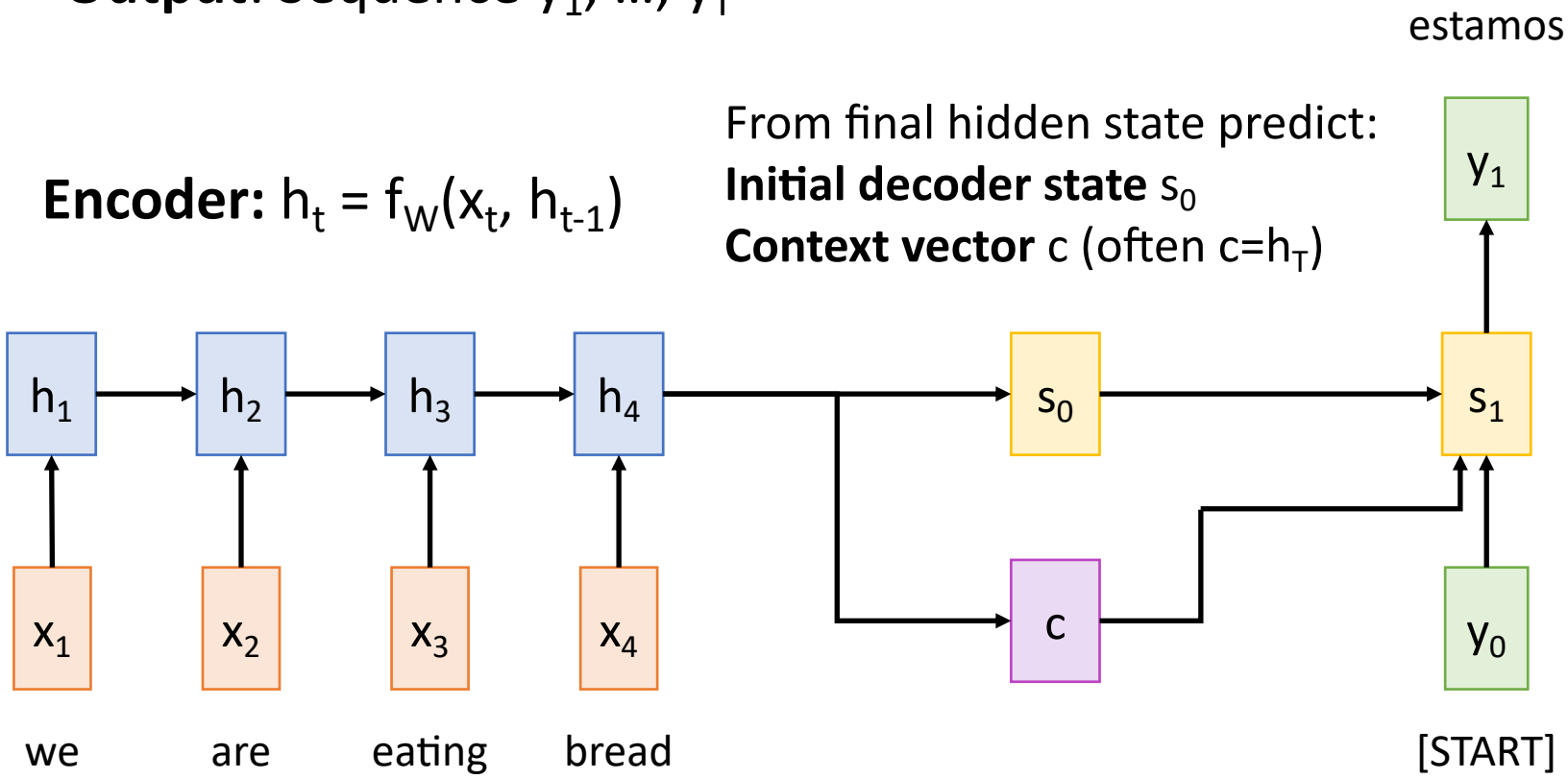
Input: Sequence $x_1, \ldots x_T$

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

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

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

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



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

Sequence-to-Sequence with RNNs

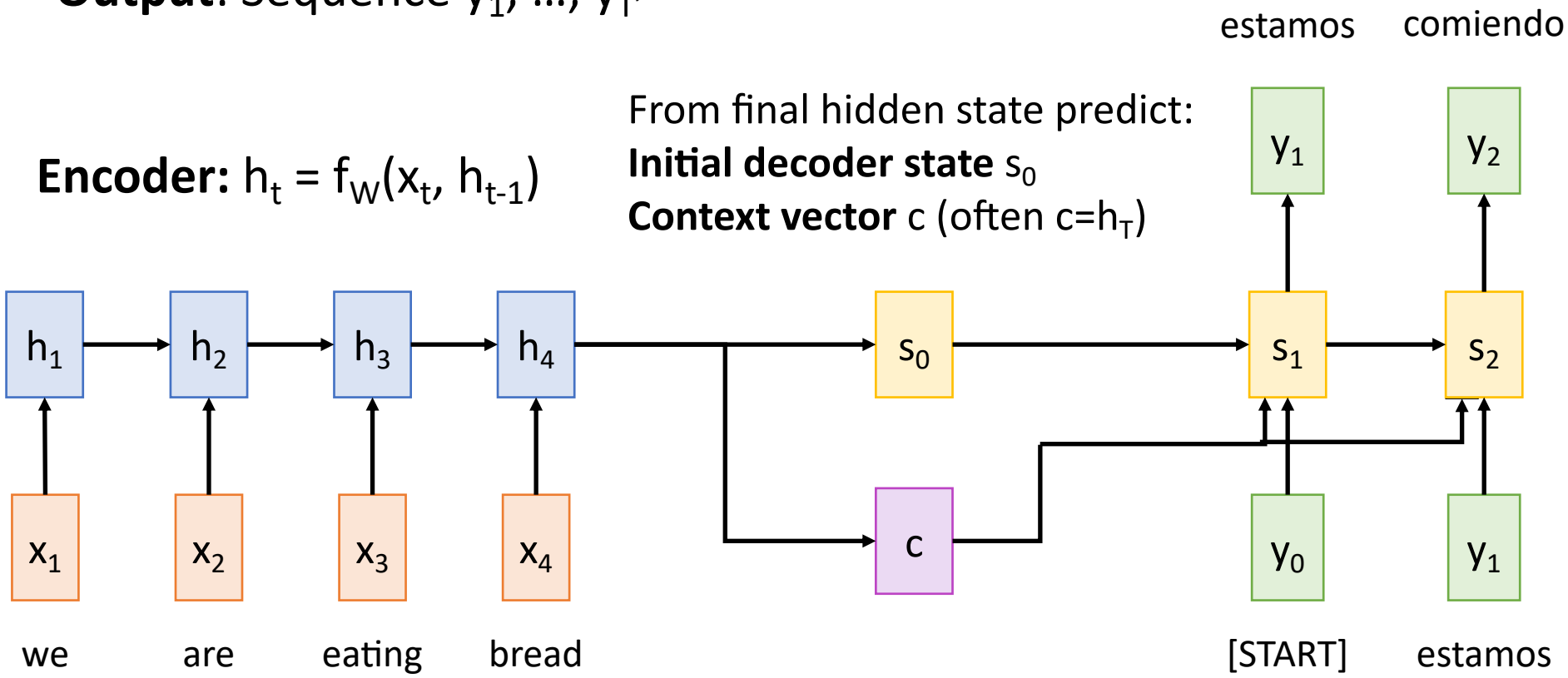
Input: Sequence $x_1, \ldots x_T$

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

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

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

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



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

Sequence-to-Sequence with RNNs

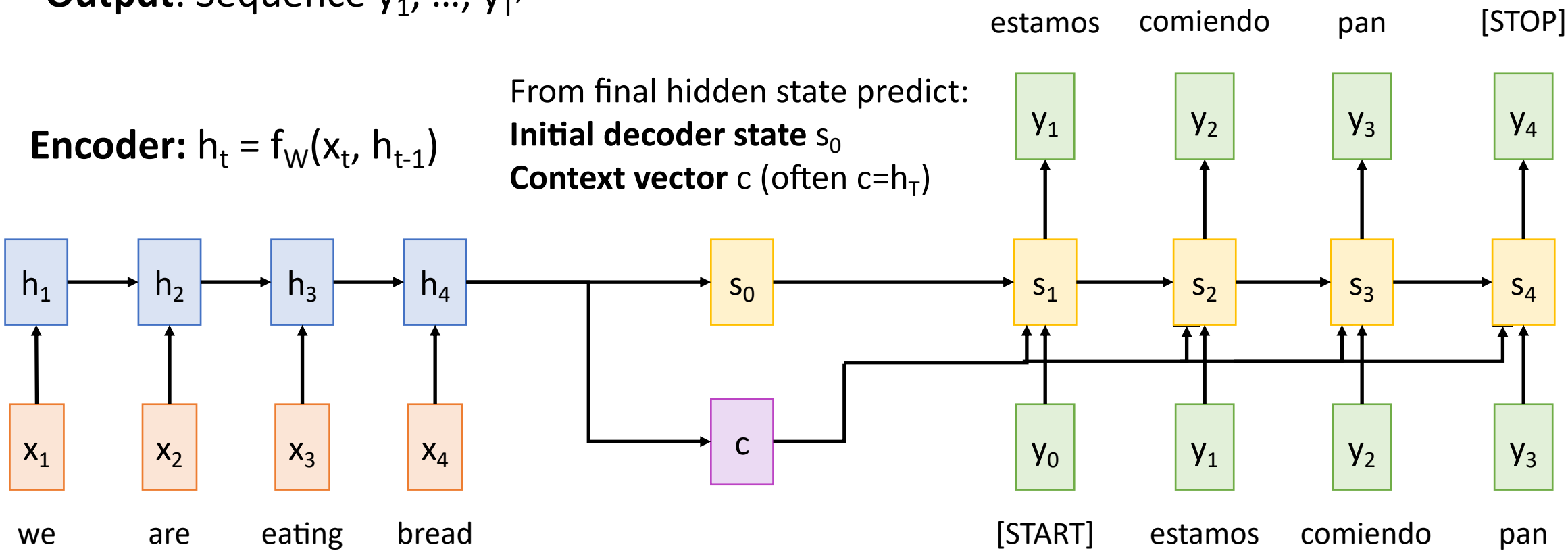
Input: Sequence $x_1, \ldots x_T$

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 $c=h_T$)

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



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

Sequence-to-Sequence with RNNs

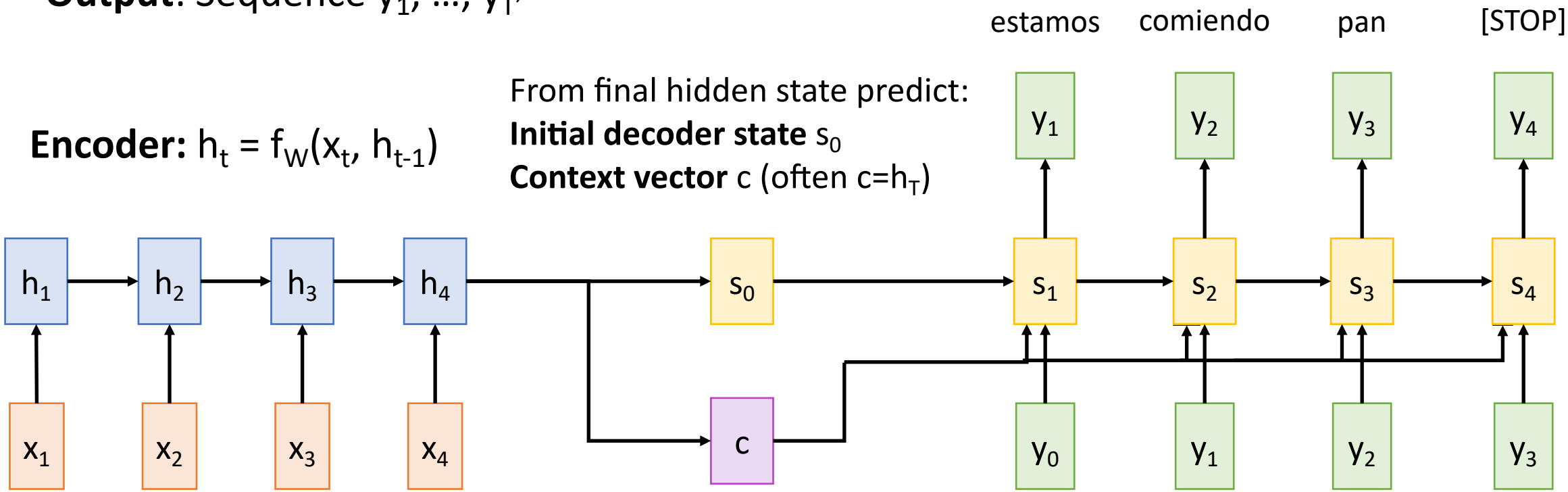
Input: Sequence $x_1, \ldots x_T$

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

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

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

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

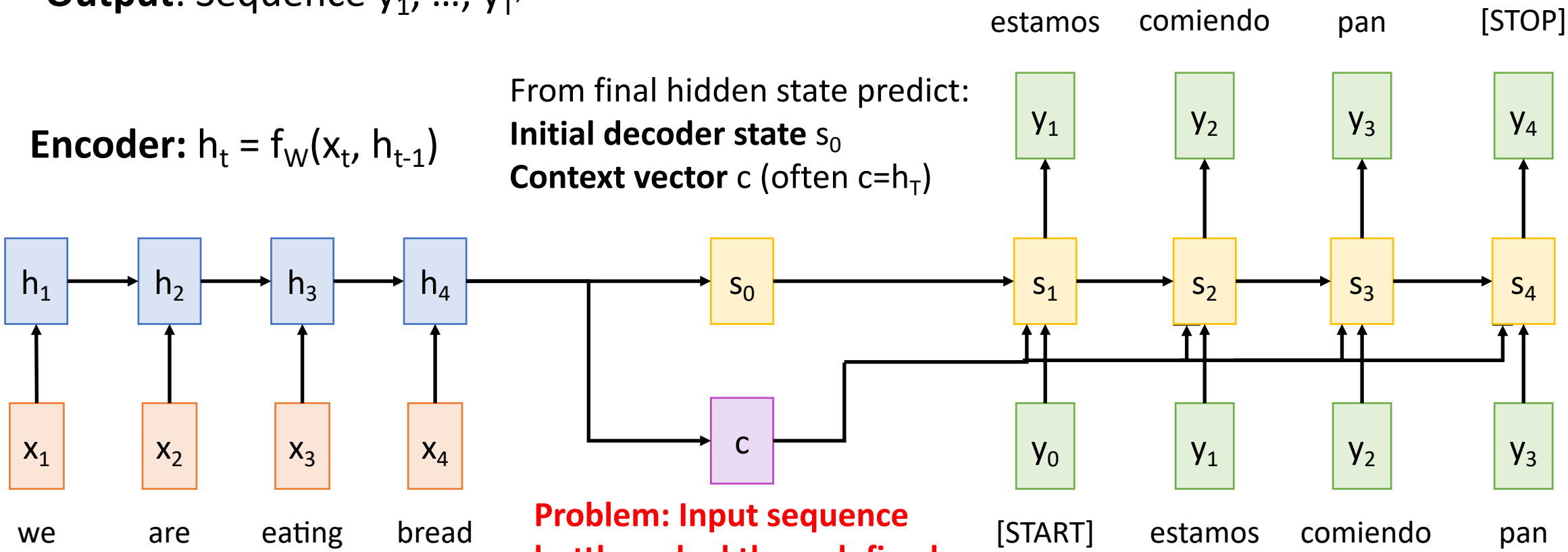
Input: Sequence $x_1, \ldots x_T$

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

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

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!

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

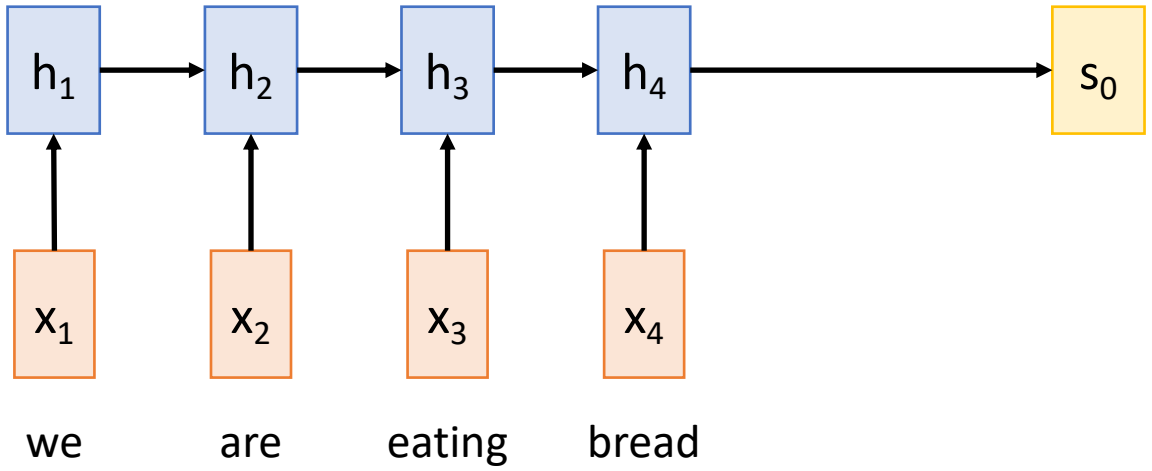
Sequence-to-Sequence with RNNs **and Attention**

Input: Sequence $x_1, \ldots x_T$

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

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

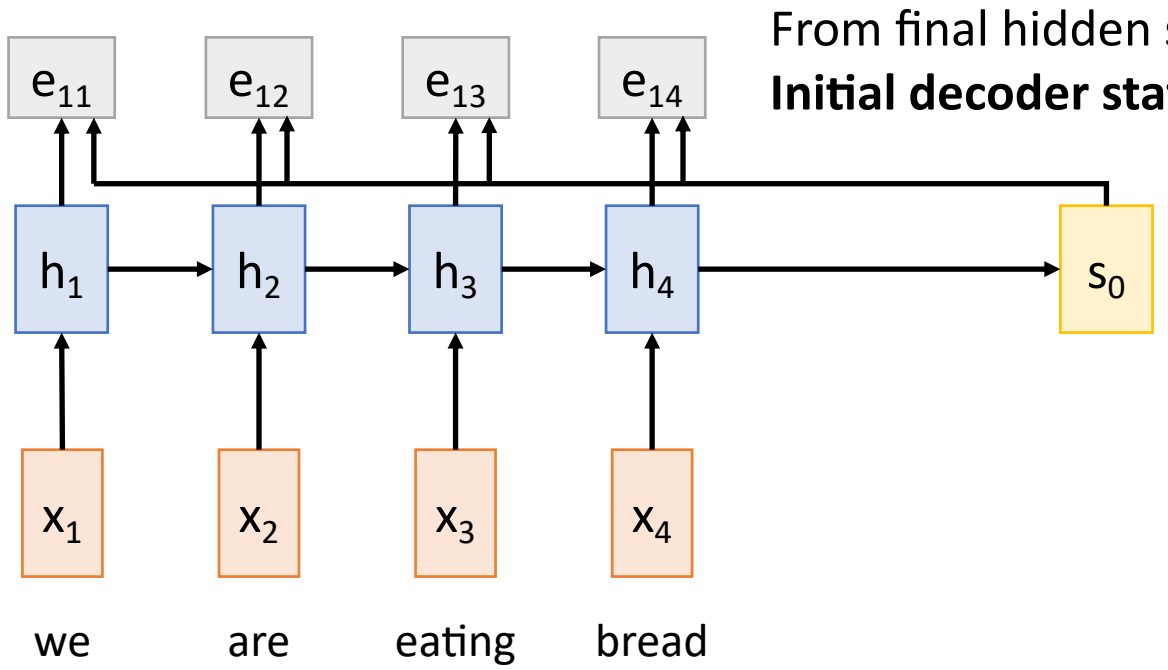
From final hidden state:
Initial decoder state s_0



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

Sequence-to-Sequence with RNNs **and Attention**

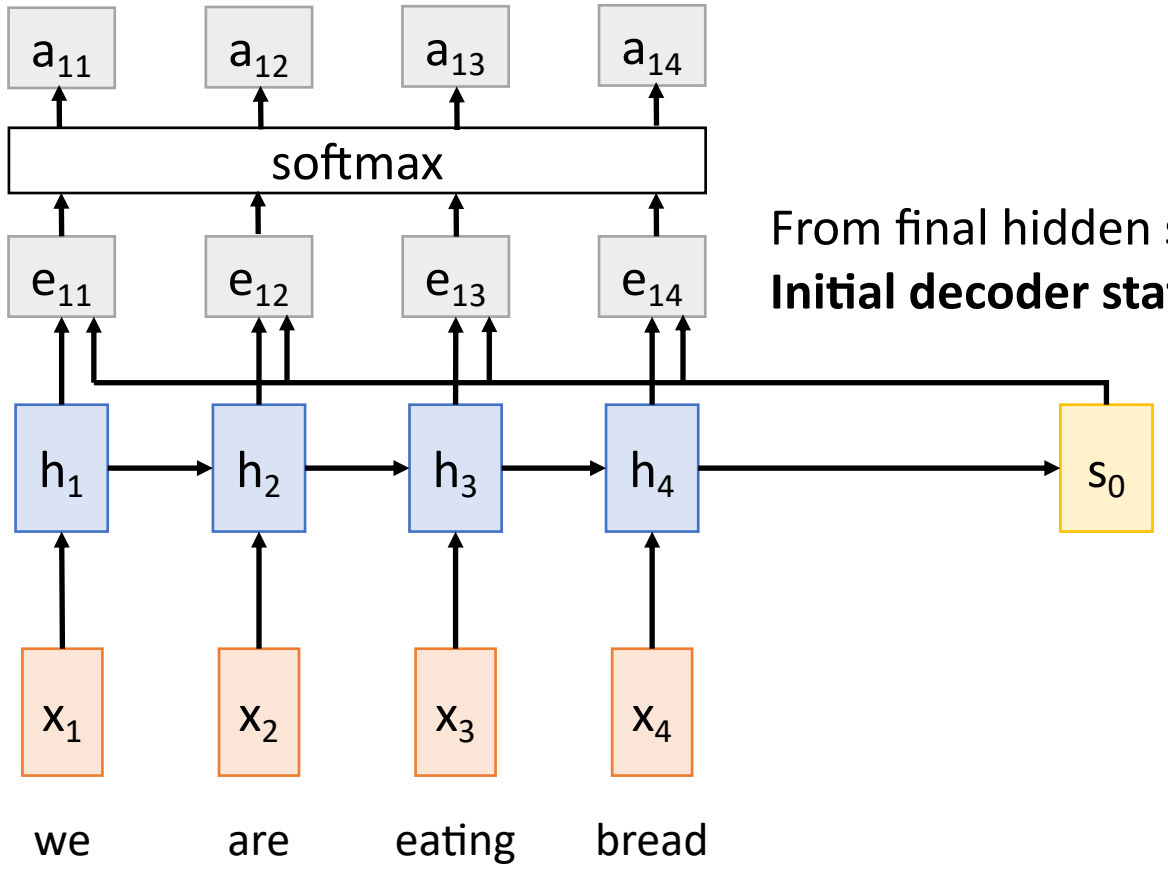
Compute (scalar) **alignment scores**
$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)



From final hidden state:
Initial decoder state s_0

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

Sequence-to-Sequence with RNNs **and Attention**



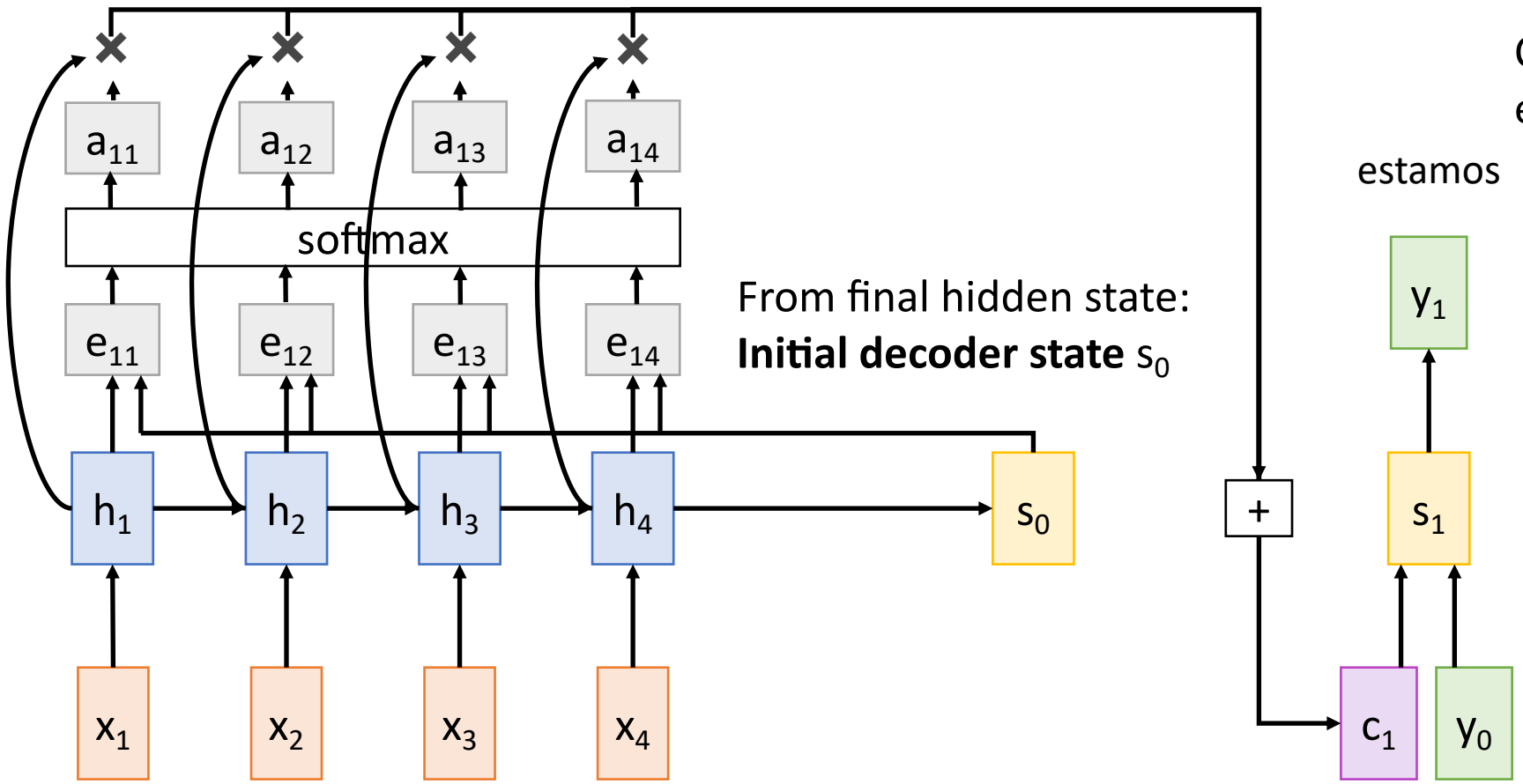
From final hidden state:
Initial decoder state s_0

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

Bahdanau et al, "Neural machine translation 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**
$0 < a_{t,i} < 1 \quad \sum_i a_{t,i} = 1$

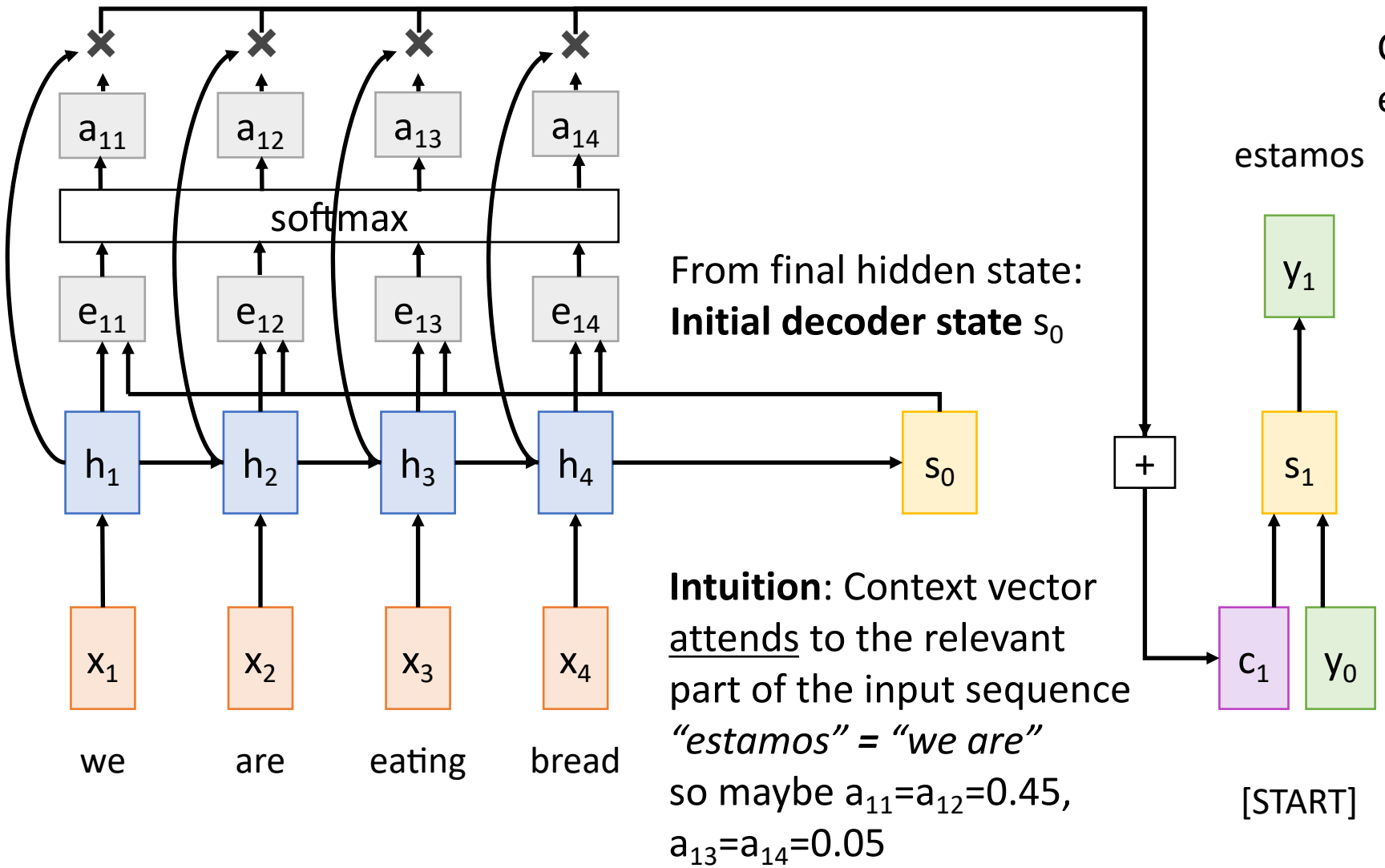
Compute context vector as linear combination of hidden states
$c_t = \sum_i a_{t,i} h_i$

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

Bahdanau et al, "Neural machine translation 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

Intuition: 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**
$0 < a_{t,i} < 1$ $\sum_i a_{t,i} = 1$

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

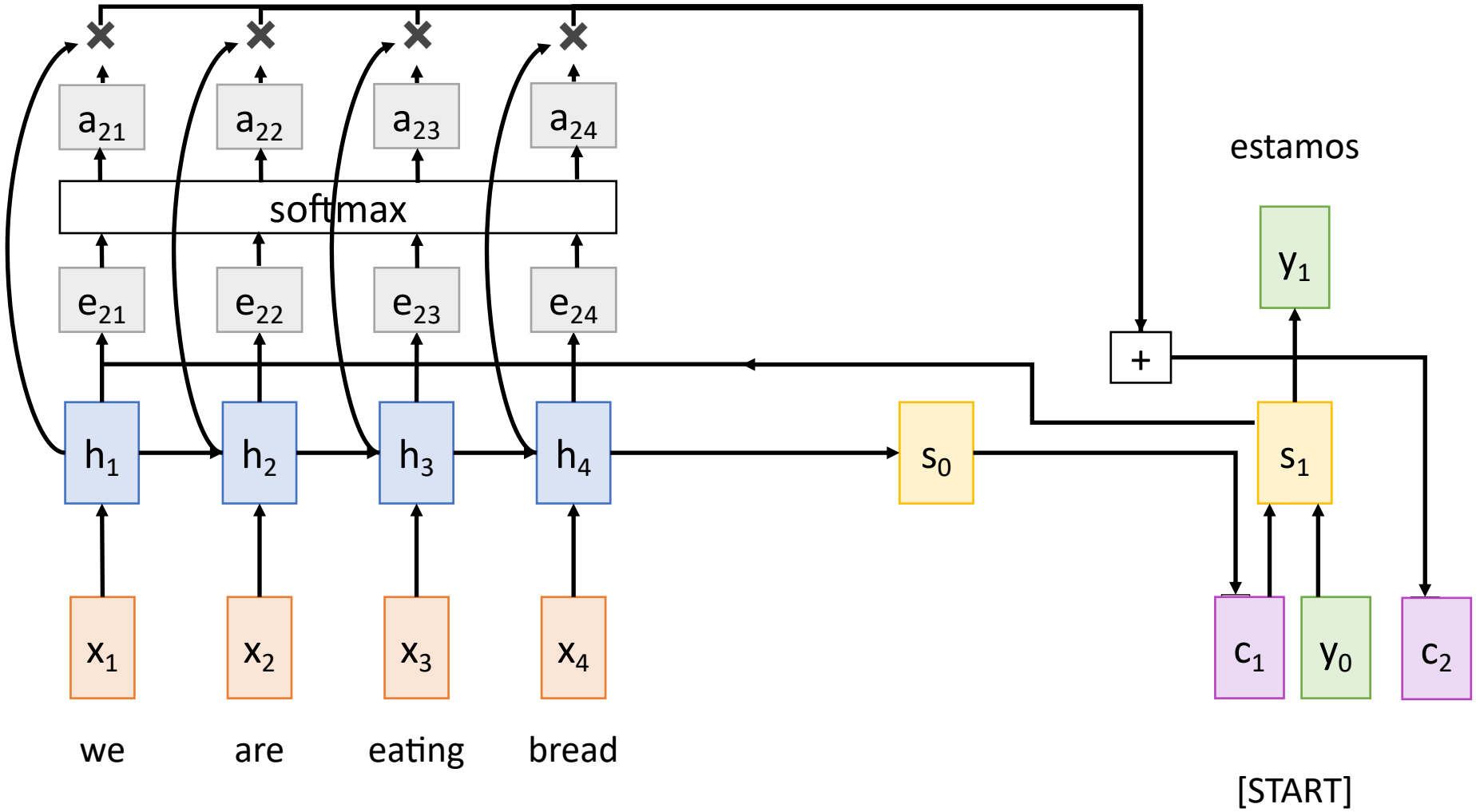
Use context vector in decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c_t)$

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

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

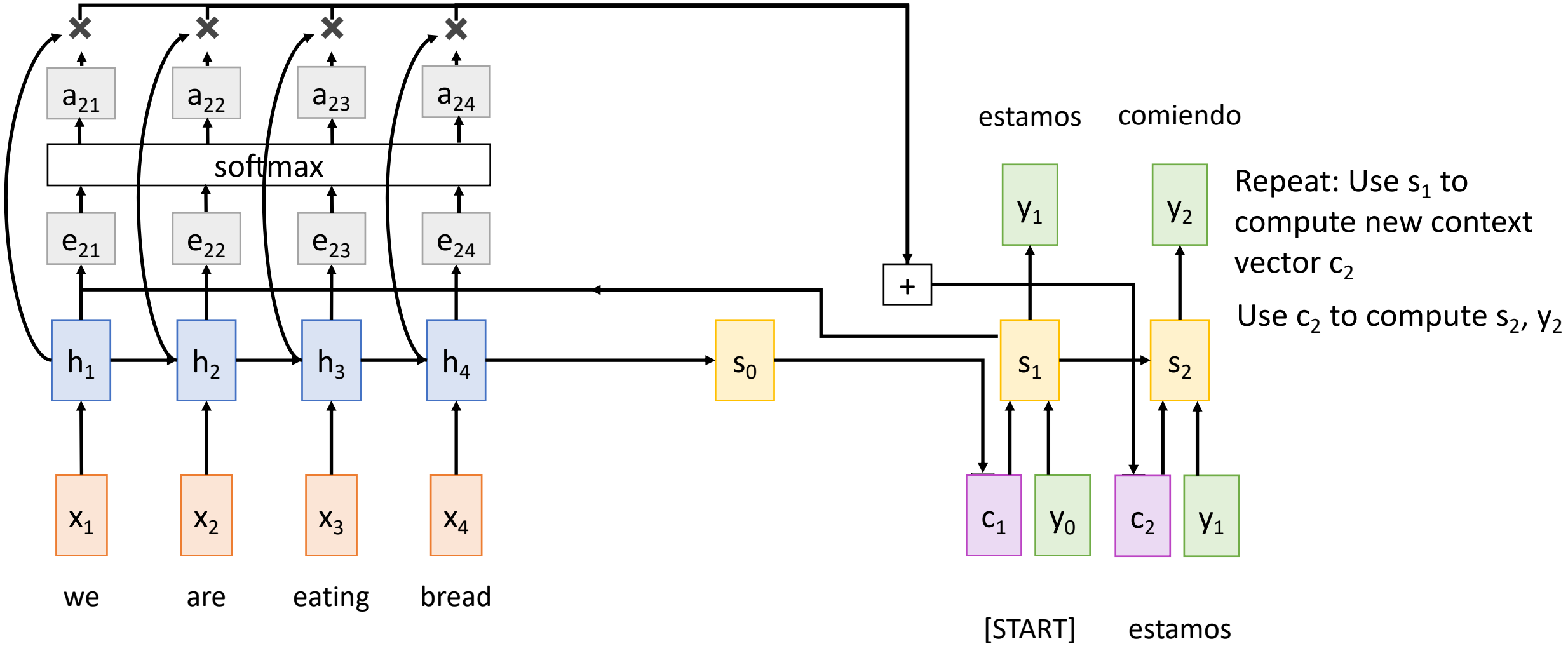
Sequence-to-Sequence with RNNs

Repeat: Use s_1 to compute new context vector c_2



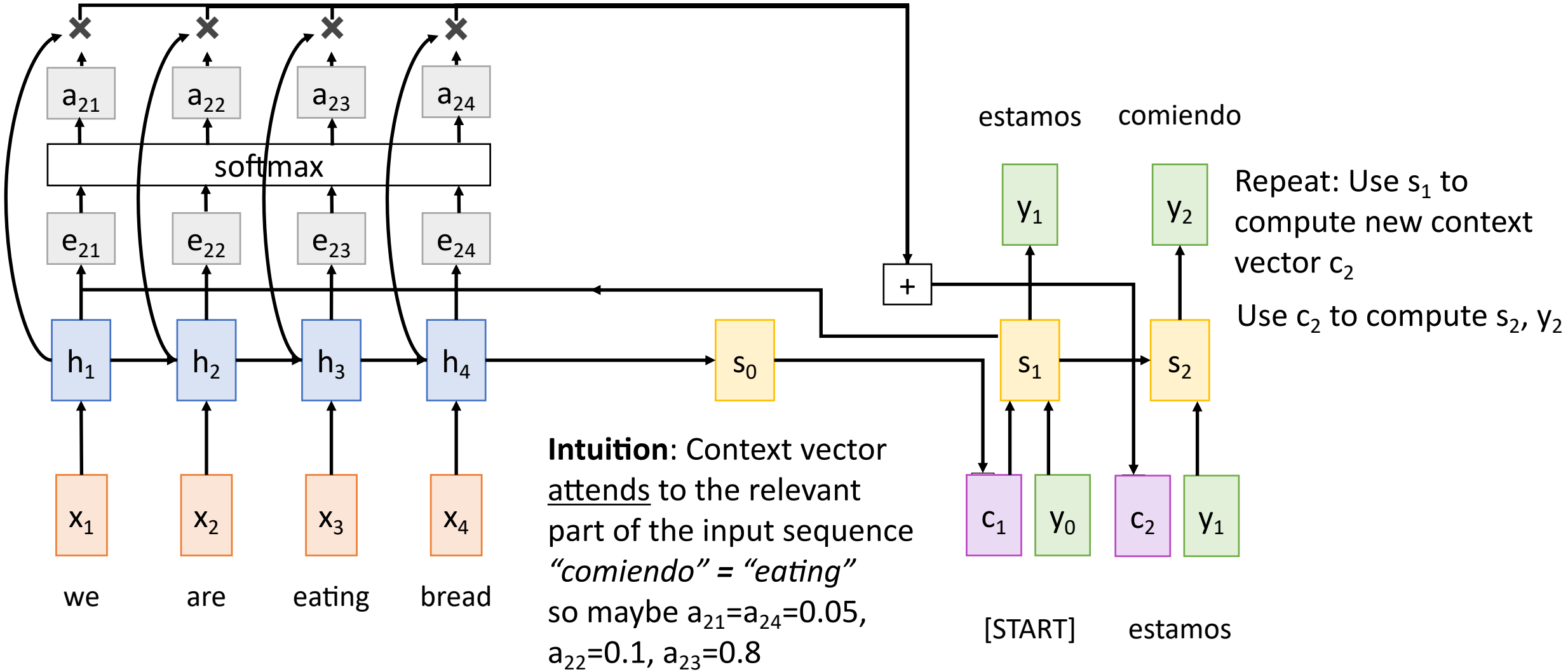
Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

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



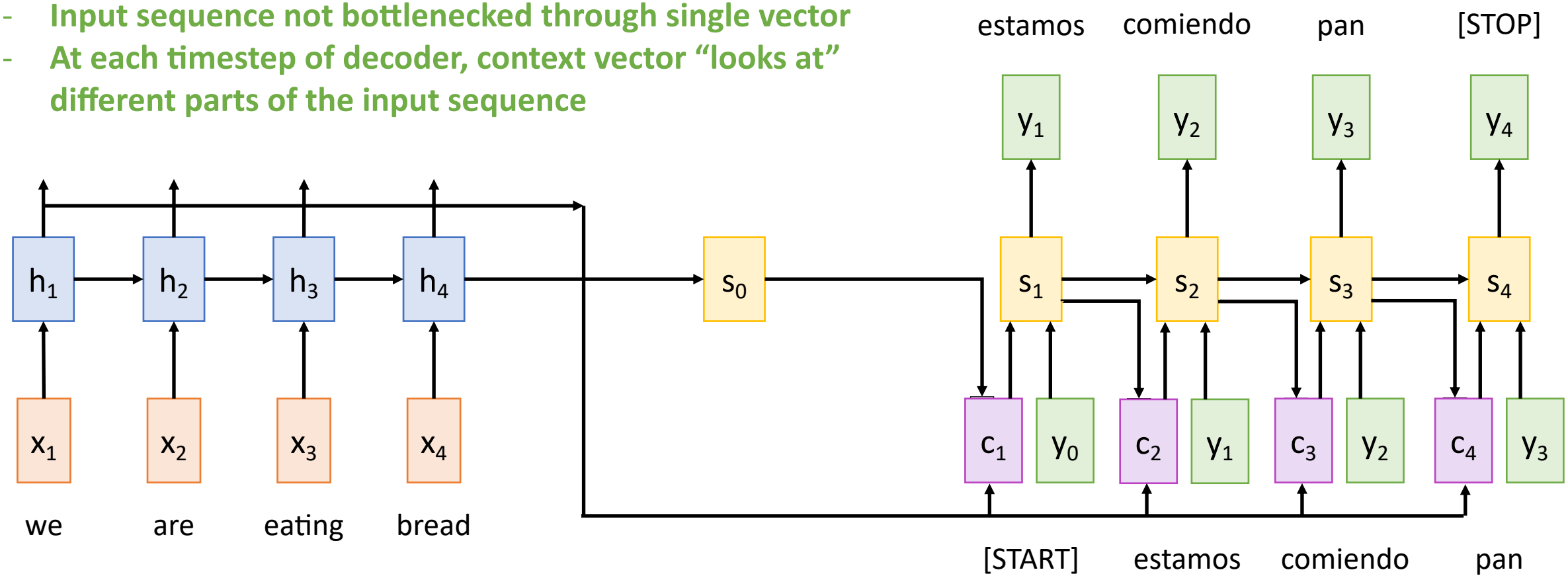
Intuition: Context vector attends to the relevant part of the input sequence
"comiendo" = "eating"
so maybe a_{21}=a_{24}=0.05, a_{22}=0.1, a_{23}=0.8

Bahdanau et al, "Neural machine 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**



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

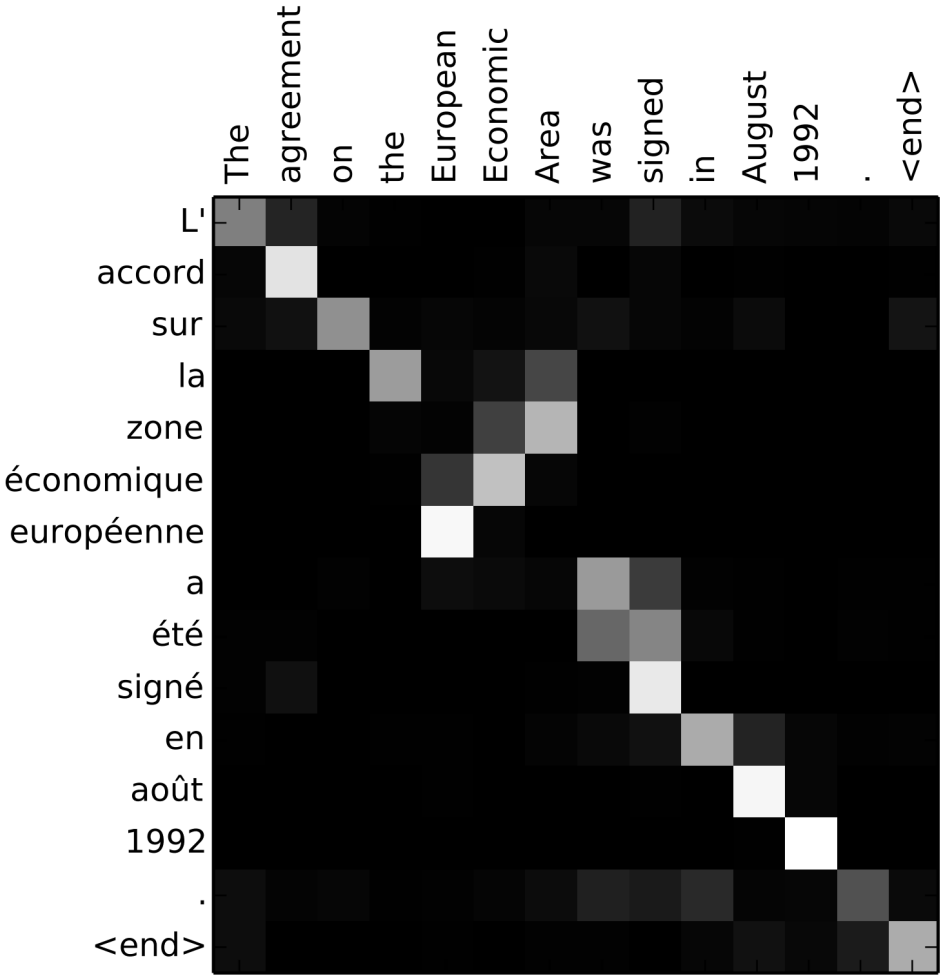
Sequence-to-Sequence with RNNs **and Attention**

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.”

Visualize attention weights $a_{t,i}$



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

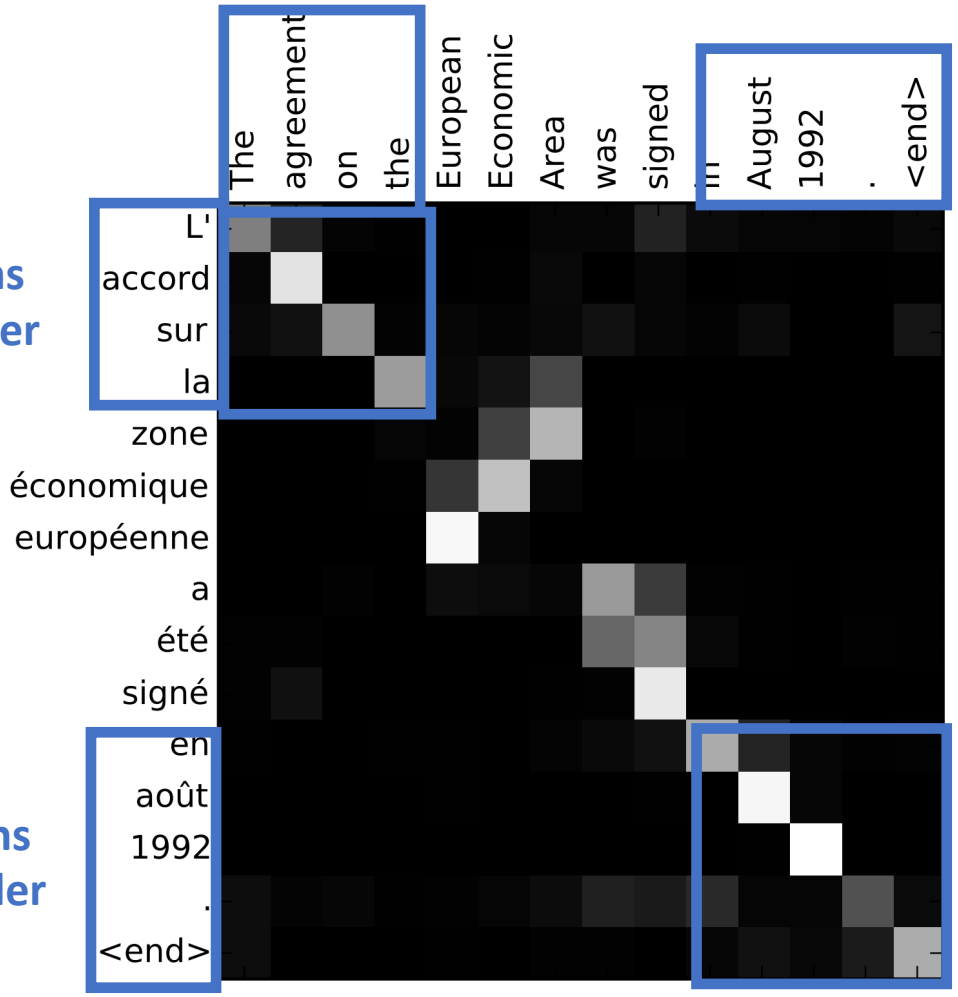
Sequence-to-Sequence with RNNs **and Attention**

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**."

Visualize attention weights $a_{t,i}$



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

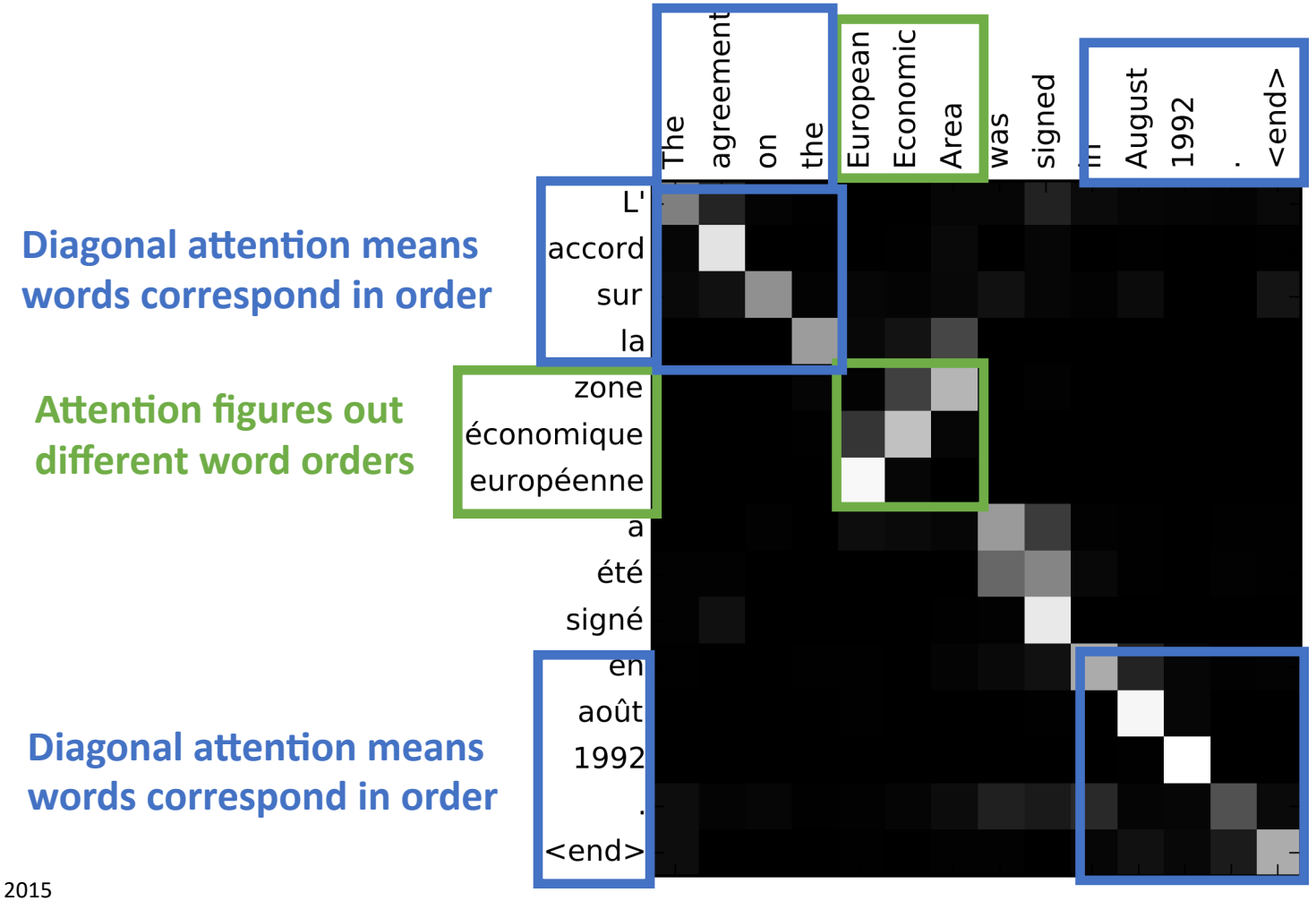
Sequence-to-Sequence with RNNs and Attention

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**."

Visualize attention weights $a_{t,i}$



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

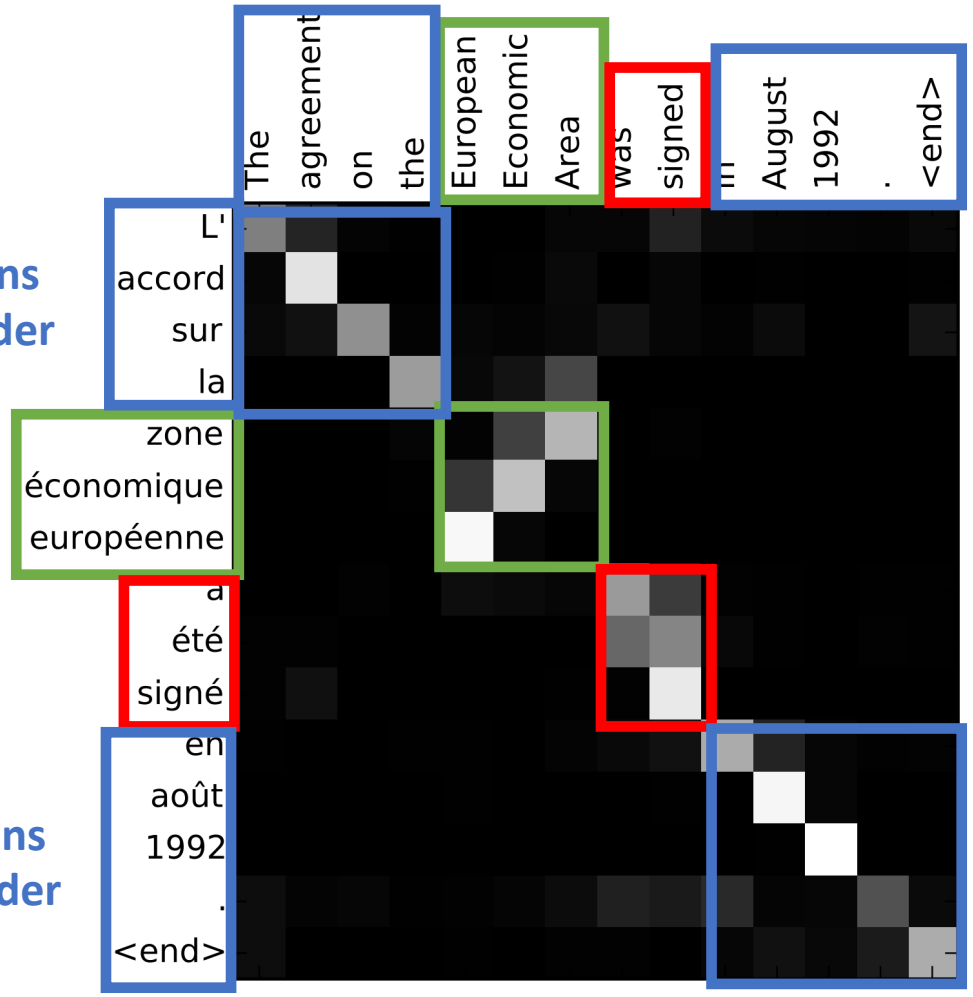
Sequence-to-Sequence with RNNs **and Attention**

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**."

Visualize attention weights $a_{t,i}$

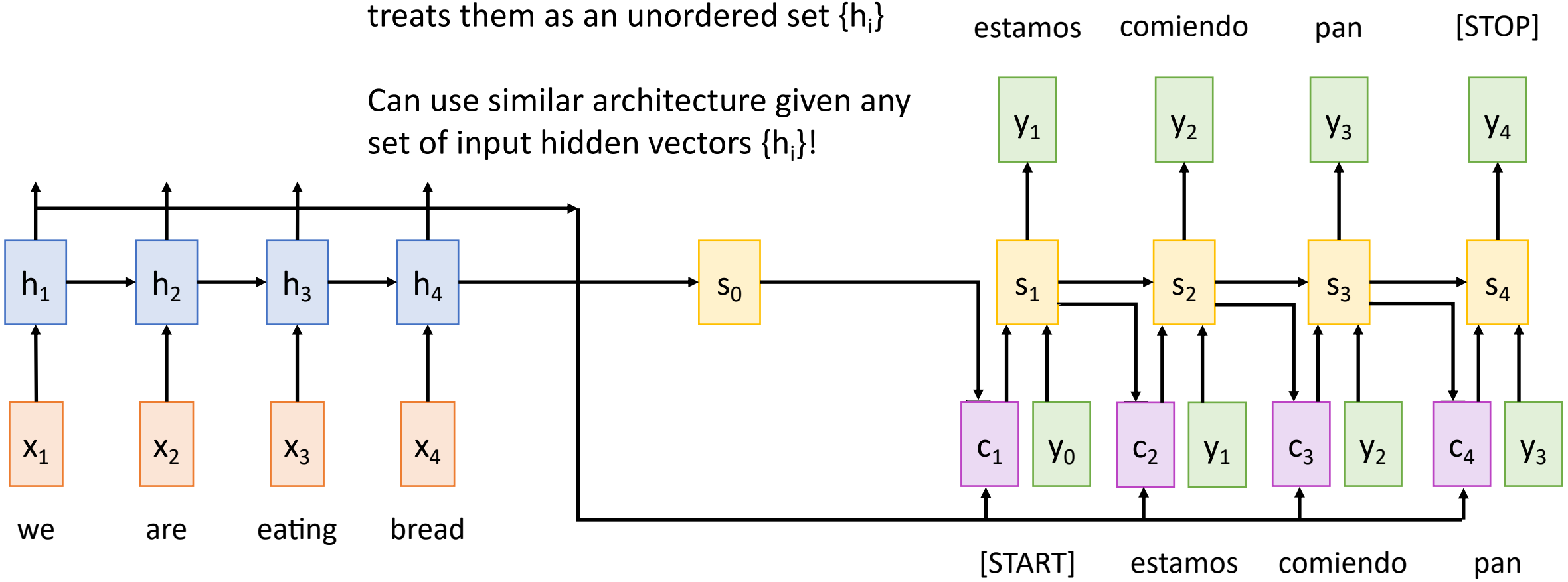


Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Sequence-to-Sequence with RNNs **and Attention**

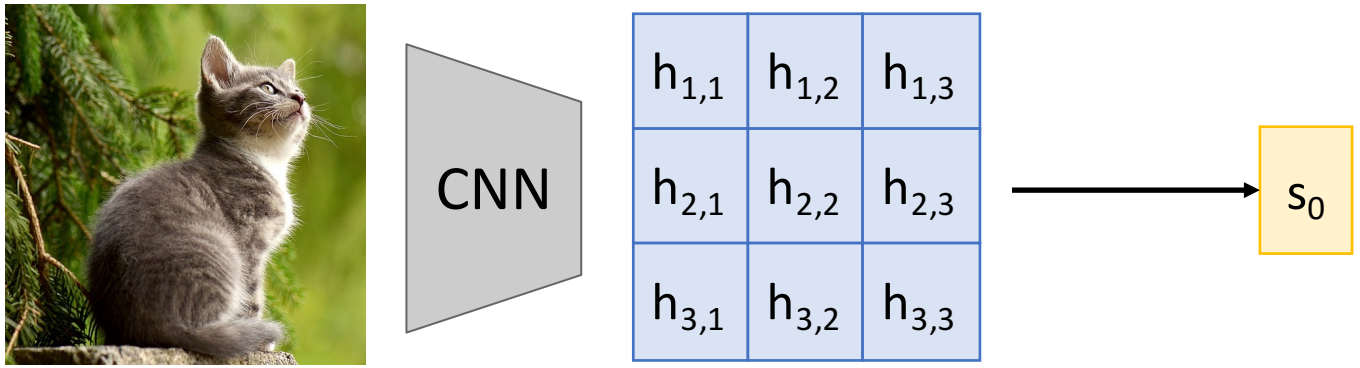
The decoder doesn't use the fact that h_i form an ordered sequence – it just treats them as an unordered set $\{h_i\}$

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



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

Image Captioning with RNNs and Attention

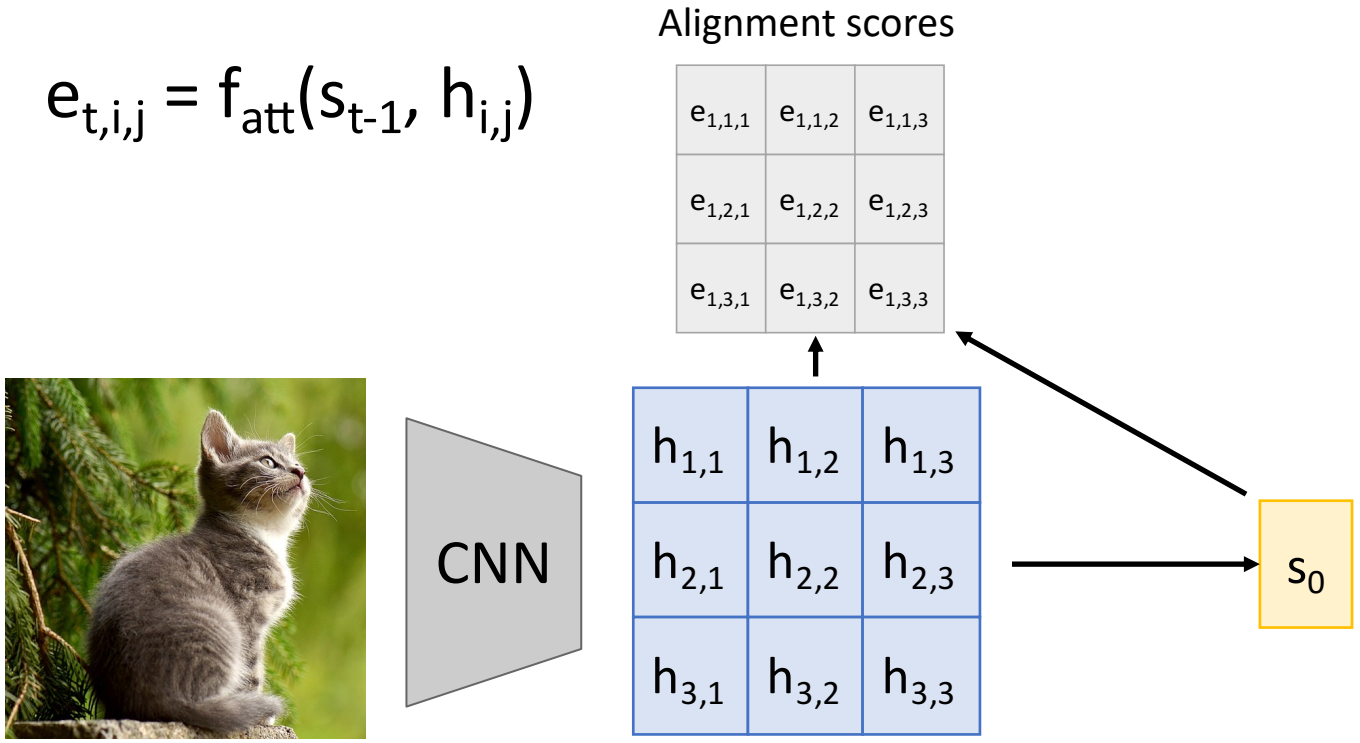


Use a CNN to compute a grid of features for an image

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

Image Captioning with RNNs and Attention

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



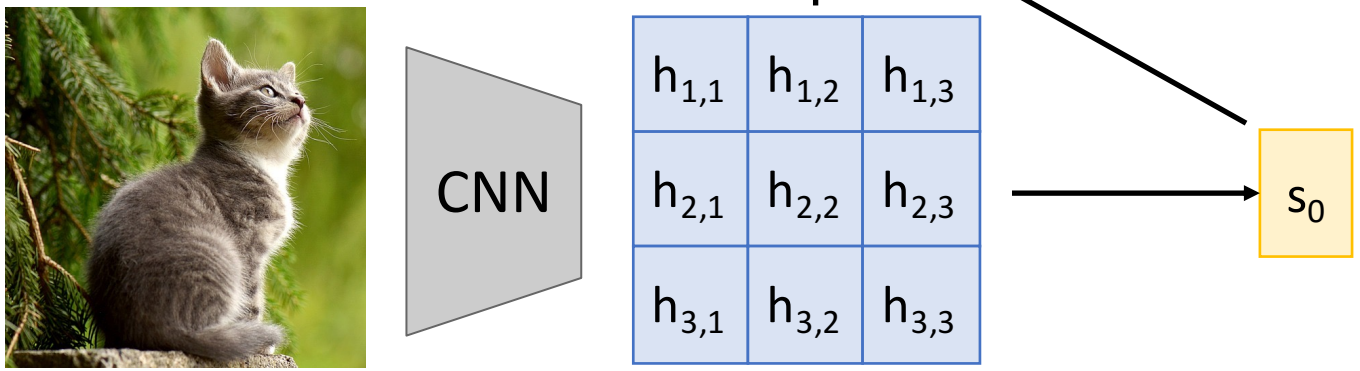
Use a CNN to compute a grid of features for an image

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

Image Captioning with RNNs and Attention

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

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



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

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

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

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

CNN

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

Attention weights

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

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

s_0

c_1

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

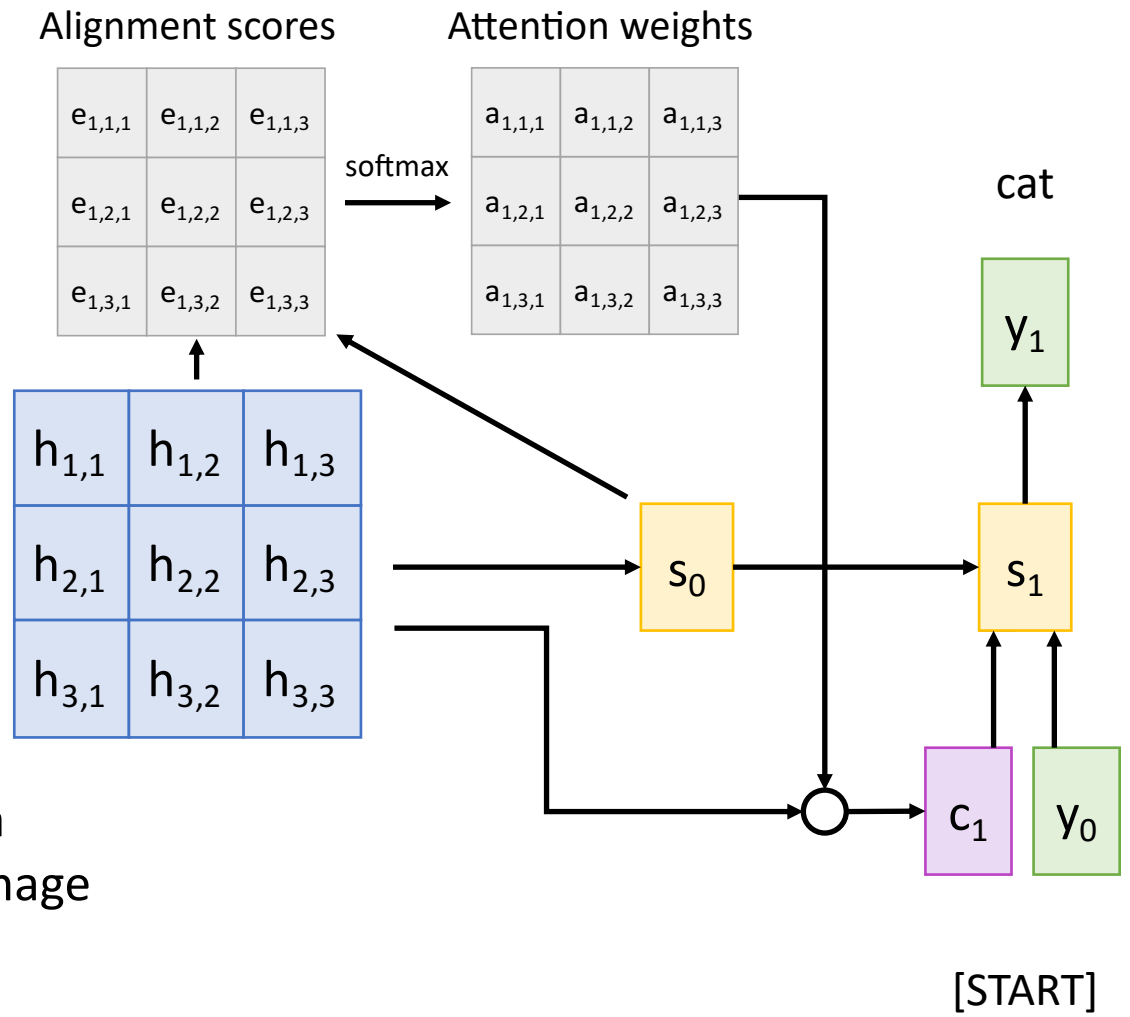
$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$

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

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

CNN

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



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

Image Captioning with RNNs and Attention

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

CNN

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

s_0 s_1

c_1 y_0 y_1

cat

[START]

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

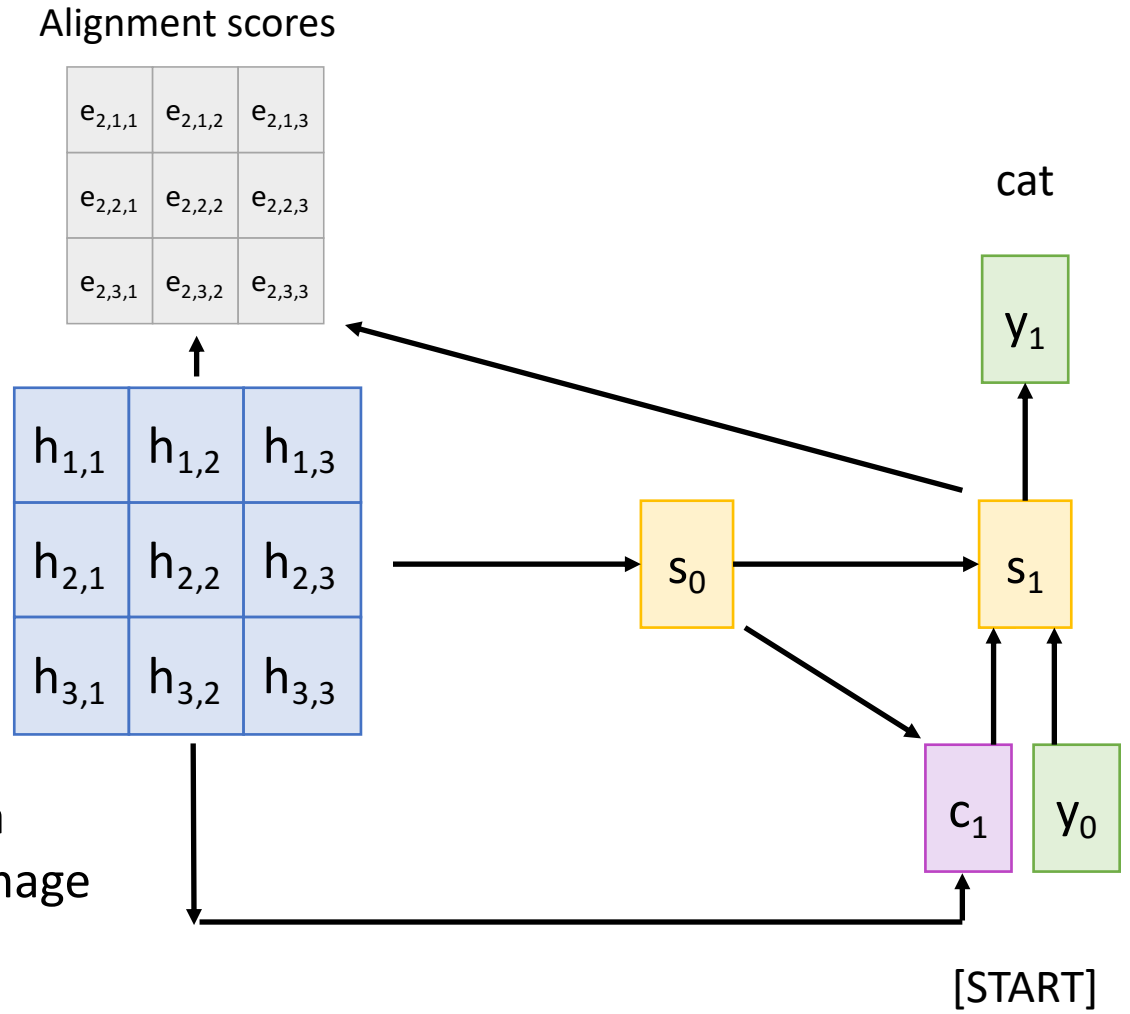
$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$

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

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

CNN

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



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

Image Captioning with RNNs and Attention

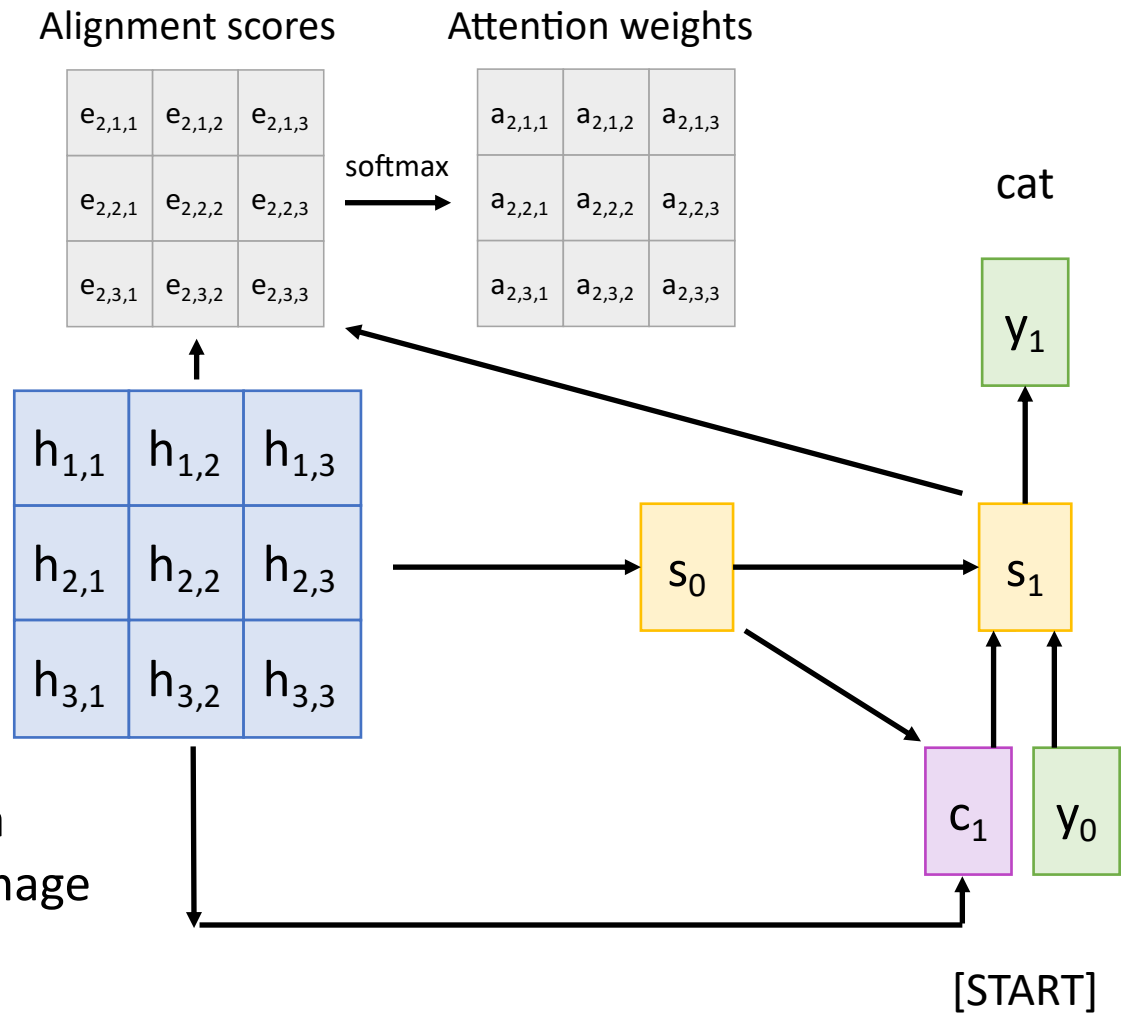
$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$

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

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

CNN

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



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

Image Captioning with RNNs and Attention

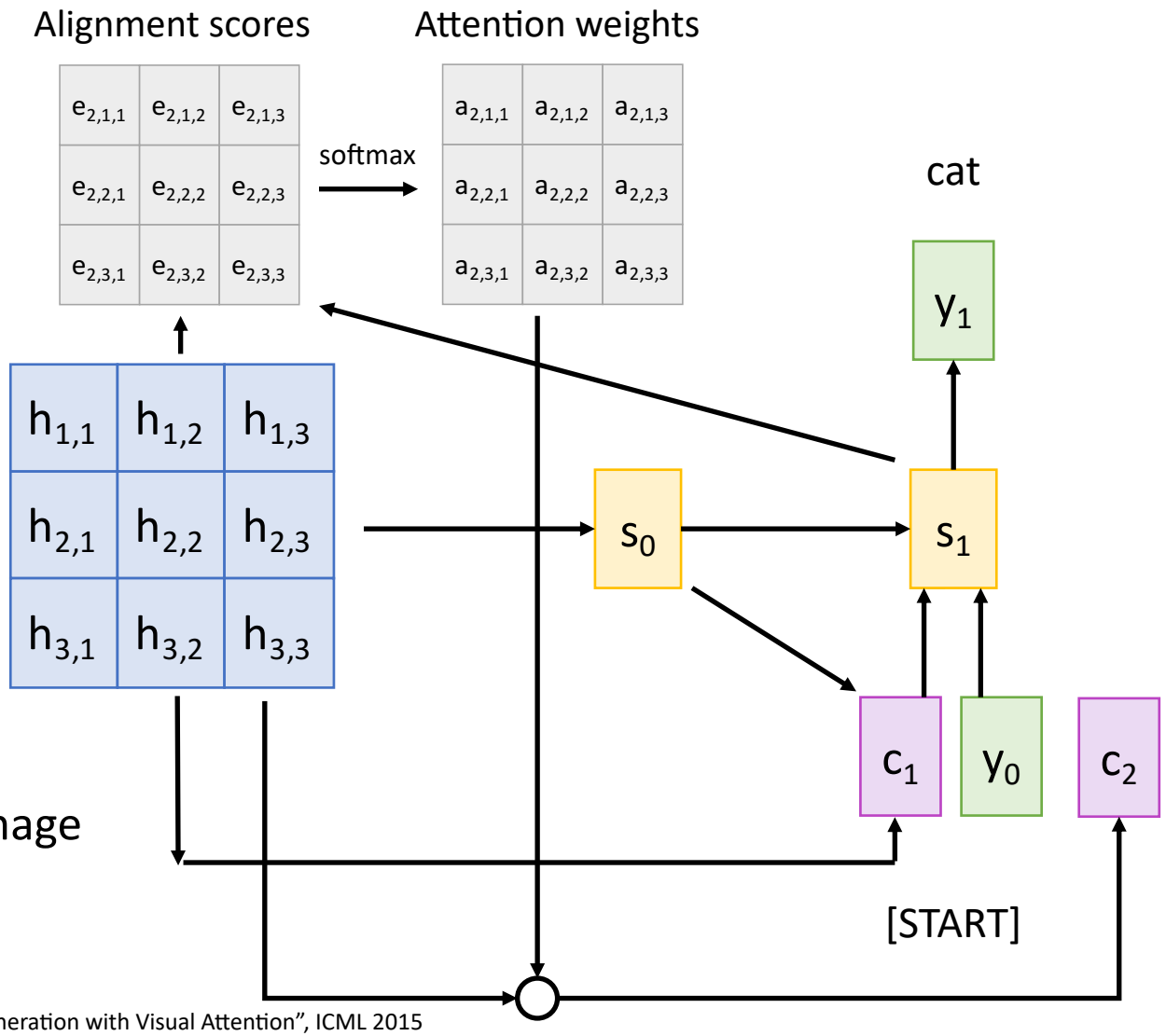
$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$

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

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

CNN

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



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

Image Captioning with RNNs and Attention

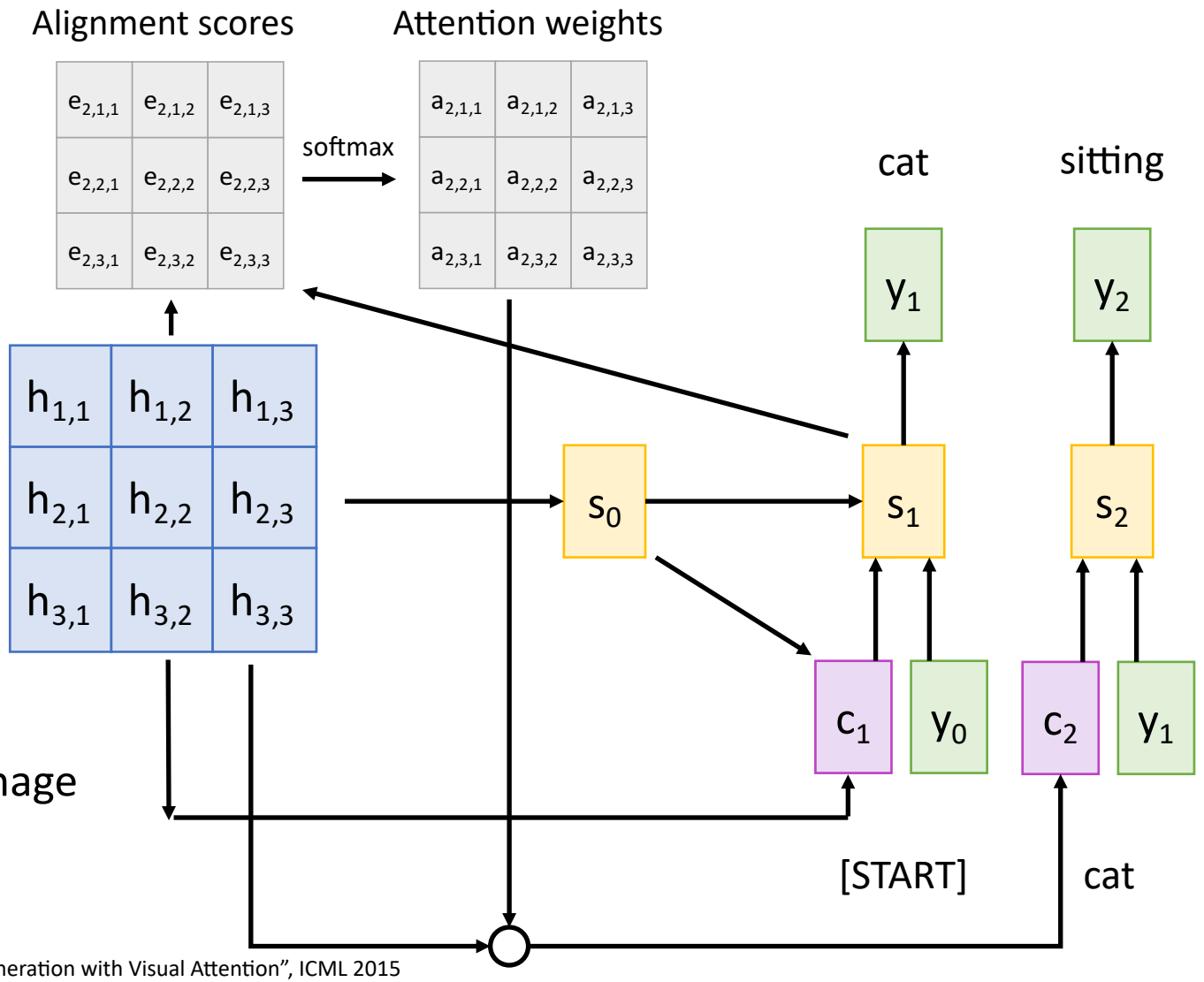
$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$

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

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

CNN

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



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

Image Captioning with RNNs and Attention

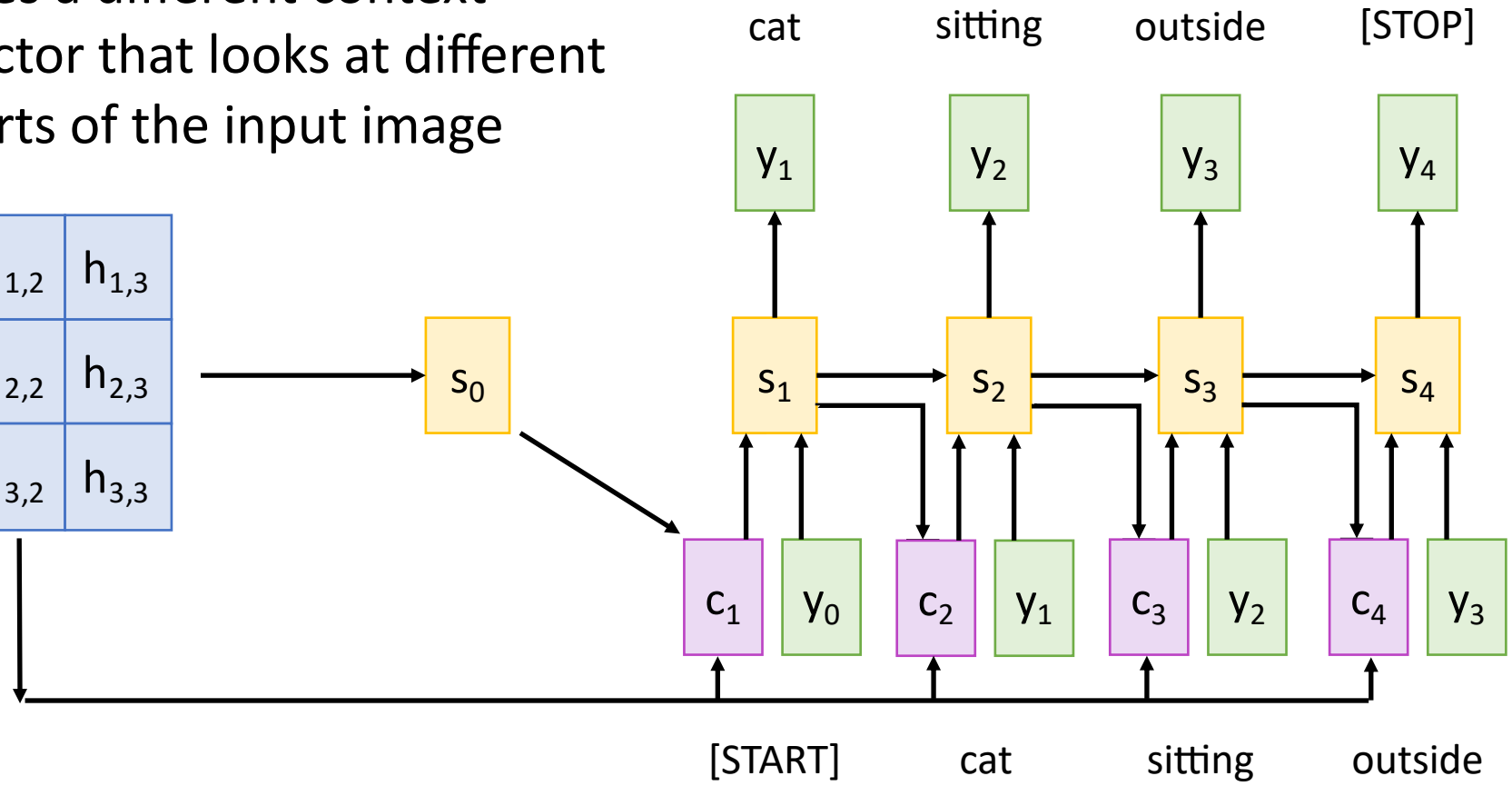
$e_{t,i,j} = f_{att}(s_{t-1}, h_{i,j})$
$a_{t,:,:} = \text{softmax}(e_{t,:,:})$
$c_t = \sum_{i,j} a_{t,i,j} h_{i,j}$

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

CNN

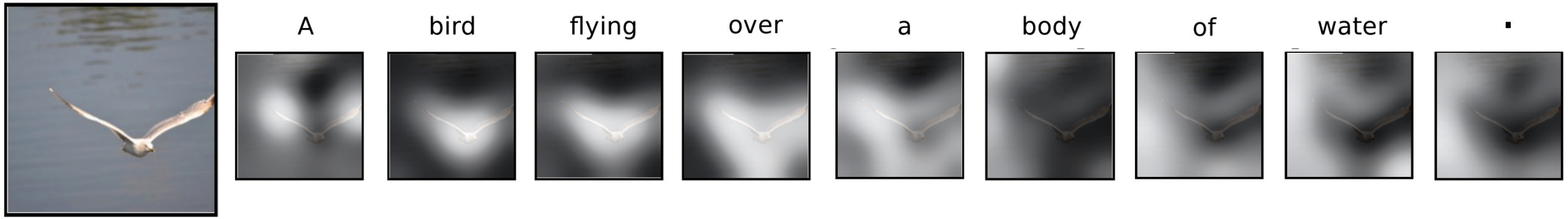
$h_{1,1}$ $h_{1,2}$ $h_{1,3}$
$h_{2,1}$ $h_{2,2}$ $h_{2,3}$
$h_{3,1}$ $h_{3,2}$ $h_{3,3}$

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



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

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.

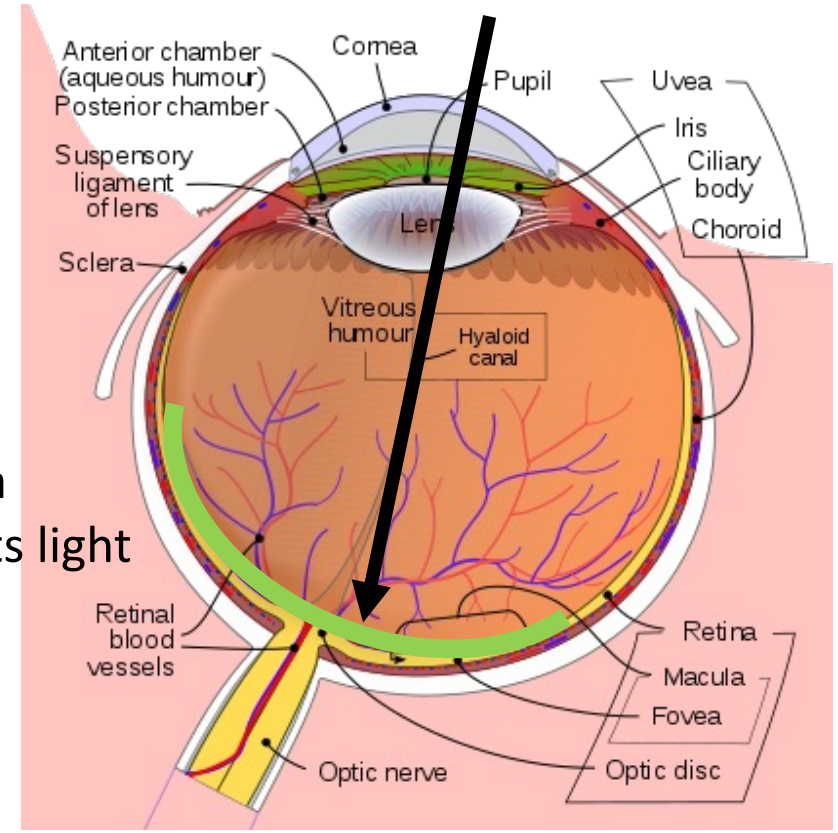
A group of people sitting on a boat in the water.

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

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

Human Vision: Fovea

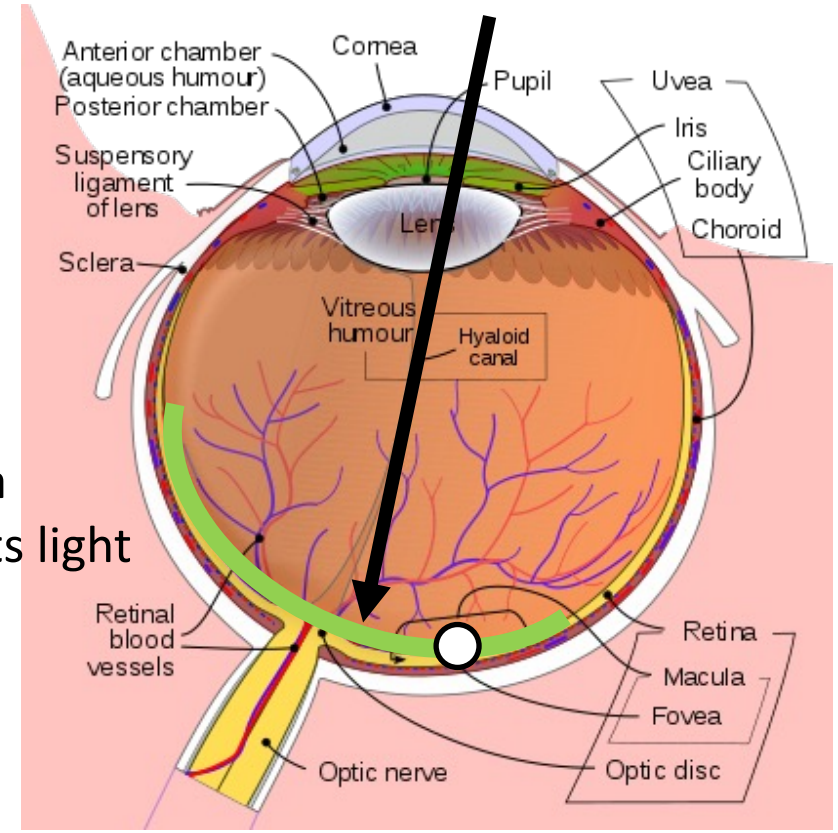
Light enters eye



Retina detects light

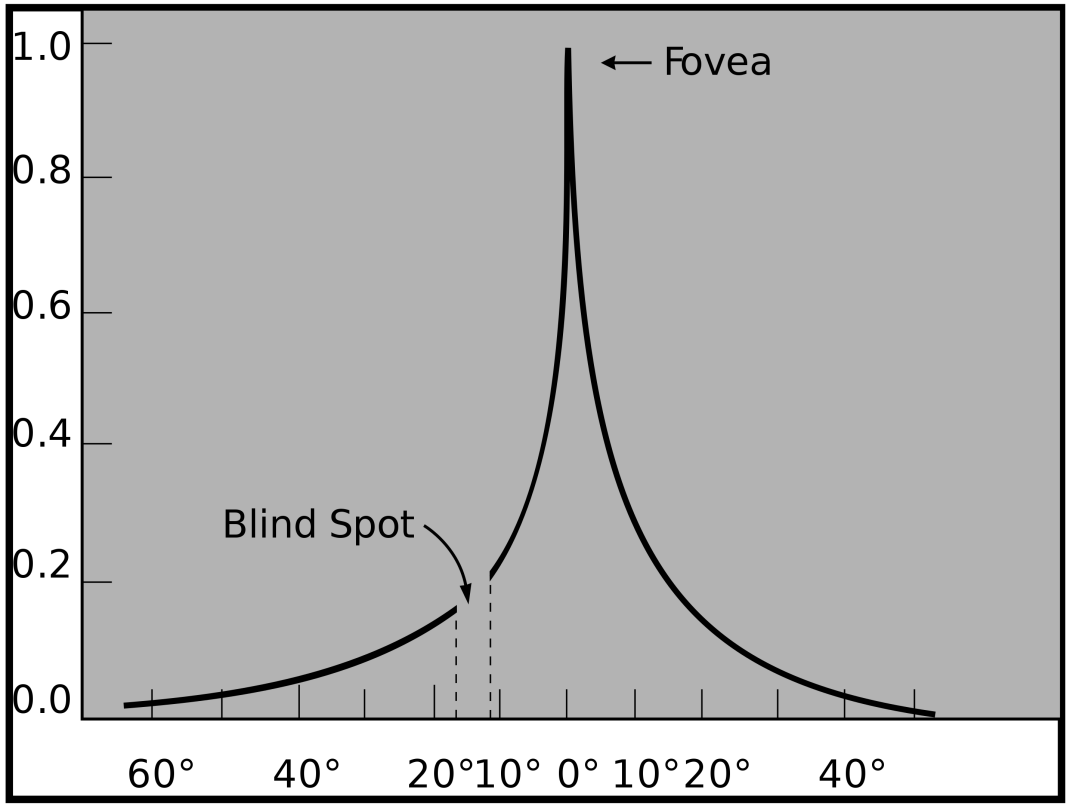
Human Vision: Fovea

Light enters eye



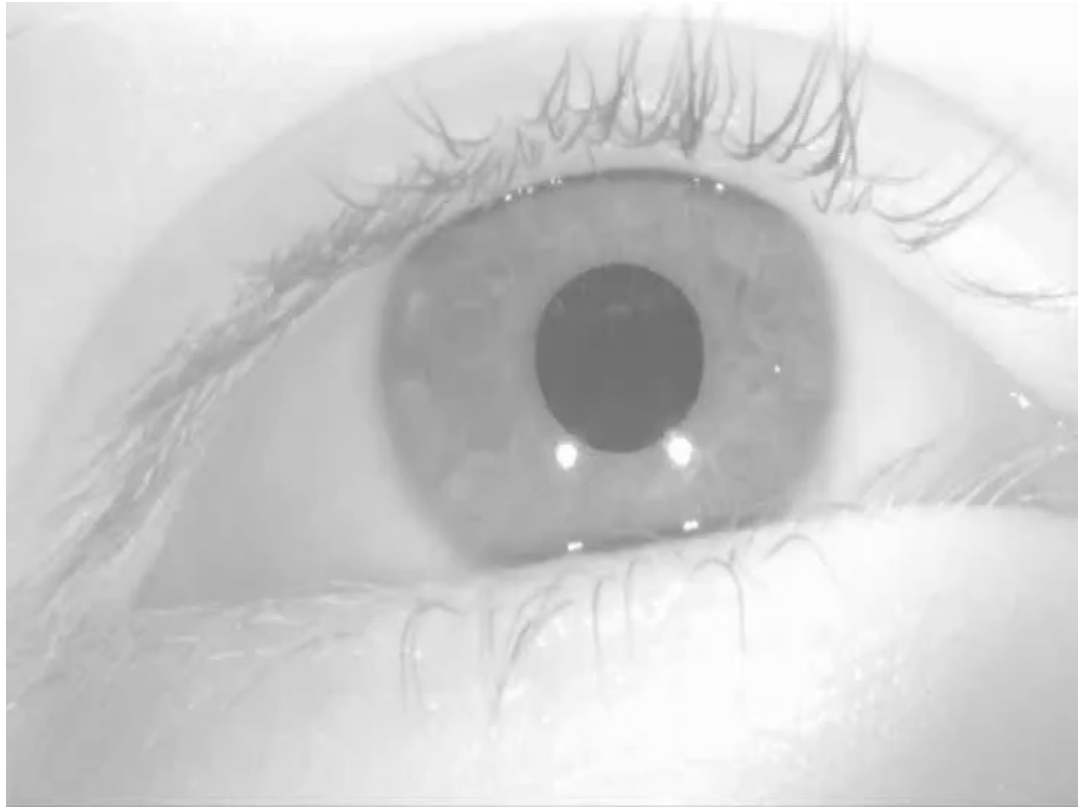
Retina detects light

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

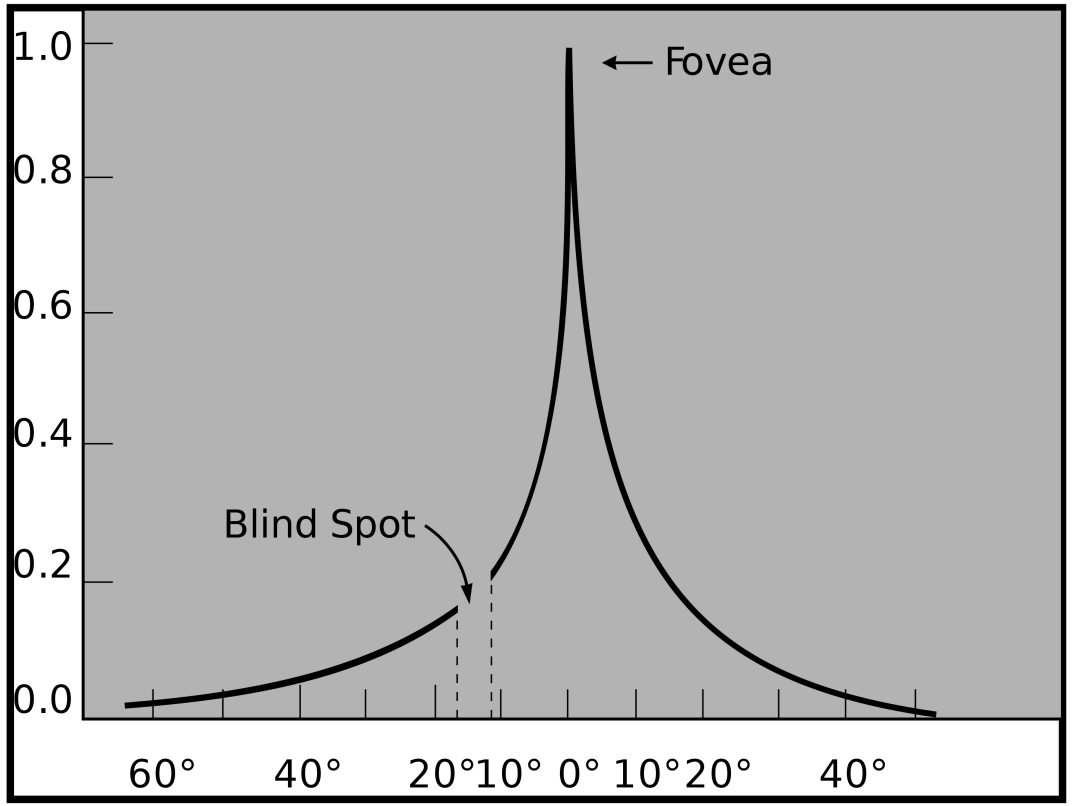
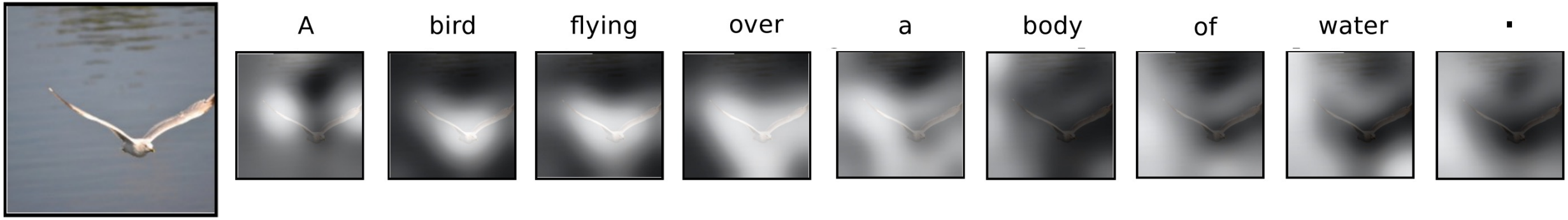
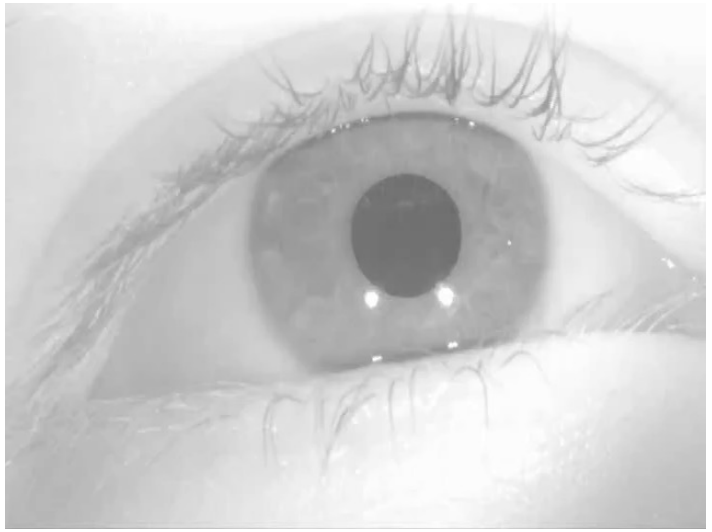


Image Captioning with RNNs and Attention



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



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

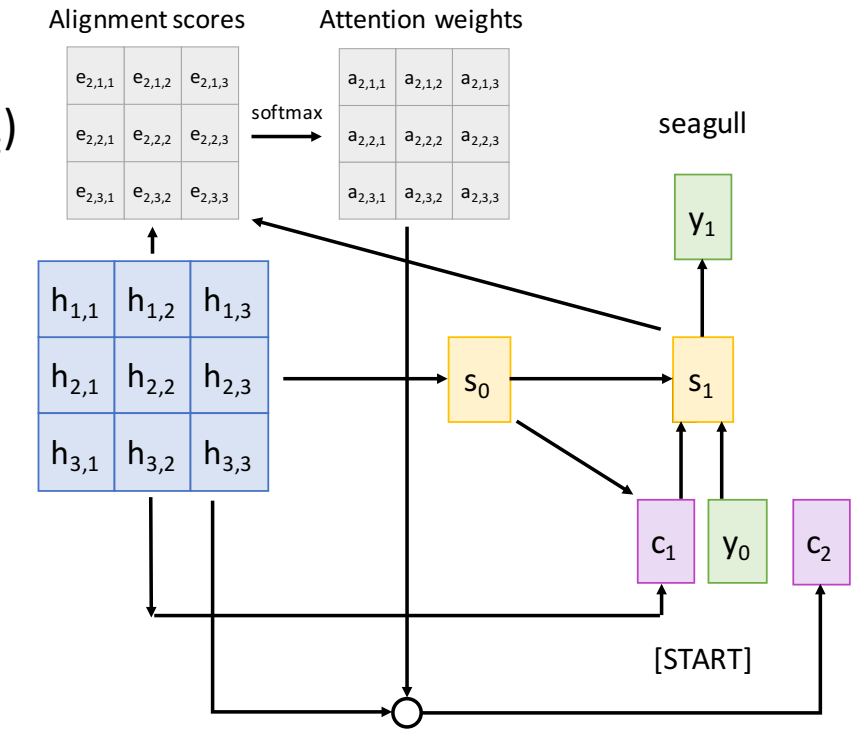
"**Show, attend, and interact**" *(Qureshi et al, ICRA 2017)*
Process image, attend to image regions, output robot control commands

"**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}

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

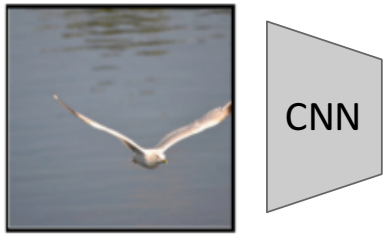


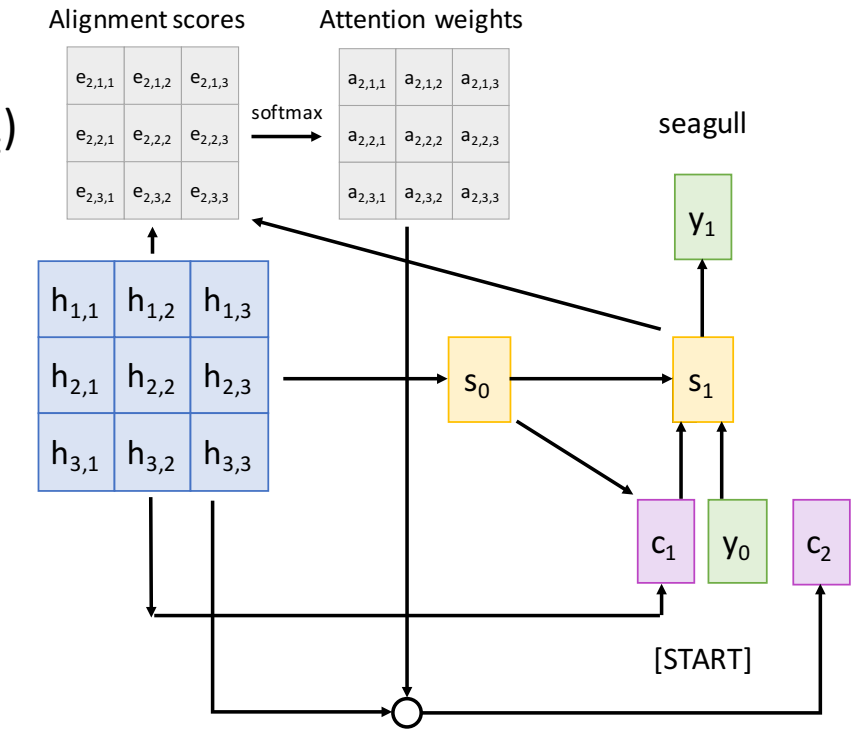
Computation:
Similarities: e (Shape: N_X) $e_i = f_{att}(\mathbf{q}, \mathbf{X}_i)$
Attention weights: a = softmax(e) (Shape: N_X)
Output vector: $y = \sum_i a_i \mathbf{X}_i$ (Shape: D_X)

Attention Layer

Inputs:
Query vector: **q** (Shape: D_Q)
Input vectors: **X** (Shape: N_X x D_Q)
Similarity function: dot product

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





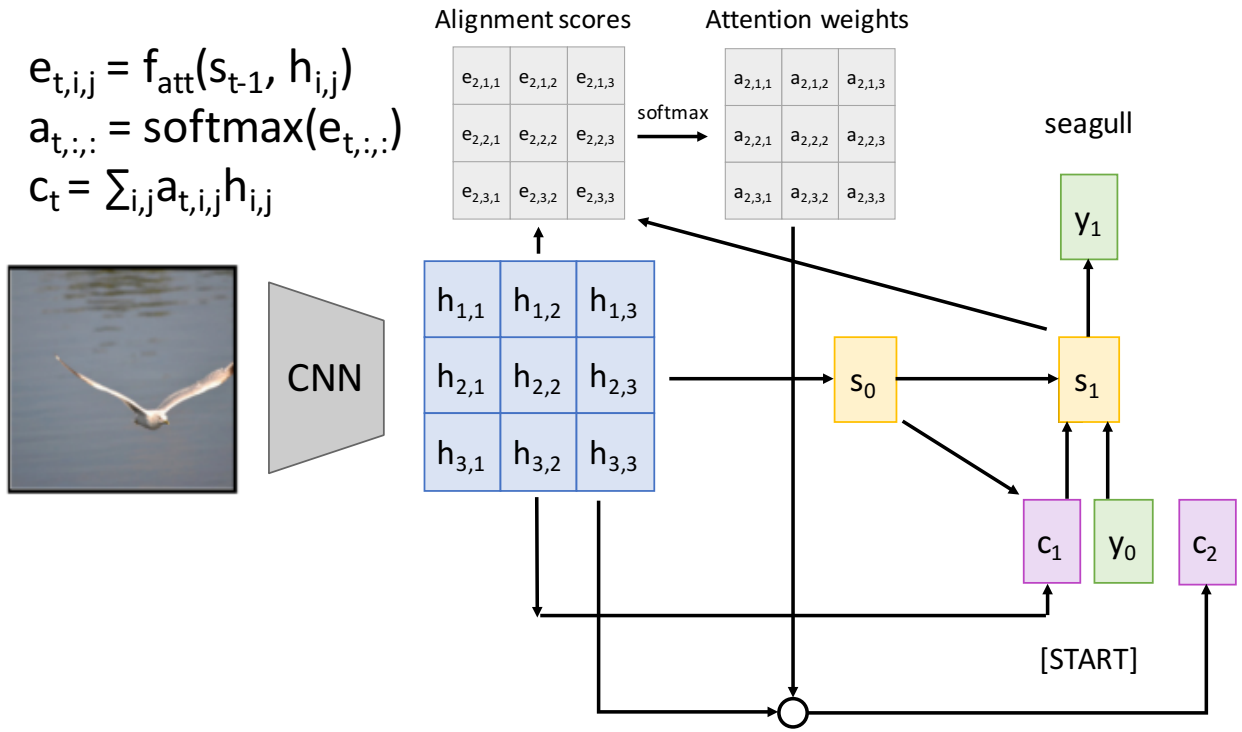
Computation:
Similarities: e (Shape: N_X) $e_i = \mathbf{q} \cdot \mathbf{X}_i$
Attention weights: a = softmax(e) (Shape: N_X)
Output vector: $y = \sum_i a_i \mathbf{X}_i$ (Shape: D_X)

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: scaled dot product

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



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

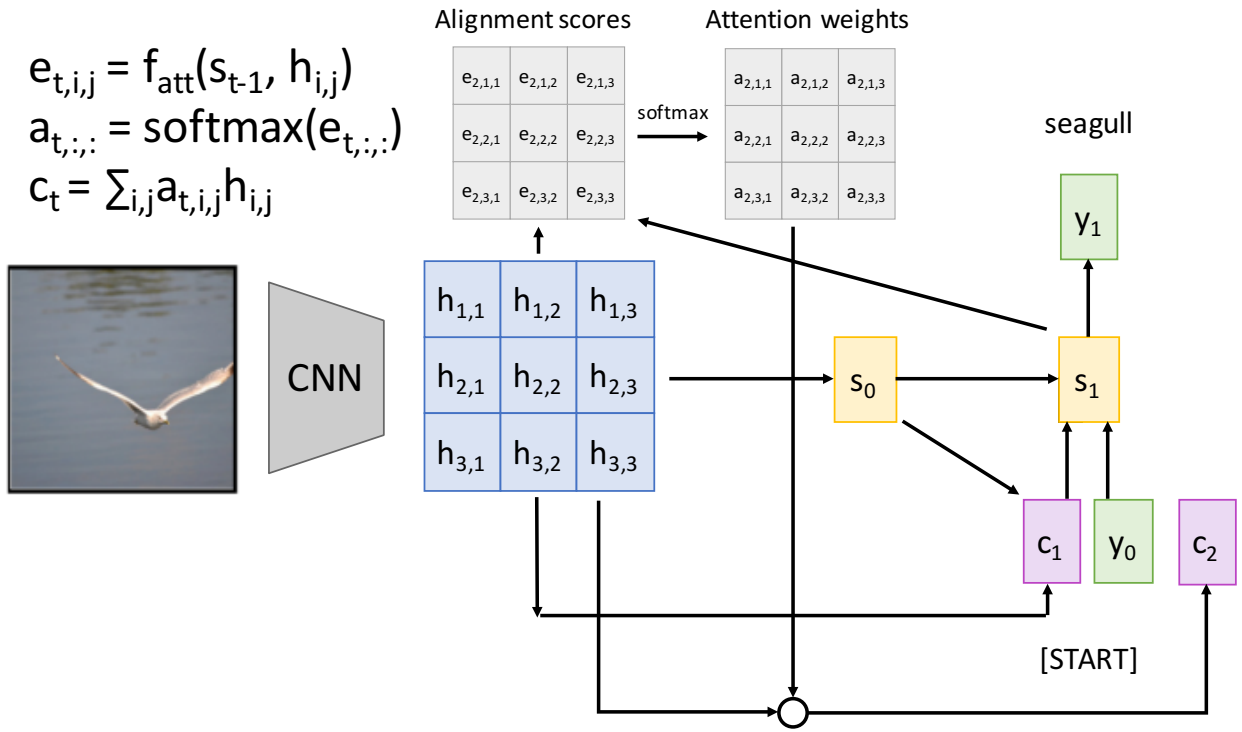
Then $|a| = (\sum_i a^2)^{1/2} = a\sqrt{D}$

Computation:

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

Attention weights: a = softmax(e) (Shape: N_X)

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



Changes:

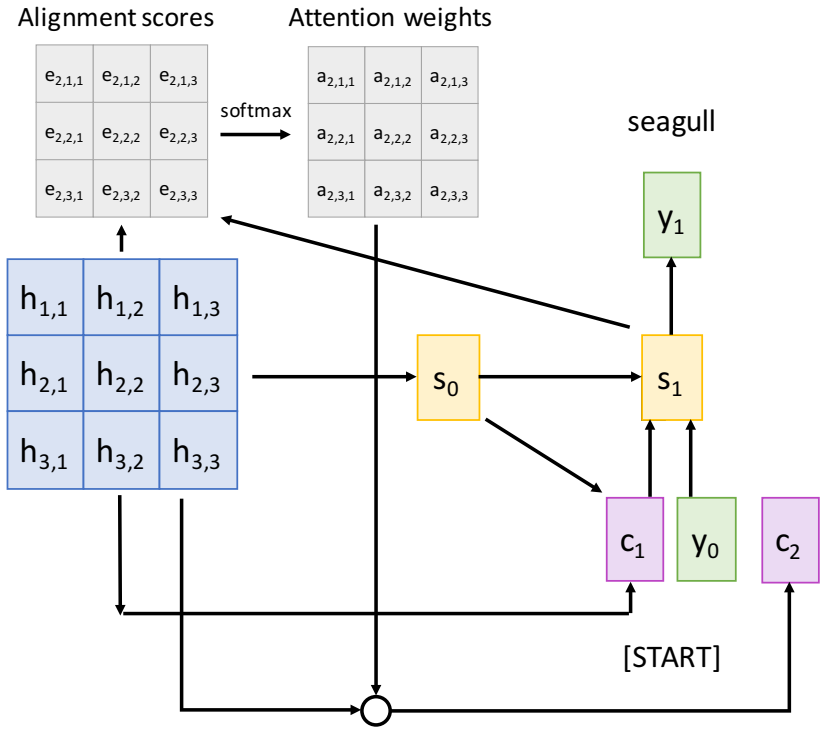
- Use **scaled** dot product for similarity

Attention Layer

Inputs:
Query vectors: Q (Shape: N_Q x D_Q)
Input vectors: X (Shape: N_X x D_Q)

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

CNN



Computation:
Similarities: $E = QX^T/\sqrt{D_Q}$ (Shape: N_Q x N_X) $E_{i,j} = (Q_i \cdot X_j)/\sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_Q x N_X)
Output vectors: Y = AX (Shape: N_Q x 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: 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: $E = QK^T / \sqrt{D_Q}$ (Shape: N_Q x N_X) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$

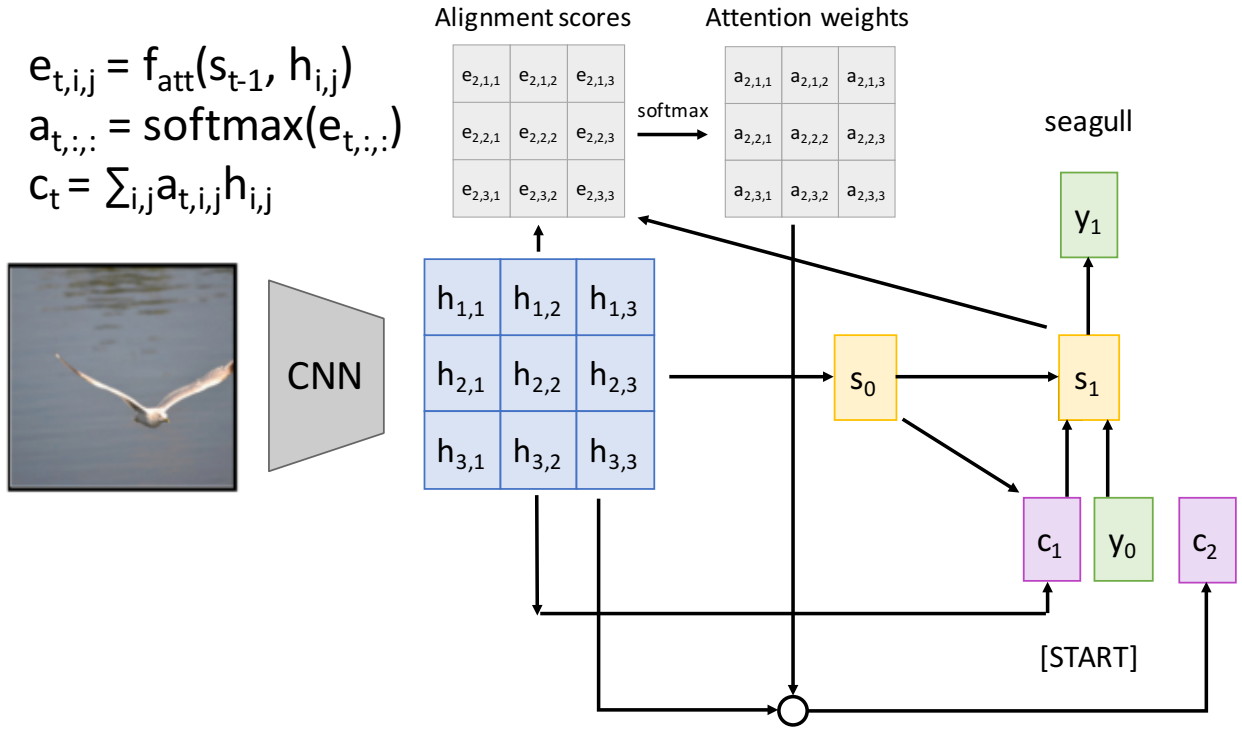
Attention weights: A = softmax(E, dim=1) (Shape: N_Q x N_X)

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

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

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

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



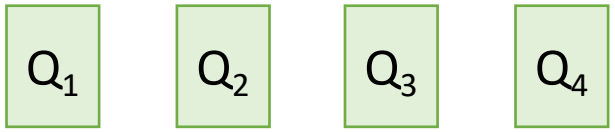
Changes:

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

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)

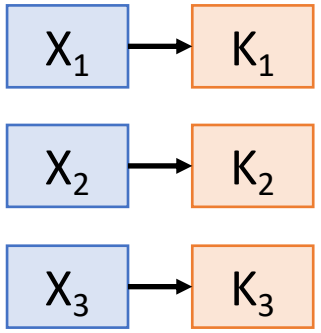
Computation:
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value Vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T / \sqrt{D_Q}$ (Shape: N_Q x N_X) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_Q x N_X)
Output vectors: Y = AV (Shape: N_Q x D_V)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

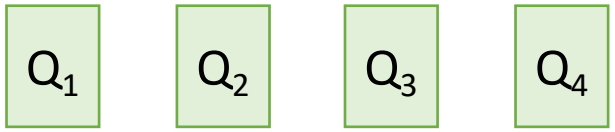


Attention Layer

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

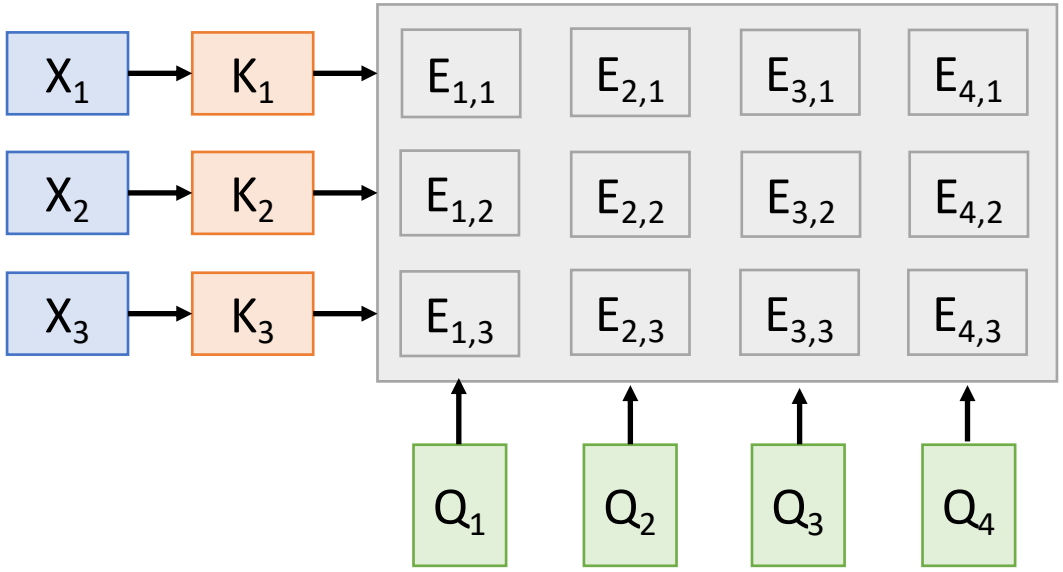




Attention Layer

Inputs:
Query vectors: 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)

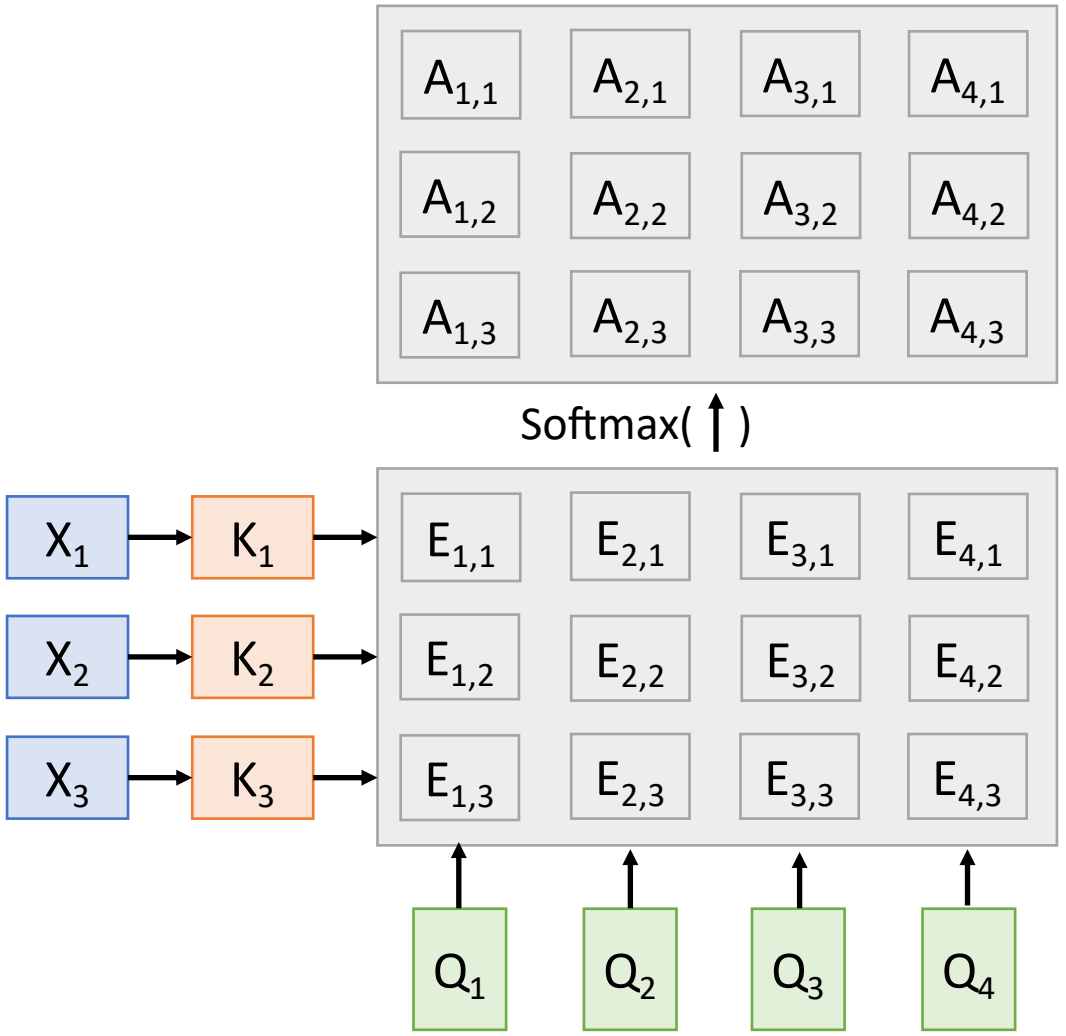
Computation:
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value Vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T / \sqrt{D_Q}$ (Shape: N_Q x N_X) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_Q x N_X)
Output vectors: Y = AV (Shape: N_Q x D_V)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



Attention Layer

Inputs:
Query vectors: 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)

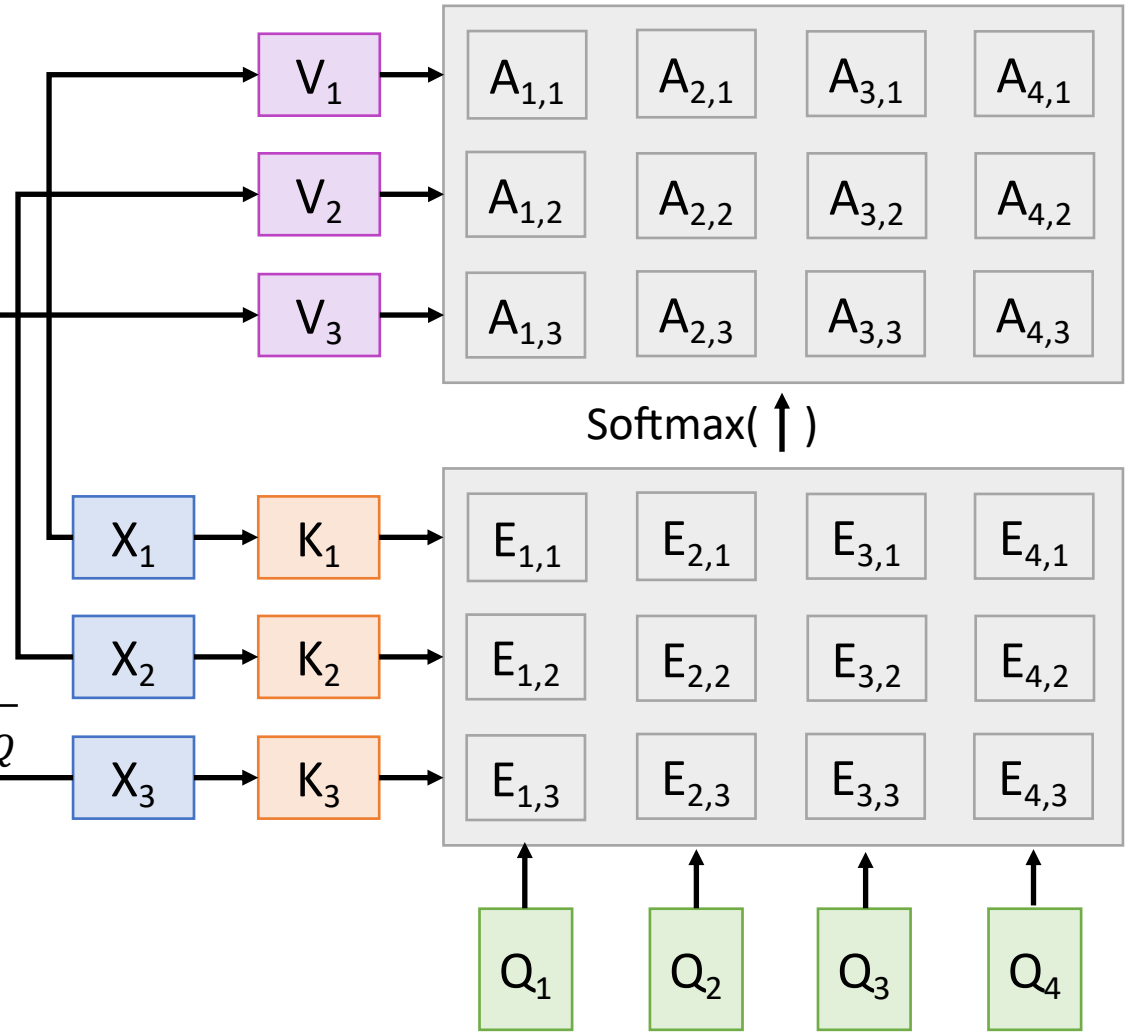
Computation:
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value Vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T / \sqrt{D_Q}$ (Shape: N_Q x N_X) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_Q x N_X)
Output vectors: Y = AV (Shape: N_Q x D_V)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



Attention Layer

Inputs:
Query vectors: 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)

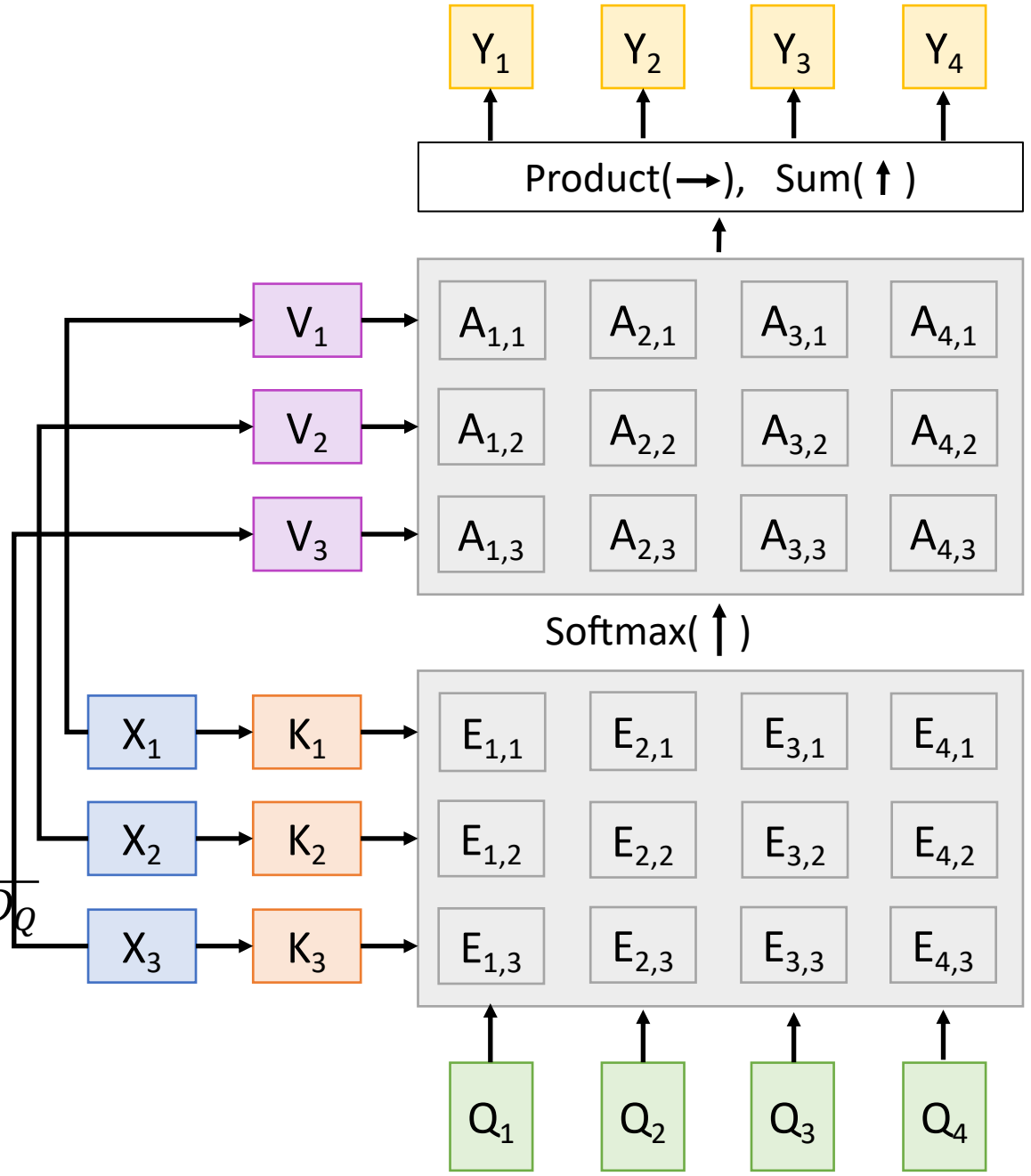
Computation:
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value Vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T / \sqrt{D_Q}$ (Shape: N_Q x N_X) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_Q x N_X)
Output vectors: Y = AV (Shape: N_Q x D_V)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



Attention Layer

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

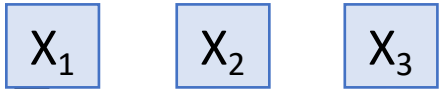


Self-Attention Layer

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

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: $\mathbf{K} = \mathbf{X}\mathbf{W_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{X}\mathbf{W_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{Q}\mathbf{K}^T / \sqrt{D_Q}$ (Shape: N_Q x N_X) $E_{i,j} = (\mathbf{Q}_i \cdot \mathbf{K}_j) / \sqrt{D_Q}$
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) $Y_i = \sum_j A_{i,j}\mathbf{V}_j$

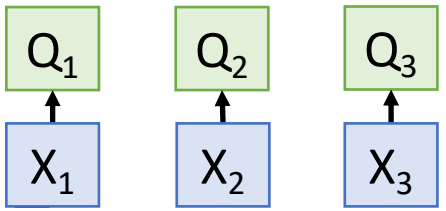


Self-Attention Layer

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

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

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

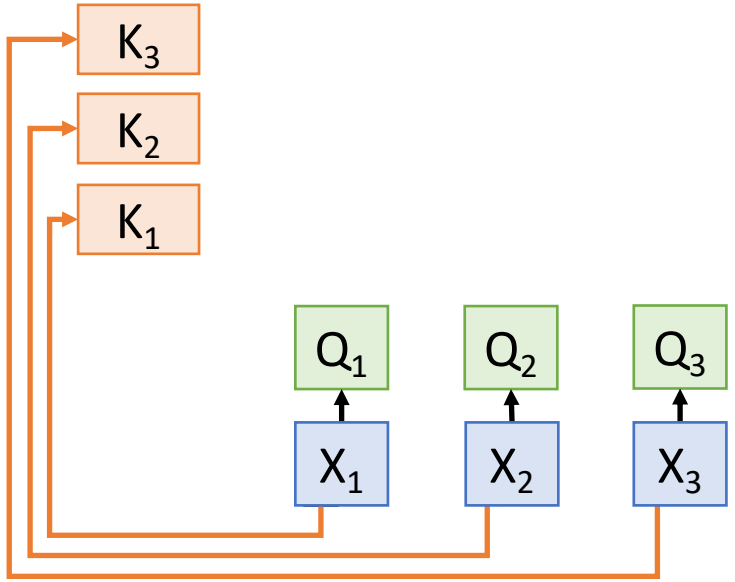


Self-Attention Layer

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

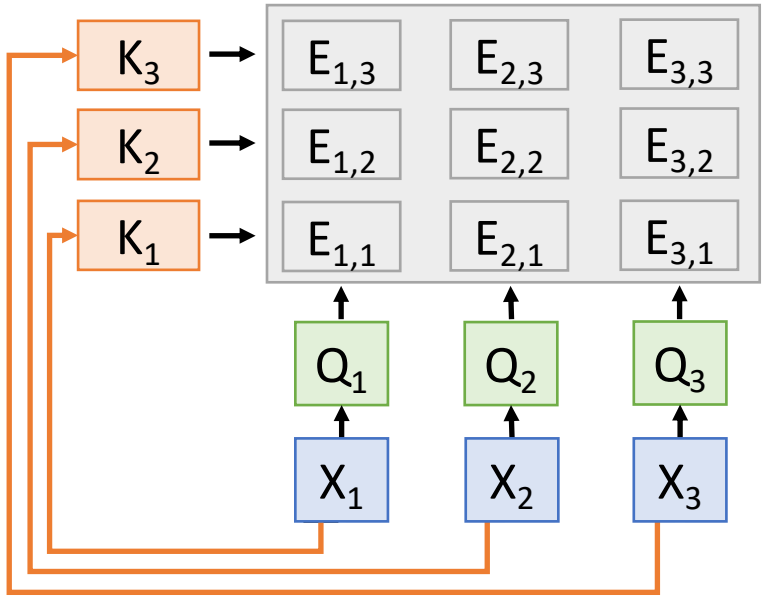


Self-Attention Layer

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

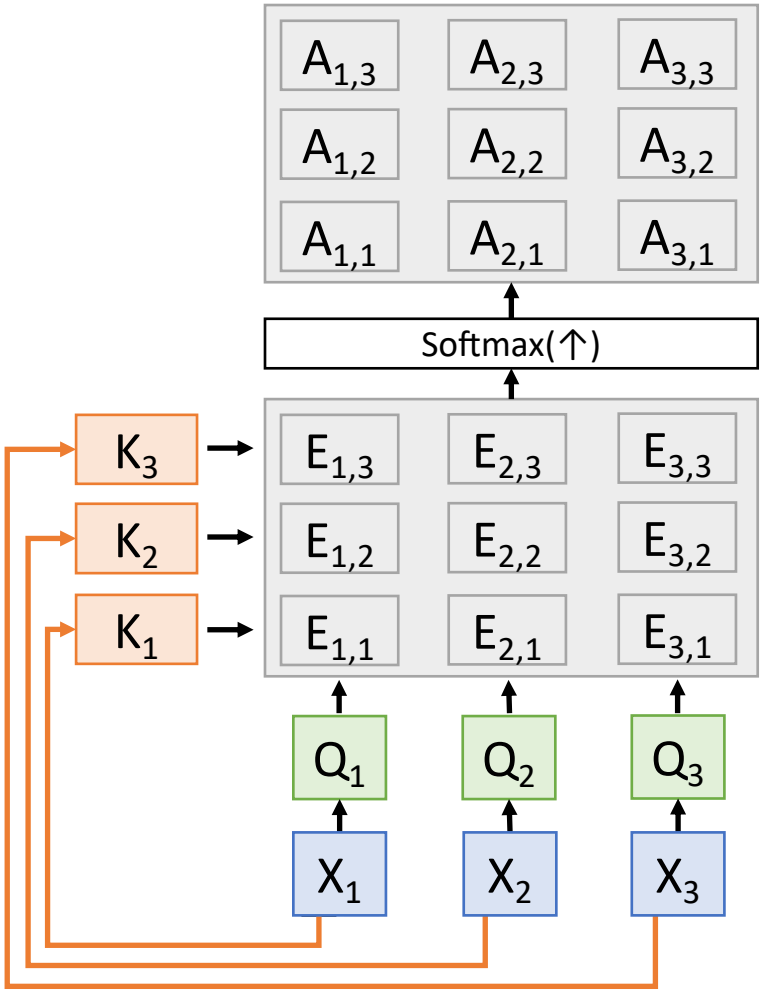


Self-Attention Layer

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

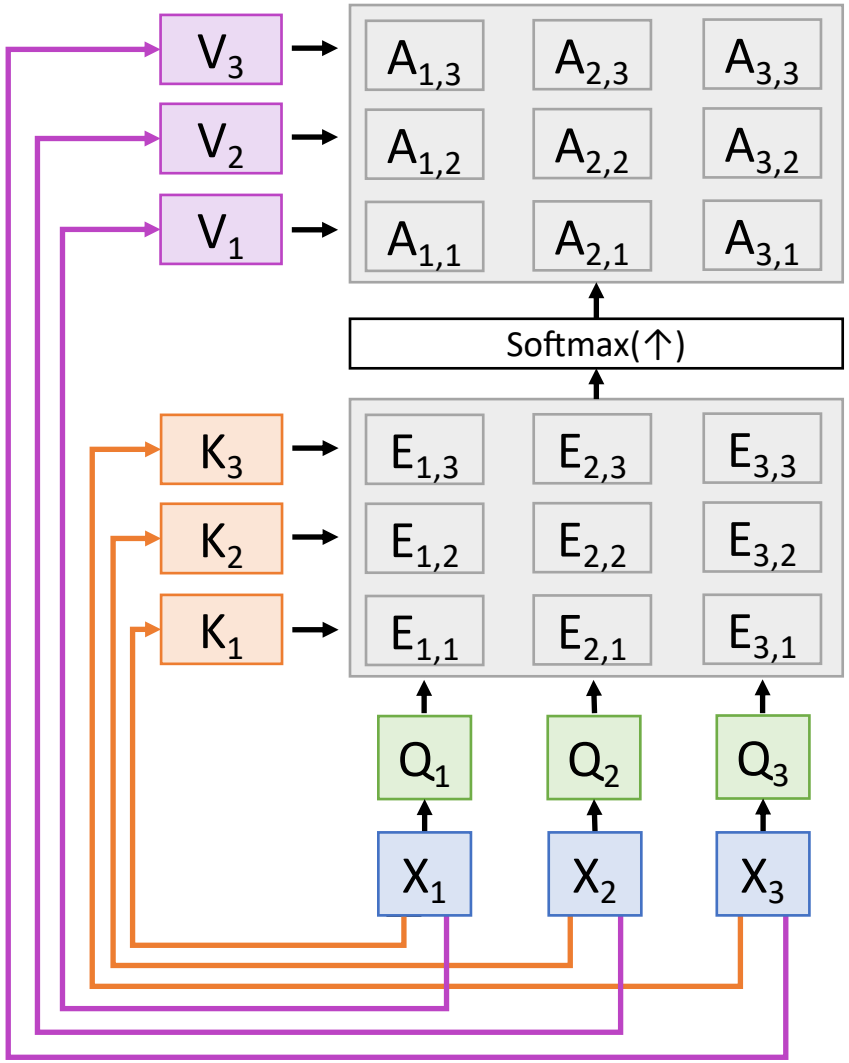


Self-Attention Layer

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

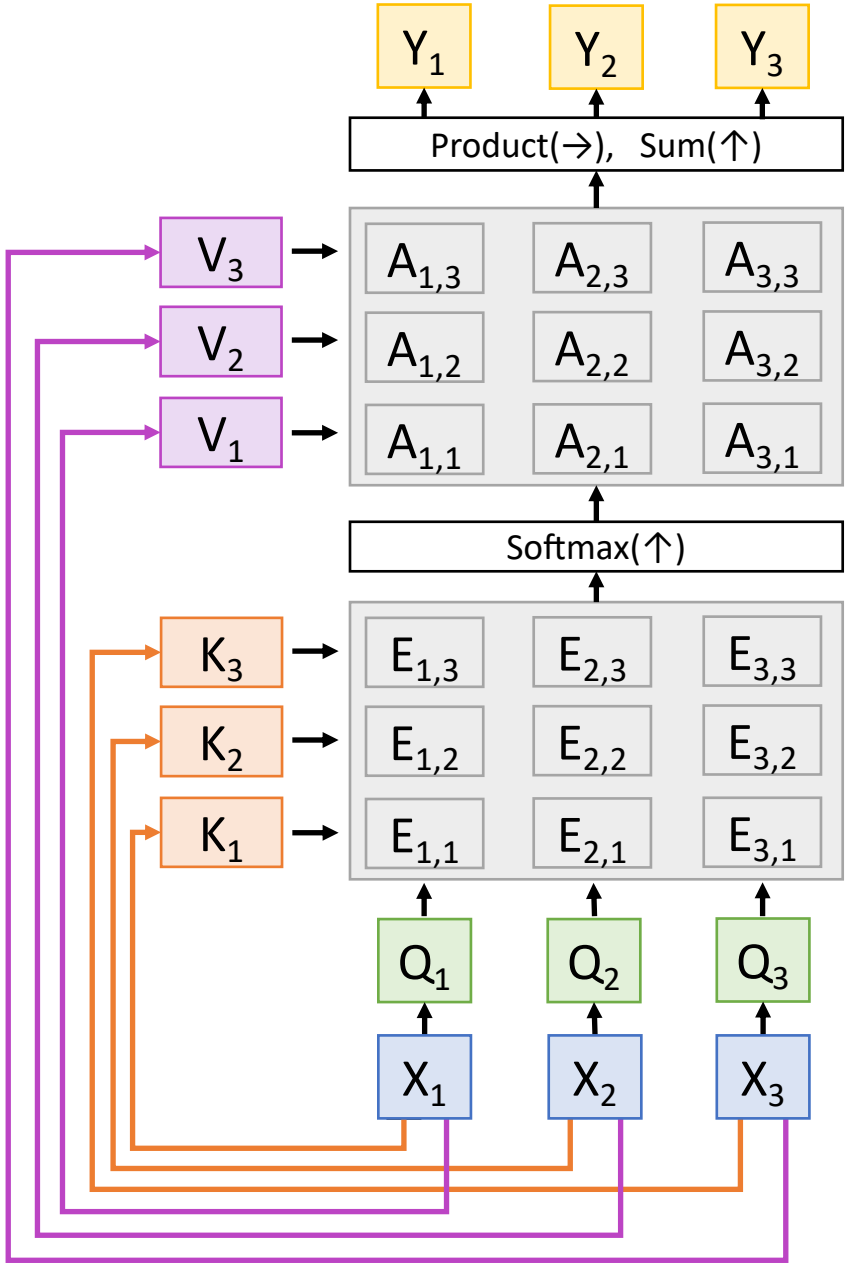


Self-Attention Layer

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

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

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

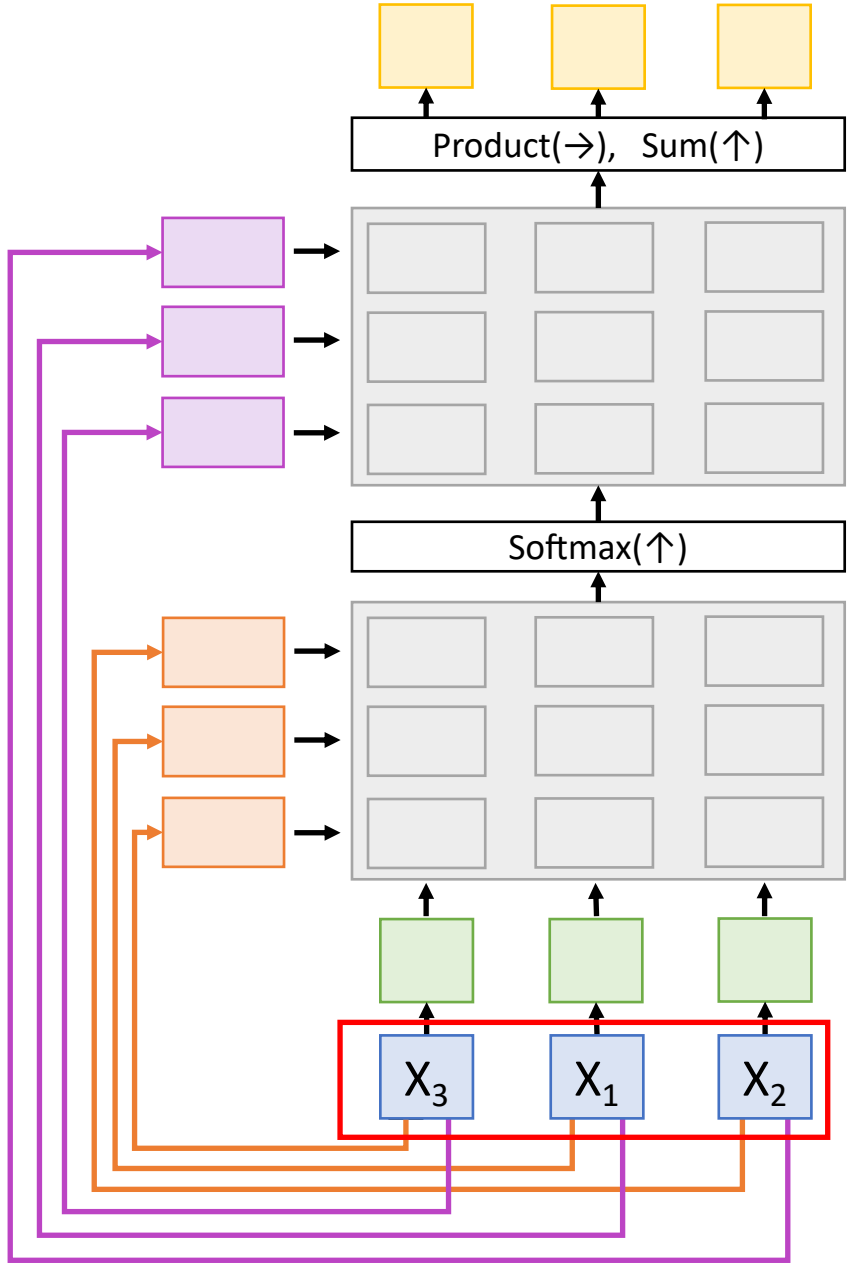


Self-Attention Layer

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)

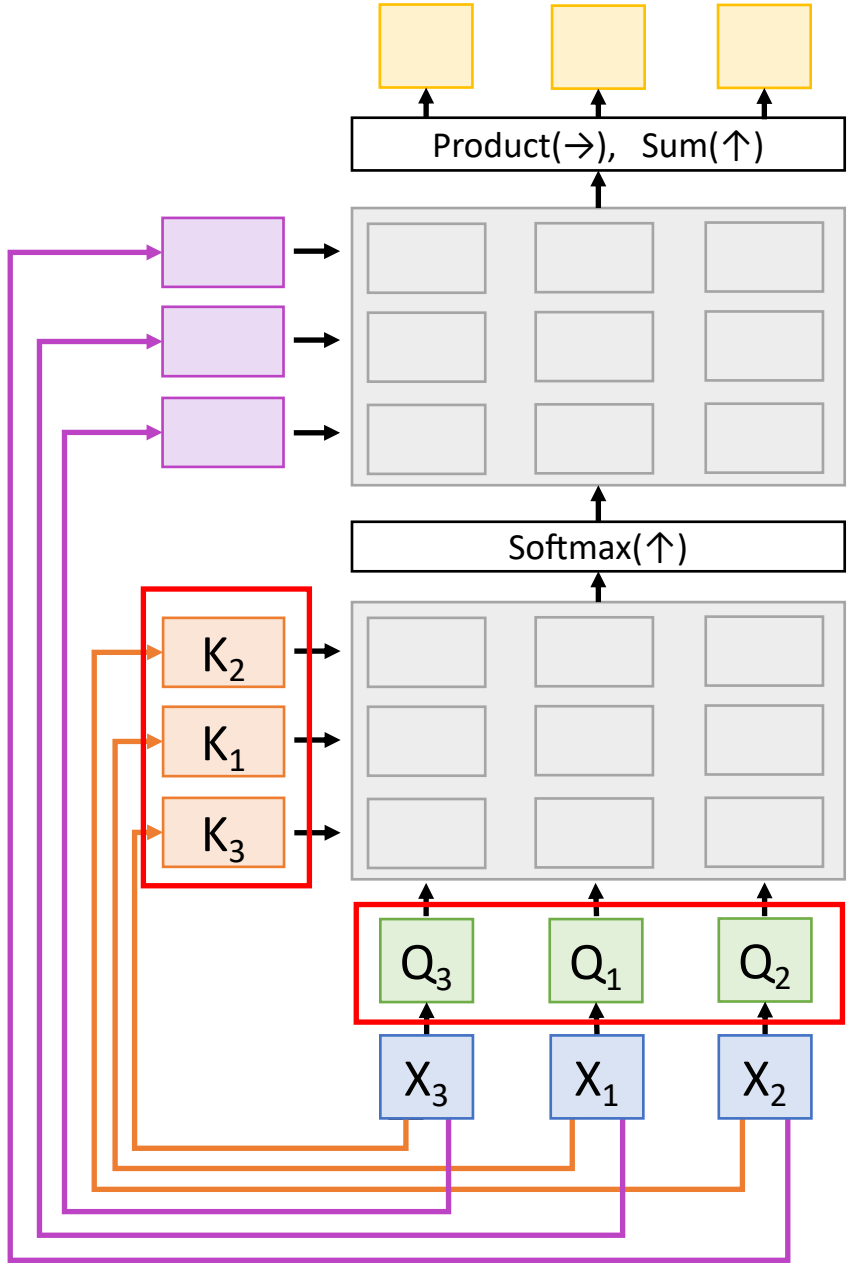
Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value Vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: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

Queries and Keys 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: $E = QK^T / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$

Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)

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

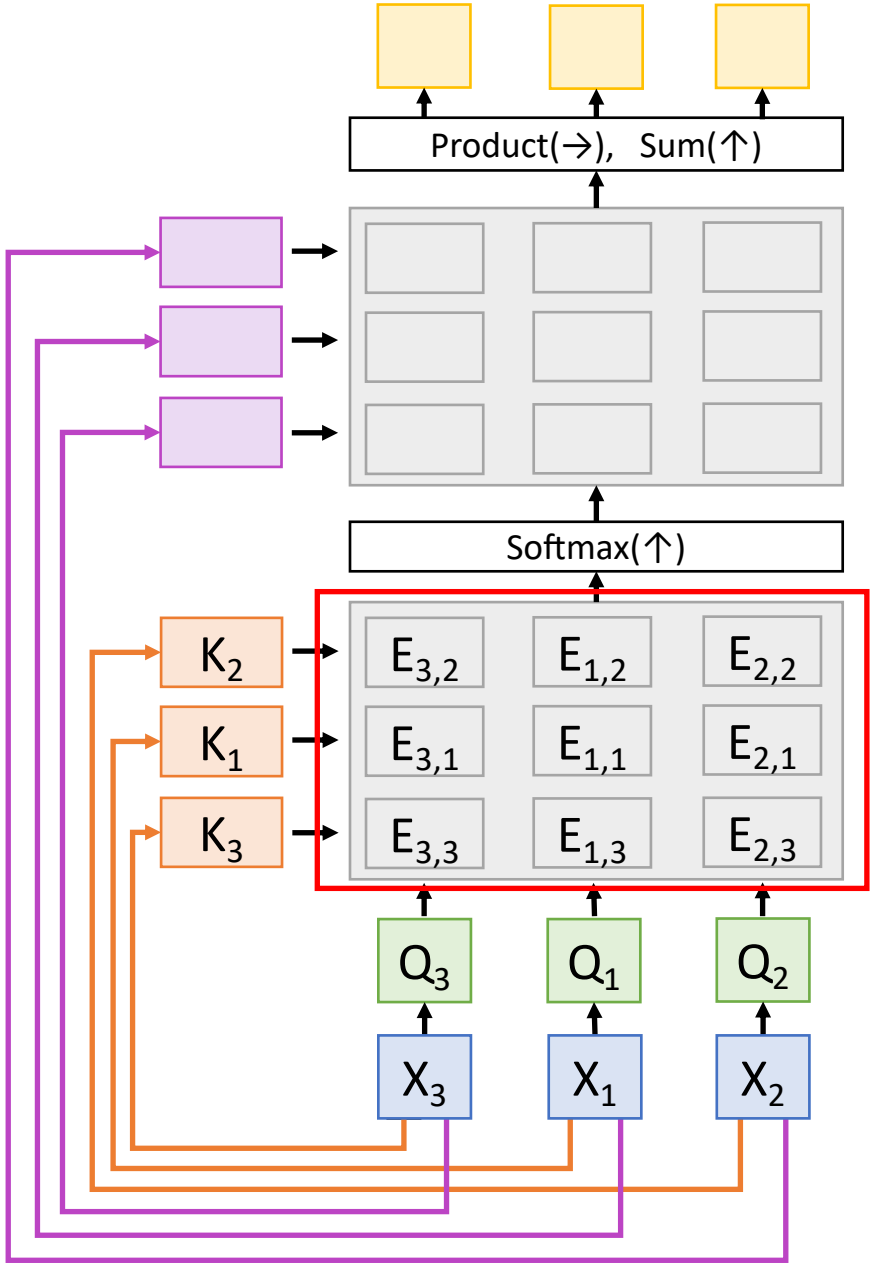
Self-Attention Layer

Consider **permuting** the input vectors:

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



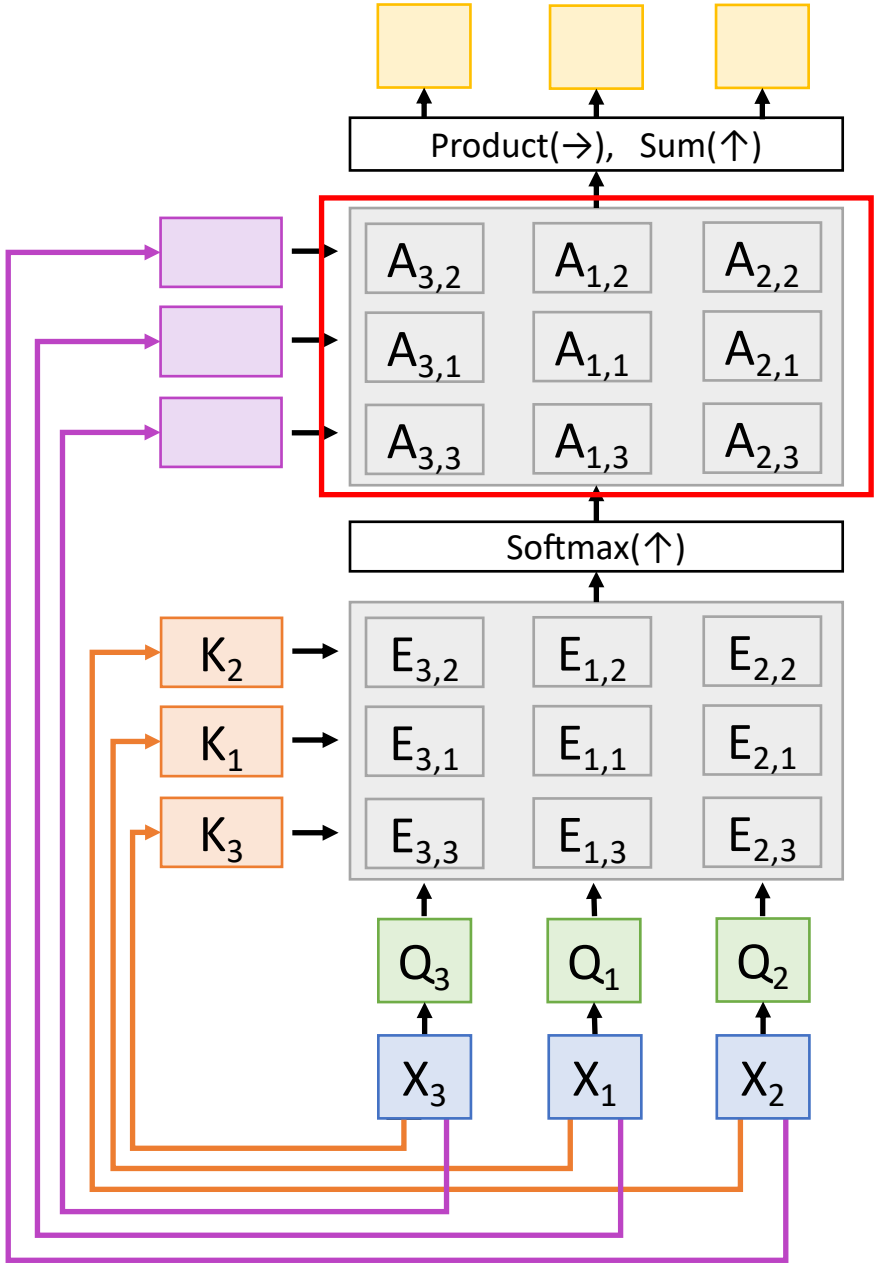
Self-Attention Layer

Consider **permuting** the input vectors:

Attention weights will be the same, but permuted

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



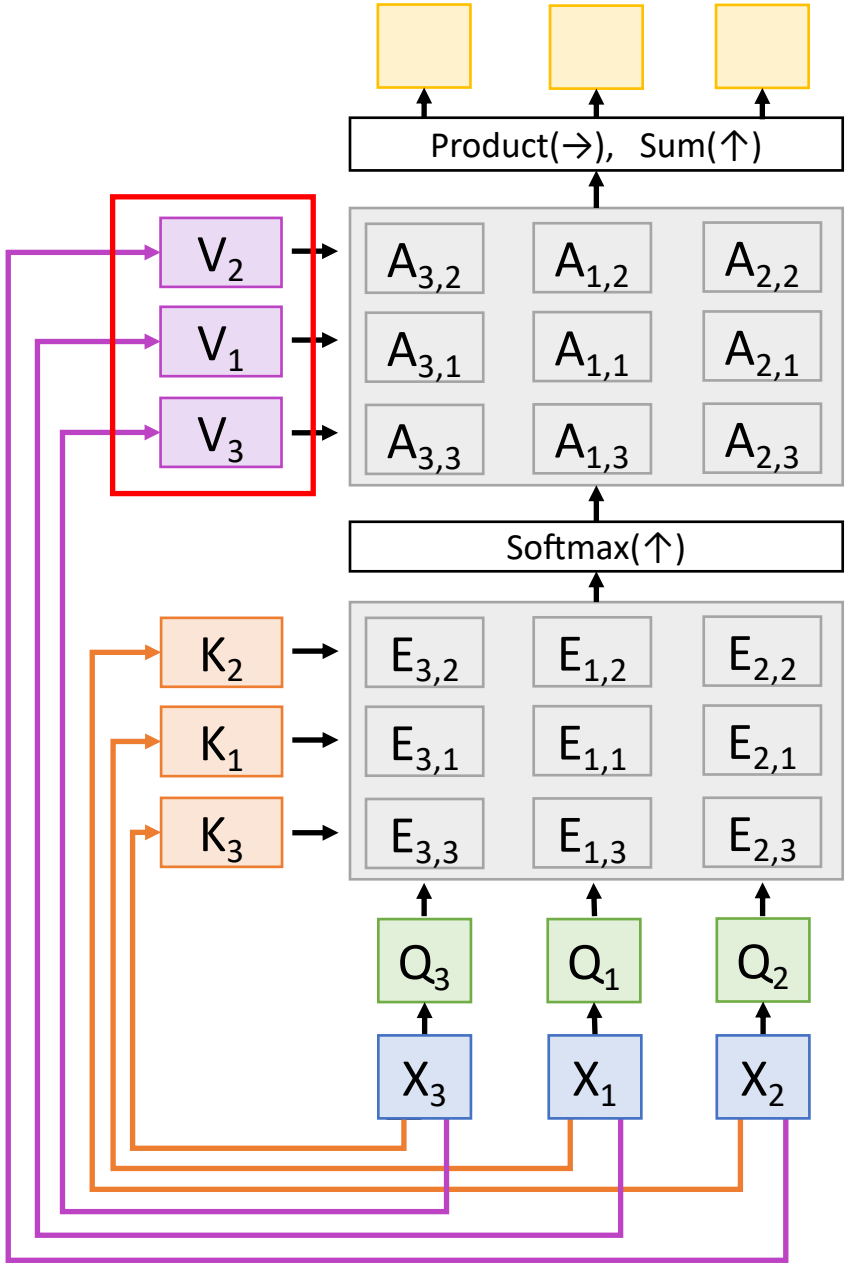
Self-Attention Layer

Consider **permuting** the input vectors:

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



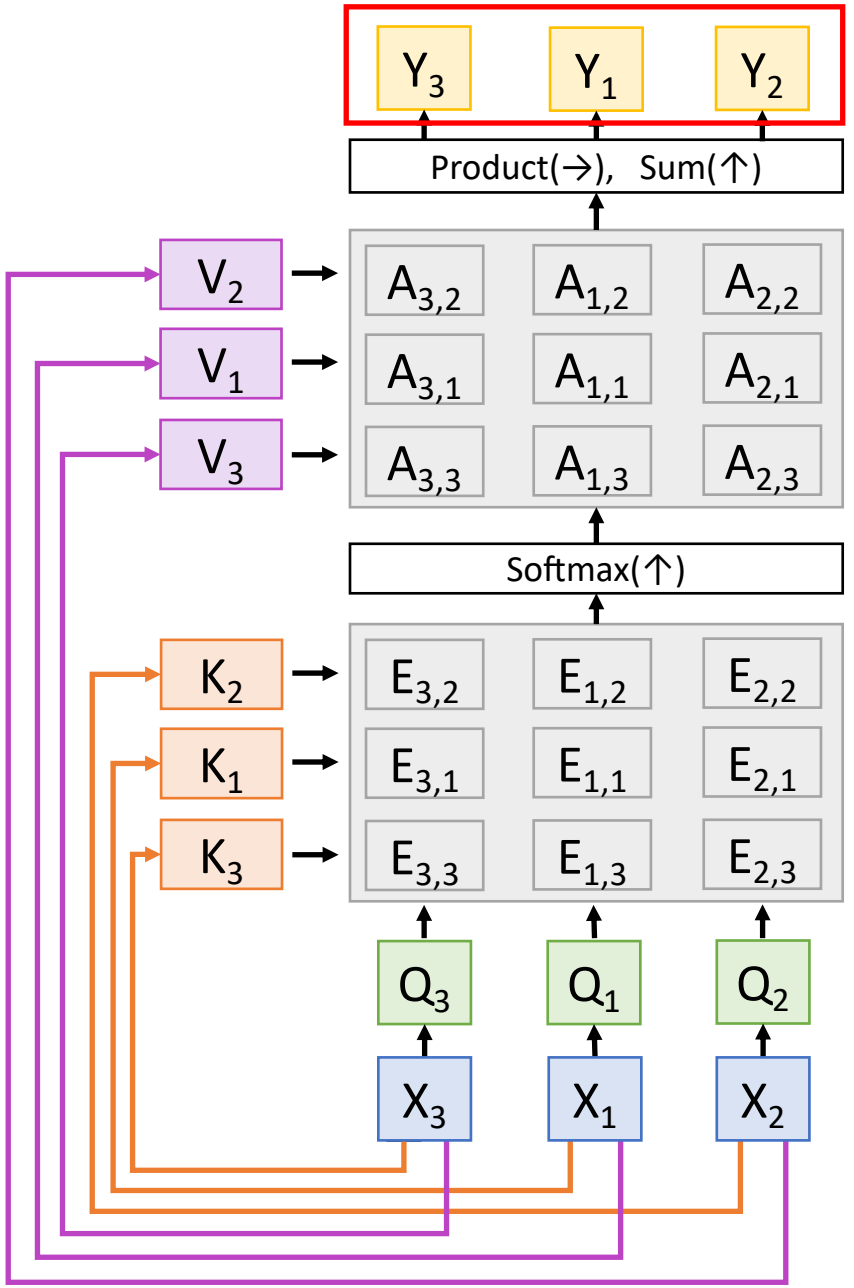
Self-Attention Layer

Consider **permuting** the input vectors:

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

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)

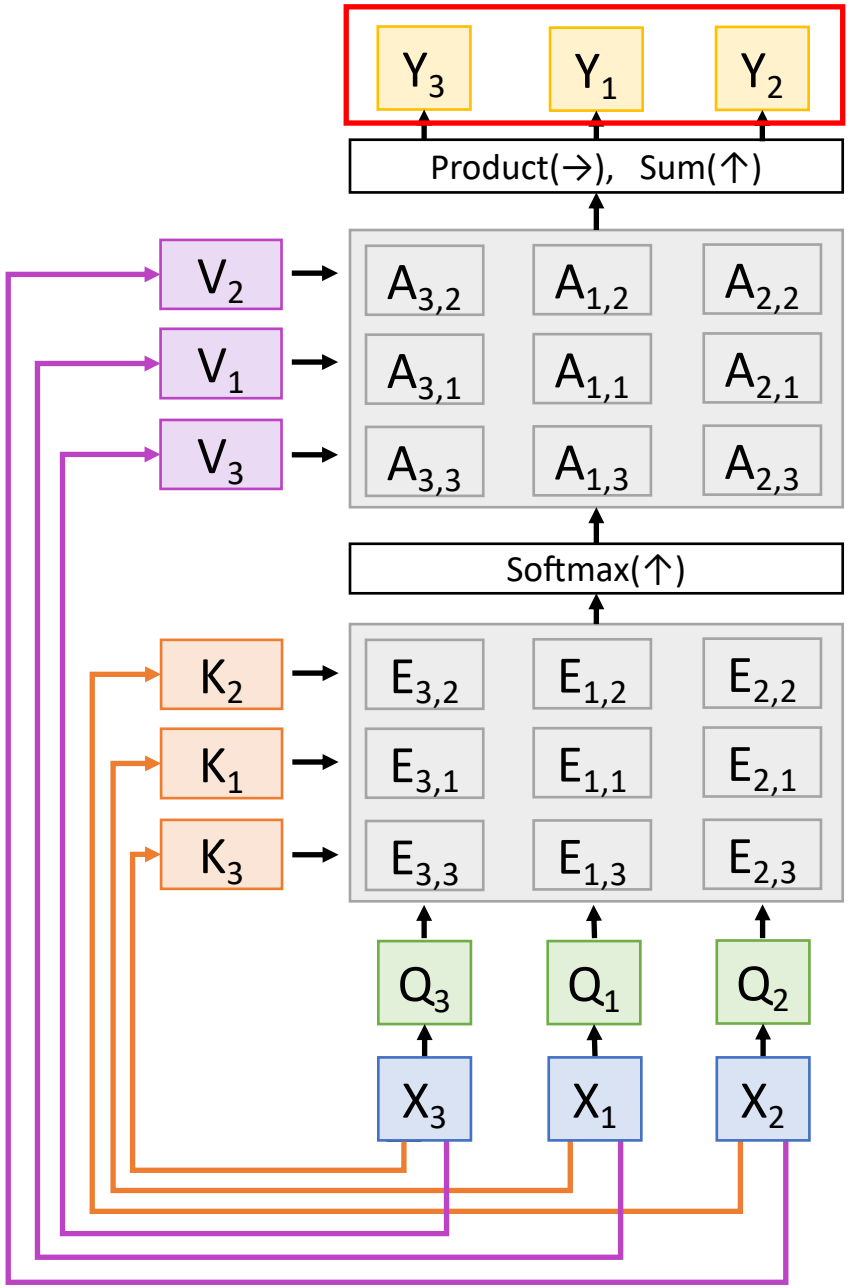
Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value Vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: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

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

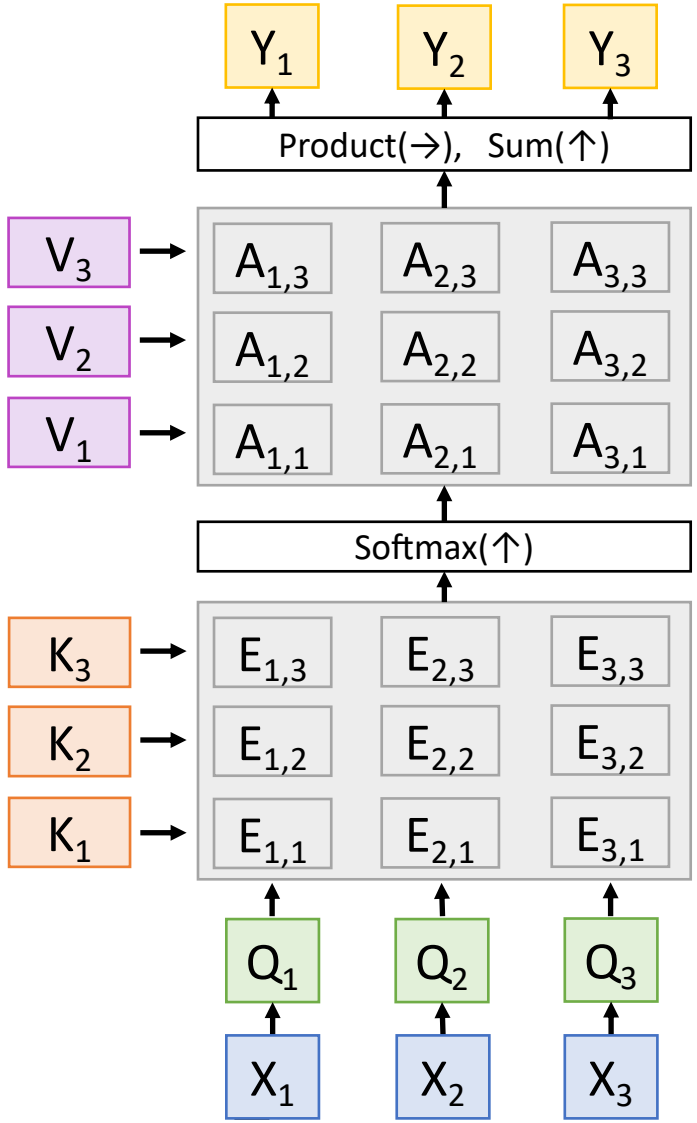


Self-Attention Layer

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

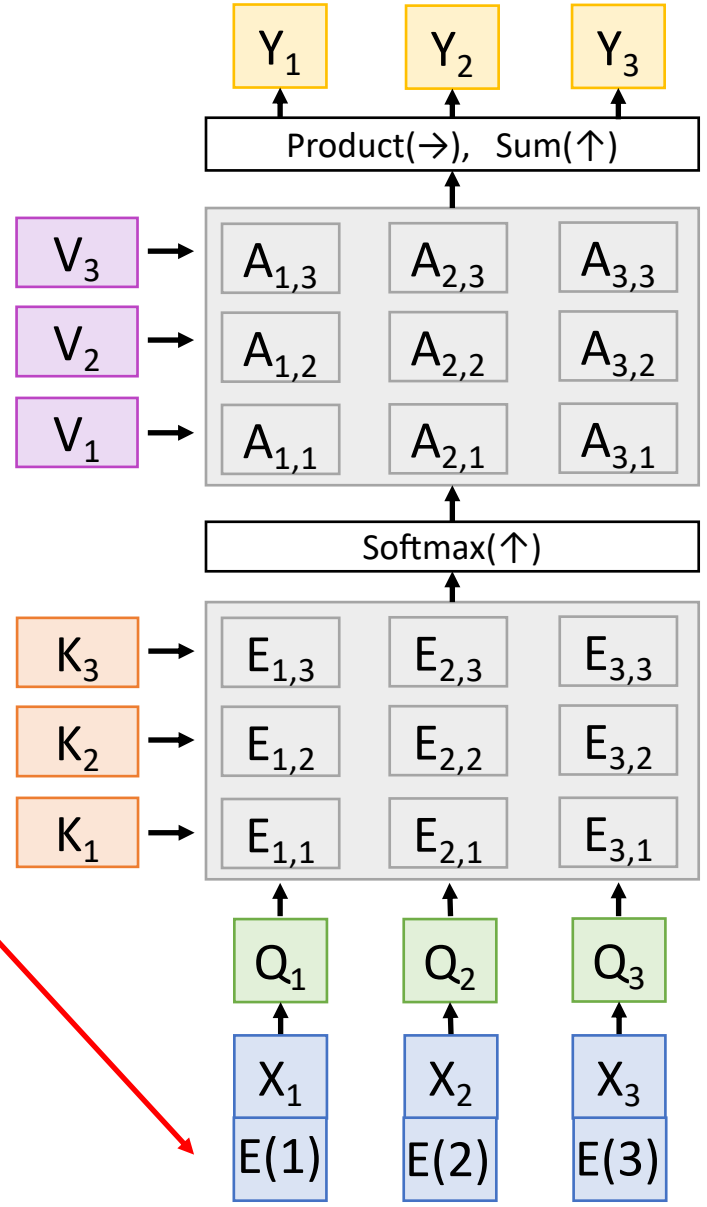
Inputs:
Input vectors: X (Shape: N_X x D_X)
Key matrix: W_K (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)
Query matrix: W_Q (Shape: D_X x D_Q)

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

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

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

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

Computation:

Query vectors: $Q = XW_Q$

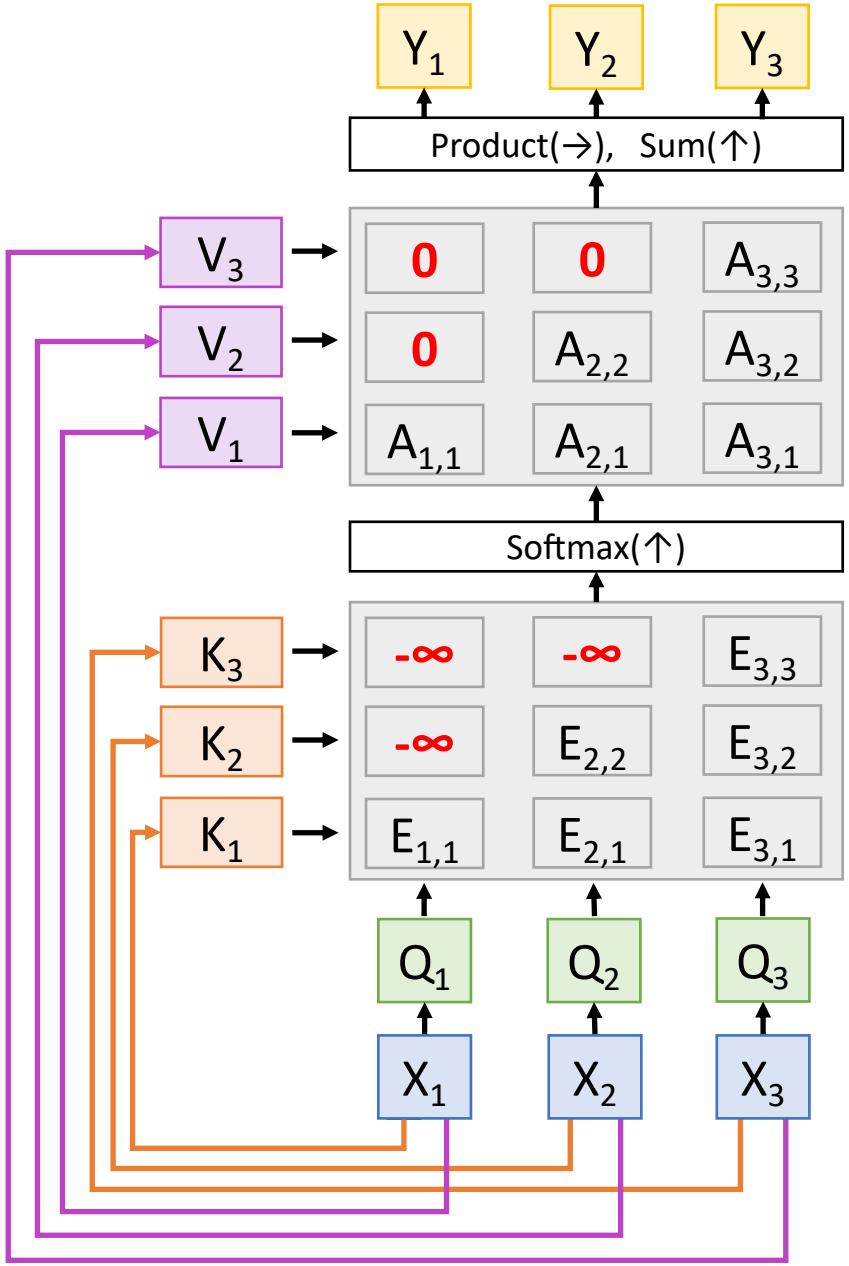
Key vectors: $K = XW_K$ (Shape: $N_X \times D_Q$)

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

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

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

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



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: $N_X \times D_X$)

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

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

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

Computation:

Query vectors: $Q = XW_Q$

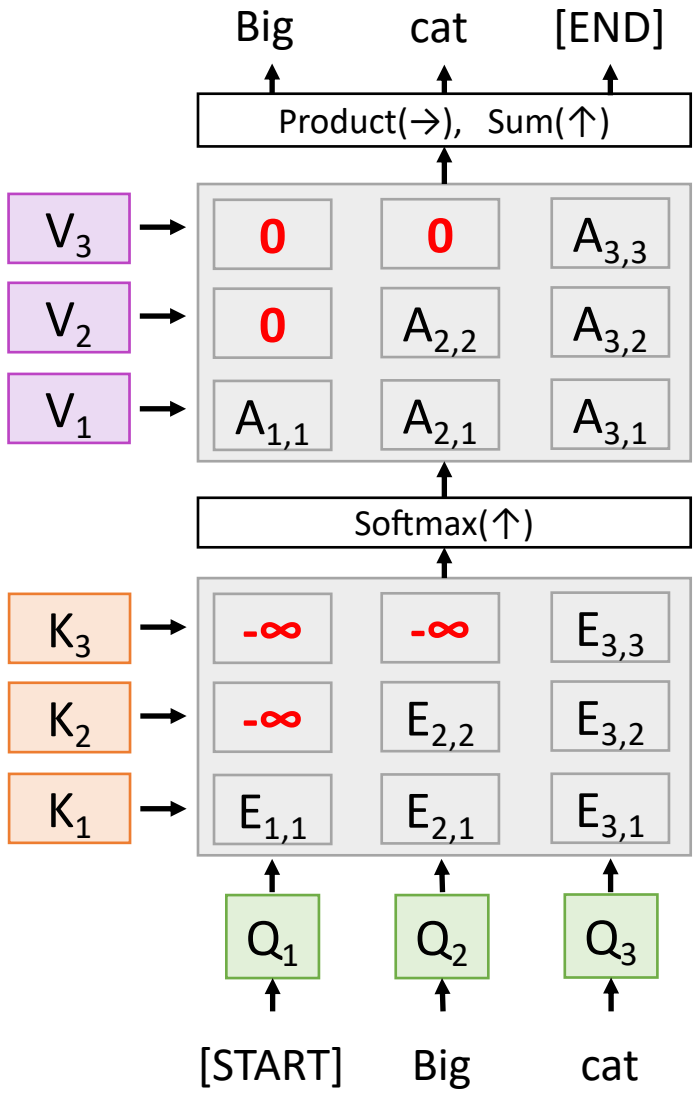
Key vectors: $K = XW_K$ (Shape: $N_X \times D_Q$)

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

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

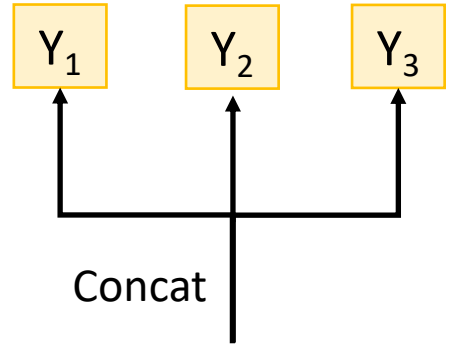
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_X \times N_X$)

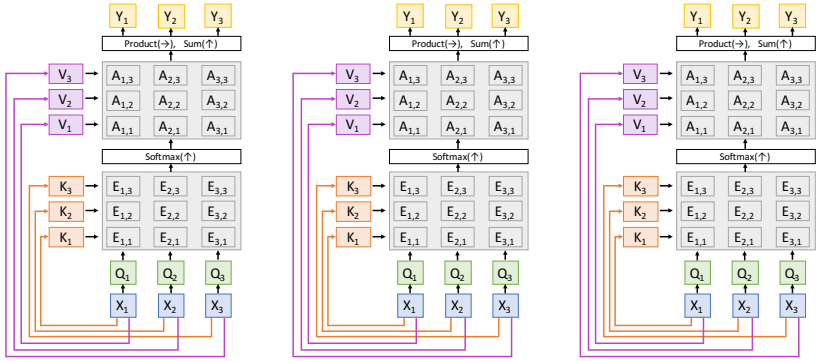
Output vectors: $Y = AV$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} V_j$

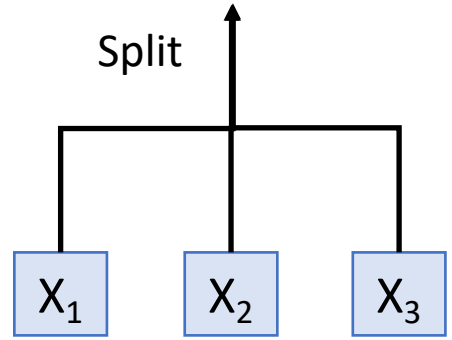


Multihead Self-Attention Layer

Use H independent
"Attention Heads" in parallel







Inputs:

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

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

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

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

Hyperparameters:

Query dimension D_Q

Number of heads H

Computation:

Query vectors: $Q = XW_Q$

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

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

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

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

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

Typically, if the dimension of the inputs X is D and there are H heads, the values, queries, and keys will all be of size D/H, as this allows for an efficient implementation.

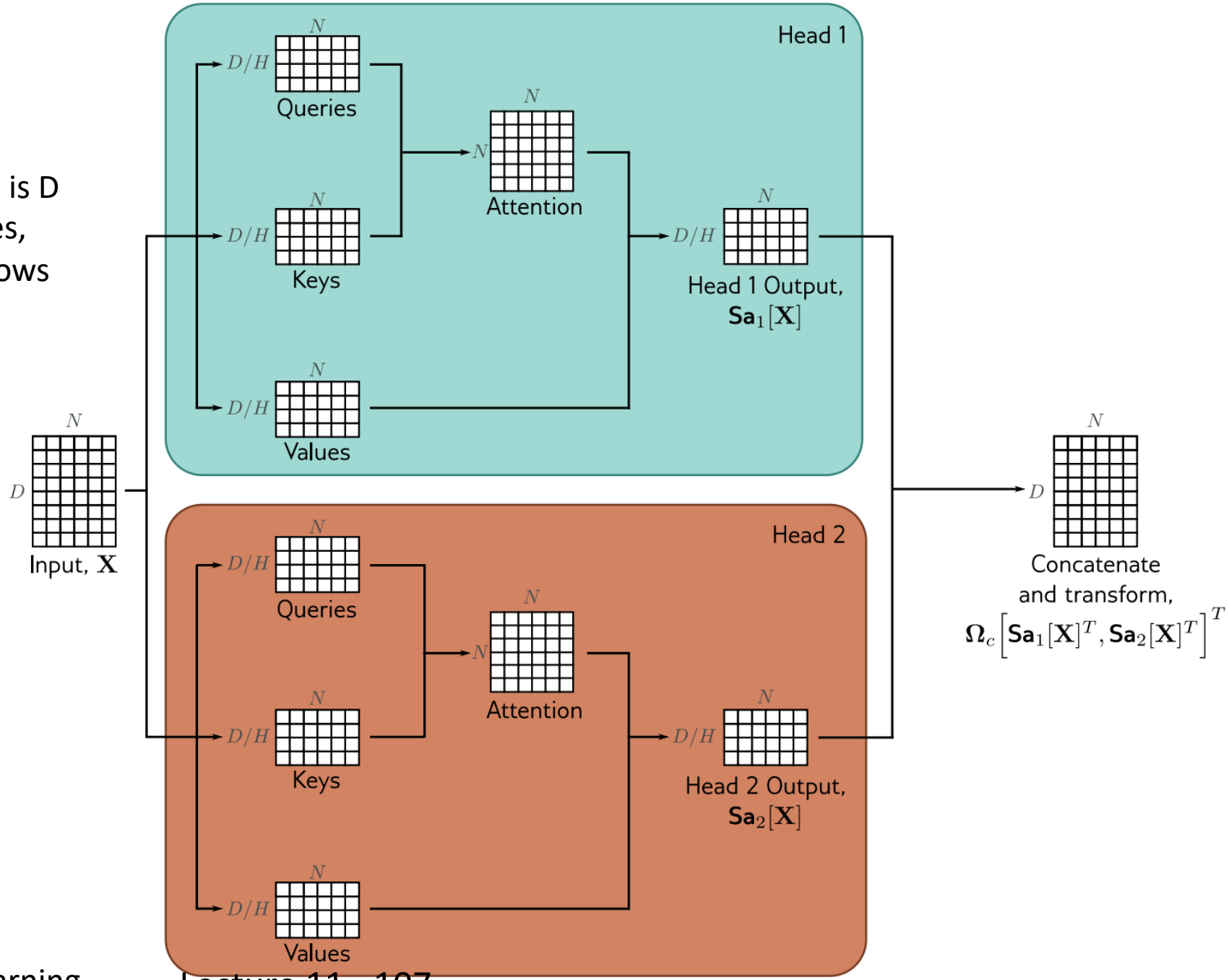
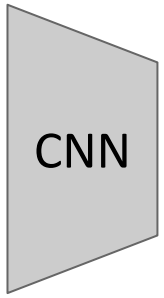
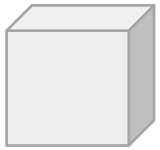


Figure 12.6 from Understanding Deep Learning

Example: CNN with Self-Attention

Input Image

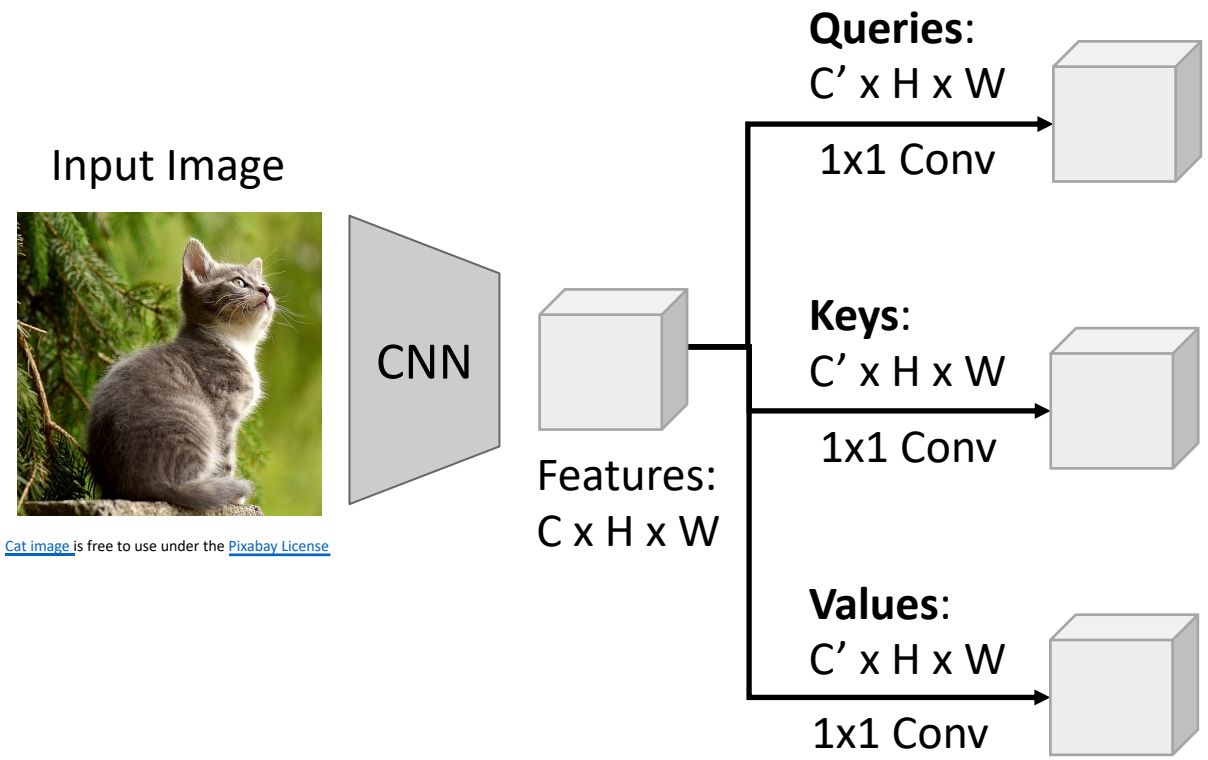




Features:
C x H x W

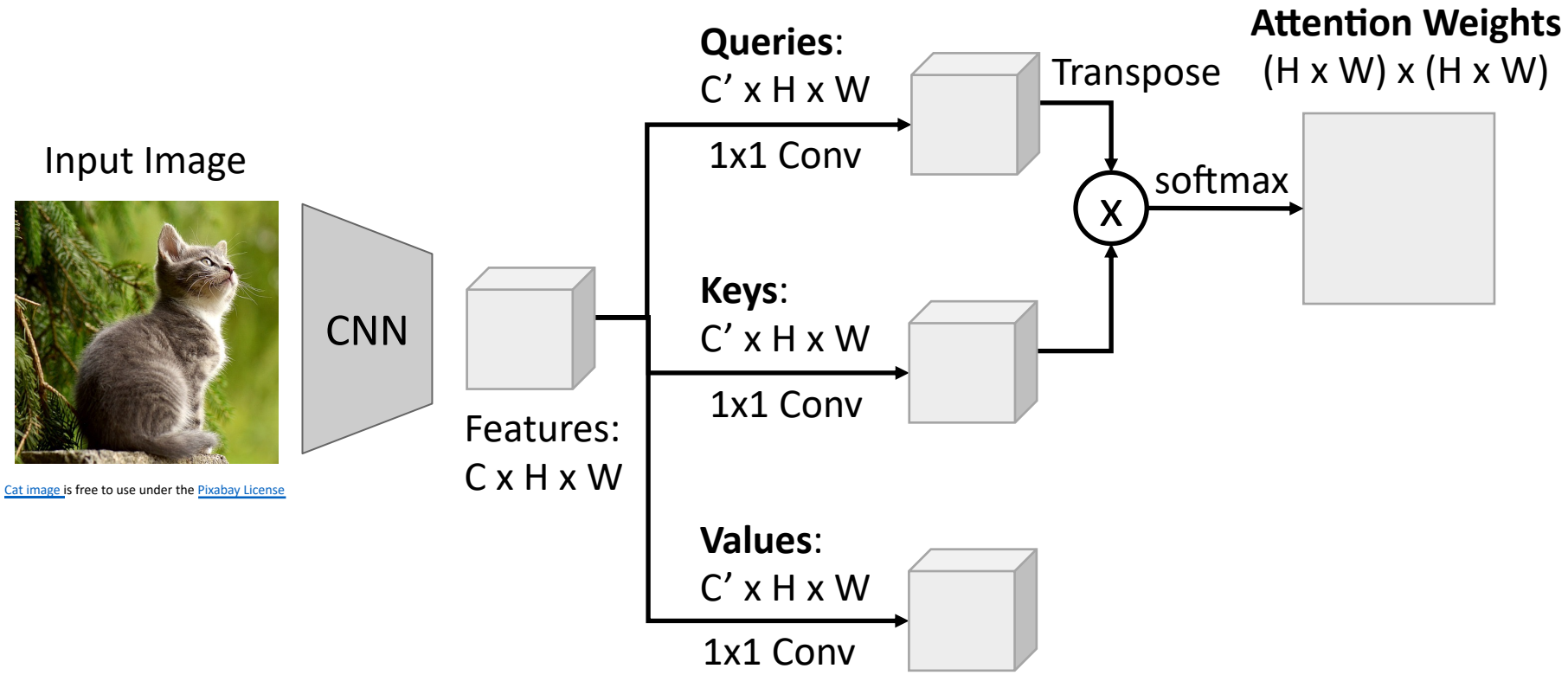
Zhang et al, "Self-Attention Generative Adversarial Networks", ICML 2018

Example: CNN with Self-Attention



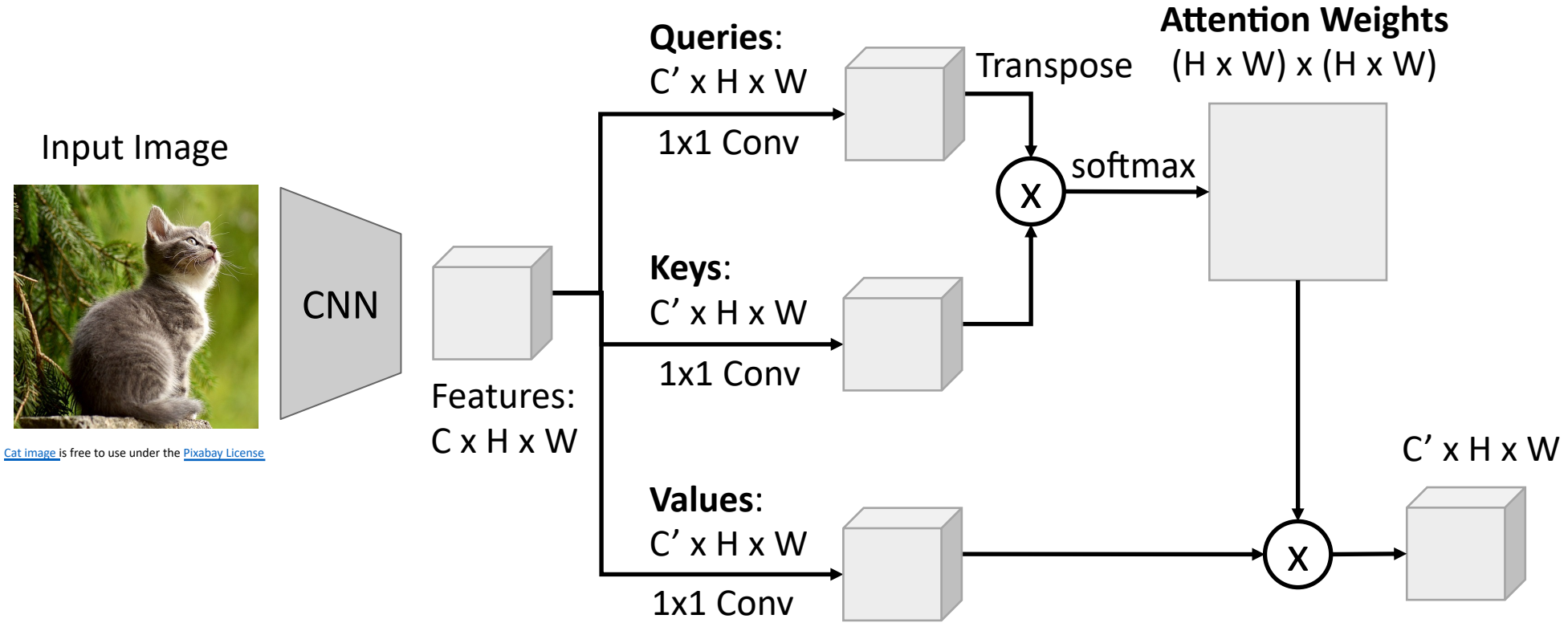
Zhang et al, "Self-Attention Generative Adversarial Networks", ICML 2018

Example: CNN with Self-Attention



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

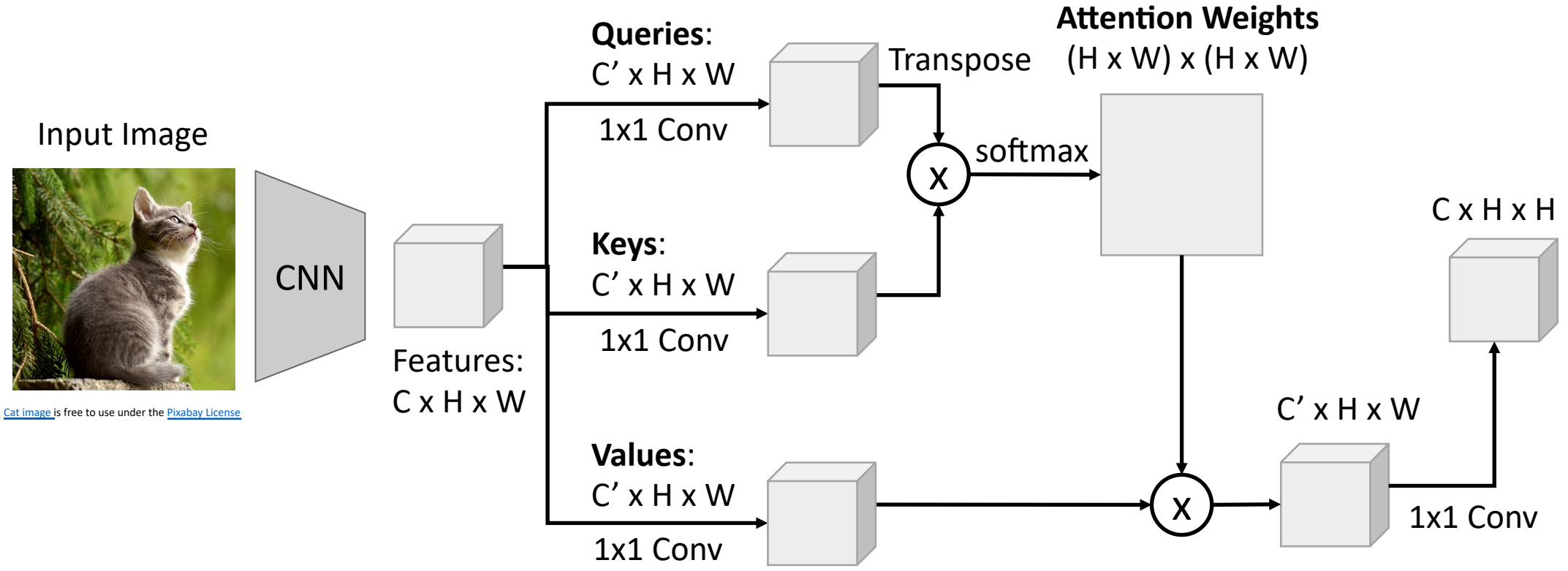
Example: CNN with Self-Attention



Cat image is free to use under the Pixabay License

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

Example: CNN with Self-Attention



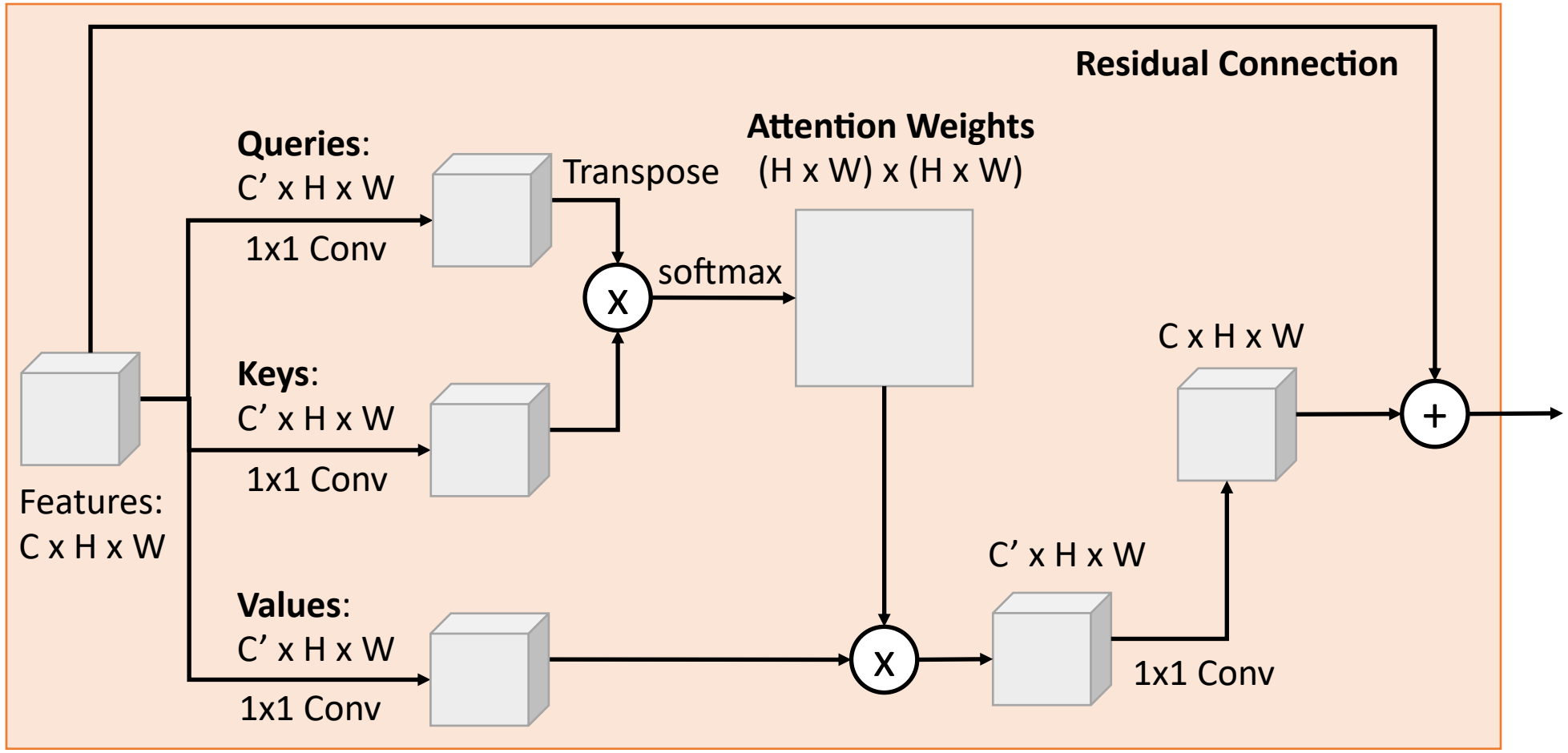
Cat image is free to use under the Pixabay License

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

Example: CNN with Self-Attention

Input Image

CNN

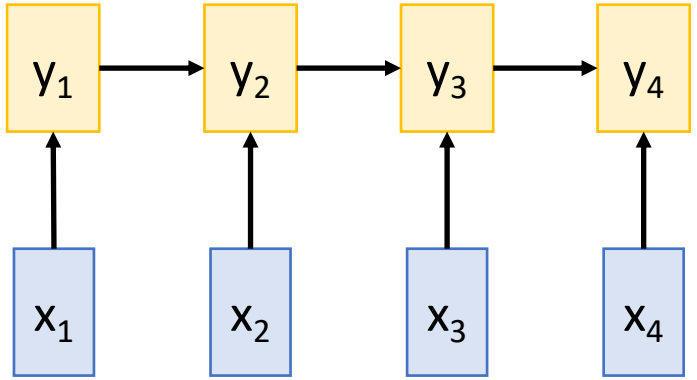


Self-Attention Module

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

Three Ways of Processing Sequences

Recurrent Neural Network



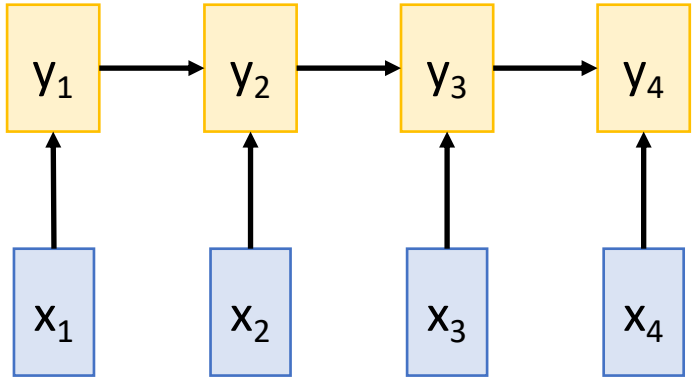
Works on **Ordered Sequences**

(+) 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

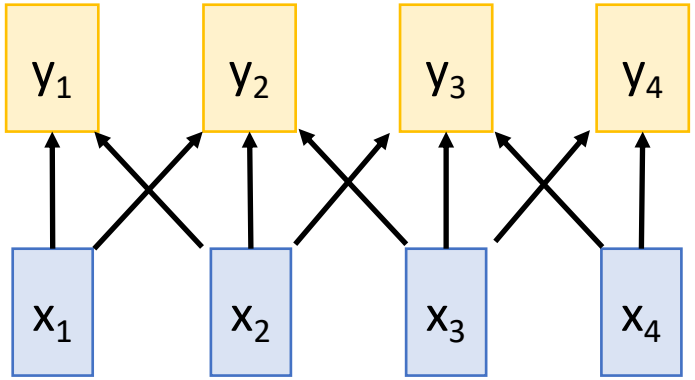


Works on **Ordered Sequences**

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

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

1D Convolution



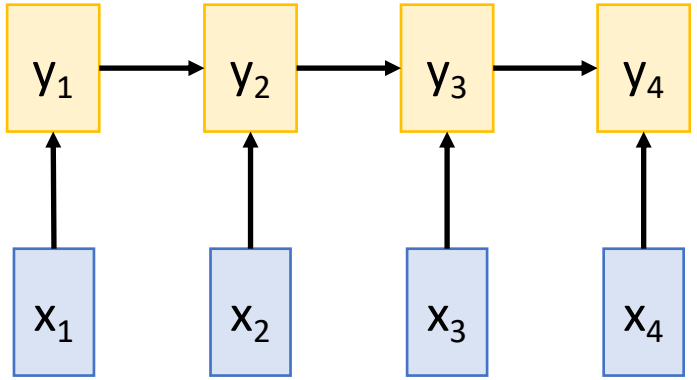
Works on **Multidimensional Grids**

(-) 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

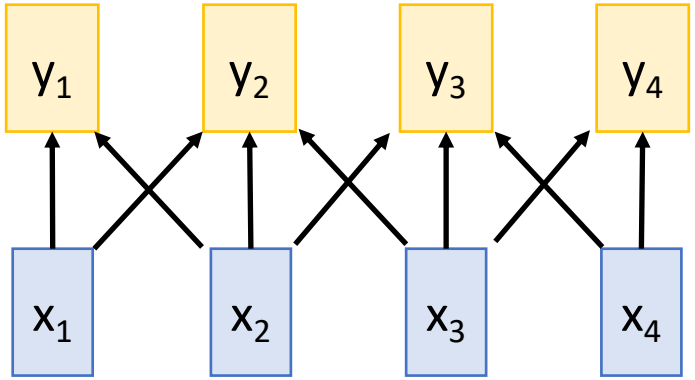


Works on **Ordered Sequences**

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

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

1D Convolution

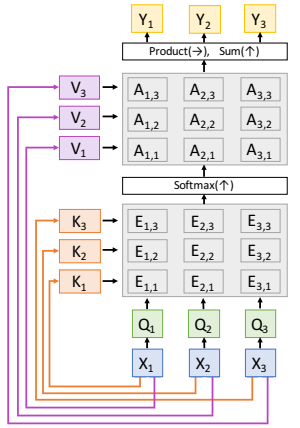


Works on **Multidimensional Grids**

(-) 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**

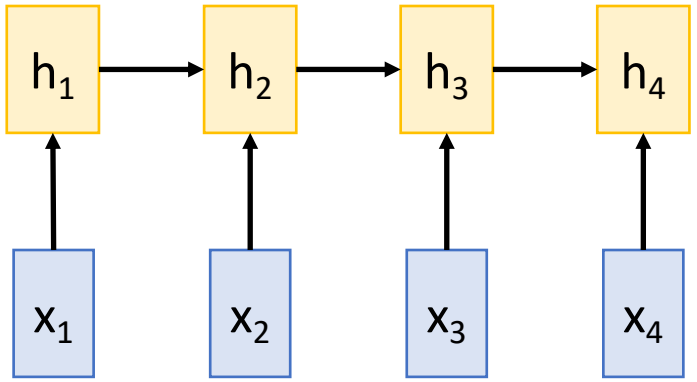
(-) Good at long sequences: after one self-attention layer, each output "sees" all inputs!

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

(-) Very memory intensive

Summary

Recurrent Neural Network (RNN) for modeling sequence



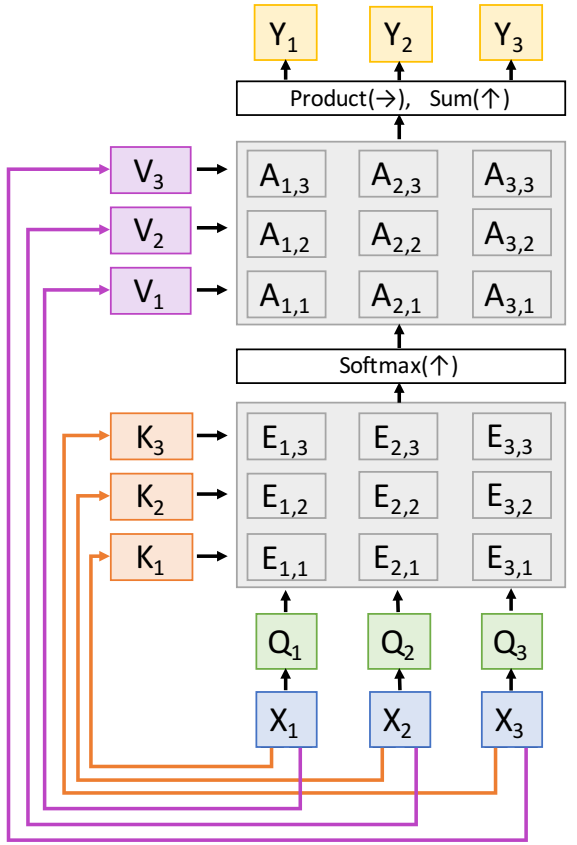
$$h_t = f_W(h_{t-1}, x_t)$$

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.

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

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



Reading and Practice

Dive into Deep learning:
RNN: Chapter 9
Attention: Chapter 10.1, 10.2, 10.3, 10.5, 10.6

Understanding Deep Learning:
Chapter 12