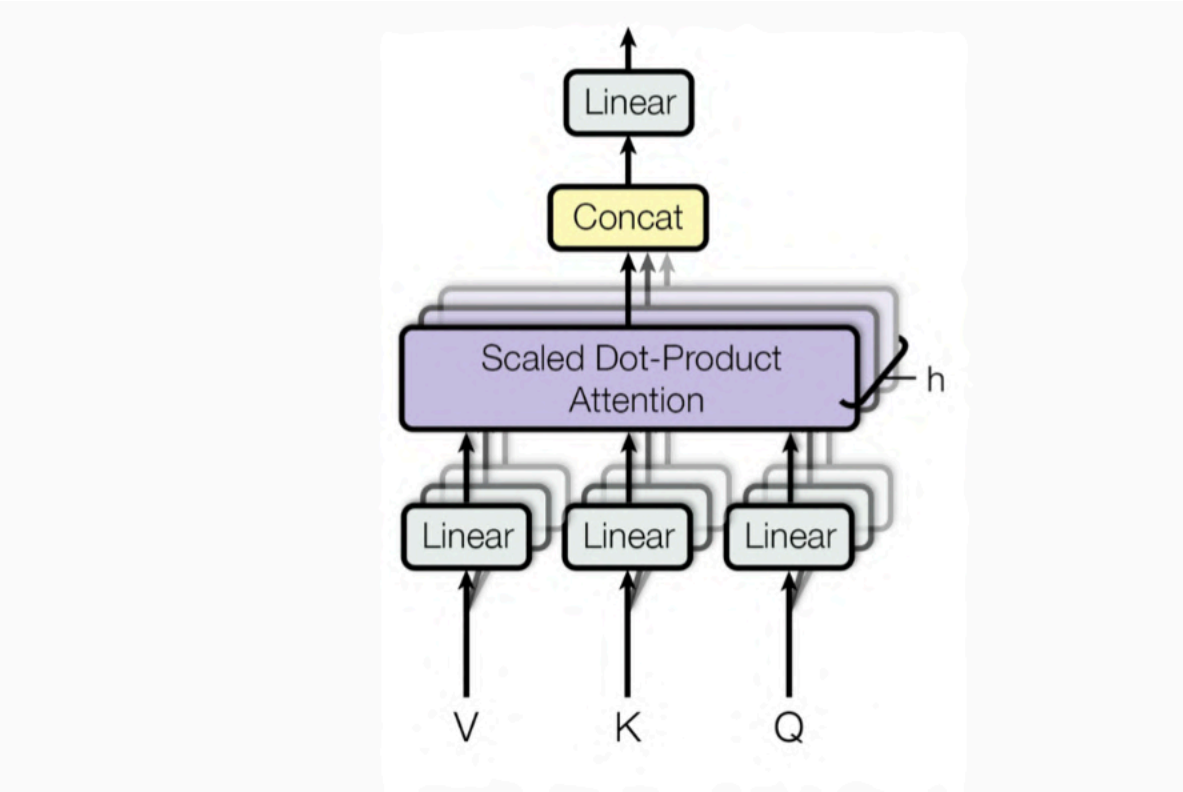
Attention and Transformers



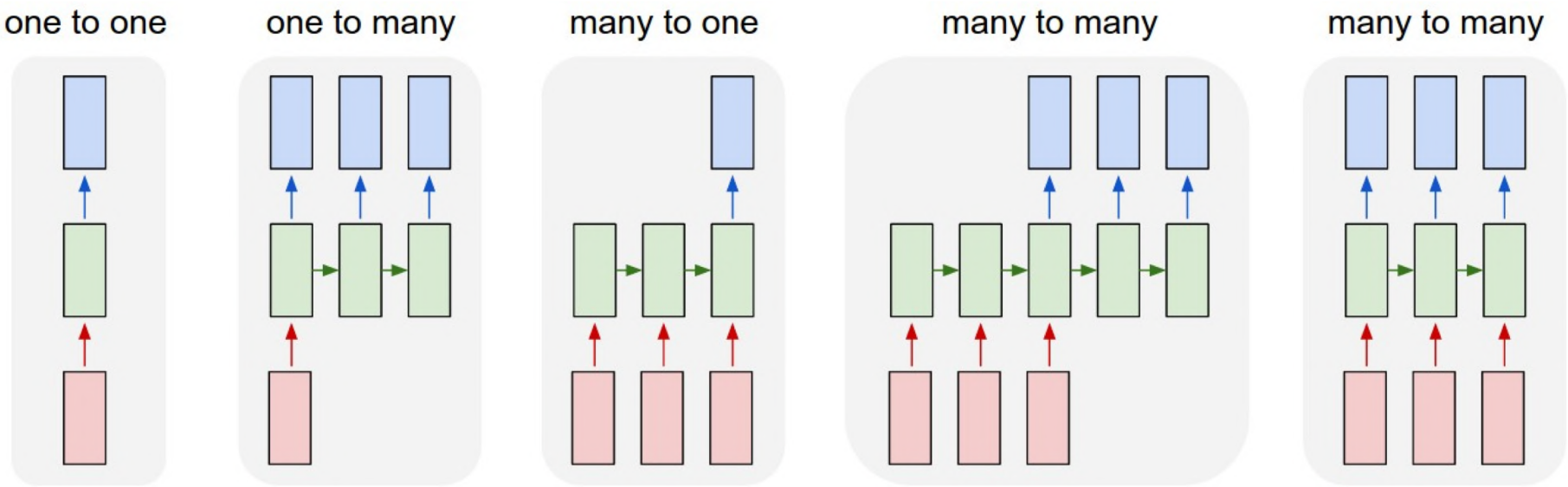
Adapted from: Stanford CS231 , Illinois cs498 , J. Johnson, Rob Fergus

Outline

- General module of Attention
- Self-Attention
- Transformers
- Visual Transformer (ViT)

Last Week

Common RNN Structures

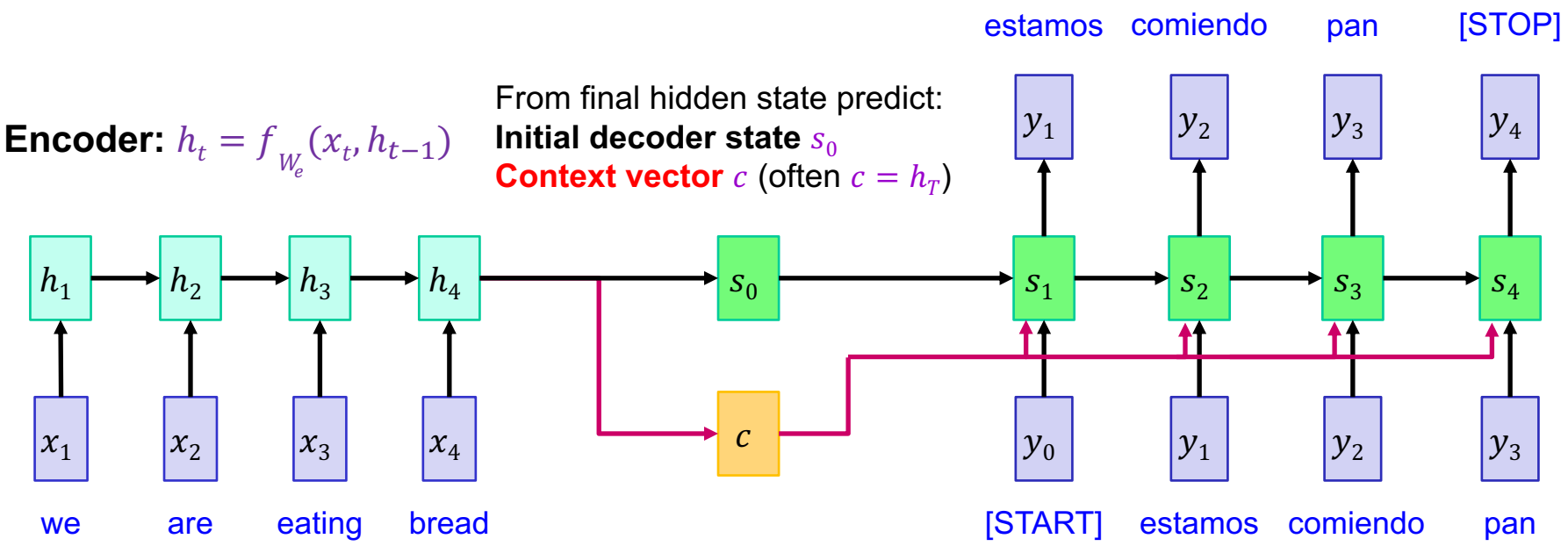


- One of the benefits of recurrent neural networks is the ability to **handle arbitrary length inputs and outputs**.
- This flexibility allows us to define a broad range of tasks.

Adapted from Stanford cs231n

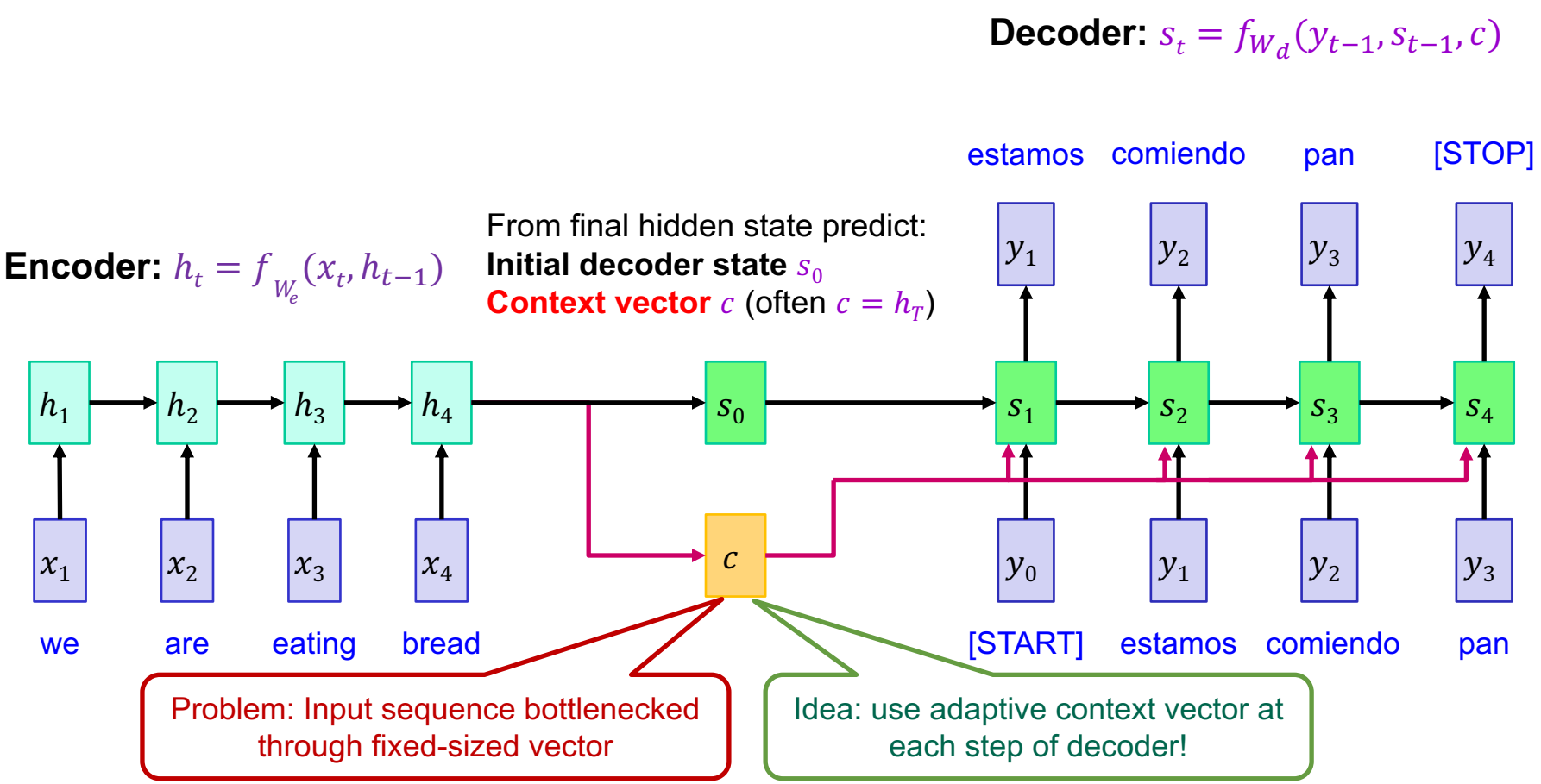
Seq2seq: RNNs with a fixed context

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



Source: J. Johnson

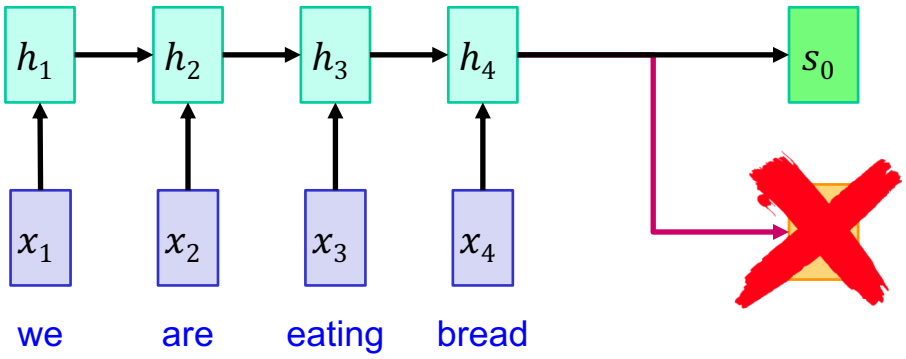
Seq2seq: RNNs with a fixed context



A. Sutskever, O. Vinyals, Q. Le, Sequence to sequence learning with neural networks, NeurIPS 2014

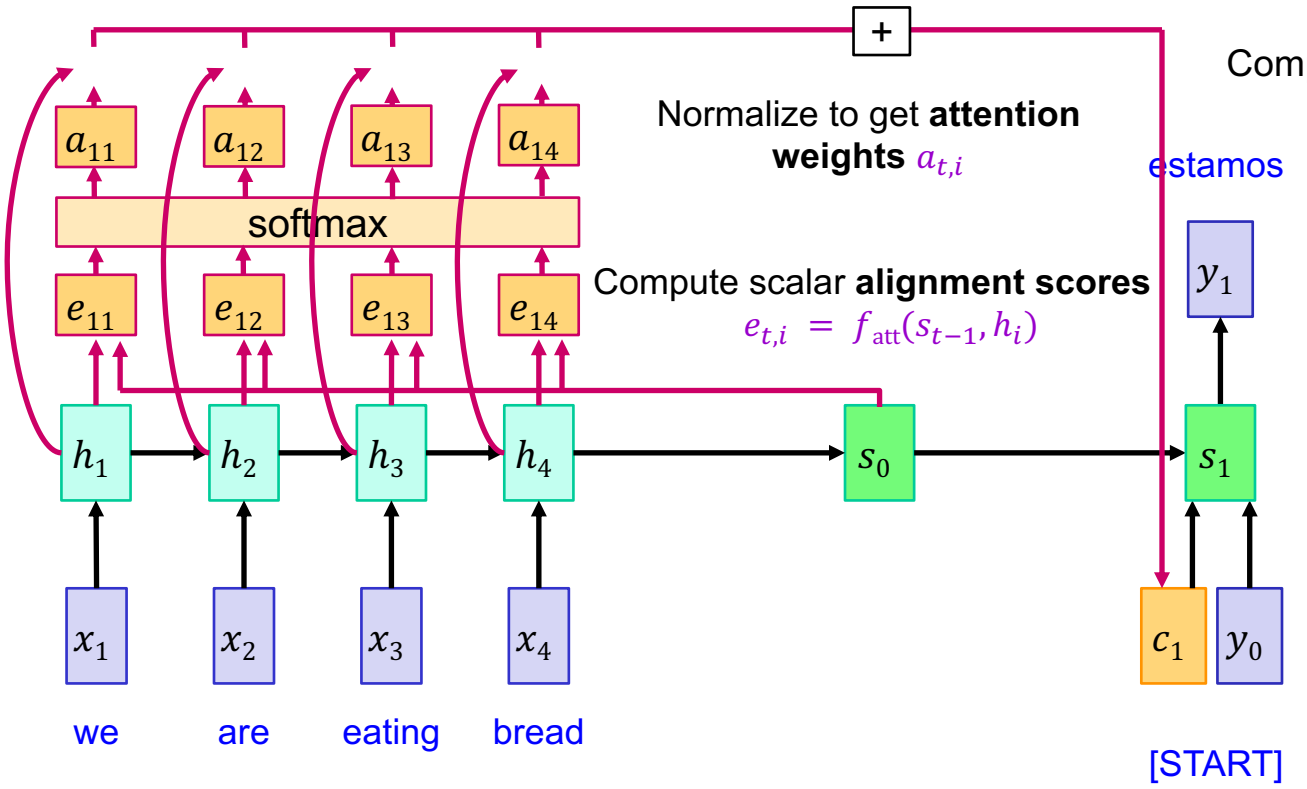
Seq2seq: RNNs with attention

Attention: At each timestep of decoder, context vector "looks at" different parts of the input sequence



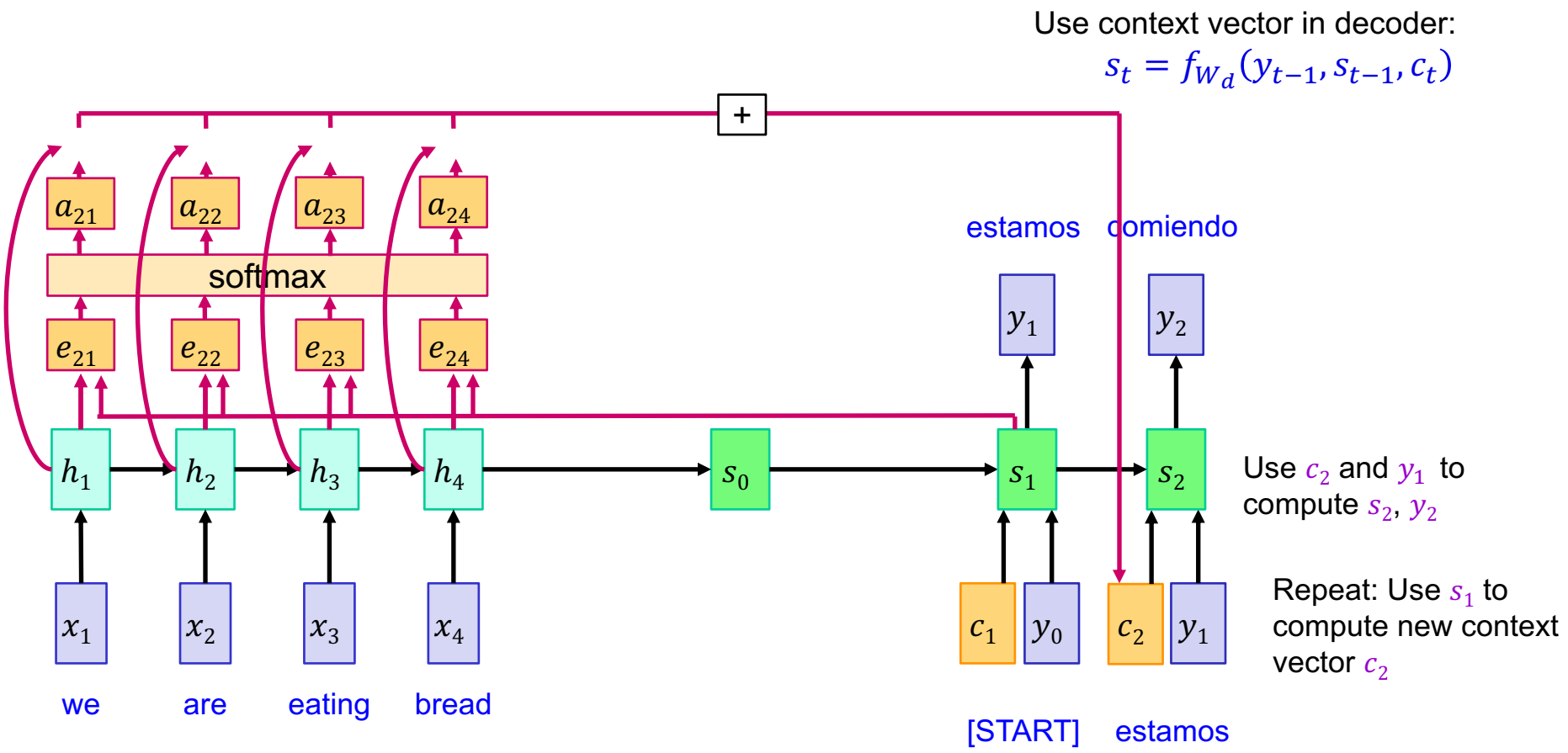
Source: J. Johnson

Seq2seq: RNNs with attention



D. Bahdanau, K. Cho, Y. Bengio, Neural Machine Translation by Jointly Learning to Align and Translate, ICLR 2015

Seq2seq: RNNs with attention

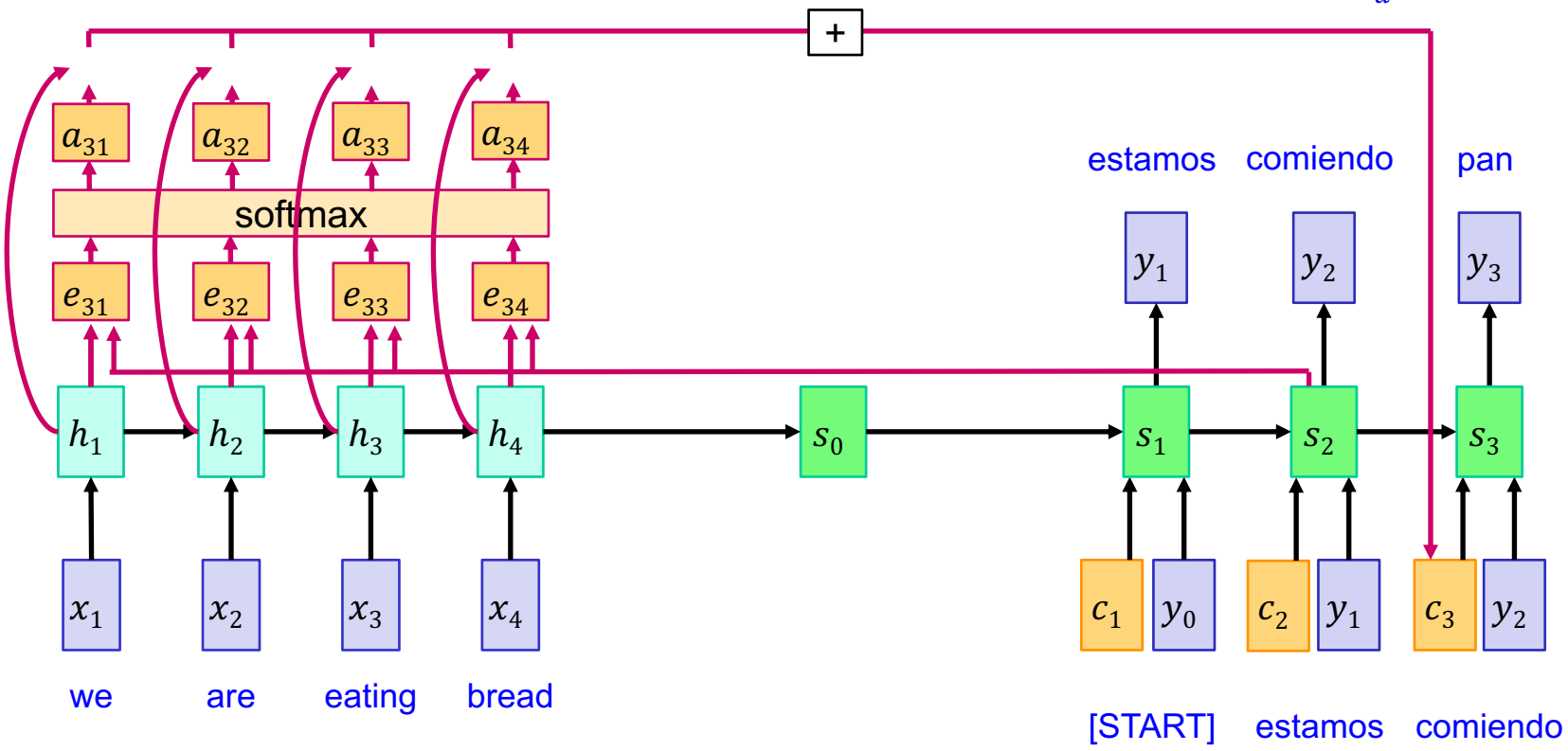


Source: J. Johnson

Seq2seq: RNNs with attention

Use context vector in decoder:

$$s_t = f_{W_d}(y_{t-1}, s_{t-1}, c_t)$$

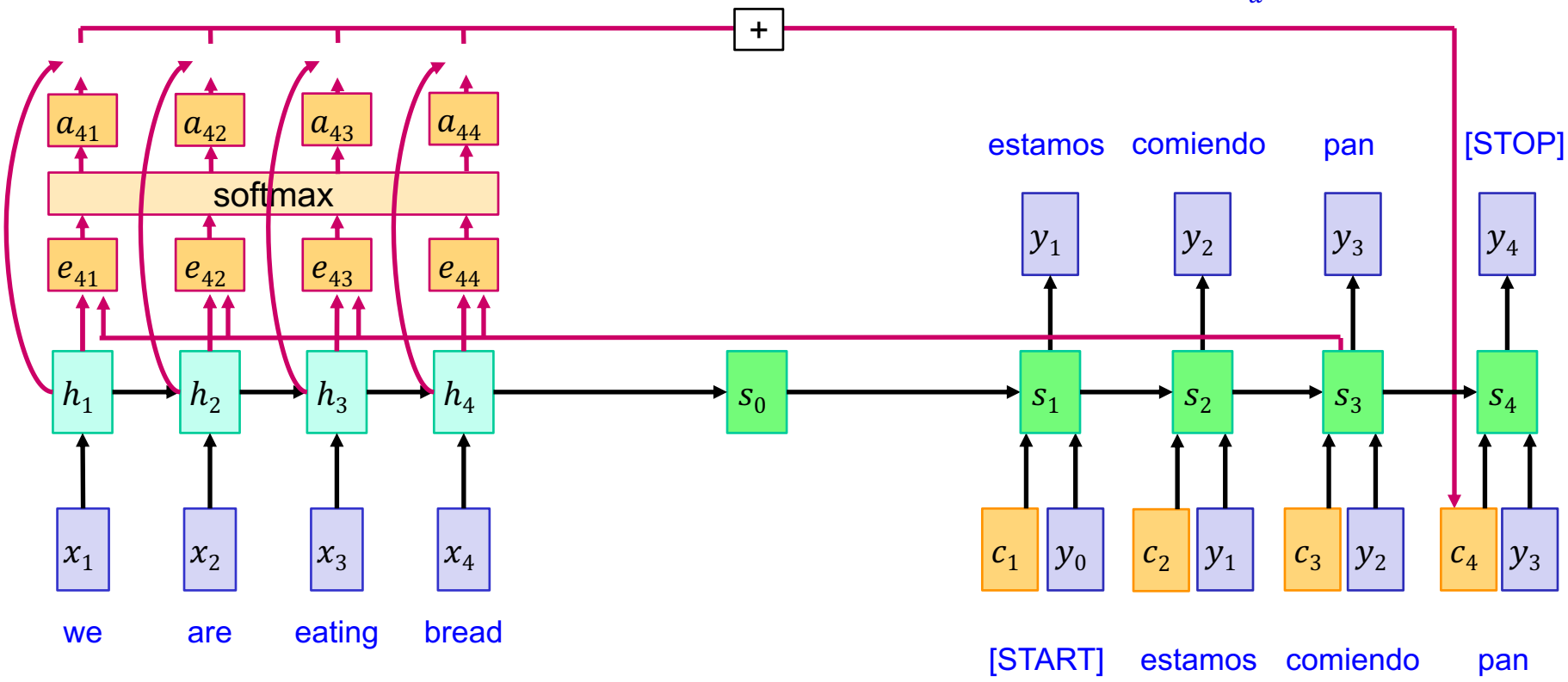


Source: J. Johnson

Seq2seq: RNNs with attention

Use context vector in decoder:

$$s_t = f_{W_d}(y_{t-1}, s_{t-1}, c_t)$$

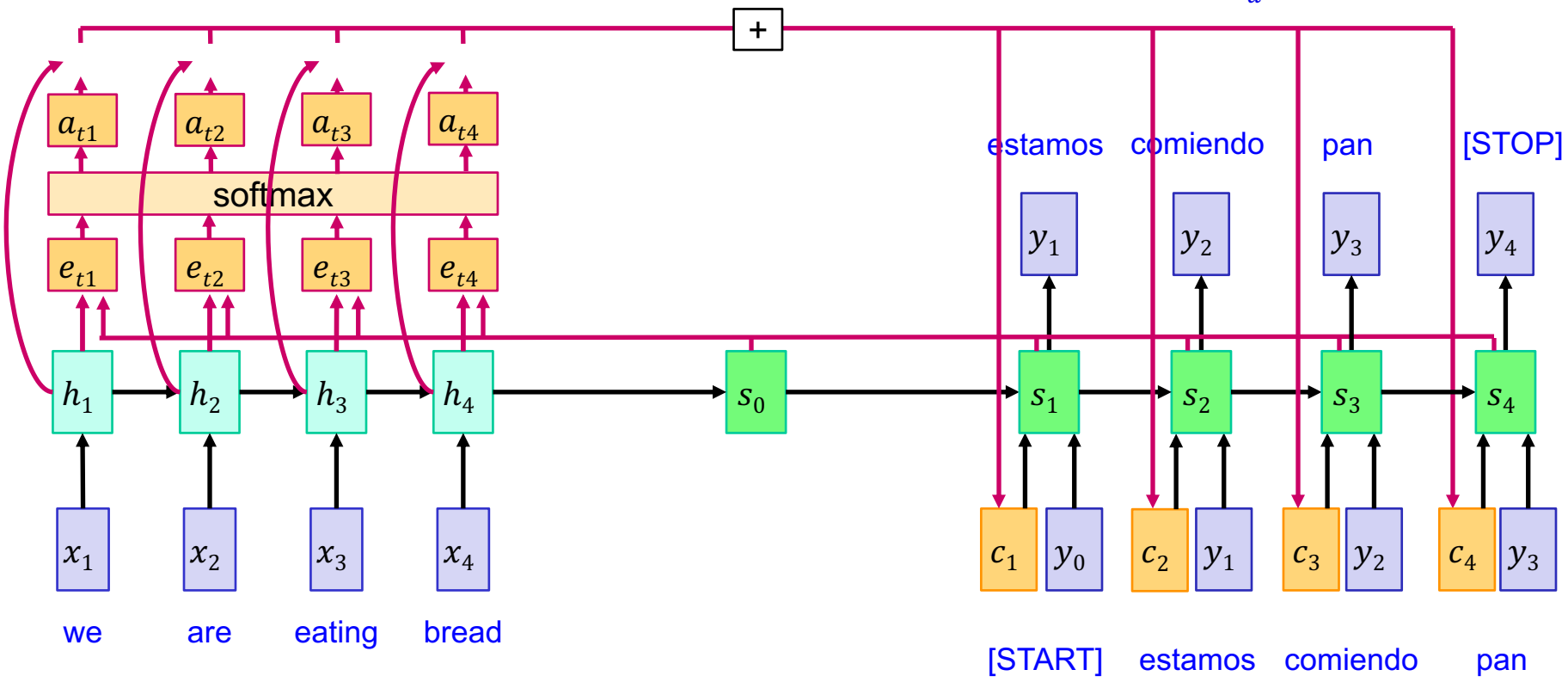


Source: J. Johnson

Seq2seq: RNNs with attention

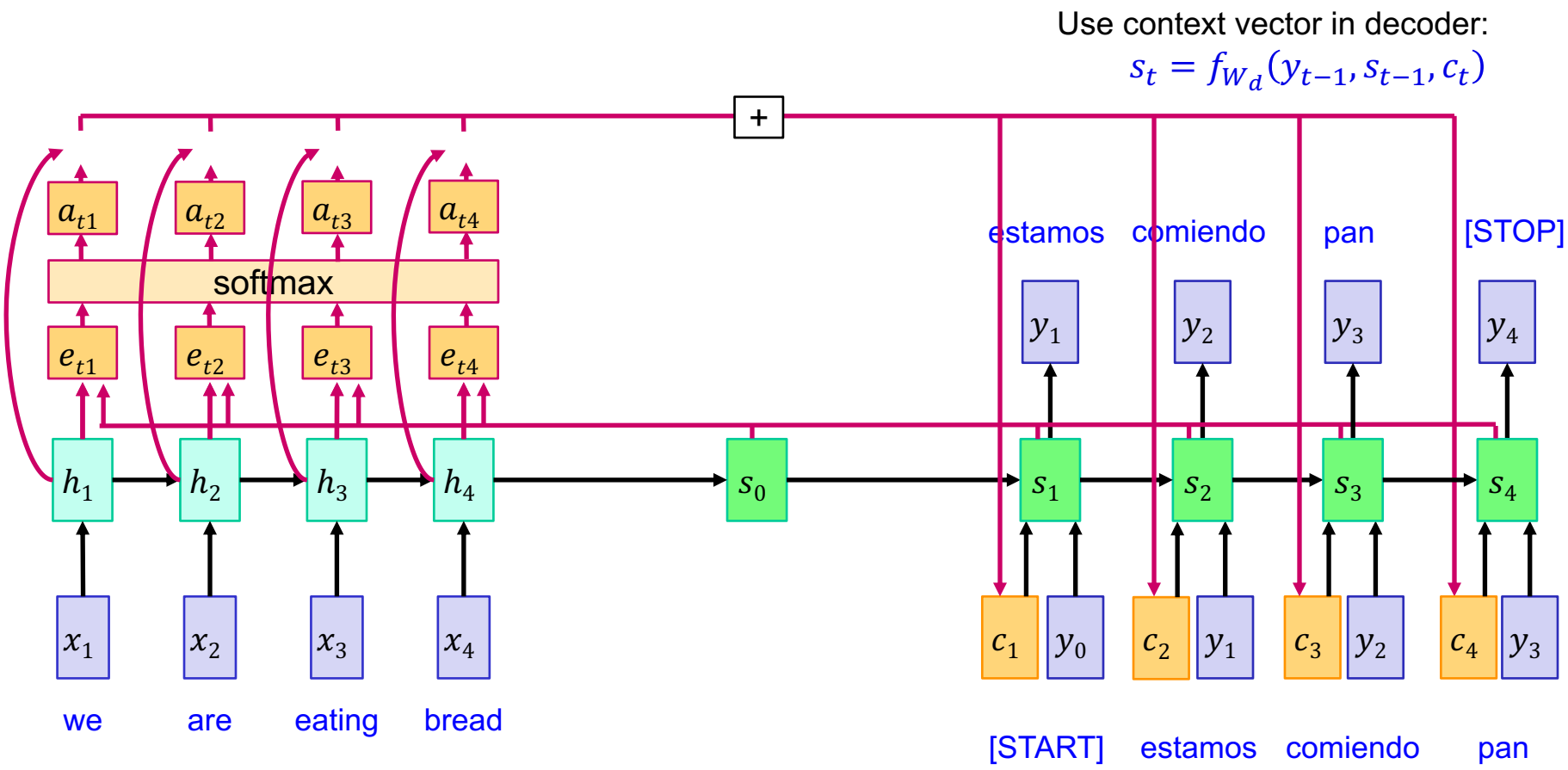
Use context vector in decoder:

$$s_t = f_{W_d}(y_{t-1}, s_{t-1}, c_t)$$



Source: J. Johnson

Seq2seq: RNNs with attention

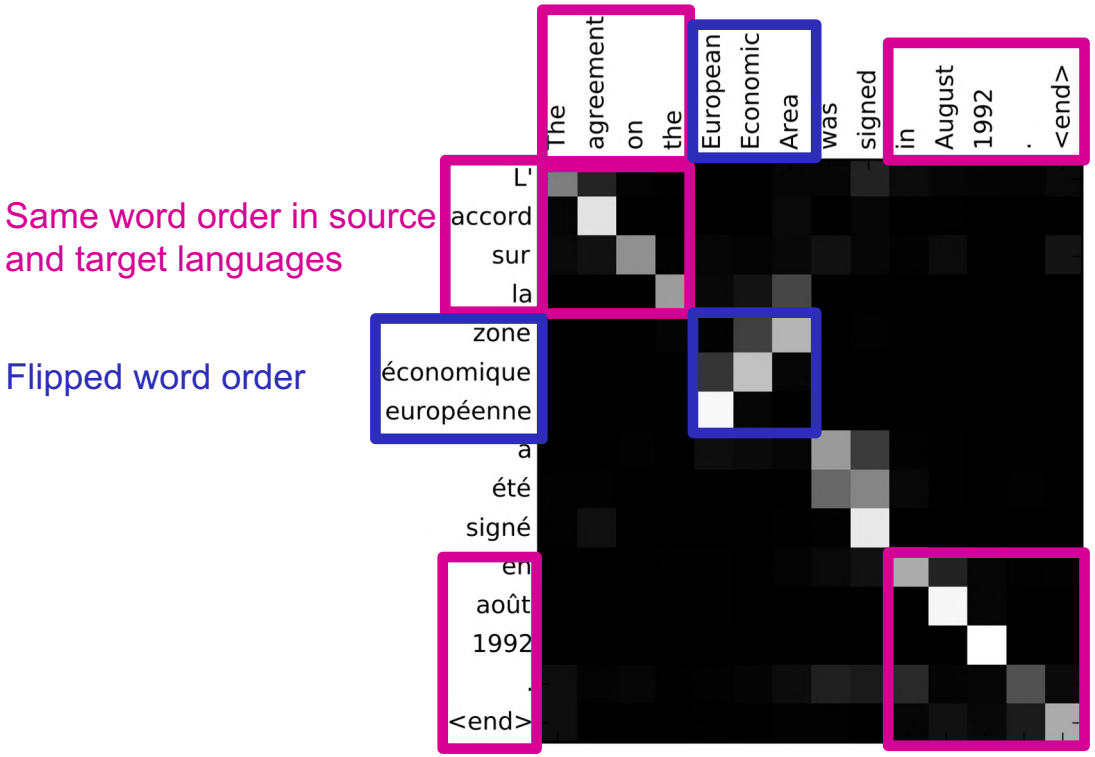


- Intuition: Context vector attends to the relevant part of the input sequence, e.g. “estamous”=“we are”, so maybe $a_{11} = a_{12} = 0.45$

  $and\ \ a_{13} = a_{14} = 0.05$

Seq2seq: RNNs with attention

- Visualizing attention weights (English source, French target):



D. Bahdanau, K. Cho, Y. Bengio, Neural Machine Translation by Jointly Learning to Align and Translate, ICLR 2015

Image captioning with attention

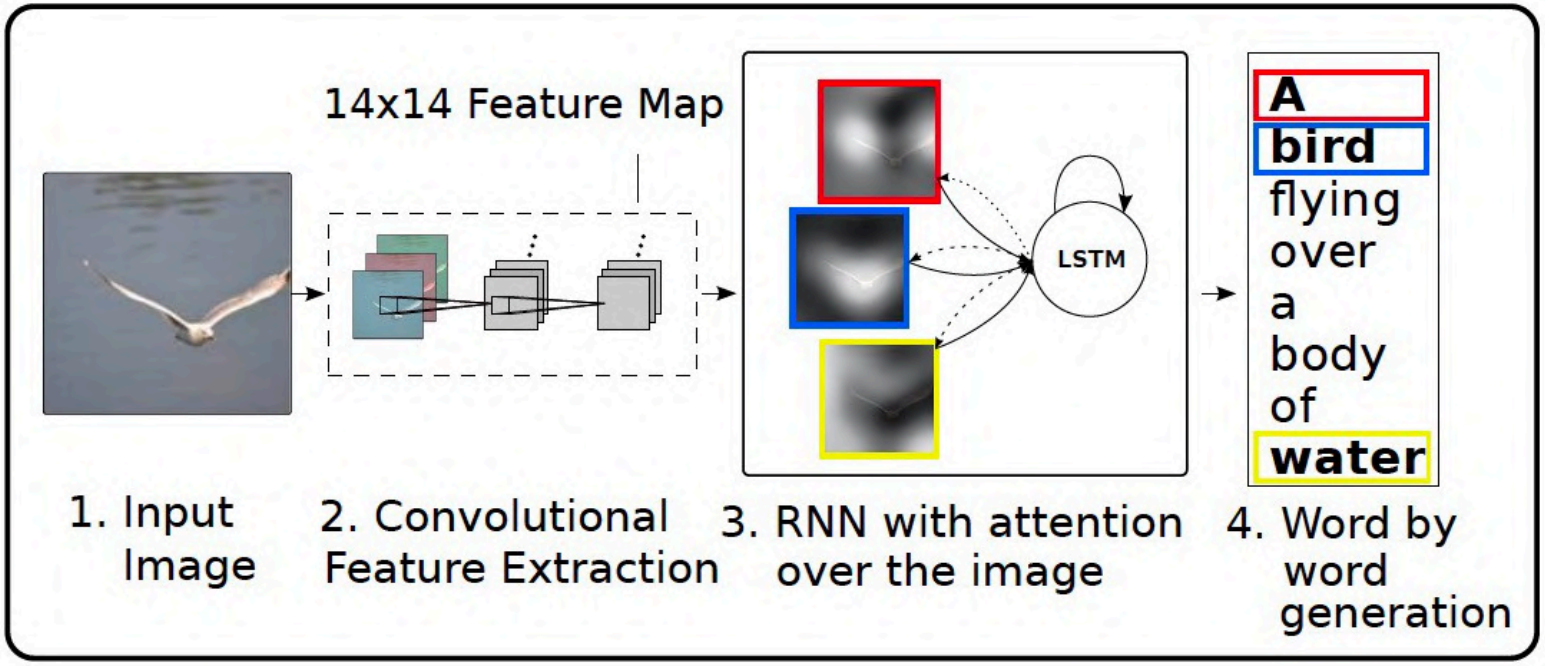
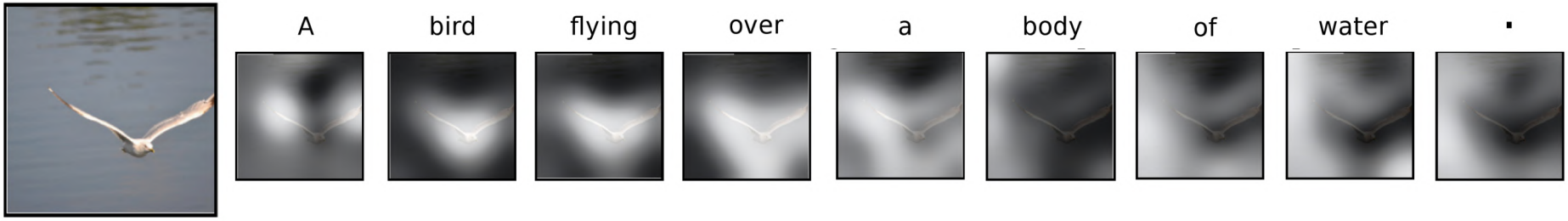


Image captioning with RNNs and attention

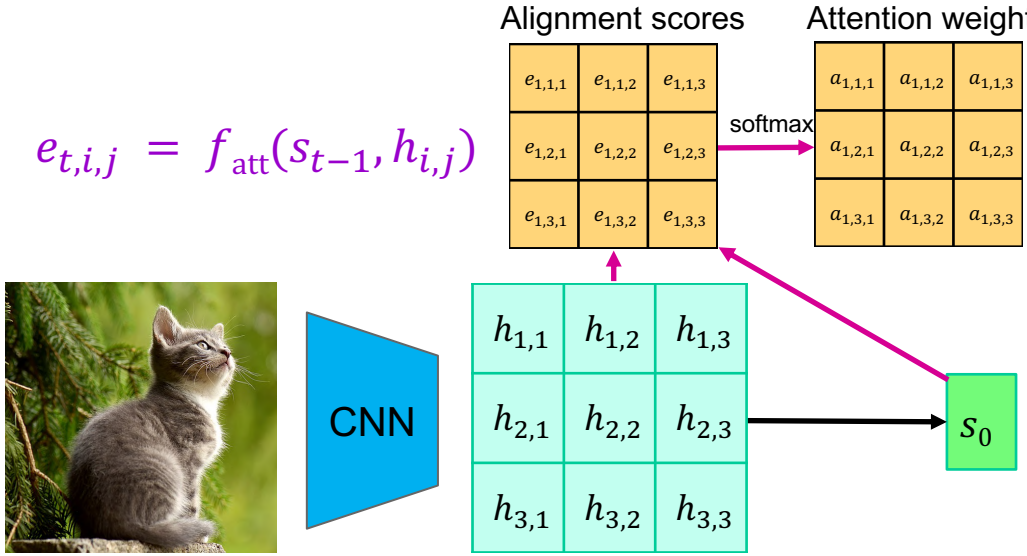
- Idea: pay attention to different parts of the image when generating different words
- Automatically learn this *grounding* of words to image regions without direct supervision



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

Image captioning with RNNs and attention

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



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

Image captioning with RNNs and attention

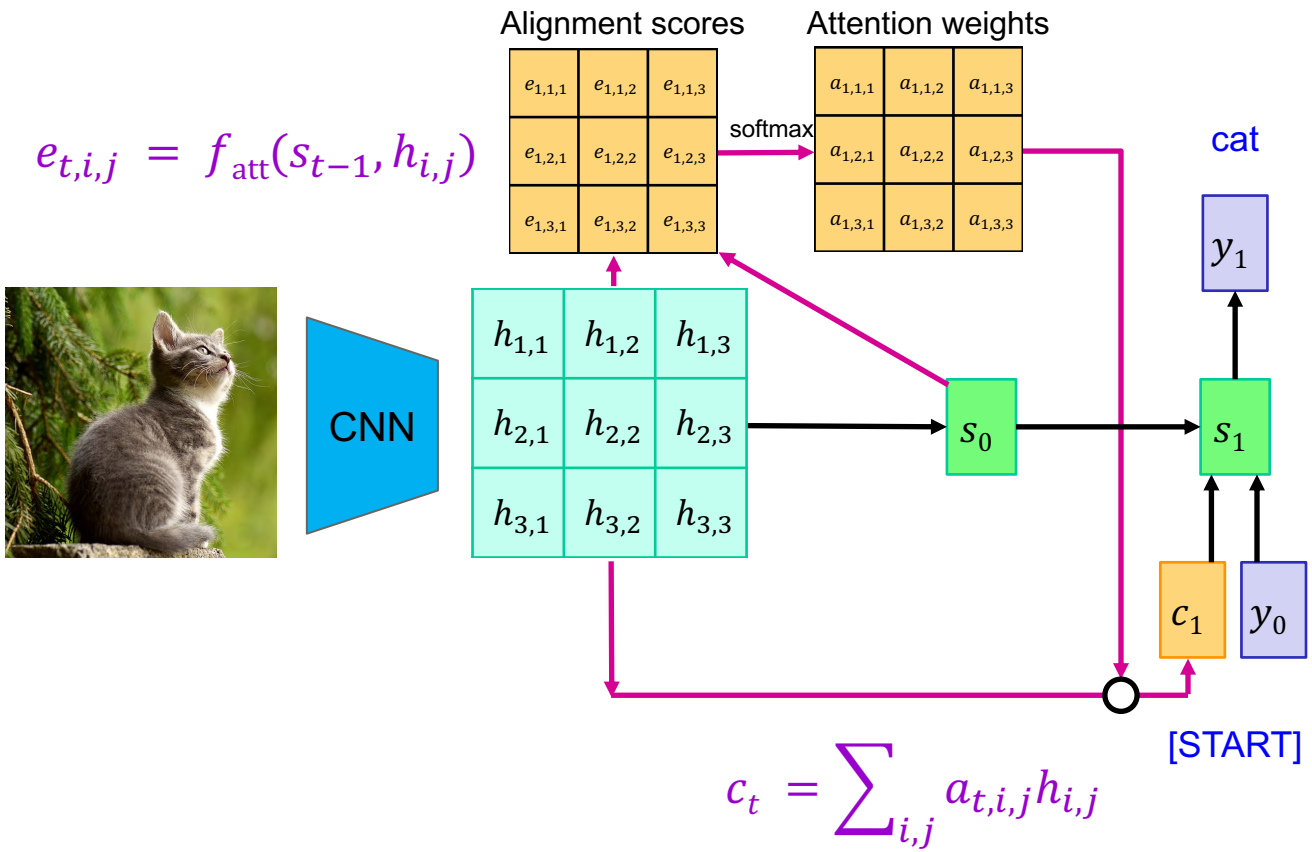


Image captioning with RNNs and attention

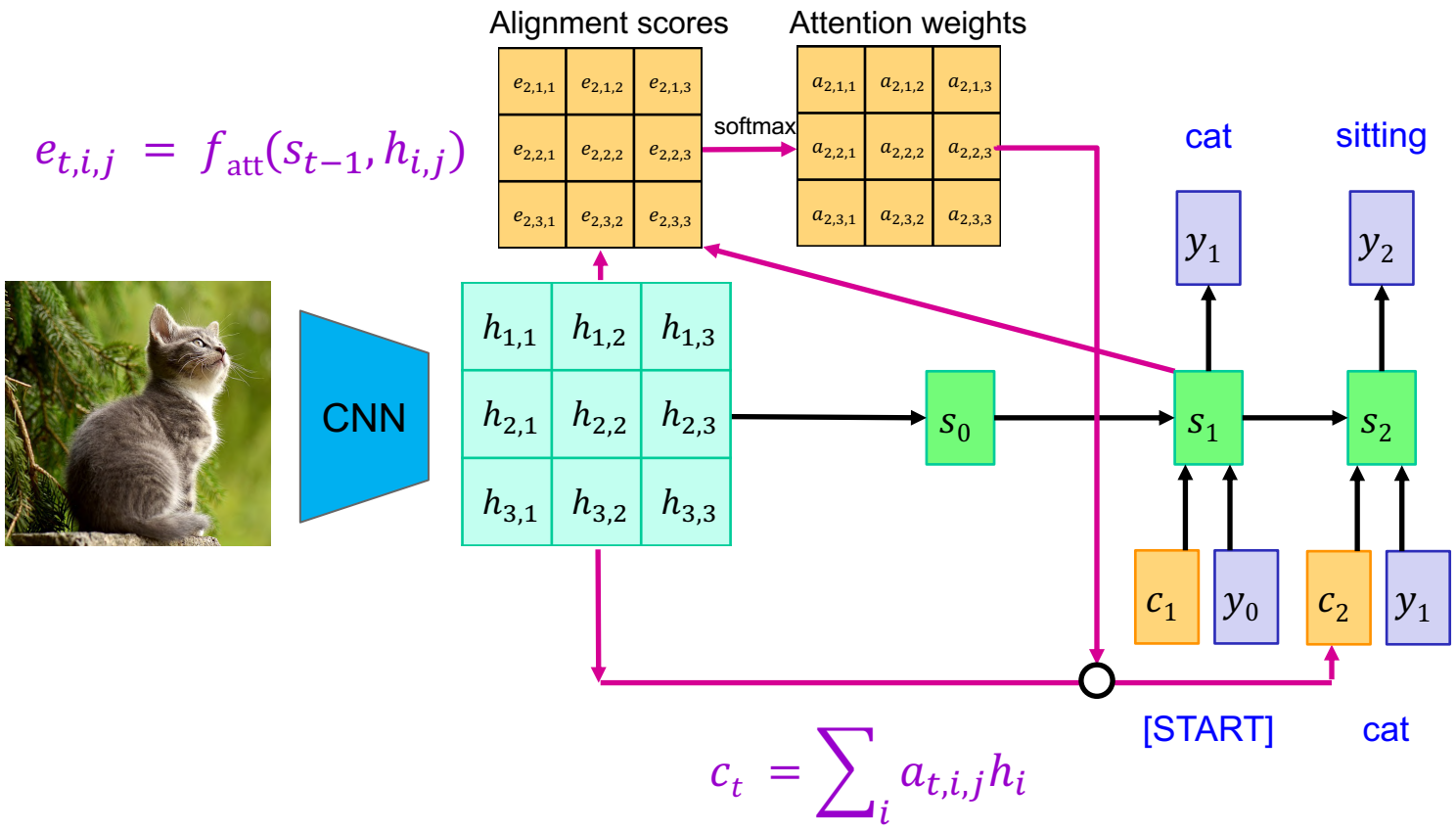
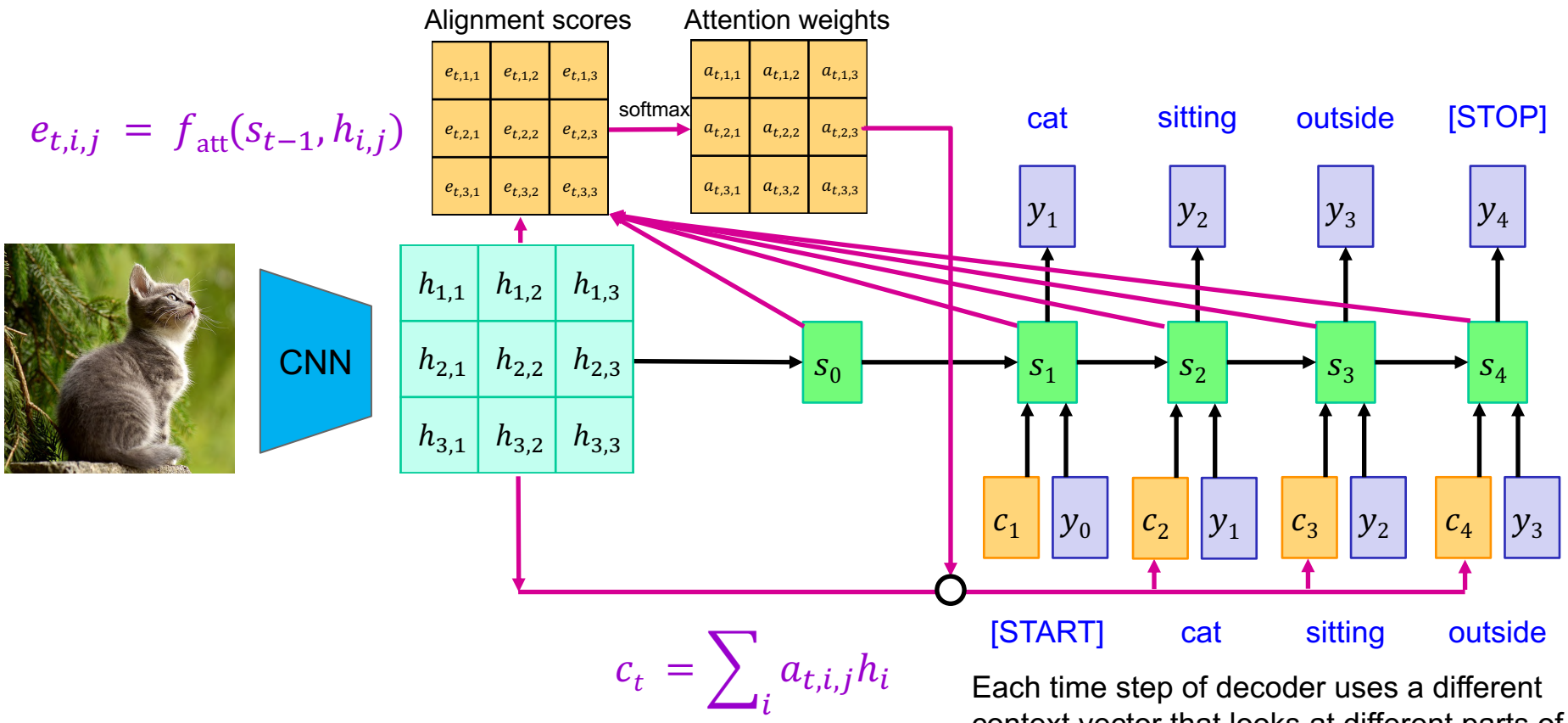


Image captioning with RNNs and attention



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

Example results

- Good captions

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

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

A little girl sitting on a bed with a teddy bear.

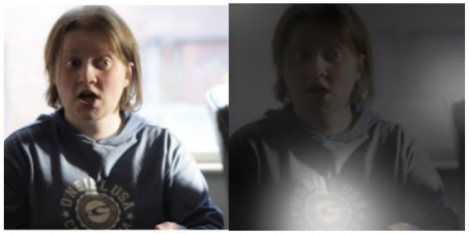
A group of people sitting on a boat in the water.

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

Example results

- Mistakes

A large white bird standing in a forest.



A woman holding a clock in her hand.

A man wearing a hat and
a hat on a skateboard.

A person is standing on a beach
with a surfboard.

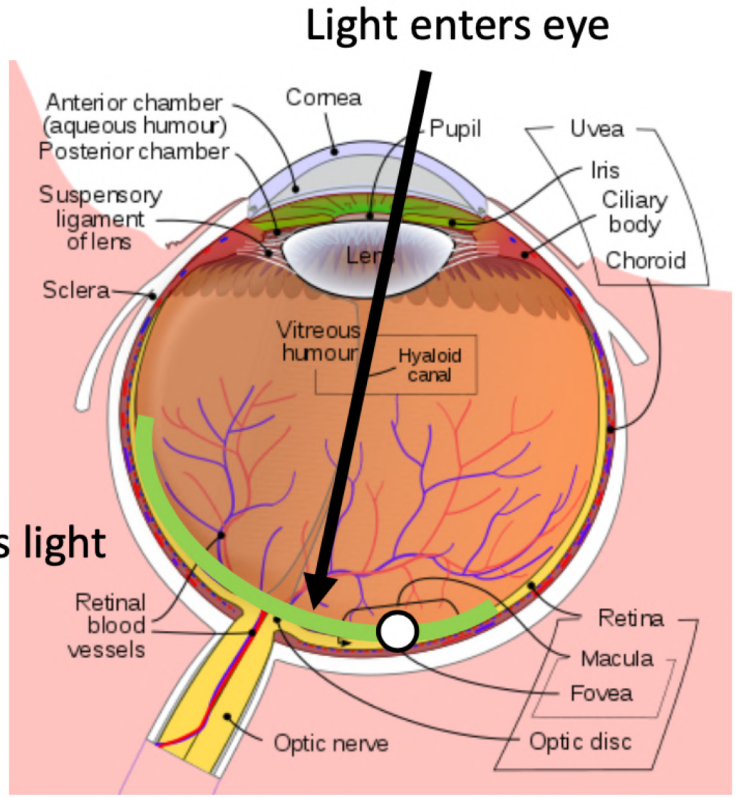
A woman is sitting at a table
with a large pizza.

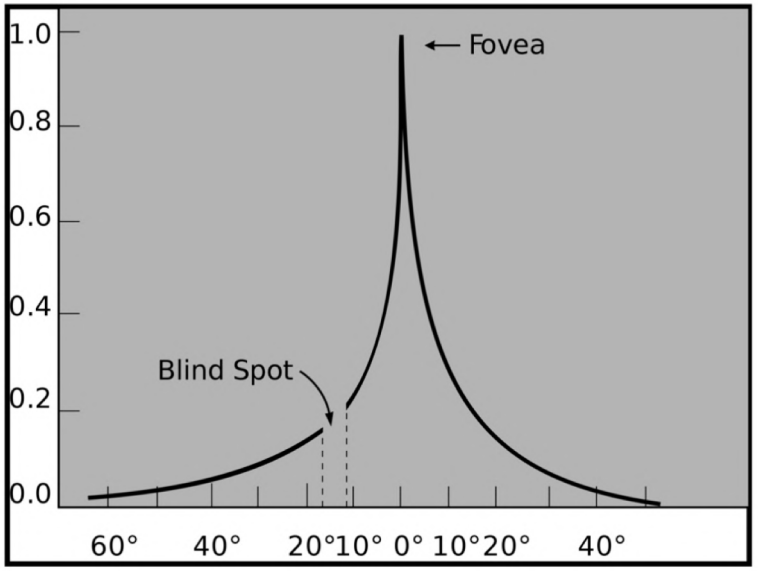
A man is talking on his cell phone
while another man watches.

Motivation

The Fovea in Human Vision:

- The Fovea is a tiny region in the retina that can see in high acuity.

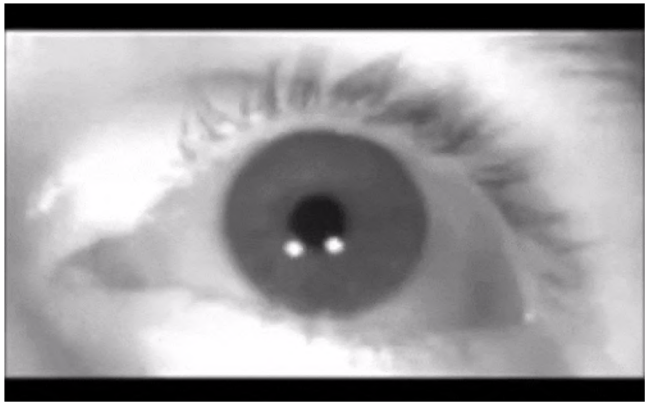


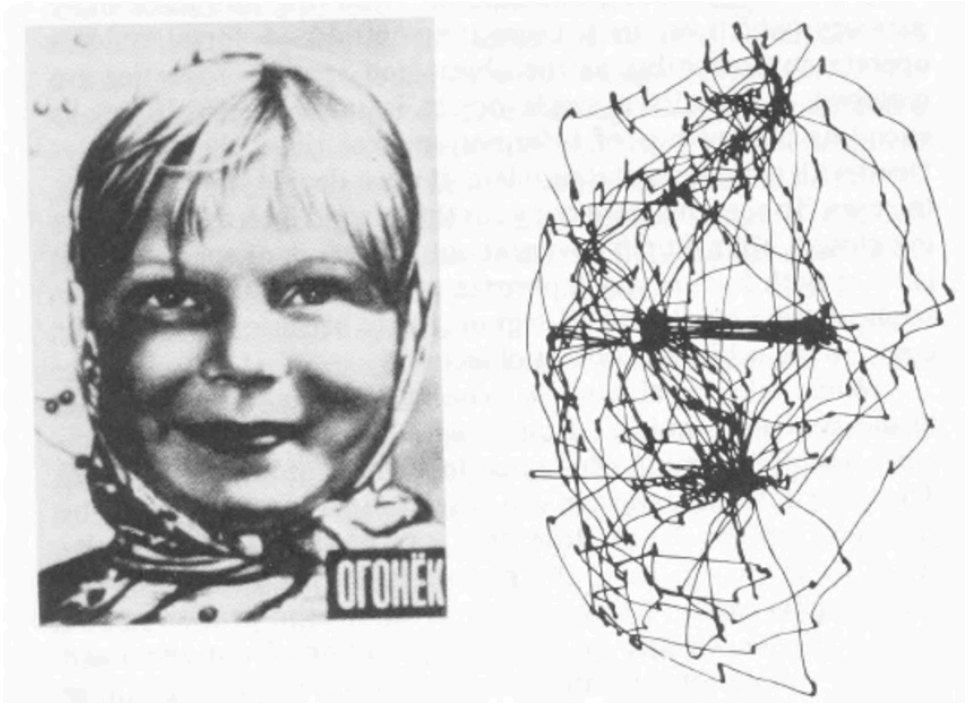


Motivation

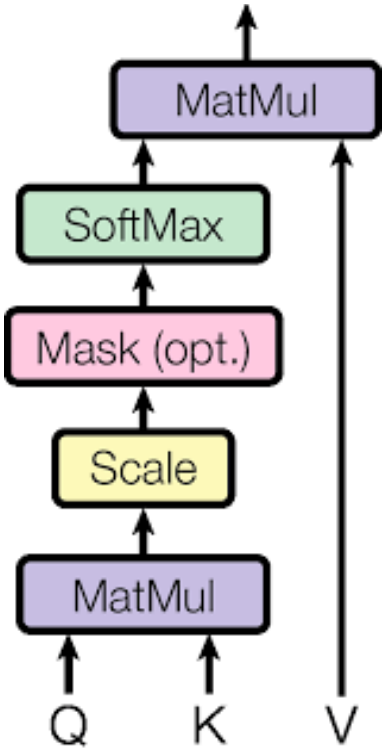
The Fovea in Human Vision:

- The Fovea is a tiny region in the retina that can see in high acuity.
- Saccade: Human eyes are constantly moving to ”fill the gaps”.
- Attention weights at each time step is similar to the saccades in the human eye.

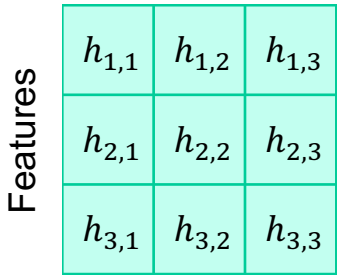


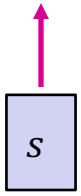


A General Attention Model



We saw Attention in image captioning

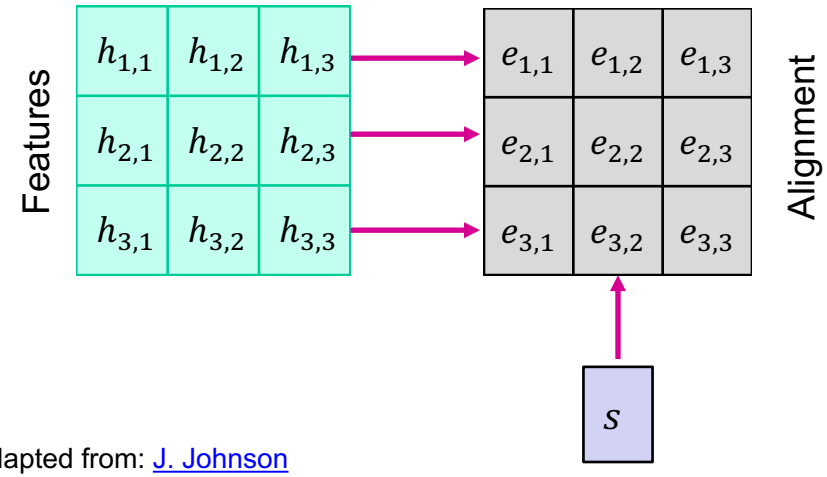




Inputs:
Features: $\boldsymbol{h}$ $(shape\colon H \times W \times D)$
Query: $\boldsymbol{s}$ $(shape\colon D)$

Adapted from: J. Johnson

Attention we saw in image captioning



Operations:

Alignments: $e_{i,j} = f_{att}(s, h_{i,j})$

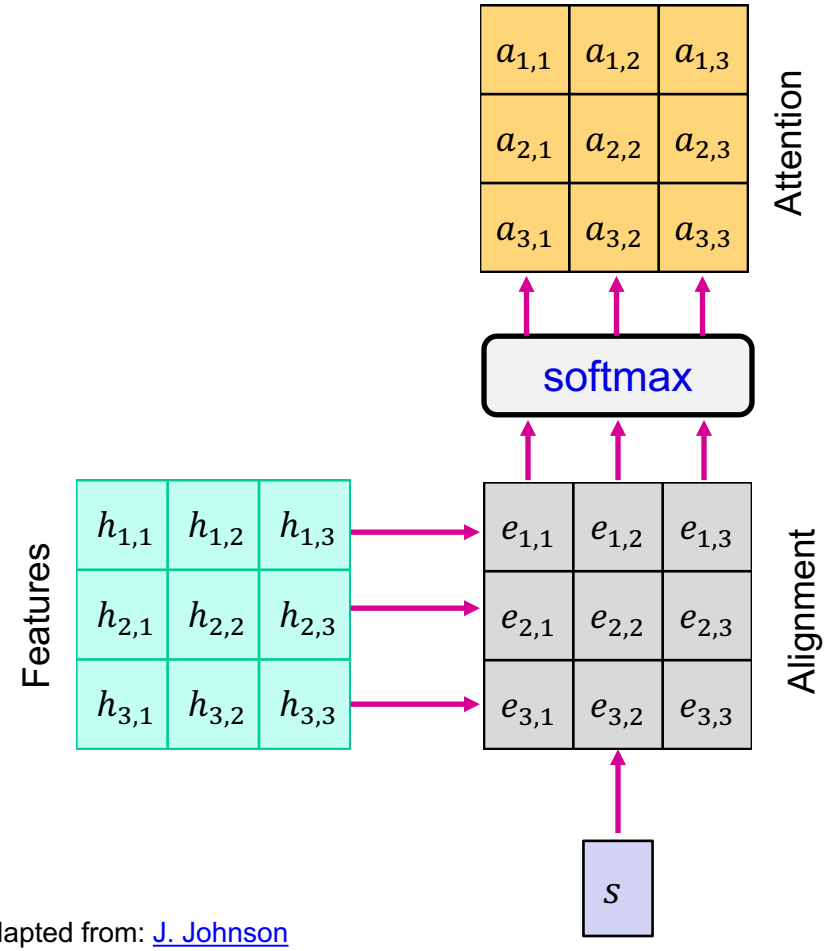
Inputs:

Features: $\boldsymbol{h}$ $(shape: H \times W \times D)$

Query: $\boldsymbol{s}$ $(shape: D)$

Adapted from: J. Johnson

Attention we saw in image captioning



Operations:

Alignments: $e_{i,j} = f_{att}(s, h_{i,j})$

Attention: $\boldsymbol{a} = softmax(\boldsymbol{e})$

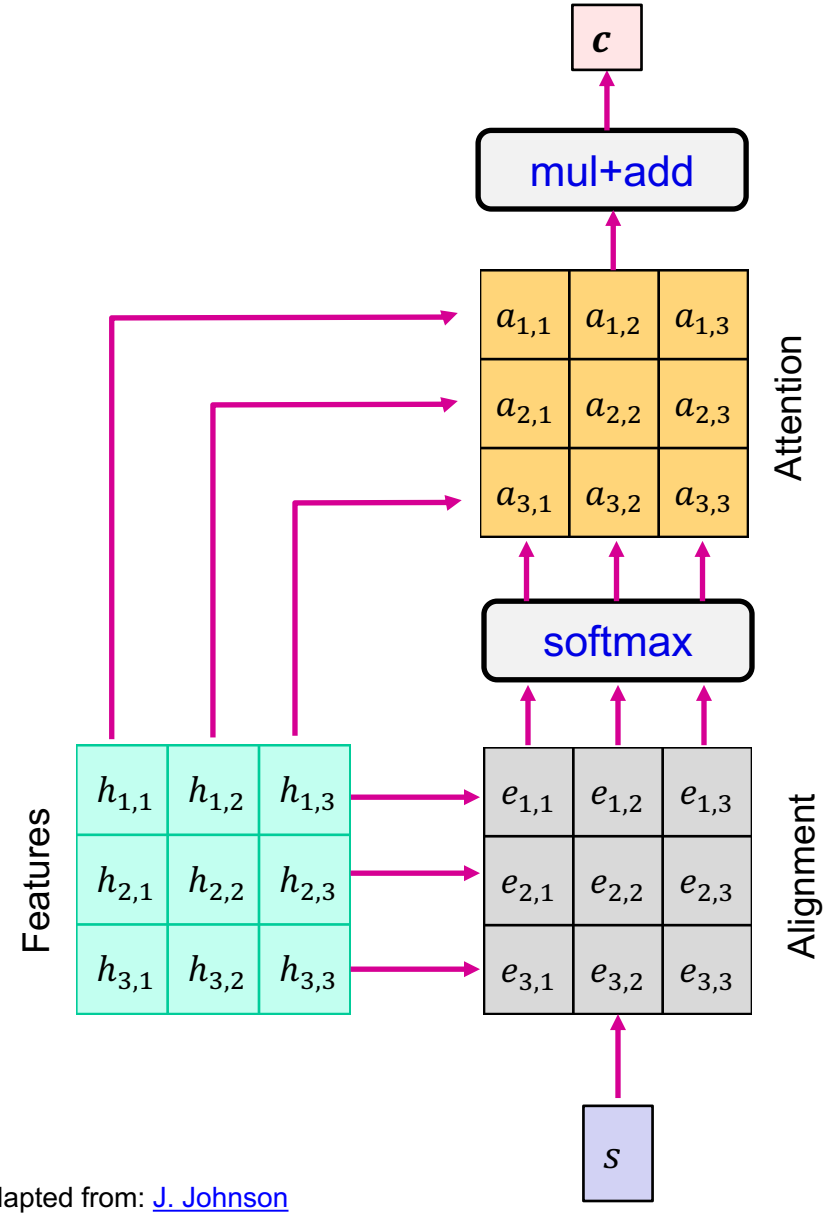
Inputs:

Features: $\boldsymbol{h}$ $(shape: H \times W \times D)$

Query: $\boldsymbol{s}$ $(shape: D)$

Adapted from: J. Johnson

Attention we saw in image captioning



Outputs:

Context vector: $\boldsymbol{c}$ (shape: D)

Operations:

Alignments: $e_{i,j} = f_{att}(s, h_{i,j})$

Attention: $\boldsymbol{a} = softmax(\boldsymbol{e})$

Output: $\boldsymbol{c} = \sum_{i,j} a_{i,j} \boldsymbol{h}_{i,j}$

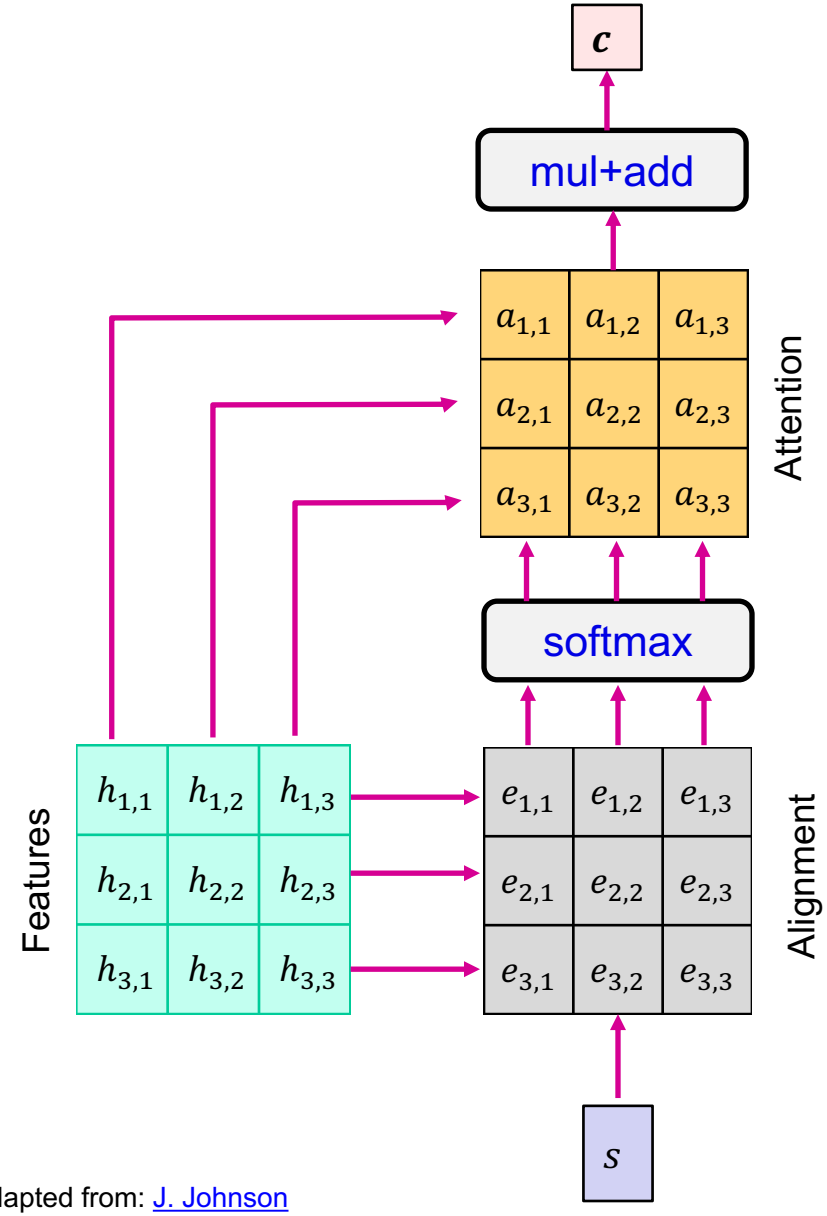
Inputs:

Features: $\boldsymbol{h}$ $(shape: H \times W \times D)$

Query: $\boldsymbol{s}$ $(shape: D)$

Adapted from: J. Johnson

Attention we saw in image captioning

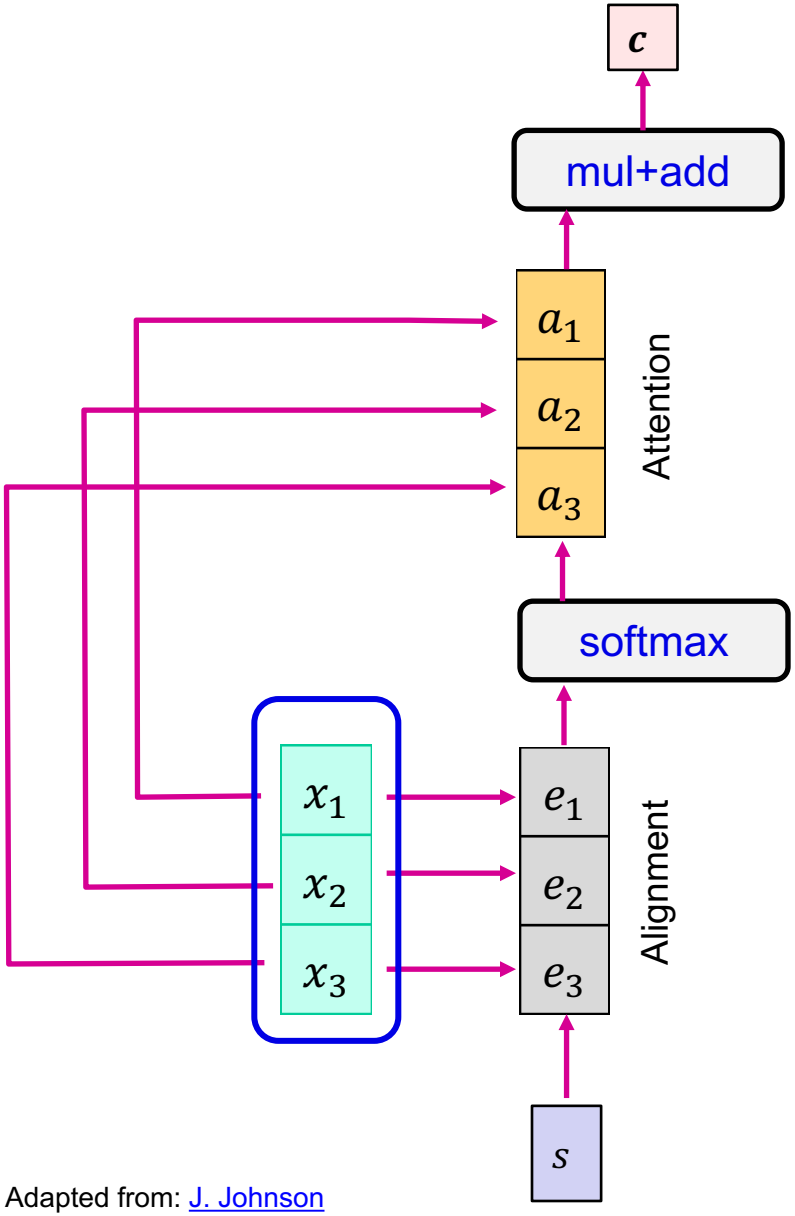


General attention Layer

- Attention operation is **permutation invariant.**
- Doesn't care about ordering of the features
- Stretch H x W = N feature vectors into N vectors x_i

Adapted from: J. Johnson

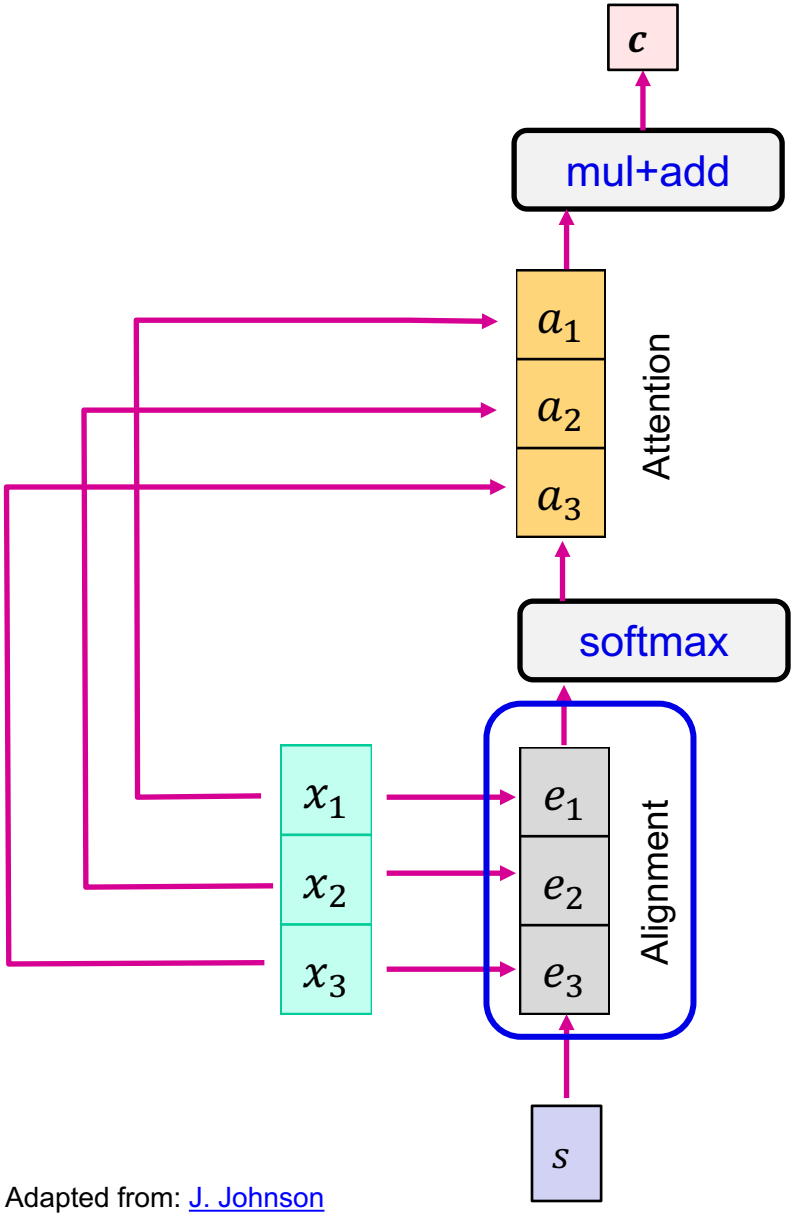
General attention layer



Inputs:
Input vectors: $\mathrm{X} = [x_1, ..]^T$ $(shape: N \times D)$
Query: $\boldsymbol{s}$ $(shape: D)$

Adapted from: J. Johnson

General attention layer



Outputs:

Operations: Scaled inner product

~~Alignments: $e_{t,j} = f_{att}(s, h_{t,j})$~~

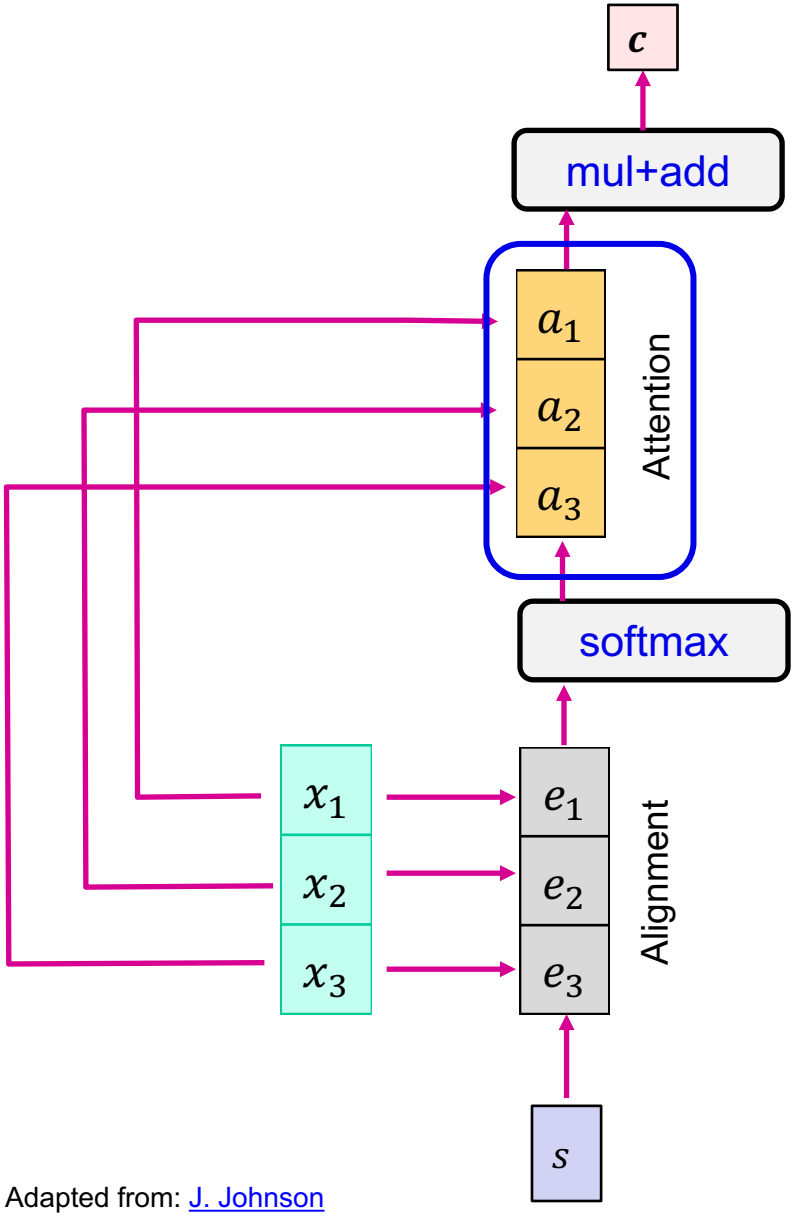
Alignments: $e_i = \boldsymbol{s} \cdot \boldsymbol{x}_i \,/\sqrt{D}$

Recall a · b = |a||b| cos(angle)
Suppose that a and b are constant vectors of dimension D.
Then, $|a| = (\sum_i a^2)^{1/2} = a\sqrt{D}$

Inputs:
Input vectors: $\mathrm{X} = [x_1, ..]^T$ $(shape: N \times D)$
Query: $\boldsymbol{s}$ $(shape: D)$

Adapted from: J. Johnson

General attention layer



Outputs:

Operations:

~~Alignments: $e_{i,j} = f_{att}(s, h_{i,j})$~~

Alignments: $e_i = \boldsymbol{s} \cdot \boldsymbol{x}_i \,/\sqrt{D}$

Attention: $\boldsymbol{a} = softmax(\boldsymbol{e})$ (shape: N)

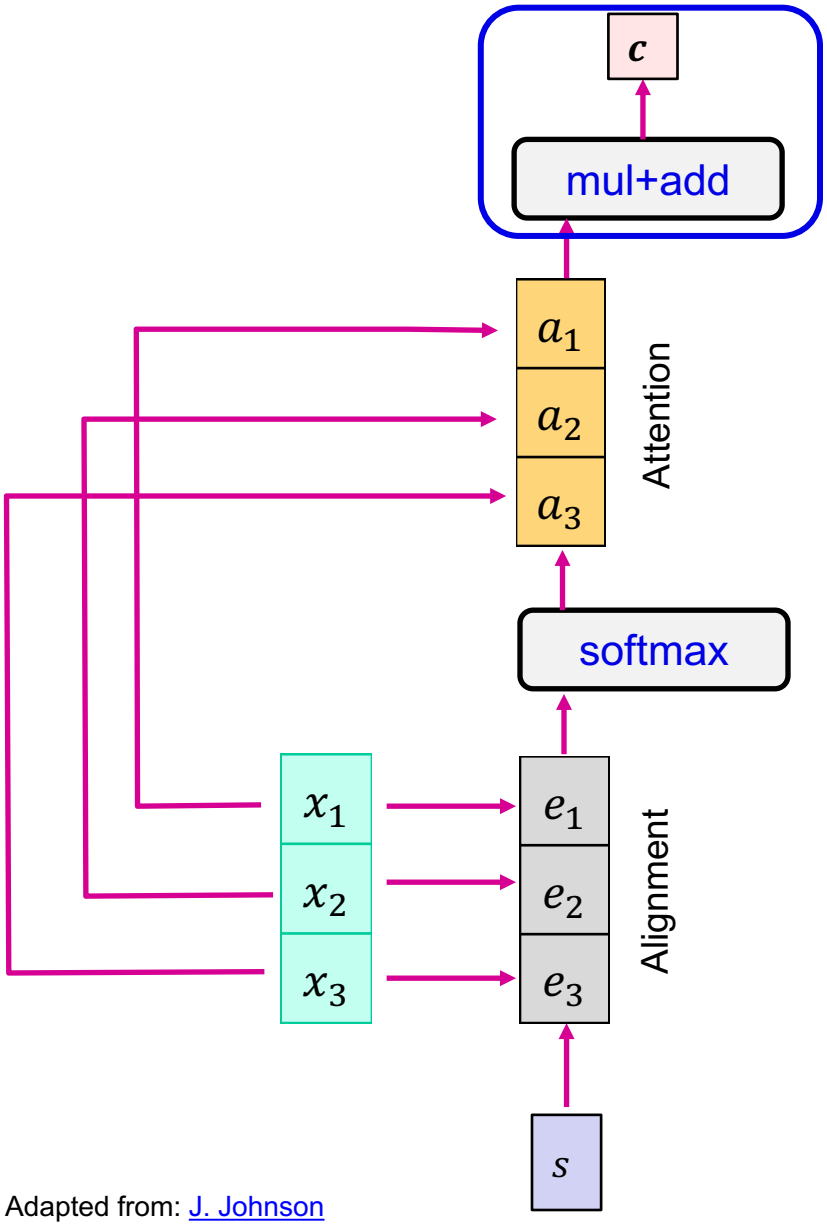
Inputs:

Input vectors: $\mathrm{X} = [x_1, ..]^T$ $(shape: N \times D)$

Query: $\boldsymbol{s}$ $(shape: D)$

Adapted from: J. Johnson

General attention layer



Outputs:

$\boldsymbol{c} = \sum_i a_i \, x_i$ (shape: D)

Operations:

~~Alignments: $e_{t,j} = f_{att}(s, h_{t,j})$~~

Alignments: $e_i = \boldsymbol{s} \cdot \boldsymbol{x}_i \, / \sqrt{D}$

Attention: $\boldsymbol{a} = softmax(\boldsymbol{e})$ (shape: N)

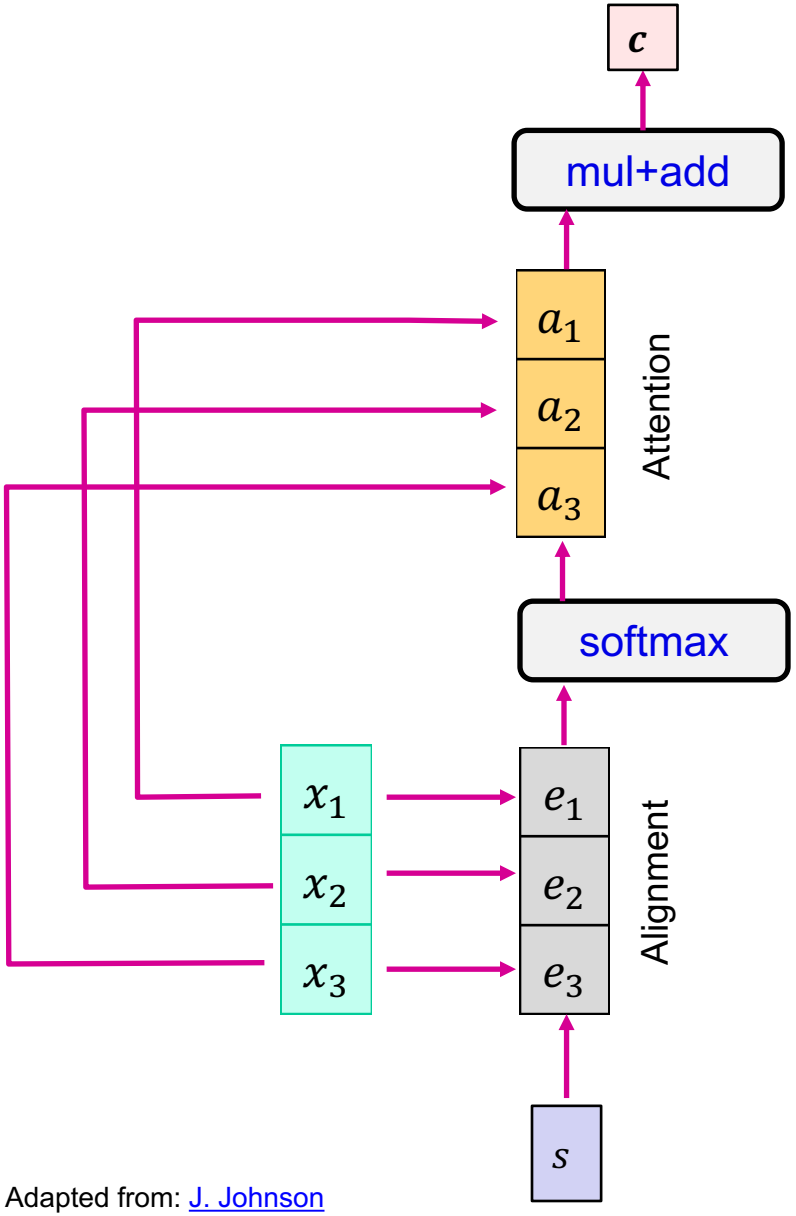
Inputs:

Input vectors: $\mathrm{X} = [x_1, . .]^T$ $(shape: N \times D)$

Query: $\boldsymbol{s}$ $(shape: D)$

Adapted from: J. Johnson

General attention layer: multiple queries

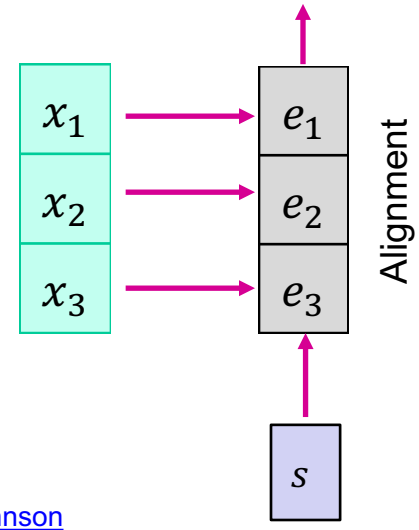


Multiple Queries

- We can extend the attention layer into multiple query vectors
- Each query creates a new output context vector

Adapted from: J. Johnson

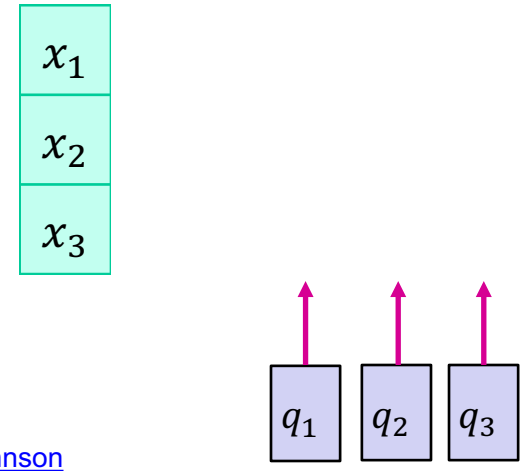
General attention layer: multiple queries



Inputs:
Input vectors: $\mathrm{X} = [x_1, ..]^T$ $(shape: N \times D)$
Query: $\boldsymbol{s}$ $(shape: D)$

Adapted from: J. Johnson

General attention layer



Inputs:
Input vectors: $\mathrm{X} = [x_1, ..]^T$ $(shape: N \times D)$
Query: $\boldsymbol{Q} = [\boldsymbol{q_1}, \boldsymbol{q_2}, ...]^T$ $(shape: M \times D)$

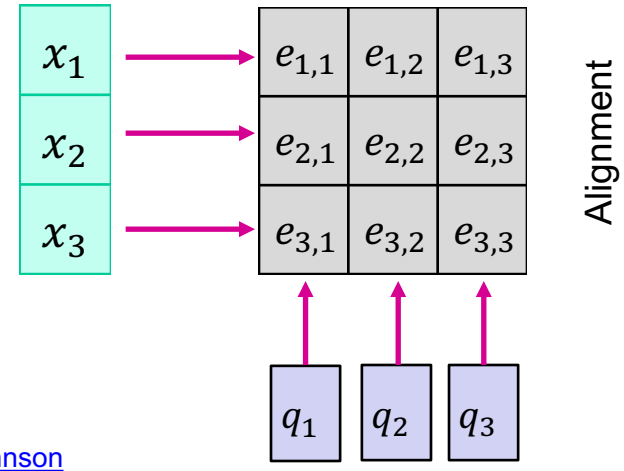
Adapted from: J. Johnson

General attention layer

Operations:

Alignments: $e_{i,j} = \boldsymbol{x}_i \cdot \boldsymbol{q}_j / \sqrt{D}$

$E = XQ^T / \sqrt{D}$ (shape: $N \times M$)



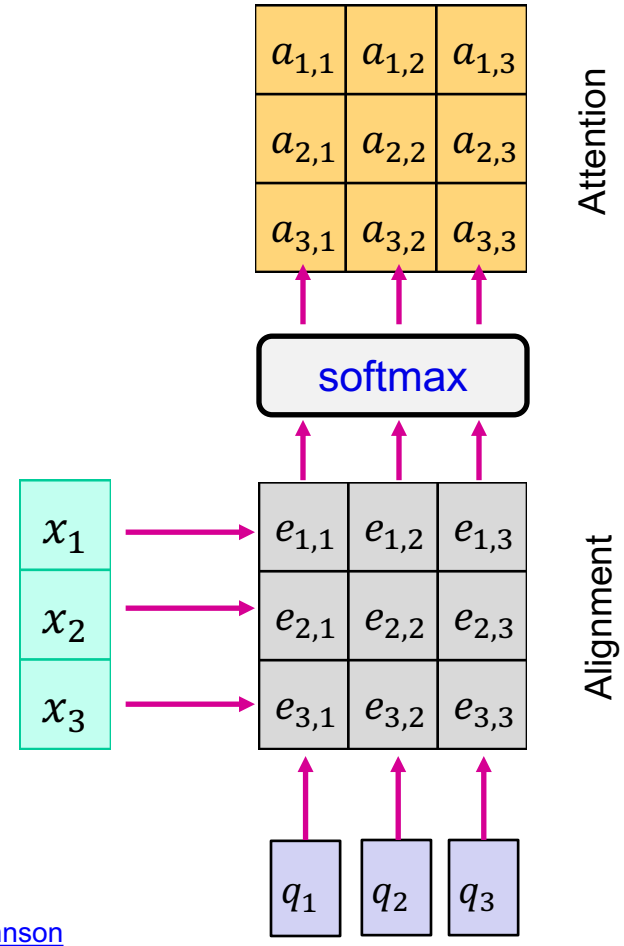
Inputs:

Input vectors: $\mathrm{X} = [x_1, ..]^T$ $(shape: N \times D)$

Query: $\boldsymbol{Q} = [\boldsymbol{q_1}, \boldsymbol{q_2}, ...]^T$ $(shape: M \times D)$

Adapted from: J. Johnson

General attention layer



Operations:

Alignments: $e_{i,j} = \boldsymbol{x}_i \cdot \boldsymbol{q}_j / \sqrt{D}$

$E = XQ^T / \sqrt{D}$ (shape: $N \times M$)

Attention: $a_{*,j} = \text{softmax}(e_{*,j})$

$A = \text{softmax}(E, dim = 1) \quad (N \times M)$

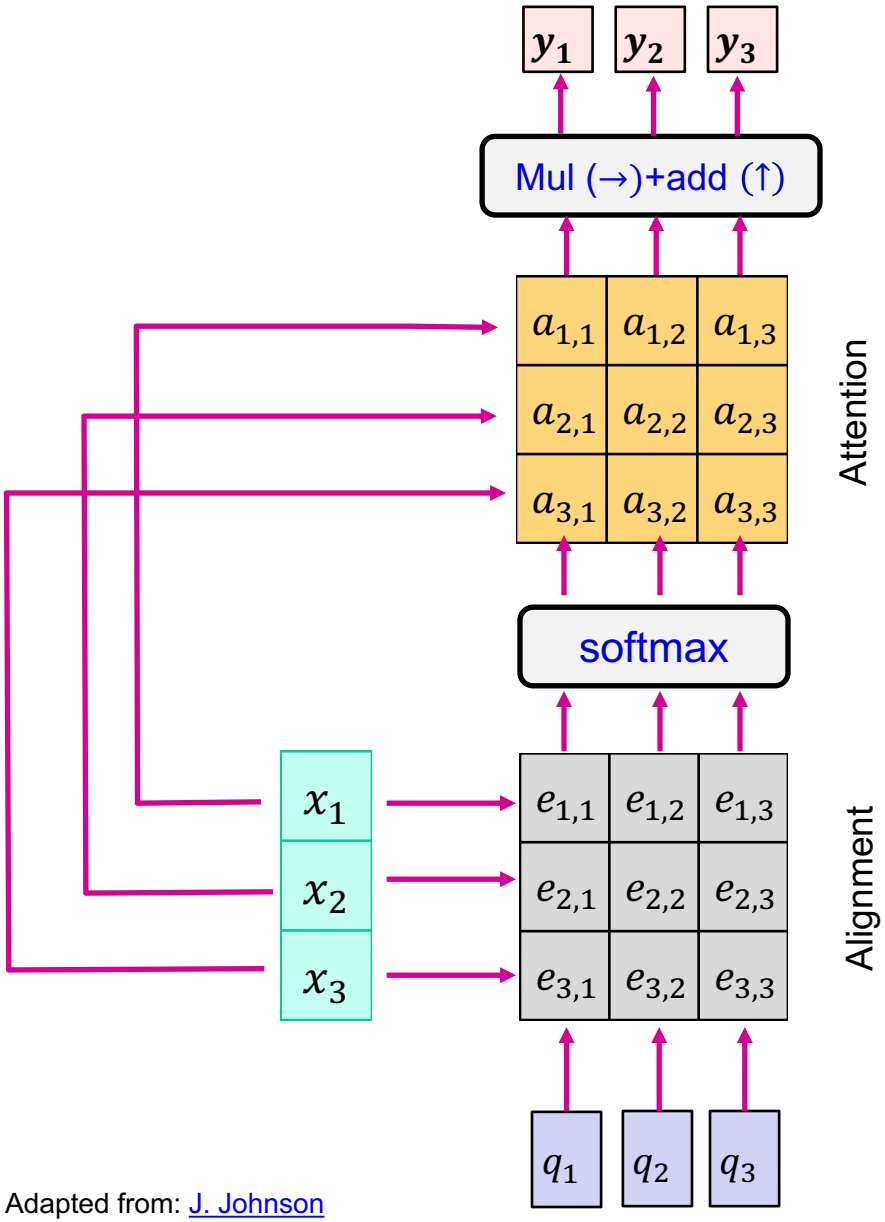
Inputs:

Input vectors: $\mathrm{X} = [x_1, ..]^T$ $(shape: N \times D)$

Query: $\boldsymbol{Q} = [\boldsymbol{q_1}, \boldsymbol{q_2}, \ldots]^T$ $(shape: M \times D)$

Adapted from: J. Johnson

General attention layer



Outputs:

$\boldsymbol{y}_j = \sum_i a_{i,j} x_i$; $\boldsymbol{y} = A^T X$ (shape: M×D)

Operations:

Alignments: $e_{i,j} = \boldsymbol{x}_i \cdot \boldsymbol{q}_j / \sqrt{D}$

$E = XQ^T/\sqrt{D}$ (shape: $N \times M$)

Attention: $a_{*,j} = \text{softmax}(e_{*,j})$

$A = \text{softmax}(E, dim = 1)$ $(N \times M)$

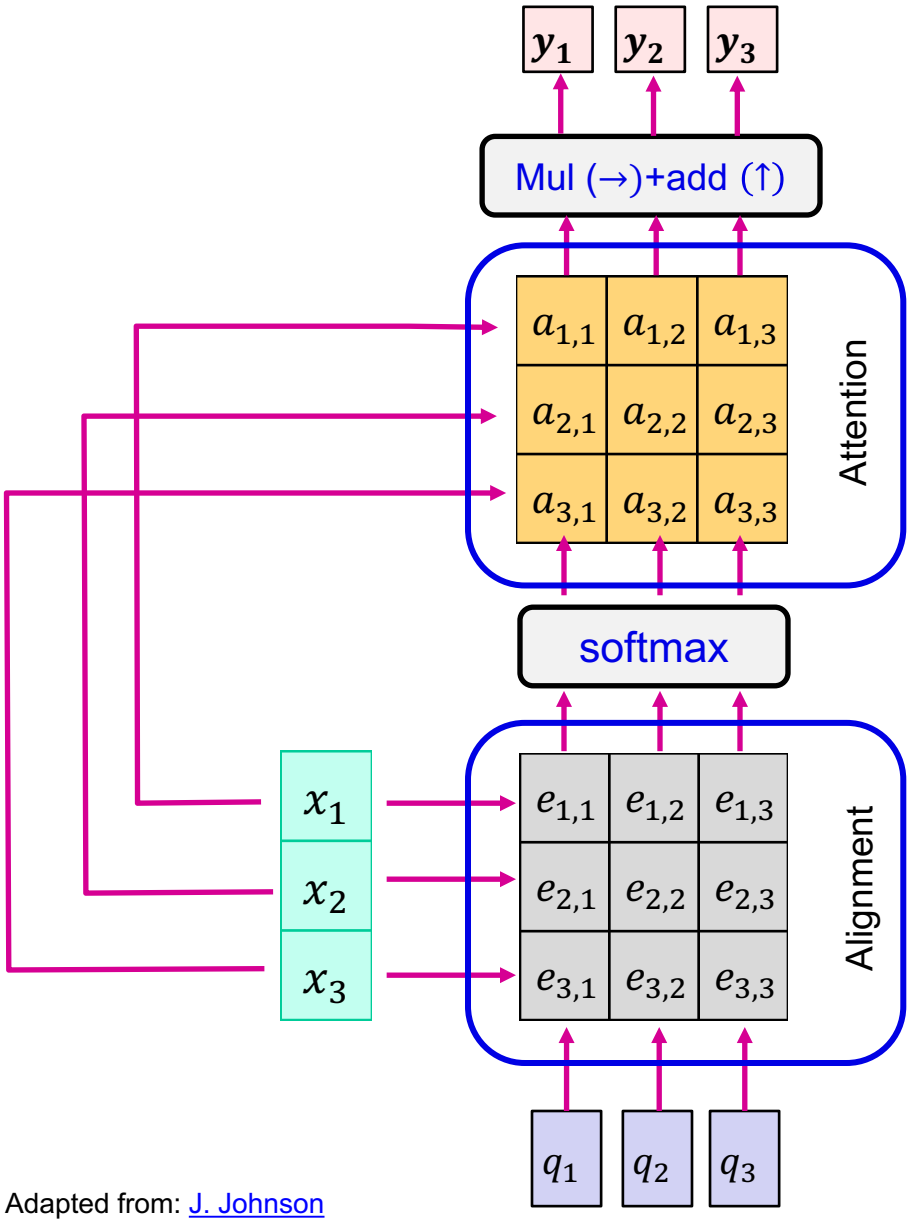
Inputs:

Input vectors: $\text{X} = [x_1, ..]^T$ $(shape: N \times D)$

Query: $\boldsymbol{Q} = [\boldsymbol{q_1}, \boldsymbol{q_2}, ...]^T$ $(shape: M \times D)$

Adapted from: J. Johnson

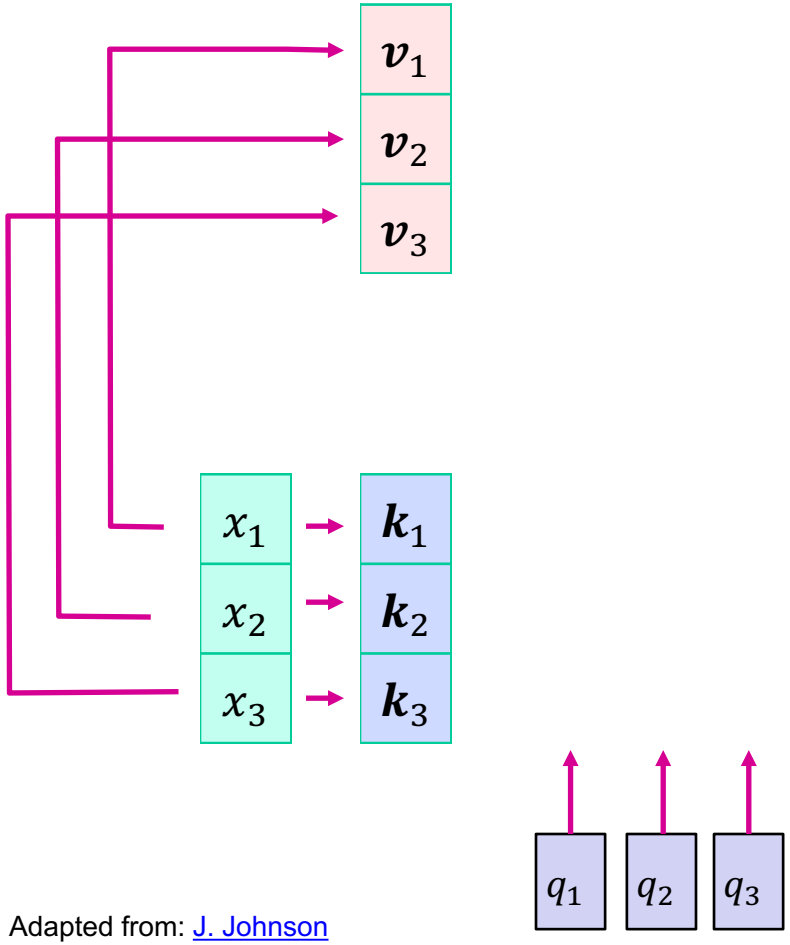
General attention layer



- Notice, that the input vectors are used for both, the alignment as well as the attention calculations.
- In this scheme the attention model is fixed and not learned.
- We can add more expressivity to the layer by adding a different FC layer before each of the two steps.

Adapted from: J. Johnson

General attention layer: adding keys/values



Operations:

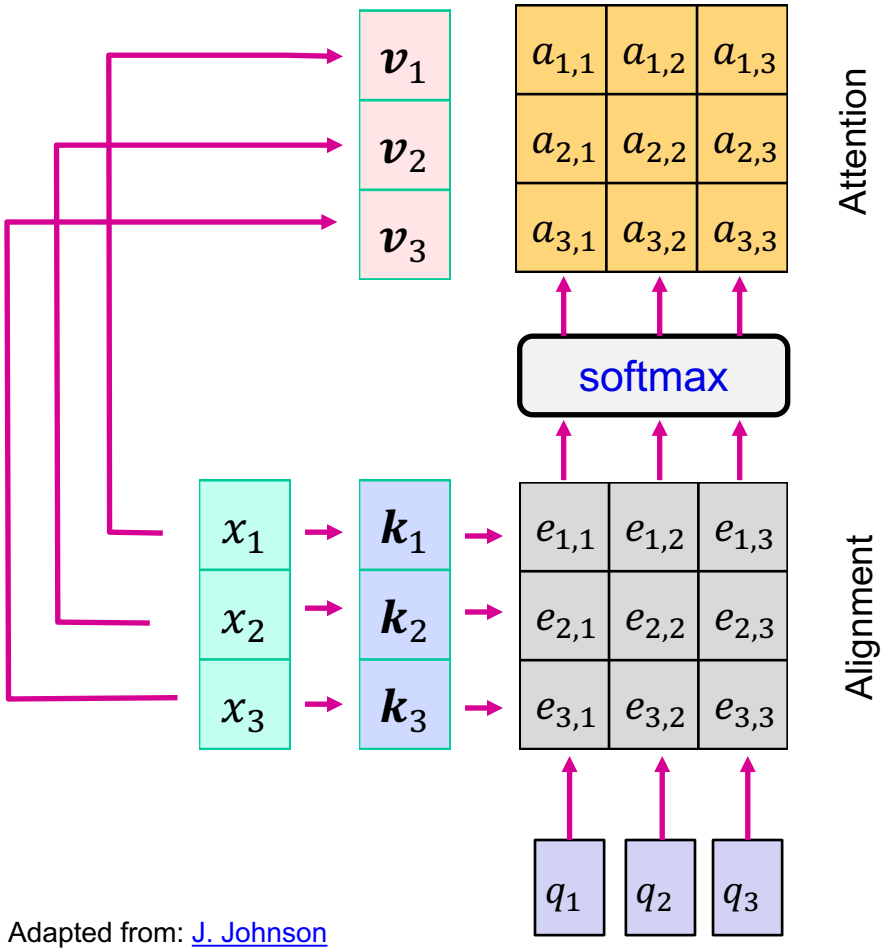
Key vectors: $\boldsymbol{k_i} = W_k\, \boldsymbol{x_i}$ (shape: D_k)

Value vectors: $\mathbf{v_i} = W_v\, \boldsymbol{x_i}$ (shape: D_v)

Both W_k and W_v are learned in the training phase

Adapted from: J. Johnson

General attention layer



Operations:

Key vectors: $\boldsymbol{k_i} = W_k\, \boldsymbol{x_i}$ (shape: D_k)

Value vectors: $\mathbf{v_i} = W_v\, \boldsymbol{x_i}$ (shape: D_v)

Alignments: $e_{i,j} = \boldsymbol{k}_i \cdot \boldsymbol{q}_j / \sqrt{D}$ $(N \times M)$

Attention: $a_{*,j} = \mathrm{softmax}(e_{*,j})$ $(N \times M)$

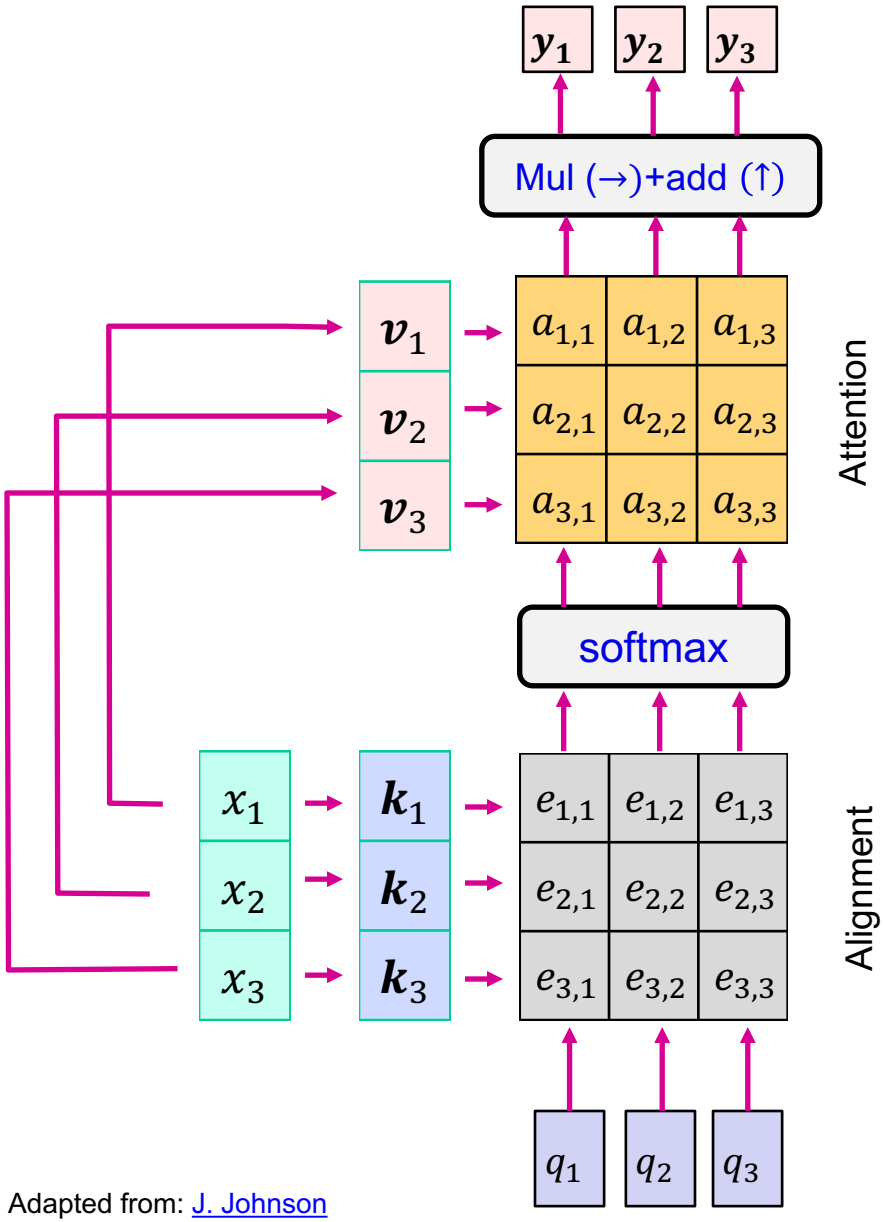
Inputs:

Input vectors: $\mathrm{X} = [x_1, ..]^T$ $(shape: N \times D)$

Query: $\boldsymbol{Q} = [\boldsymbol{q_1}, \boldsymbol{q_2}, ...]^{\boldsymbol{T}}$ $(shape: M \times D_k)$

Adapted from: J. Johnson

General attention layer



Outputs:

$$y_j = \sum_i a_{i,j}\, v_i \;\; ; \;\; \mathrm{Y} \;\; (\mathrm{M}\times D_v)$$

Operations:

Key vectors: $k_i = W_k\, x_i$ (shape: D_k)

Value vectors: $\mathbf{v_i} = W_v\, x_i$ (shape: D_v)

Alignments: $e_{i,j} = k_i \cdot q_j/\sqrt{D}$ $(N\times M)$

Attention: $a_{*,j} = \mathrm{softmax}(e_{*,j})$ $(N\times M)$

Inputs:

Input vectors: $\mathrm{X} = [x_1, ..]^T$ $(shape: N\times D)$

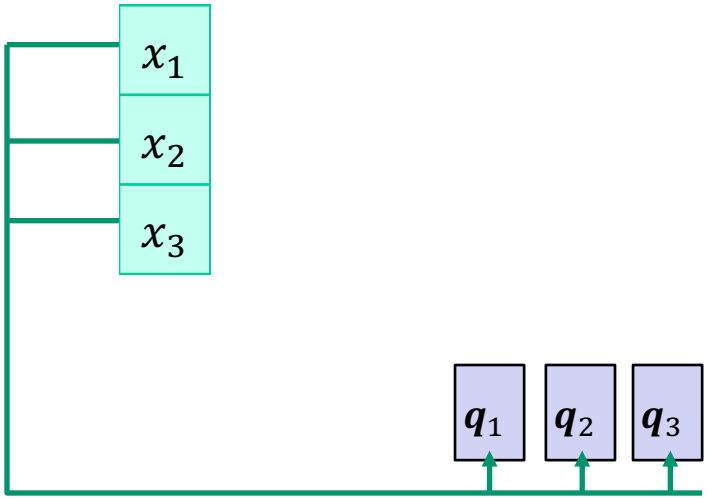
Query: $\boldsymbol{Q} = [\boldsymbol{q_1}, \boldsymbol{q_2}, \ldots]^{\boldsymbol{T}}$ $(shape: M\times D_k)$

Adapted from: J. Johnson

General attention layer: The self-attention

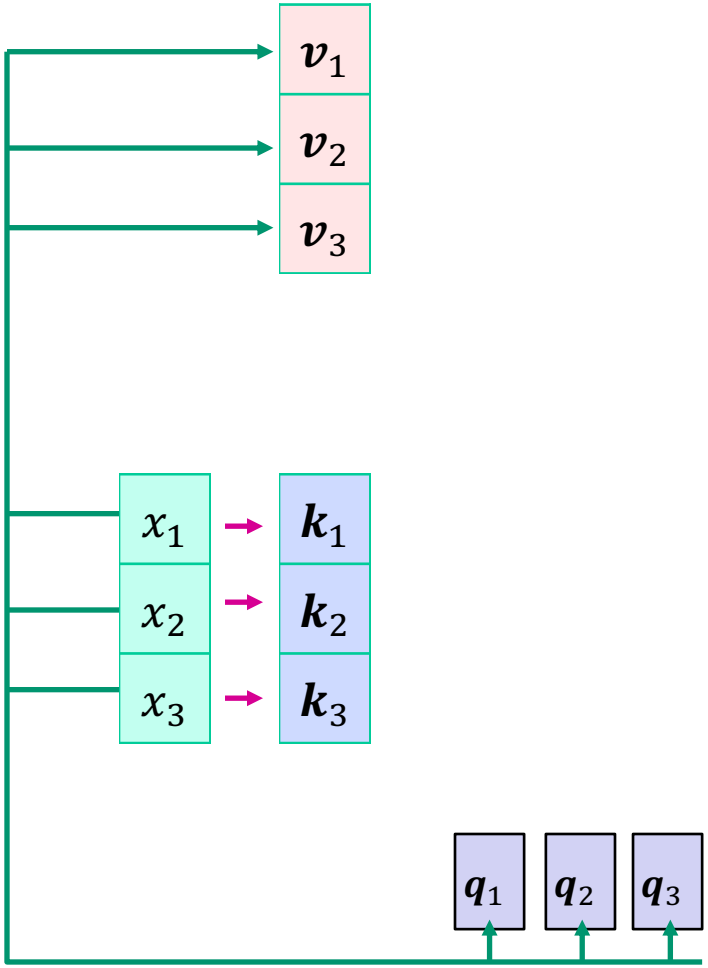
Self attention layer

- We calculate also the query vectors from the input vectors
- This defines a "self-attention" layer.
- Query vectors: $\boldsymbol{q} = W_q\, \boldsymbol{x}$



Adapted from: J. Johnson

Self attention layer



Operations:

Query vectors: $\boldsymbol{q} = W_q\,\boldsymbol{x}$ $(N \times D_k)$

Key vectors: $\boldsymbol{k} = W_k\,\boldsymbol{x}$ $(N \times D_k)$

Value vectors: $\mathbf{v} = W_v\,\boldsymbol{x}$ $(N \times D)$

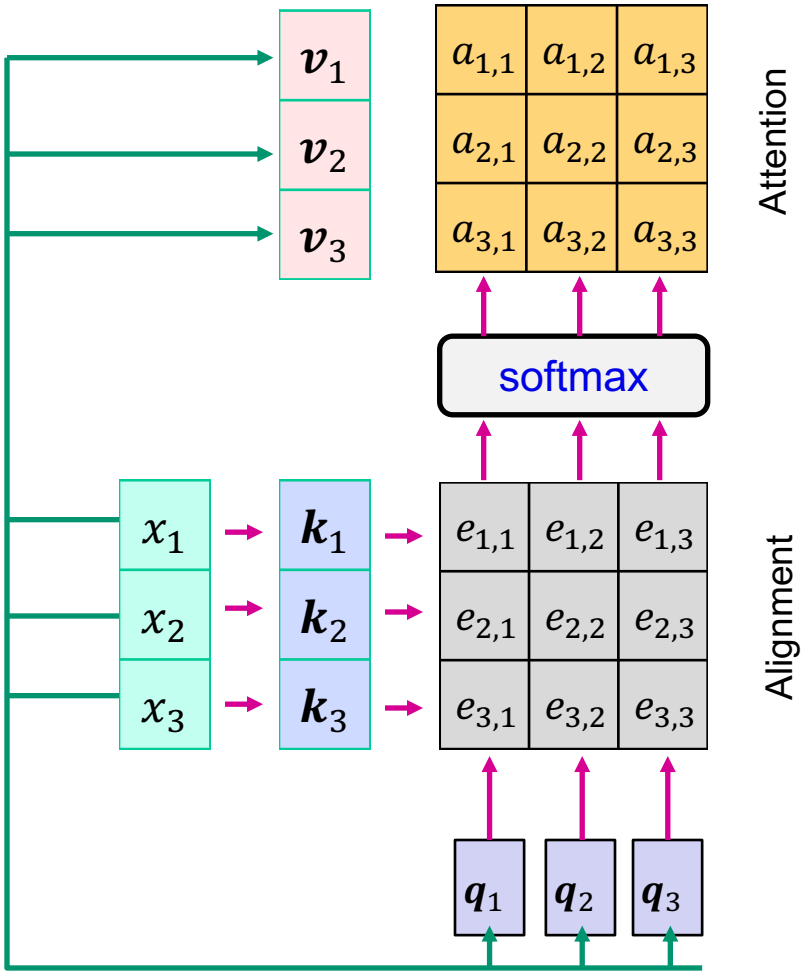
All matrices W_q and W_k and W_v are learned in the training phase

Inputs:

Input vectors: $\boldsymbol{x}\ (shape: N \times D)$

Adapted from: J. Johnson

Self attention layer



Operations:

Query vectors: $\boldsymbol{q} = W_q\,\boldsymbol{x}$ $(N \times D_k)$

Key vectors: $\boldsymbol{k} = W_k\,\boldsymbol{x}$ $(N \times D_k)$

Value vectors: $\mathbf{v} = W_v\,\boldsymbol{x}$ $(N \times D)$

Alignments: $e_{i,j} = \boldsymbol{k}_i \cdot \boldsymbol{q}_j / \sqrt{D}$

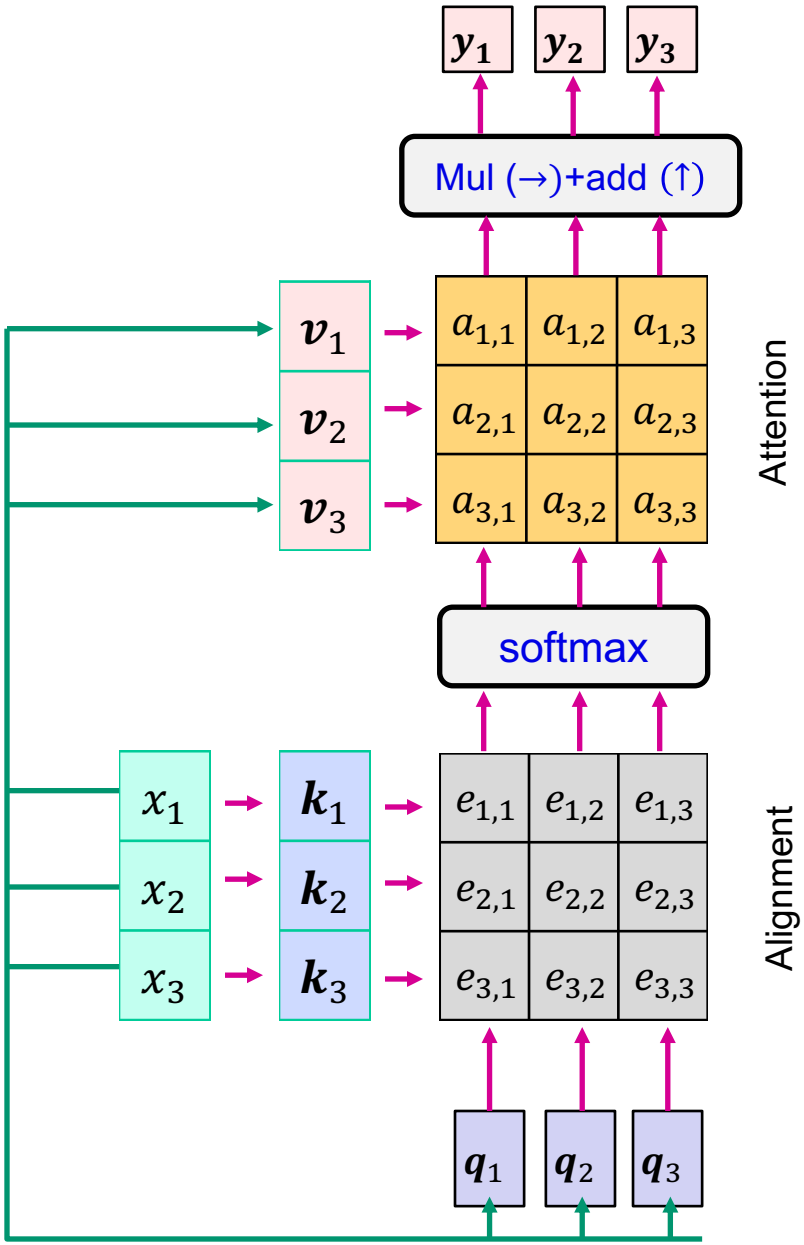
Attention: $a_{*,j} = \mathrm{softmax}(e_{*,j})$

Inputs:

Input vectors: $\boldsymbol{x}\ (shape: N \times D)$

Adapted from: J. Johnson

Self attention layer



Outputs:

Multiple Context vectors: $\boldsymbol{y}$ $(\mathrm{N}\times D)$

Operations:

Query vectors: $\boldsymbol{q} = W_q\,\boldsymbol{x}$ $(N\times D_k)$

Key vectors: $\boldsymbol{k} = W_k\,\boldsymbol{x}$ $(N\times D_k)$

Value vectors: $\mathbf{v} = W_v\,\boldsymbol{x}$ $(N\times D)$

Alignments: $e_{i,j} = \boldsymbol{k}_i \cdot \boldsymbol{q}_j/\sqrt{D}$

Attention: $a_{*,j} = \mathrm{softmax}(e_{*,j})$

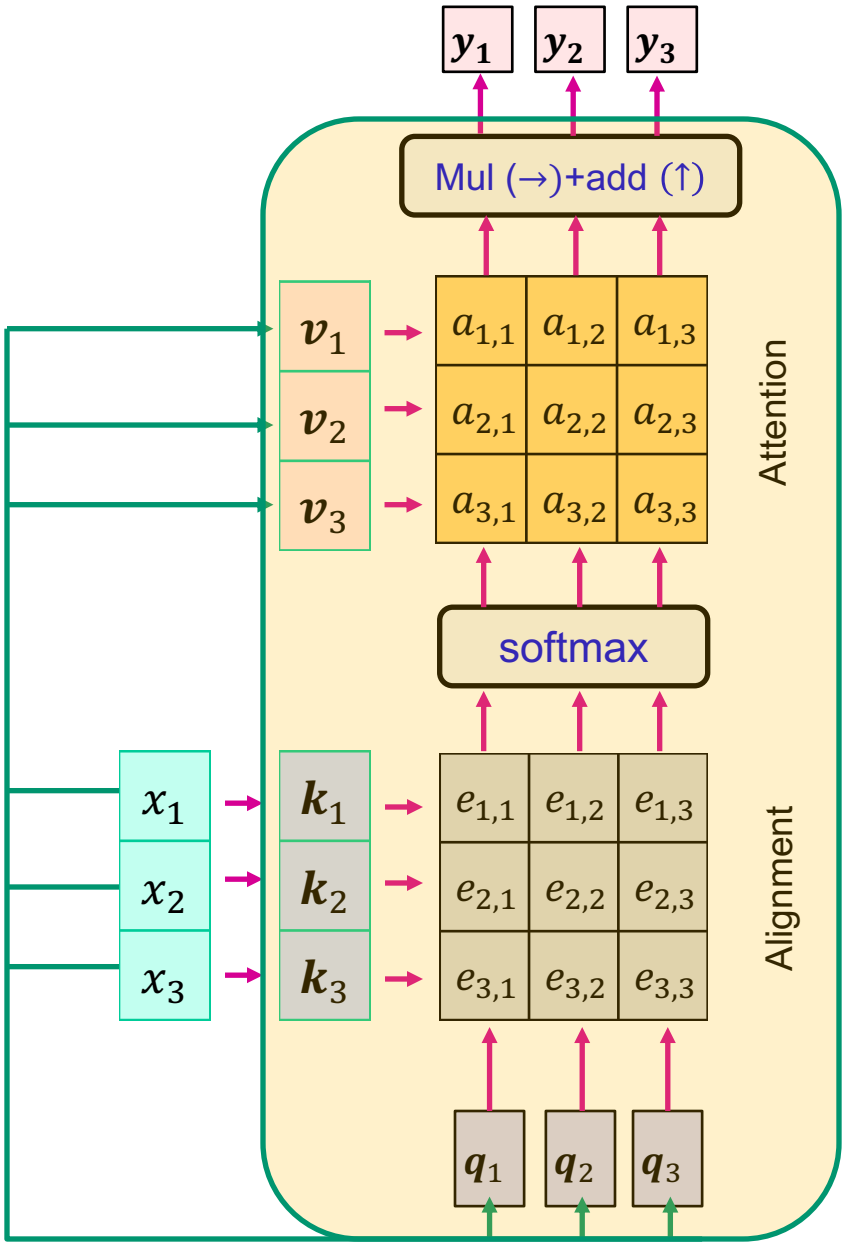
Output: $\boldsymbol{y}_j = \sum_i a_{i,j}\,\boldsymbol{v}_i$

Inputs:

Input vectors: $\boldsymbol{x}$ $(shape: N\times D)$

Adapted from: J. Johnson

Self attention layer

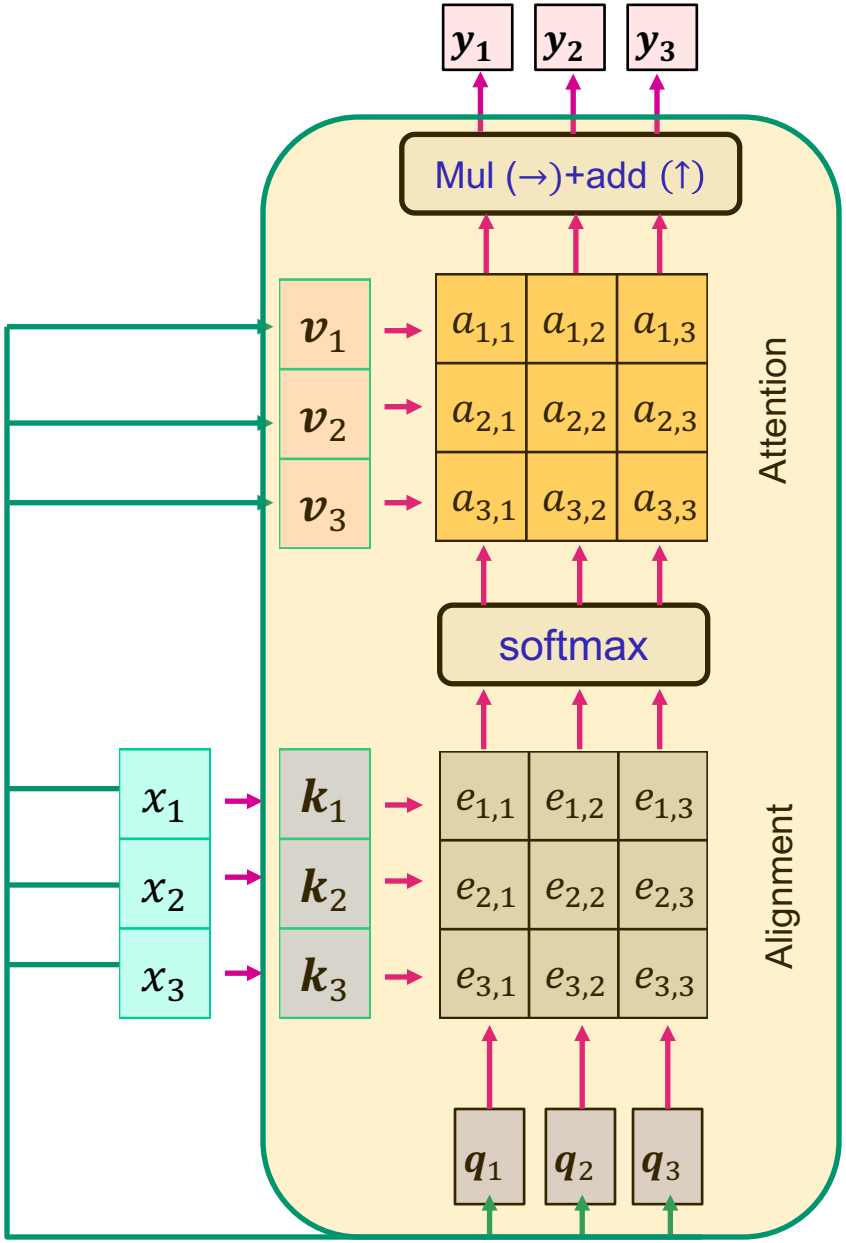


Input:
Input vectors: $N \times D$ inputs

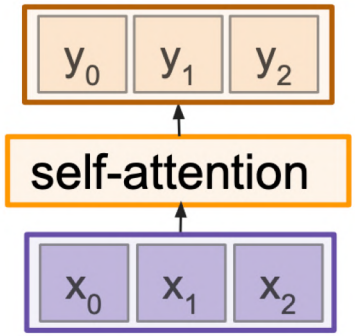
output:
output vectors: $N \times D$ inputs

Learned: W_q , W_k , W_v

Self attention layer



Self-attention layer:
is used to capture context *within the sequence.*

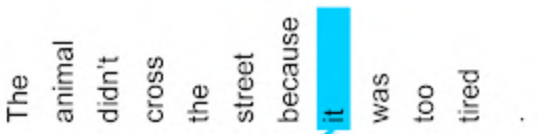


Input:
Input vectors: $N \times D$ inputs

output:
output vectors: $N \times D$ inputs

Self attention layer

- Used to capture context *within the sequence*



As we are encoding “it”, we should focus on “the animal”

As we are encoding “it”, we should focus on “the street”

Image source

General attention layer: Matrix form

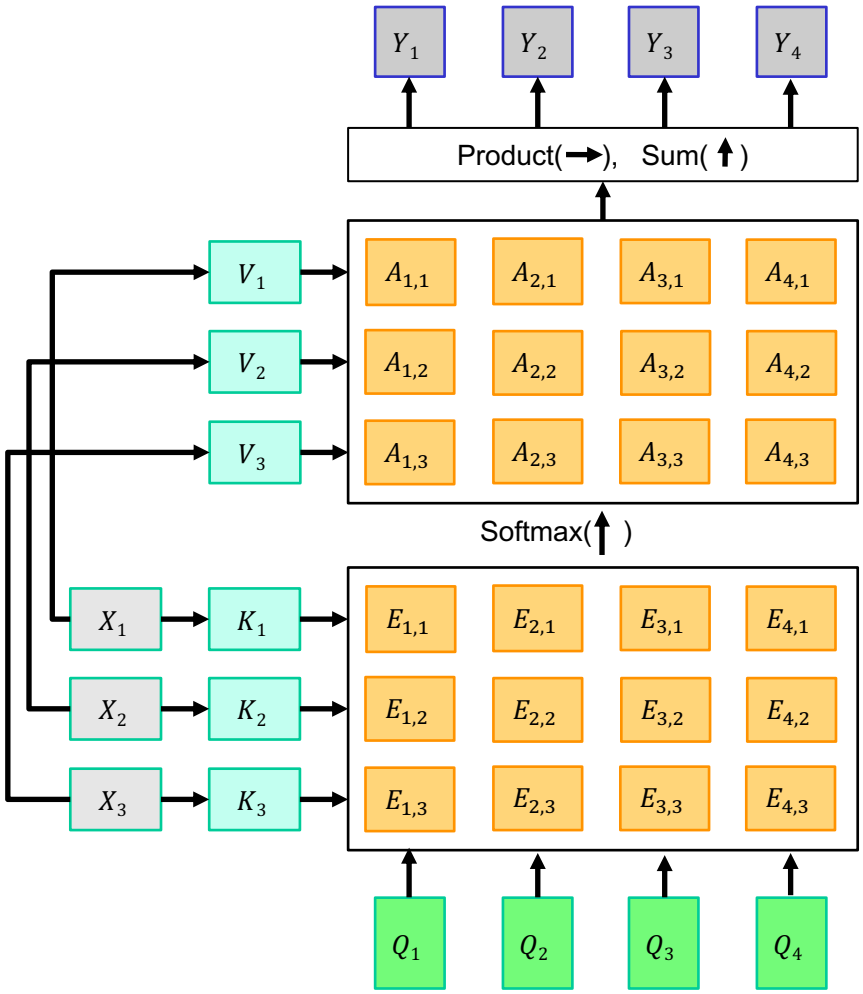
- Key vectors: $K = XW_K$ (NxD$_k$)
- Value Vectors: $V = XW_V$ (NxD$_v$)
- Query vectors (MxD$_k$)
- Similarities: *scaled dot-product attention*

$$E_{i,j} = \frac{(Q_i \cdot K_j)}{\sqrt{D}} \text{ or } E = QK^T/\sqrt{D}$$

(D is the dimensionality of the keys)

- Attn. weights: $A = \text{softmax}(E, \text{dim} = 1)$
- Output vectors:

$Y_i = \sum_j A_{i,j} V_j$ or $Y = AV$ (MxD$_v$)



Self attention layer: Matrix form

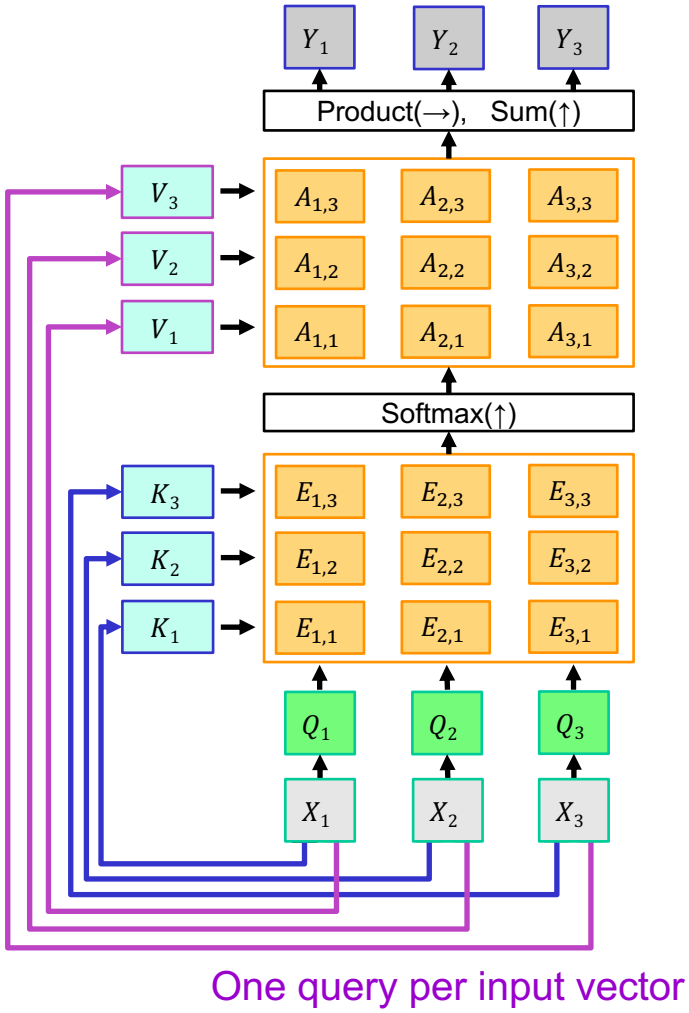
- Query vectors: $Q = XW_Q$
- Key vectors: $K = XW_K$
- Value vectors: $V = XW_V$
- Similarities: *scaled dot-product attention*

$$E_{i,j} = \frac{(Q_i \cdot K_j)}{\sqrt{D}} \text{ or } E = QK^T/\sqrt{D}$$

(D is the dimensionality of the keys)

- Attn. weights: $A = \text{softmax}(E, \text{dim} = 1)$
- Output vectors:

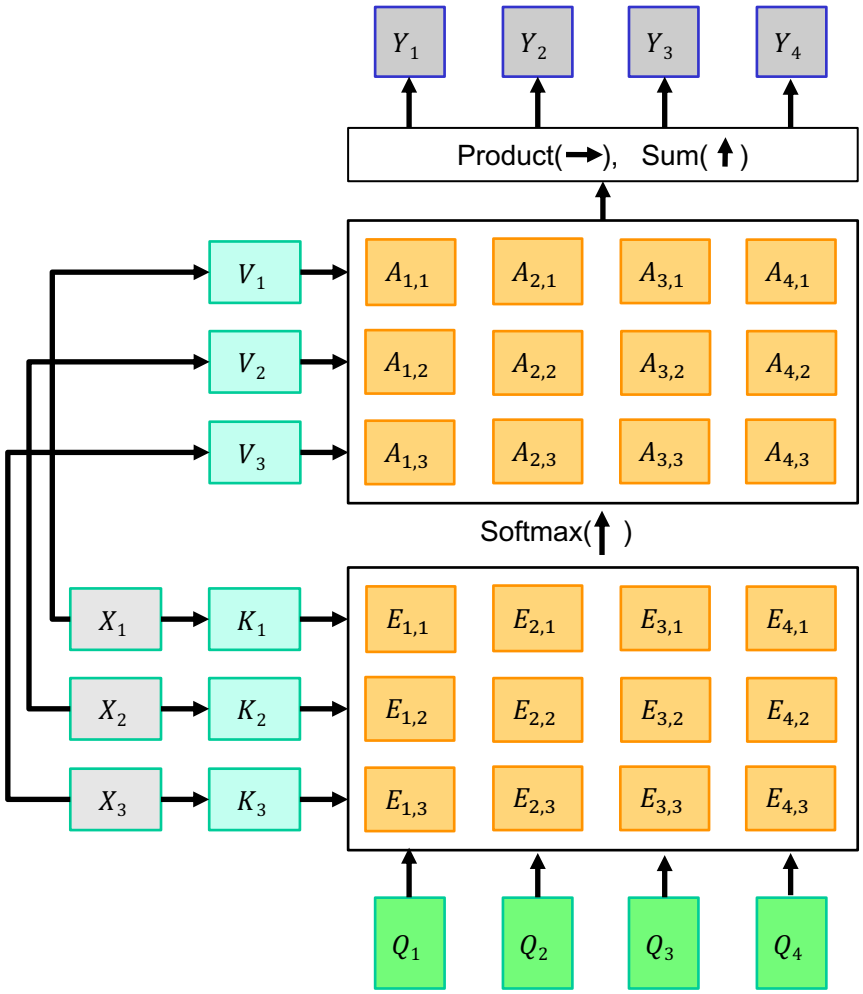
$$Y_i = \sum_j A_{i,j} V_j \text{ or } Y = AV \quad \text{(NxD)}$$



One query per input vector

Attention layer:

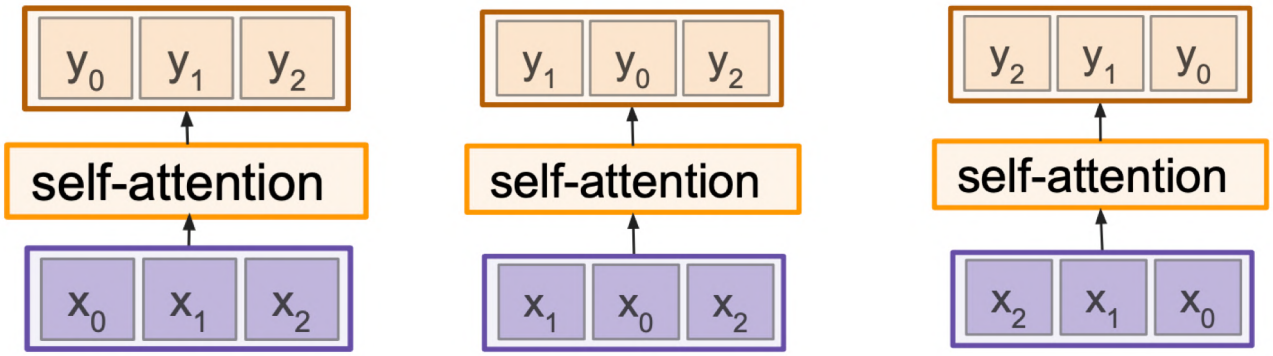
- How does permuting the order of the *queries* change the output?
- How does changing the order of the *keys/values* change the output?



Attention layer: Positional Encoding

Permutation invariant:

- The self attention layer attends over a set of inputs
- Attention operation is **permutation invariant:** it doesn't care about ordering of the inputs
- **Problem:** how can we encode ordered sequences like language or spatially ordered image features?



Adapted from: J. Johnson

Attention layer: Positional Encoding

- ***Positional encoding*** provide the network information about the position of each input.
- Concatenate *positional encoding* $\boldsymbol{p}_j$ to each input vector $\boldsymbol{x}_j$
- We use a function ***pos***: N $\rightarrow$ R^d to encode the position j of the vector $\boldsymbol{x}_j$ into a d-dimensional vector, so $\boldsymbol{p}_j = pos(j)$
- Requirements for pos(j):
  - It should output a **unique** encoding for each position
  - **Distance** between any two time-steps should be consistent across sequences with different lengths.
  - Should generalize to **longer** sentences without any efforts.

Adapted from: J. Johnson

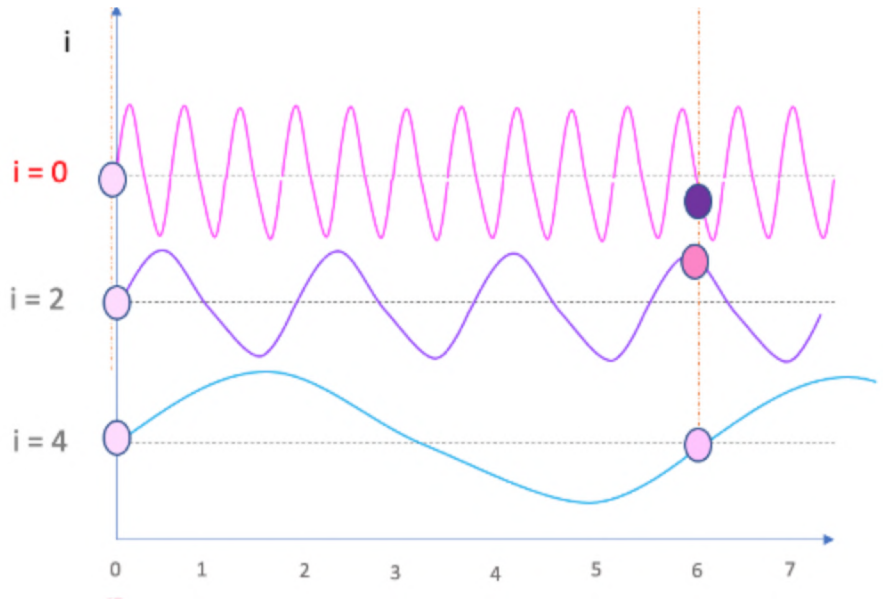
Attention layer: Positional Encoding

- Common use for pos(j): sin+cos waves:

$$\begin{bmatrix} \sin(\omega_1 \cdot j) \\ \cos(\omega_1 \cdot j) \\ \sin(\omega_2 \cdot j) \\ \cos(\omega_2 \cdot j) \\ \vdots \\ \sin(\omega_{d/2} \cdot j) \\ \cos(\omega_{d/2} \cdot j) \end{bmatrix} \quad \text{where} \quad \omega_k = \frac{1}{10000^{2k/d}}$$

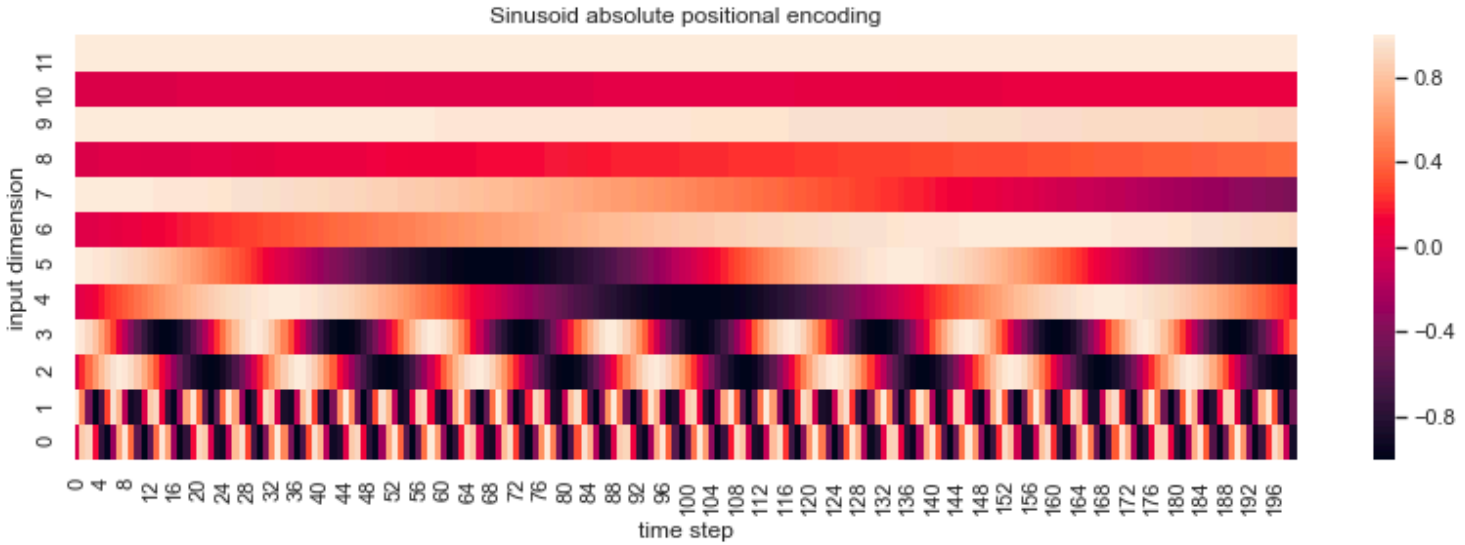
Adapted from: J. Johnson

Attention layer: Positional Encoding



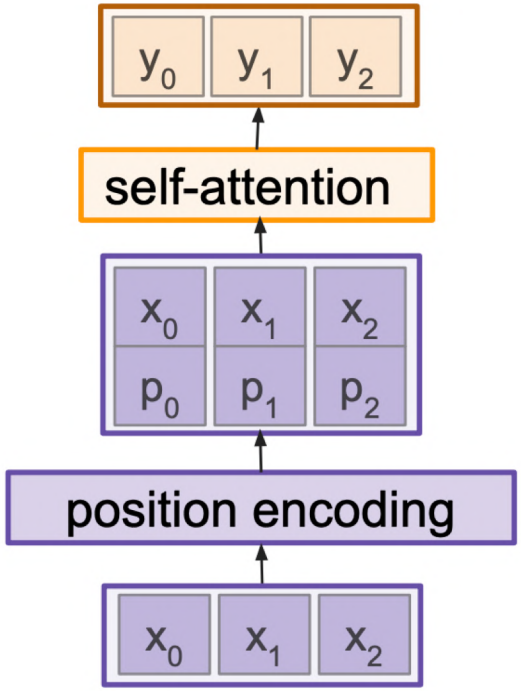
Intuition:

0 :	0	0	0	0	8 :	1	0	0	0
1 :	0	0	0	1	9 :	1	0	0	1
2 :	0	0	1	0	10 :	1	0	1	0
3 :	0	0	1	1	11 :	1	0	1	1
4 :	0	1	0	0	12 :	1	1	0	0
5 :	0	1	0	1	13 :	1	1	0	1
6 :	0	1	1	0	14 :	1	1	1	0
7 :	0	1	1	1	15 :	1	1	1	1



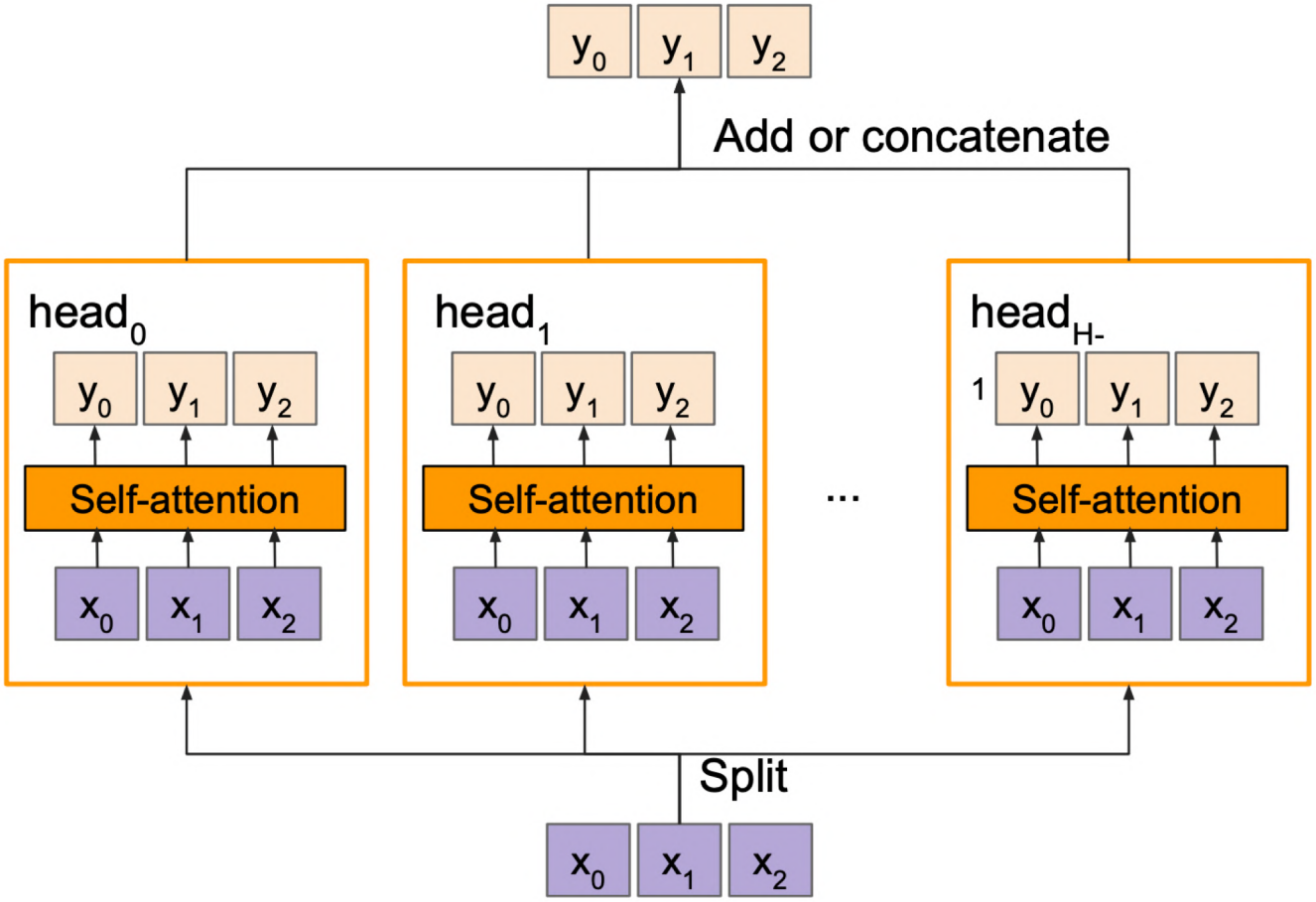
Attention layer: Positional Encoding

Self attention layer with positional encoding:

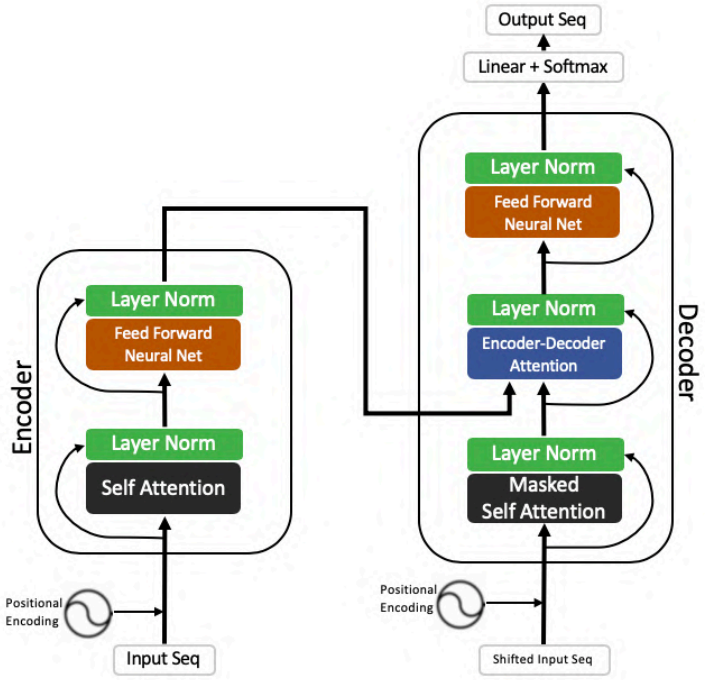


Multi-head self attention layer

Multi-head self attention layer:



Transformers

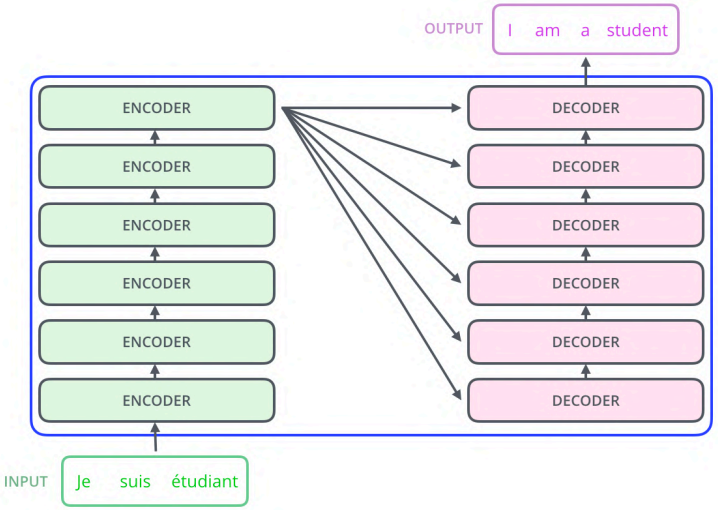


Basic transformer model (review)

- Sequence-to-sequence architecture using only point-wise processing and attention (no recurrent units or convolutions)
- More efficient and parallelizable than recurrent or convolutional architectures, faster to train, better accuracy.

Encoder: receives entire input sequence and outputs encoded sequence of the same length

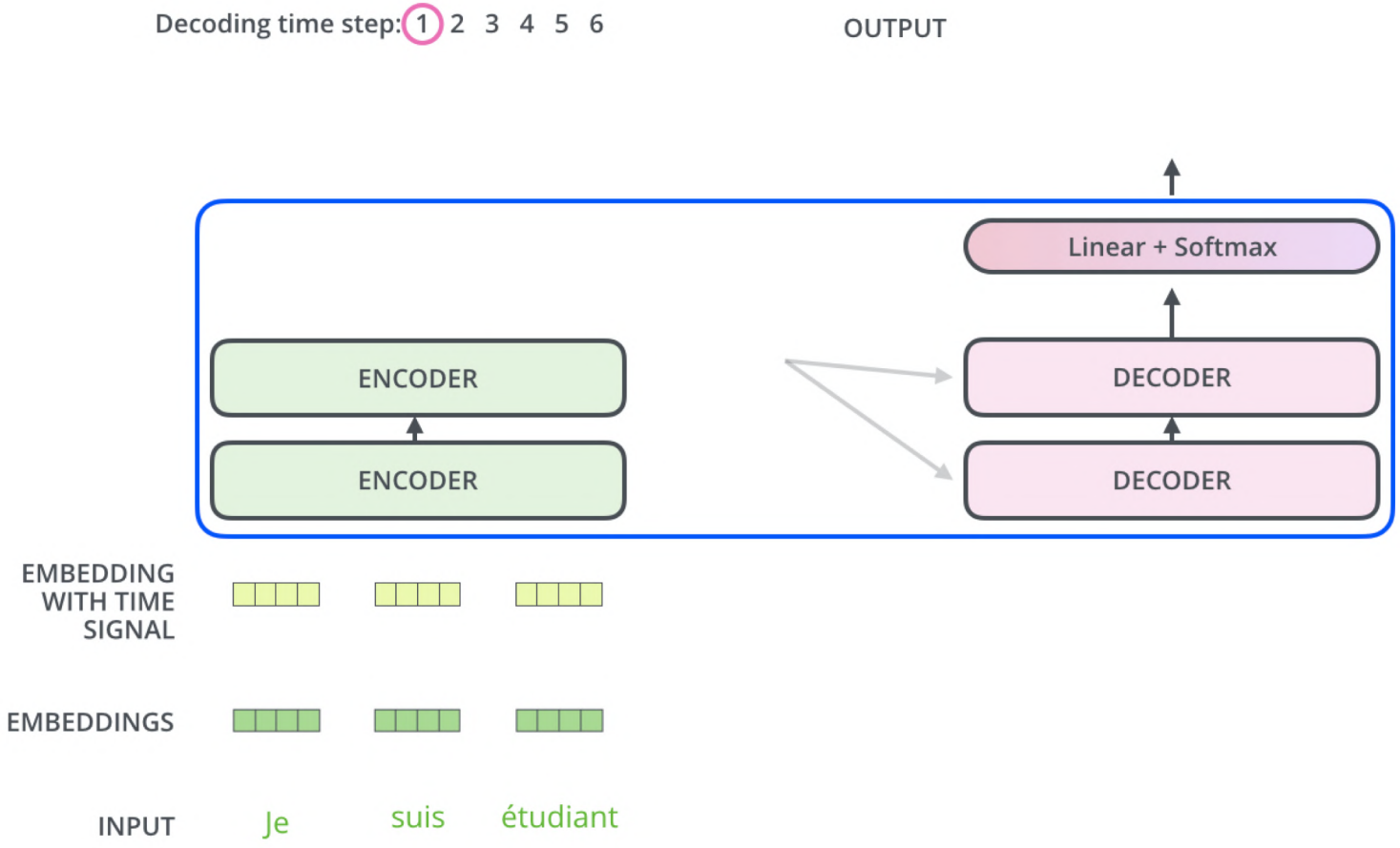
Decoder: predicts next token conditioned on encoder output and previously predicted tokens



A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, NeurIPS 2017

Basic transformer model (review)

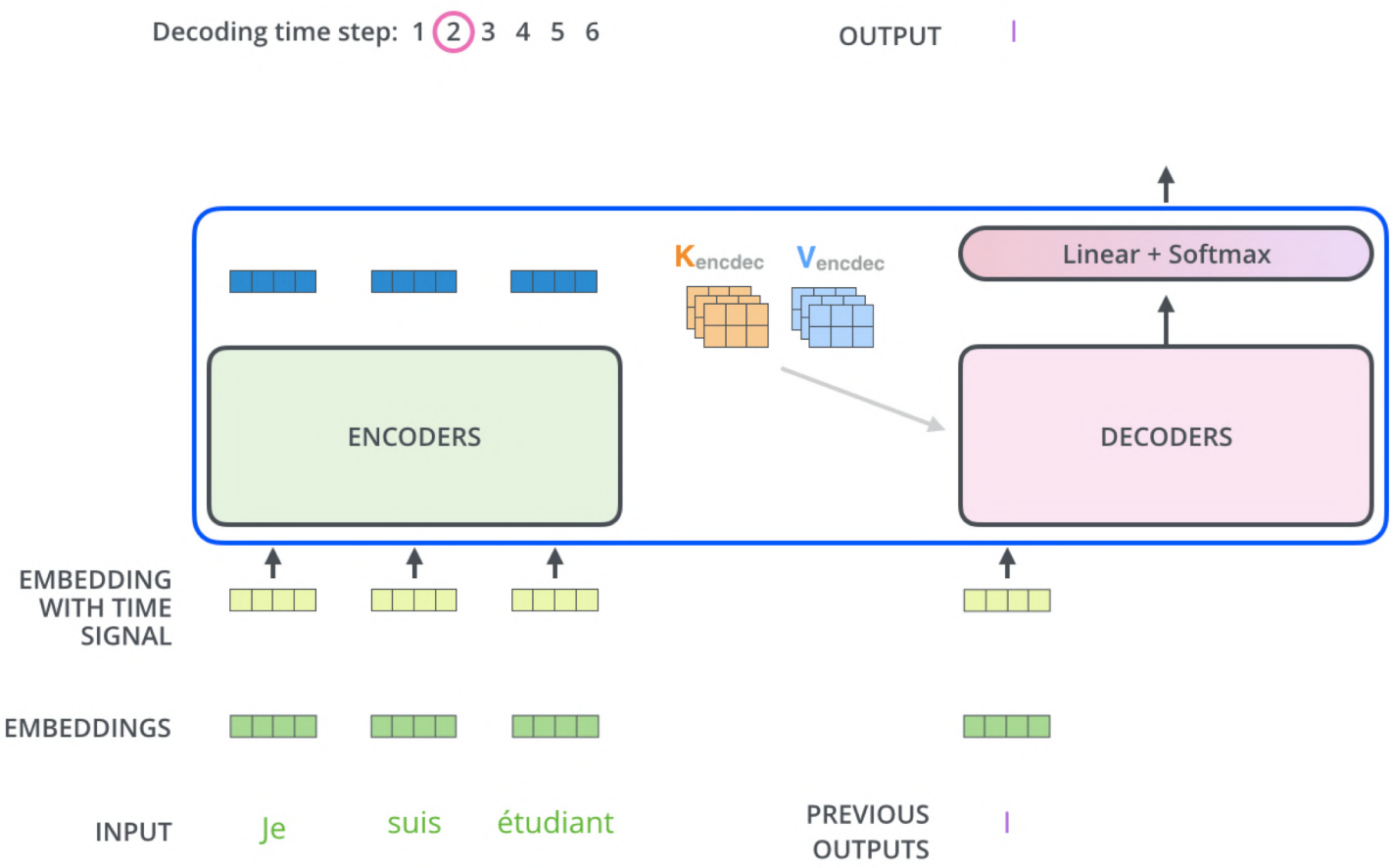
Illustrated transformer - 1st step:



From: [The Illustrated Transformer](): Jay Alammar

Basic transformer model (review)

Illustrated transformer – next steps:



From: [The Illustrated Transformer](): Jay Alammar

Image captioning using transformers

Input: image I

Output: $y = y_1, y_2, y_3 \ldots, y_T$

CNN

$z_{1,1}$	$z_{1,2}$	$z_{1,3}$
$z_{2,1}$	$z_{2,2}$	$z_{2,3}$
$z_{3,1}$	$z_{3,2}$	$z_{3,3}$

Spatial features from a pretrained CNN

Features H x W x D

Image captioning using transformers

Input: image I

Output: $y = y_1, y_2, y_3 \ldots, y_T$

Encoder: $\boldsymbol{c} = T_w(\boldsymbol{z})$

where $\boldsymbol{z}$ is spatial CNN features and $T_w(\boldsymbol{z})$ is the transformer encoder

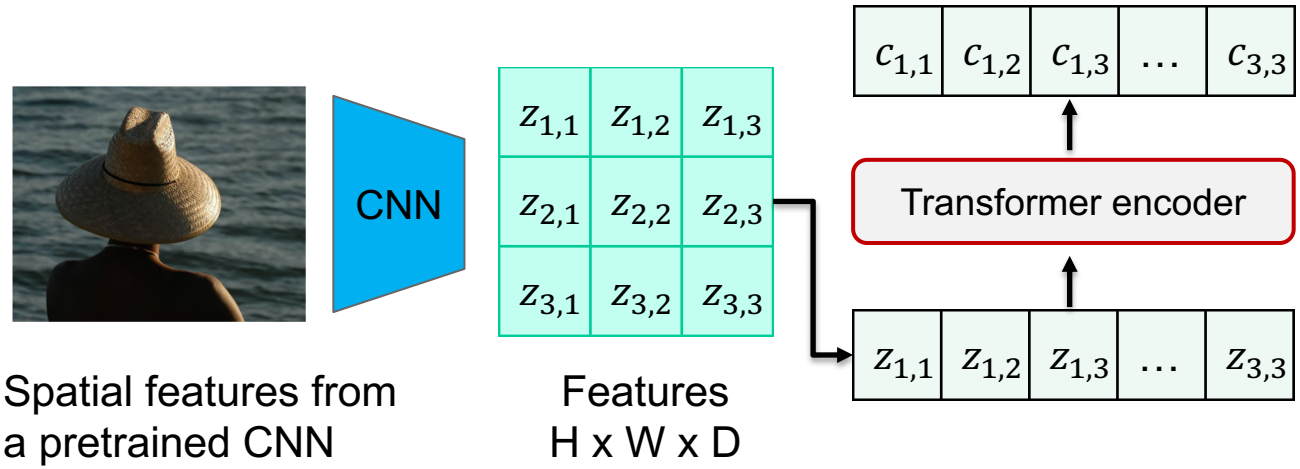


Image captioning using transformers

Input: image I

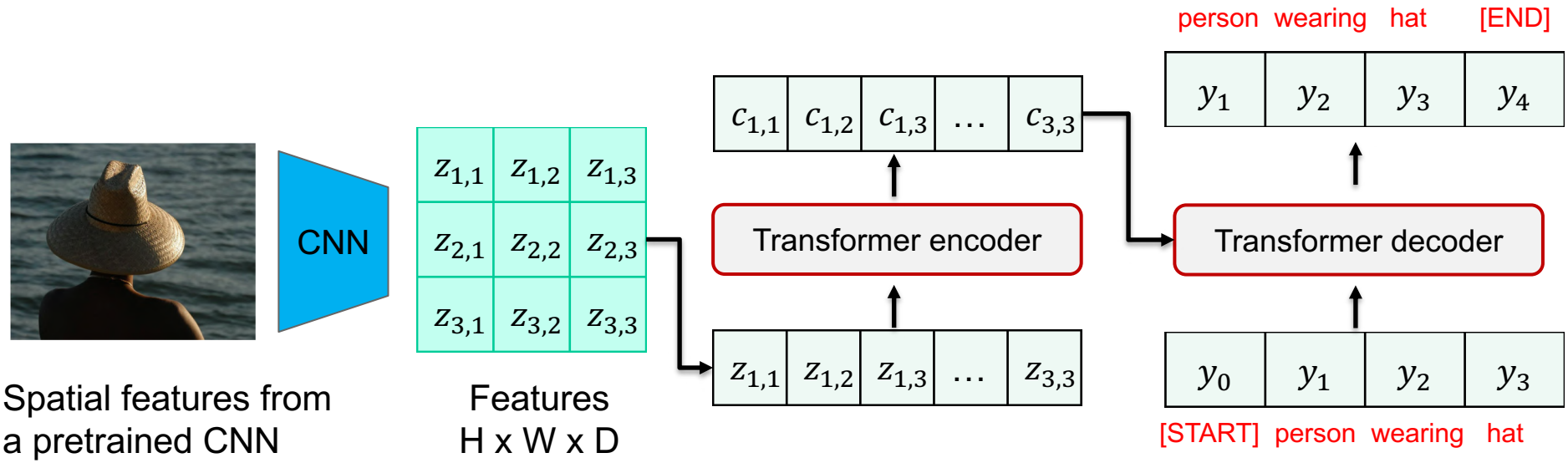
Output: $y = y_1, y_2, y_3 \ldots, y_T$

Encoder: $\boldsymbol{c} = T_w(\boldsymbol{z})$

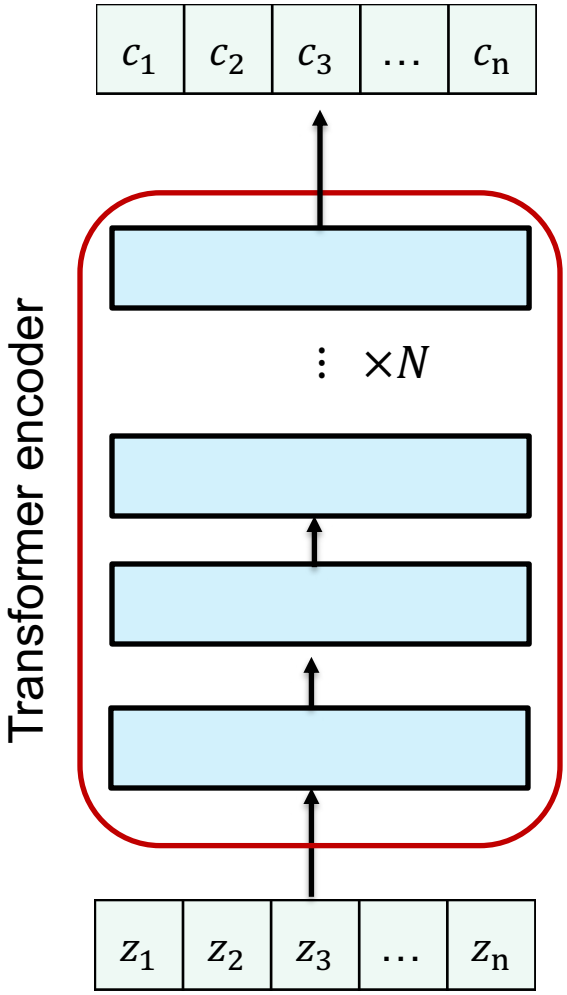
where $\boldsymbol{z}$ is spatial CNN features and $T_w(\boldsymbol{z})$ is the transformer encoder

Decoder: $\boldsymbol{y}_t = T_D(\boldsymbol{y}_{0:t-1}, \boldsymbol{c})$

where $T_D(.)$ is the decoder transformer



The Transformer encoder block

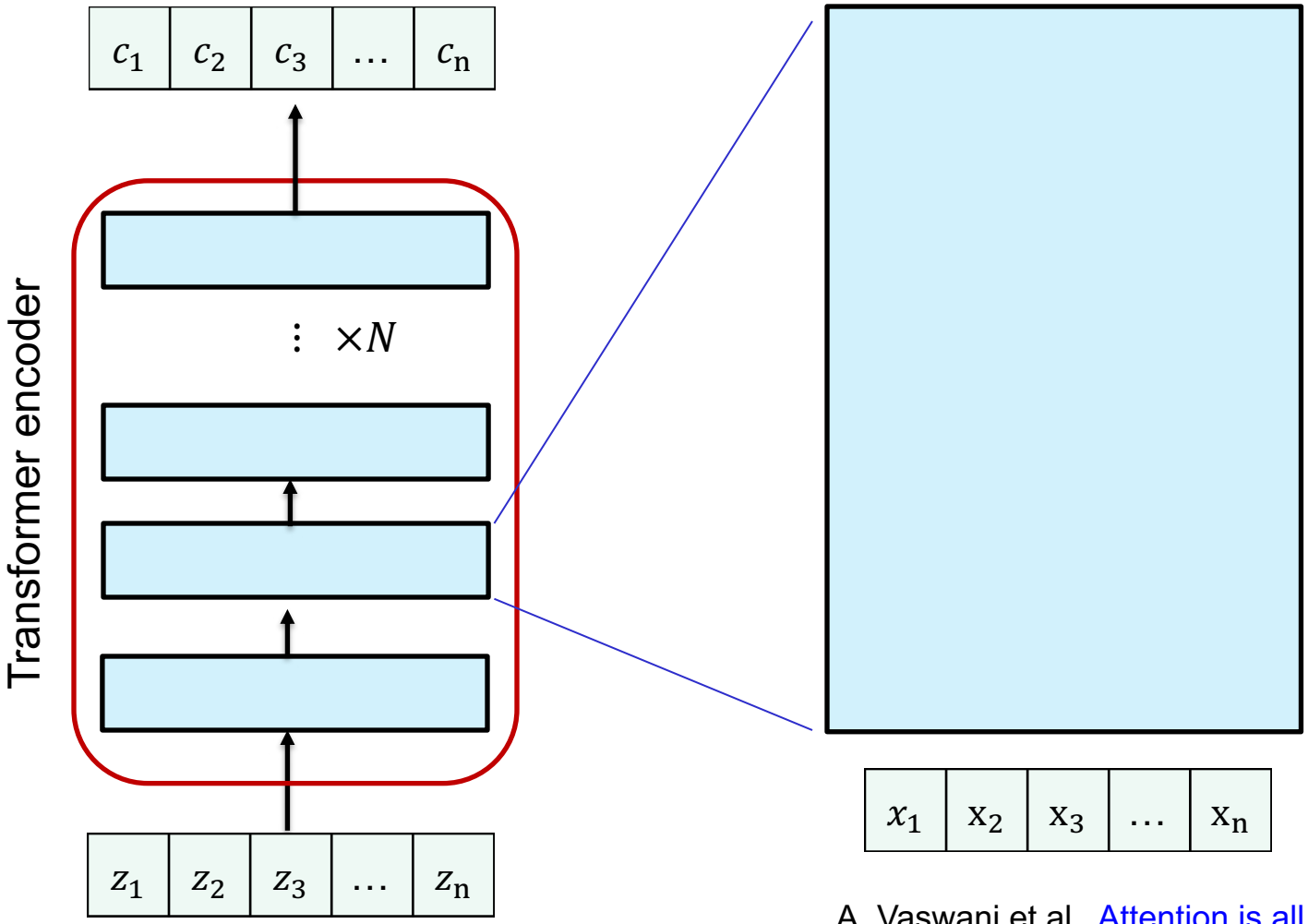


Made up of N encoder blocks

In Vaswani et al. N=12 , D_q=512

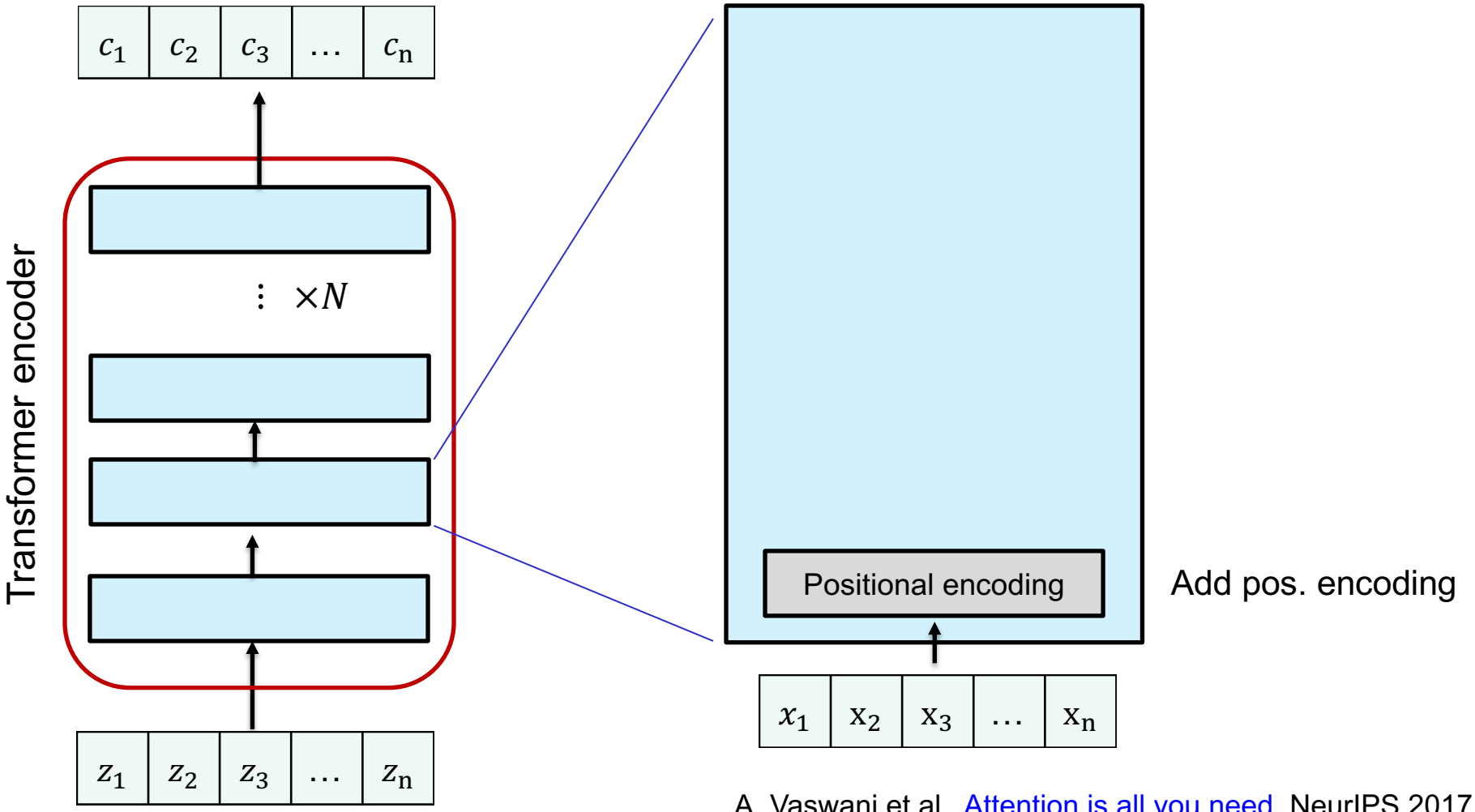
A. Vaswani et al., Attention is all you need, NeurIPS 2017

The Transformer encoder block



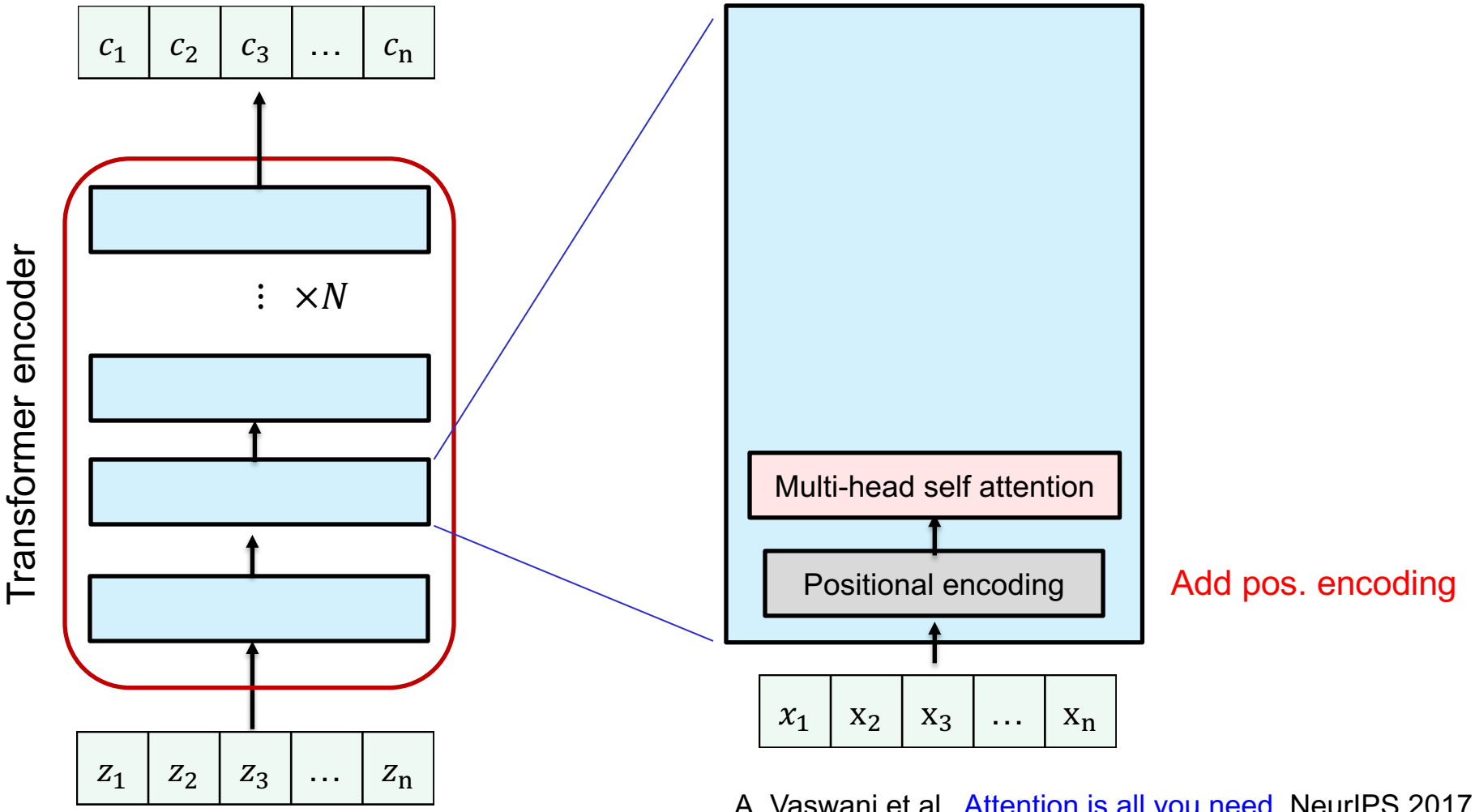
A. Vaswani et al., Attention is all you need, NeurIPS 2017

The Transformer encoder block



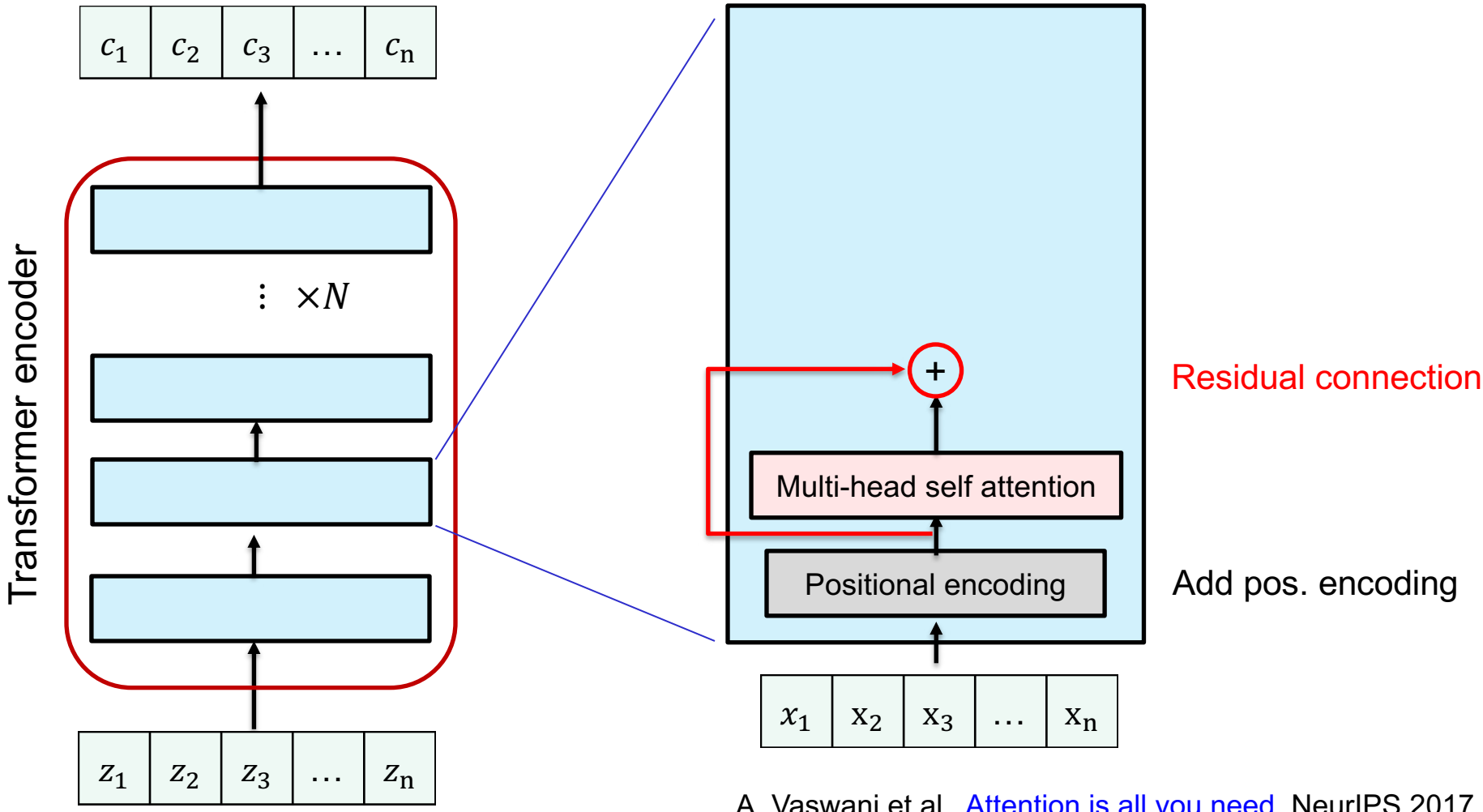
A. Vaswani et al., Attention is all you need, NeurIPS 2017

The Transformer encoder block



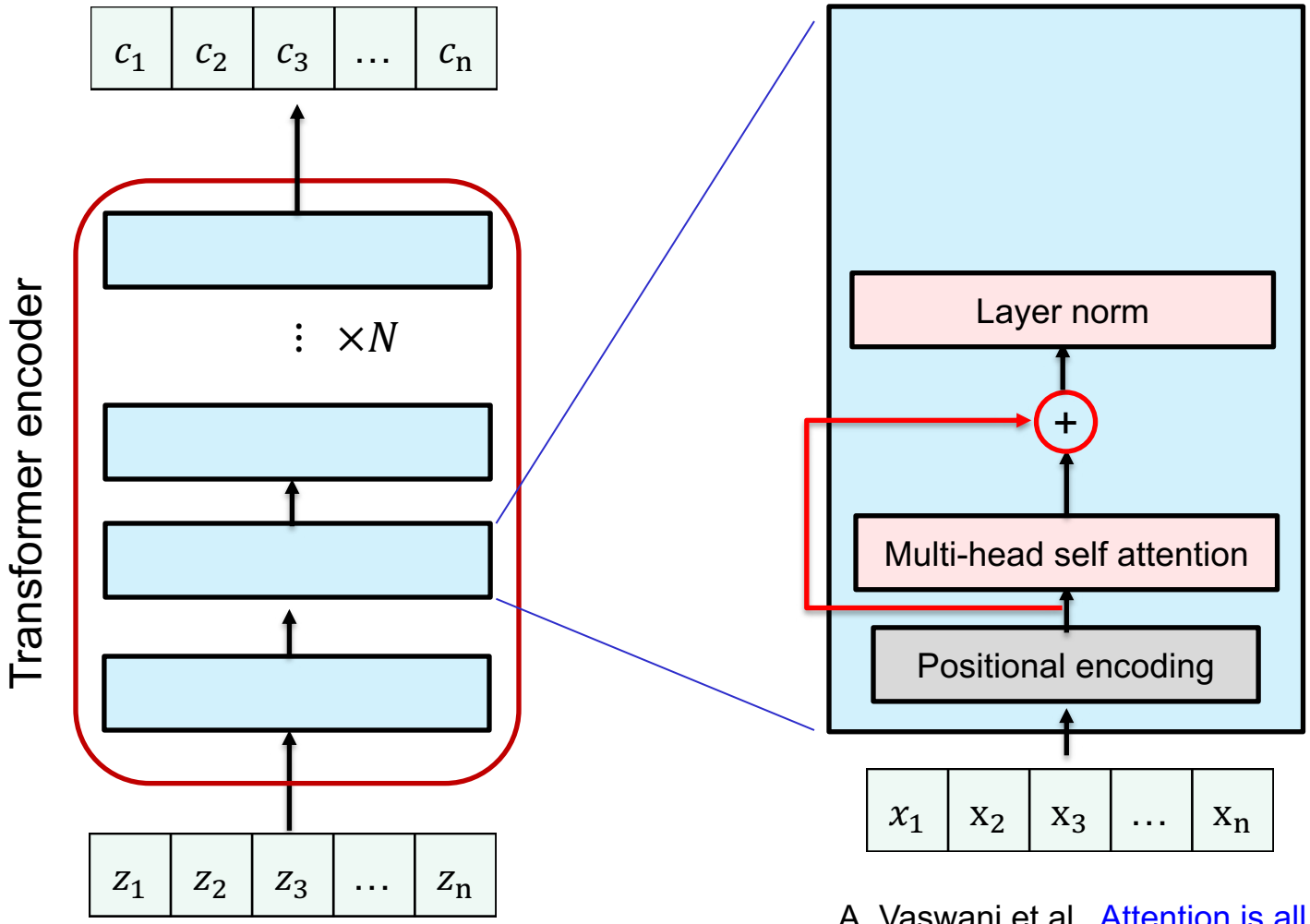
A. Vaswani et al., Attention is all you need, NeurIPS 2017

The Transformer encoder block



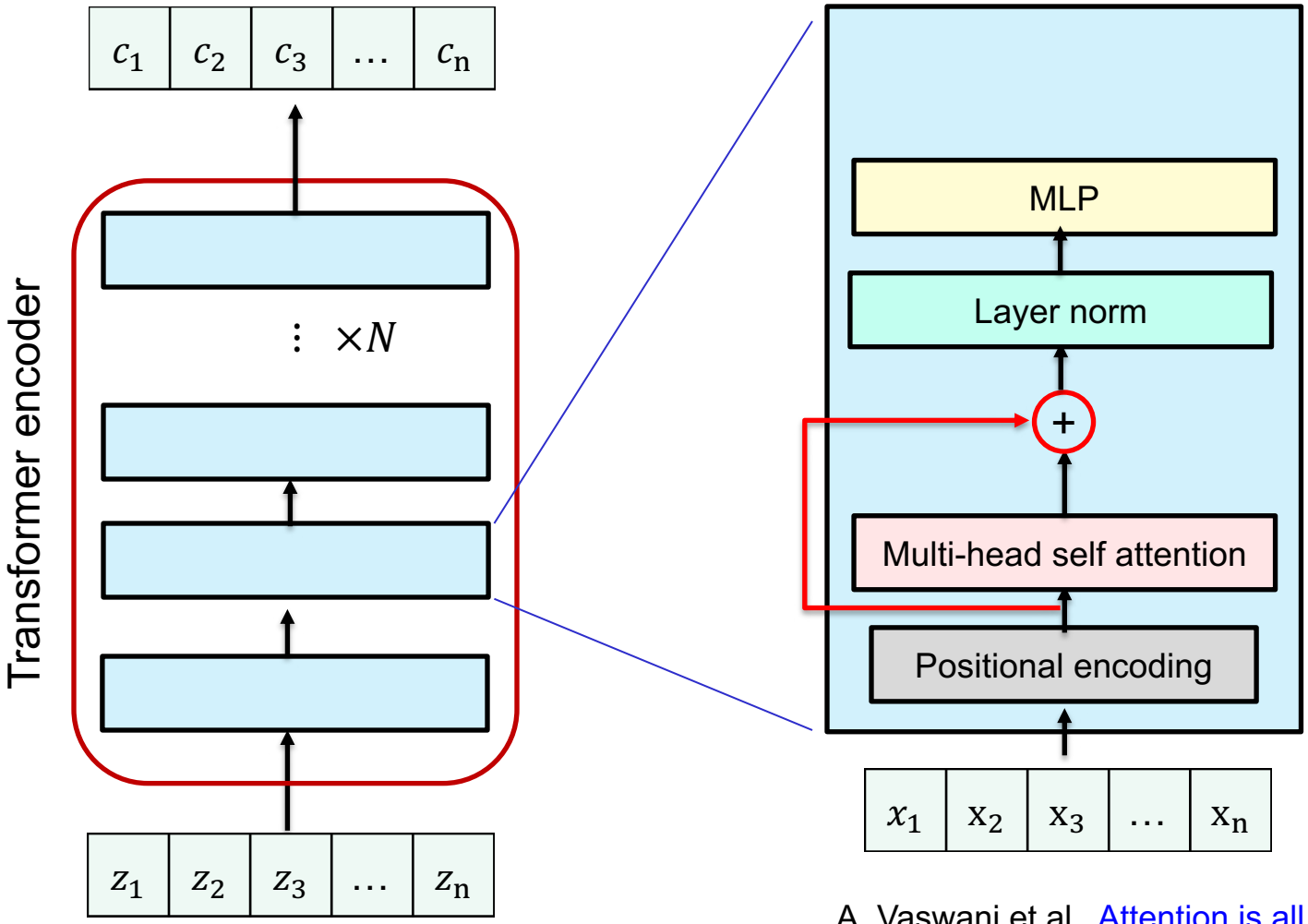
A. Vaswani et al., Attention is all you need, NeurIPS 2017

The Transformer encoder block



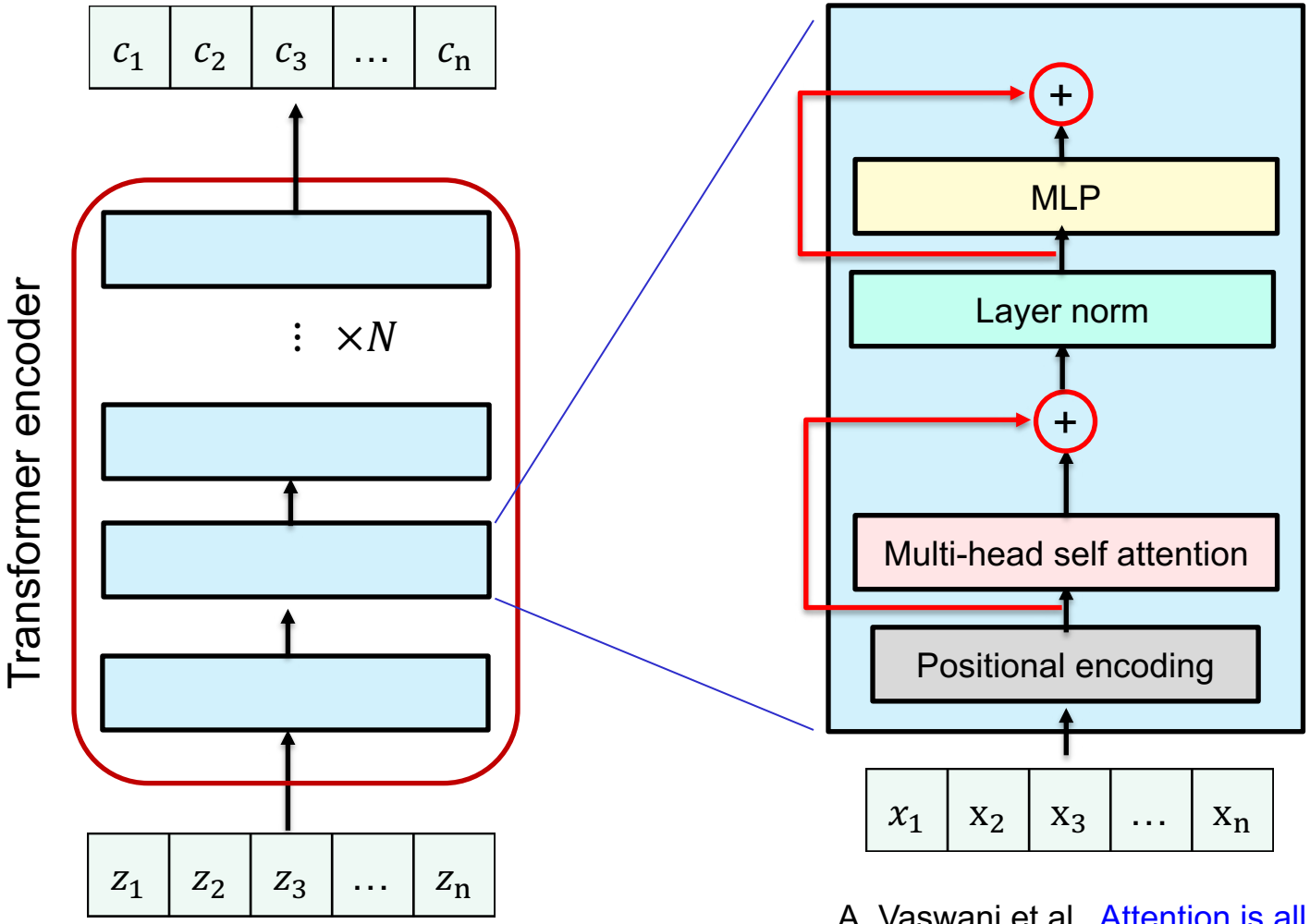
A. Vaswani et al., Attention is all you need, NeurIPS 2017

The Transformer encoder block



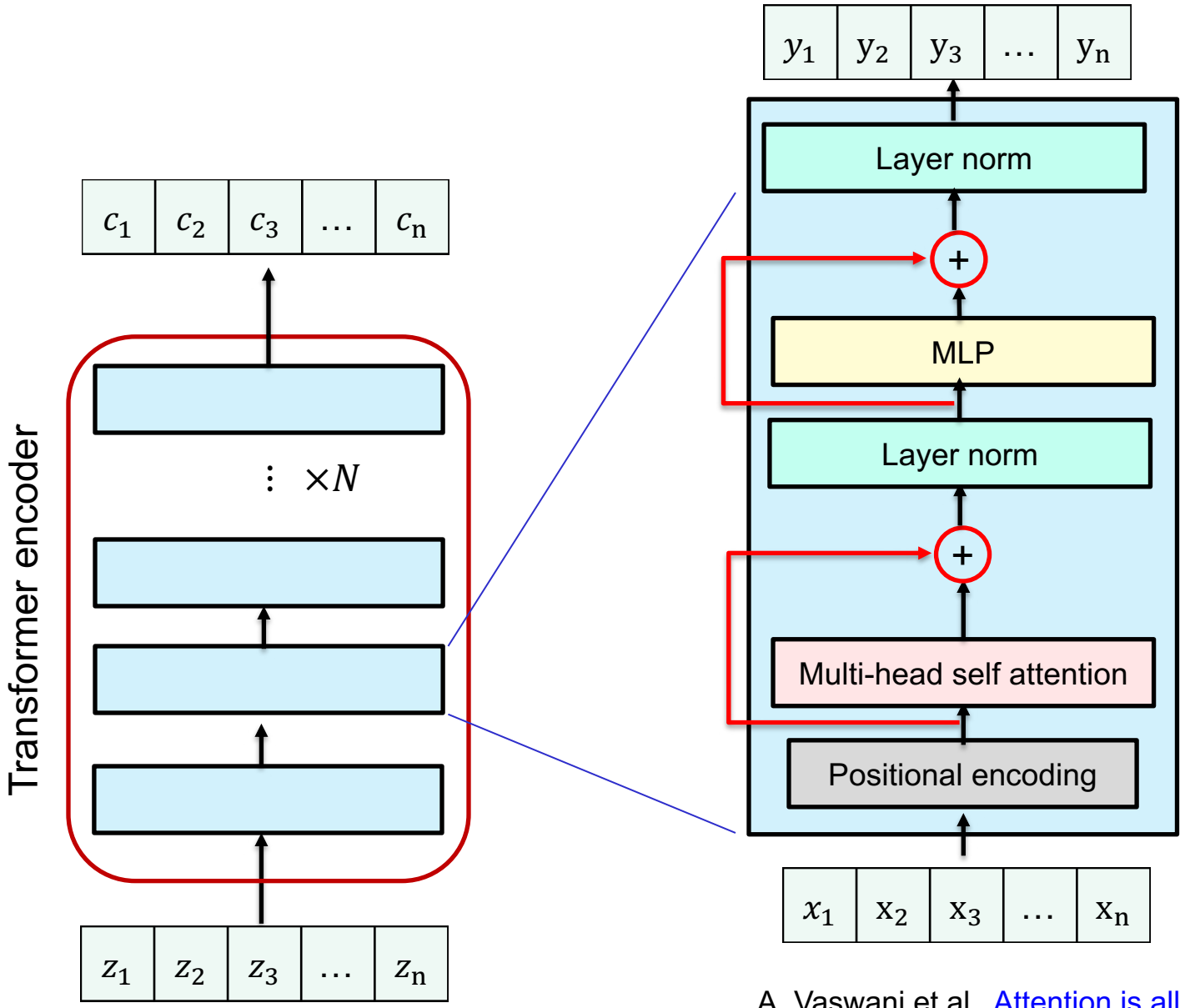
A. Vaswani et al., Attention is all you need, NeurIPS 2017

The Transformer encoder block



A. Vaswani et al., Attention is all you need, NeurIPS 2017

The Transformer encoder block



Encoder Block:

Inputs: Set of vectors **x**

Outputs: Set of vectors **y**

Self-attention is the only interaction between vectors.

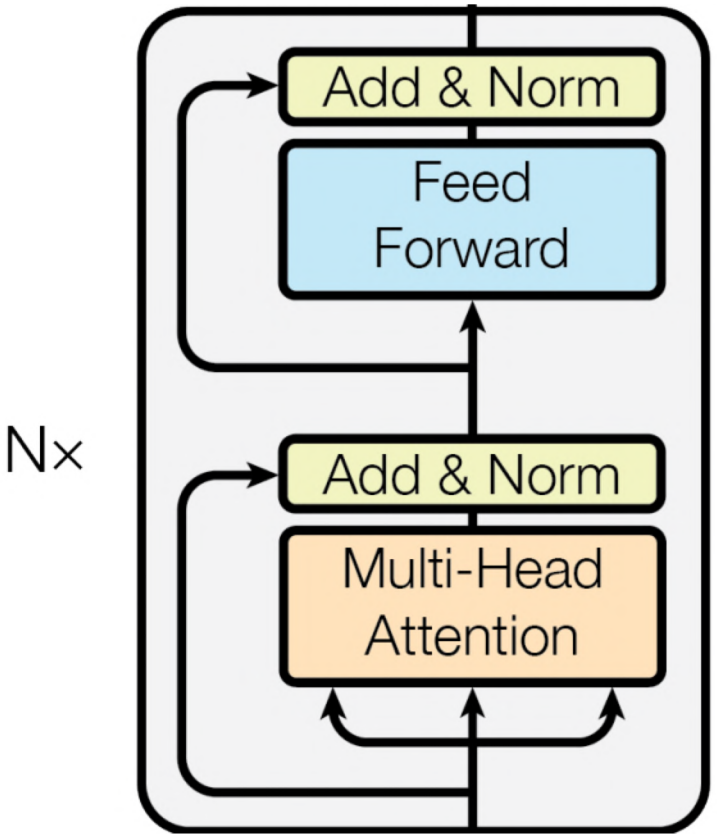
Layer norm and MLP operate independently per vector.

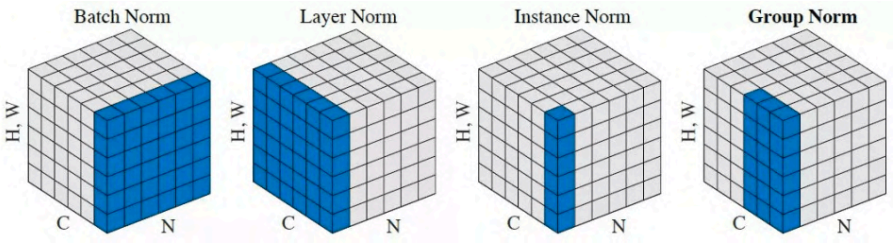
Highly scalable, highly parallelizable, but high memory usage.

A. Vaswani et al., Attention is all you need, NeurIPS 2017

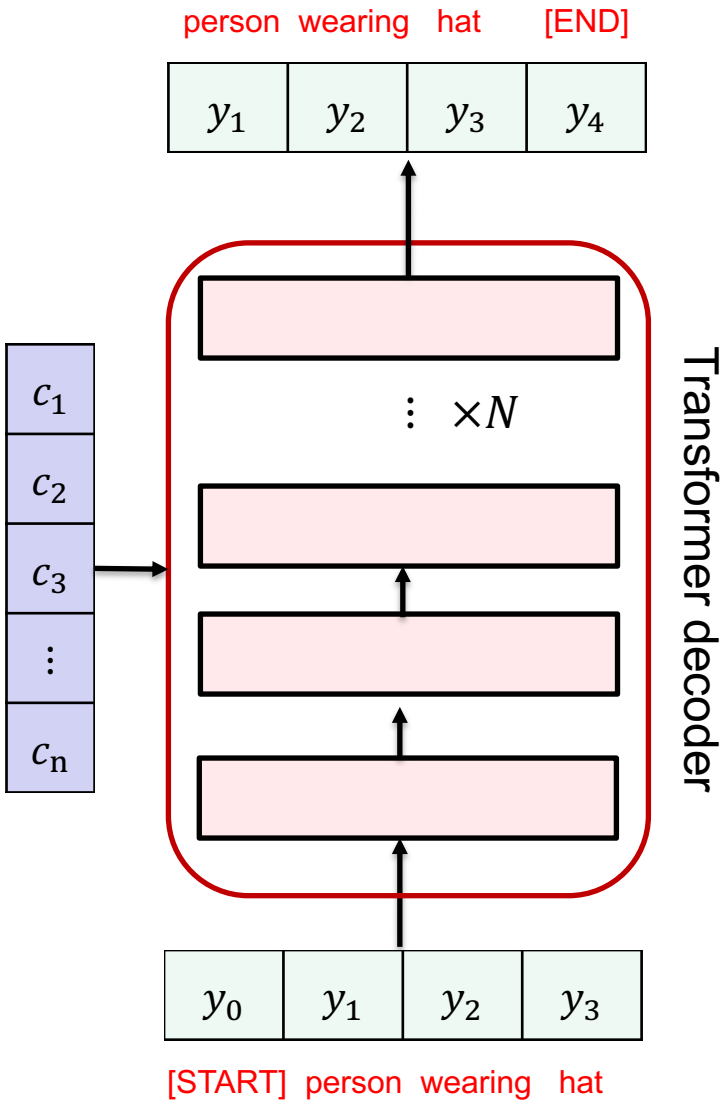
The Transformer encoder block

- A **Transformer** is a sequence of transformer blocks
  - Vaswani et al.: N=12 blocks, embedding dimension = 512, 6 attention heads
  - **Add & Norm:** residual connection followed by layer normalization
  - **Feedforward:** two linear layers with ReLUs in between, applied independently to each vector
- Attention is the only interaction between inputs!



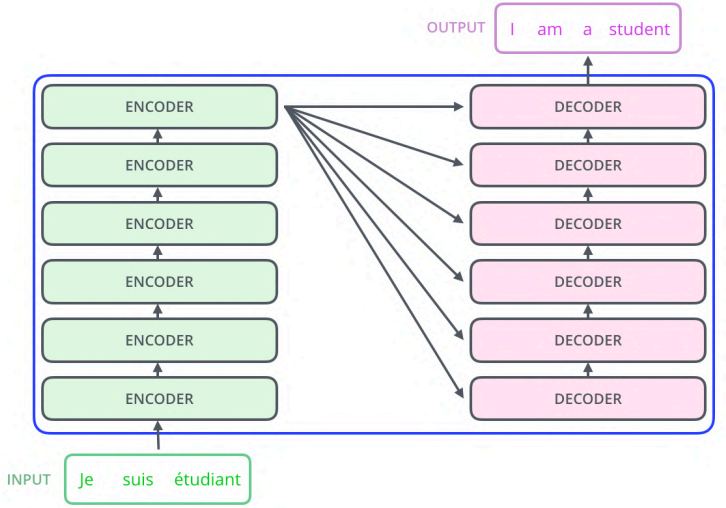


The Transformer decoder block

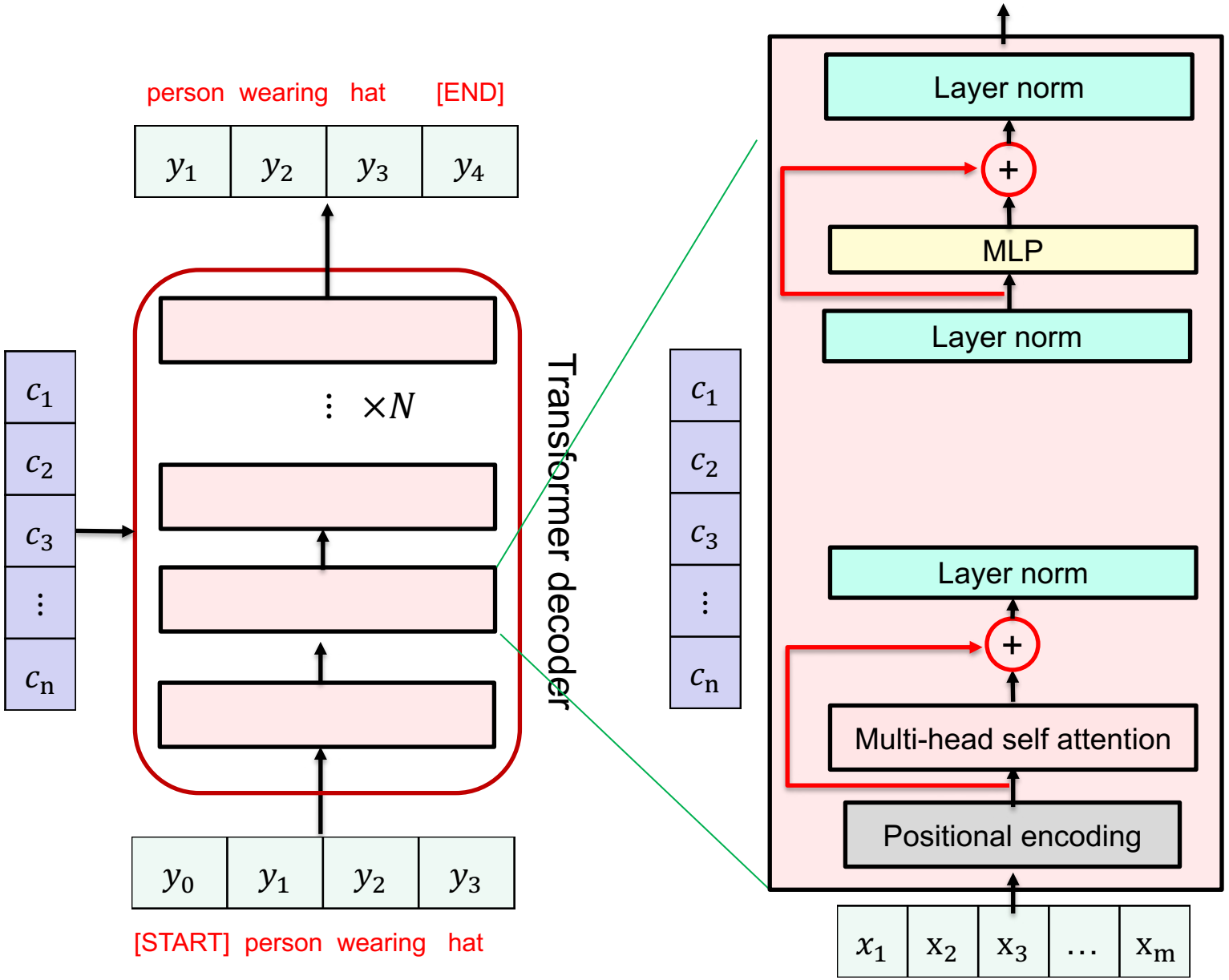


Made up of N decoder blocks

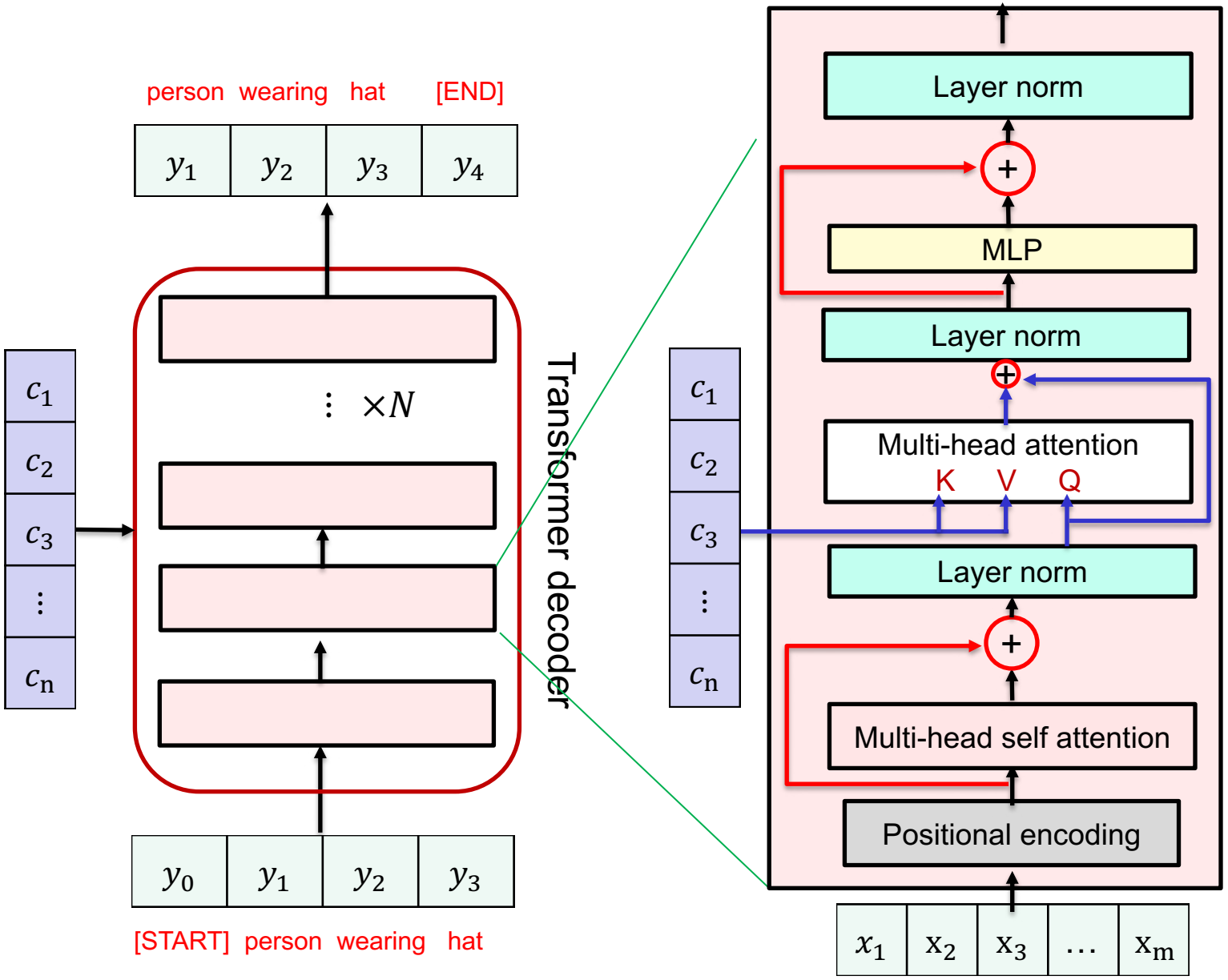
In Vaswani et al. N=6 , D_q=512



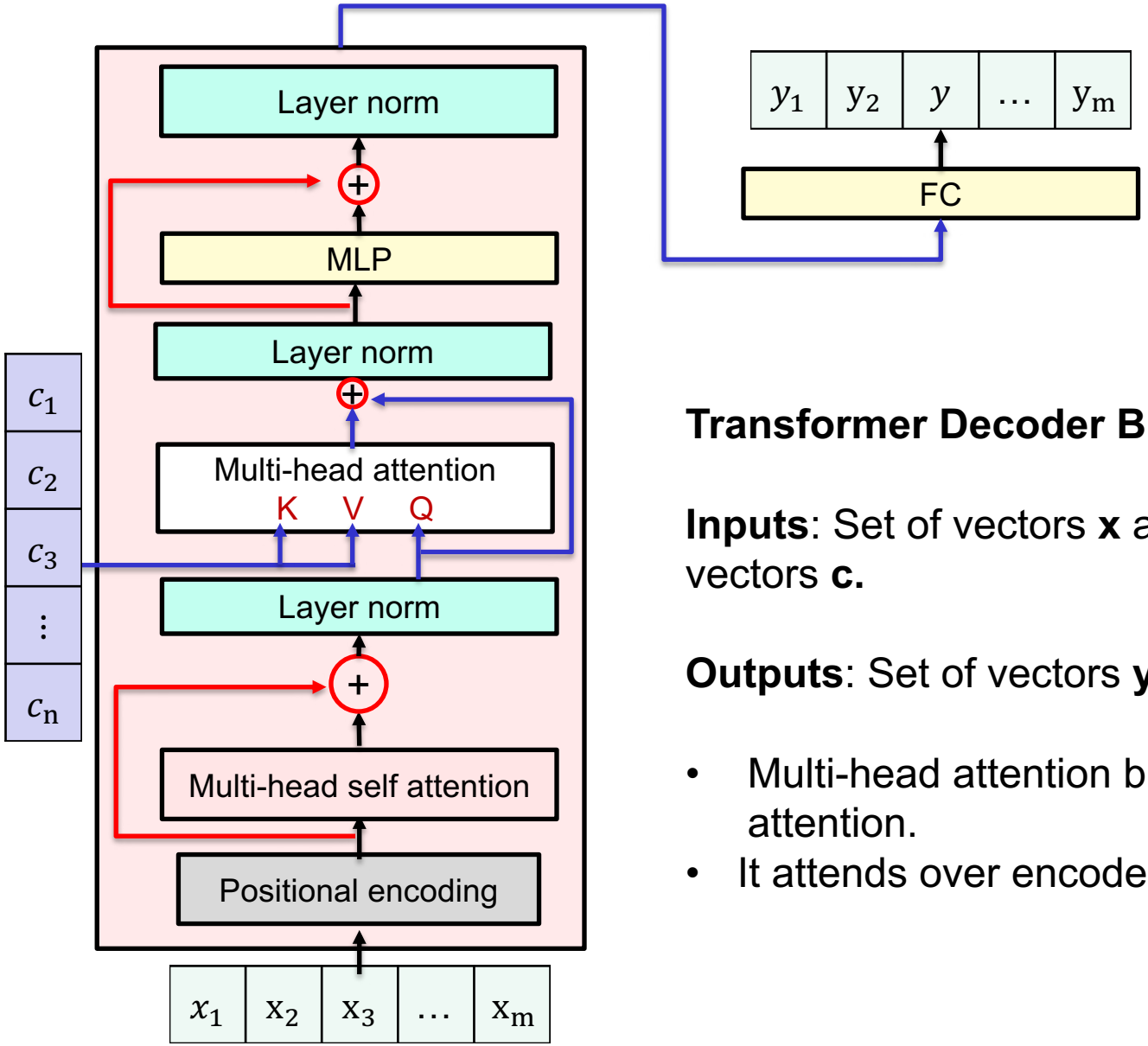
The Transformer decoder block



The Transformer decoder block



The Transformer decoder block



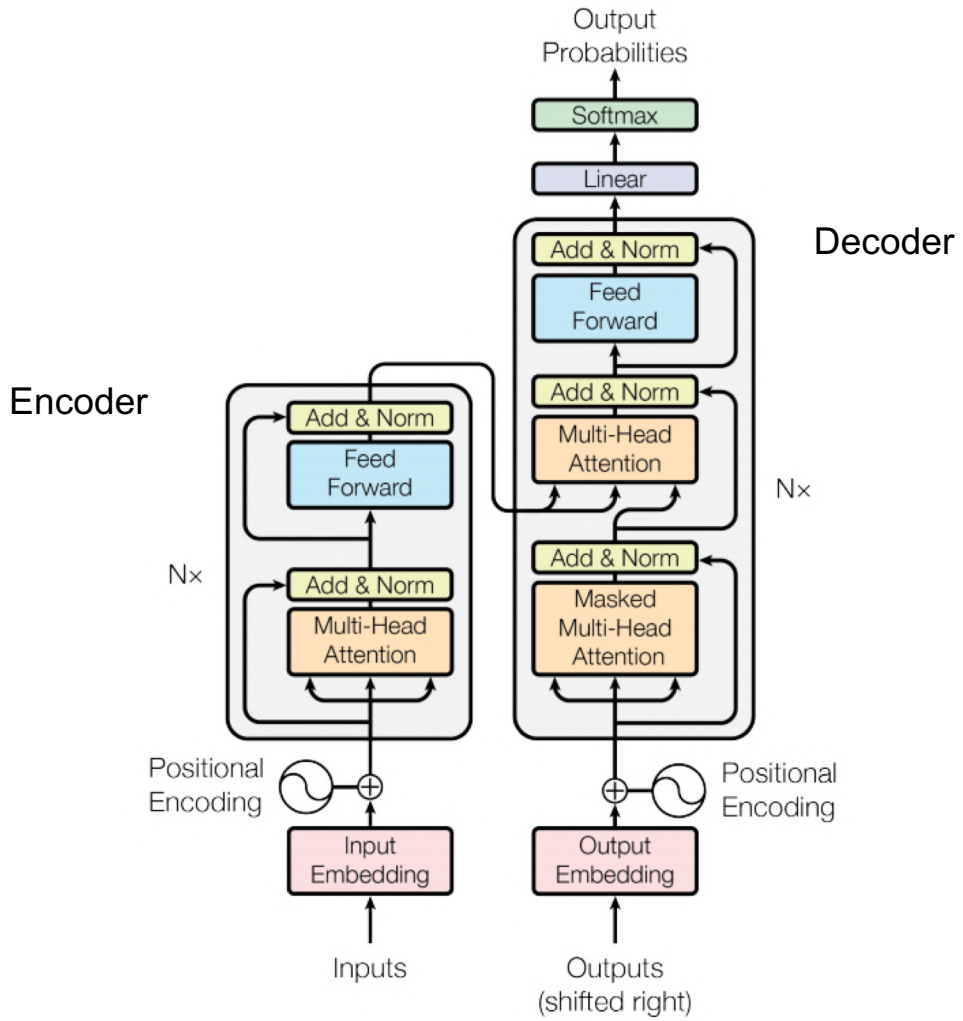
Transformer Decoder Block:

Inputs: Set of vectors **x** and Set of context vectors **c.**

Outputs: Set of vectors **y.**

- Multi-head attention block is NOT self-attention.
- It attends over encoder outputs.

Transformer architecture: Details

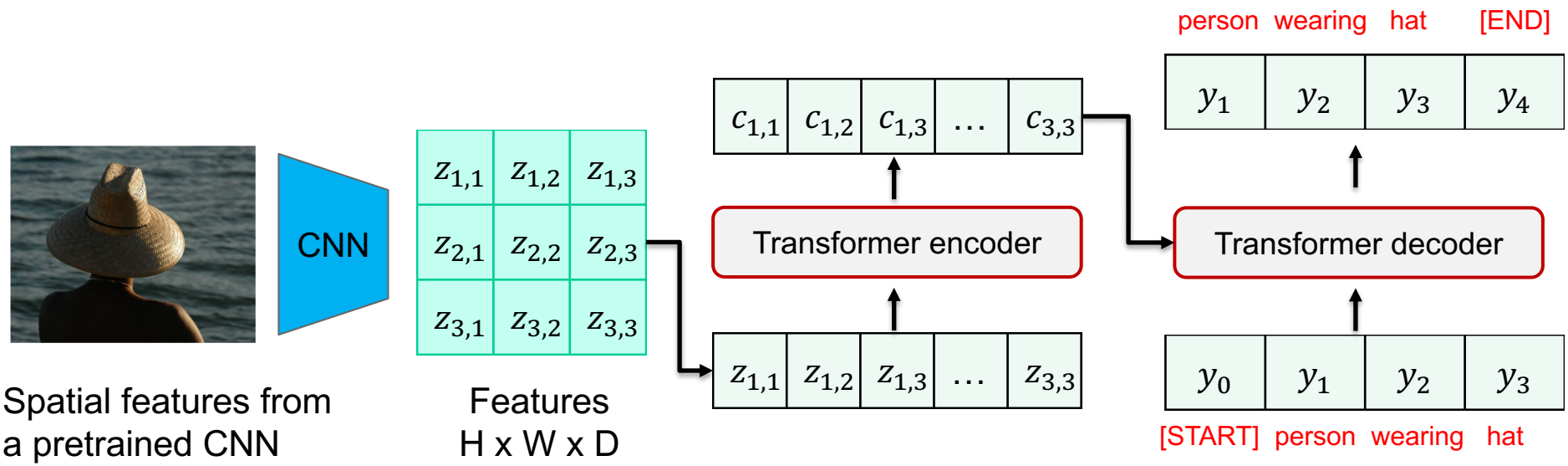


A. Vaswani et al., Attention is all you need, NeurIPS 2017

Back to Image captioning

Input: image I

Output: $y = y_1, y_2, y_3 \ldots, y_T$



Back to Image captioning

Perhaps we don't need convolution at all!

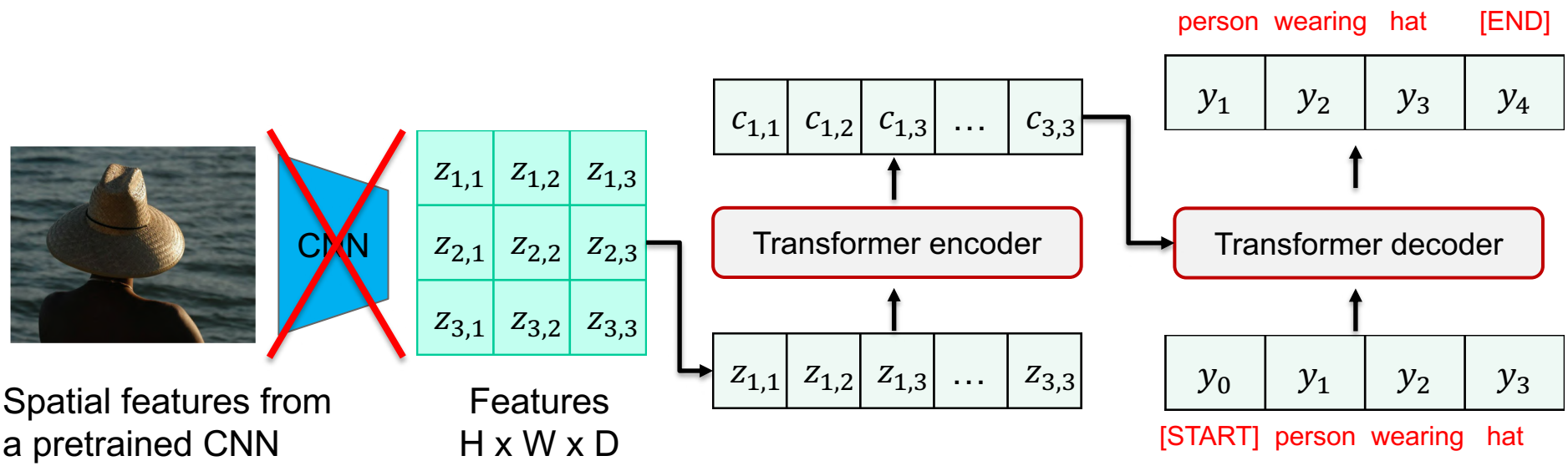
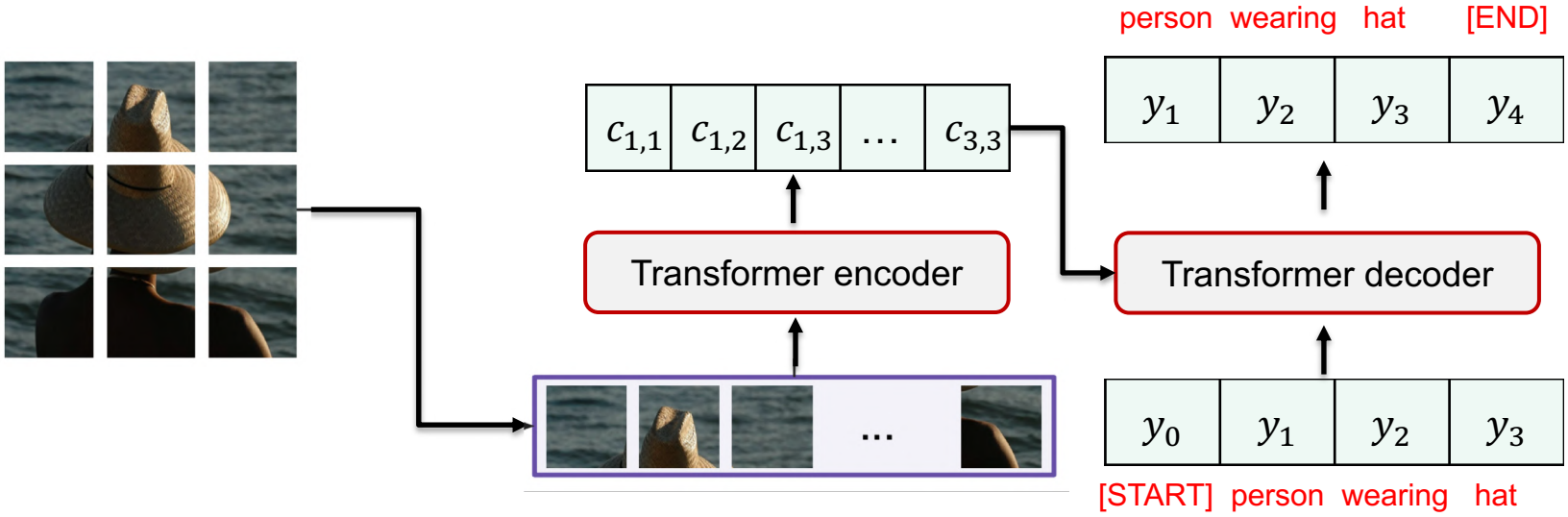


Image captioning using **only** transformers

Transformers directly from pixels to language.

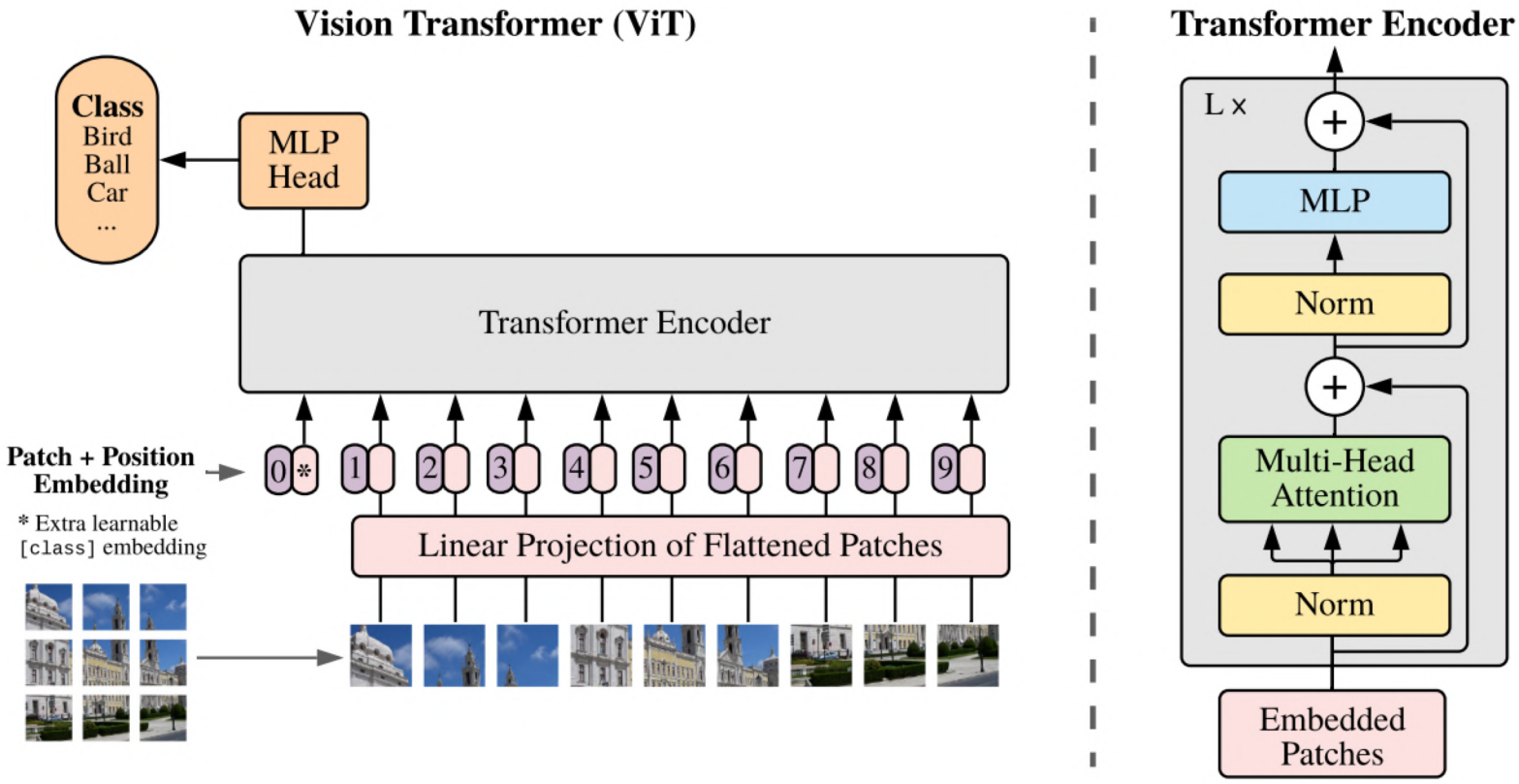
This is called a "***Vision Transformer***" (ViT).

Suggested in: A. Dosovitskiy et al. An image is worth 16x16 words: Transformers for image recognition at scale. ICLR 2021



Vision Transformers for Image Classification

- Split an image into patches, feed linearly projected patches into standard transformer encoder

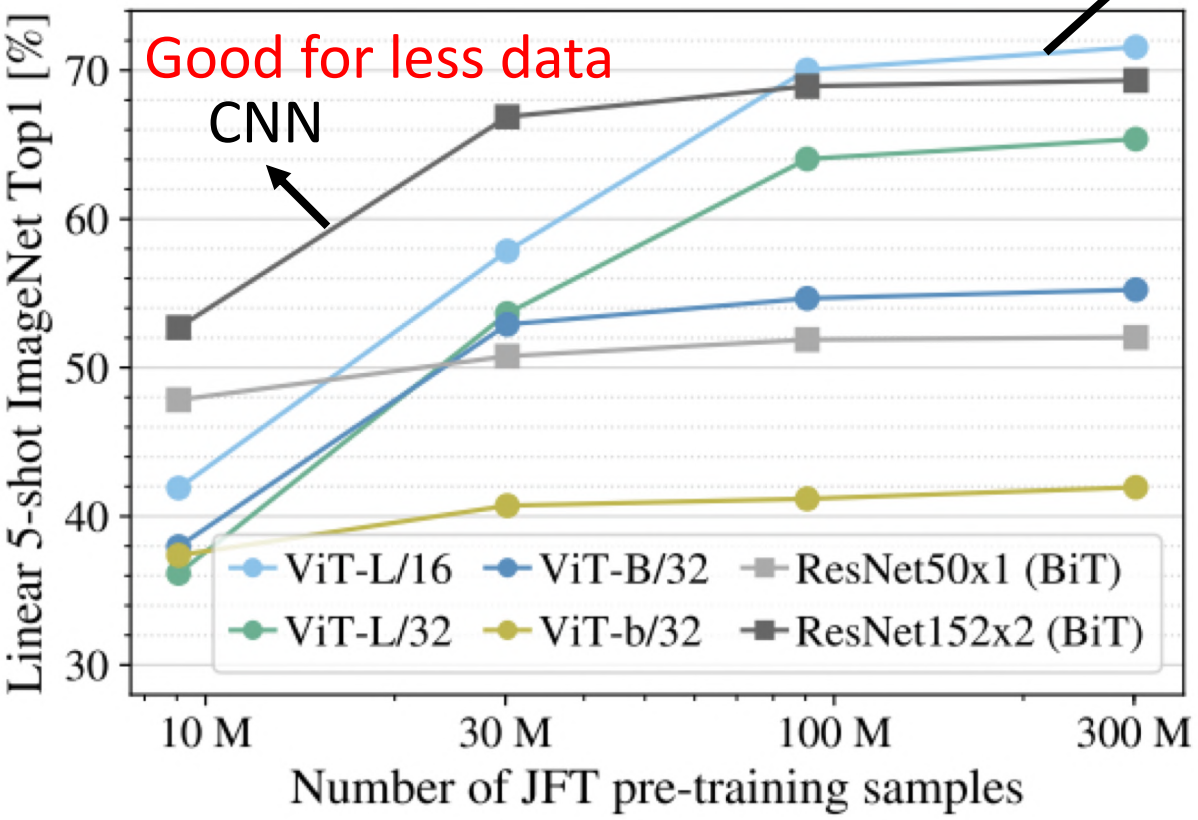


A. Dosovitskiy et al. An image is worth 16x16 words: Transformers for image recognition at scale. ICLR 2021

Self-attention v.s. CNN

Good for more data

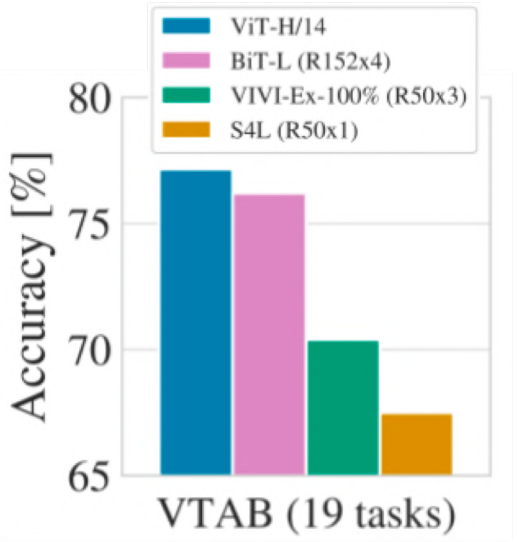
Self-attention



A. Dosovitskiy et al. An image is worth 16x16 words: Transformers for image recognition at scale. ICLR 2021

ViT Results

	ViT-H	Previous SOTA
ImageNet	88.55	88.5
ImageNet-ReaL	90.72	90.55
Cifar-10	99.50	99.37
Cifar-100	94.55	93.51
Pets	97.56	96.62
Flowers	99.68	99.63



Vision Transformer matches or outperforms state-of-the-art CNNs on popular benchmarks. Left: Popular image classification tasks (ImageNet, including new validation labels ReaL, and CIFAR, Pets, and Flowers). Right: Average across 19 tasks in the VTAB classification suite.

Image transformer – Google

- Image generation and super-resolution with 32x32 output, attention restricted to local neighborhoods

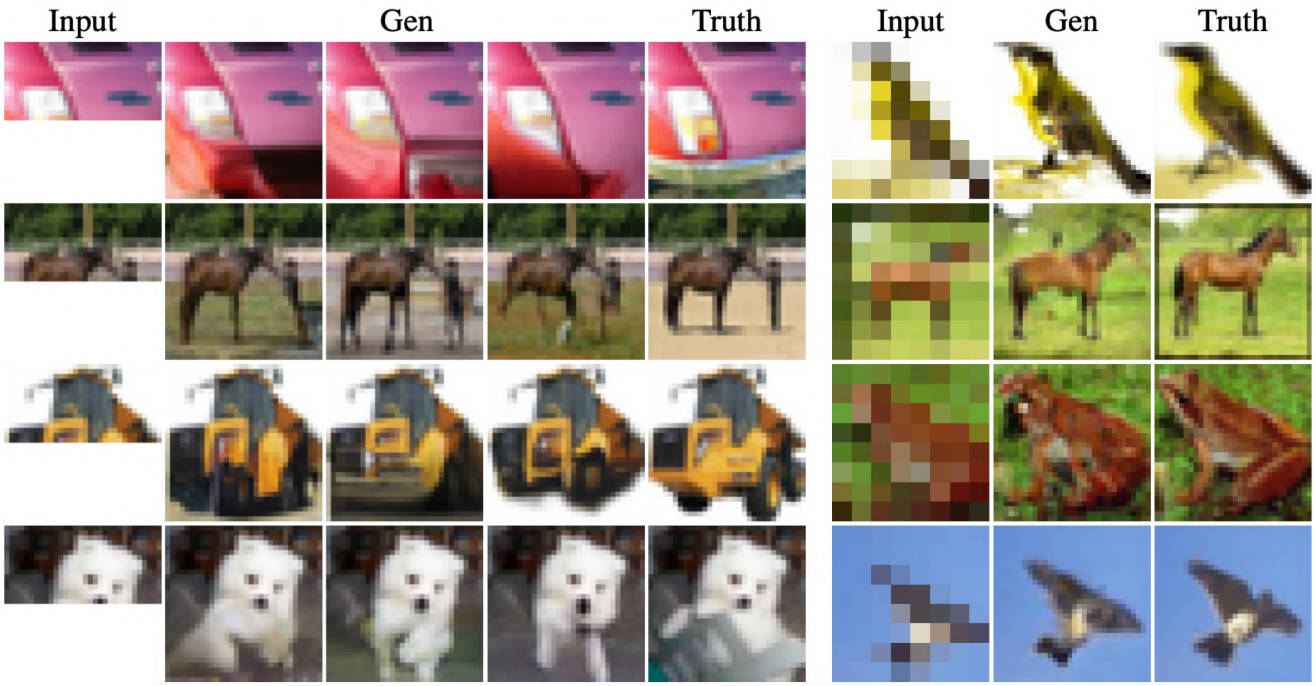
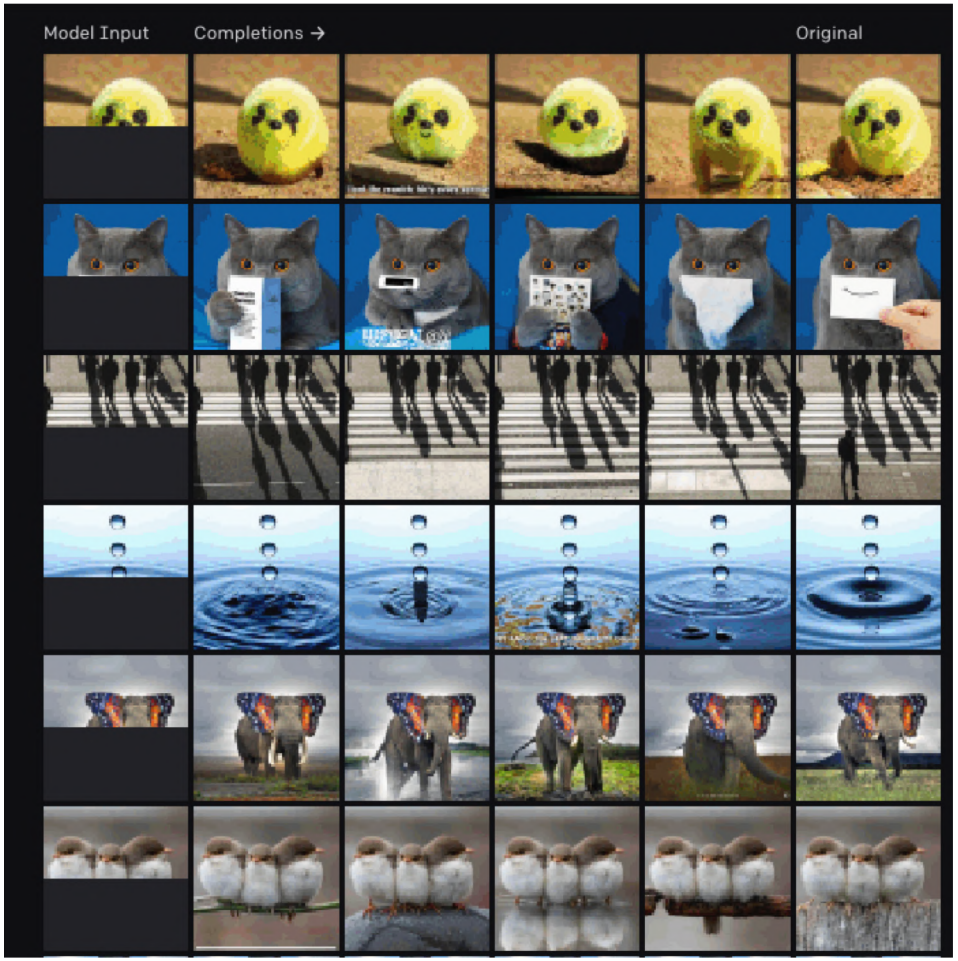


Table 2. On the left are image completions from our best conditional generation model, where we sample the second half. On the right are samples from our four-fold super-resolution model trained on CIFAR-10. Our images look realistic and plausible, show good diversity among the completion samples and observe the outputs carry surprising details for coarse inputs in super-resolution.

N. Parmar et al., Image transformer, ICML 2018

Image GPT – OpenAI



M. Chen et al., Generative pretraining from pixels, ICML 2020

https://openai.com/blog/image-gpt/

Image transformer – Google

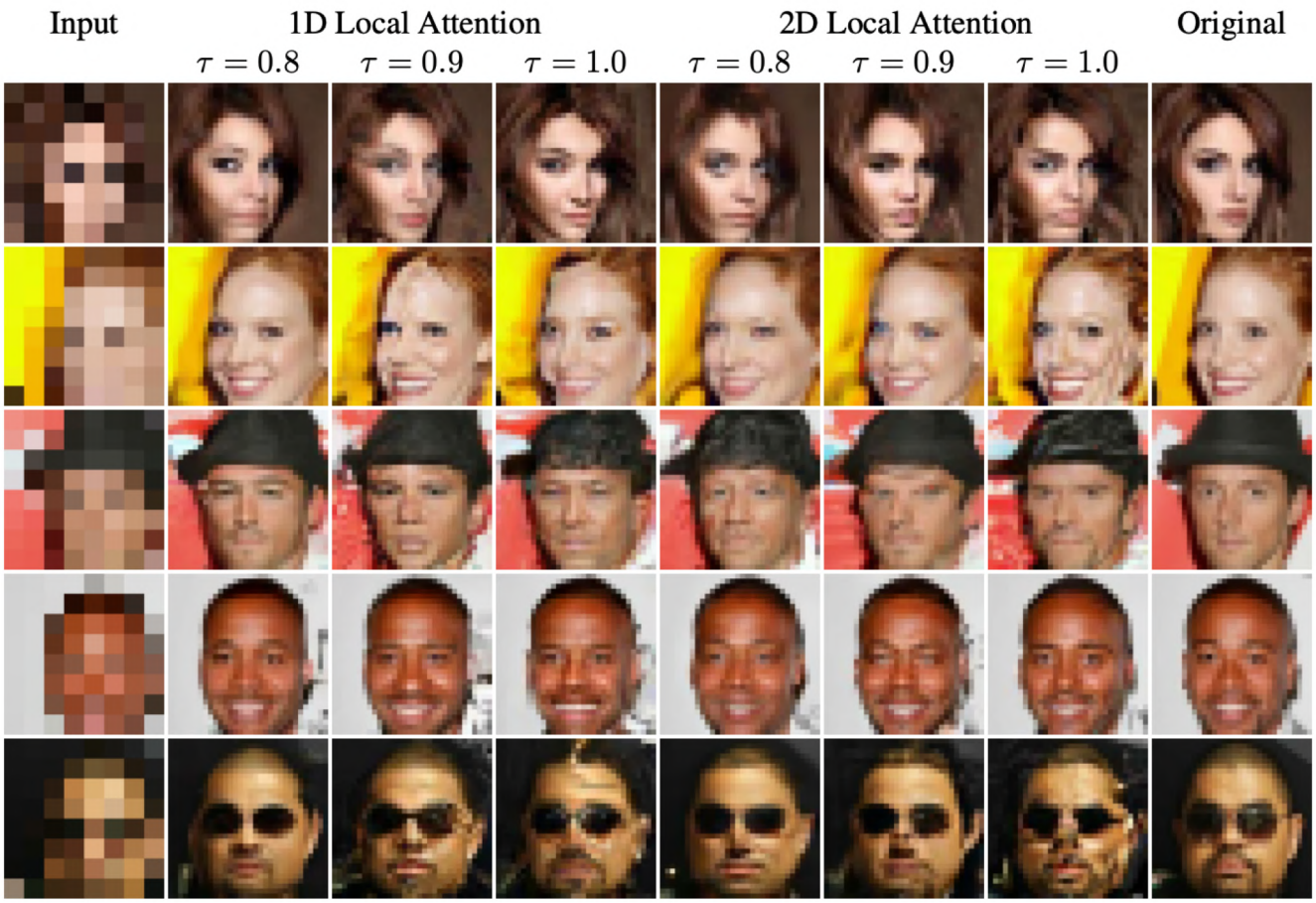
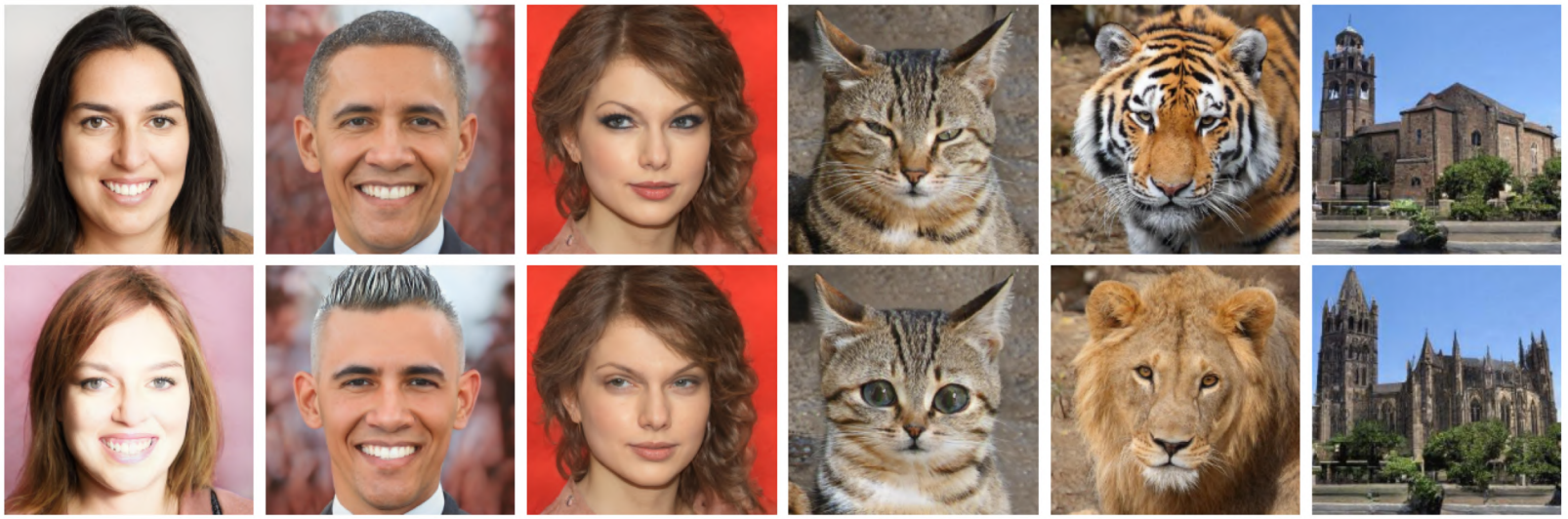


Table 6. Images from our 1D and 2D local attention super-resolution models trained on CelebA, sampled with different temperatures. 2D local attention with $\tau = 0.9$ scored highest in our human evaluation study.

N. Parmar et al., Image transformer, ICML 2018

CLIP for image editing

- Combine powerful recent image-text embedding technique (CLIP) with pre-trained StyleGAN for text-based image editing



O. Patashnik et al.. StyleCLIP: Text-Driven Manipulation of StyleGAN Imagery. arXiv 2021

Text-based image generation: DALL-E

- Learn a joint sequential transformer model that can be used to generate image based on text prompt

(a) a tapir made of accordion. a tapir with the texture of an accordion.

(b) an illustration of a baby hedgehog in a christmas sweater walking a dog

(c) a neon sign that reads "backprop". a neon sign that reads "backprop". backprop neon sign

A. Ramesh et al., Zero-Shot Text-to-Image Generation, arXiv 2021. https://openai.com/blog/dall-e/

Text-based image generation: DALL-E

- Learn a joint sequential transformer model that can be used to generate image based on text prompt

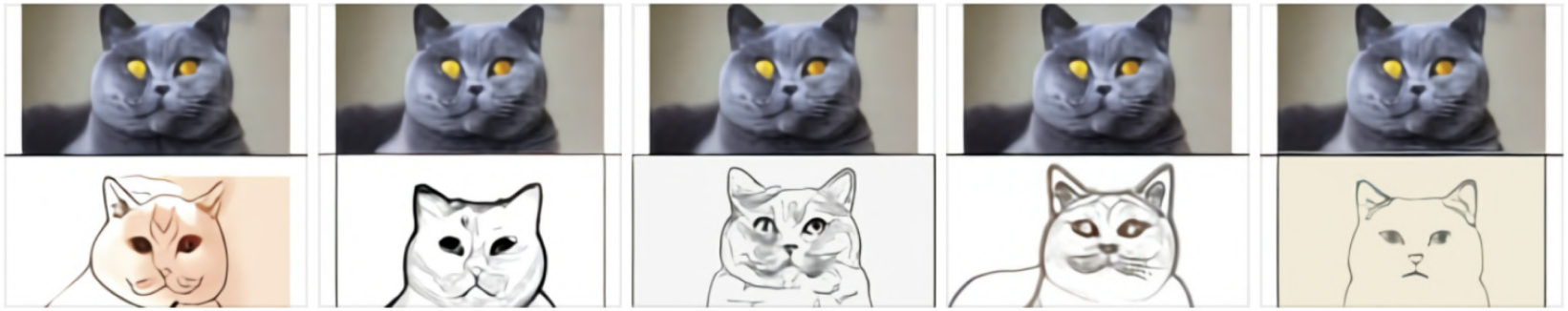
a store front that has the word ‘openai’ written on it. . . .

A. Ramesh et al., Zero-Shot Text-to-Image Generation, arXiv 2021. https://openai.com/blog/dall-e/

Text-based image generation: DALL-E

- Learn a joint sequential transformer model that can be used to generate image based on text prompt

the exact same cat on the top as a sketch on the bottom



A. Ramesh et al., Zero-Shot Text-to-Image Generation, arXiv 2021. https://openai.com/blog/dall-e/

Text-based image generation: DALL-E

- Learn a joint sequential transformer model that can be used to generate image based on text prompt

an armchair in the shape of an avocado. . . .

A. Ramesh et al., Zero-Shot Text-to-Image Generation, arXiv 2021. https://openai.com/blog/dall-e/

Summary

- Adding **attention** to RNNs allows them to "attend" to different parts of the input at every time step
- The **general attention layer** is a new type of layer that can be used to design new neural network architectures
- **Transformers** are a type of layer that uses **self-attention** and layer norm.
  - It is highly **scalable** and highly **parallelizable**
  - **Faster** training, **larger** models, **better** performance across vision and language tasks
  - They are quickly replacing RNNs, LSTMs, and may even replace convolutions.