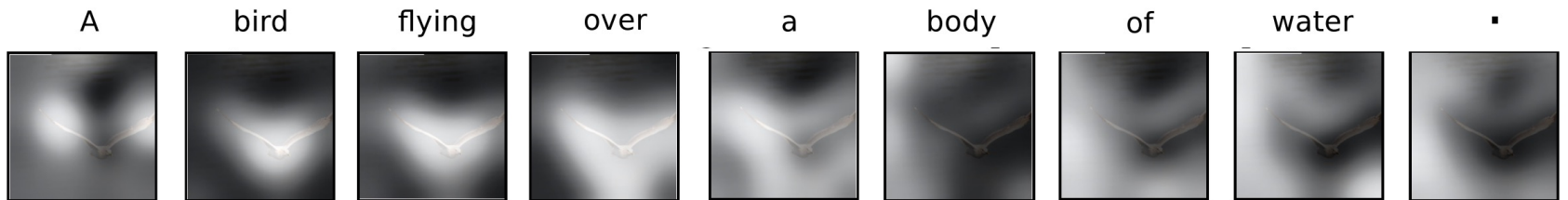
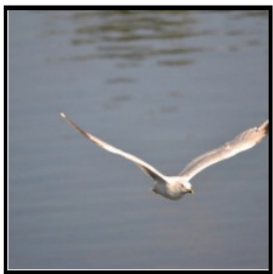
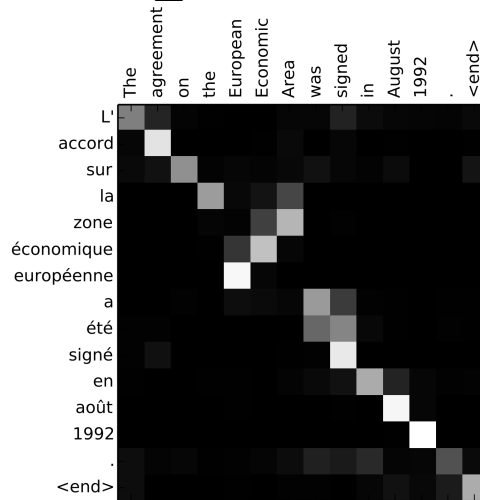


# Sequence-to-sequence modeling with attention



Many slides adapted from [J. Johnson](#)

# Outline

---

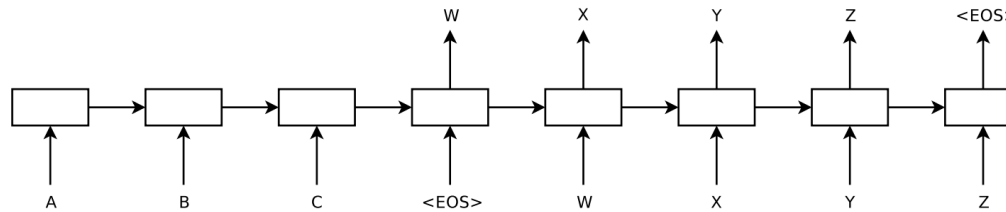
- Vanilla seq2seq with RNNs
- Seq2seq with RNNs and attention
- Image captioning with attention
- Transformers

# Sequence-to-sequence modeling: Machine translation

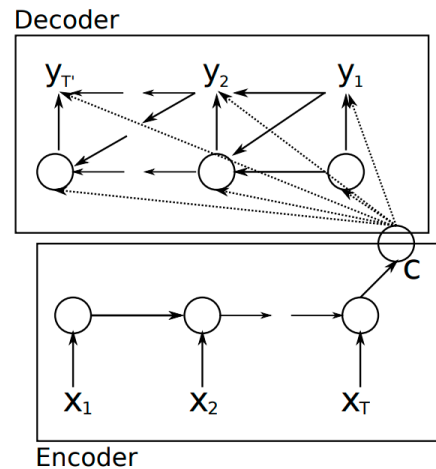
“We are eating bread”  “Estamos comiendo pan”

# Sequence-to-sequence modeling with RNNs

---

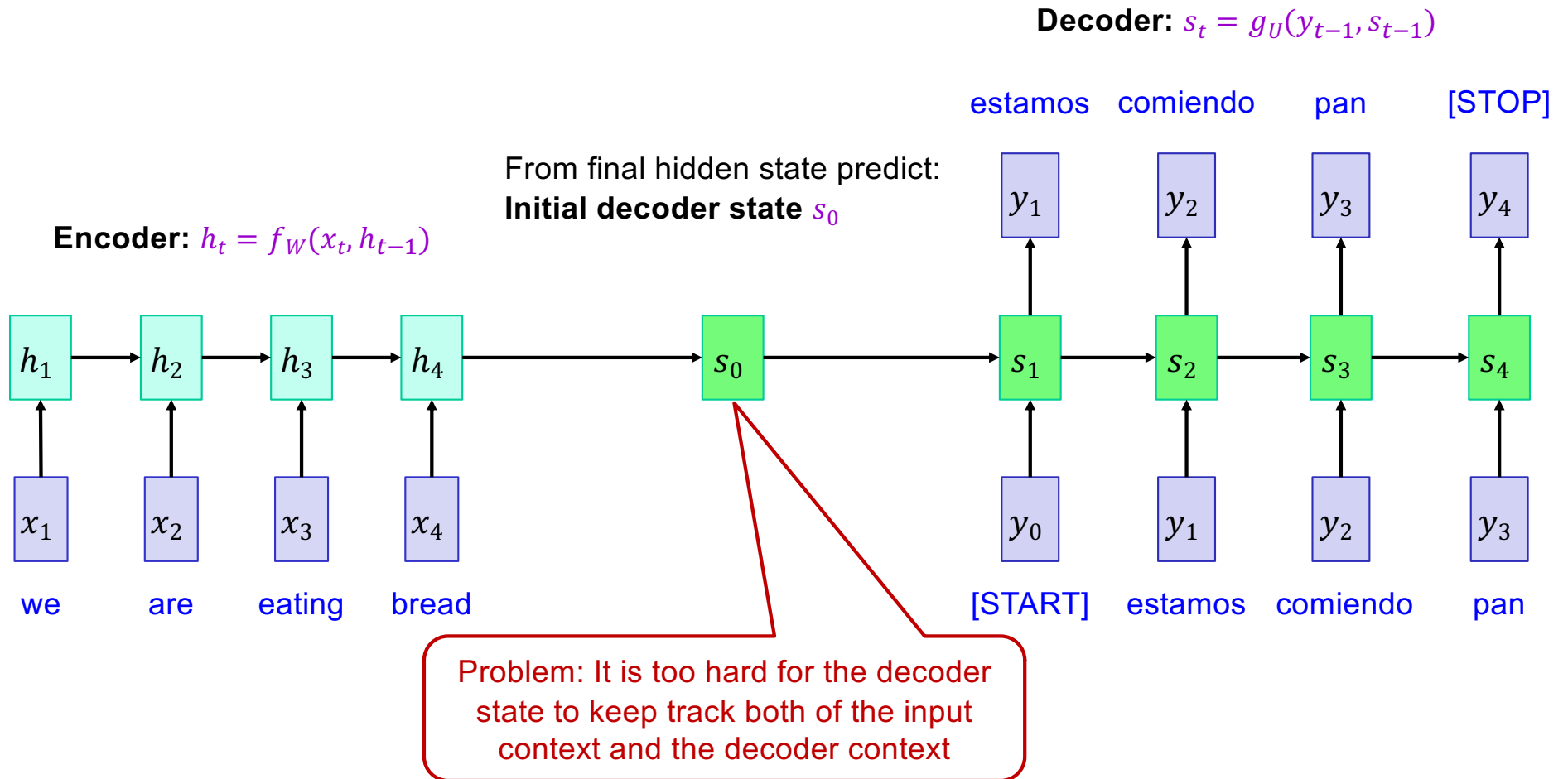


I. Sutskever et al. [Sequence to Sequence Learning with Neural Networks](#). NeurIPS 2014



K. Cho et al. [Learning phrase representations using RNN encoder-decoder for statistical machine translation](#). ACL 2014

# Sequence-to-sequence modeling with RNNs

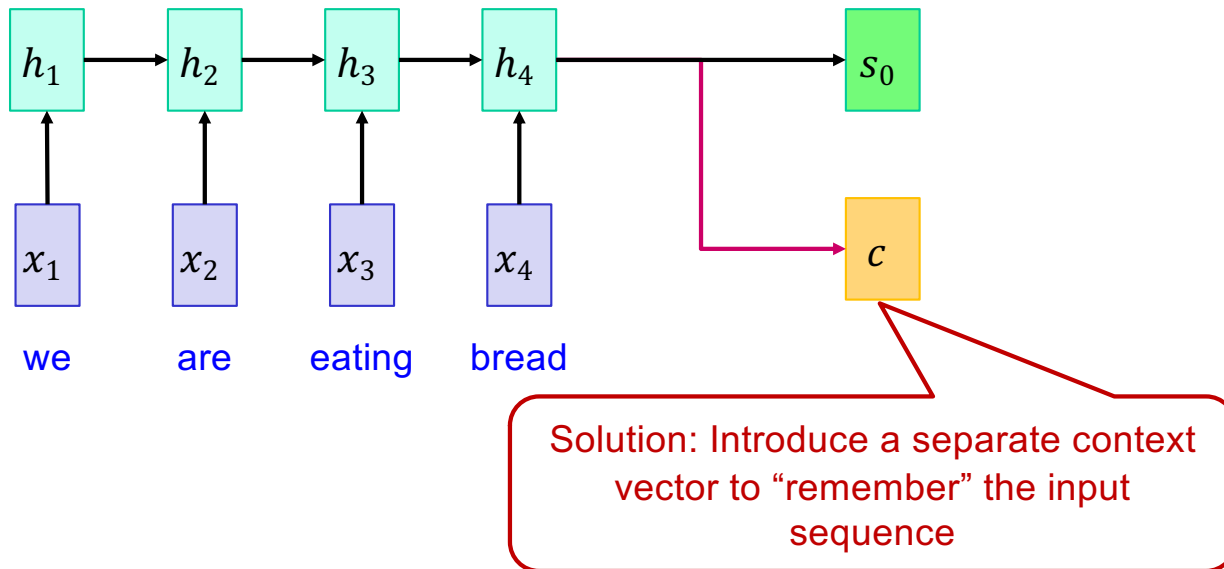


# Sequence-to-sequence modeling with RNNs

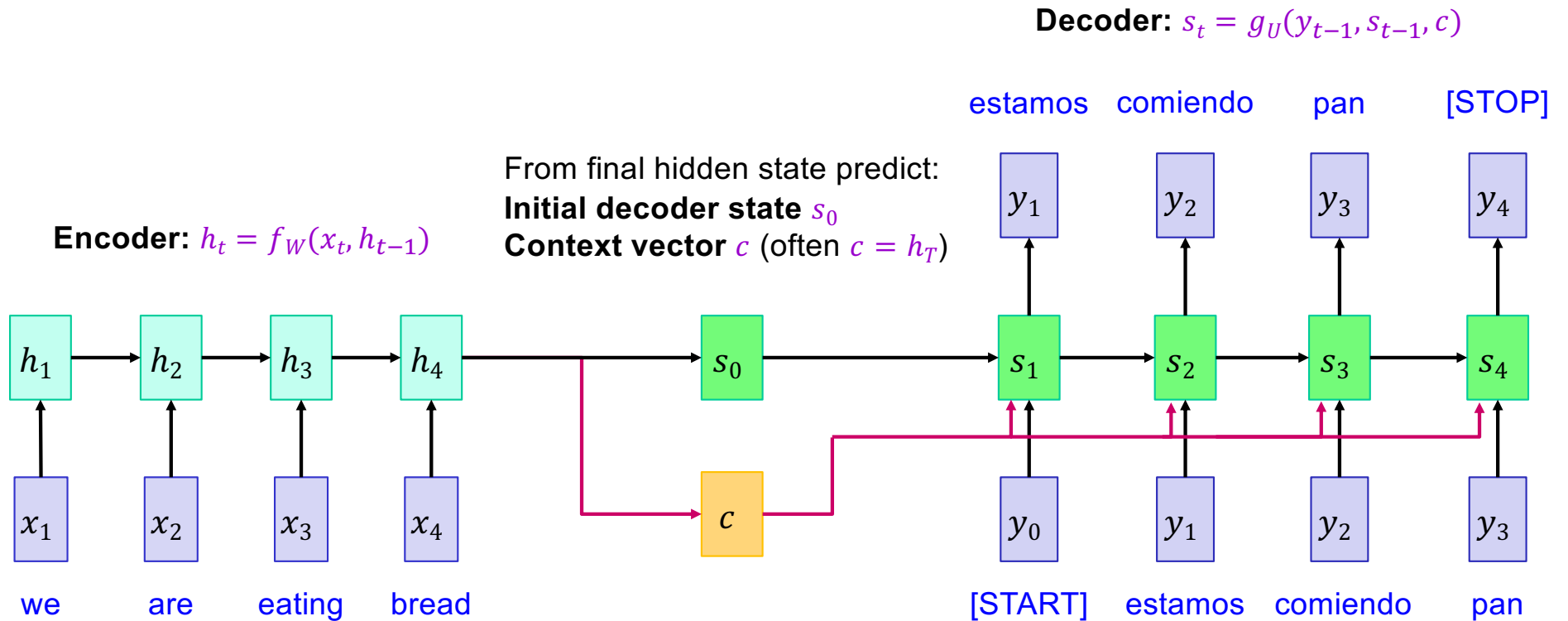
---

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

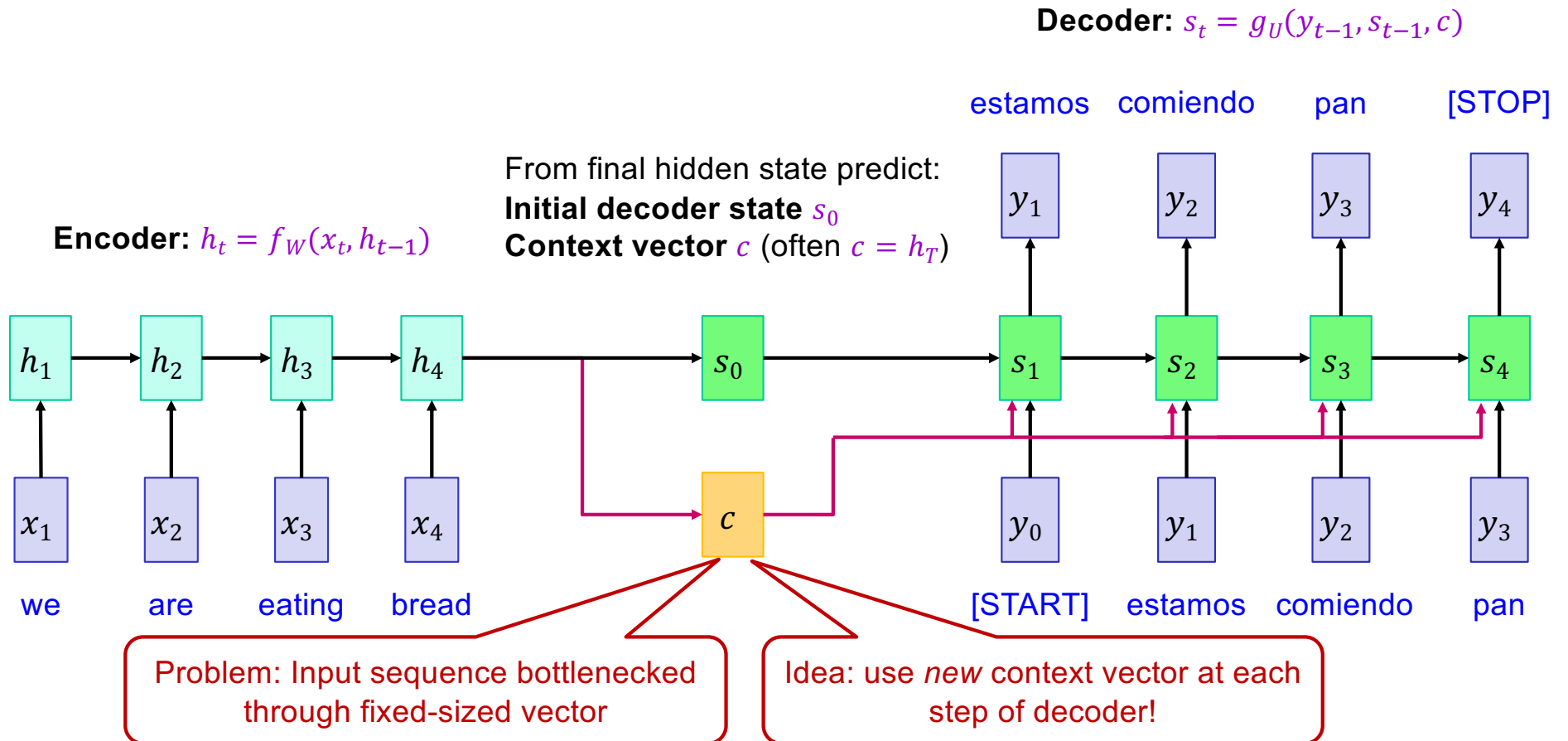
From final hidden state predict:  
**Initial decoder state**  $s_0$   
**Context vector**  $c$  (often  $c = h_T$ )



# Sequence-to-sequence modeling with RNNs



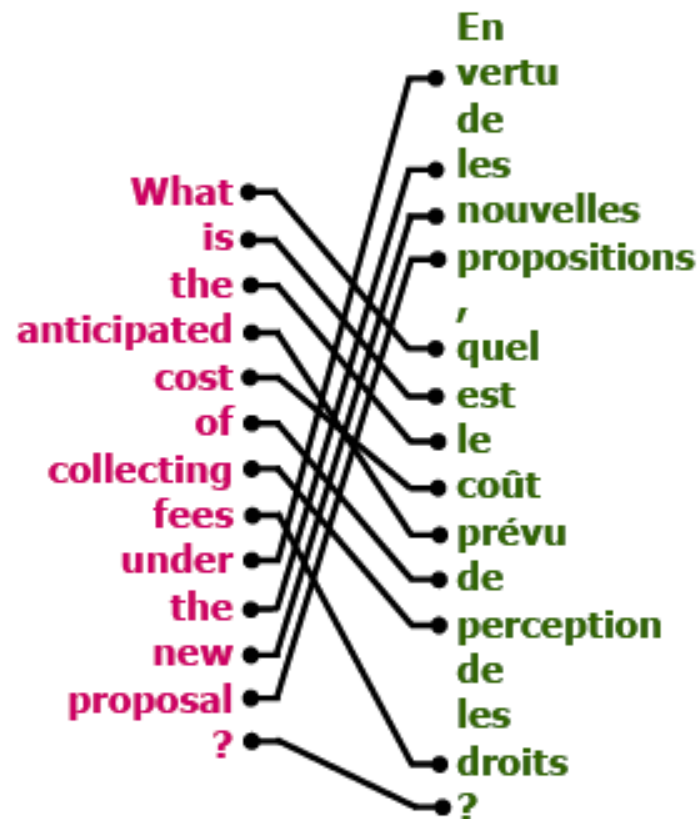
# Sequence-to-sequence modeling with RNNs



# Sequence-to-sequence modeling with RNNs *and attention*

---

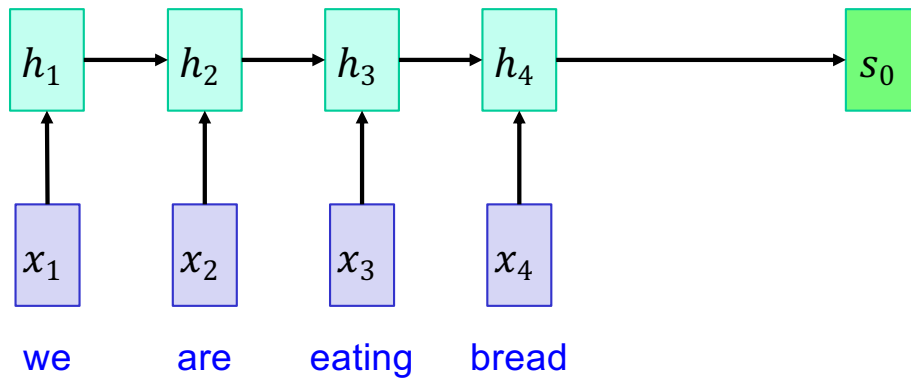
- Intuition: translation requires *alignment*



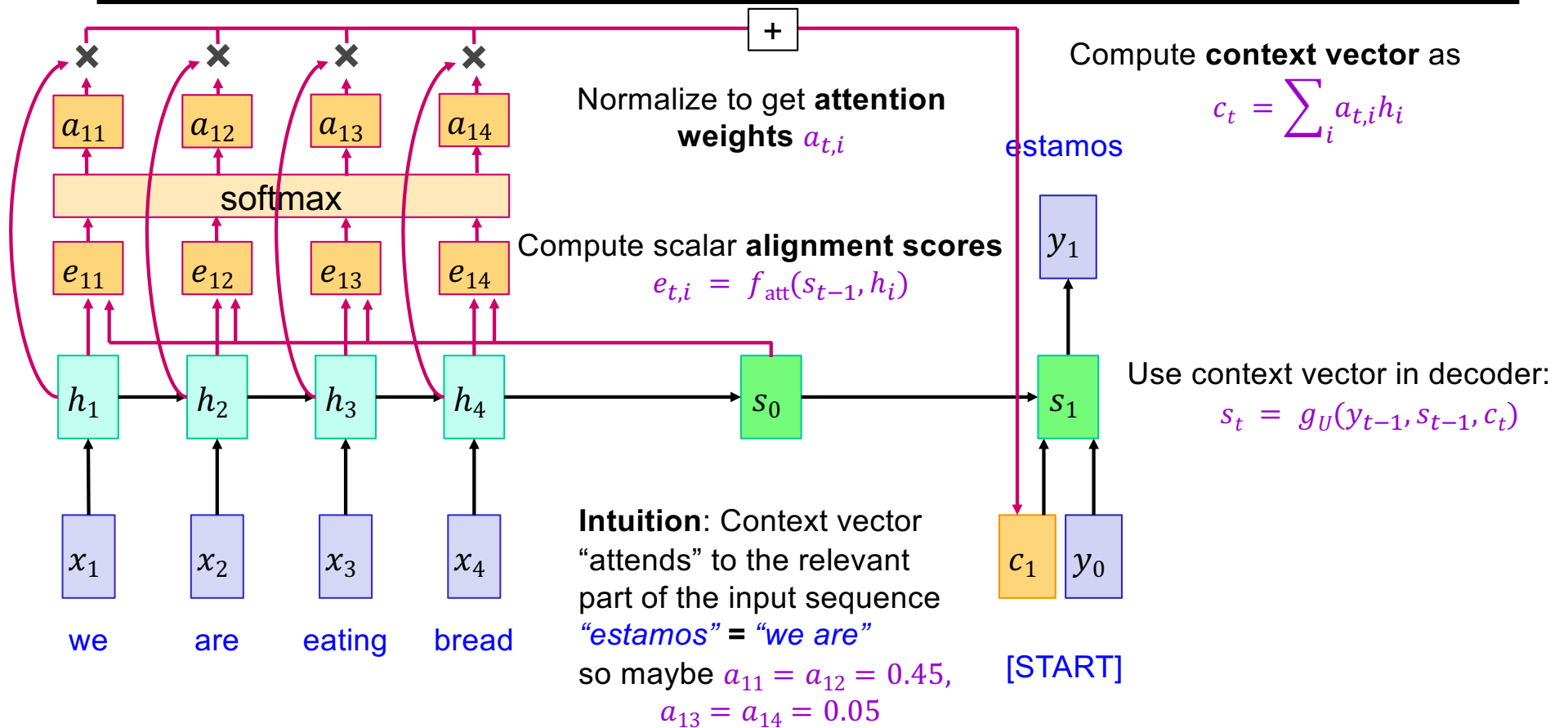
# Sequence-to-sequence modeling with RNNs *and attention*

---

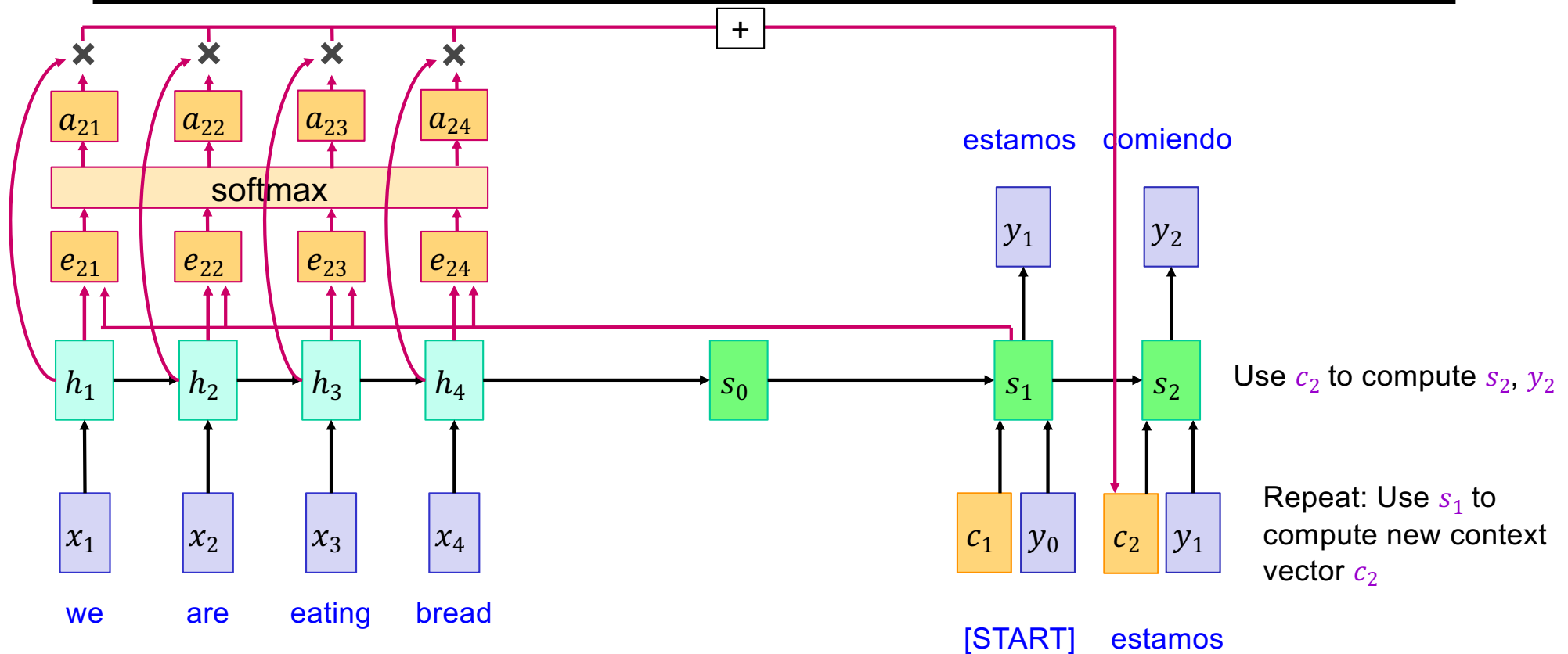
- At each timestep of decoder, context vector “looks at” different parts of the input sequence



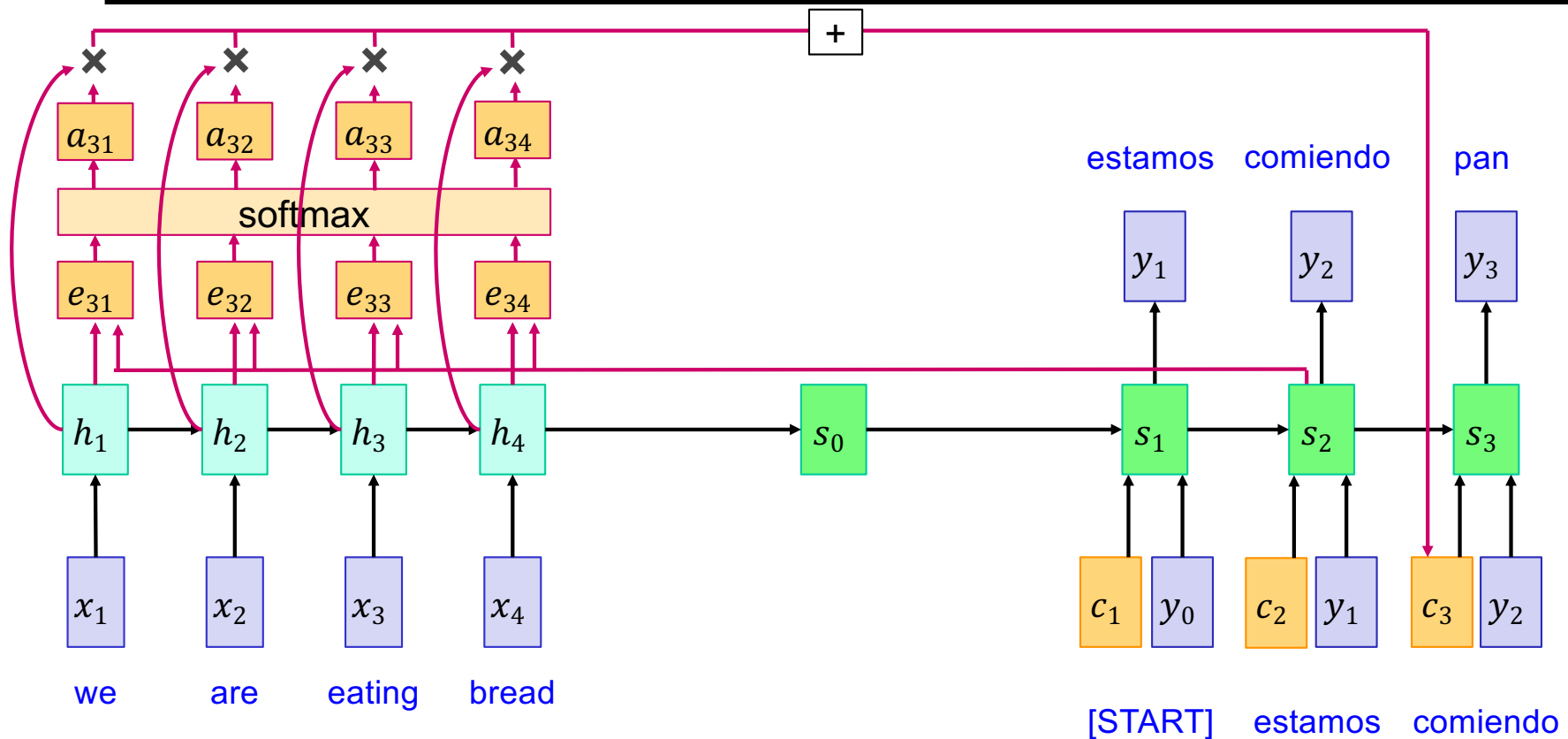
# Sequence-to-sequence modeling with RNNs *and attention*



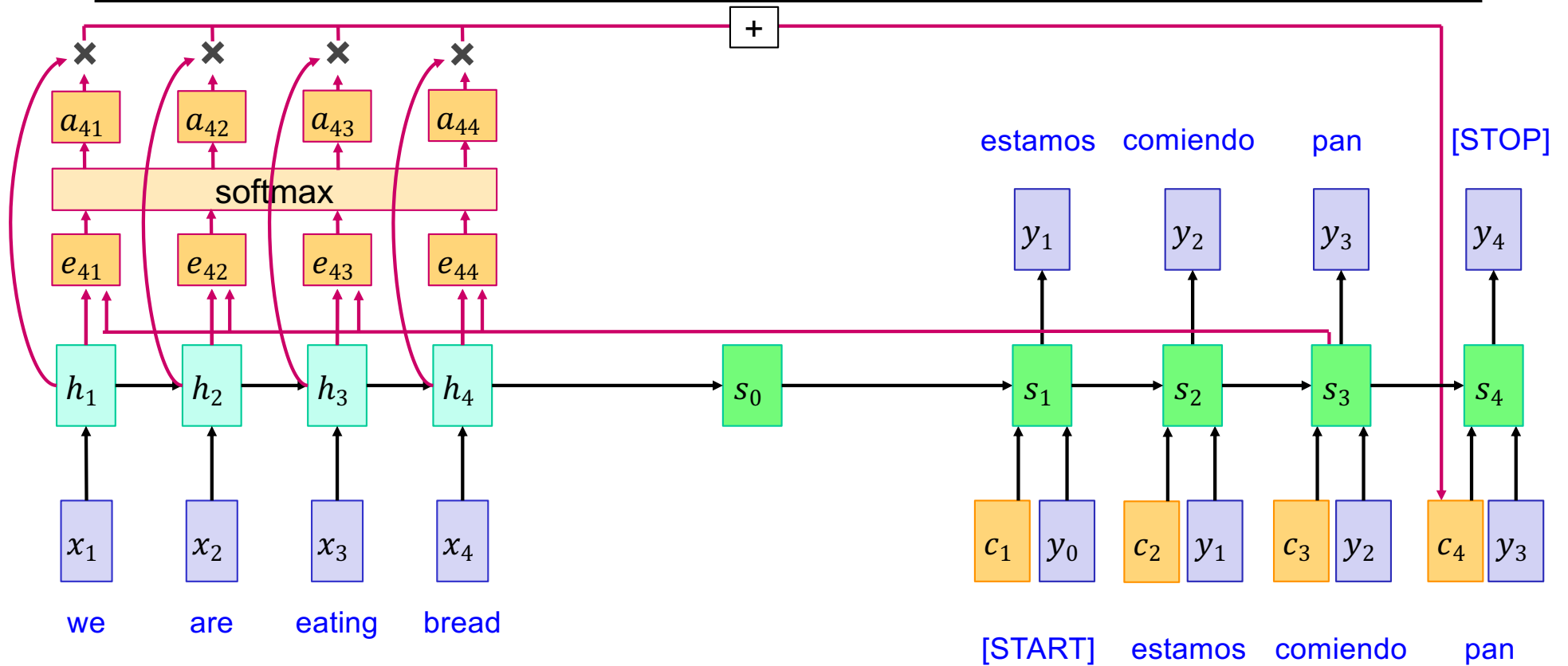
# Sequence-to-sequence modeling with RNNs *and attention*



# Sequence-to-sequence modeling with RNNs *and attention*

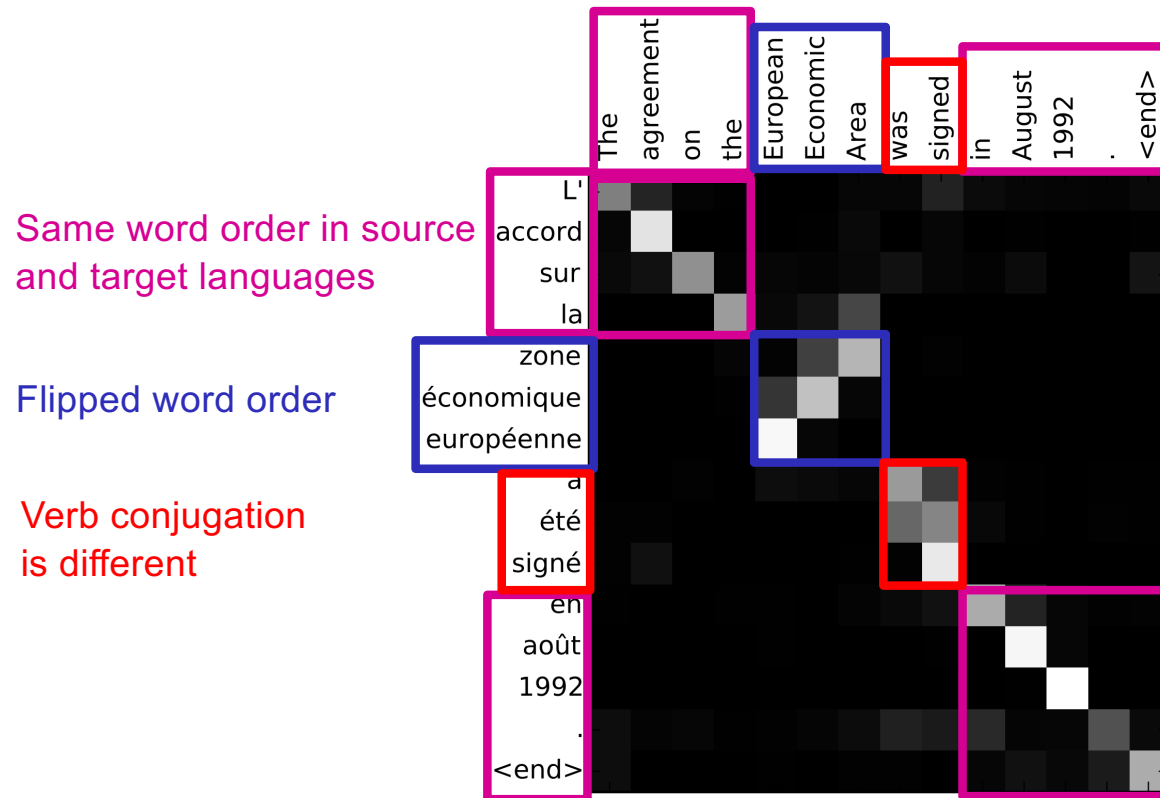


# Sequence-to-sequence modeling with RNNs *and attention*

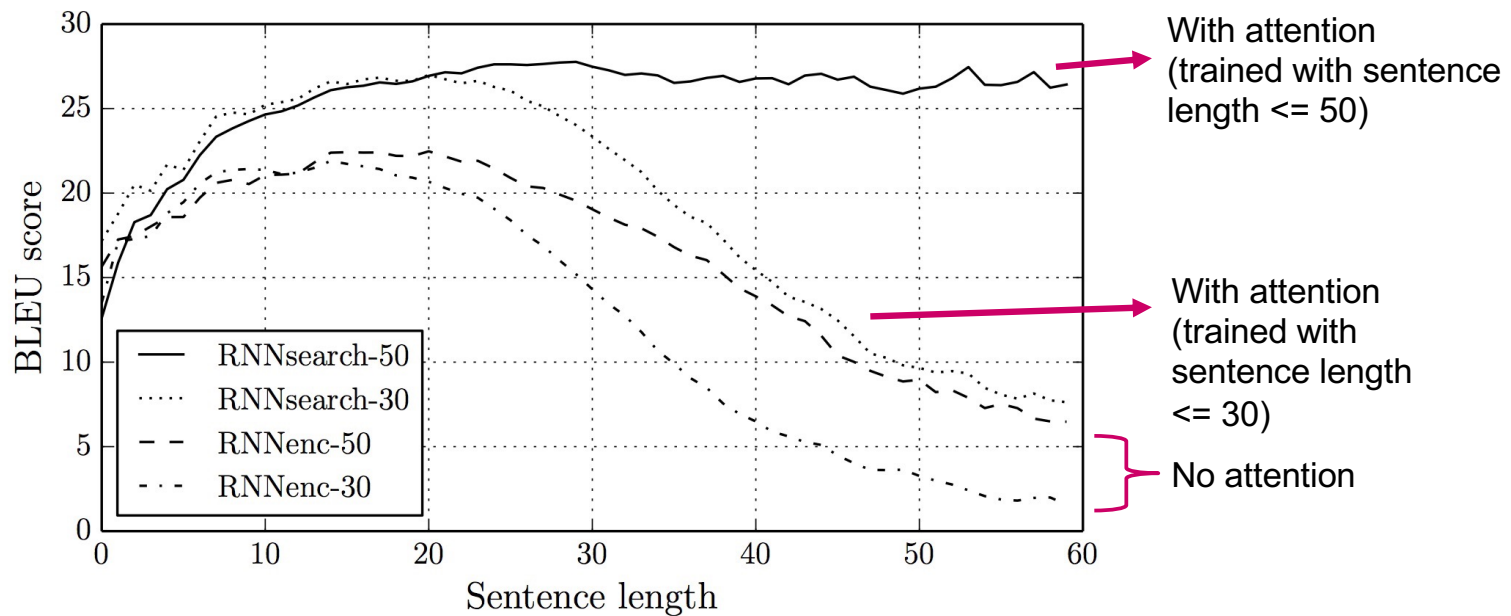


# Visualizing attention weights

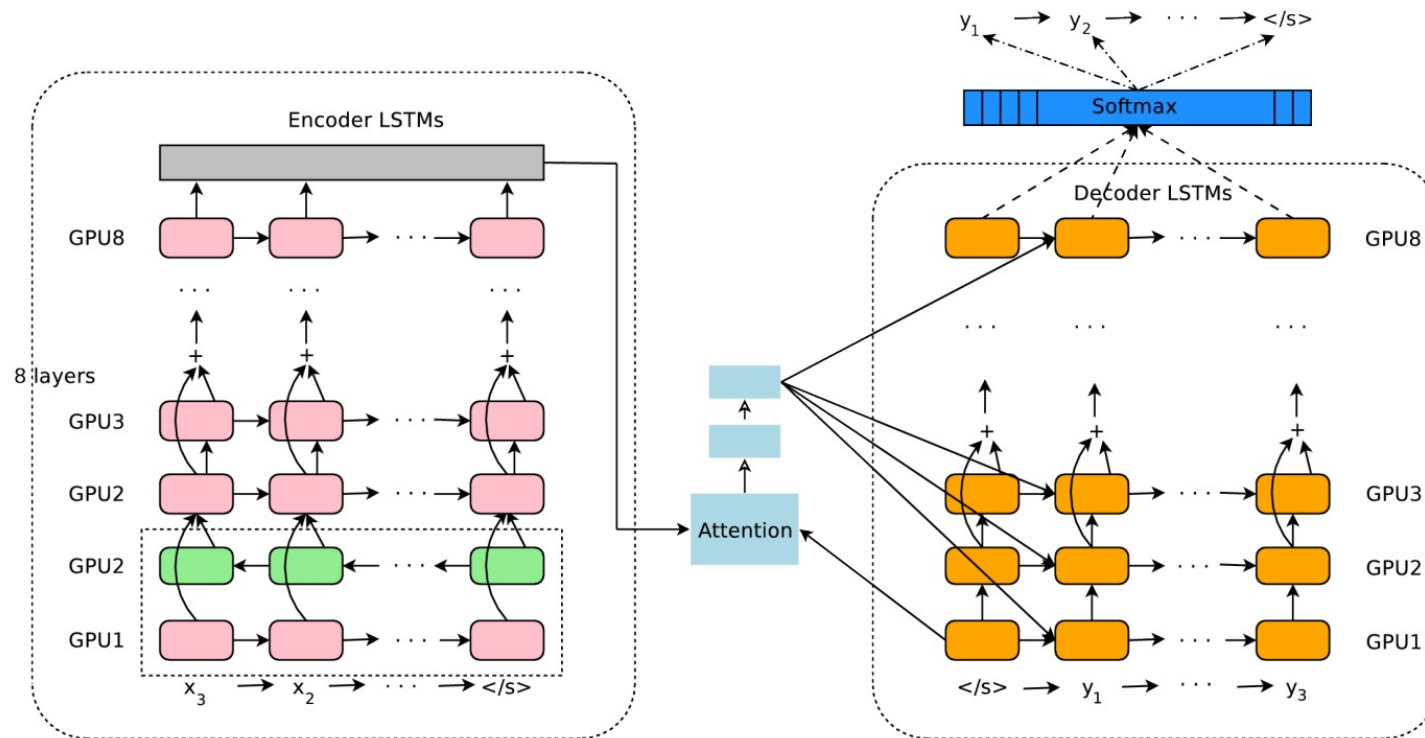
- English source, French target:



# Quantitative evaluation



# Recall: GNMT



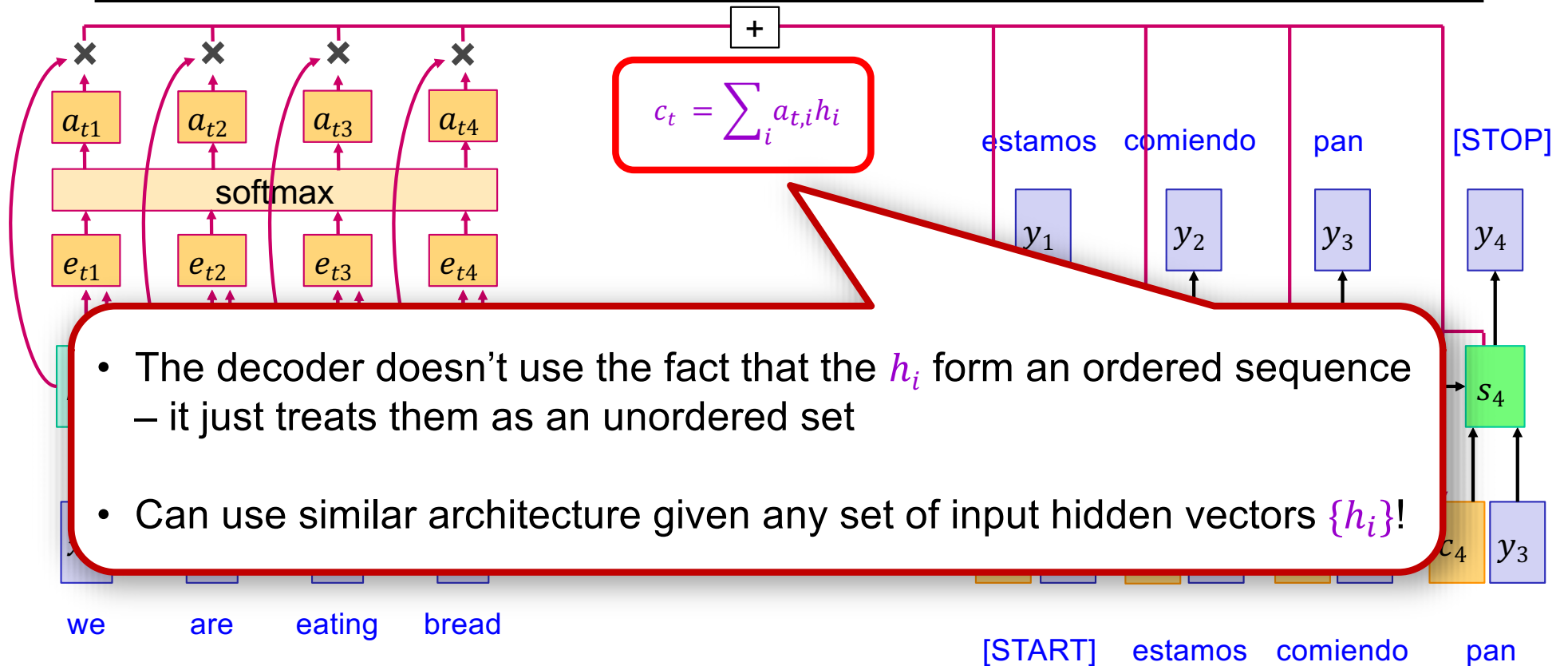
Y. Wu et al. [Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation](#). arXiv 2016

# Outline

---

- Vanilla seq2seq with RNNs
- Seq2seq with RNNs and attention
- Image captioning with attention

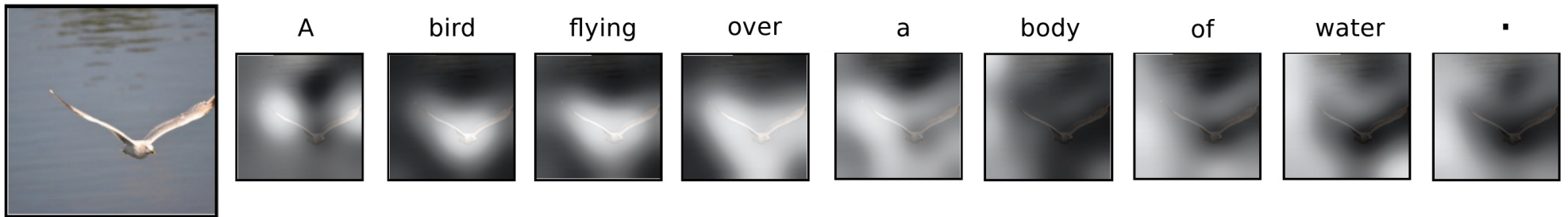
# Generalizing attention



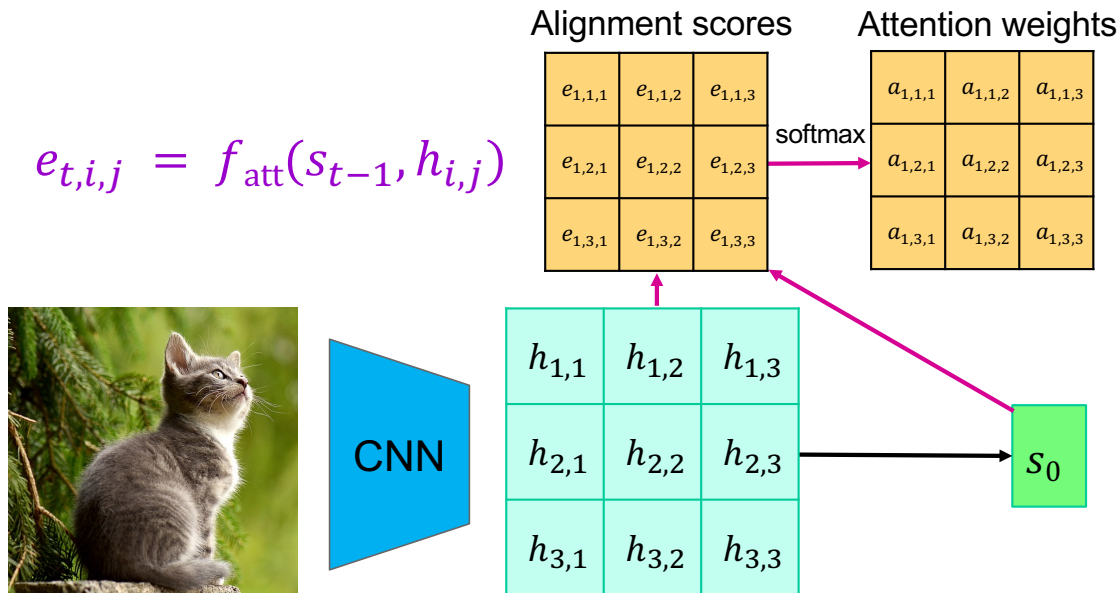
# Image captioning with RNNs and attention

---

- 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

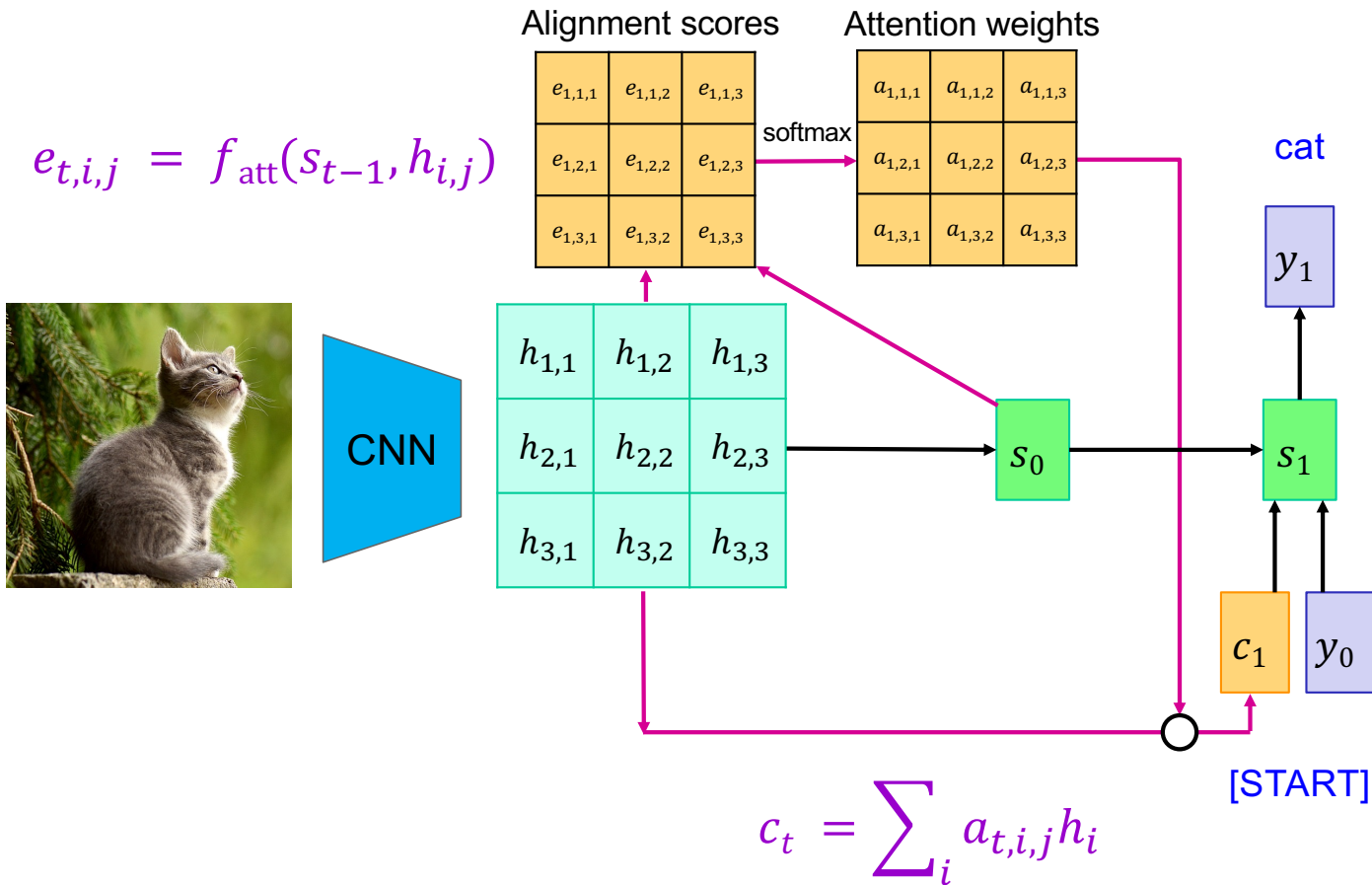


# Image captioning with RNNs and attention

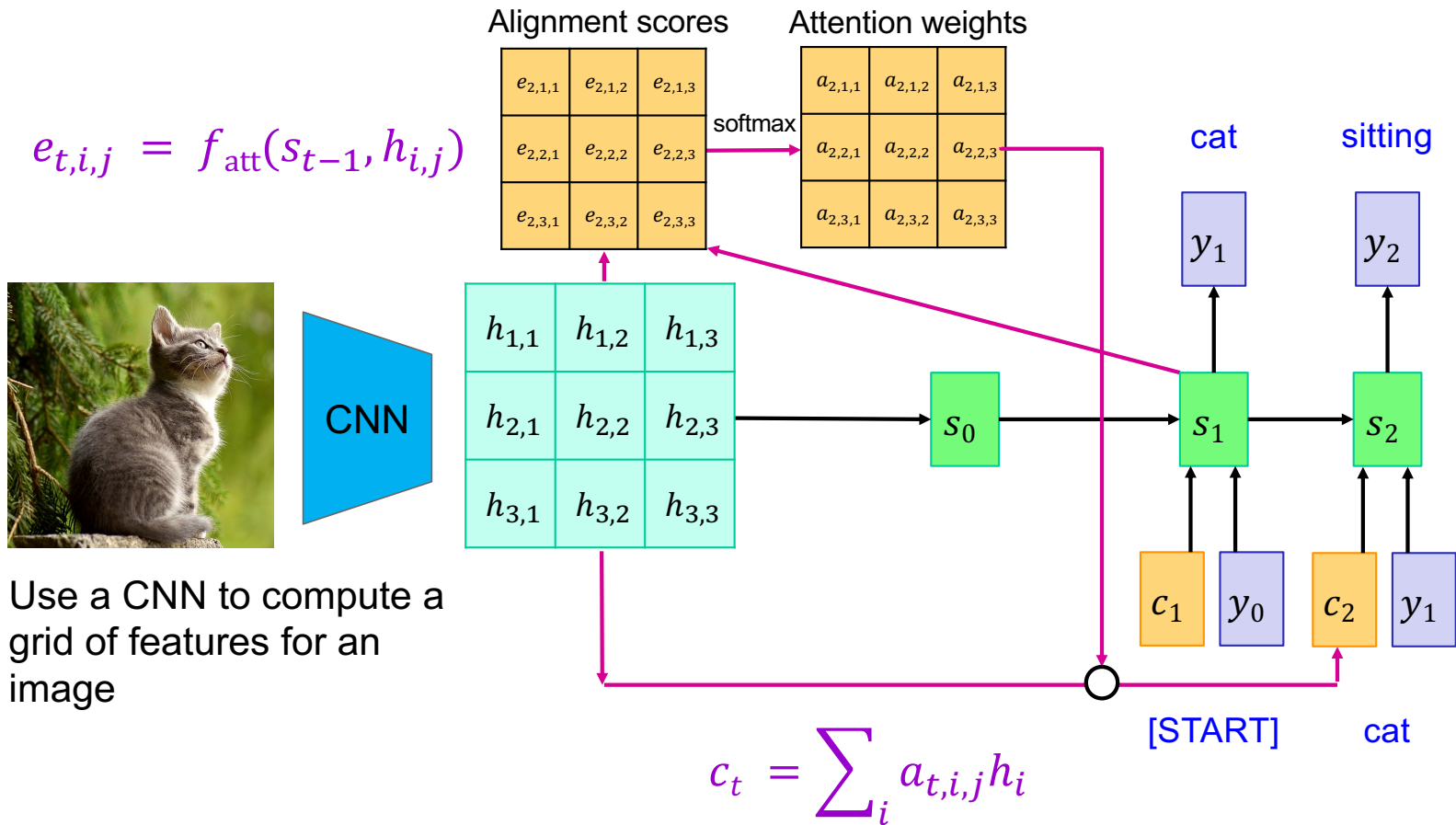


Use CNN to extract a grid of features

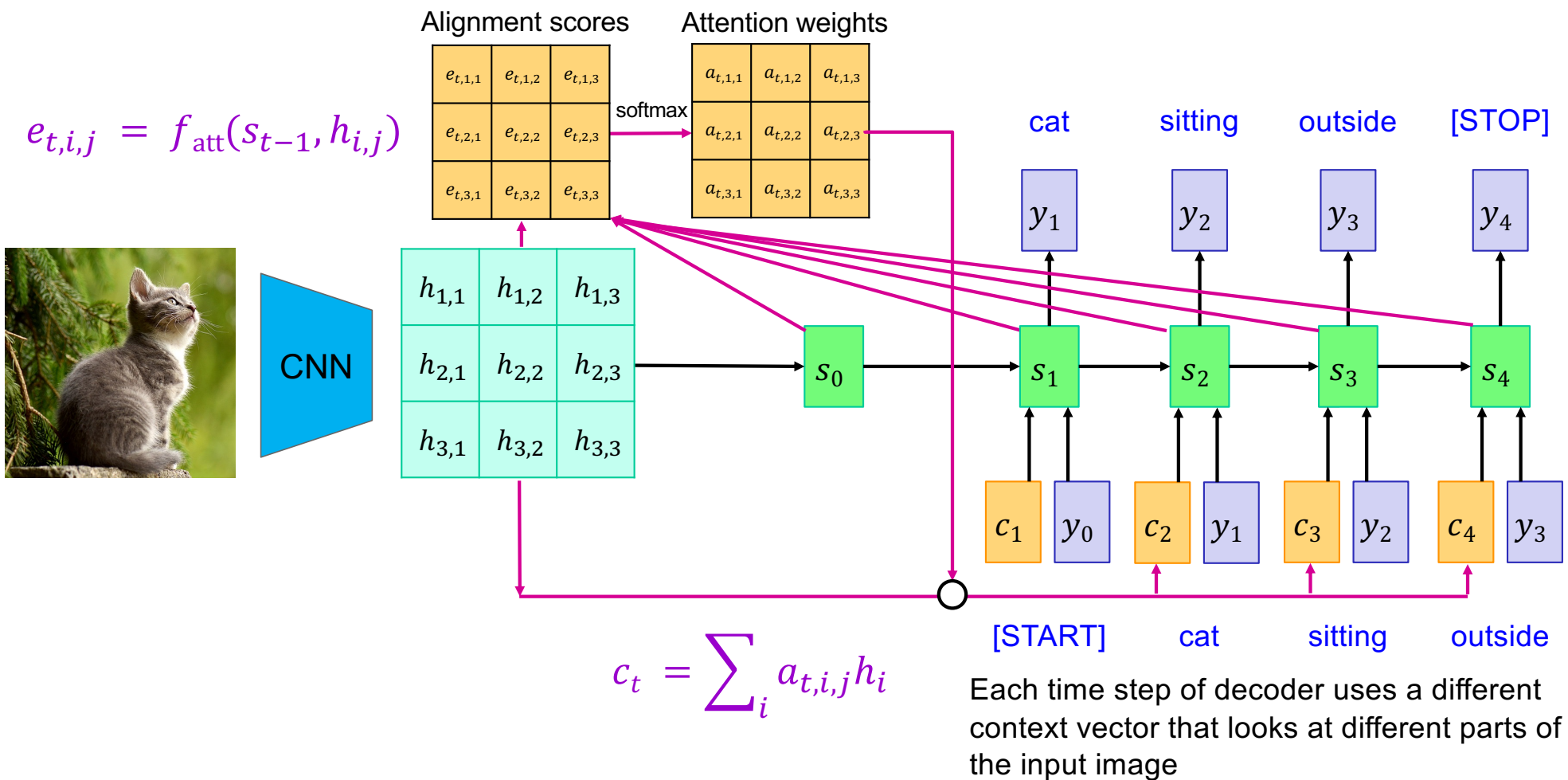
# Image captioning with RNNs and attention



# Image captioning with RNNs and attention



# Image captioning with RNNs and attention



# Example results

---

- Good captions



A woman is throwing a frisbee in a park.



A dog is standing on a hardwood floor.



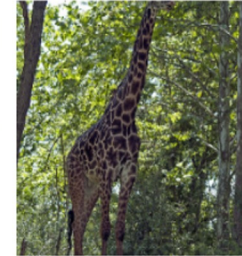
A stop sign is on a road with a mountain in the background.



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



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

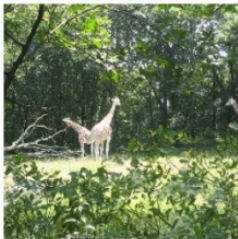


A giraffe standing in a forest with trees in the background.

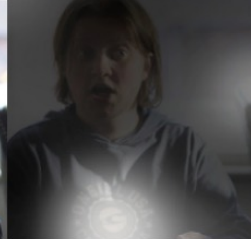
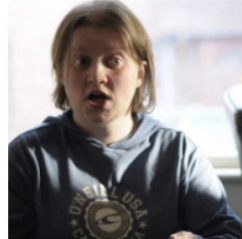
# Example results

---

- Mistakes



A large white bird standing in a forest.



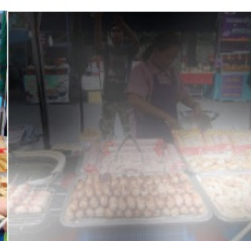
A woman holding a clock in her hand.



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.

# Quantitative results

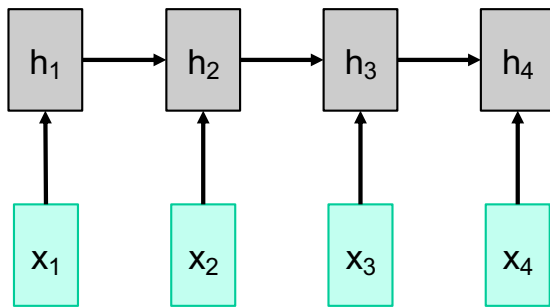
---

Dataset	Model	BLEU-1	BLEU-2	BLEU-3	BLEU-4	METEOR
Flickr8k	Google NIC	63	41	27	-	-
	Soft-Attention	<b>67</b>	44.8	29.9	19.5	18.93
	Hard-Attention	<b>67</b>	<b>45.7</b>	<b>31.4</b>	<b>21.3</b>	<b>20.30</b>
Flickr30k	Google NIC	66.3	42.3	27.7	18.3	-
	Soft-Attention	66.7	43.4	28.8	19.1	<b>18.49</b>
	Hard-Attention	<b>66.9</b>	<b>43.9</b>	<b>29.6</b>	<b>19.9</b>	18.46
COCO	Google NIC	66.6	46.1	32.9	24.6	-
	Soft-Attention	70.7	49.2	34.4	24.3	<b>23.90</b>
	Hard-Attention	<b>71.8</b>	<b>50.4</b>	<b>35.7</b>	<b>25.0</b>	23.04

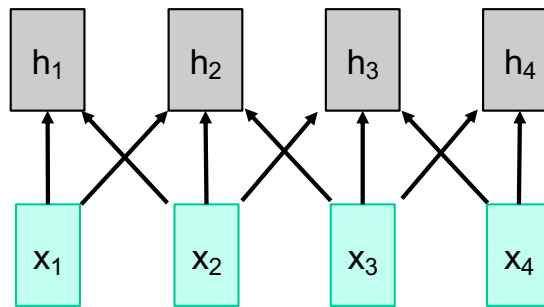
[Source](#)

# Sequence modeling beyond RNNs

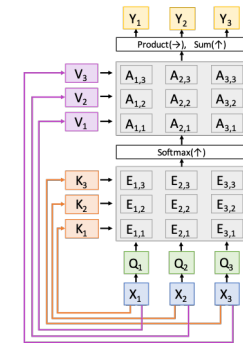
RNNs



1D convolutional networks



Transformers



Works on **ordered sequences**

- Pros: Not limited by fixed context size (in principle): After one RNN layer,  $h_T$  "sees" the whole sequence
- Con: Hidden states have limited expressive capacity
- Con: Not parallelizable: need to compute hidden states sequentially

Works on **multidimensional grids**

- Pro: Each output can be computed in parallel (at training time)
- Con: Bad at long sequences: Need to stack many conv layers for outputs to "see" the whole sequence

• Works on **sets of vectors**

# Outline

---

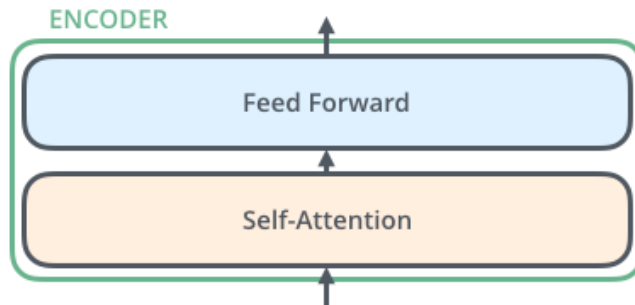
- Vanilla seq2seq with RNNs
- Seq2seq with RNNs and attention
- Image captioning with attention
- Transformers

# Basic transformer model

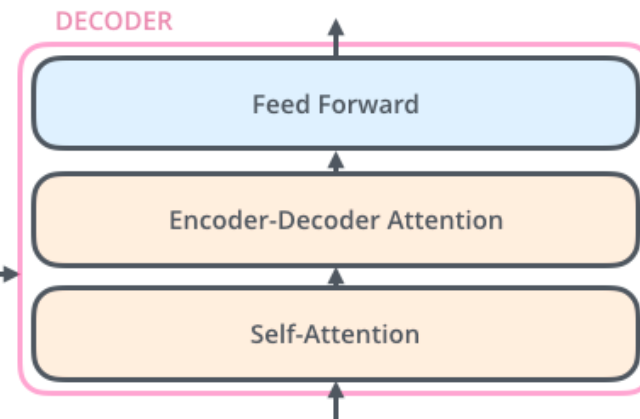
---

- Sequence-to-sequence architecture using *only point-wise processing and attention* – no recurrent units or convolutions

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



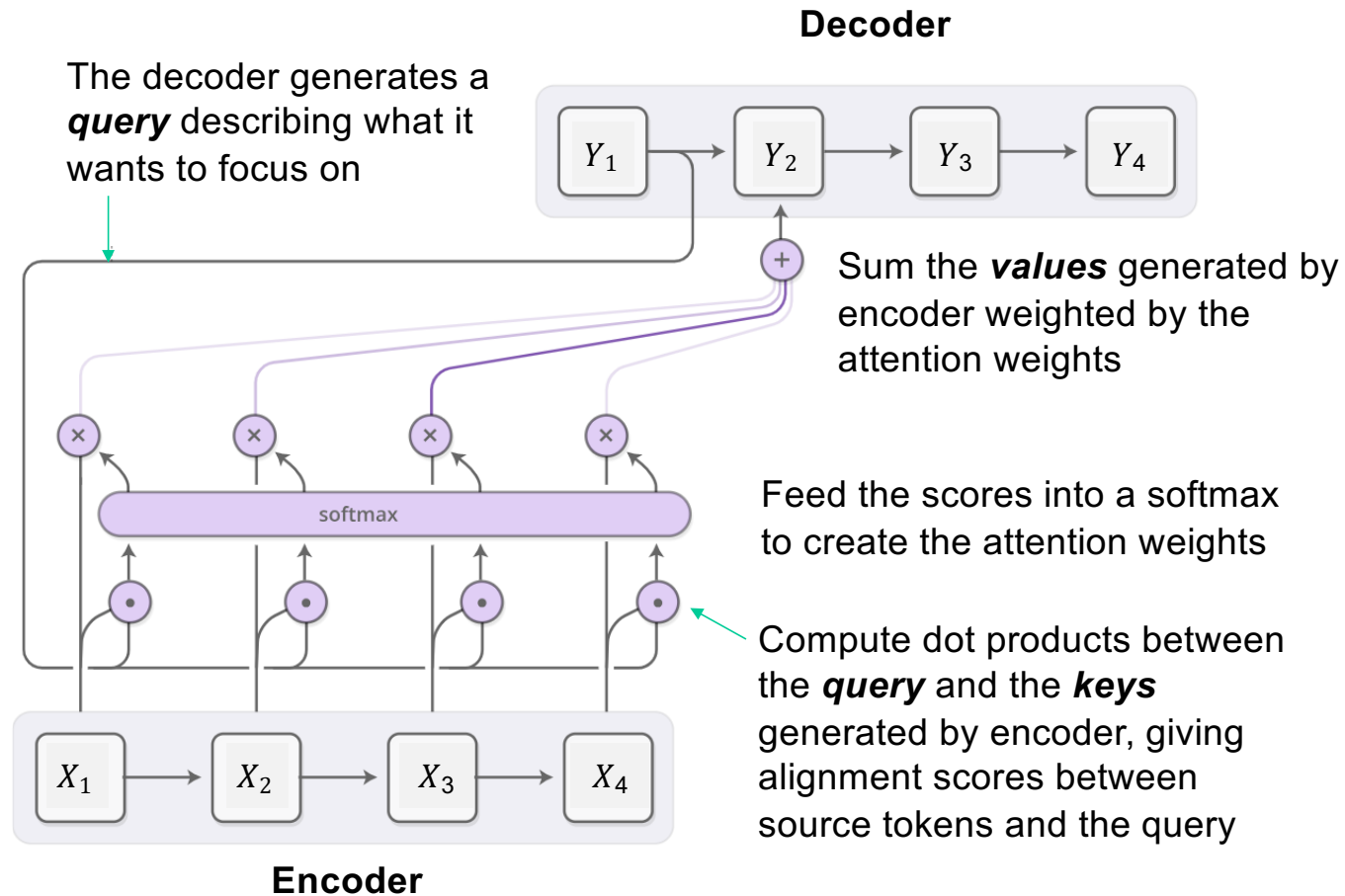
**Decoder:** predicts next token conditioned on encoder output and previously predicted tokens



A. Vaswani et al. [Attention is all you need](#). NeurIPS 2017

[Image source](#)

# Key-Value-Query attention model



# Key-Value-Query attention model

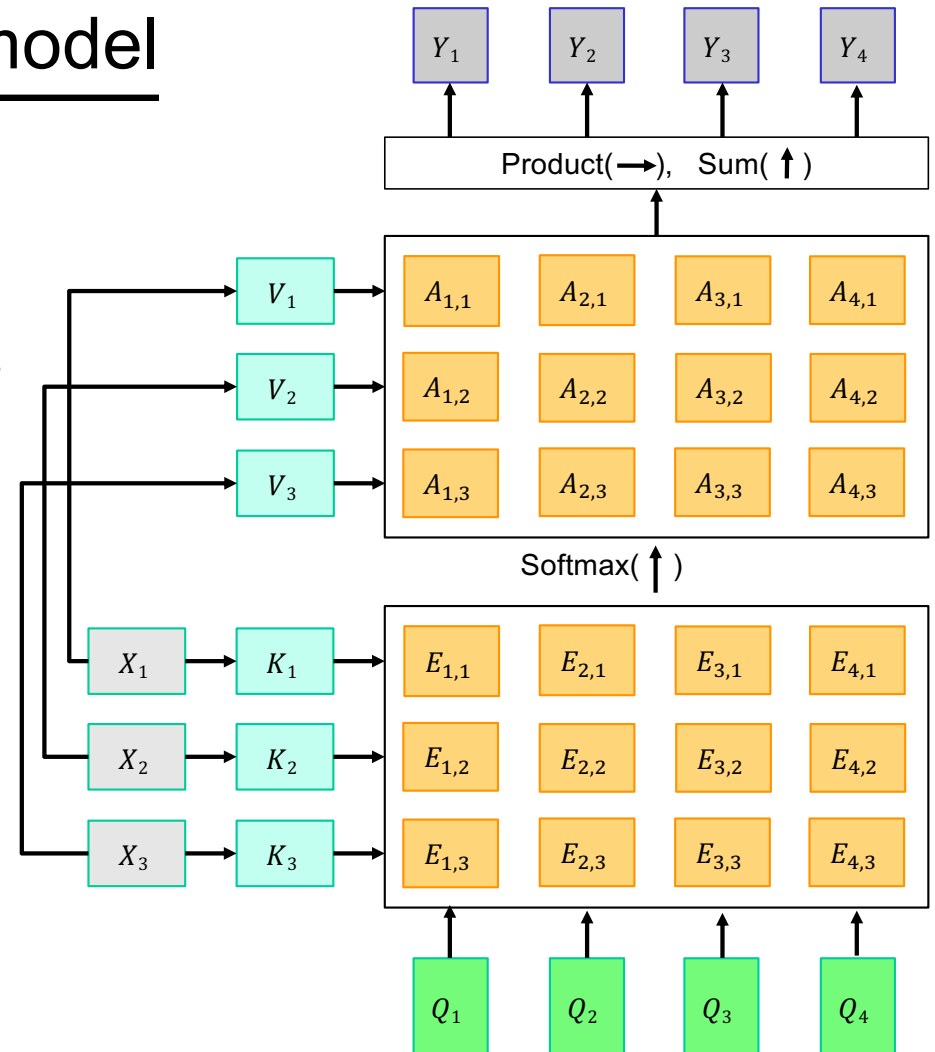
- Key vectors:  $K = XW_K$
- Value Vectors:  $V = XW_V$
- Query vectors
- Similarities: *scaled dot-product attention*

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

( $D$  is the dimensionality of the keys)

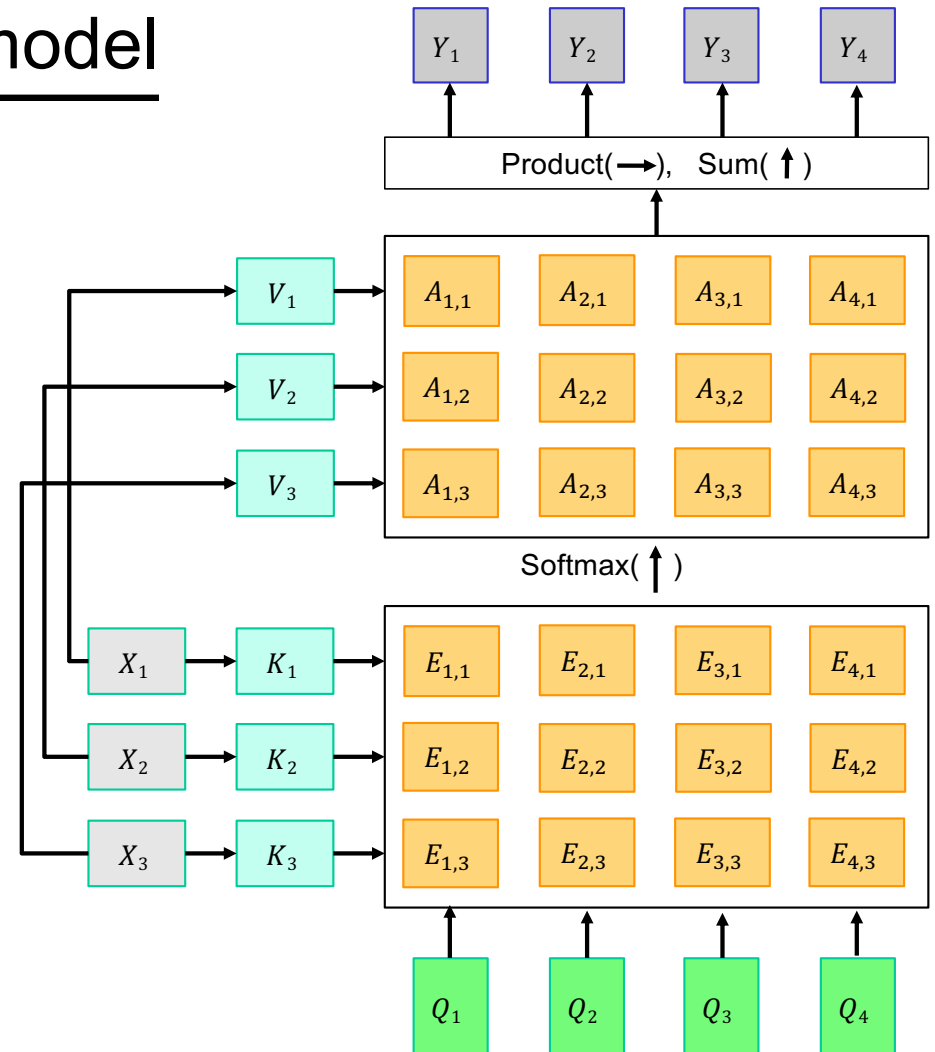
- Attn. weights:  $A = \text{softmax}(E, \text{dim} = 1)$
- Output vectors:

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



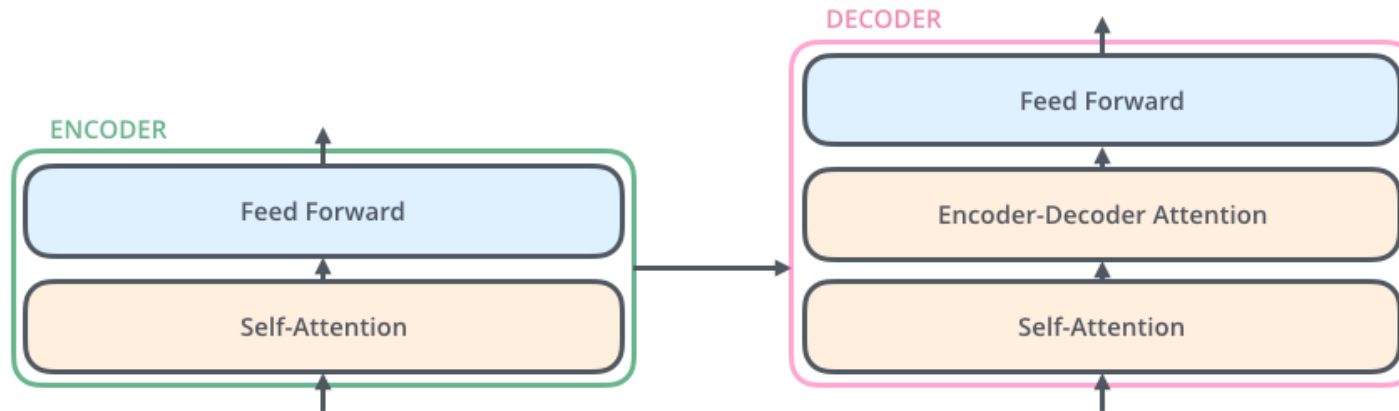
# Key-Value-Query attention model

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



# Attention mechanisms

---



- **Encoder self-attention:** queries, keys, and values come from previous layer of encoder
- **Decoder self-attention:** values corresponding to future decoder outputs are masked out
- **Encoder-decoder attention:** queries come from previous decoder layer, keys and values come from output of encoder

# Self-attention

---

- Used to capture context *within the sequence*

The animal didn't cross the street because it was too tired .

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

The animal didn't cross the street because it was too wide .

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

# Self-attention layer

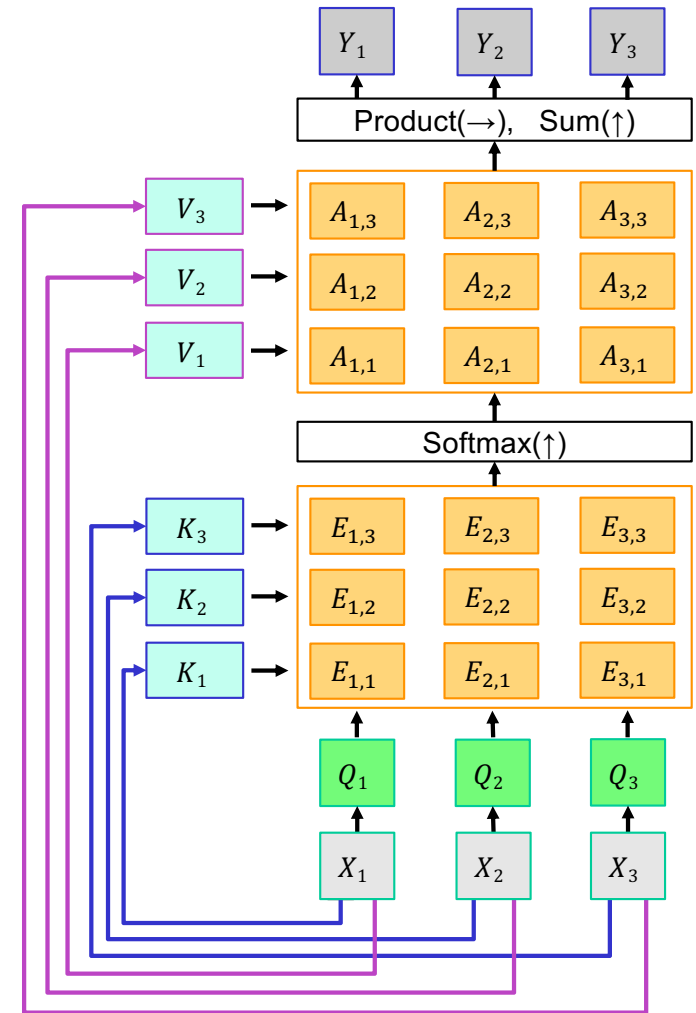
- Query vectors:  $Q = XW_Q$
- Key vectors:  $K = XW_K$
- Value vectors:  $V = XW_V$
- Similarities: *scaled dot-product attention*

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

( $D$  is the dimensionality of the keys)

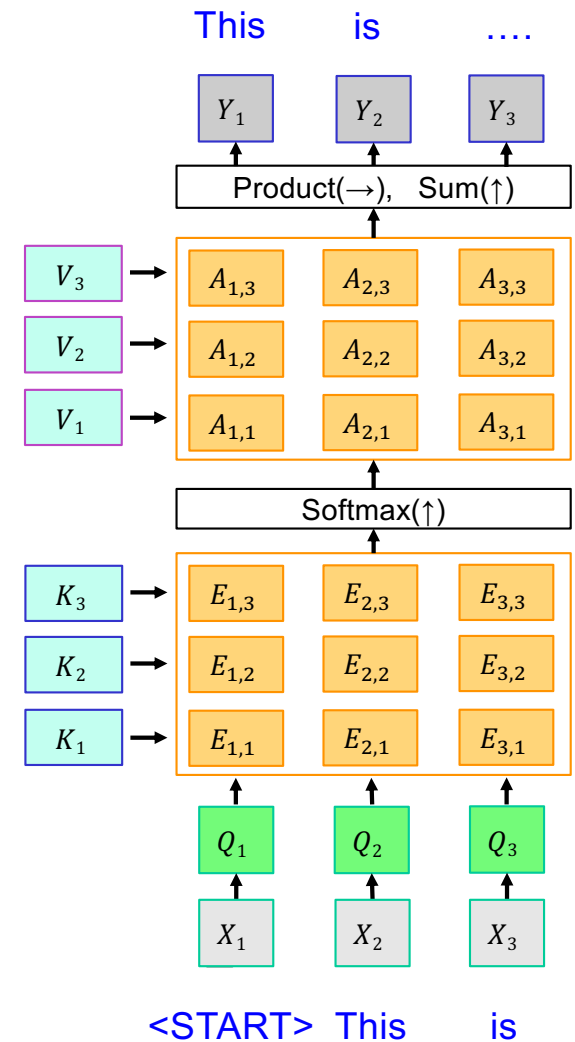
- Attn. weights:  $A = \text{softmax}(E, \text{dim} = 1)$
- Output vectors:

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



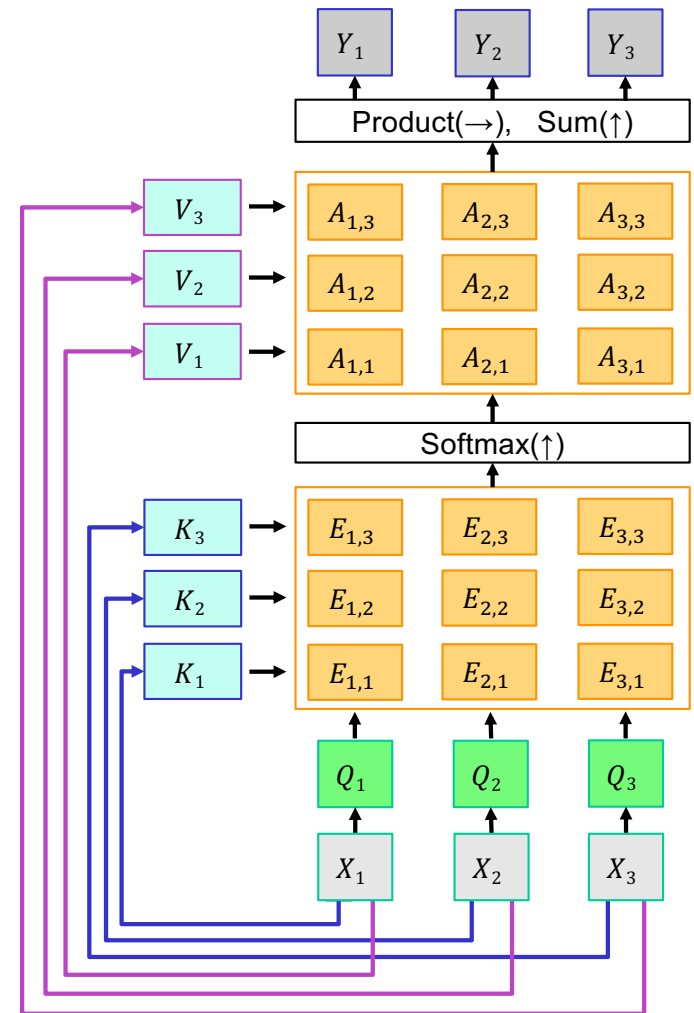
# Masked self-attention layer

- The decoder should not “look ahead” in the output sequence



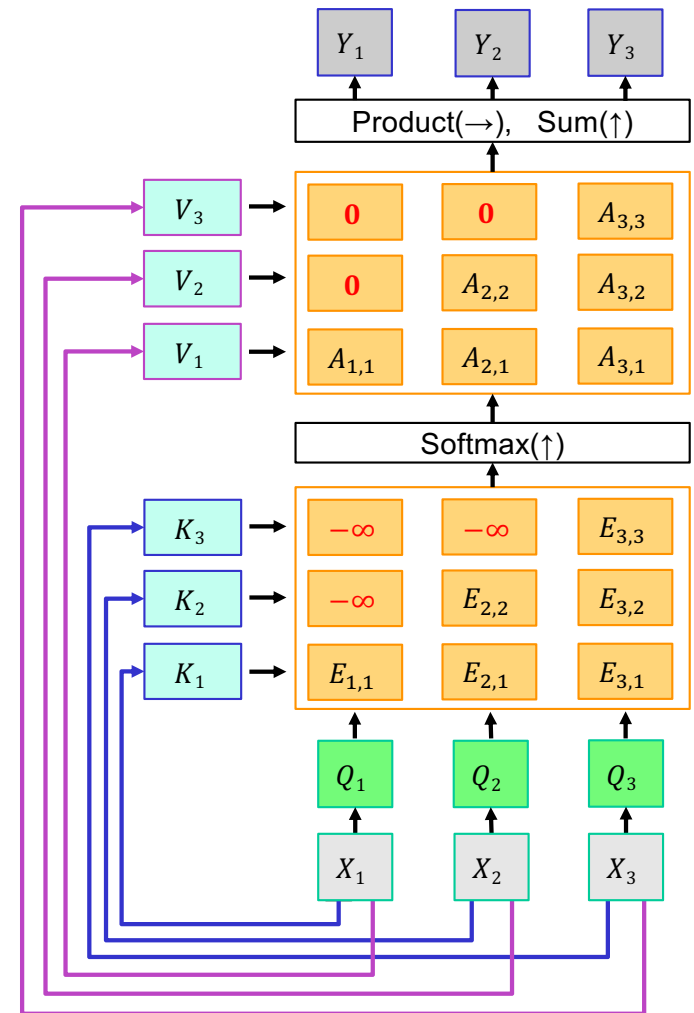
# Masked self-attention layer

- The decoder should not “look ahead” in the output sequence



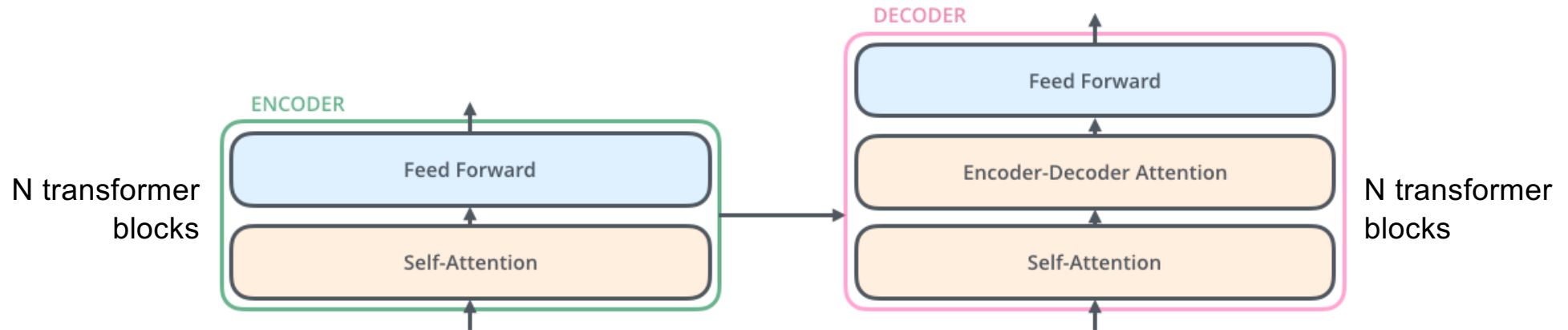
# Masked self-attention layer

- The decoder should not “look ahead” in the output sequence



# Attention mechanisms: Summary

---



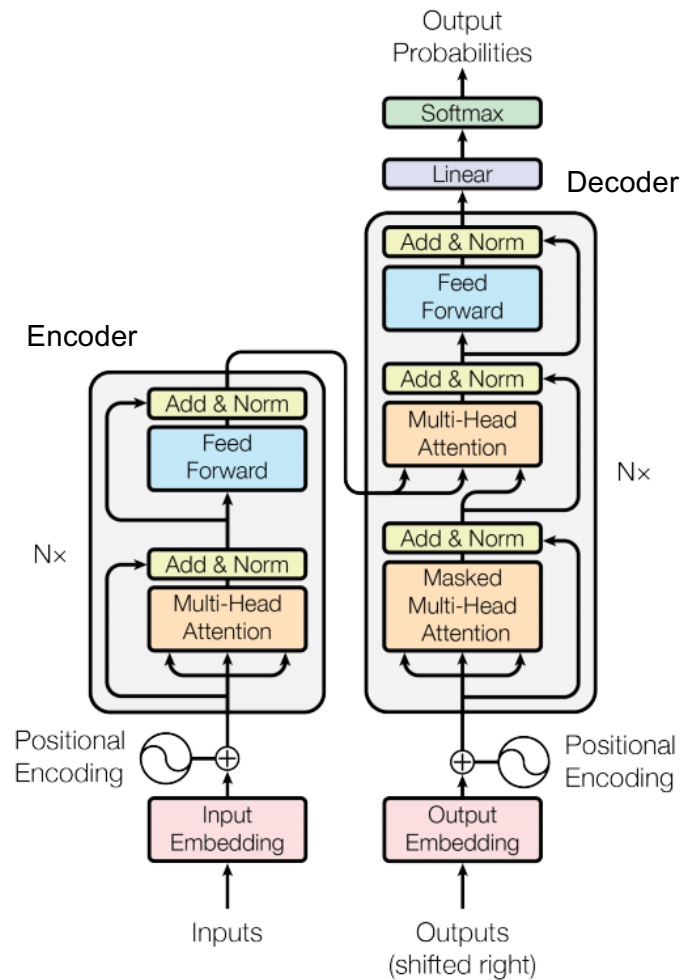
- **Encoder self-attention:** queries, keys, and values come from previous layer of encoder
- **Decoder self-attention:** values corresponding to future decoder outputs are masked out
- **Encoder-decoder attention:** queries come from previous decoder layer, keys and values come from output of encoder

# Attention mechanisms: Illustration

---

<https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html>

# Transformer architecture: Details

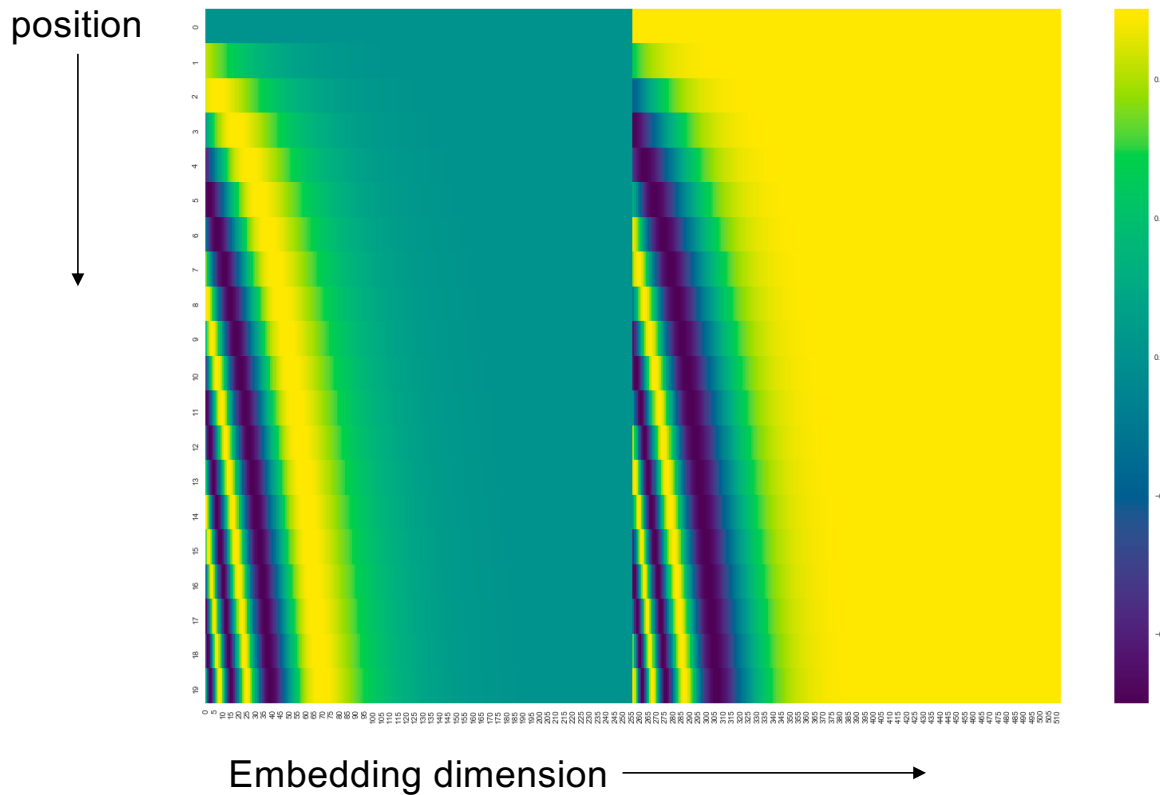


A. Vaswani et al., [Attention is all you need](#), NeurIPS 2017

# Positional encoding

---

- To give transformer information about ordering of tokens, add function of position (based on sines and cosines) to every input

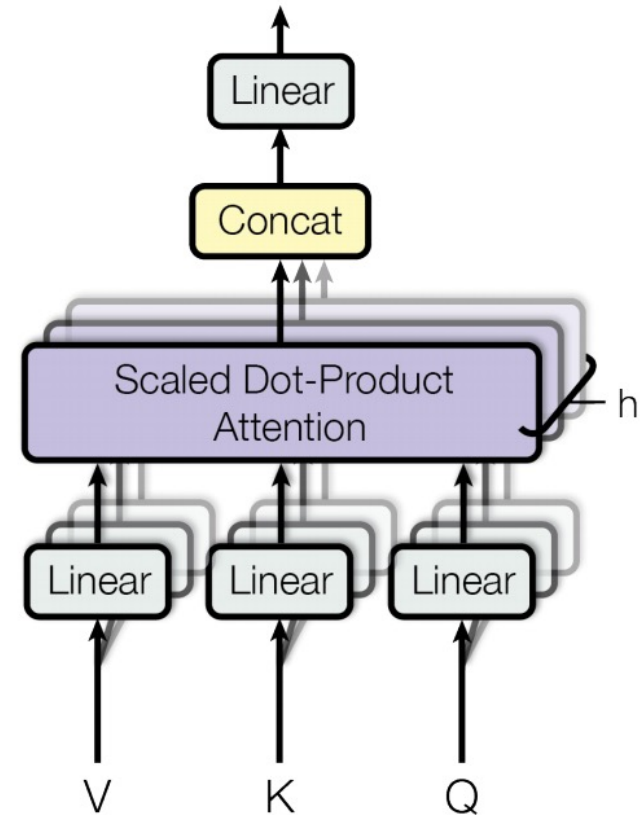


[Image source](#)

# Multi-head attention

---

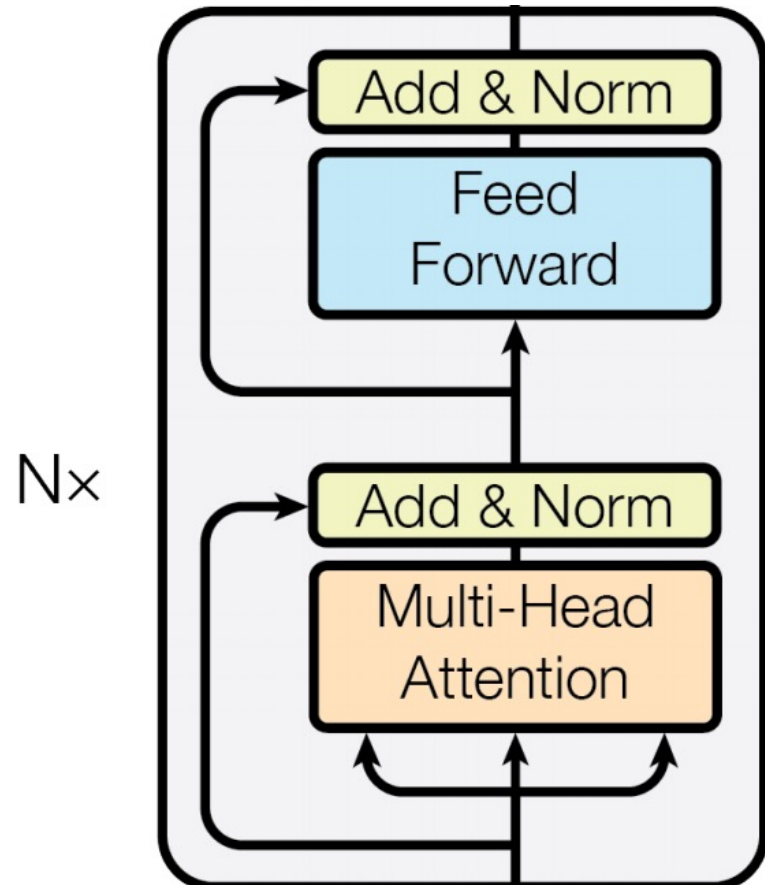
- Run  $h$  attention models in parallel on top of different linearly projected versions of  $Q, K, V$ ; concatenate and linearly project the results
- Intuition: enables model to attend to different kinds of information at different positions (see [visualization tool](#))



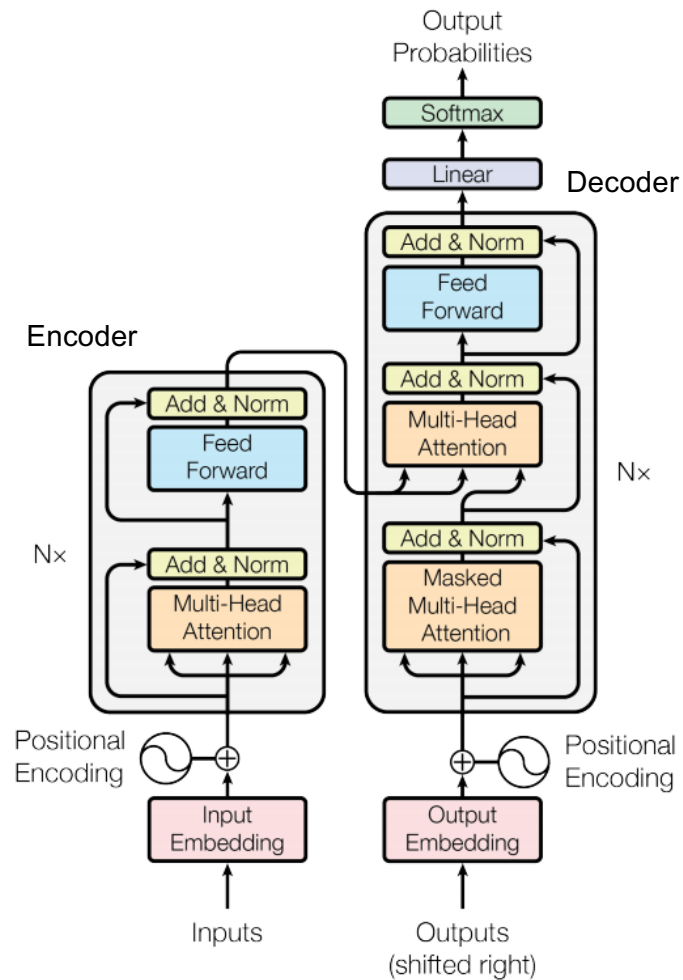
# Transformer blocks

---

- 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!



# Transformer architecture: Zooming back out

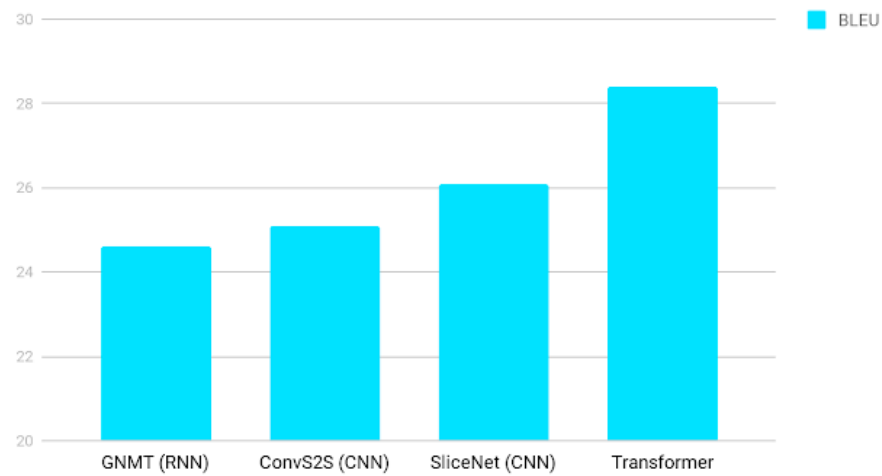


A. Vaswani et al., [Attention is all you need](#), NeurIPS 2017

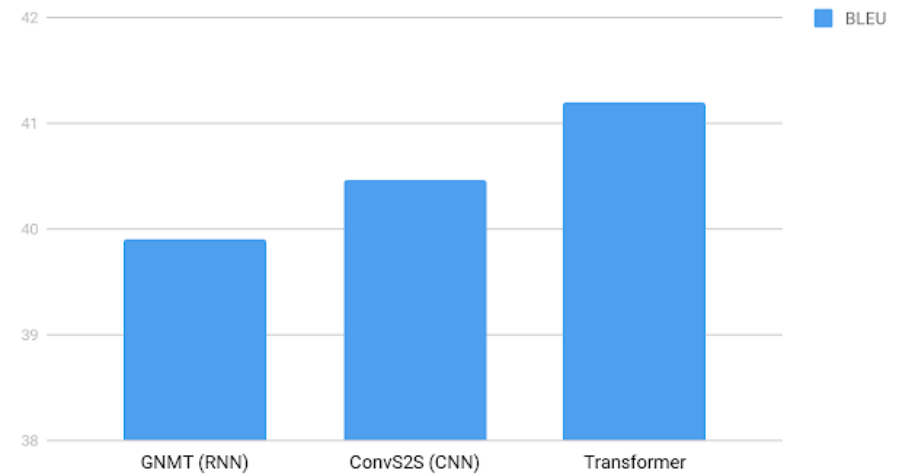
# Results

---

English German Translation quality



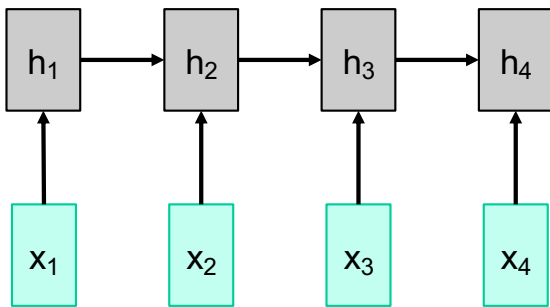
English French Translation Quality



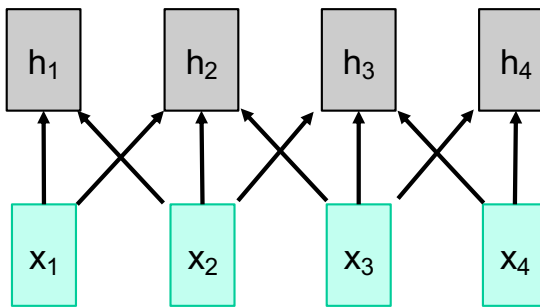
<https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html>

# Transformers: Pros and cons

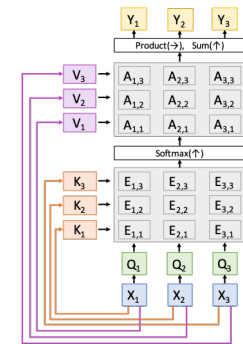
## RNNs



## 1D convolutional networks



## Transformers



### Works on **ordered sequences**

- Pros: Not limited by fixed context size (in principle): After one RNN layer,  $h_T$  "sees" the whole sequence
- Con: Not parallelizable: need to compute hidden states sequentially
- Con: Hidden states have limited expressive capacity

### Works on **multidimensional grids**

- Pro: Each output can be computed in parallel (at training time)
- Con: Need to stack many conv layers for outputs to "see" the whole sequence

### Works on **sets of vectors**

- Pro: Good at long sequences: after one self-attention layer, each output "sees" all inputs!
- Pro: Each output can be computed in parallel (at training time)
- Con: Memory-intensive: cost of attention operator is *quadratic* in input size