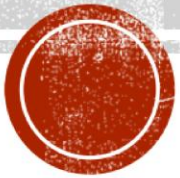
SELF-ATTENTION AND TRANSFORMER

Localizing the Focus



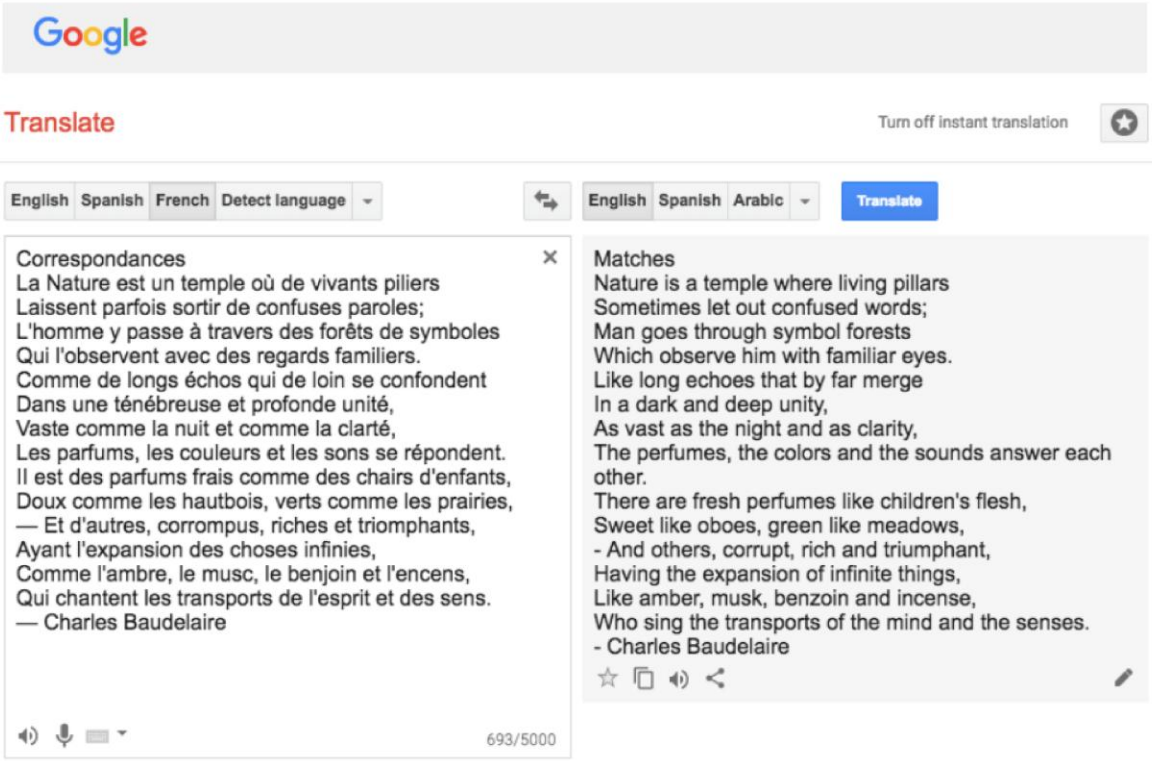
DISCLAIMER

The content (text, image, and graphics) used in this slide are adopted from many sources for academic purposes. Broadly, the sources have been given due credit appropriately. However, there is a chance of missing out some original primary sources. The authors of this material do not claim any copyright of such material.

Outline

- Recurrent network architecture
- Sequence-to-sequence models with attention
- Self-attention
- Transformer Network

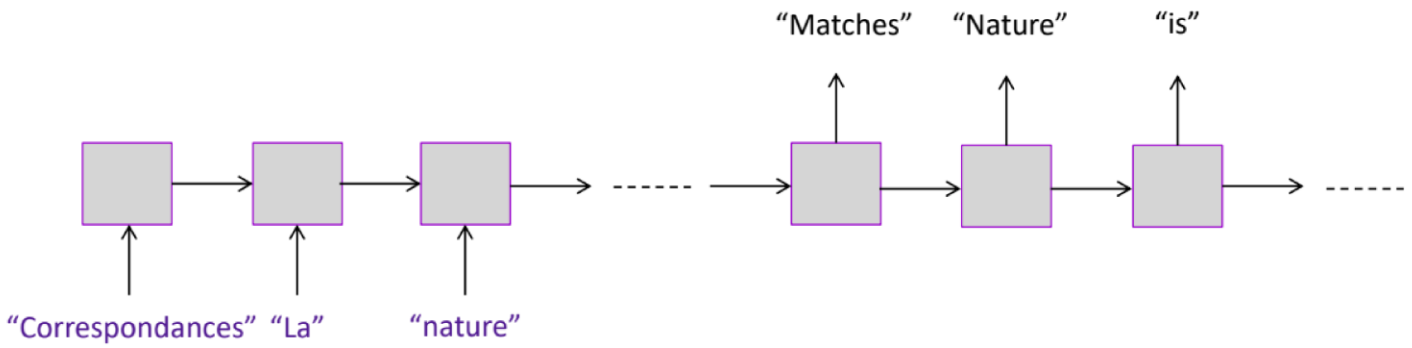
Example: Machine translation



https://translate.google.com/

Example: Machine translation

- Multiple input – multiple output (or sequence to sequence) scenario:



Example: Image caption generation

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

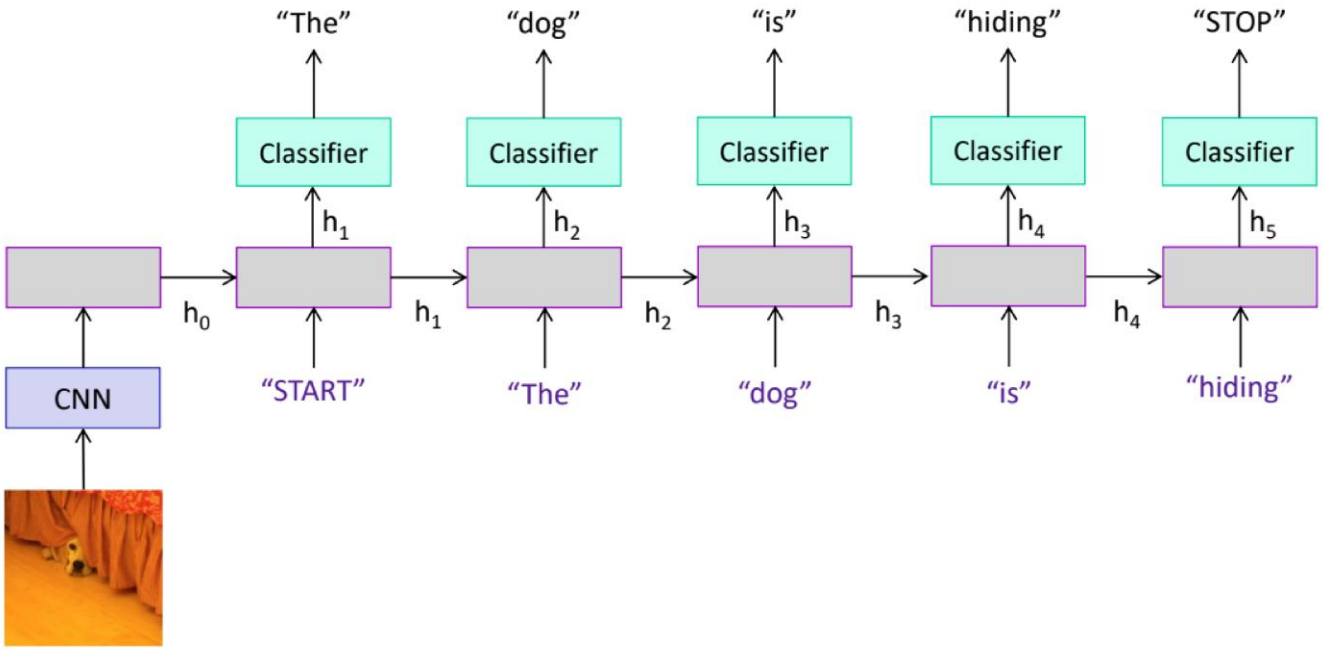
A tennis player in action on the court

Two giraffes standing in a grassy field

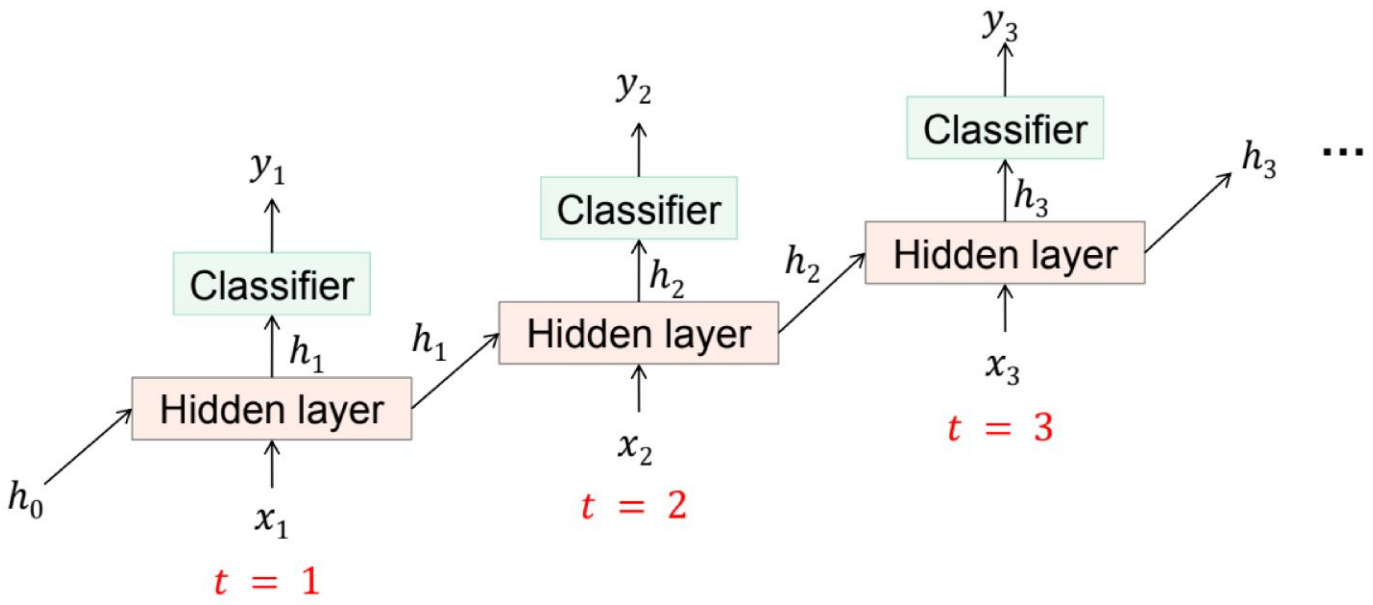
A man riding a dirt bike on a dirt track

Source: J. Johnson
Captions generated using neuraltalk2

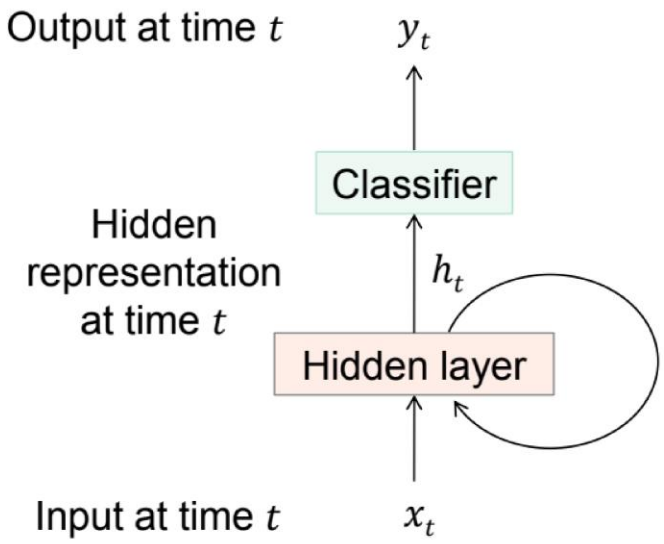
Example: Image caption generation



Recurrent unit



Recurrent unit

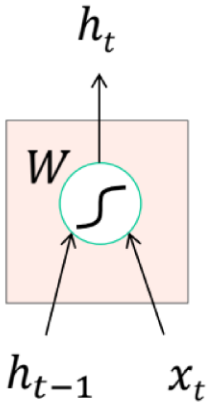


Recurrence:

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

new state | function of W | input at time t | old state

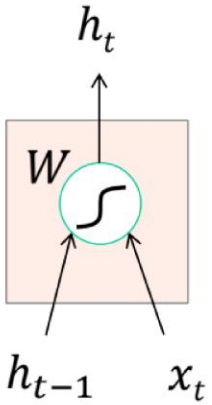
Vanilla RNN cell



$$h_t = f_W(x_t, h_{t-1})$$
$$= \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

J. Elman, Finding structure in time, Cognitive science 14(2), pp. 179–211, 1990

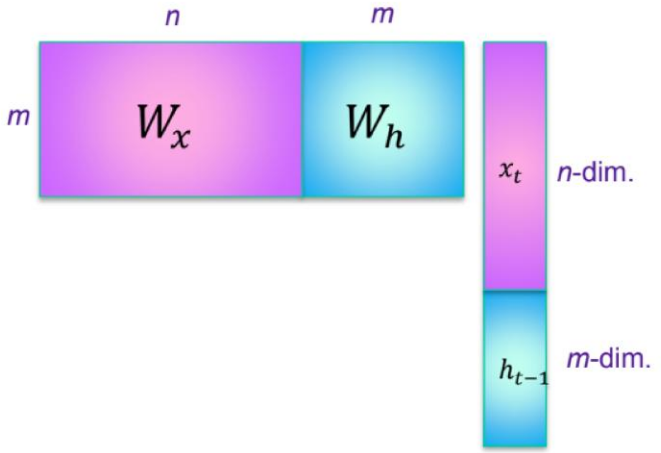
Vanilla RNN cell



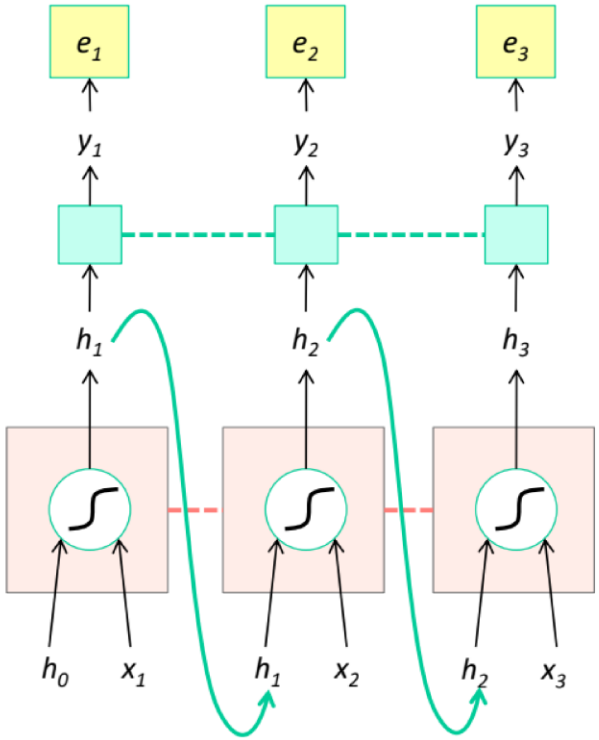
$$h_t = f_W(x_t, h_{t-1})$$

$$= \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

$$= \tanh(W_x x_t + W_h h_{t-1})$$



RNN forward pass



$$e_t = -\log(y_t(GT_t))$$

$$y_t = \text{softmax}(W_y h_t)$$

$$h_t = \tanh W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

------ shared weights

Image caption generation: Training time

- Minimize negative log-likelihood of the ground truth caption $Y^* = (Y_1^*, \ldots, Y_N^*)$ given image I:

$$L(I, Y^*) = -\sum_{t=1}^{N} \log P_W(Y_i^* | Y_1^*, \ldots, Y_{i-1}^*, I)$$

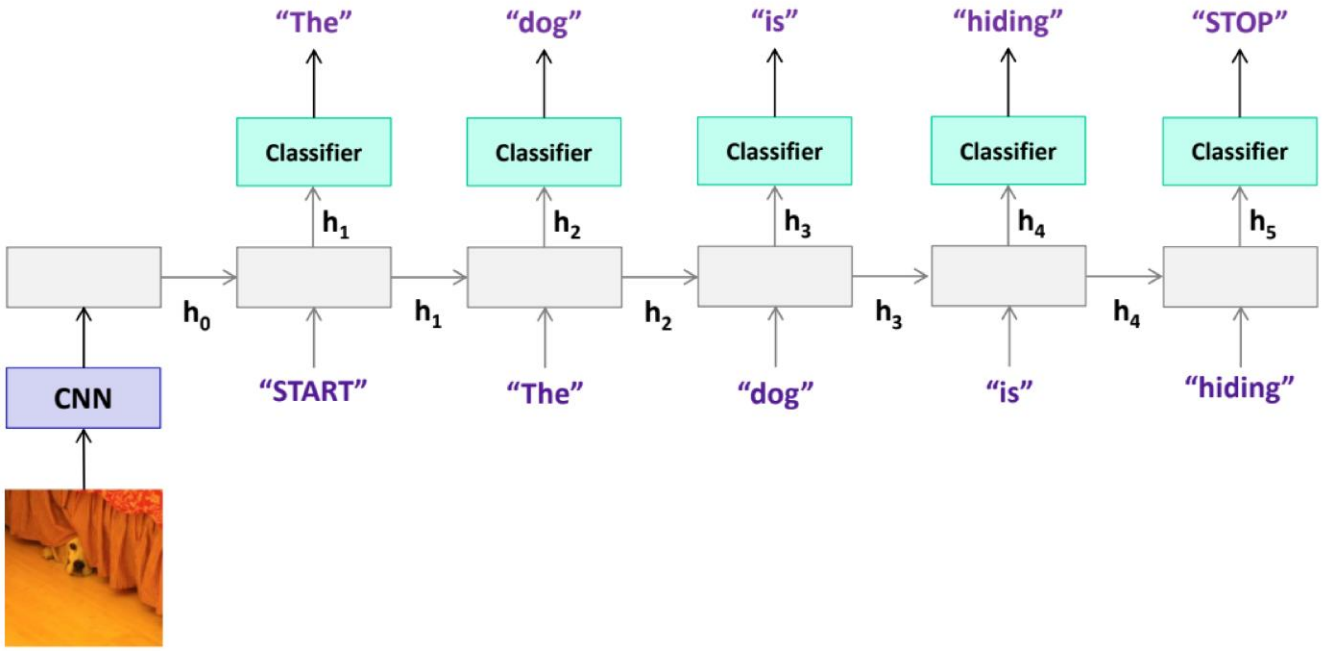
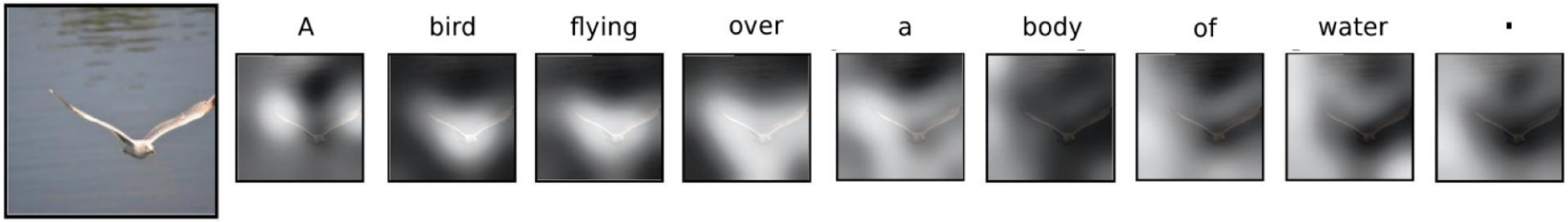


Image caption generation: Example outputs

Image captioning with RNNs and attention

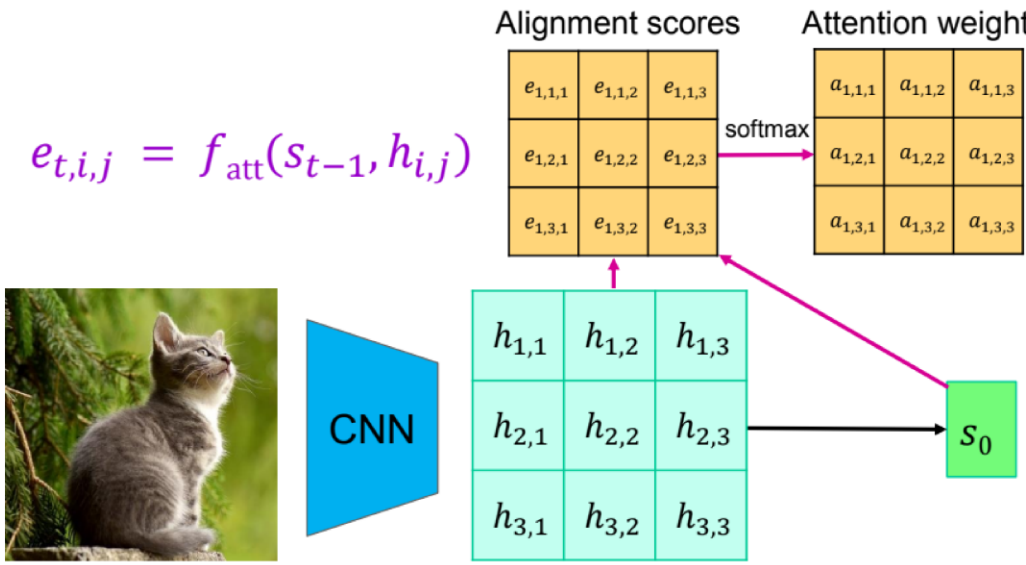
- Idea: pay attention to different parts of the image when generating different words
- Automatically learn this grounding of words to image regions without direct supervision



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

Image captioning with RNNs and attention

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



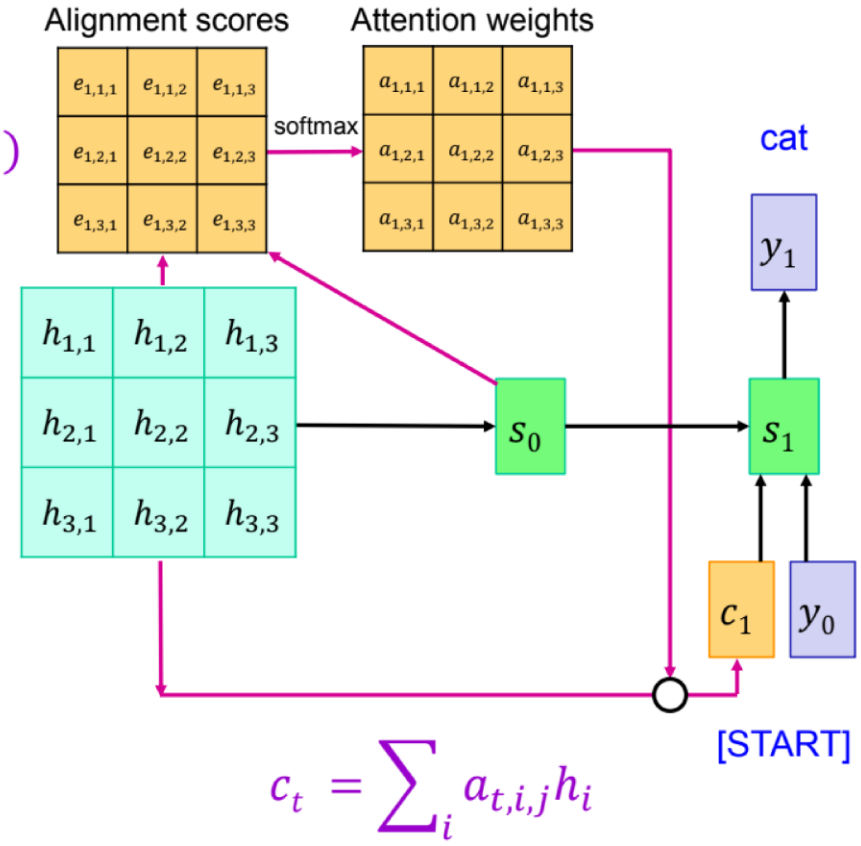
Use a CNN to compute a grid of features for an image

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



$$c_t = \sum_i a_{t,i,j} h_i$$

Image captioning with RNNs and attention

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

Image captioning with RNNs and attention

CNN

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

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

Alignment scores

Attention weights

softmax

$c_t = \sum_i a_{t,i,j} h_i$

cat sitting outside [STOP]

[START] cat sitting outside

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.

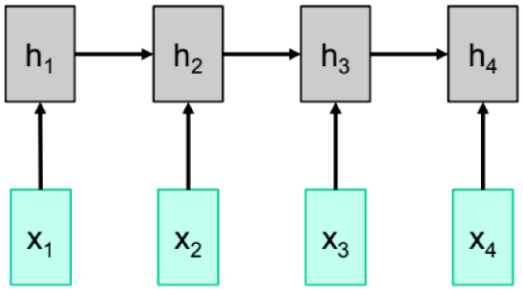
A person is standing on a beach with a surfboard.

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

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

Different ways of processing sequences

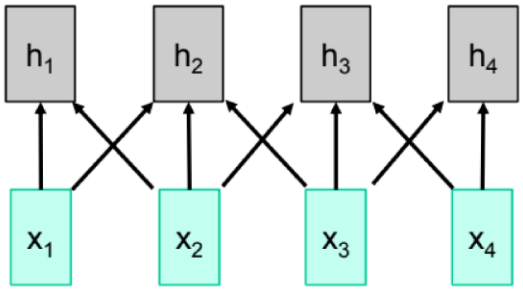
RNN



Works on **ordered sequences**

- Pros: Good at long sequences: the last hidden vector encapsulates the whole sequence
- Cons: Not parallelizable: need to compute hidden states sequentially

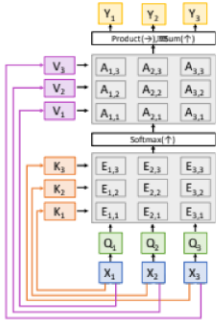
1D convolutional network



Works on **multidimensional grids**

- Con: Bad at long sequences: Need to stack many conv layers for outputs to “see” the whole sequence
- Pro: Highly parallel: Each output can be computed in parallel

Self-Attention and Transformer



- Works on **sets of vectors**
- Pro: Good at long sequences: after one self-attention layer, each output “sees” all inputs!
- Pro: Highly parallel: Each output can be computed in parallel
- Con: Very memory-intensive

Natural language processing

Problems that can be solved by NLP

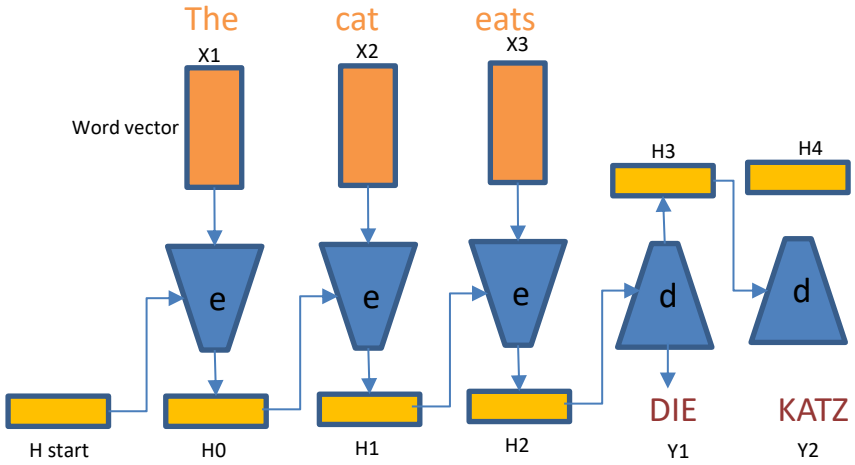
:Sentence classification
:Sentence to sentence
:Language conversion

Methods

:RNN,

:LSTM,

:GRN



Problems in Rnn

This sequential nature.
reduce the connectivity of hidden states to original inputs.
Not effective when sequence length is long.
Prevents parallelization within training samples
Require Attention Mechanism for remembering the focused area

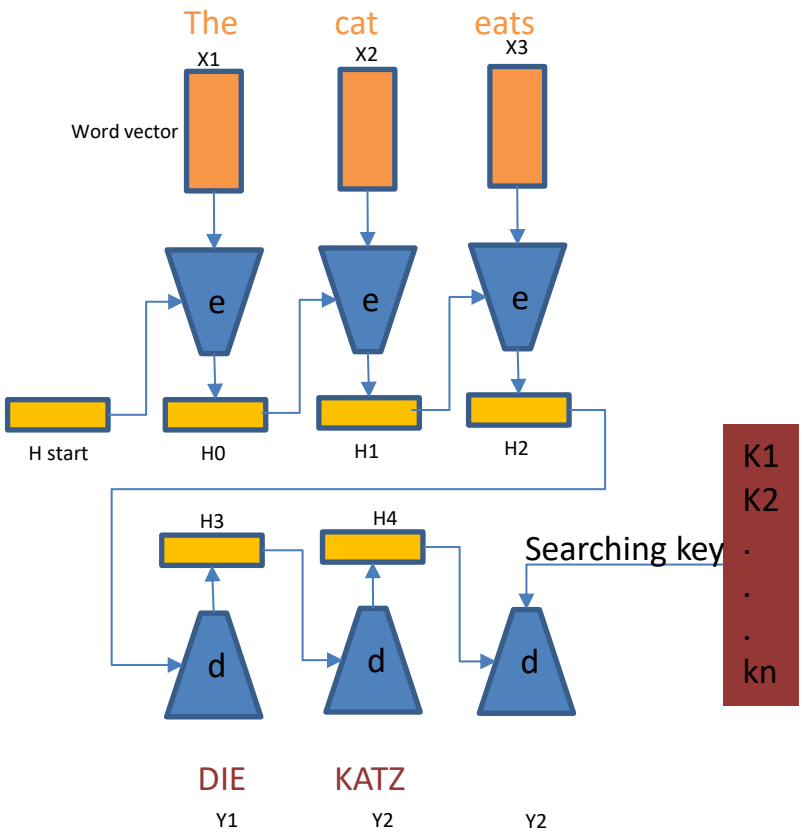
Problems in CNN

:DON'T ALLOW THE TIME SERIES CONTEXT TO FLOW.
:CAN PERFORM ONE TO ONE OUTPUT

Summary:

Need attention mechanisms allow us to draw global dependencies between input and output by a constant number of operations

Attention:



What is Attention?

Attention Weight

- $Attention(query, Key, Value) = Softmax(query \cdot Key^T) \cdot Value$

Search keys which are similar to a query, and return the corresponding values.

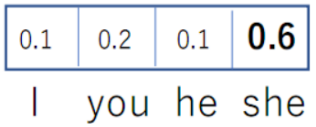
- Single query Attention : (vector, matrix, matrix) →vector

  $Attention(query, Key, Value) = Softmax(query \cdot Key^T) \cdot Value$

- We can input multiple queries at one time :
  (matrix, matrix, matrix) →matrix

  $Attention(Query, Key, Value) = Softmax(Query \cdot Key^T) \cdot Value$

When selecting model output, we can take the word with the highest probability and throw away the rest word candidates. : greedy decoding



Another way to select model output is beam-search.

beam-search

Instead of only predicting the token with the best score, we keep track of k hypotheses (for example k=4, we refer to k as the beam size).

Experiment- sequence to sequence task

- Data
  - WMT2014 English-German : 4.5 million sentence pairs
  - WMT2014 English-French : 36 million sentences
- Hardware and Schedule
  - 8 NVIDIA P100 GPUs
  - Base model : 100,000 steps or 12 hours
  - Big model : 300,000 steps (3.5 days)
- Optimizer : Adam
  - Warm up, and then decrease learning rate.

Experiment – evaluation metrics

- **BLEU** (Bilingual Evaluation Understudy)
  - Evaluation metrics on how machine translation(MT) and ground truth(GT) translation are similar.

Result

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	

Experiment – Changing the parameters in Transformer

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)	2									6.11	23.7	36
	4									5.19	25.3	50
	8									4.88	25.5	80
		256			32	32				5.75	24.5	28
		1024			128	128				4.66	26.0	168
			1024							5.12	25.4	53
			4096							4.75	26.2	90
(D)							0.0			5.77	24.6	
							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)	positional embedding instead of sinusoids									4.92	25.7	
big	6	1024	4096	16			0.3		300K	**4.33**	**26.4**	213

Key-Value-Query attention model

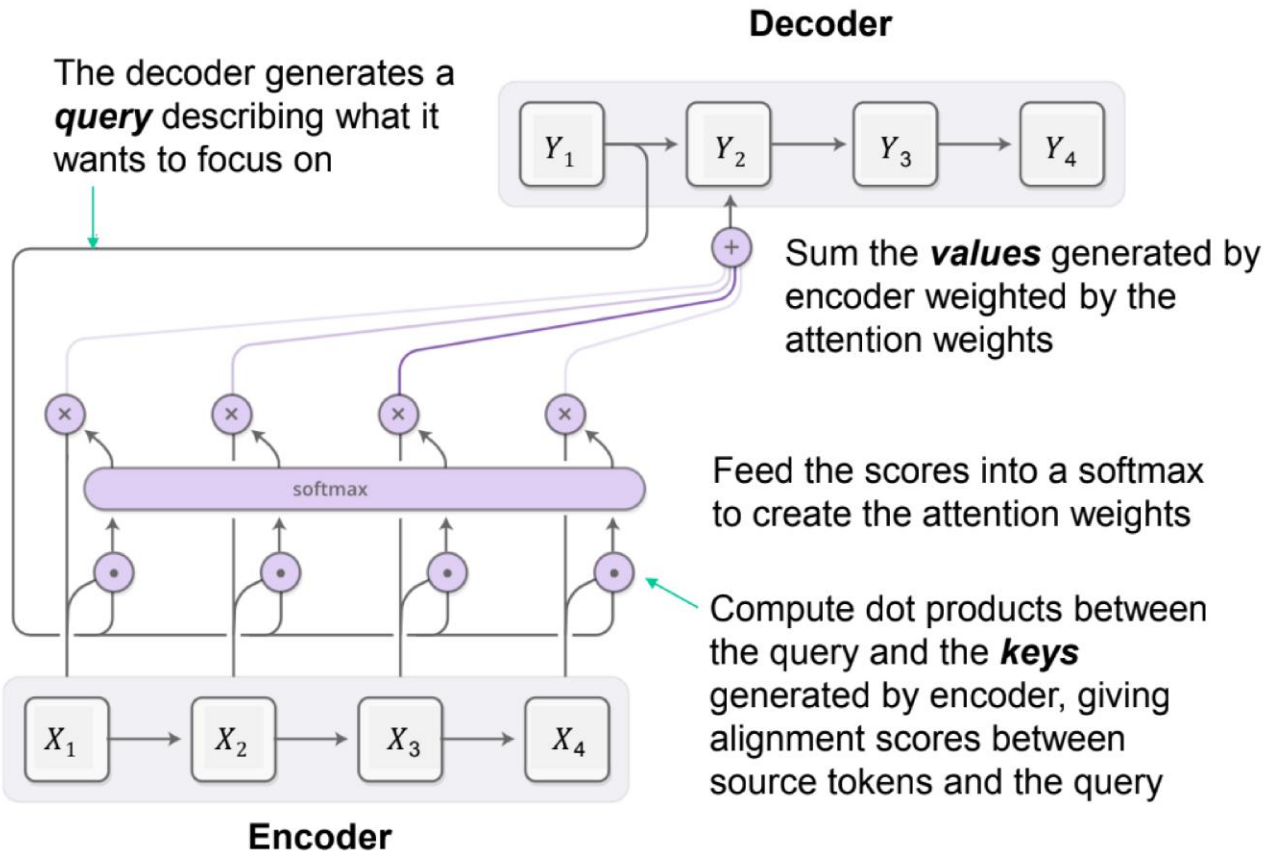


Image source

Key-Value-Query attention model

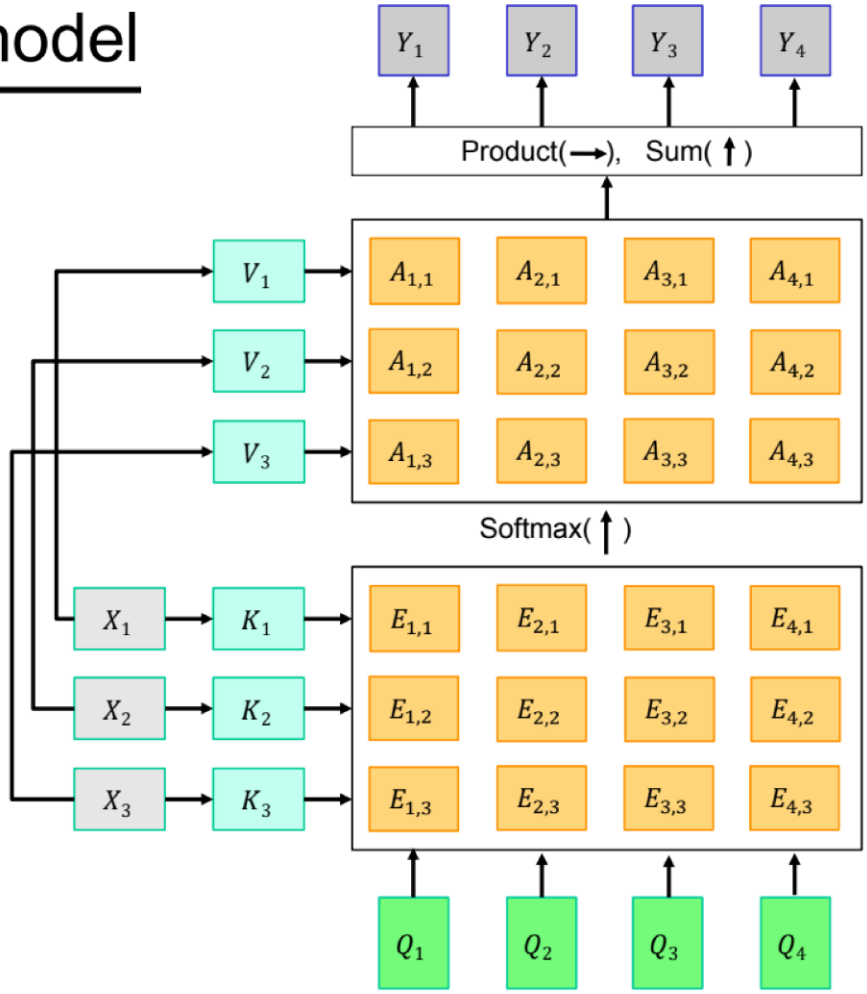
- Key vectors: $K = XW_K$
- Value Vectors: $V = XW_V$
- Query vectors
- Similarities: *scaled dot-product attention*

$$E_{i,j} = \frac{(Q_i \cdot K_j)}{\sqrt{D}}$$

(D is the dimensionality of the keys)

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

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



Self-attention layer

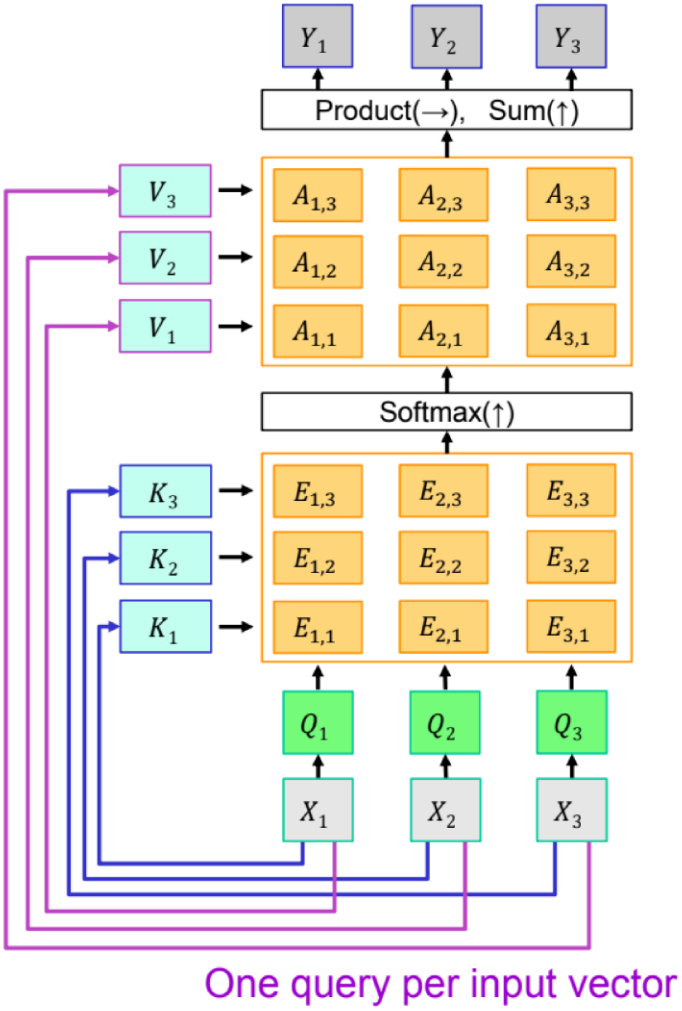
- Query vectors: $Q = XW_Q$
- Key vectors: $K = XW_K$
- Value vectors: $V = XW_V$
- Similarities: *scaled dot-product attention*

$$E_{i,j} = \frac{(Q_i \cdot K_j)}{\sqrt{D}}$$

(D is the dimensionality of the keys)

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

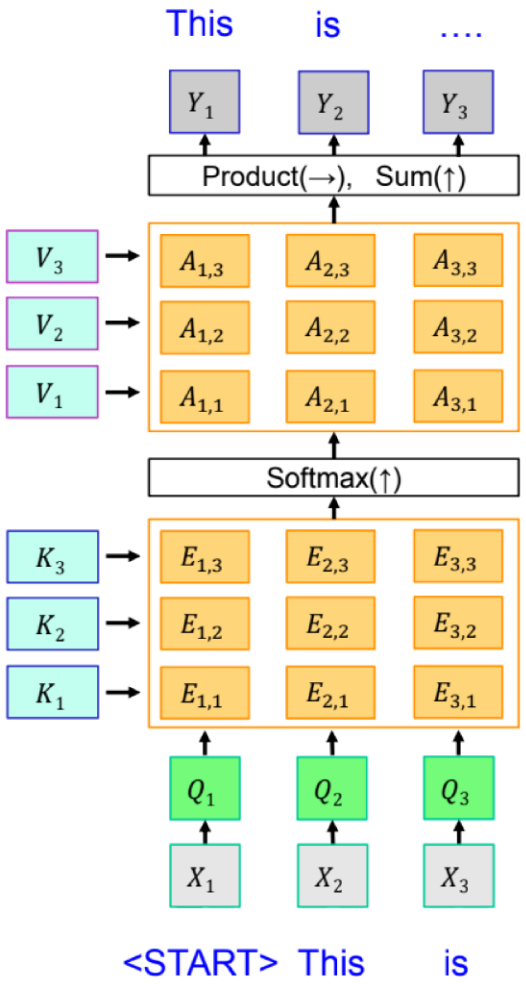
$$Y_i = \sum_j A_{i,j} V_j \text{ or } Y = AV$$



One query per input vector

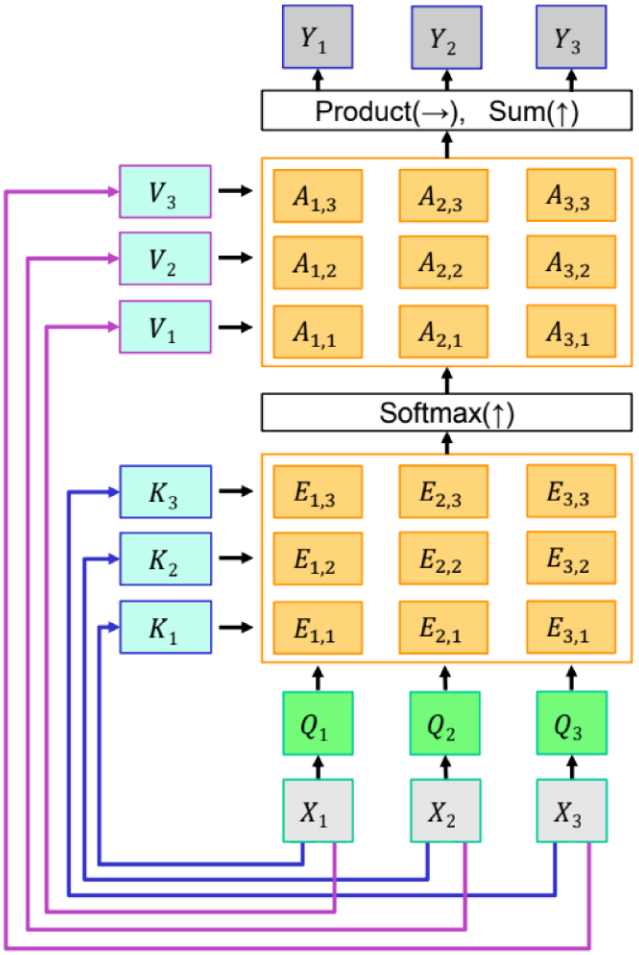
Masked self-attention layer

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



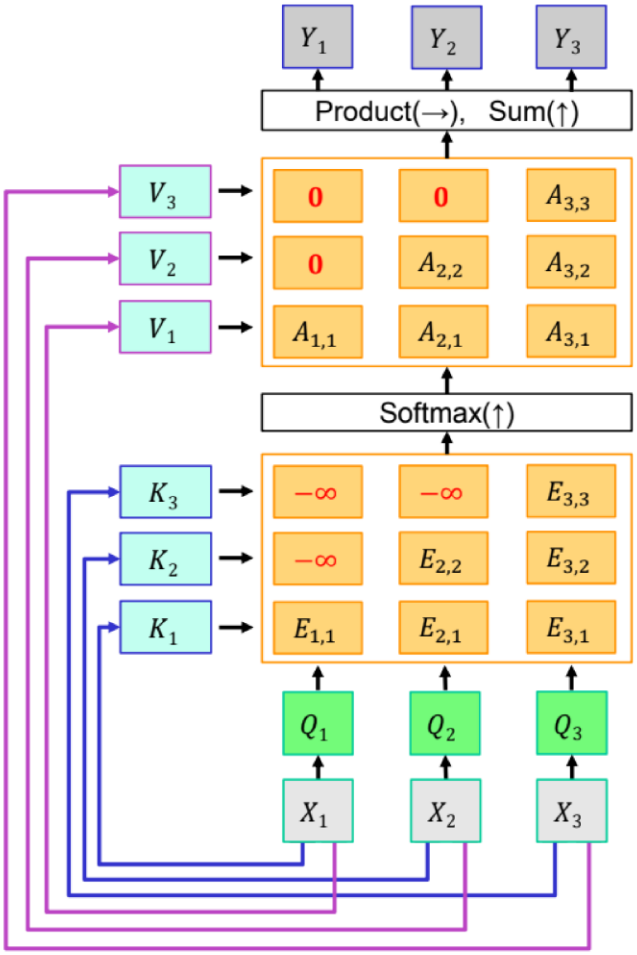
Masked self-attention layer

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



Masked self-attention layer

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



Proposed Transformer Architecture

Input: Sequence of symbol representations (x1, x2,…, xn)
Output: Sequence of symbol representations (y1, y2,…, yn)

- Left side : Encoder
- Right side : Decoder

- Consisting layers:
  - **Multi-Head Attention layer**
  - **Position-wise Feed-Forward layer**
  - **Positional Encoding**
  - (Residual Adding and Normalization layer)