

Deep learning

Transformers

2024/12/16

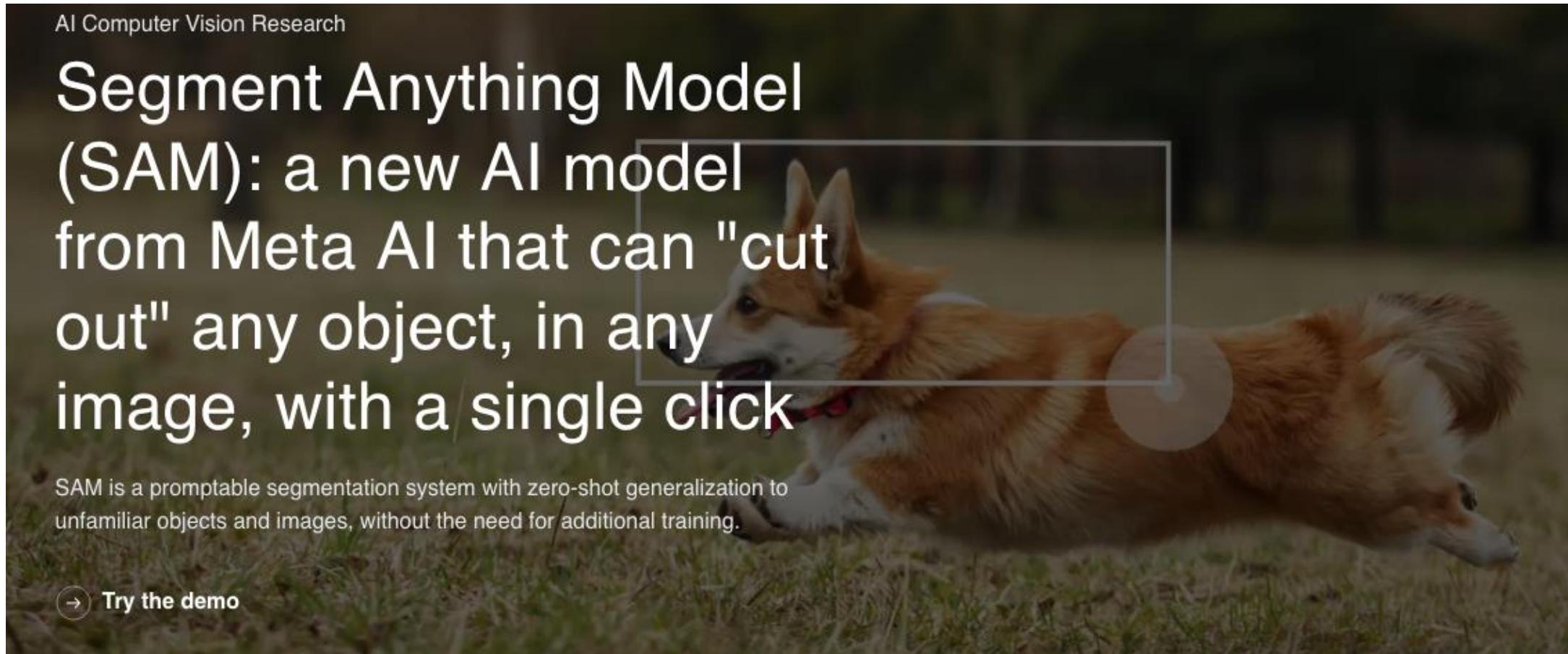
Jon Sporring,
Department of Computer Science

UNIVERSITY OF COPENHAGEN



Segment Anything

<https://segment-anything.com/>

A photograph of a Corgi dog running through a grassy field. A white rectangular segmentation mask is overlaid on the dog's body, indicating the area that can be cut out by the AI model. A circular cursor icon is positioned at the bottom right corner of the mask.

AI Computer Vision Research

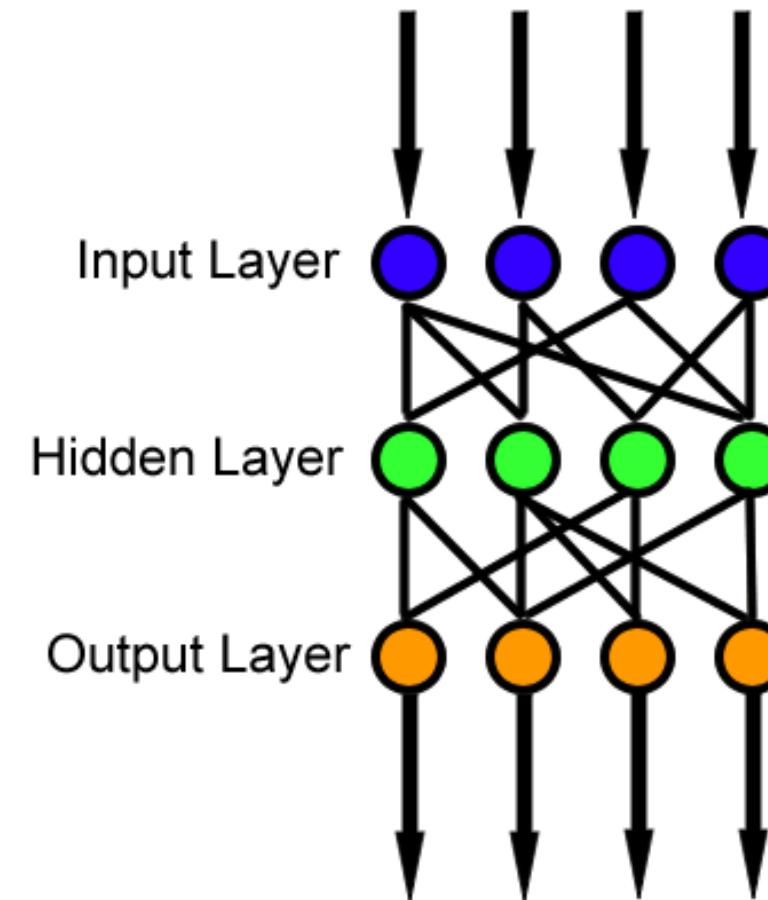
Segment Anything Model (SAM): a new AI model from Meta AI that can "cut out" any object, in any image, with a single click

SAM is a promptable segmentation system with zero-shot generalization to unfamiliar objects and images, without the need for additional training.

→ Try the demo

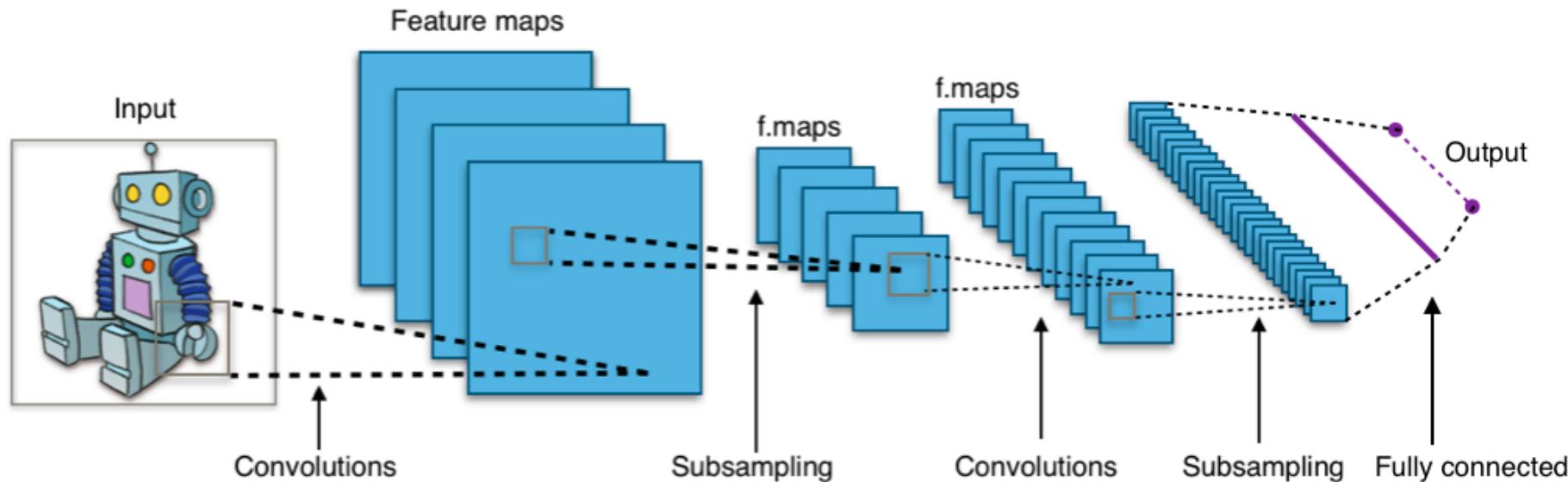
Feed forward neural networks

- Gauss, Legendre (around 1800):
Single layer, least squares
- McCulloch & Pitts (1940's):
Artificial neuron
- RosenBlatt, Jospbeh (around 1960):
Multilayer perceptron
- Linnainmaa (1970's):
Backpropagation
- Bengio ea (early 2000's): Deep
learning



- By Paskari at the English-language Wikipedia, CC BY-SA 3.0, <https://commons.wikimedia.org/w/index.php?curid=146663611>

Convolutional neural networks

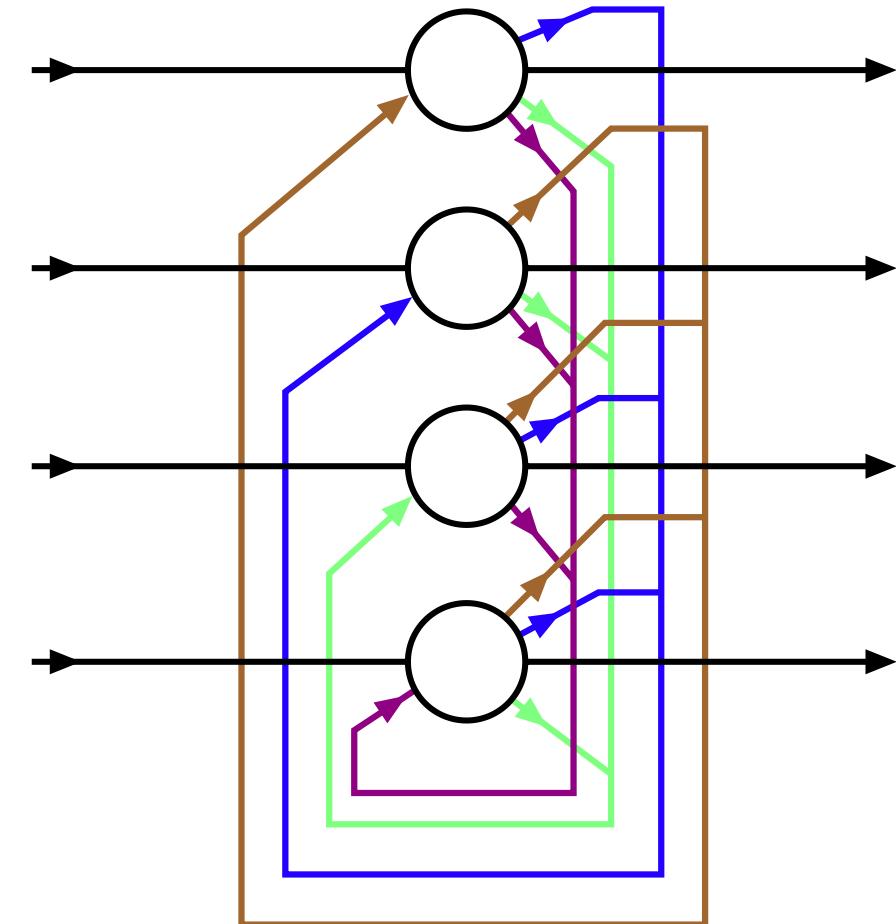


- Hubel & Wiesel (1950's): visual cortex studies
- Fukushima (1969): CNN + ReLU, hand-designed kernels
- Fukushima (1979), LeCun (1987): training kernels
- Oh & Jung (2004): GPU implementation

By Aphex34 - Own work, CC BY-SA 4.0,
<https://commons.wikimedia.org/w/index.php?curid=45679374>

Recurrent neural networks

- Cajal (early 1900's): Recurrence in cerebellar cortex
- McCulloch & Pitts: Recurrent neuron
- Rosenblatt (1960's), Hopfield (1982): Hebbian learning
- Hochreiter & Schmidhuber (1995): Long short-term memory (LSTM)
- Schuster ea (1997): Bidirectional recurrent neural networks (BRNN)



By Zawersh, CC BY-SA 3.0,
<https://commons.wikimedia.org/w/index.php?curid=37811881>

Recurrence

Consider: $x_t, \alpha \in \mathbb{R}$

Running average: $y_0 = x_0, y_t = \alpha y_{t-1} + (1 - \alpha)x_t$

Exampel:

$$x = [7 \ 0 \ 3 \ 0 \ 1 \ 8 \ 7 \ 3 \ 10 \ 0]$$

$$y_{\alpha=0.3} = [7 \ 2 \ 3 \ 1 \ 1 \ 6 \ 7 \ 4 \ 8 \ 2]$$

$$y_{\alpha=0.7} = [7 \ 5 \ 5 \ 4 \ 3 \ 4 \ 5 \ 5 \ 7 \ 5]$$

$$y_0 = x_0$$

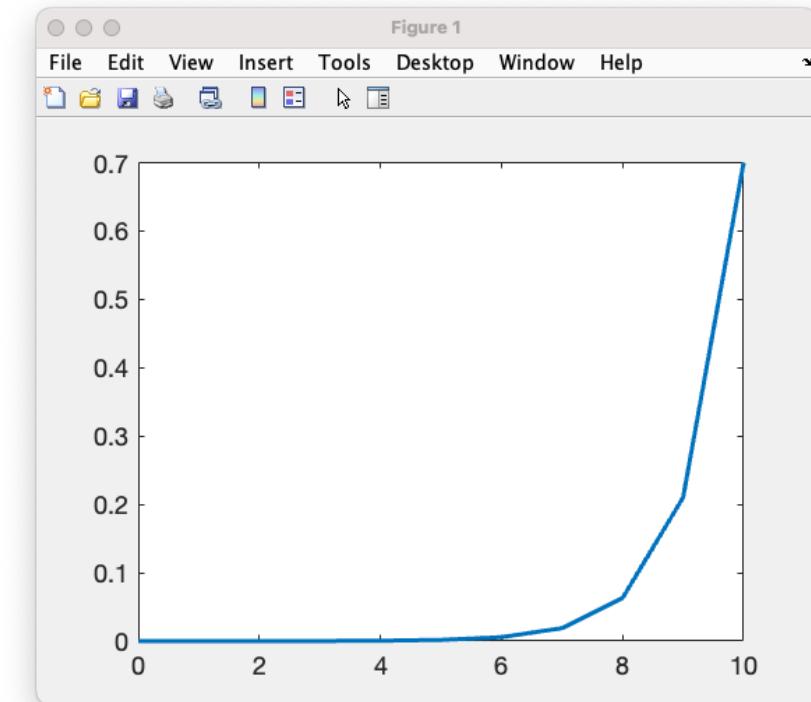
$$y_1 = \alpha y_0 + (1 - \alpha)x_1 = \alpha x_0 + (1 - \alpha)x_1$$

$$y_2 = \alpha y_1 + (1 - \alpha)x_2 = \alpha^2 x_0 + \alpha(1 - \alpha)x_1 + (1 - \alpha)x_2$$

$$y_3 = \alpha y_2 + (1 - \alpha)x_3 = \alpha^3 x_0 + \alpha^2(1 - \alpha)x_1 + \alpha(1 - \alpha)x_2 + (1 - \alpha)x_3$$

...

$$y_t = (1 - \alpha) \sum_{i=0}^t \alpha^{t-i} x_i$$



Long short-term memory (LSTM)

https://en.wikipedia.org/wiki/Long_short-term_memory

$$x = [7 \ 0 \ 3 \ 0 \ 1 \ 8 \ 7 \ 3 \ 10 \ 0]$$

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$$

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$$

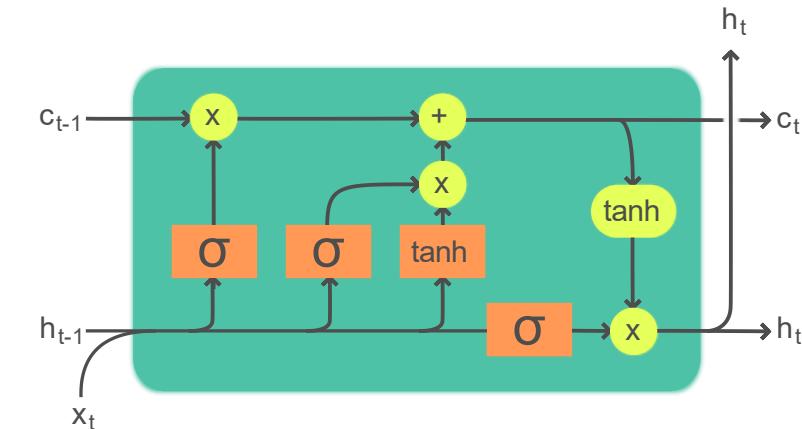
$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$$

$$\tilde{c}_t = \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

$$h_t = o_t \odot \sigma_h(c_t)$$

- $x_t \in \mathbb{R}^d$: input vector to the LSTM unit
- $f_t \in (0, 1)^h$: forget gate's activation vector
- $i_t \in (0, 1)^h$: input/update gate's activation vector
- $o_t \in (0, 1)^h$: output gate's activation vector
- $h_t \in (-1, 1)^h$: hidden state vector also known as output vector of the LSTM unit
- $\tilde{c}_t \in (-1, 1)^h$: cell input activation vector
- $c_t \in \mathbb{R}^h$: cell state vector
- $W \in \mathbb{R}^{h \times d}$, $U \in \mathbb{R}^{h \times h}$ and $b \in \mathbb{R}^h$: weight matrices and bias vector parameters which need to be learned during training



Legend:

Layer	Componentwise	Copy	Concatenate

- By Guillaume Chevalier - File:The_LSTM_Cell.svg, CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=109362147>

Transformer networks

Vision Transformers

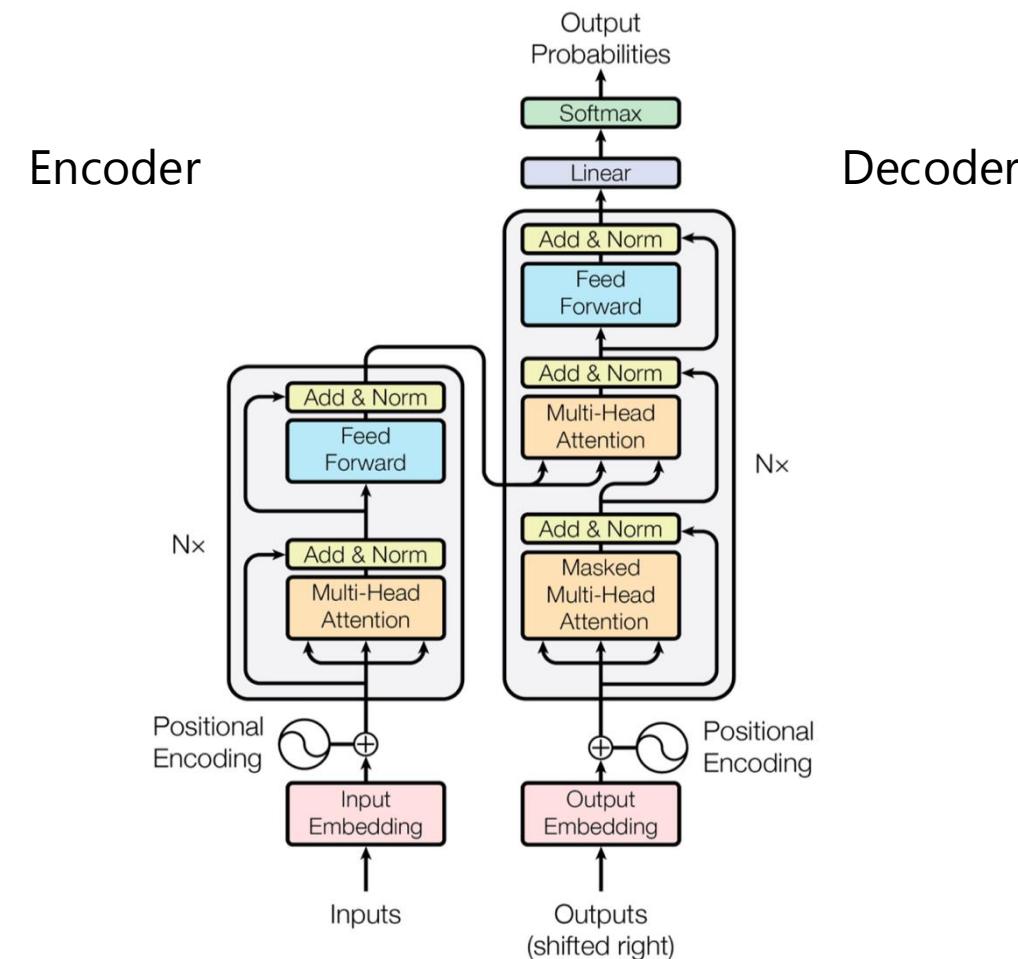
- Cho ea, Sutskever ea (2014): seq2seq
- Vaswani ea (2017):
Attention is all you need
- Dosovitskiy ea (2020):
An Image is Worth 16x16 Words ...

Transformers | Davide Coccomin | 2021

By Davide Coccomin - Own work, CC BY-SA 4.0,
<https://commons.wikimedia.org/w/index.php?curid=110678226>

Vaswani: Attention Is All You Need

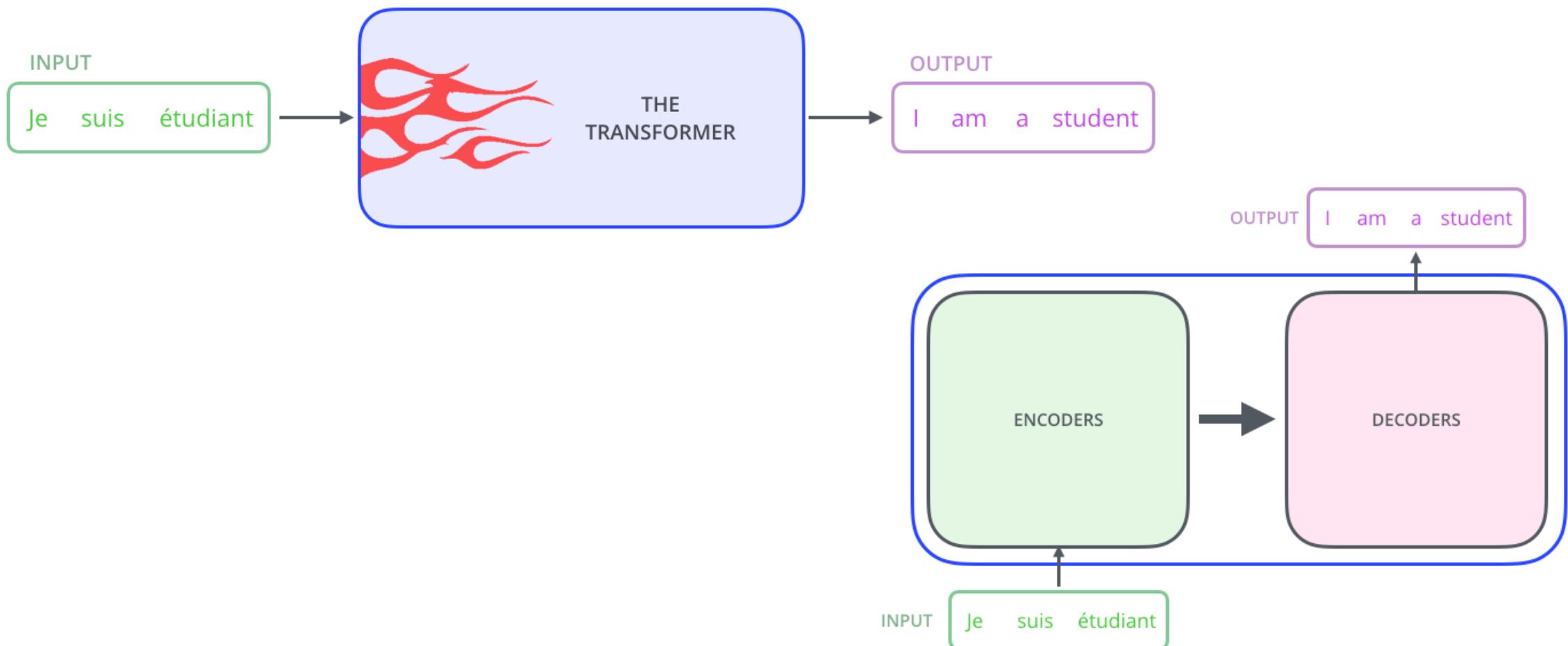
Dosovitskiy ea (2020): An Image is Worth 16x16 Words...



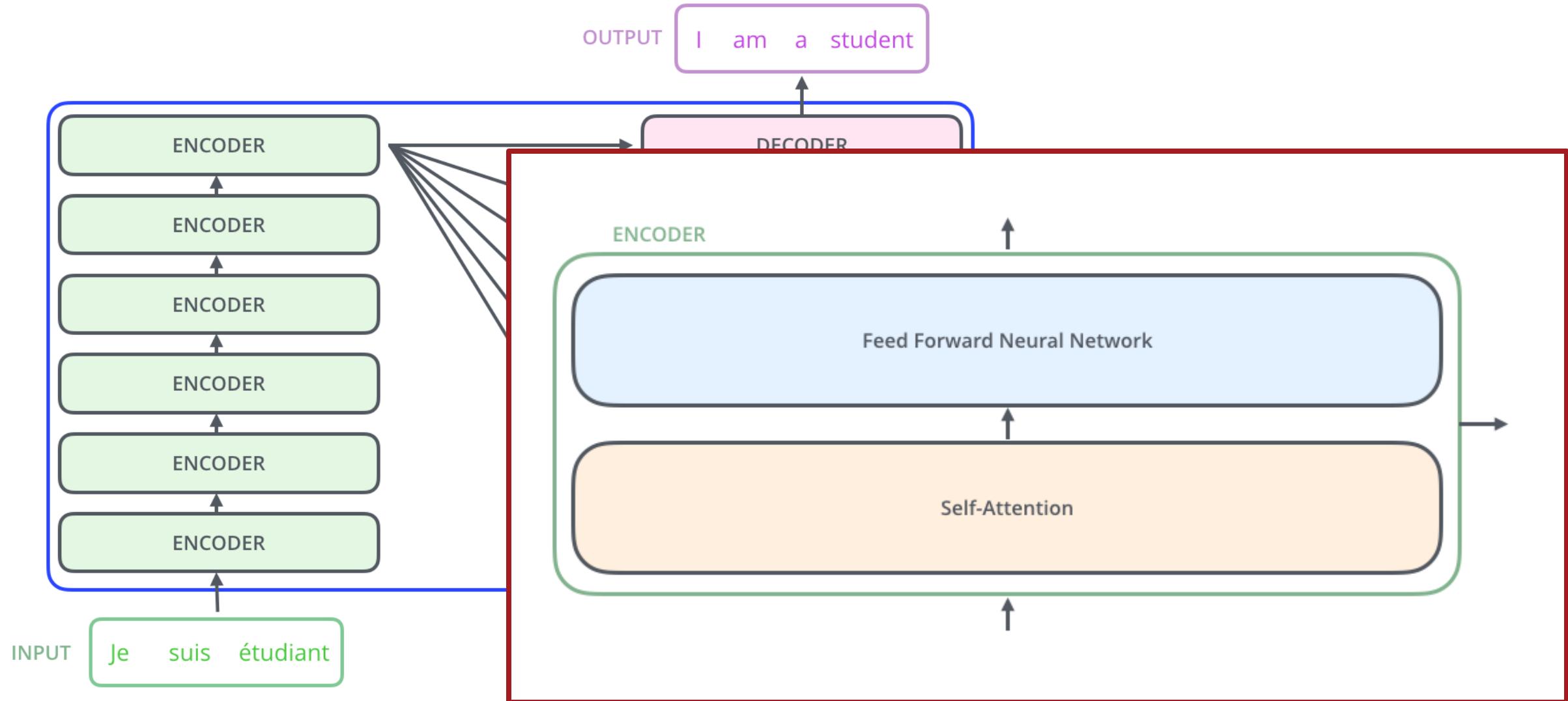
The Illustrated Transformer

<http://jalammar.github.io/illustrated-transformer/>

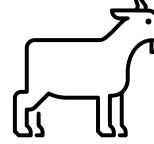
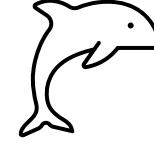
Transformers in the context of languages:

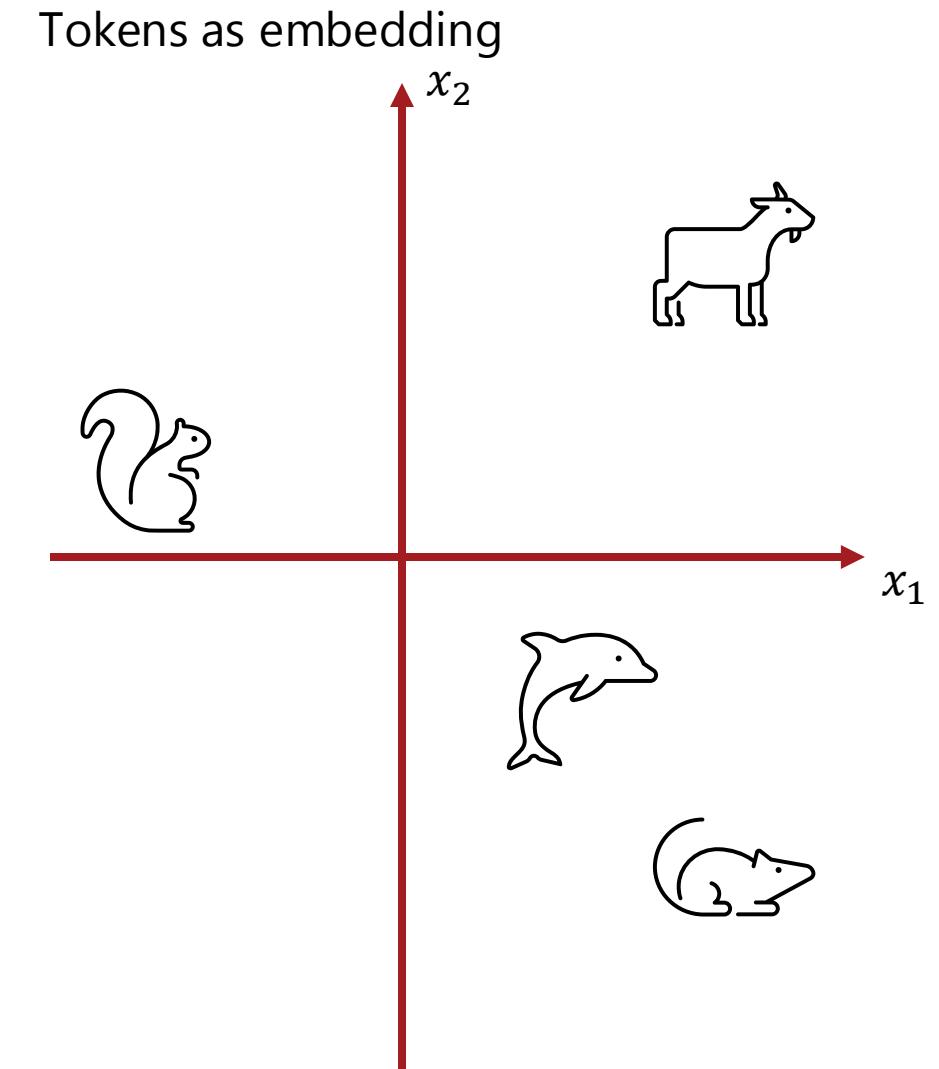


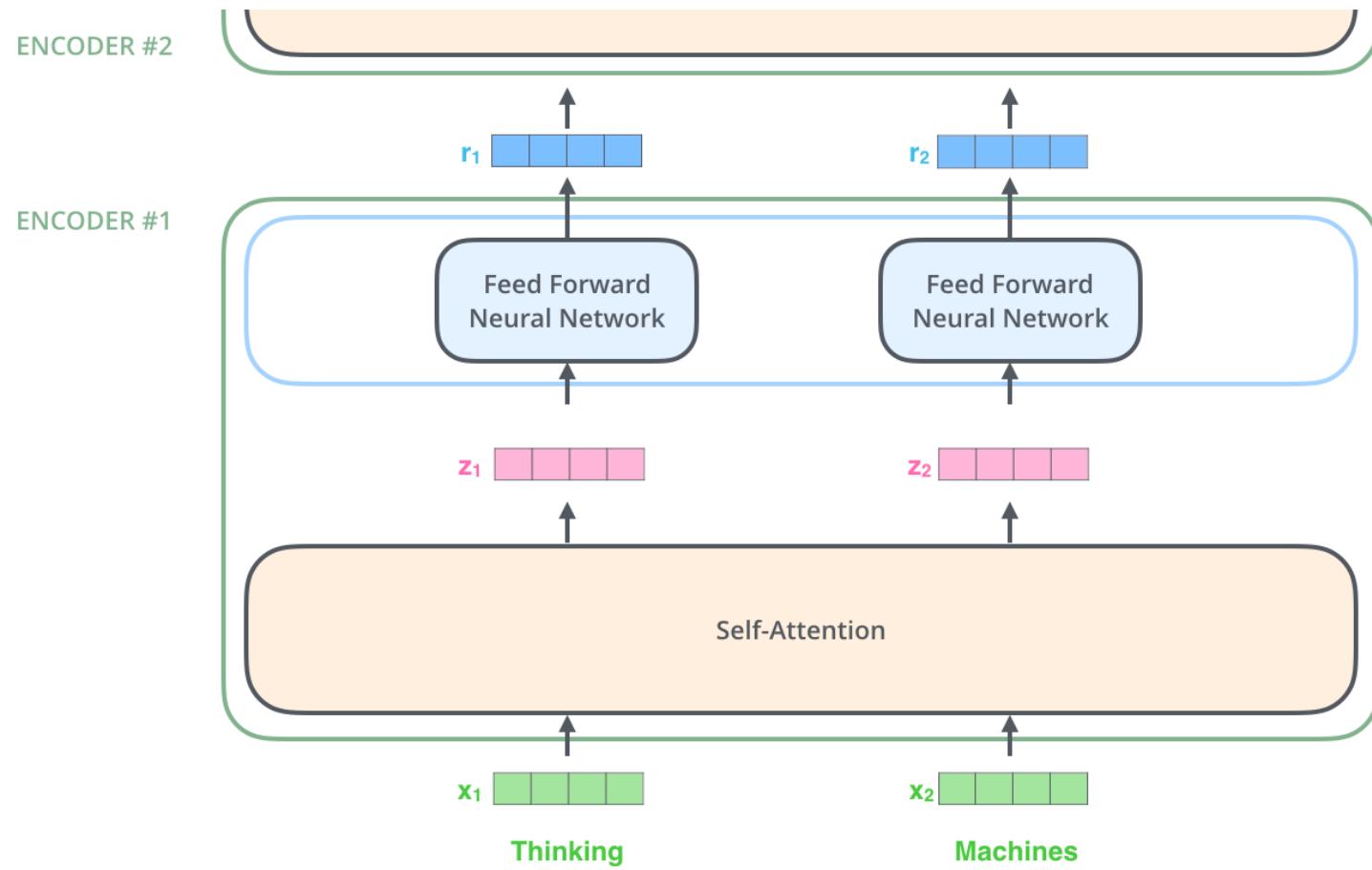
Encoder (and decoder) are stacks of identical units



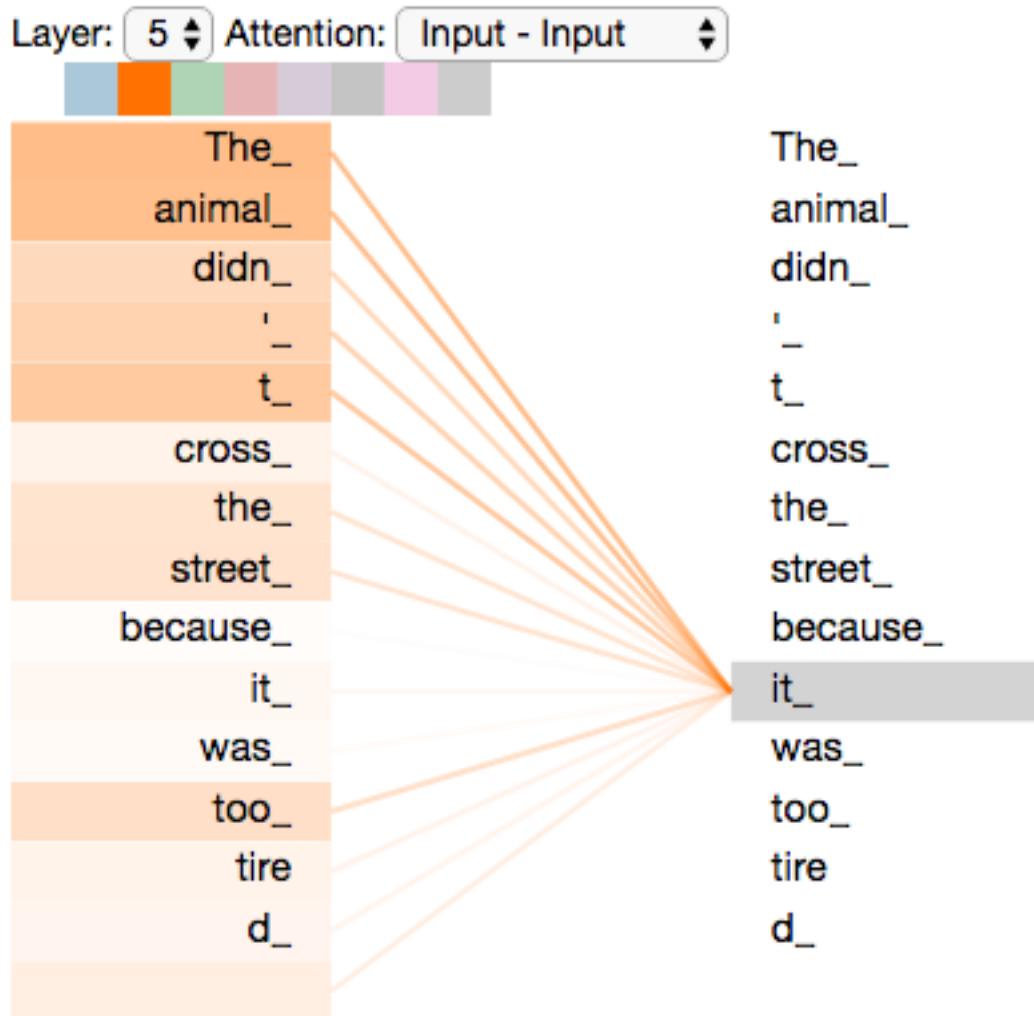
Classification by Query, Key, Value and Tokens

	key	value
query		
		Goat
?		Squirrel
		Dolphin
		Mouse





Self attention: The animal didn't cross the street because it was too tired



$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

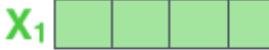
Matrix form

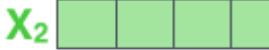
Input

Thinking

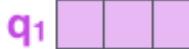
Machines

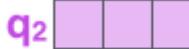
Embedding

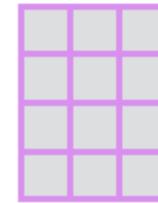
x_1 

x_2 

Queries

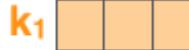
q_1 

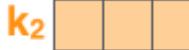
q_2 

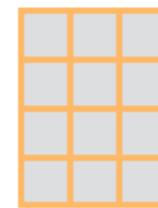


W^Q

Keys

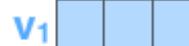
k_1 

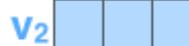
k_2 



W^K

Values

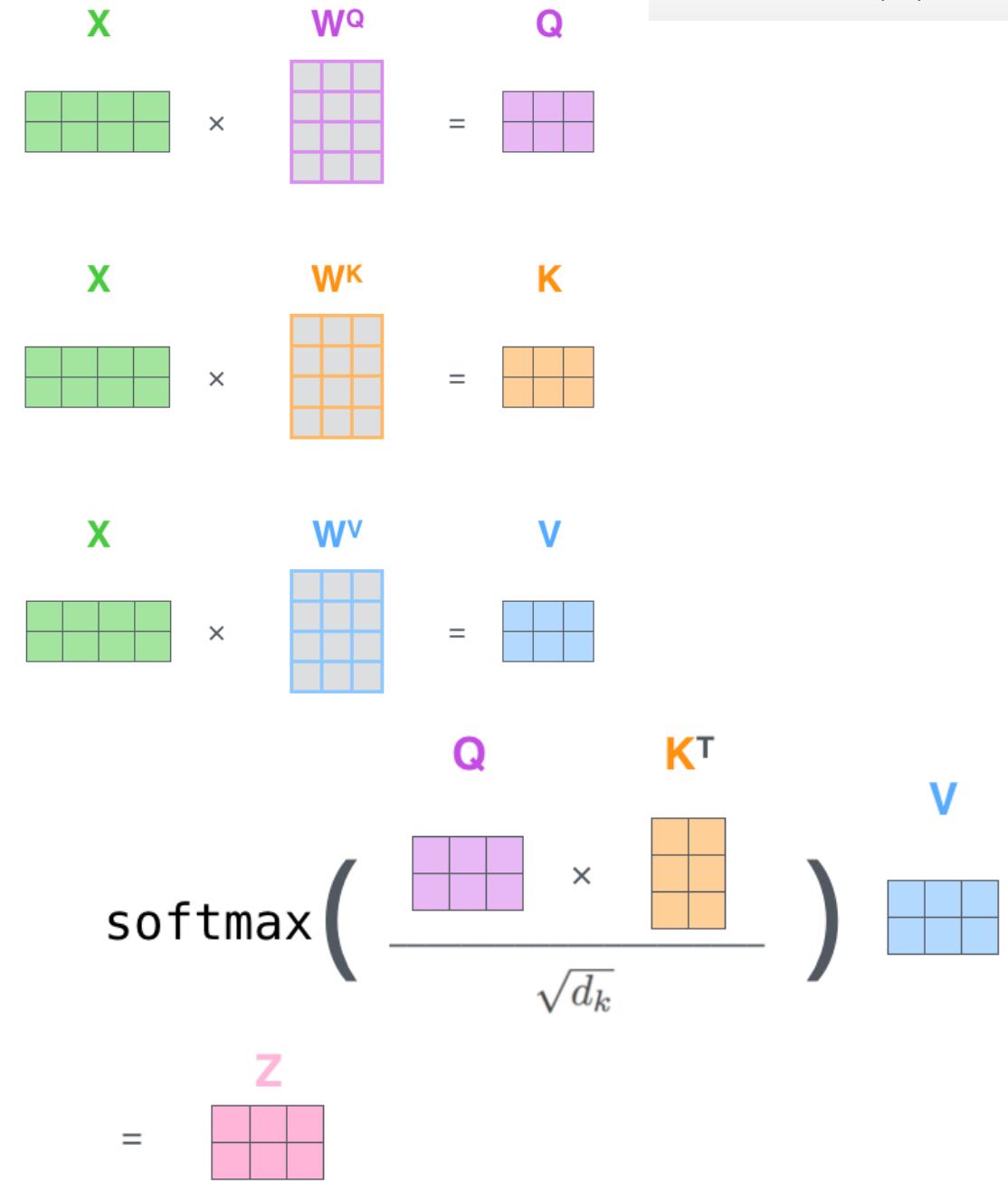
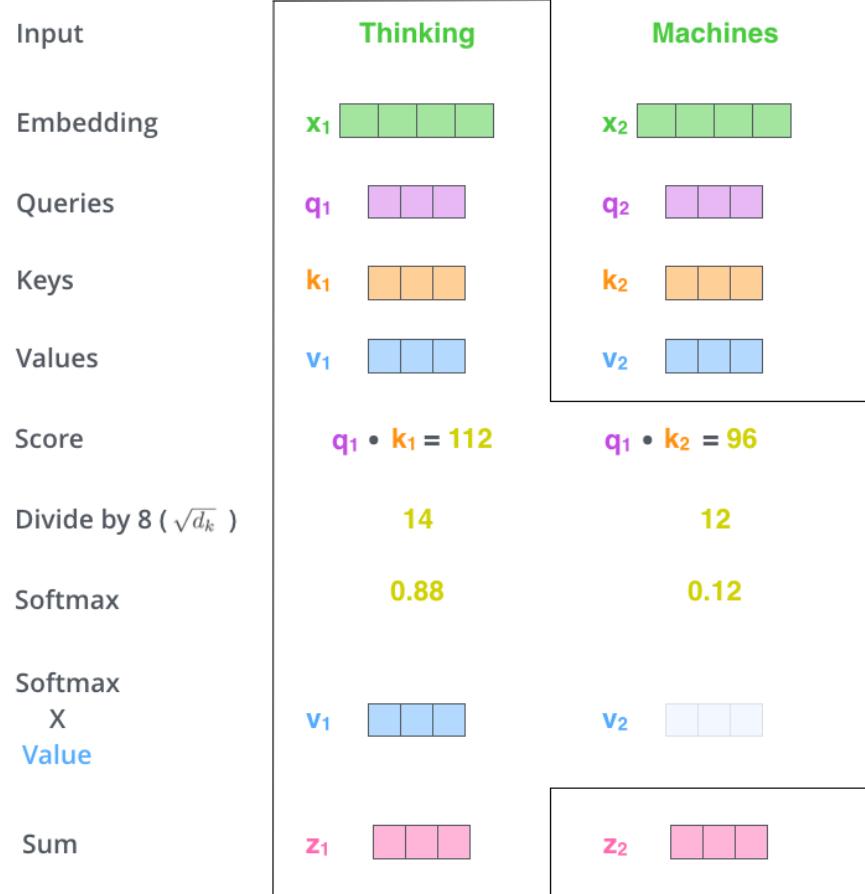
v_1 

v_2 



W^V

Self attention, single head



Papers with code

<https://paperswithcode.com/method/transformer>

The screenshot shows a web browser window with the URL <https://paperswithcode.com/method/transformer>. The page title is "Transformer". It includes a brief introduction by Vaswani et al. from the paper "Attention Is All You Need", a source link to the paper, and two buttons: "Read Paper" and "See Code". To the right of the text is a detailed diagram of the Transformer architecture. The diagram illustrates the flow of data through an encoder and a decoder. In the encoder, inputs pass through an "Input Embedding" layer, followed by a stack of N_x layers. Each layer contains a "Multi-Head Attention" block (orange), which is followed by an "Add & Norm" block (light blue). The output of the encoder is then passed through a "Feed Forward" block (blue) and another "Add & Norm" block. In the decoder, the input from the encoder is combined with "Positional Encoding" (green circle with a plus sign) at the bottom. The sequence then passes through a stack of N_x layers. Each layer in the decoder contains a "Masked Multi-Head Attention" block (orange), followed by an "Add & Norm" block. The output of the decoder is then passed through a "Multi-Head Attention" block (orange), followed by an "Add & Norm" block. Finally, the output goes through a "Feed Forward" block (blue) and an "Add & Norm" block. The final output is a "Softmax" layer (green) followed by a "Linear" layer (purple) to produce "Output Probabilities".