Lecture 8: Ttransformer: Attention

Lan Xu
SIST, ShanghaiTech
Fall, 2023

Schedule Update

Lecture 8	Wednesday 25/10 Week 5	**Transformer - I** Attention
Lecture 9	Monday 30/10 Week 6	**Transformer - II** Transformer architectures
CVPR	Wednesday 1/11 Week 6	NO CLASS
CVPR	Monday 6/11 Week 7	NO CLASS
CVPR	Wednesday 8/11 Week 7	NO CLASS
Lecture 10	Monday 13/11 Week 8	**Transformer - III** Transformer Variants
Lecture 11	Wednesday 15/11 Week 8	**Neural networks for Prediction - I** Prediction task and applications
Lecture 12	Monday 20/11 Week 9	**Neural networks for Prediction - II** Prediction task Visual Large Model

→

Lecture 8	Wednesday 25/10 Week 5	**Transformer - I** Attention
Lecture 9	Monday 30/10 Week 6	**Transformer - II** Transformer architectures
Lecture 10	Wednesday 1/11 Week 6	**Transformer - III** Transformer Variants
Lecture 11	Monday 6/11 Week 7	**Neural networks for Prediction - I** Prediction task and applications
CVPR	Wednesday 8/11 Week 7	NO CLASS
CVPR	Monday 13/11 Week 8	NO CLASS
CVPR	Wednesday 15/11 Week 8	NO CLASS
Lecture 12	Monday 20/11 Week 9	**Neural networks for Prediction - II** Prediction task Visual Large Model

Outline

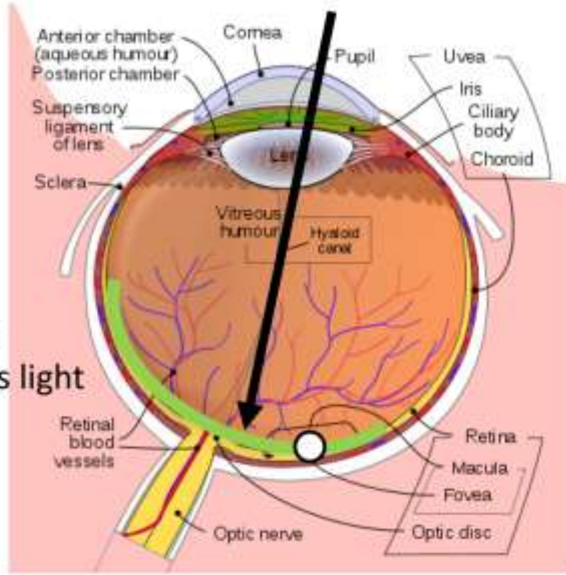
- Recall Attention in Seq2seq
  - Attention models: NMT and Image Captioning
- General Attention Layer
  - From General-attention to Self-attention
  - Positional encoding, Self-attention and CNN
- Transformer
  - Encoder-Decoder Artichecture
  - Transfer Learning and Vision Transformer

Acknowledgement: Feifei Li et al's cs231n notes

Attention Mechanism

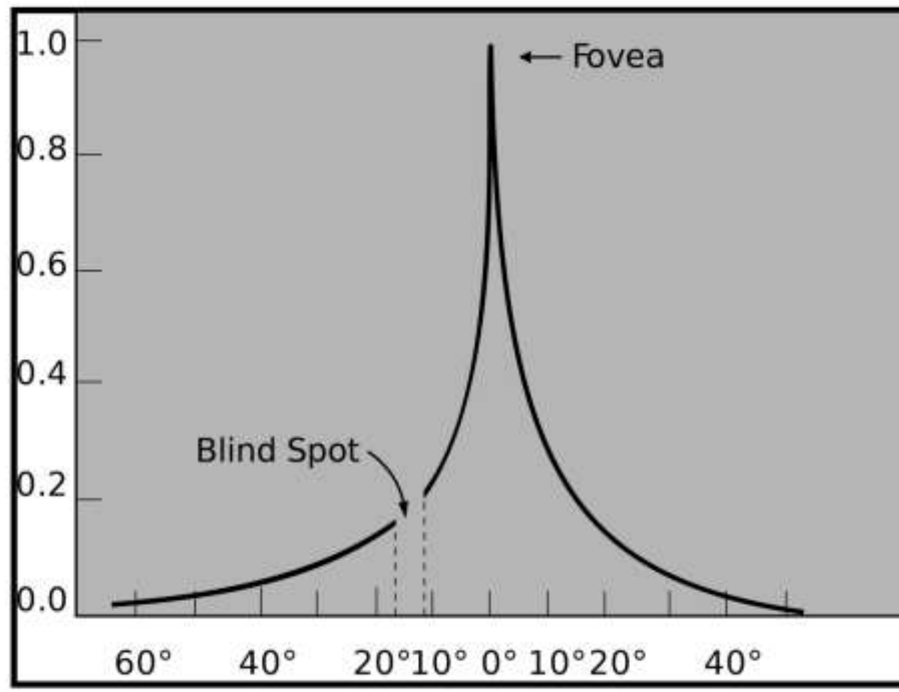
- Human Vision: Fovea

Light enters eye

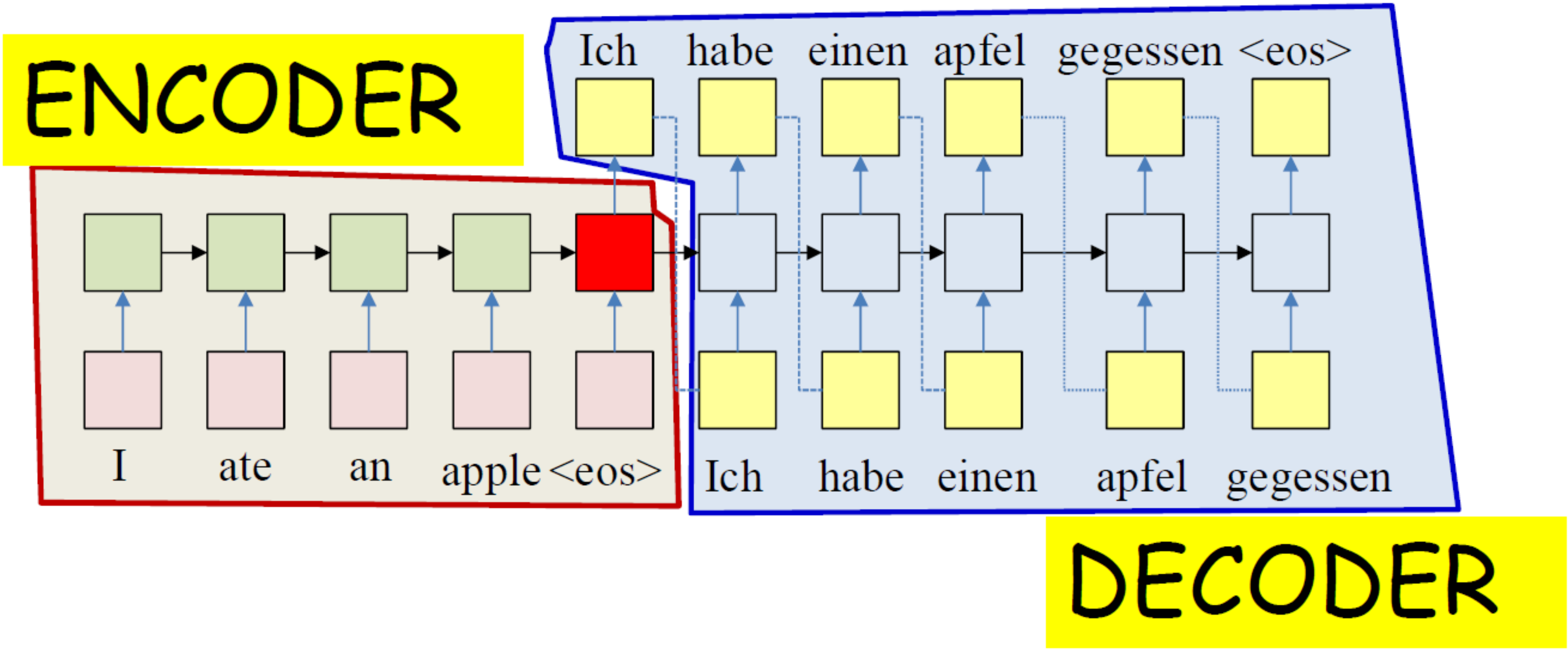


Retina detects light

The **fovea** is a tiny region of the retina that can see with high acuity



Recall the encoder-decoder structure



NMT with RNN and Attention

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

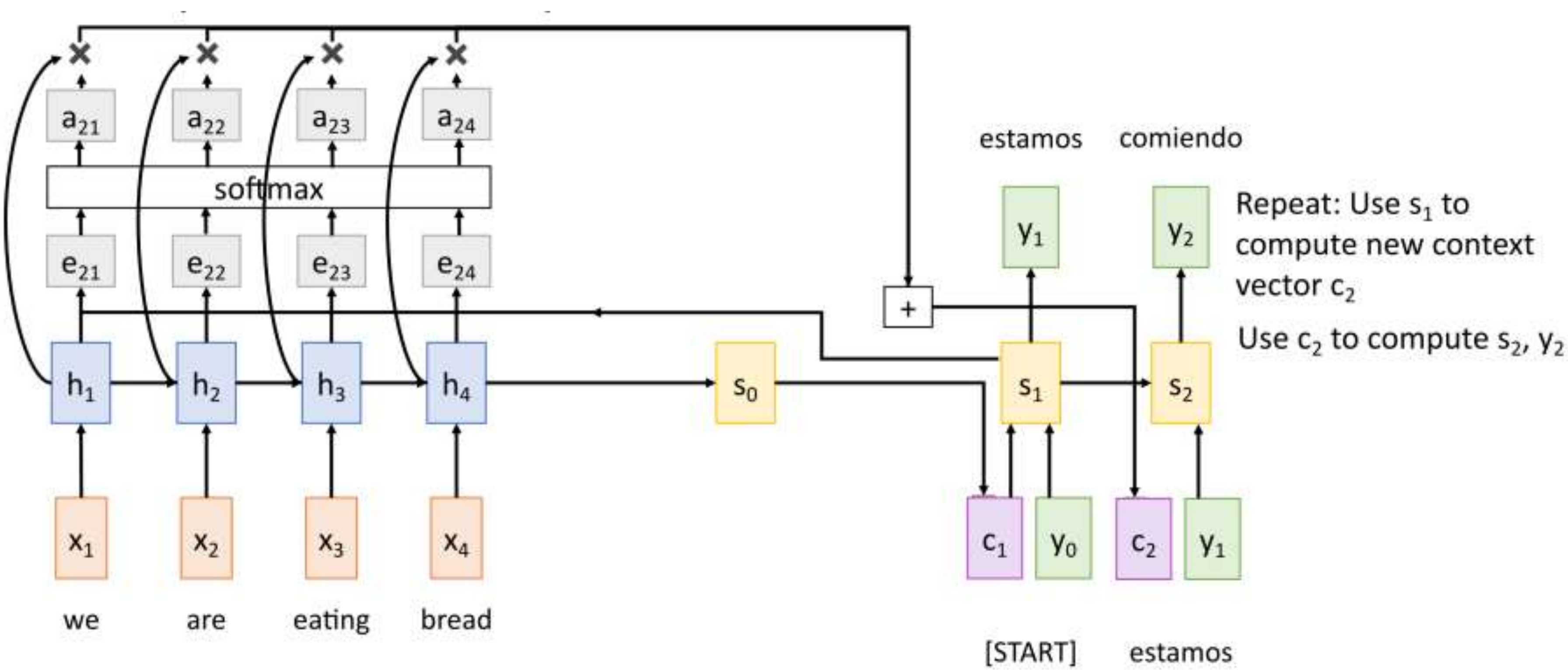


Image Caption with RNN and Attention

- Recall the one using spatial features
- Problem: Input is “bottlenecked” through c

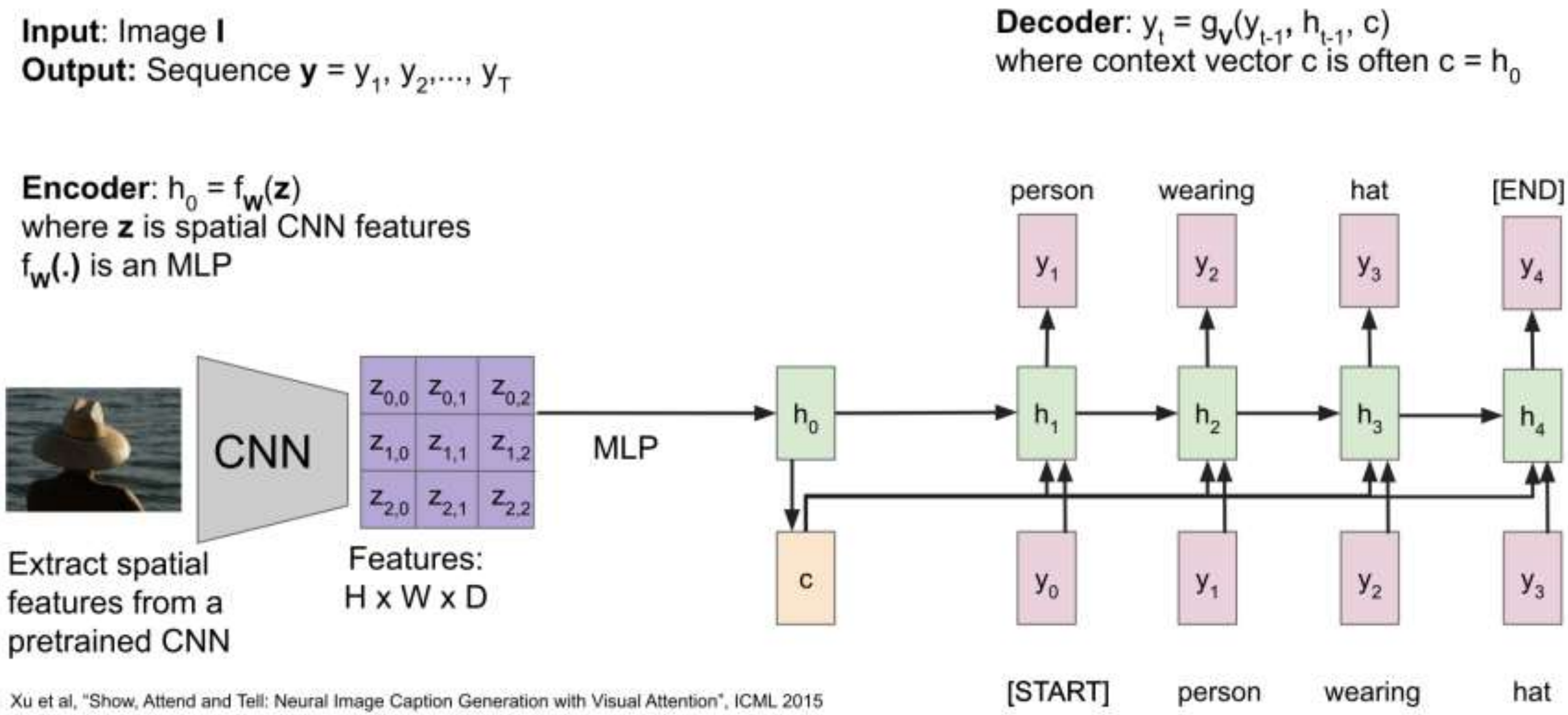
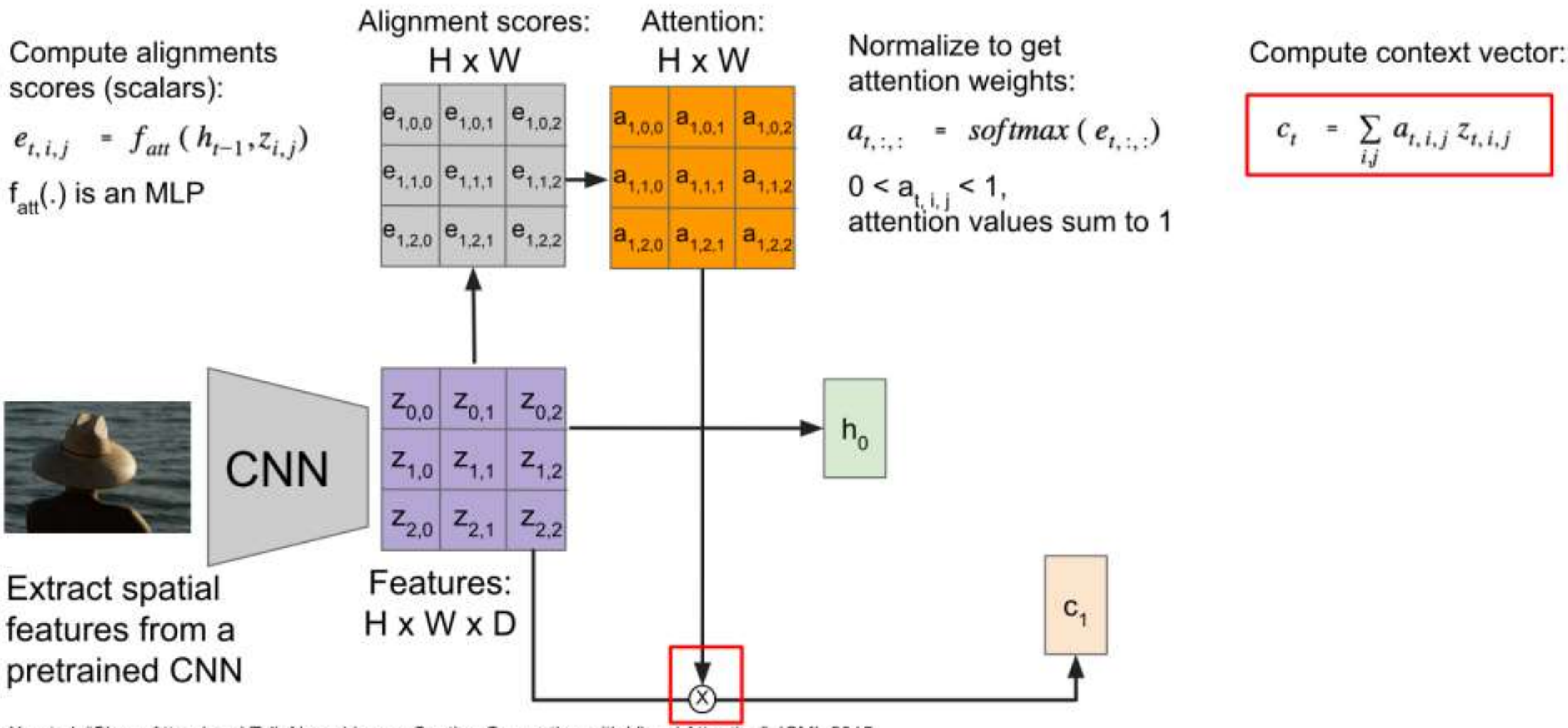


Image Caption with RNN and Attention

- Alignment → Attention



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

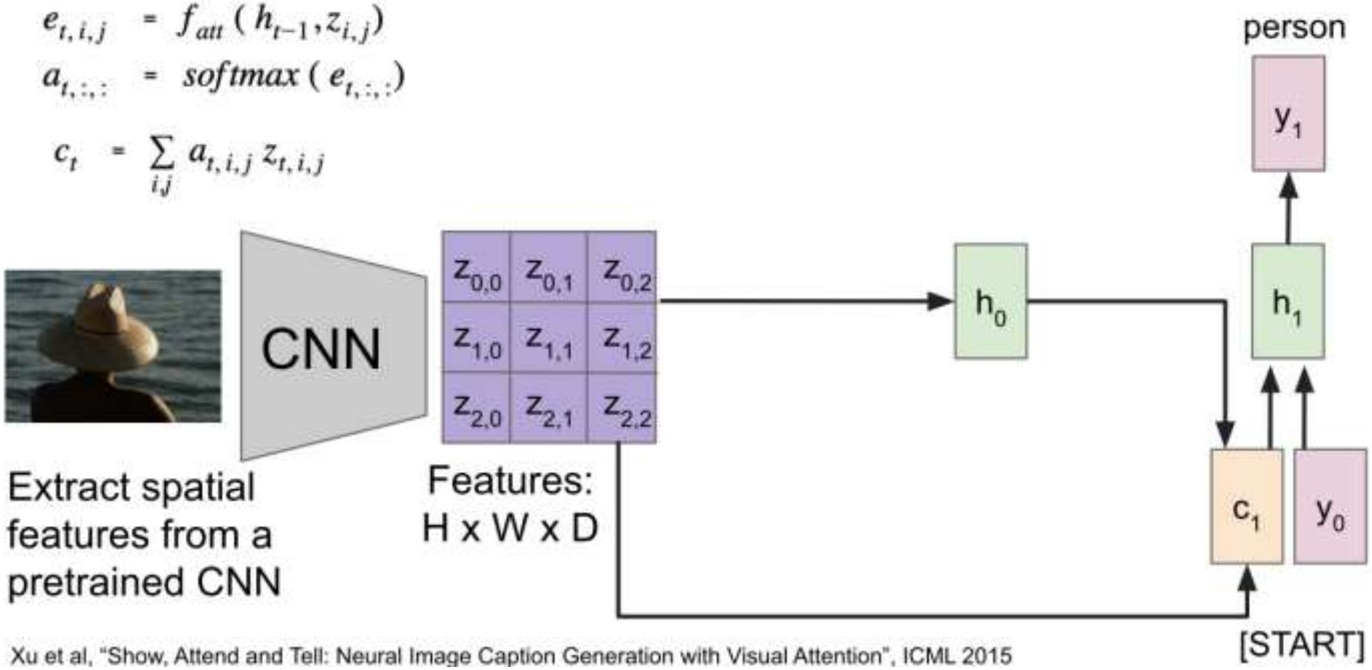
Image Caption with RNN and Attention

- Weighted context → decoder

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

Decoder: $y_t = g_V(y_{t-1}, h_{t-1}, c_t)$
New context vector at every time step

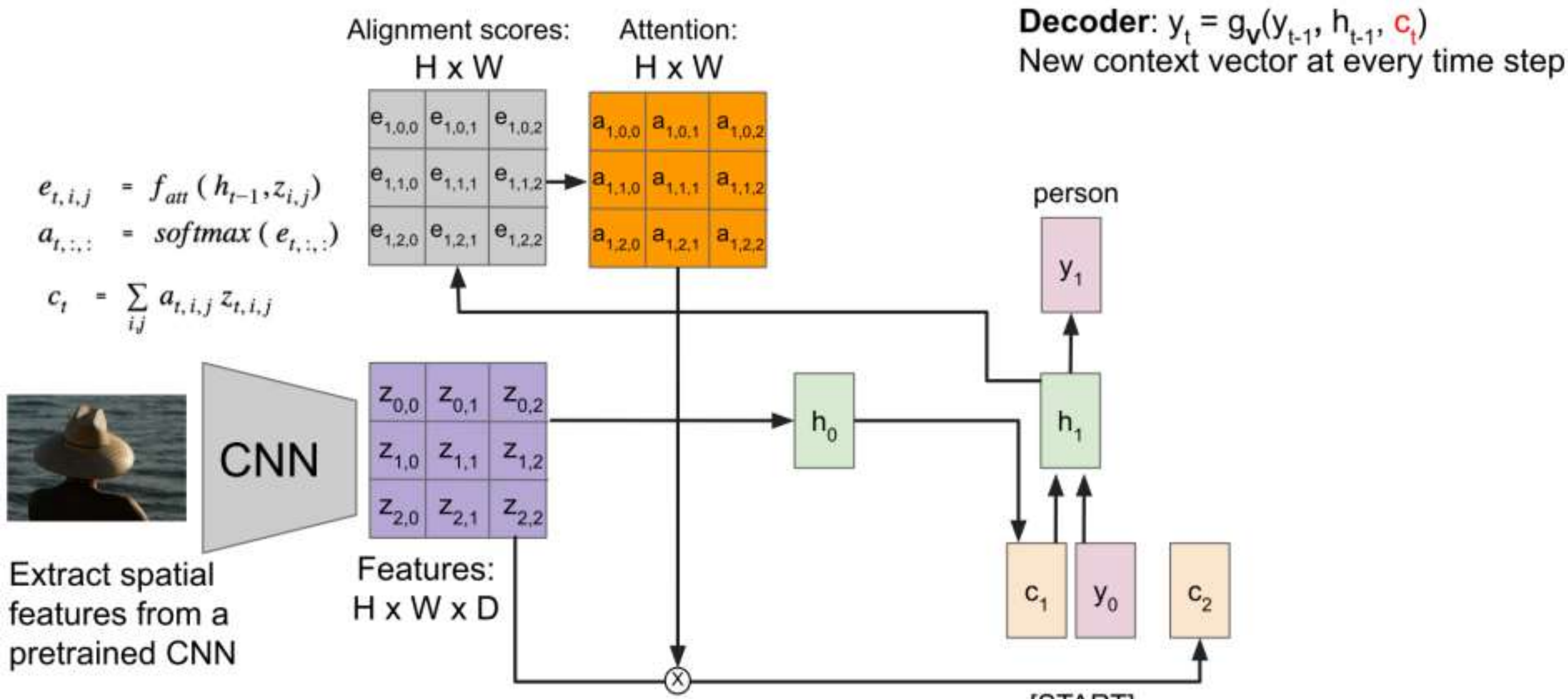
$$e_{t,i,j} = f_{att}(h_{t-1}, z_{i,j})$$
$$a_{t,:,:} = softmax(e_{t,:,:})$$
$$c_t = \sum_{i,j} a_{t,i,j} z_{t,i,j}$$



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

Image Caption with RNN and Attention

- Compute the new Alignment & Attention maps

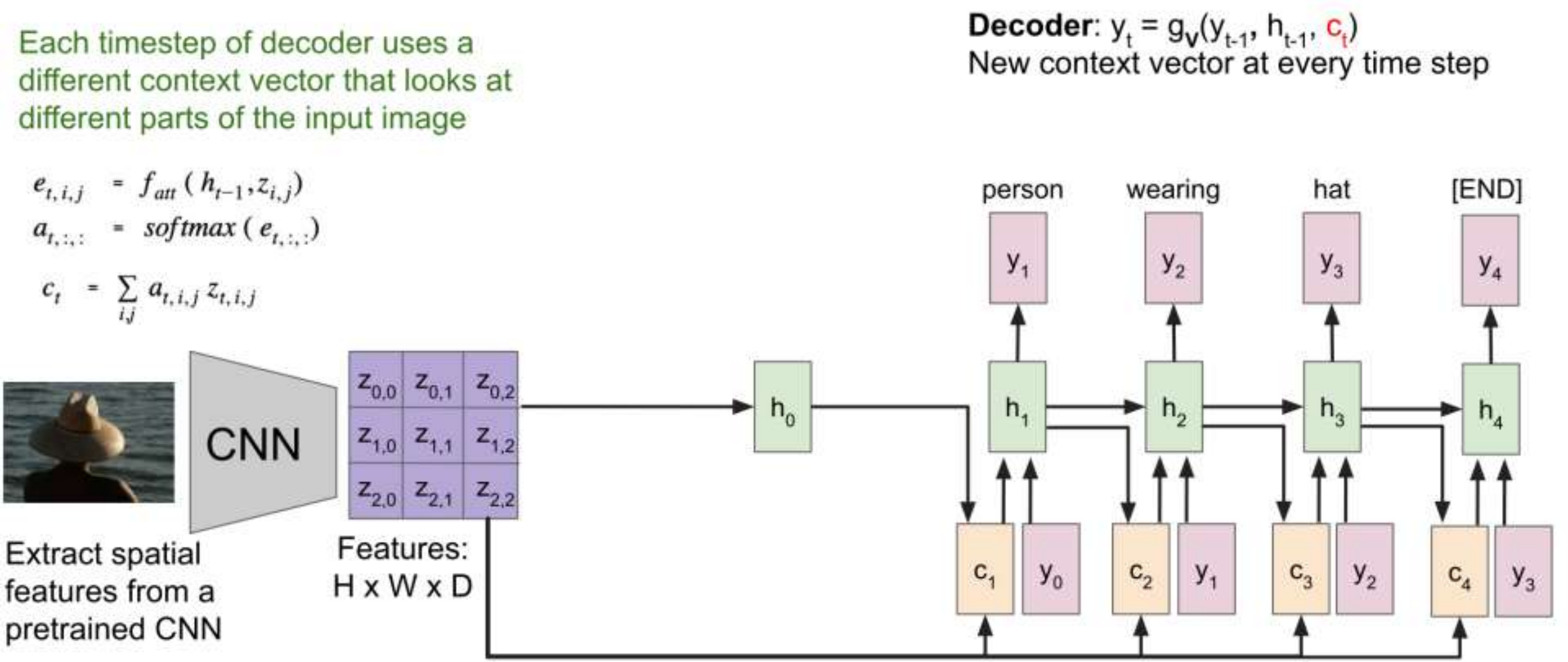


$$e_{t,i,j} = f_{att}(h_{t-1}, z_{i,j})$$
$$a_{t,:,:} = softmax(e_{t,:,:})$$
$$c_t = \sum_{i,j} a_{t,i,j} z_{t,i,j}$$

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

Image Caption with RNN and Attention

- Repeat …



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

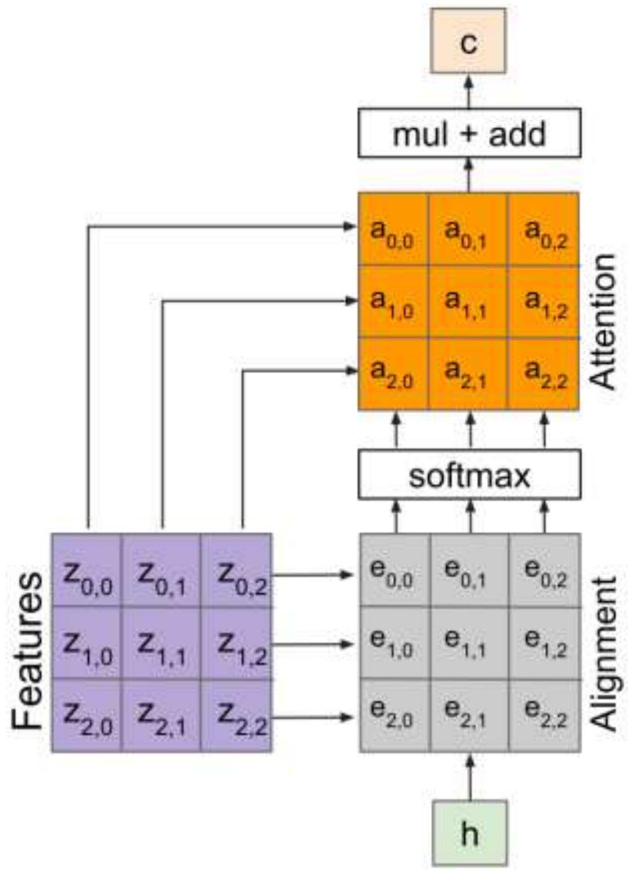
Outline

- Recall RNNs in Vision and NLP
  - Attention models: NMT and Image Captioning
- General Attention Layer
  - From General-attention to Self-attention
  - Positional encoding
  - Self-attention and CNN

Acknowledgement: Feifei Li et al's cs231n notes

General Attention Layer

- Let's fetch the Attention design for more general task!
- Take image caption for example: Feature & Query



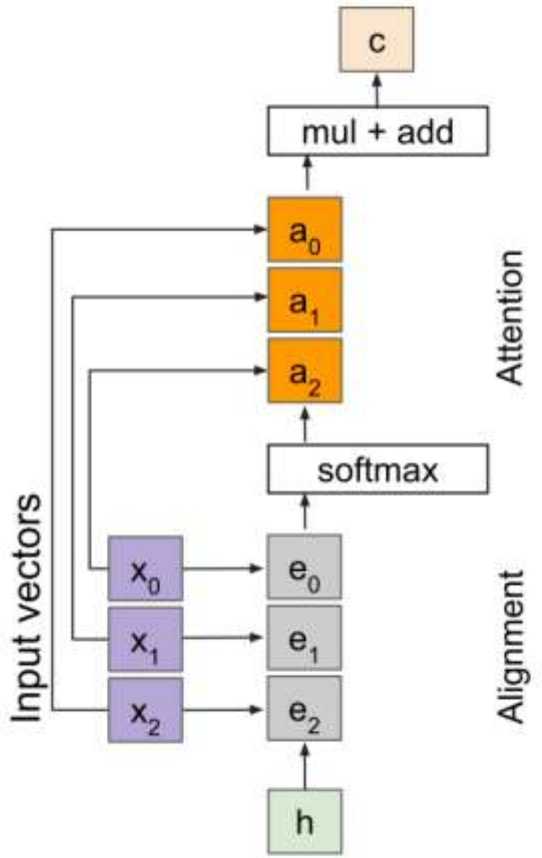
Outputs:
context vector: **c** (shape: D)

Operations:
Alignment: $e_{i,j} = f_{att}(h, z_{i,j})$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $\mathbf{c} = \sum_{i,j} a_{i,j} z_{i,j}$

Inputs:
Features: **z** (shape: H x W x D)
Query: **h** (shape: D)

General Attention Layer

- From image to general input vectors



Outputs:
context vector: **c** (shape: D)

Change $f_{att}(.)$ to a simple dot product

- only works well with key & value transformation trick (will mention in a few slides)

Operations:
Alignment: $e_i = h \cdot x_i$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $\mathbf{c} = \sum_i a_i x_i$

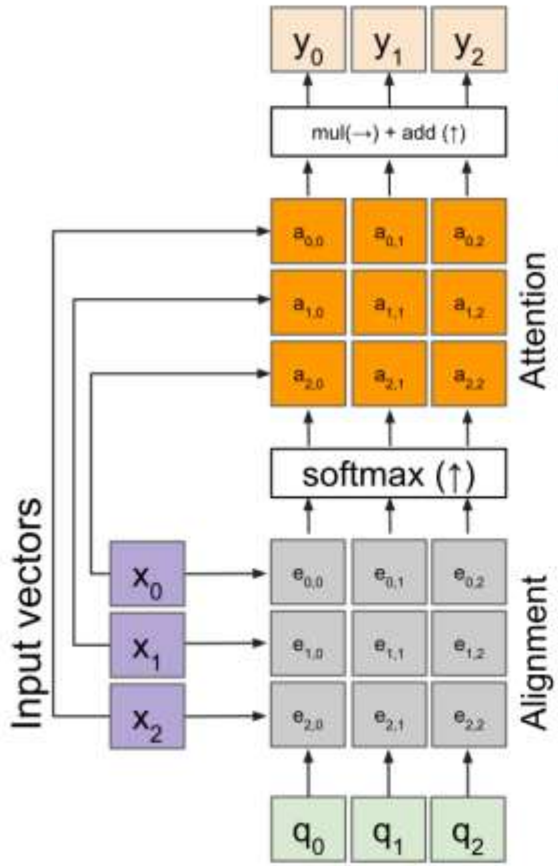
Attention operation is **permutation invariant.**

- Doesn't care about ordering of the features
- Stretch H x W = N into N vectors

Inputs:
Input vectors: **x** (shape: N x D)
Query: **h** (shape: D)

General Attention Layer

- From single to multiple query vectors



Outputs:
context vectors: **y** (shape: D)

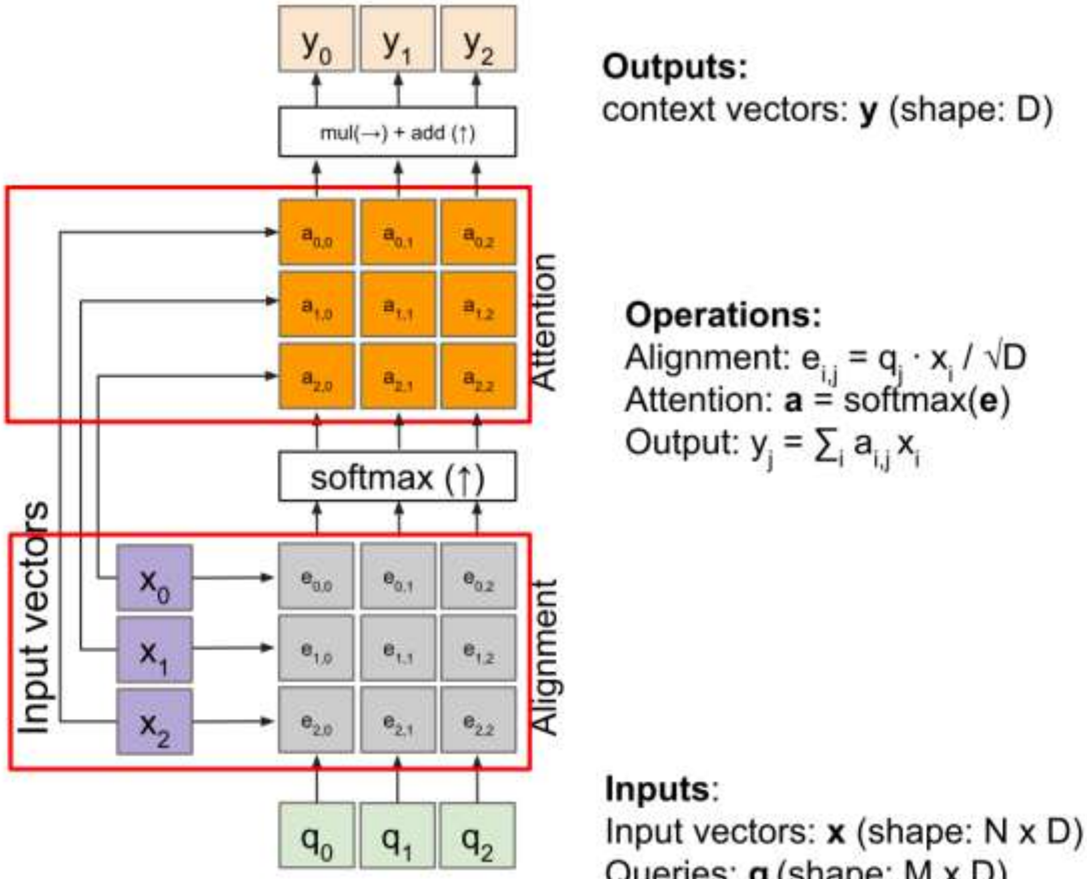
Operations:
Alignment: $e_{i,j} = q_j \cdot x_i / \sqrt{D}$
Attention: **a** = softmax(**e**)
Output: $y_j = \sum_i a_{i,j} x_i$

Multiple query vectors
- each query creates a new output context vector

Inputs:
Input vectors: **x** (shape: N x D)
Queries: **q** (shape: M x D)

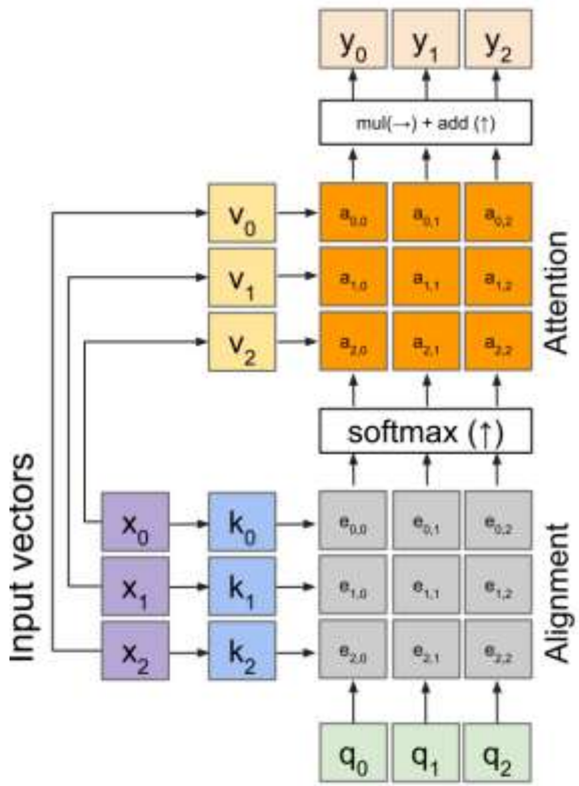
General Attention Layer

- Make the Alignment and Attention more flexible
- Use FC Layers!



General Attention Layer

- Split into Key vectors and Value vectors
- Q, K, V



Outputs:
context vectors: **y** (shape: D_v)

Operations:
Key vectors: $\mathbf{k} = \mathbf{x}W_k$
Value vectors: $\mathbf{v} = \mathbf{x}W_v$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $y_j = \sum_i a_{i,j} v_i$

Inputs:
Input vectors: **x** (shape: N x D)
Queries: **q** (shape: M x D_k)

The input and output dimensions can now change depending on the key and value FC layers

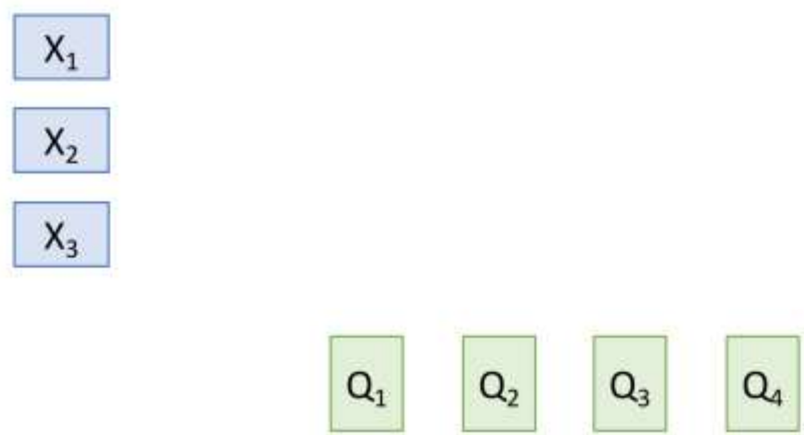
Notice that the input vectors are used for both the alignment as well as the attention calculations.

- We can add more expressivity to the layer by adding a different FC layer before each of the two steps.

General Attention Layer Summary

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

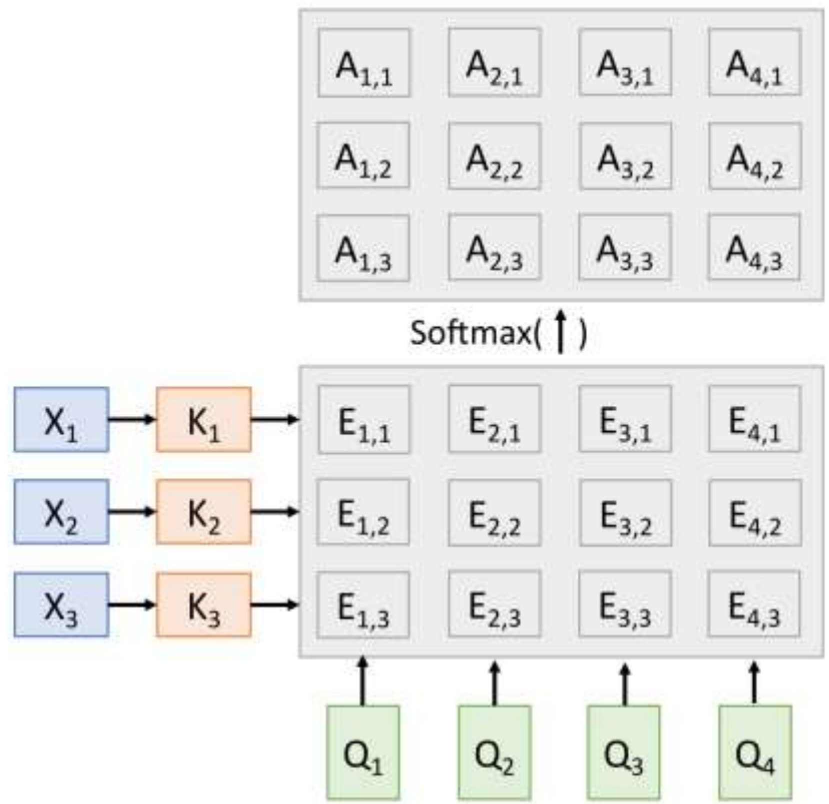
Computation:
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value Vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T$ (Shape: N_Q x N_X) $E_{i,j} = Q_i \cdot K_j / \mathrm{sqrt}(D_Q)$
Attention weights: $A = \mathrm{softmax}(E, \mathrm{dim}=1)$ (Shape: N_Q x N_X)
Output vectors: $Y = AV$ (Shape: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



General Attention Layer Summary

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

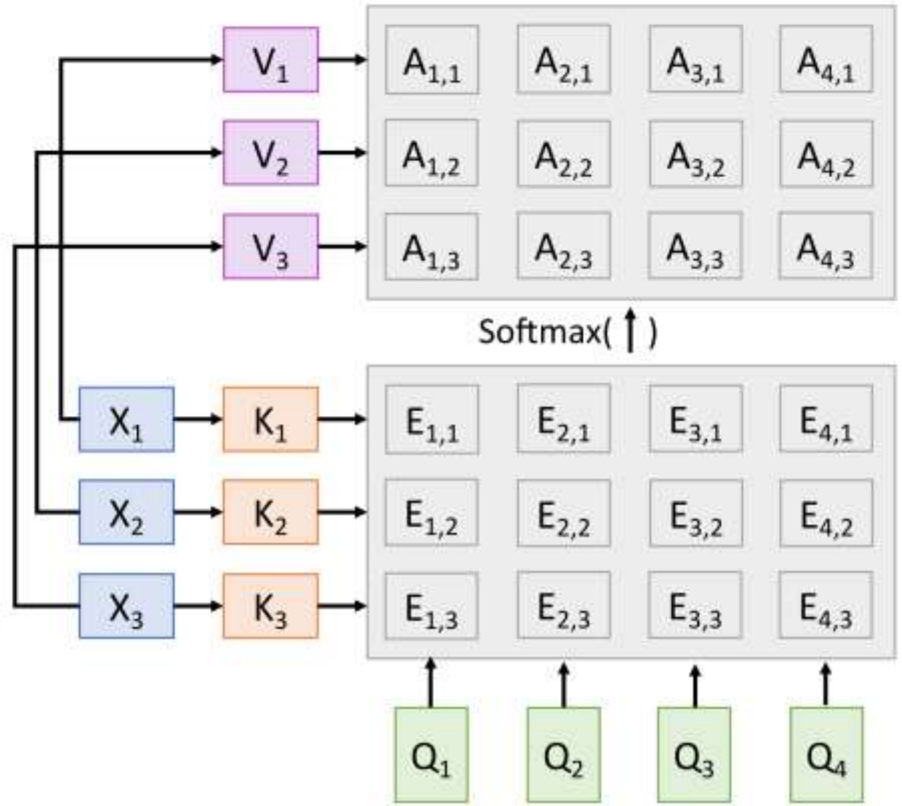
Computation:
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value Vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T$ (Shape: N_Q x N_X) $E_{i,j} = Q_i \cdot K_j / \mathrm{sqrt}(D_Q)$
Attention weights: $A = \mathrm{softmax}(E, \mathrm{dim}=1)$ (Shape: N_Q x N_X)
Output vectors: $Y = AV$ (Shape: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



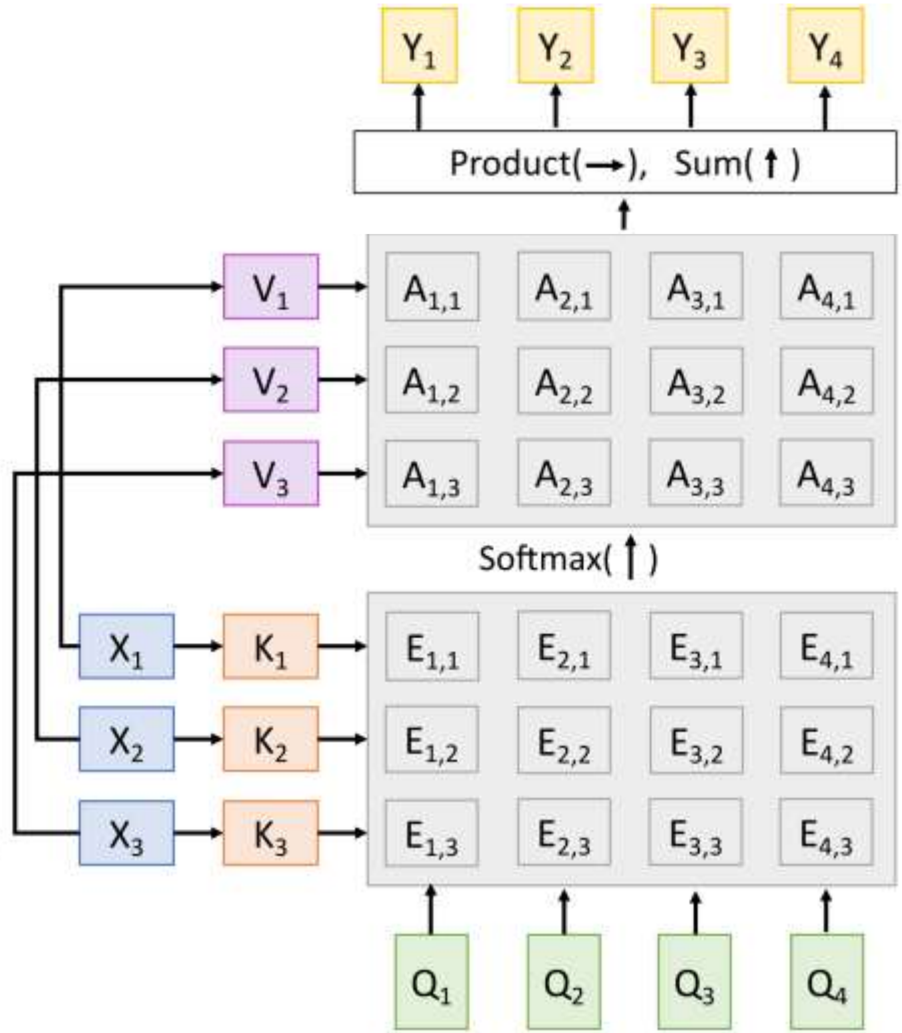
General Attention Layer Summary

<u>**Inputs**</u>:
Query vectors: Q (Shape: $N_Q \times D_Q$)
Input vectors: X (Shape: $N_X \times D_X$)
Key matrix: W_K (Shape: $D_X \times D_Q$)
Value matrix: W_V (Shape: $D_X \times D_V$)

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



General Attention Layer Summary

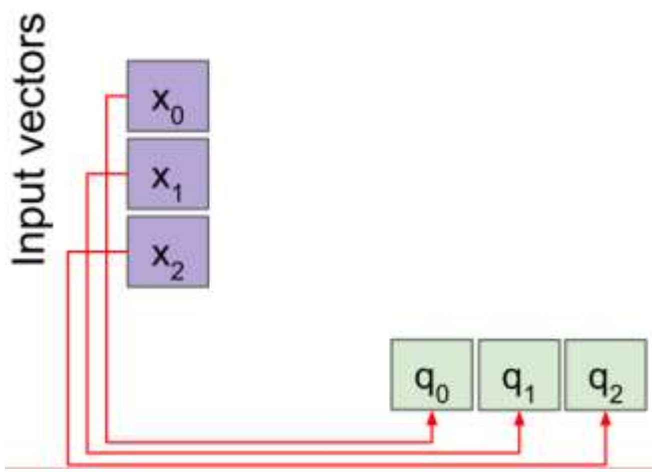


Inputs:
Query vectors: Q (Shape: $N_Q \times D_Q$)
Input vectors: X (Shape: $N_X \times D_X$)
Key matrix: W_K (Shape: $D_X \times D_Q$)
Value matrix: W_V (Shape: $D_X \times D_V$)

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

Self-attention Layer

- No input query vectors anymore
- Instead, query vectors are calculated using a FC layer.

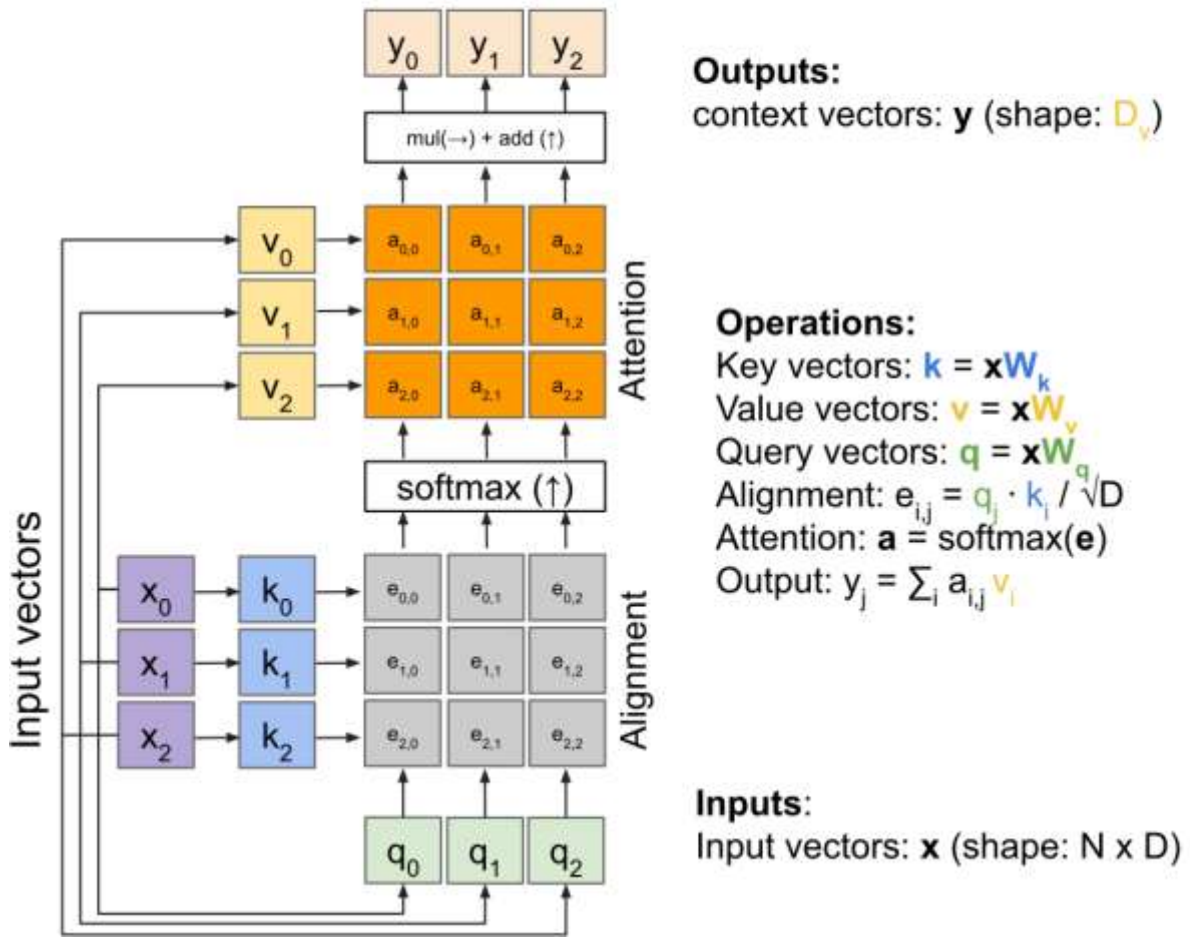


Operations:
Key vectors: $\mathbf{k} = \mathbf{x}W_k$
Value vectors: $\mathbf{v} = \mathbf{x}W_v$
$\boxed{\text{Query vectors: } \mathbf{q} = \mathbf{x}W_q \quad \text{Alignment: } e_{i,j} = q_j \cdot k_i / \sqrt{D}}$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $y_j = \sum_i a_{i,j} v_i$

Inputs:
Input vectors: $\mathbf{x}$ (shape: N x D)
~~Queries: **q** (shape: M x D_k)~~

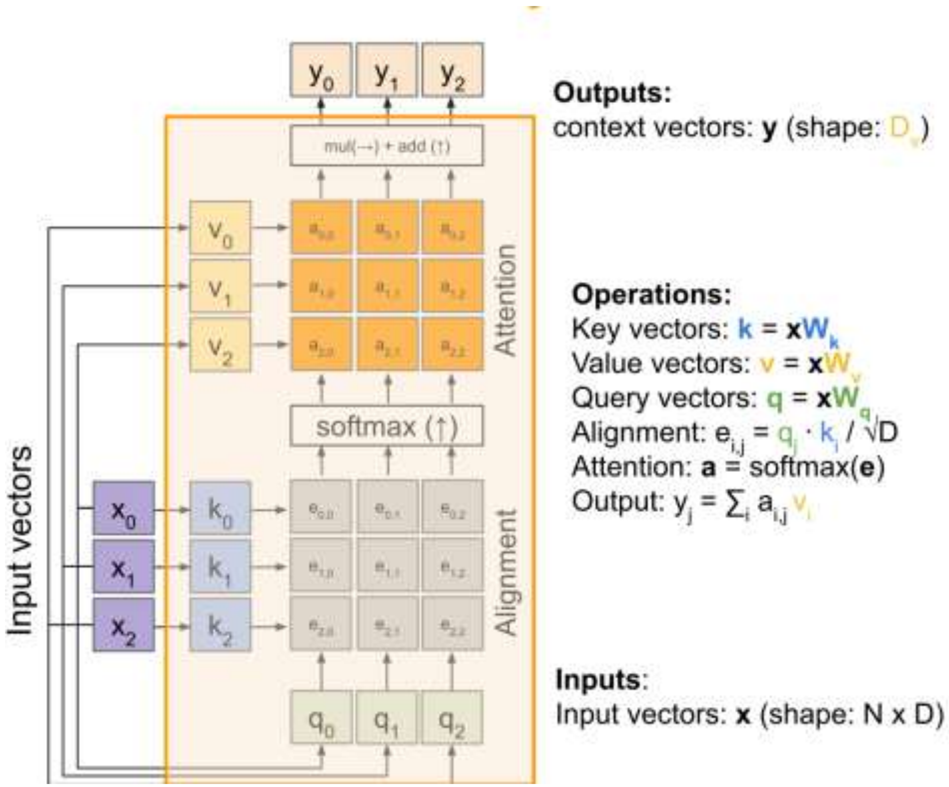
Self-attention Layer

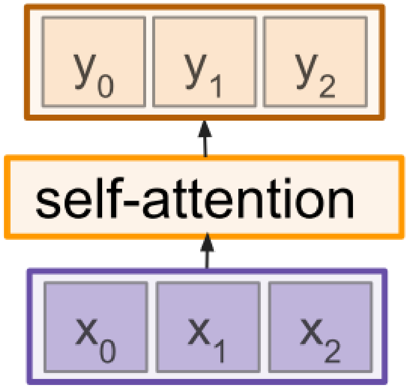
- Query vectors from the input vectors → Self-attention!



Self-attention Layer

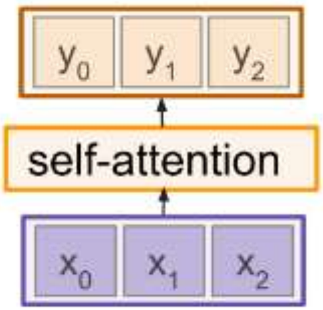
- Self-attention as a new mechanism for feature mapping

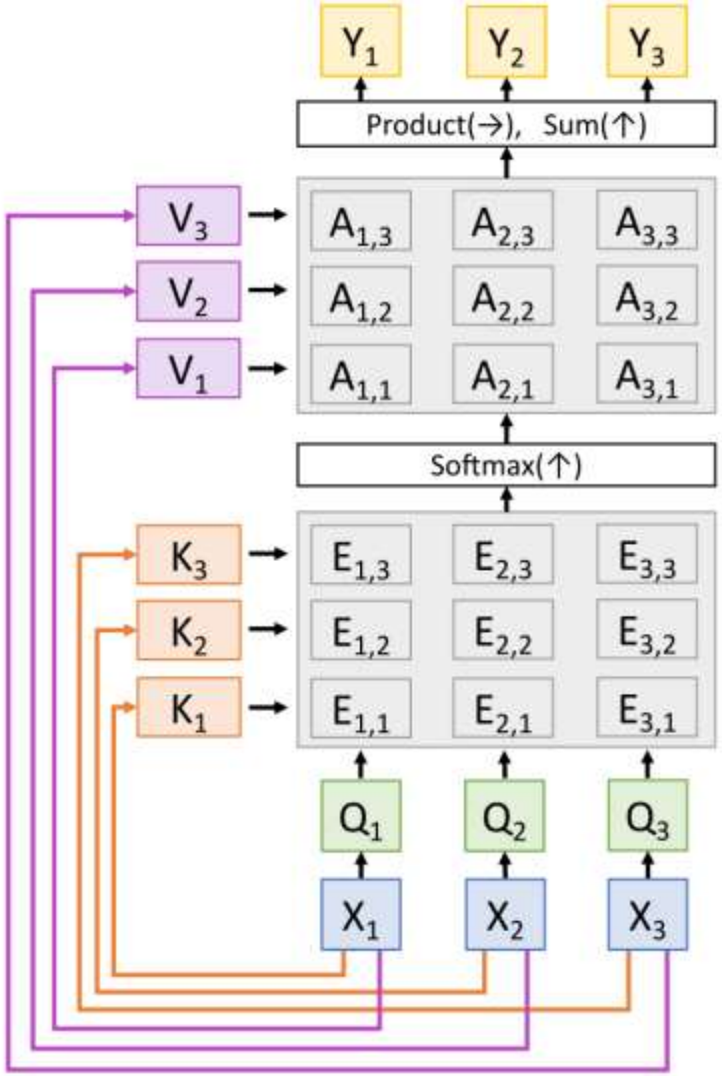




Self-attention Layer Summary

- One query per input vector



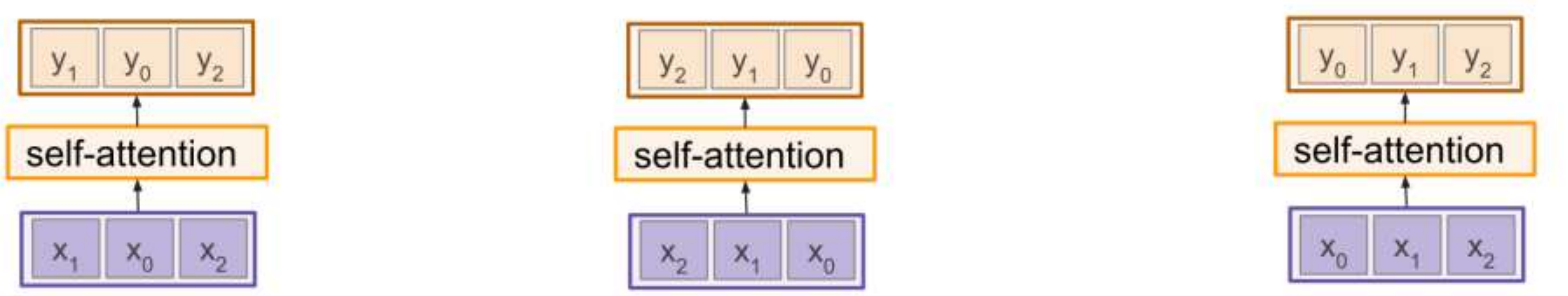


Inputs:
Input vectors: X (Shape: N_X x D_X)
Key matrix: W_K (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)
Query matrix: W_Q (Shape: D_X x D_Q)

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

Positional Encoding

- Self-attention doesn't "know" the order of the vectors it is processing!
- Permutation invariant!
- How can we encode ordered sequences like language or spatially ordered image features?



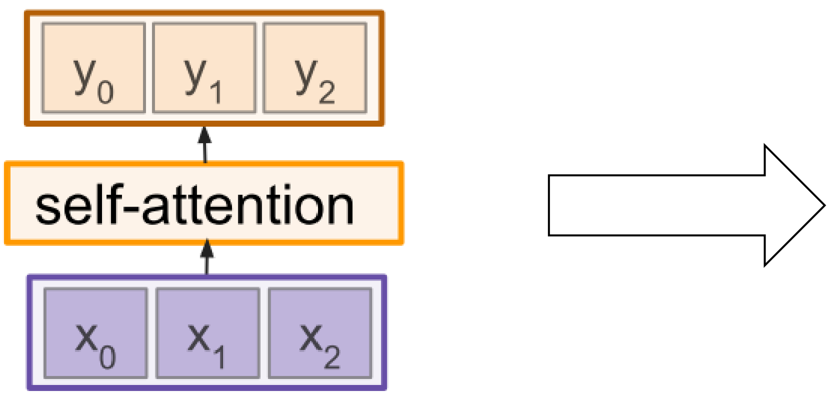
Positional Encoding

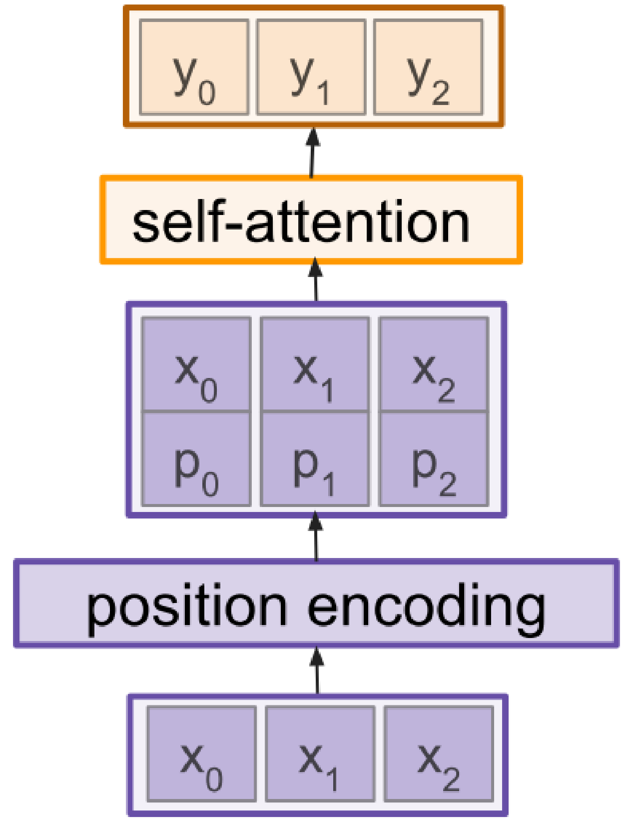
- Concatenate special positional encoding to each input vector

$$p_i = pos(j)$$

Desiderata of *pos*(.) :

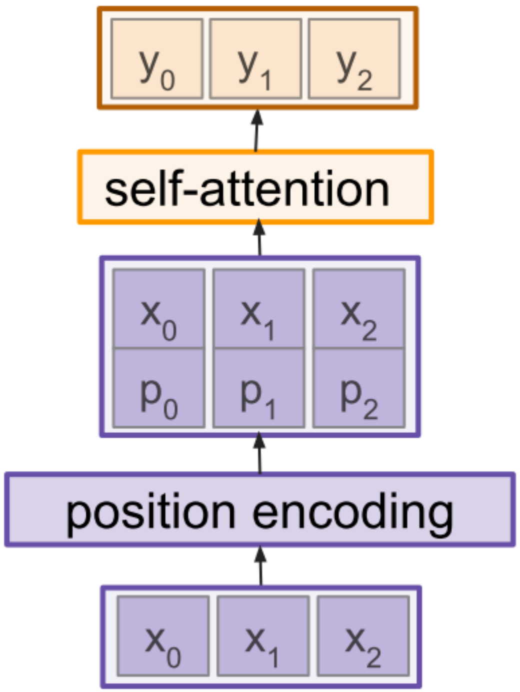
1. It should output a **unique** encoding for each time-step (word's position in a sentence)
2. **Distance** between any two time-steps should be consistent across sentences with different lengths.
3. Our model should generalize to **longer** sentences without any efforts. Its values should be bounded.
4. It must be **deterministic**.





Positional Encoding

- Learn a lookup table
- Design a fixed function with the desiderata



- Learn parameters to use for *pos*(t) for t ε [0, T)
- Lookup table contains T x d parameters.

$$p(t) = \begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}_d$$

where $\omega_k = \frac{1}{10000^{2k/d}}$

Intuition:

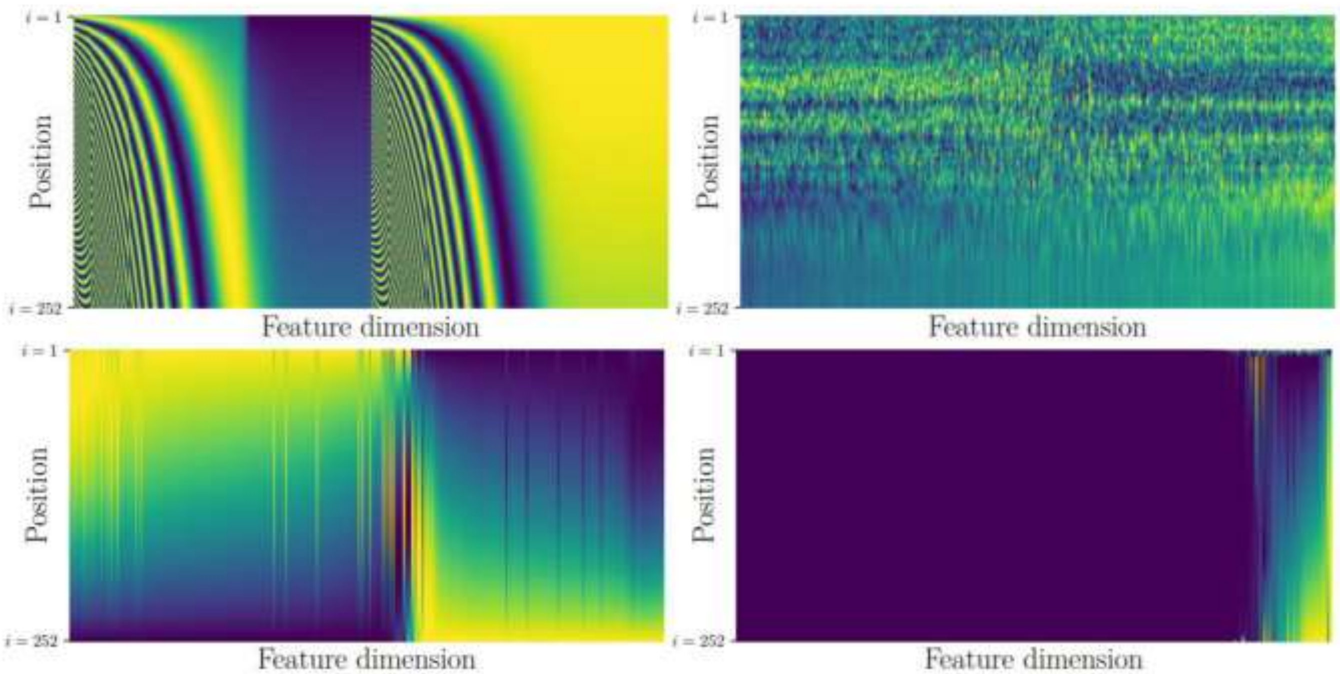
0 :	0	0	0	0	8 :	1	0	0	0
1 :	0	0	0	1	9 :	1	0	0	1
2 :	0	0	1	0	10 :	1	0	1	0
3 :	0	0	1	1	11 :	1	0	1	1
4 :	0	1	0	0	12 :	1	1	0	0
5 :	0	1	0	1	13 :	1	1	0	1
6 :	0	1	1	0	14 :	1	1	1	0
7 :	0	1	1	1	15 :	1	1	1	1

Positional Encoding

Table 1. Comparing position representation methods

Methods	Inductive	Data-Driven	Parameter Efficient
Sinusoidal (Vaswani et al., 2017)	✓	✗	✓
Embedding (Devlin et al., 2018)	✗	✓	✗
Relative (Shaw et al., 2018)	✗	✓	✓
This paper	✓	✓	✓

(a) Sinusoidal (b) Position embedding

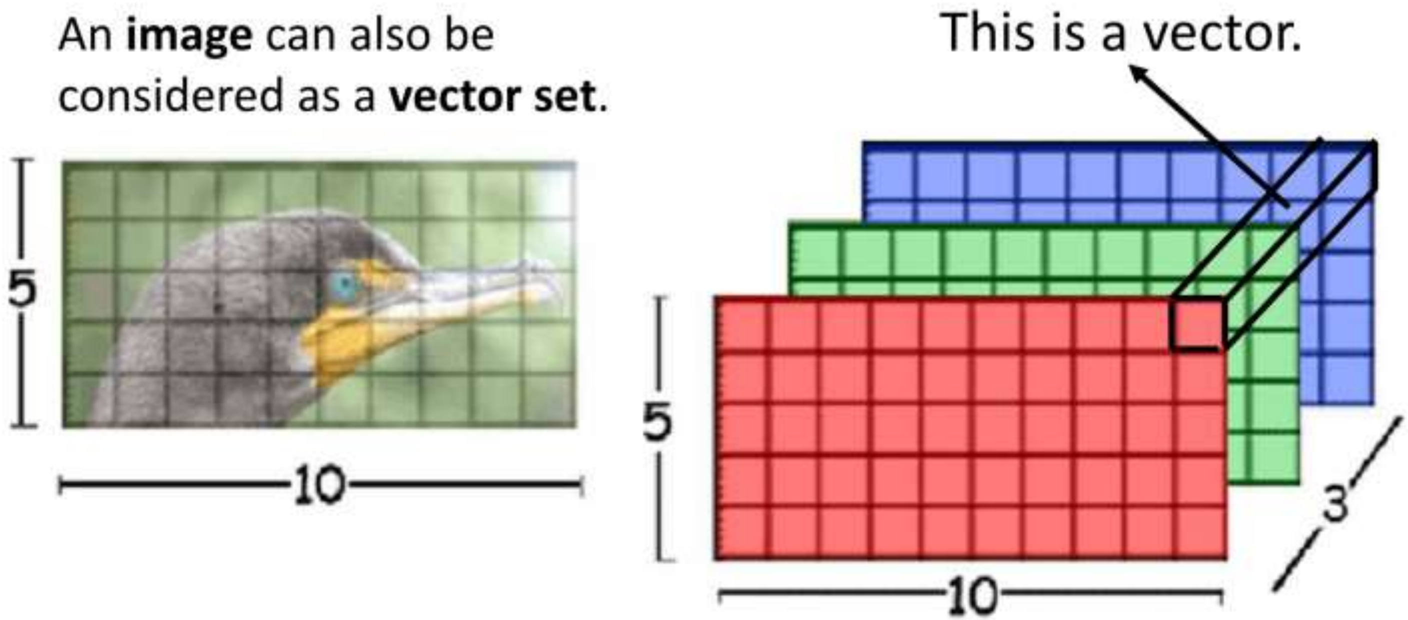


(c) FLOATER (d) RNN

Liu et al, "Learning to Encode Position for Transformer with Continuous Dynamical Model", ICML 2020

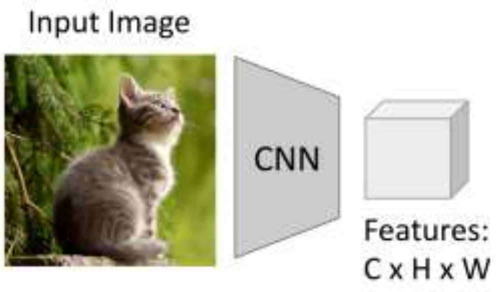
Self-attention & CNN

- CNN with Self-attention for feature mapping



Self-attention & CNN

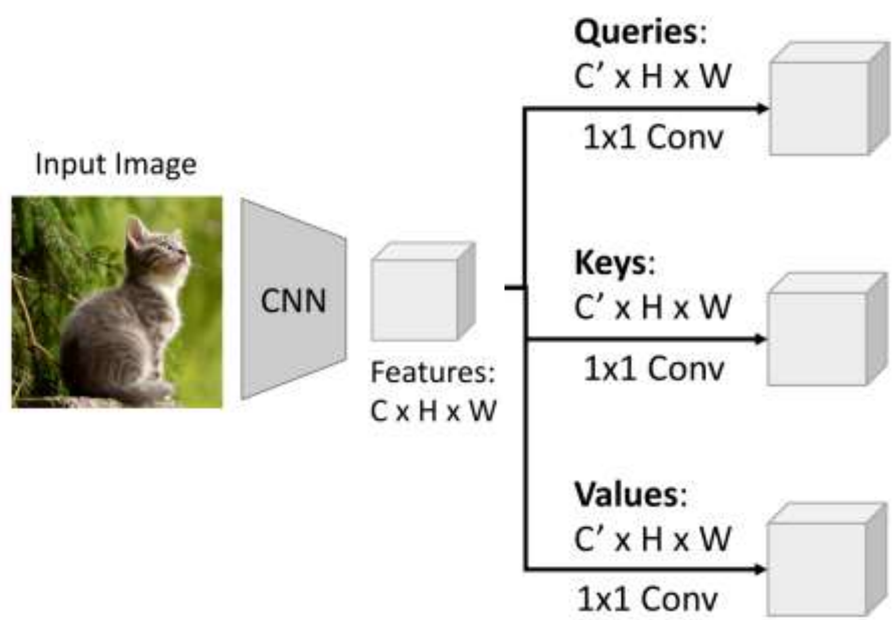
- CNN with Self-attention for feature mapping



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

Self-attention & CNN

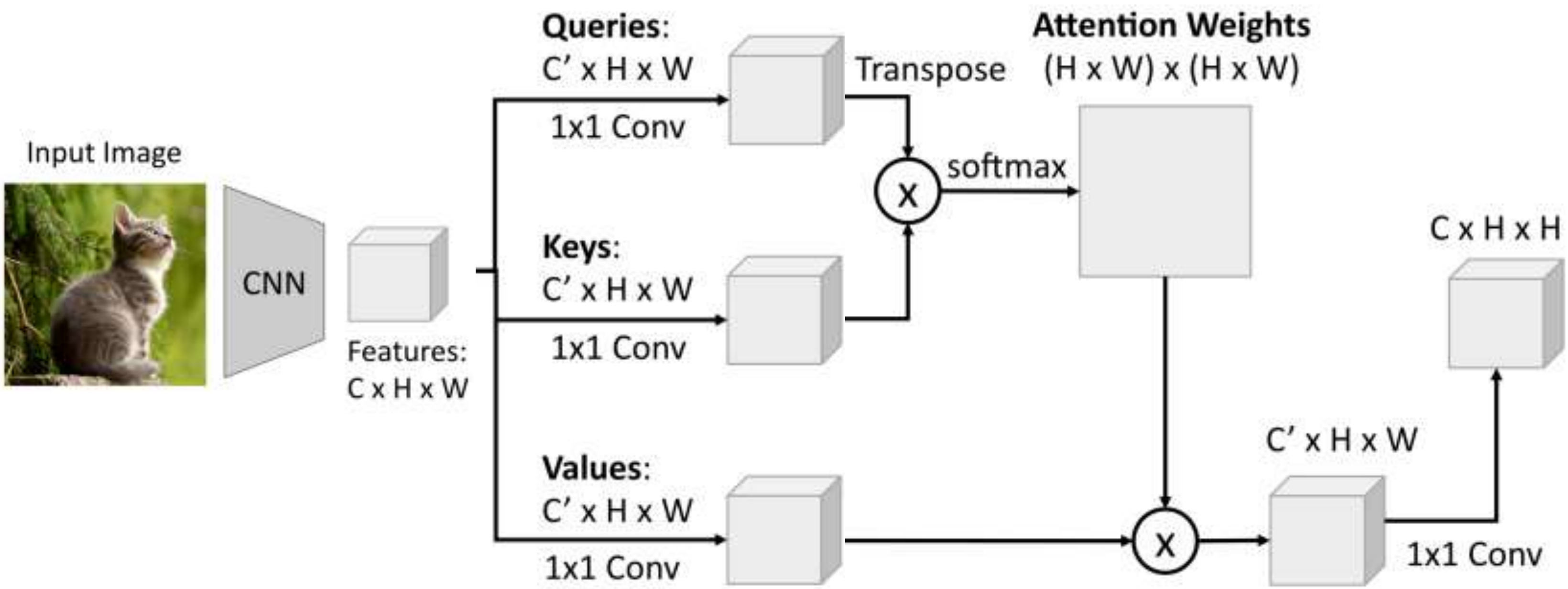
- CNN with Self-attention for feature mapping



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

Self-attention & CNN

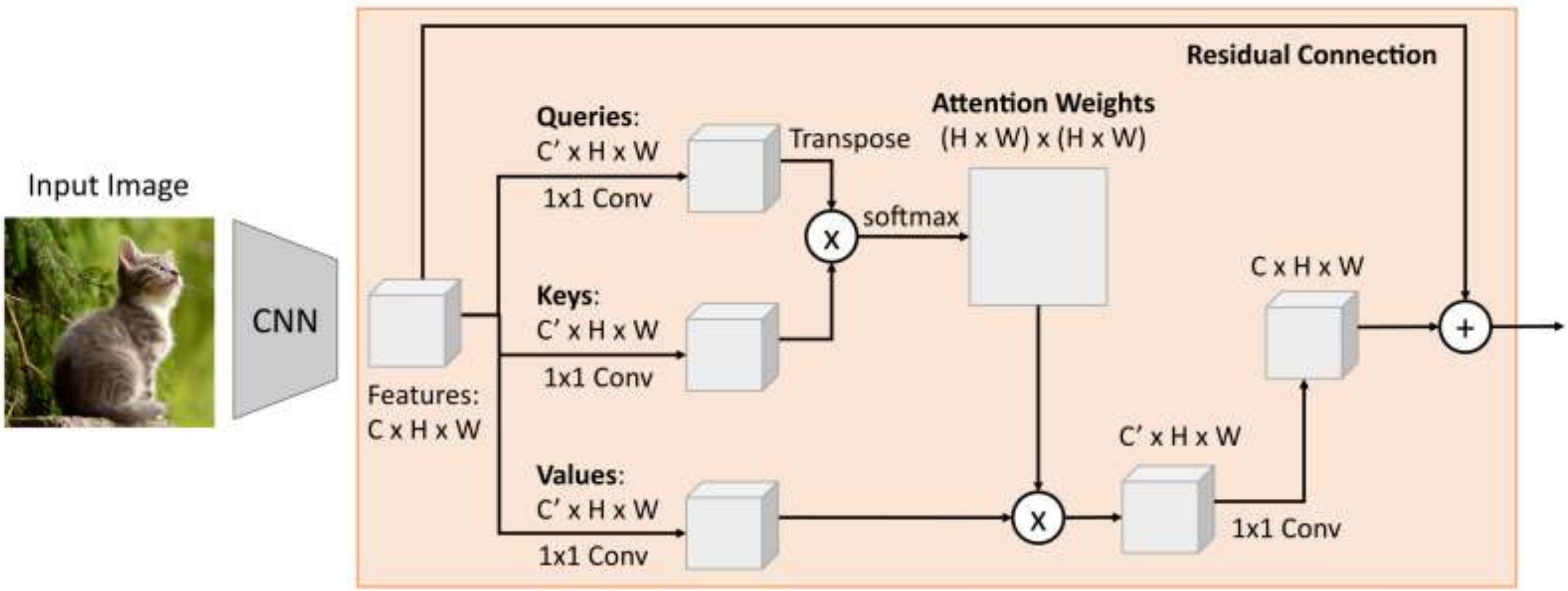
- CNN with Self-attention for feature mapping



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

Self-attention & CNN

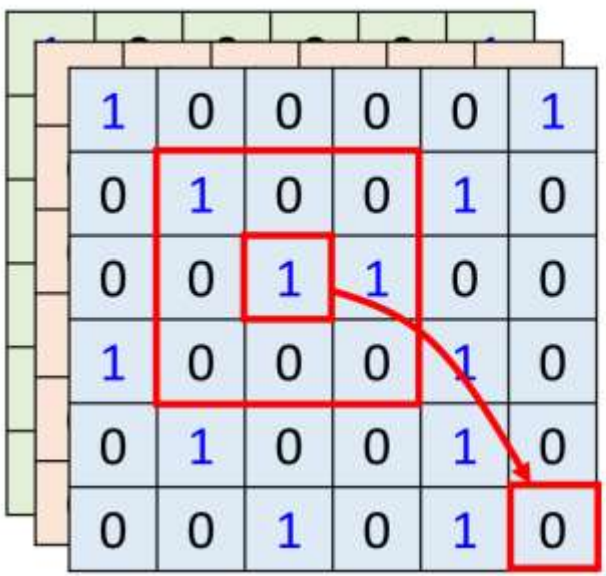
- Self-attention Module

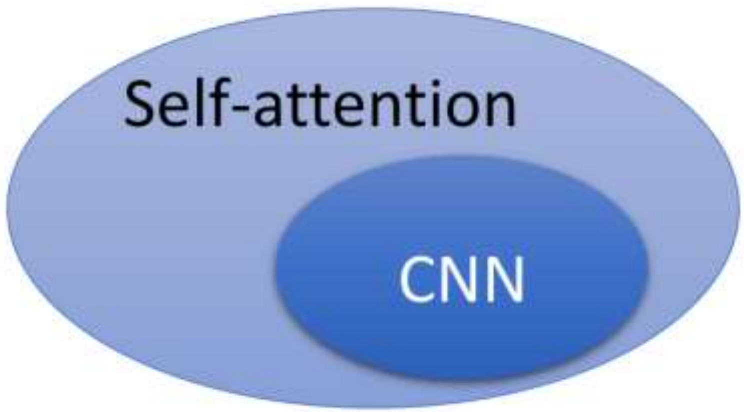


Zhang et al, "Self-Attention Generative Adversarial Networks", ICML 2018

Self-attention & CNN

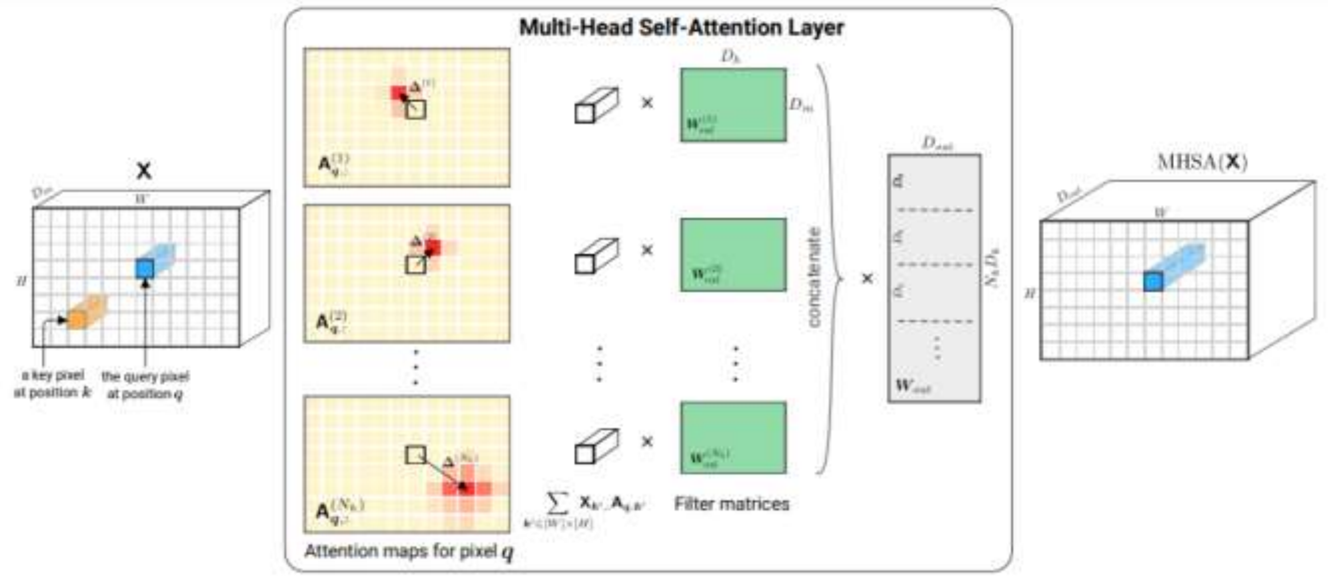
- CNN: self-attention that can only attends in a receptive field → CNN is simplified self-attention
- Self-attention: CNN with learnable receptive field → Self-attention is the complex version of CNN





Self-attention & CNN

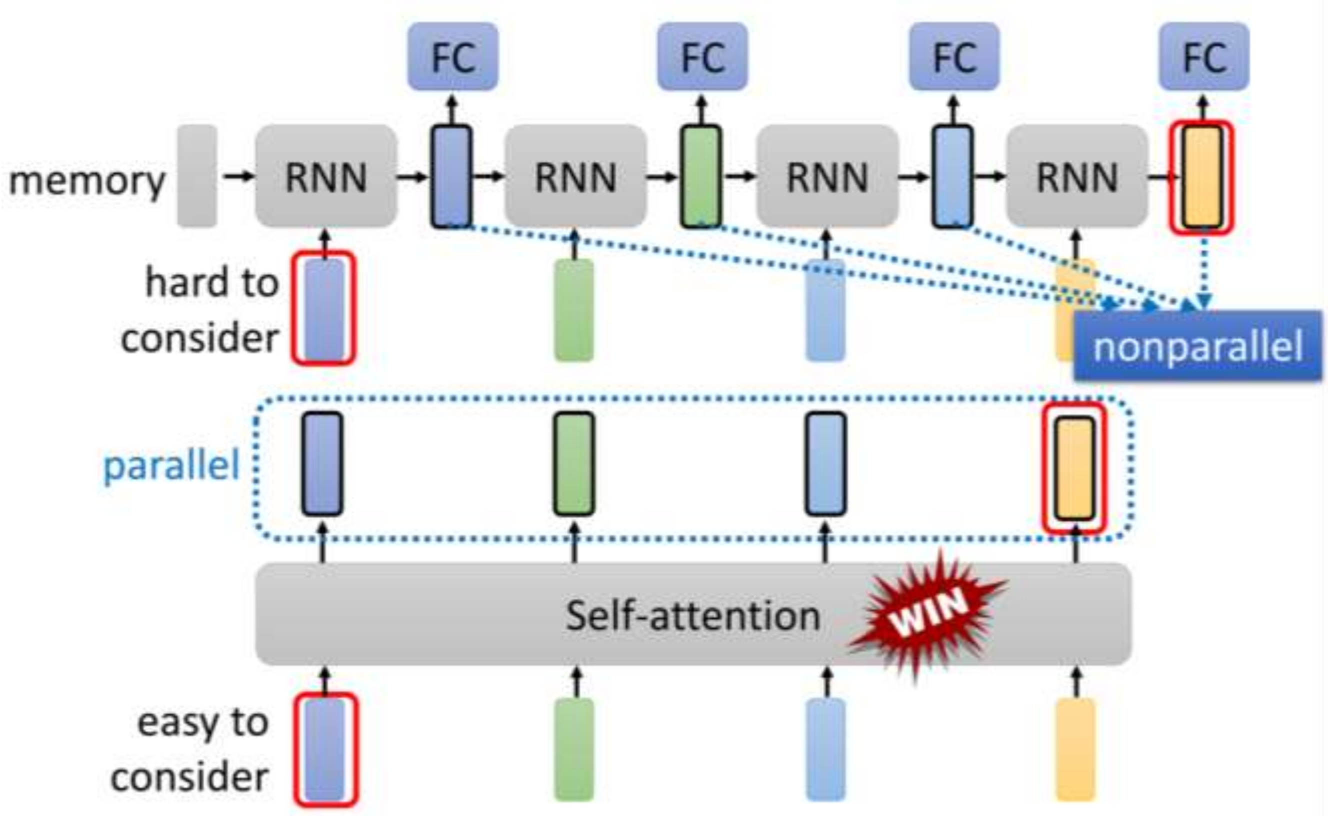
- CNN: self-attention that can only attends in a receptive field → CNN is simplified self-attention
- Self-attention: CNN with learnable receptive field → Self-attention is the complex version of CNN



Jean-Baptiste Cordonnier, Andreas Loukas, Martin Jaggi, “On the Relationship between Self-Attention and Convolutional Layers”, ICLR 2020

Self-attention & RNN

- Handle sequence data in a parallel / nonparallel manner



Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, François Fleuret,
"Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention", ICML 2020

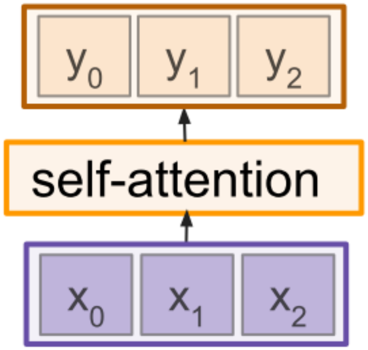
Outline

- Transformer
  - Encoder-Decoder Artichecture
  - Transfer Learning and Vision Transformer

Acknowledgement: Feifei Li et al’s cs231n notes

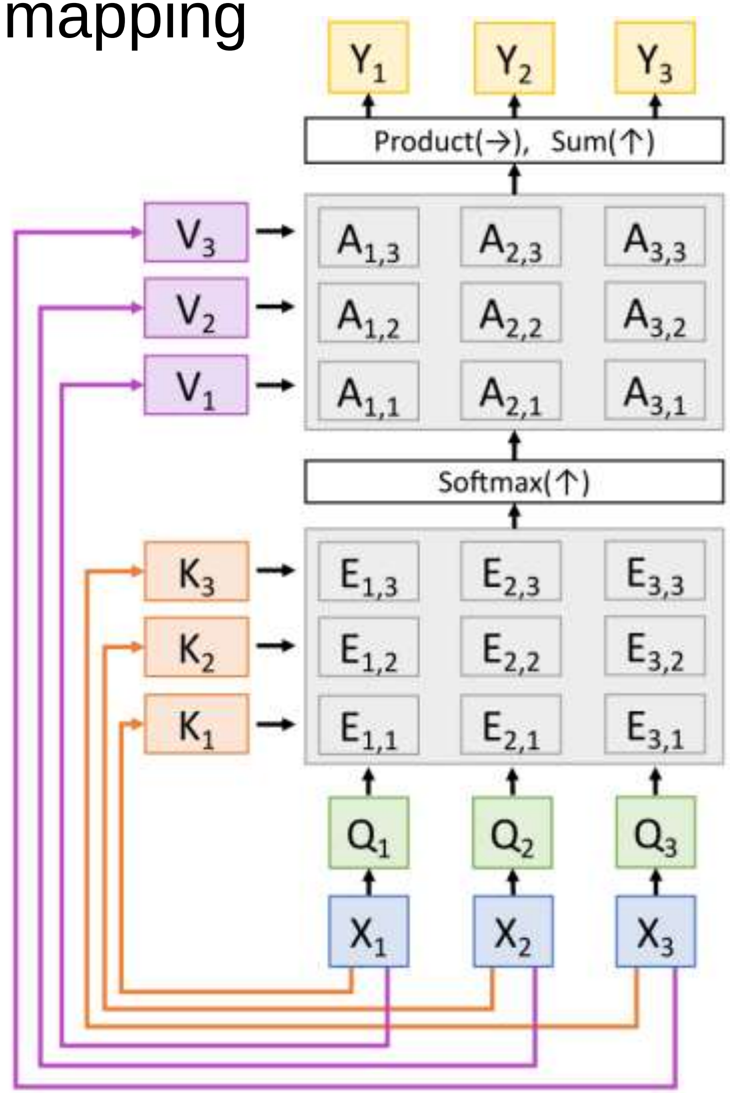
Self-attention Layer Summary

- A new mechanism for feature mapping



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

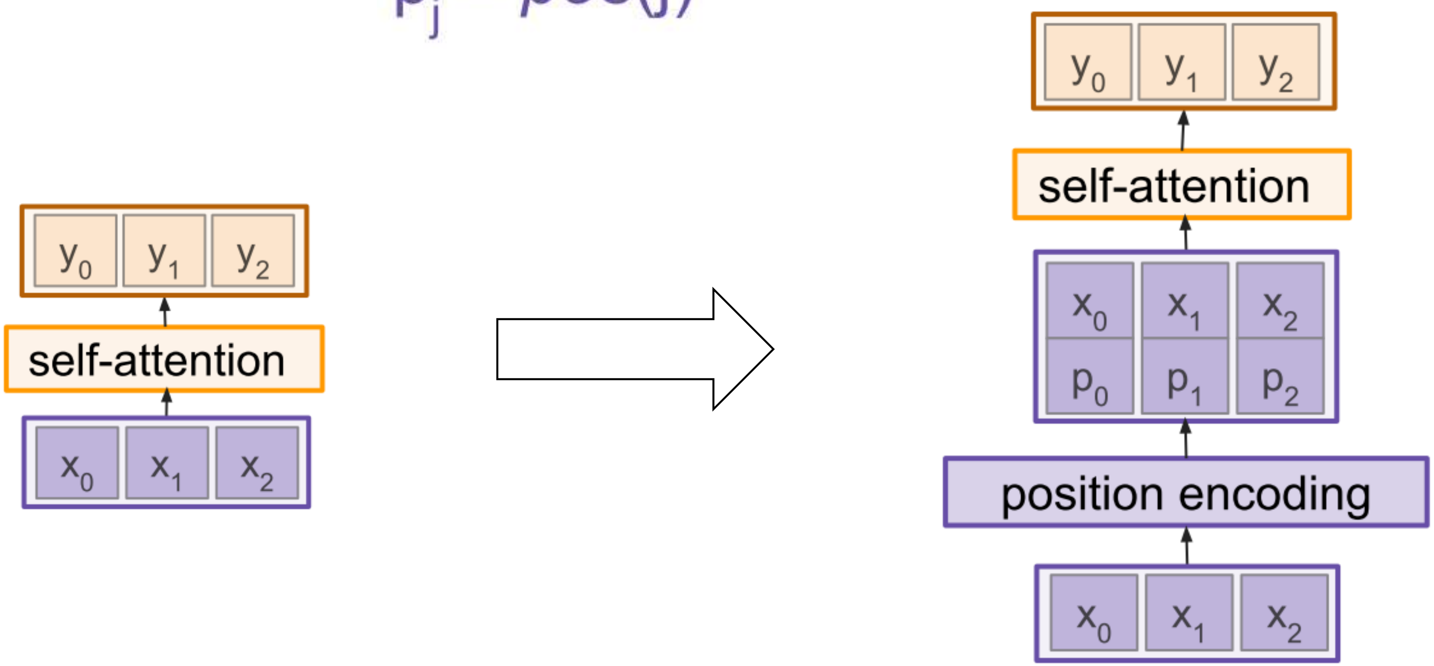
<u>**Computation**</u>:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value Vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T$ (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j / \mathrm{sqrt}(D_Q)$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: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



Self-attention Layer Summary

- Positional Encoding so solve the order-ambiguity
- Concatenate special positional encoding to each input vector

$$p_j = pos(j)$$

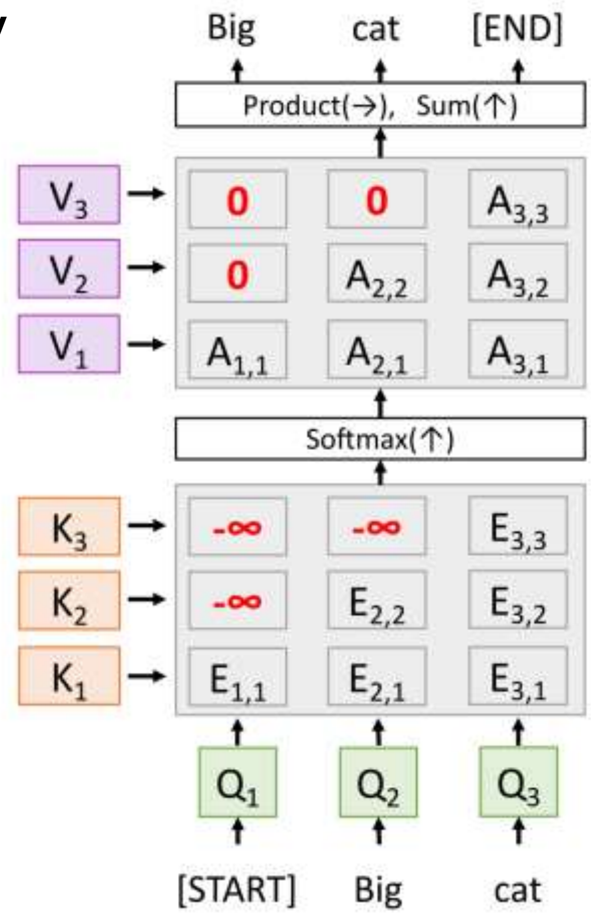


Masked Self-Attention

- Don't let vectors "look ahead" in the sequence
- Used for language modeling (predict next word)
- Set alignment scores to -infinity

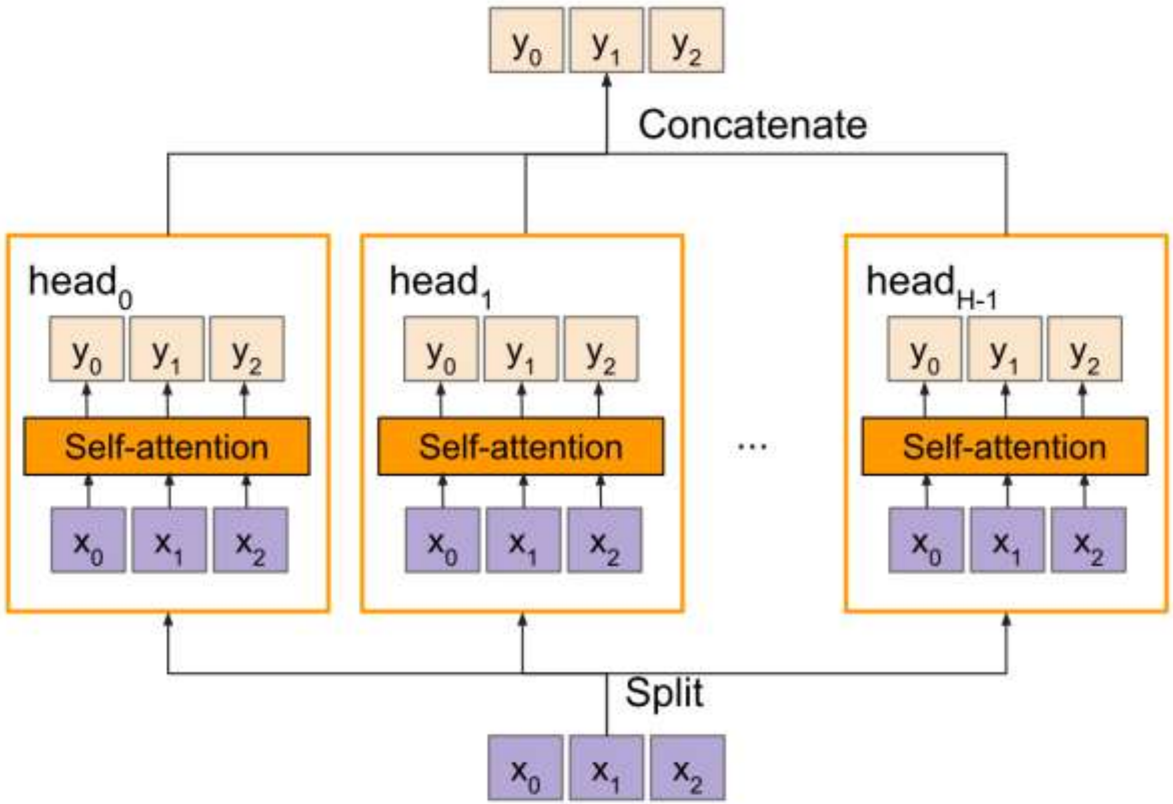
Inputs:
Input vectors: X (Shape: $N_X \times D_X$)
Key matrix: W_K (Shape: $D_X \times D_Q$)
Value matrix: W_V (Shape: $D_X \times D_V$)
Query matrix: W_Q (Shape: $D_X \times D_Q$)

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



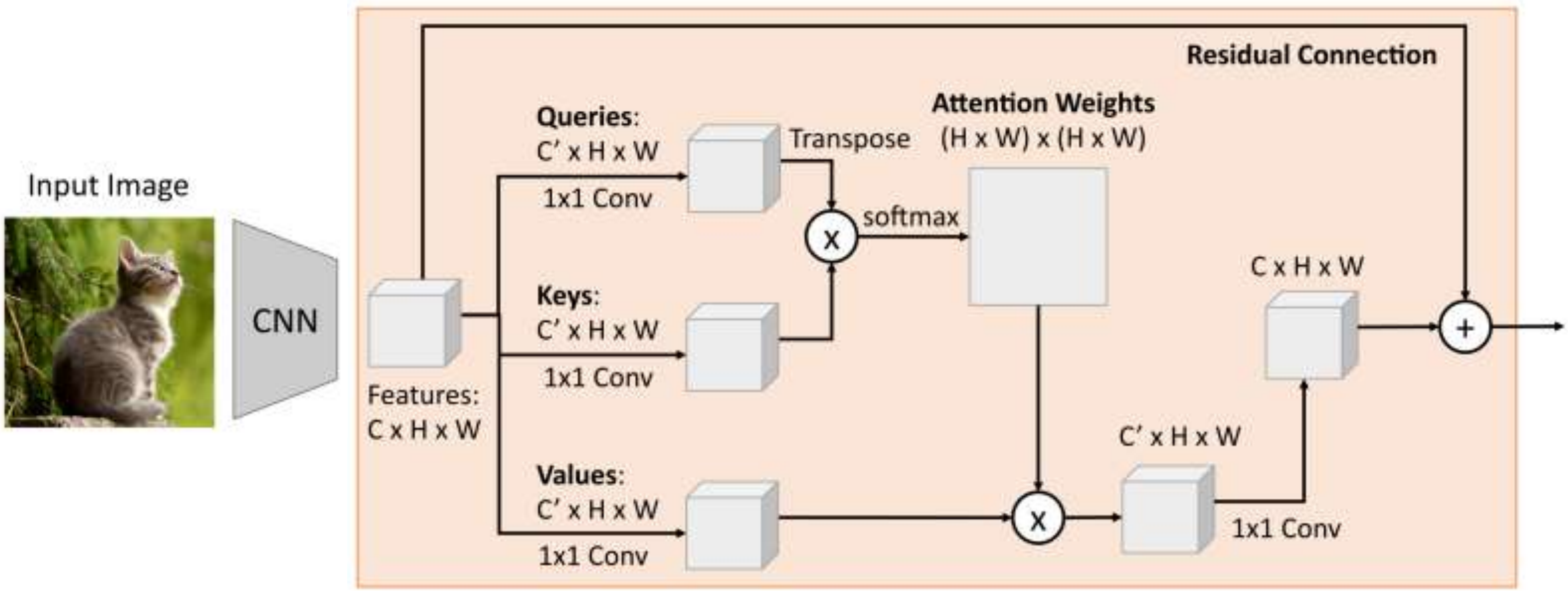
Multihead Self-Attention

- Multiple self-attention heads in parallel
- Hyperparameters: Query dimension and Number of heads



Self-attention for feature mapping

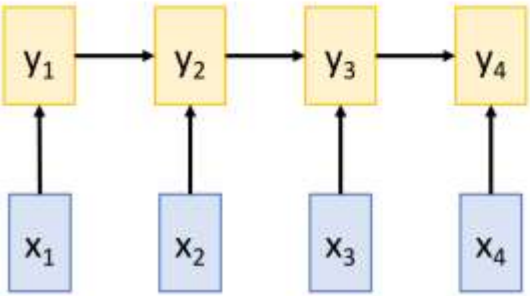
- Self-attention Module



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

Three Ways of Processing Sequences

Recurrent Neural Network

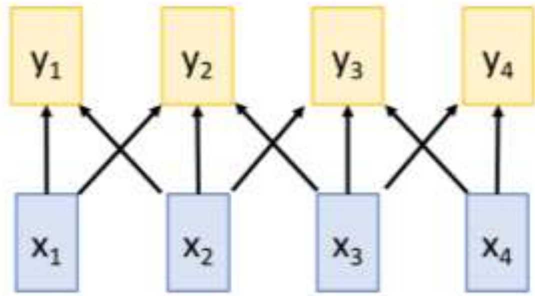


Works on **Ordered Sequences**

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

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

1D Convolution

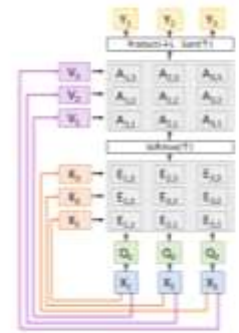


Works on **Multidimensional Grids**

(-) Bad at long sequences: Need to stack many conv layers for outputs to "see" the whole sequence

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

Self-Attention



Works on **Sets of Vectors**

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

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

(-) Very memory intensive

Three Ways of Processing Sequences

Recurrent Neural Network

1D Convolution

Self-Attention

y1

x1

Attention is all you need

Vaswani et al, NeurIPS 2017

Work

(+) G

one

sequ

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

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

er one

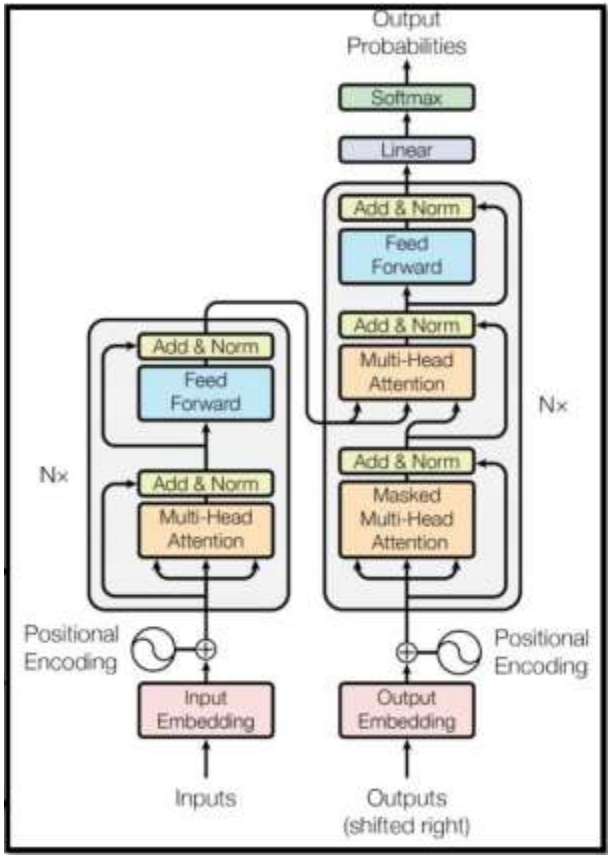
it

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

(-) Very memory intensive

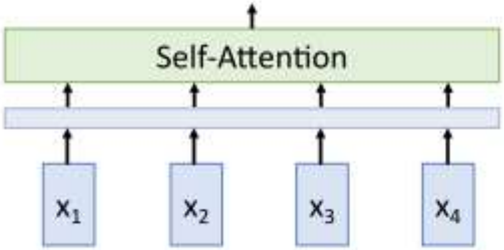
Transformer

- A new block type in term of encoder-decoder
- Attention only !

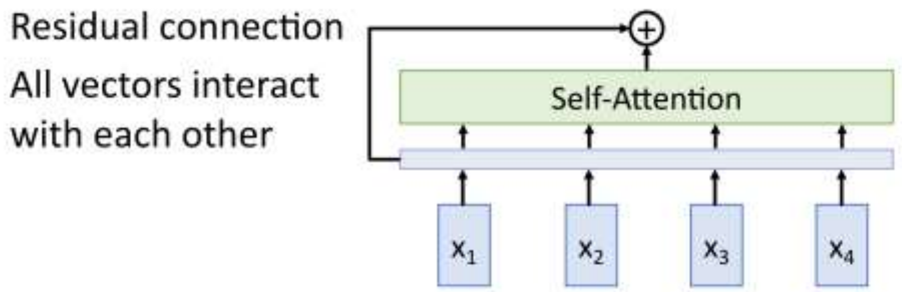


The Transformer Block

All vectors interact with each other



The Transformer Block



The Transformer Block

Recall **Layer Normalization**:

Given $h_1, \ldots, h_N$ (Shape: D)

scale: γ (Shape: D)

shift: β (Shape: D)

$\mu_i = (1/D)\sum_j h_{i,j}$ (scalar)

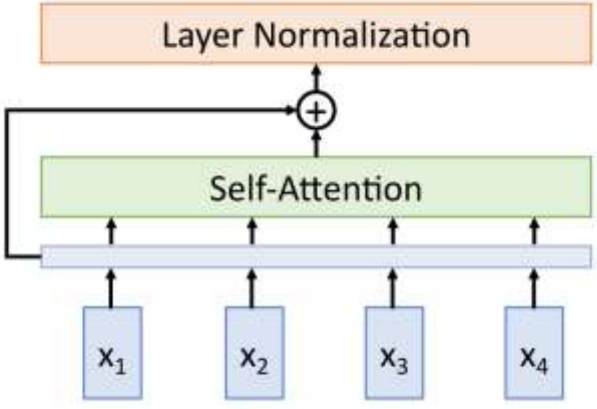
$\sigma_i = (\sum_j (h_{i,j} - \mu_i)^2)^{1/2}$ (scalar)

$z_i = (h_i - \mu_i) / \sigma_i$

$y_i = \gamma * z_i + \beta$

Residual connection

All vectors interact with each other



The Transformer Block

Recall **Layer Normalization**:

Given $h_1, \ldots, h_N$ (Shape: D)

scale: γ (Shape: D)

shift: β (Shape: D)

$\mu_i = (1/D)\sum_j h_{i,j}$ (scalar)

$\sigma_i = (\sum_j (h_{i,j} - \mu_i)^2)^{1/2}$ (scalar)

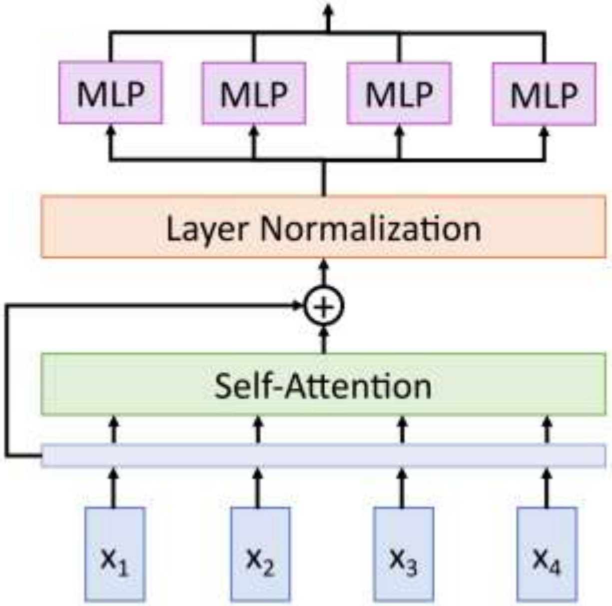
$z_i = (h_i - \mu_i) / \sigma_i$

$y_i = \gamma * z_i + \beta$

MLP independently on each vector

Residual connection

All vectors interact with each other



The Transformer Block

Recall **Layer Normalization**:

Given $h_1, \ldots, h_N$ (Shape: D)

scale: γ (Shape: D)

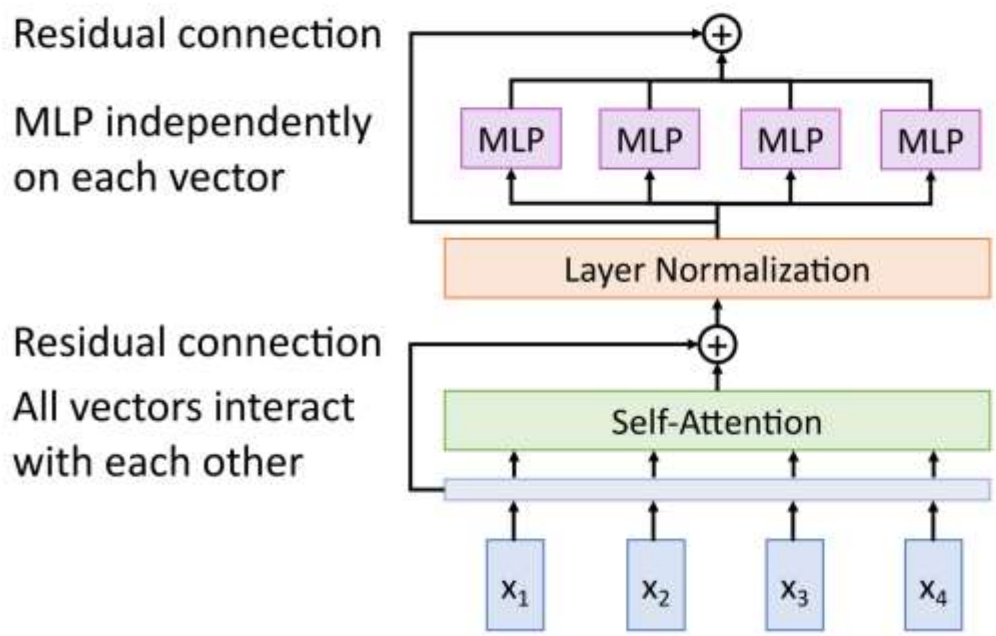
shift: β (Shape: D)

$\mu_i = (1/D)\sum_j h_{i,j}$ (scalar)

$\sigma_i = (\sum_j (h_{i,j} - \mu_i)^2)^{1/2}$ (scalar)

$z_i = (h_i - \mu_i) / \sigma_i$

$y_i = \gamma * z_i + \beta$



The Transformer Block

Recall **Layer Normalization**:

Given $h_1, \ldots, h_N$ (Shape: D)

scale: γ (Shape: D)

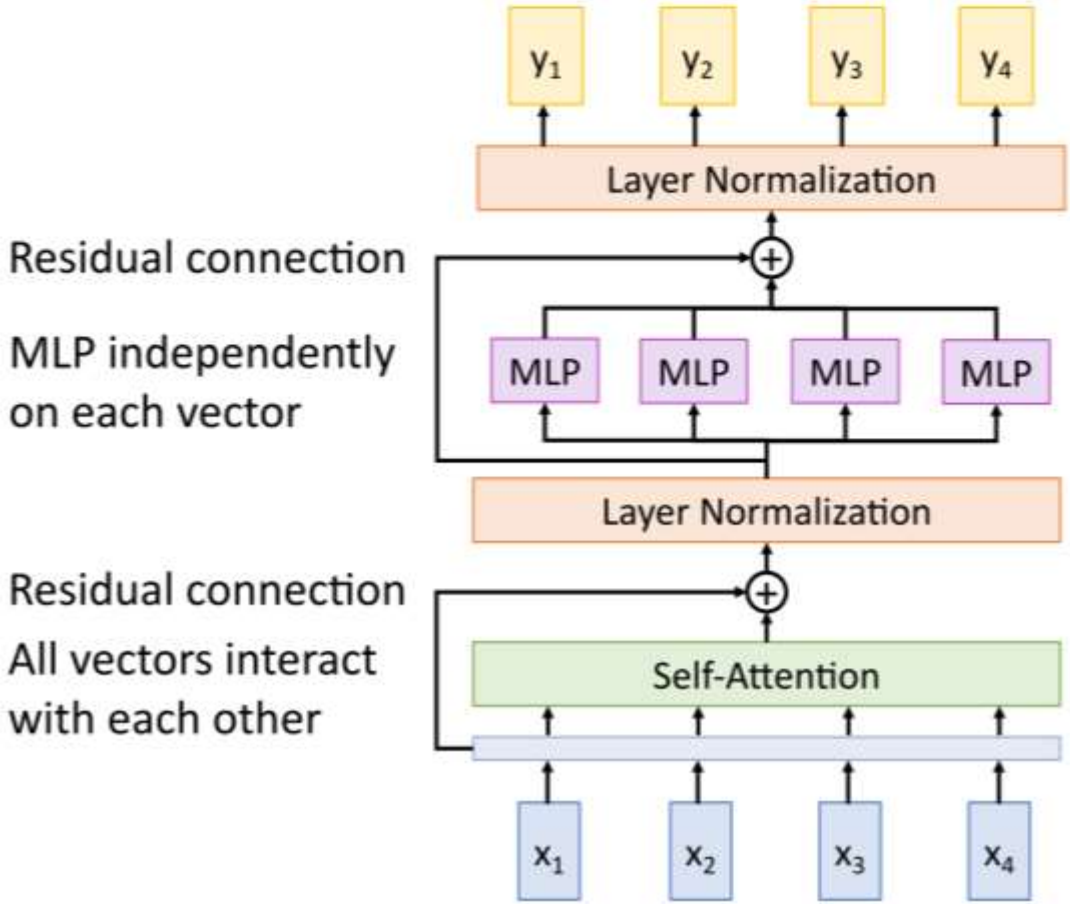
shift: β (Shape: D)

$\mu_i = (1/D)\sum_j h_{i,j}$ (scalar)

$\sigma_i = (\sum_j (h_{i,j} - \mu_i)^2)^{1/2}$ (scalar)

$z_i = (h_i - \mu_i) / \sigma_i$

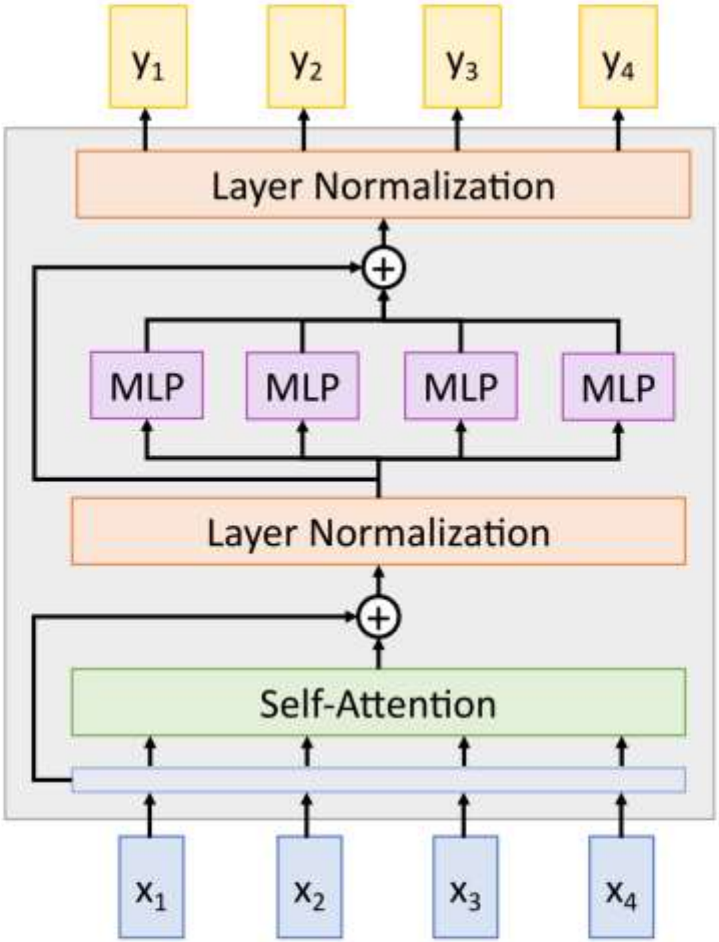
$y_i = \gamma * z_i + \beta$



The Transformer Block

- Block Summary
- Input: Set of vectors x
- Output: Set of vectors y

- Self-attention is the only interaction between vectors!
- Layer norm and MLP work independently per vector
- Highly scalable,
- highly parallelizable

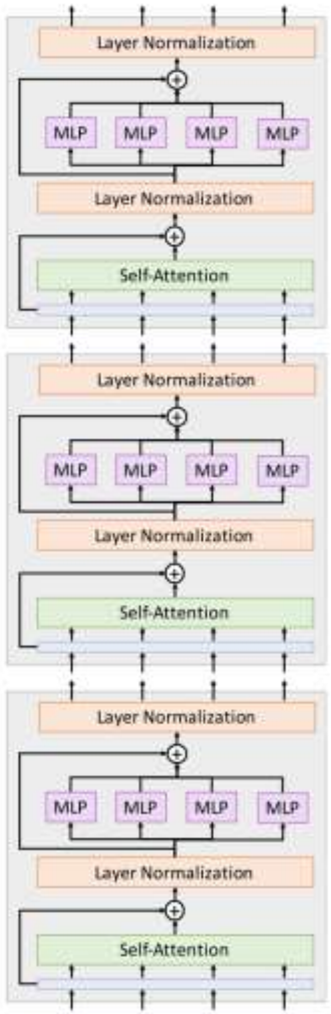


The Transformer

- A sequence of transformer blocks
- Input: Set of vectors x
- Output: Set of vectors y

- Self-attention is the only interaction between vectors!
- Layer norm and MLP work independently per vector
- Highly scalable,
- highly parallelizable

- ImageNet Moment for NLP



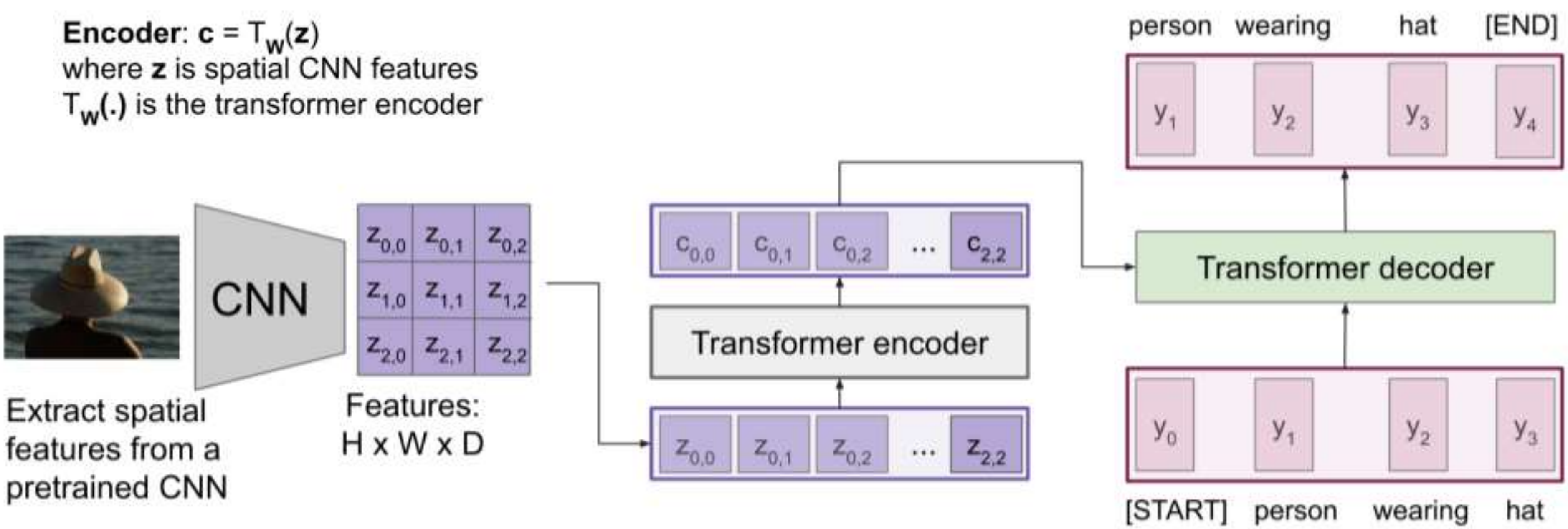
The Transformer

- Recall the Image Captioning task
- Transformer in terms of encoder-decoder
- No recurrence at all!

Input: Image **I**
Output: Sequence $\mathbf{y} = y_1, y_2, \ldots, y_T$

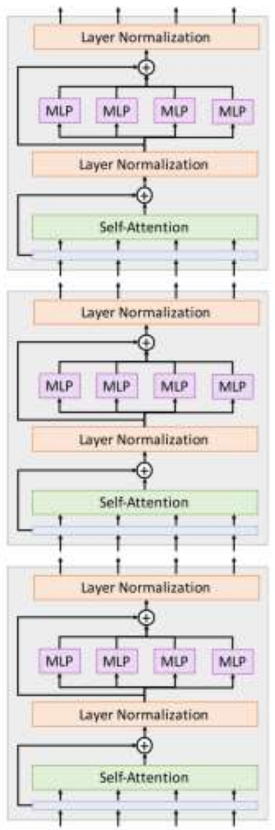
Decoder: $y_t = T_D(\mathbf{y}_{0:t-1}, \mathbf{c})$
where $T_D(.)$ is the transformer decoder

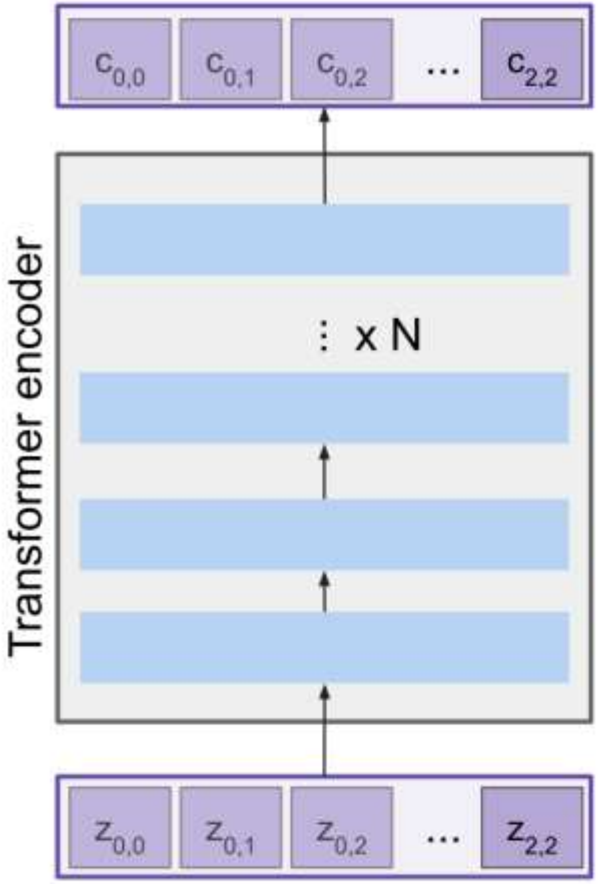
Encoder: $\mathbf{c} = T_W(\mathbf{z})$
where **z** is spatial CNN features
$T_W(.)$ is the transformer encoder



The Transformer encoder block

- Made up of N encoder blocks
- In vaswani et al. N = 6, D

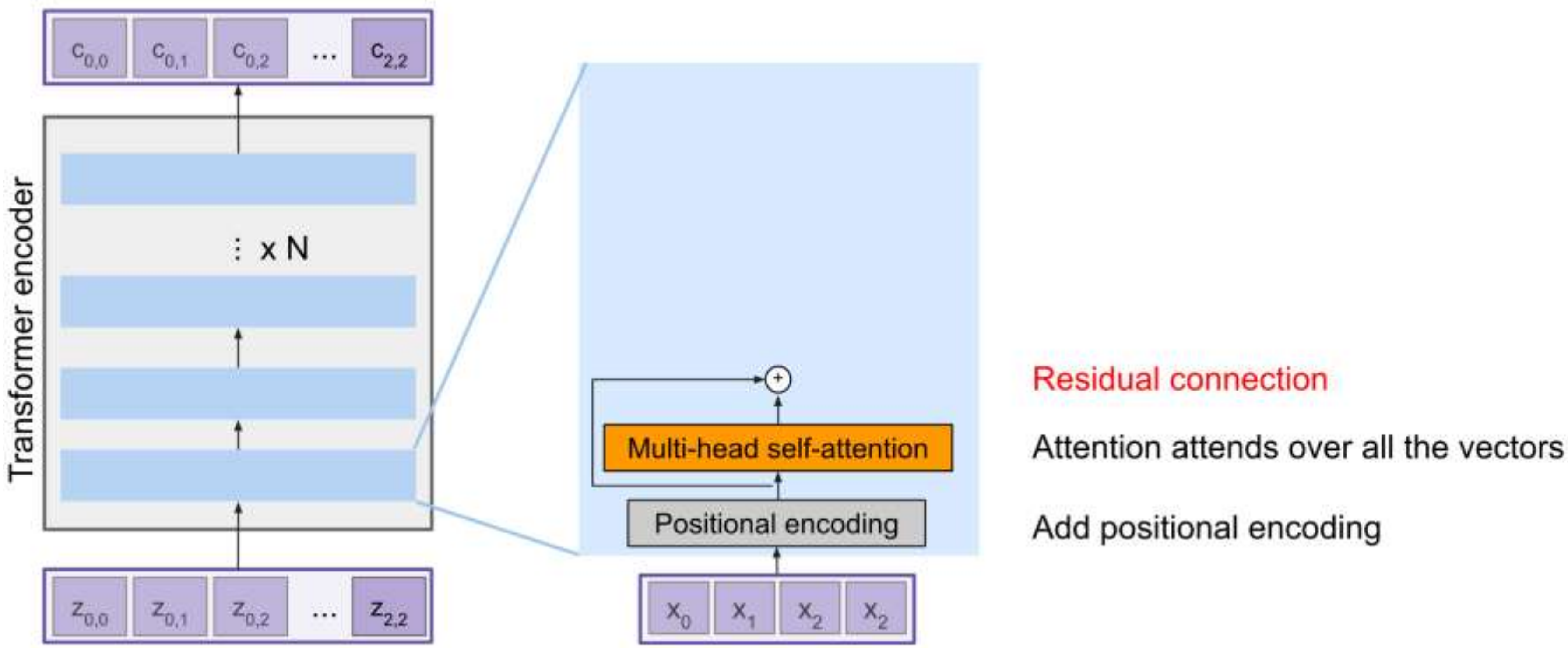




Vaswani et. al, "Attention is all you need", NeurIPS 2017

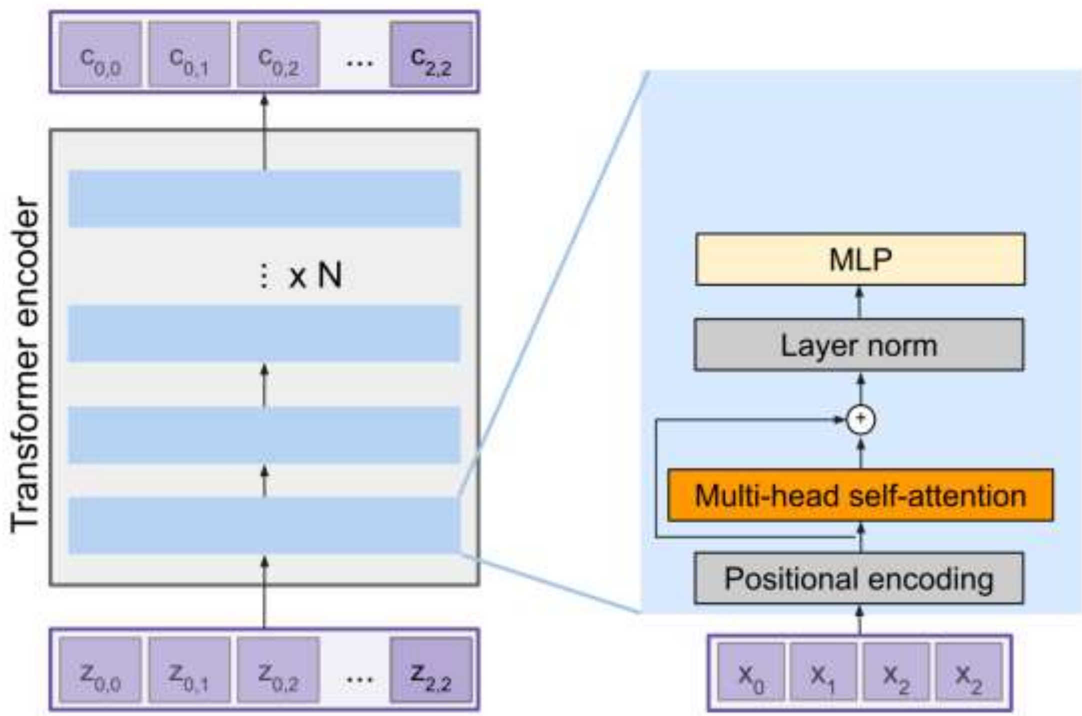
The Transformer encoder block

- Made up of N encoder blocks



The Transformer encoder block

- Made up of N encoder blocks



MLP over each vector individually

LayerNorm over each vector individually

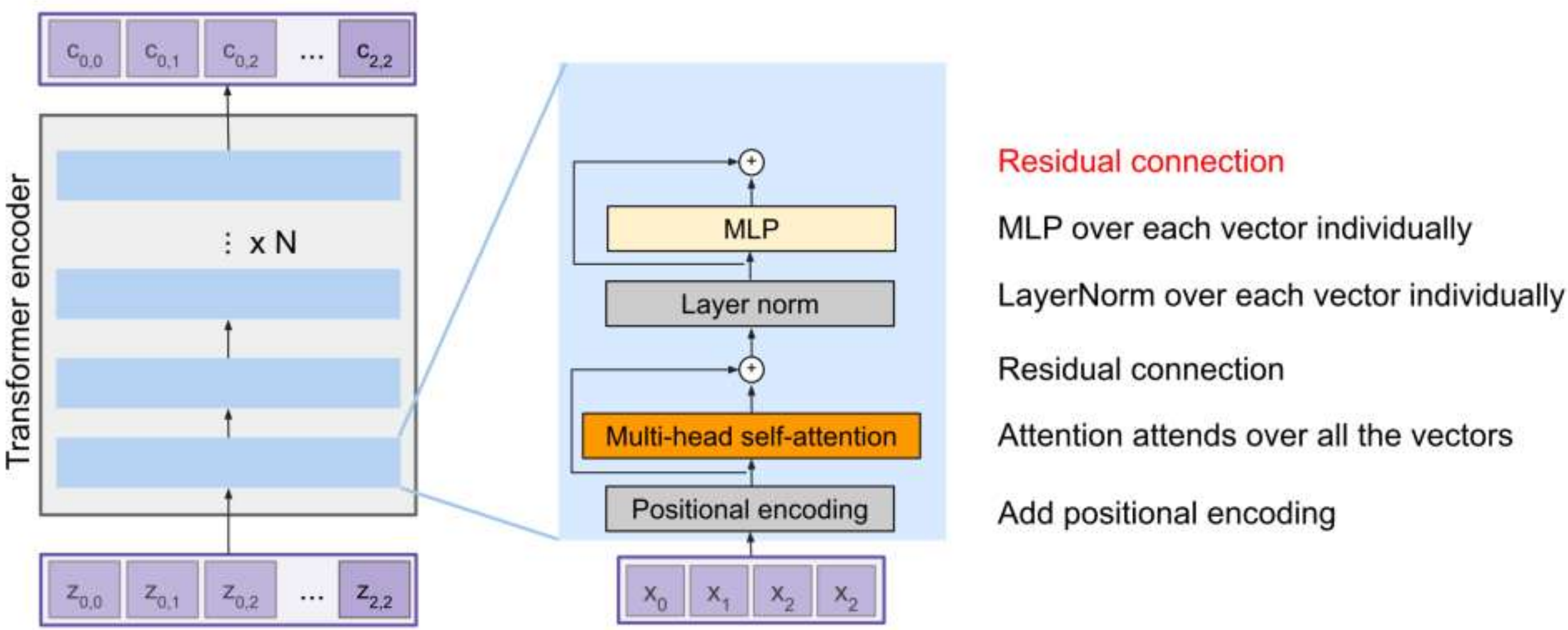
Residual connection

Attention attends over all the vectors

Add positional encoding

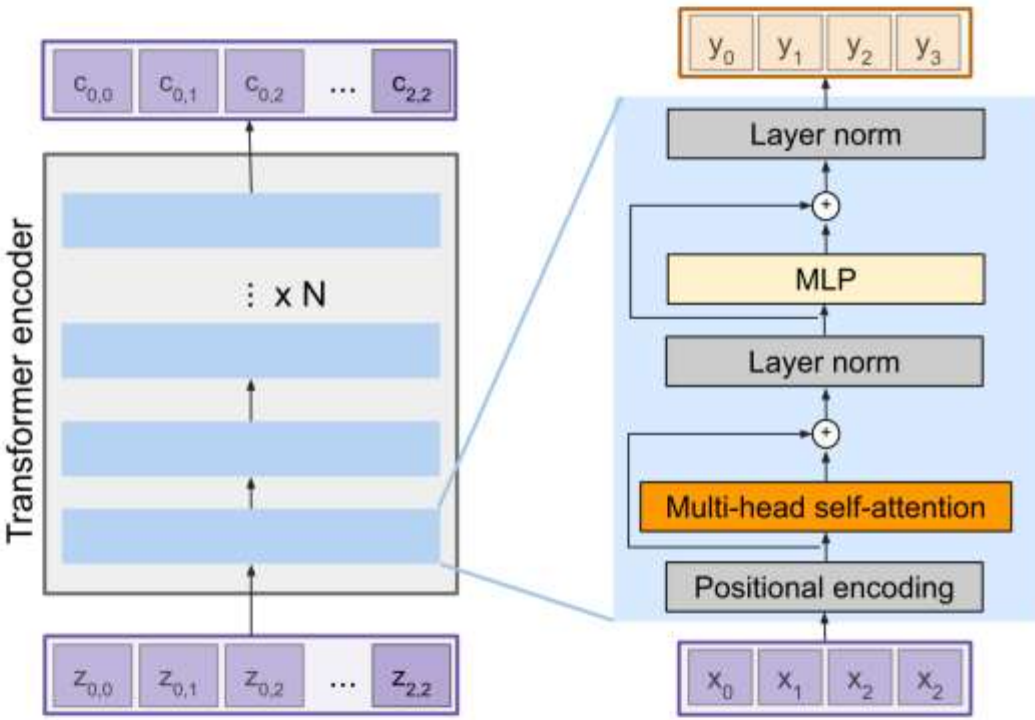
The Transformer encoder block

- Made up of N encoder blocks



The Transformer encoder block

- Made up of N encoder blocks



Transformer Encoder Block:

Inputs: Set of vectors **x**
Outputs: Set of vectors **y**

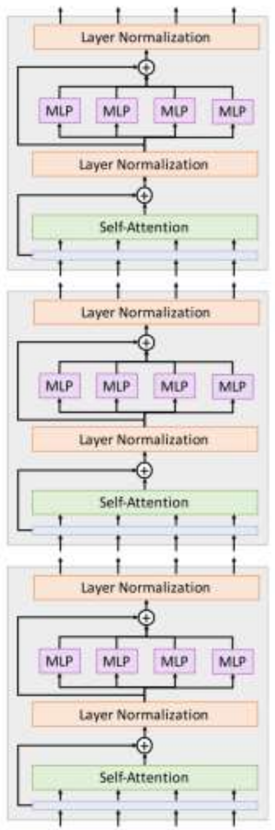
Self-attention is the only interaction between vectors.

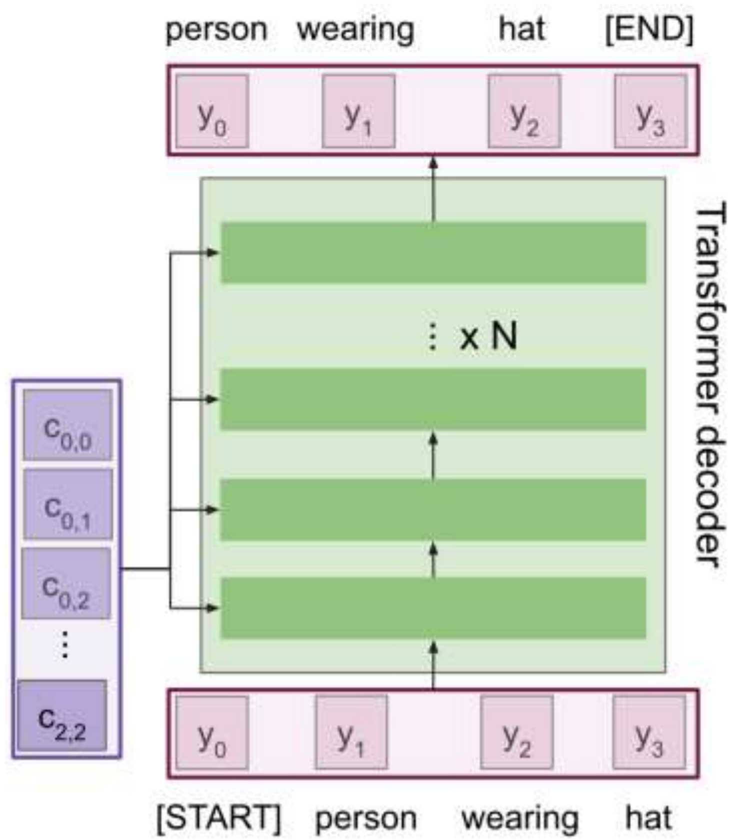
Layer norm and MLP operate independently per vector.

Highly scalable, highly parallelizable, but high memory usage.

The Transformer decoder block

- Made up of N decoder blocks
- In vaswani et al. N = 6, D

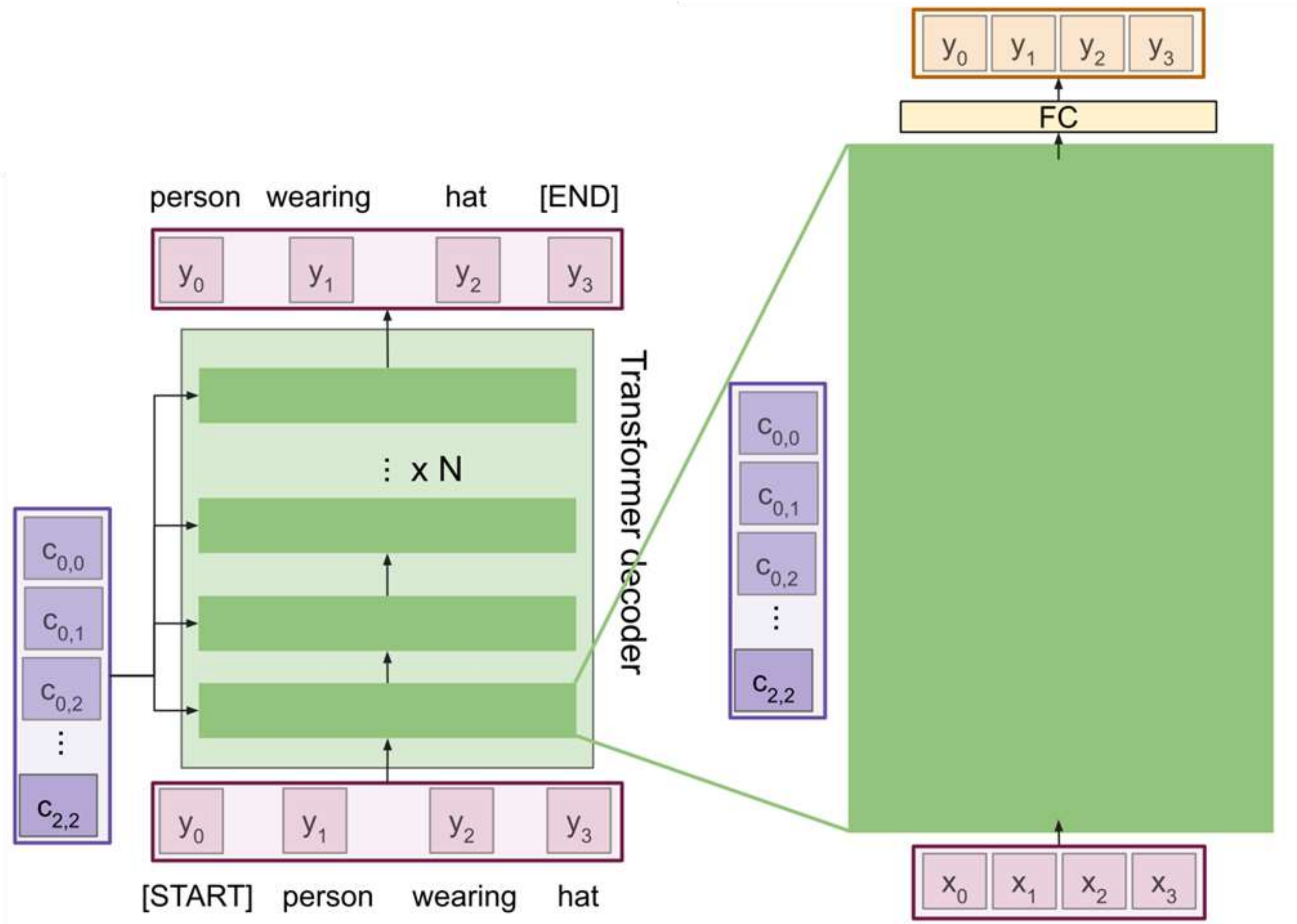




Vaswani et. al, “Attention is all you need”, NeurIPS 2017

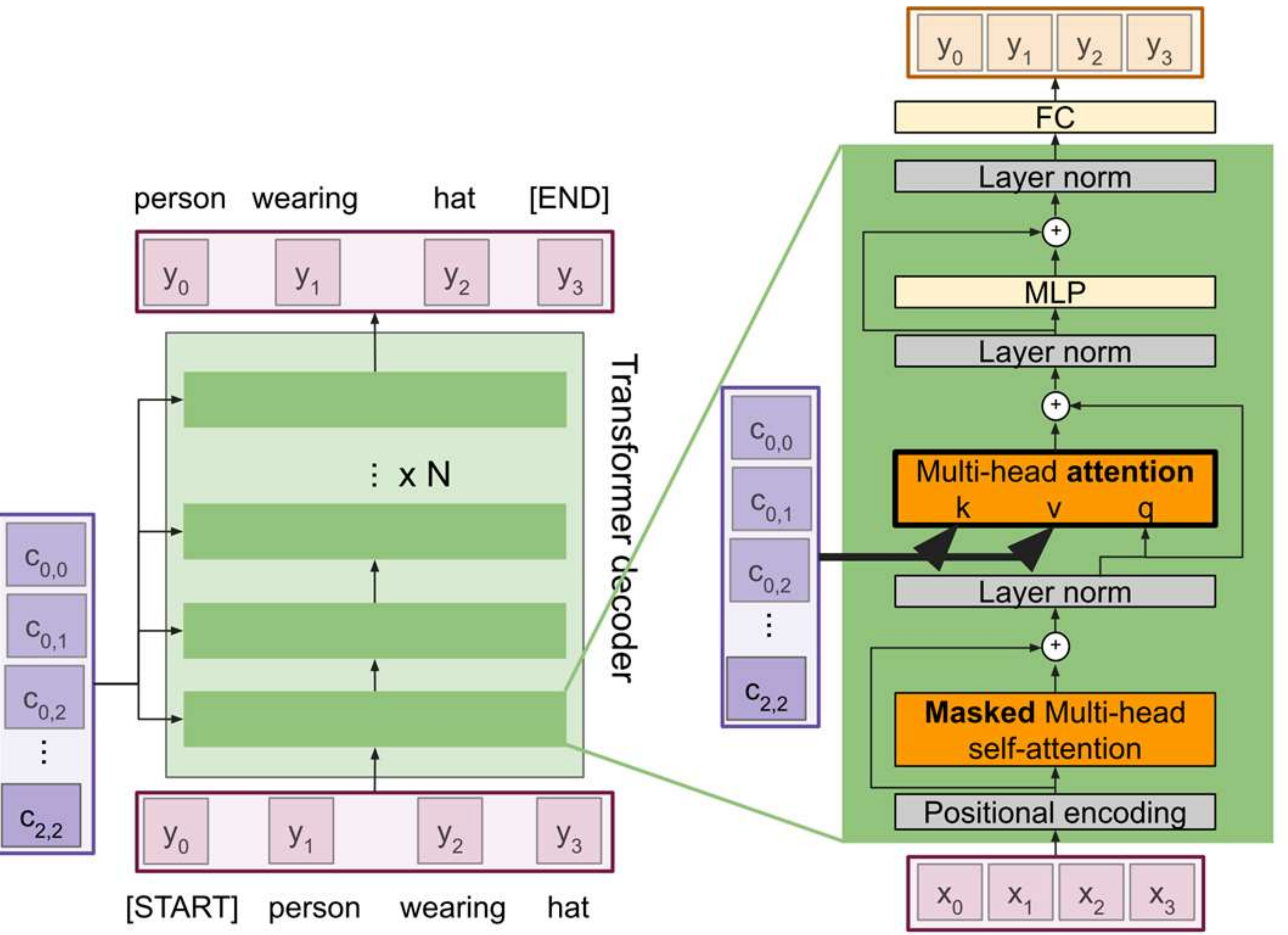
The Transformer decoder block

- Let's dive into the transformer decoder block
- Almost the same as the transformer encoder



The Transformer decoder block

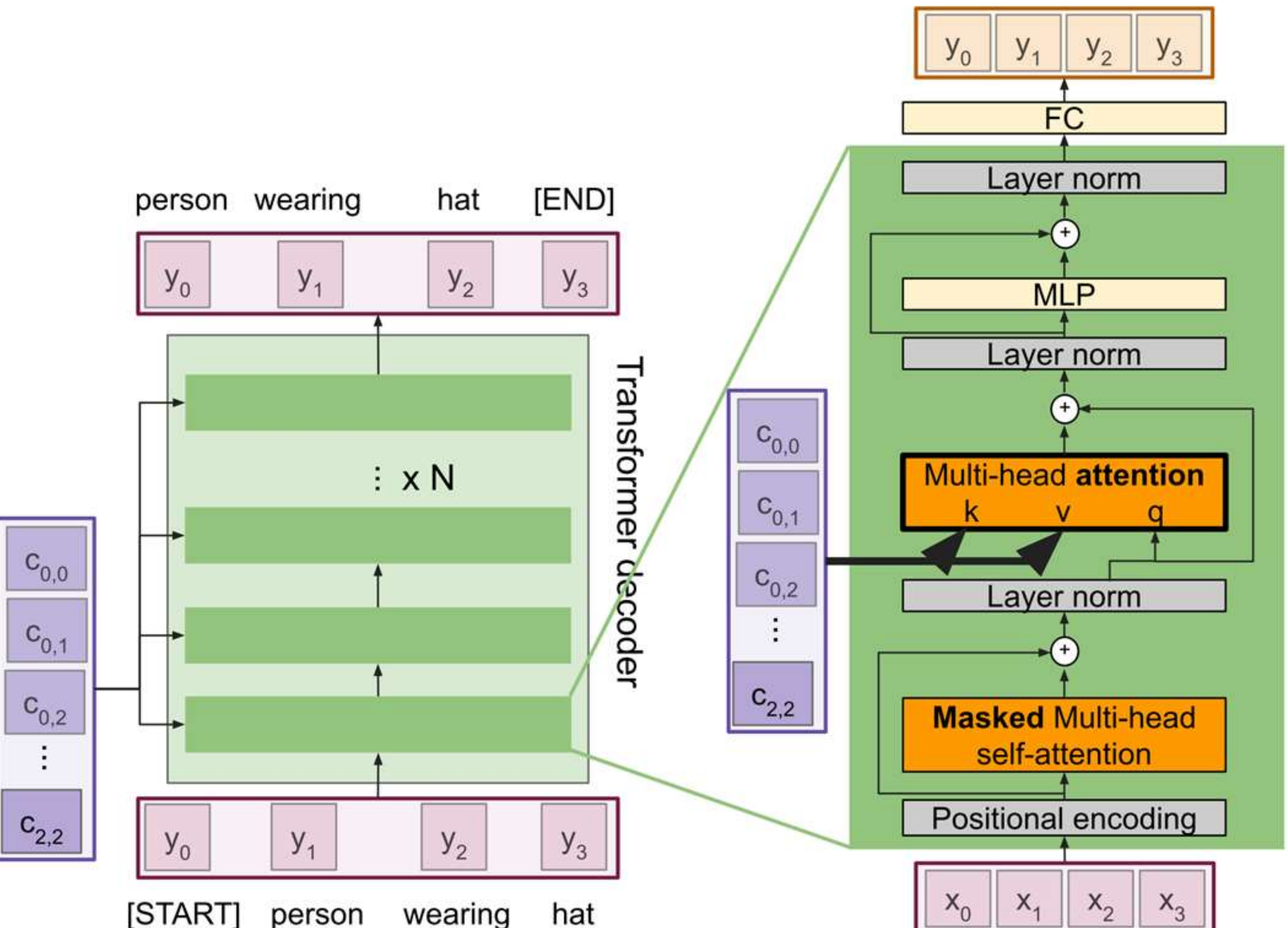
- Inject features from encoder to the decoder



Multi-head attention block attends over the transformer encoder outputs.

The Transformer decoder block

- Inject features from encoder to the decoder



Transformer Decoder Block:

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

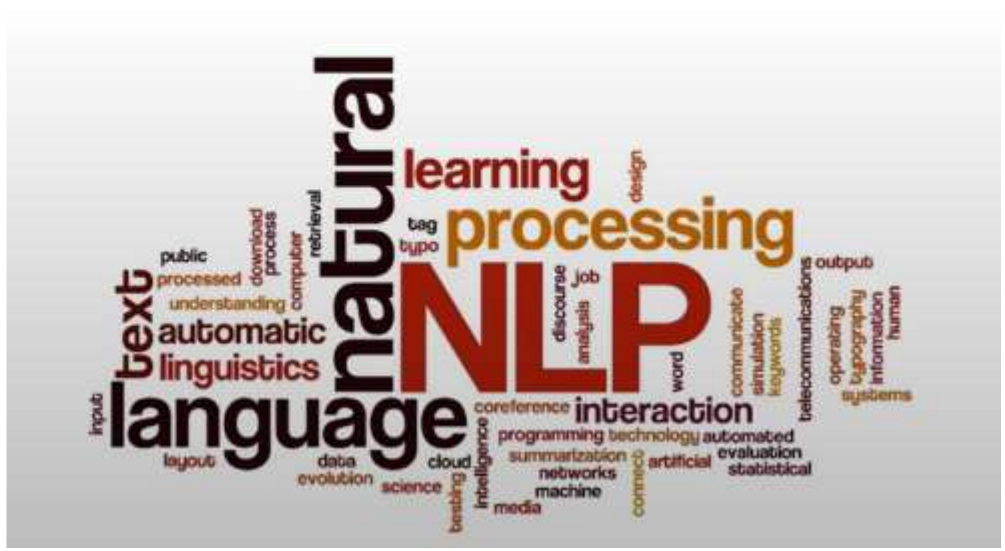
Masked Self-attention only interacts with past inputs.

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

Highly scalable, highly parallelizable, but high memory usage.

The Transformer: Transfer Learning

- Refresh & dominate various NLP tasks
- “ImageNet Moment for Natural Language Processing”
- Pretraining:
  - Download a lot of text from the internet
  - Train a giant Transformer model for language modeling
- Finetuning
  - Fine-tune the Transformer on your own NLP task



Scaling up Transformers

- Still on going ……

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)
BERT-Base	12	768	12	110M	13 GB	
BERT-Large	24	1024	16	340M	13 GB	
XLNet-Large	24	1024	16	~340M	126 GB	512x TPU-v3 (2.5 days)
RoBERTa	24	1024	16	355M	160 GB	1024x V100 GPU (1 day)
GPT-2	12	768	?	117M	40 GB	
GPT-2	24	1024	?	345M	40 GB	
GPT-2	36	1280	?	762M	40 GB	
GPT-2	48	1600	?	1.5B	40 GB	
Megatron-LM	40	1536	16	1.2B	174 GB	64x V100 GPU
Megatron-LM	54	1920	20	2.5B	174 GB	128x V100 GPU
Megatron-LM	64	2304	24	4.2B	174 GB	256x V100 GPU (10 days)
Megatron-LM	72	3072	32	8.3B	174 GB	512x V100 GPU (9 days)

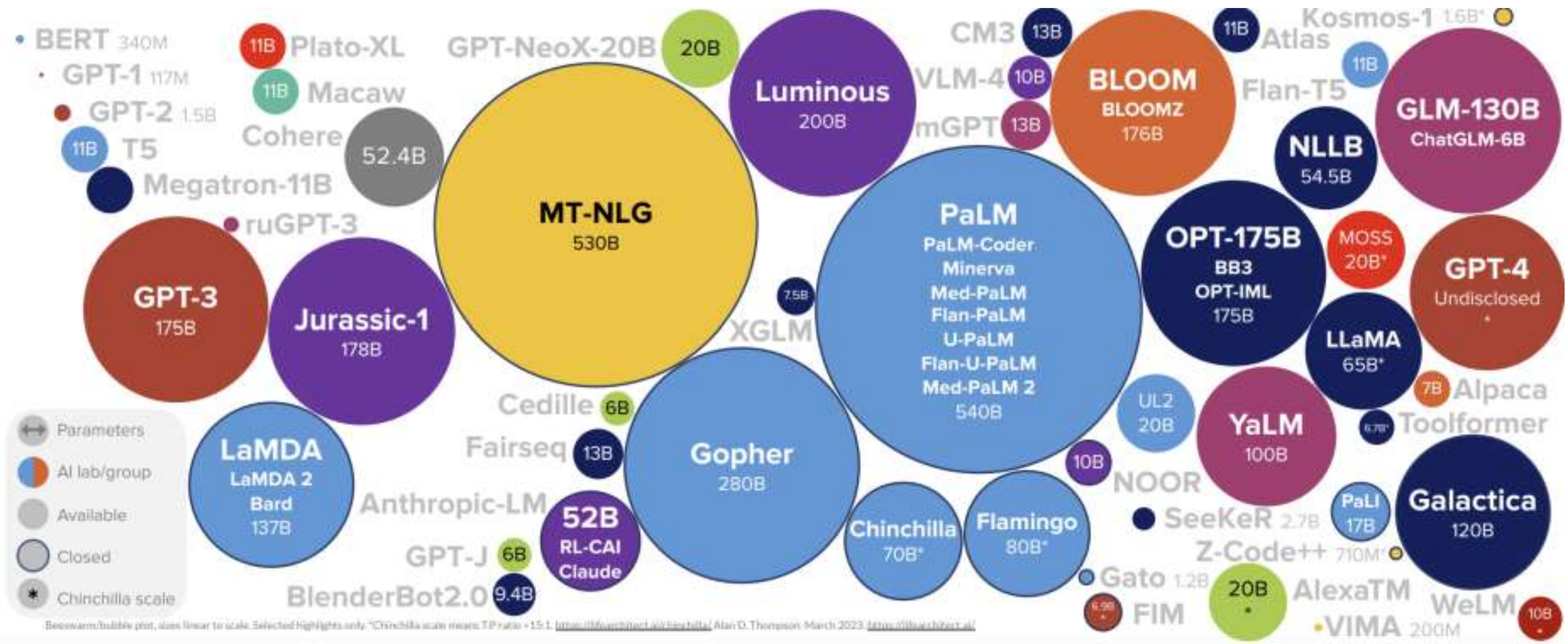
Scaling up Transformers

- Still ongoing ……

~$430,000 on Amazon AWS!

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)
BERT-Base	12	768	12	110M	13 GB	
BERT-Large	24	1024	16	340M	13 GB	
XLNet-Large	24	1024	16	~340M	126 GB	512x TPU-v3 (2.5 days)
RoBERTa	24	1024	16	355M	160 GB	1024x V100 GPU (1 day)
GPT-2	12	768	?	117M	40 GB	
GPT-2	24	1024	?	345M	40 GB	
GPT-2	36	1280	?	762M	40 GB	
GPT-2	48	1600	?	1.5B	40 GB	
Megatron-LM	40	1536	16	1.2B	174 GB	64x V100 GPU
Megatron-LM	54	1920	20	2.5B	174 GB	128x V100 GPU
Megatron-LM	64	2304	24	4.2B	174 GB	256x V100 GPU (10 days)
Megatron-LM	72	3072	32	8.3B	174 GB	$\boxed{\text{512x V100 GPU (9 days)}}$

Language Model Size to MAR/2023



Refresh NLP applications

PROMPT (Human-written): *In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.*

Refresh NLP applications

PROMPT (Human-written): *In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.*

COMPLETION (Transformer-written): The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.
Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.
Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.
Pérez and the others then ventured further into the valley. "By the time we reached the top of one peak, the water looked blue, with some crystals on top," said Pérez.
Pérez and his friends were astonished to see the unicorn herd. These creatures could be seen from the air without having to move too much to see them – they were so close they could touch their horns.
While examining these bizarre creatures the scientists discovered that the creatures also spoke some fairly regular English. Pérez stated, "We can see, for example, that they have a common 'language,' something like a dialect or dialectic."
Dr. Pérez believes that the unicorns may have originated in Argentina, where the animals were believed to be descendants of a lost race of people who lived there before the arrival of humans in those parts of South America.
While their origins are still unclear, some believe that perhaps the creatures were created when a human and a unicorn met each other in a time before human civilization. According to Pérez, "In South America, such incidents seem to be quite common."
However, Pérez also pointed out that it is likely that the only way of knowing for sure if unicorns are indeed the descendants of a lost alien race is through DNA. "But they seem to be able to communicate in English quite well, which I believe is a sign of evolution, or at least a change in social organization," said the scientist.

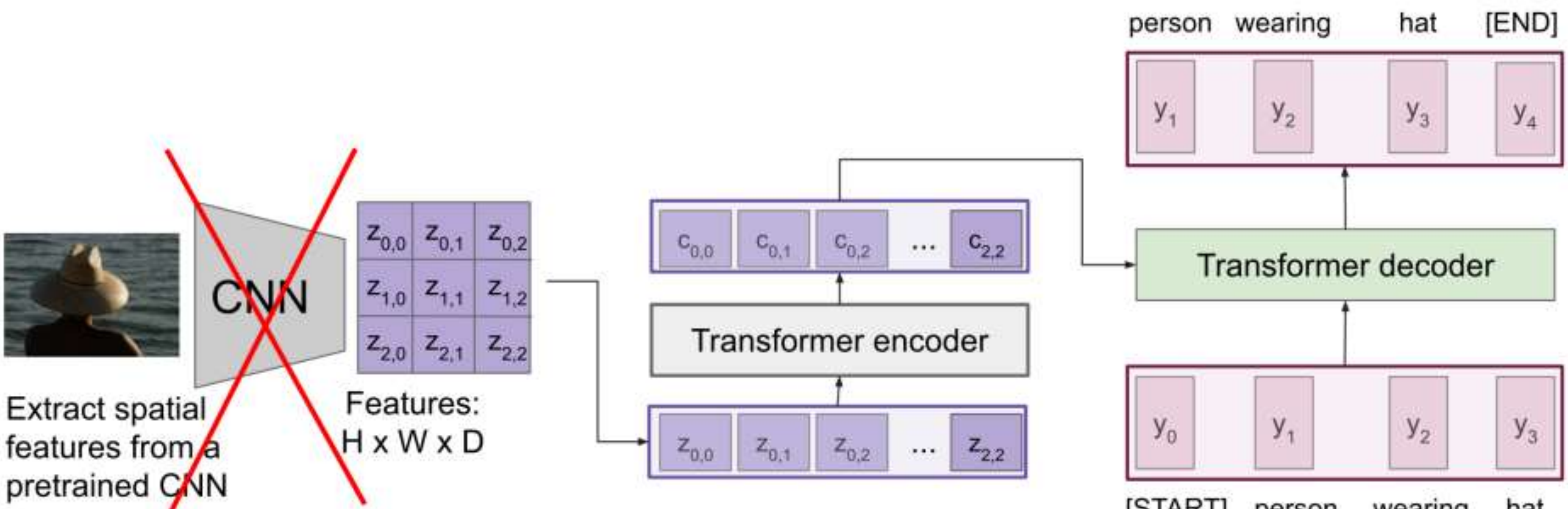
OpenAI, "Better Language Models and their Implications", 2019, https://openai.com/blog/better-language-models/

https://app.inferkit.com/demo

Refresh NLP (HCI, AI ……) applications

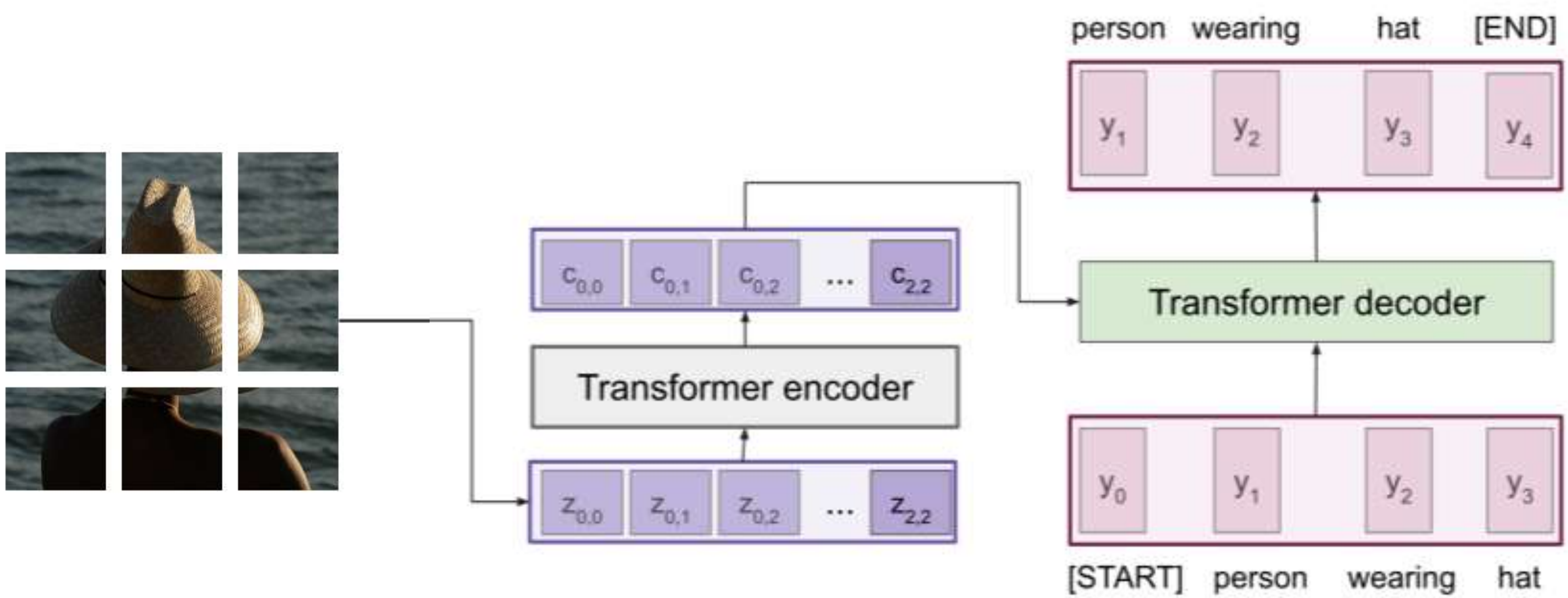
Vision Transformer

- Perhaps we don't need convolutions at all?



Vision Transformer

- Image Captioning using ONLY transformers
- Transformers from pixels to language

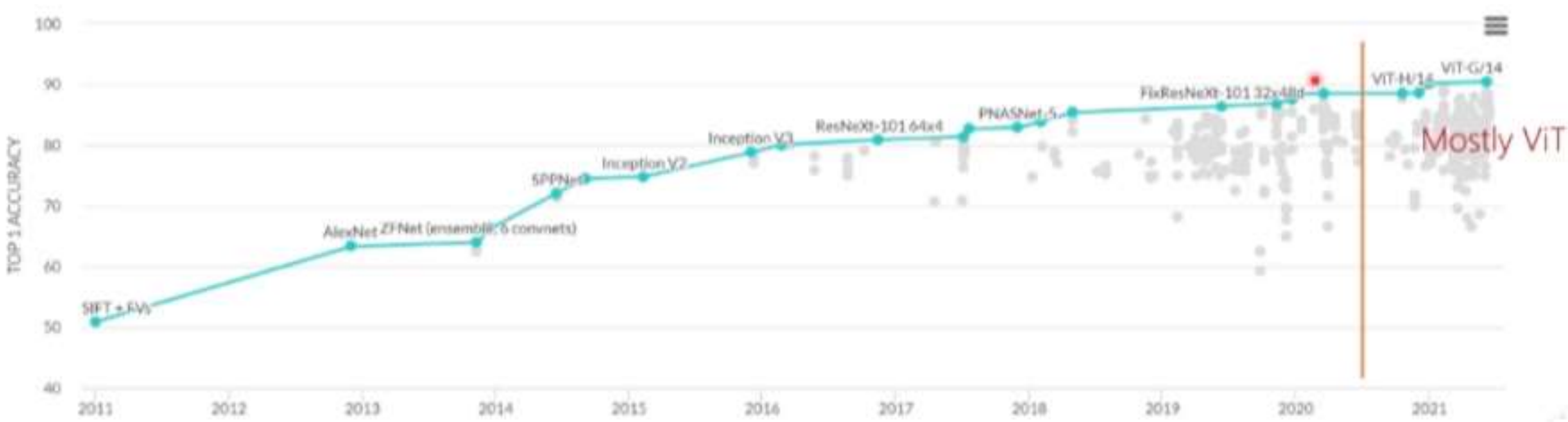


Dosovitskiy et al, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, ECCV2020

Vision Transformer

- Image Captioning using ONLY transformers
- Transformers from pixels to language
- SOTA performance on ImageNet-1K

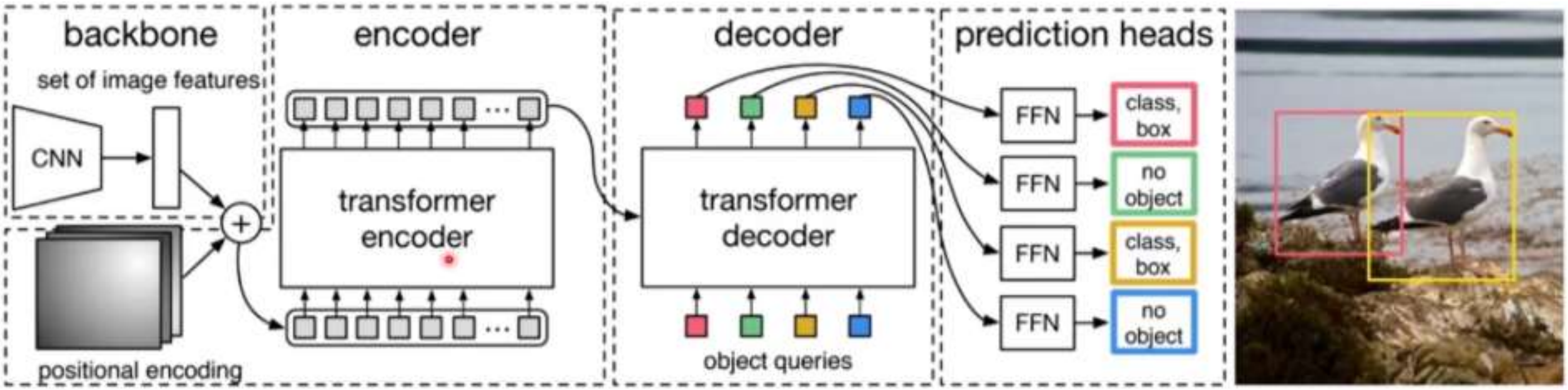
ImageNet-1K image classification



Dosovitskiy et al, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, ECCV2020

Vision Transformer

- Similarly, treat object detection as machine translation (or encoder-decoder) problem



Nicolas Carion et al, "End to end object detection with transformers", ECCV2020

Vision Transformer

- Image Captioning using ONLY Transformers
- Vision Transformers vs. ResNets

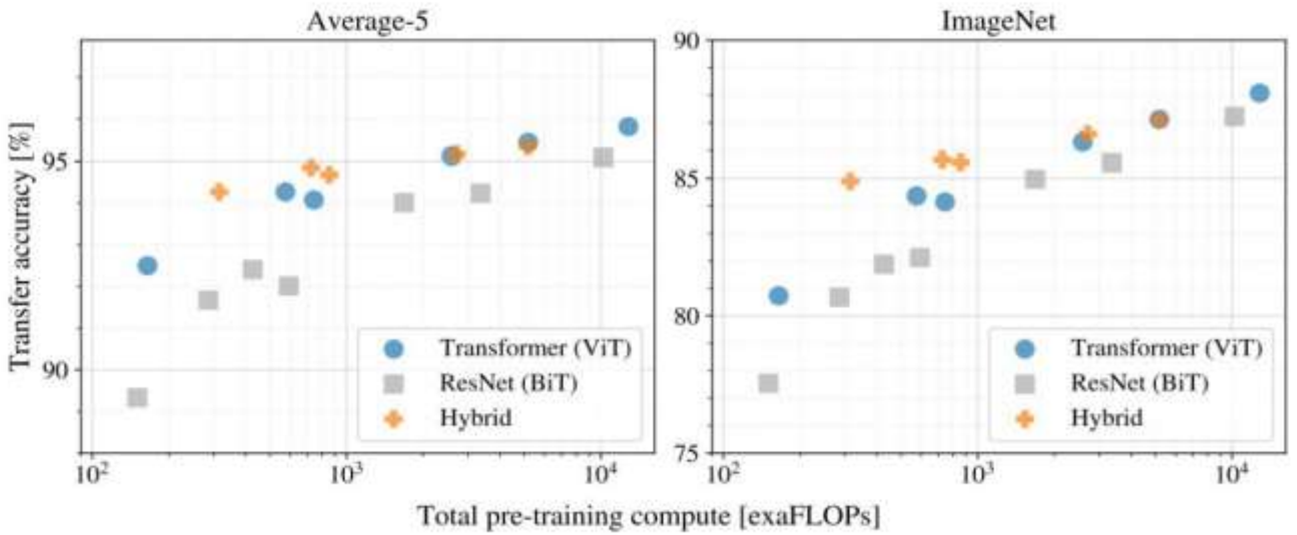
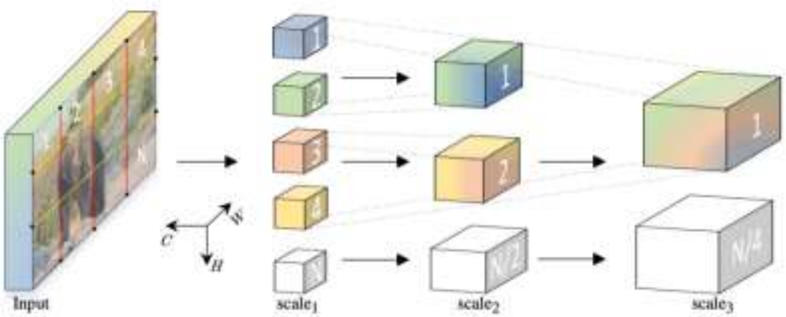


Figure 5: Performance versus cost for different architectures: Vision Transformers, ResNets, and hybrids. Vision Transformers generally outperform ResNets with the same computational budget. Hybrids improve upon pure Transformers for smaller model sizes, but the gap vanishes for larger models.

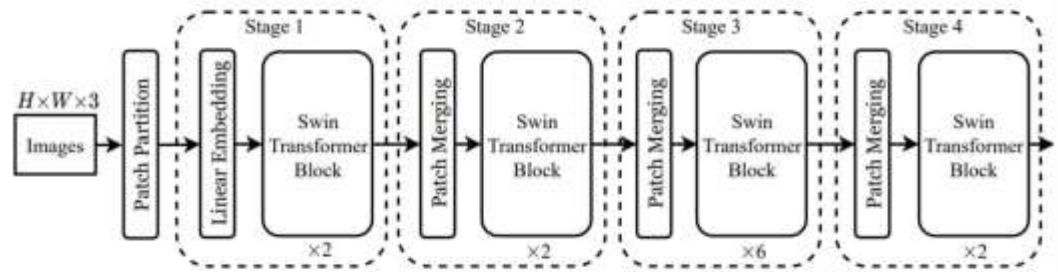
Dosovitskiy et al, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, ECCV2020

Vision Transformers

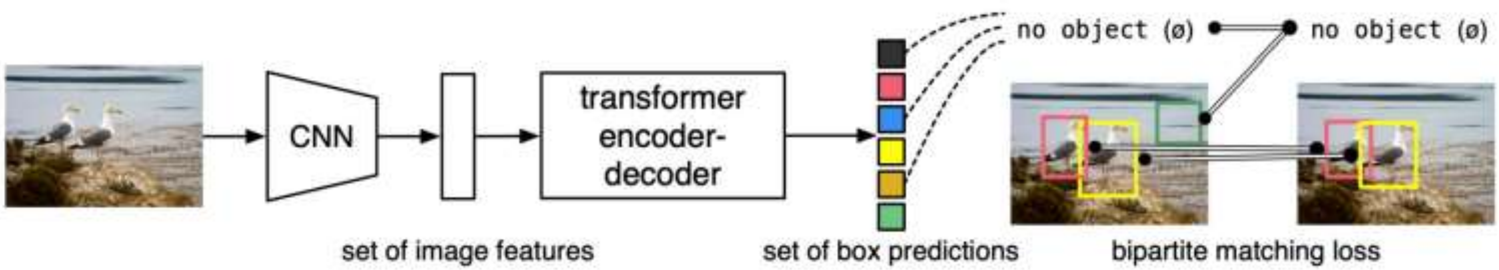
- Still ongoing ……



Fan et al, "Multiscale Vision Transformers", ICCV 2021



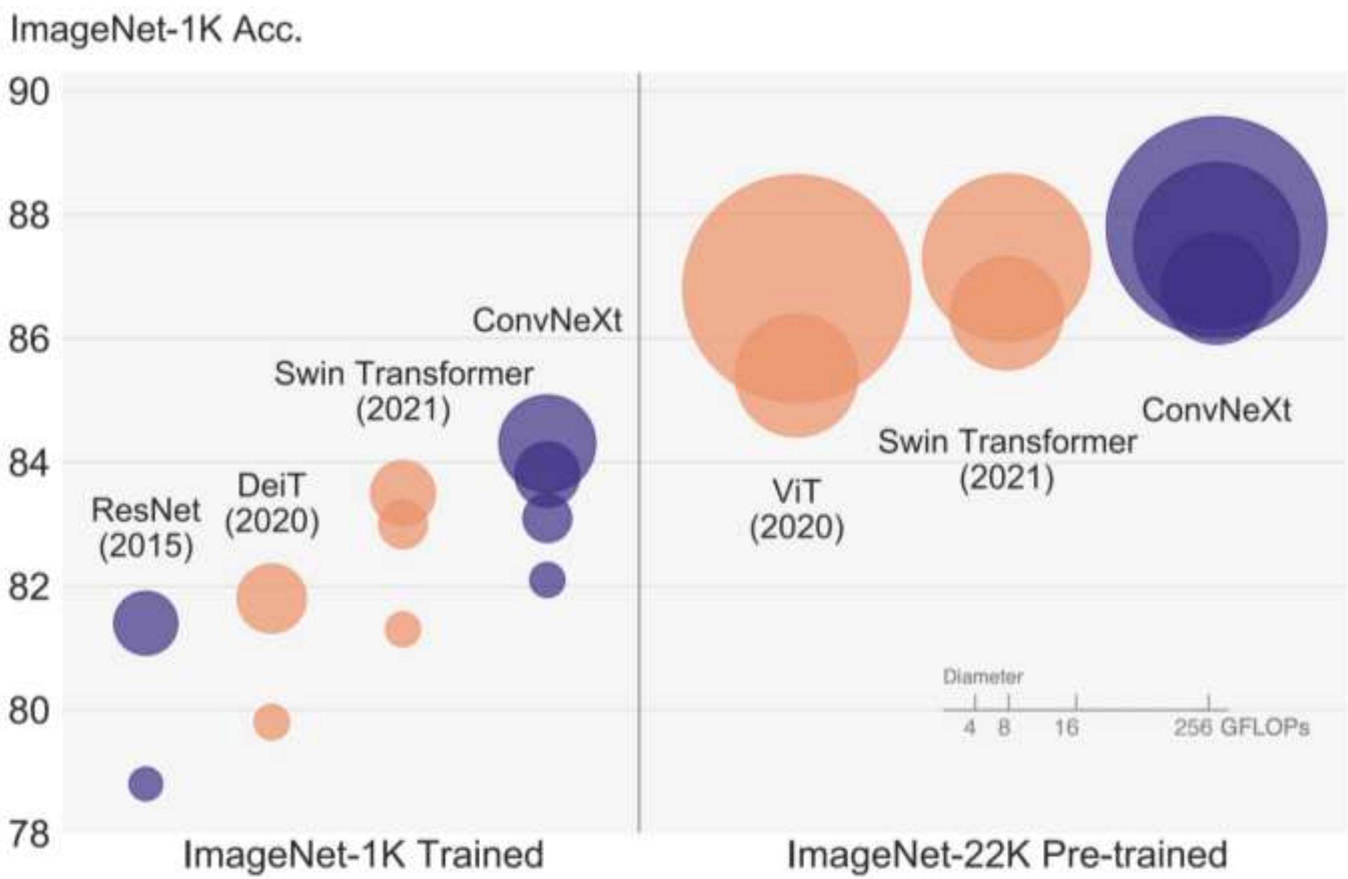
Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021



Carion et al, "End-to-End Object Detection with Transformers", ECCV 2020

Vision Transformers

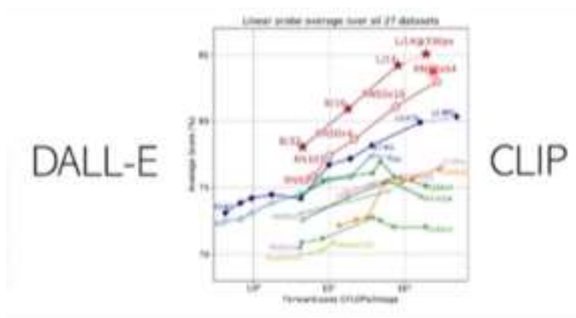
- ConvNets strike back!



A ConvNet for the 2020s. Liu et al. CVPR 2022

Vision Transformers

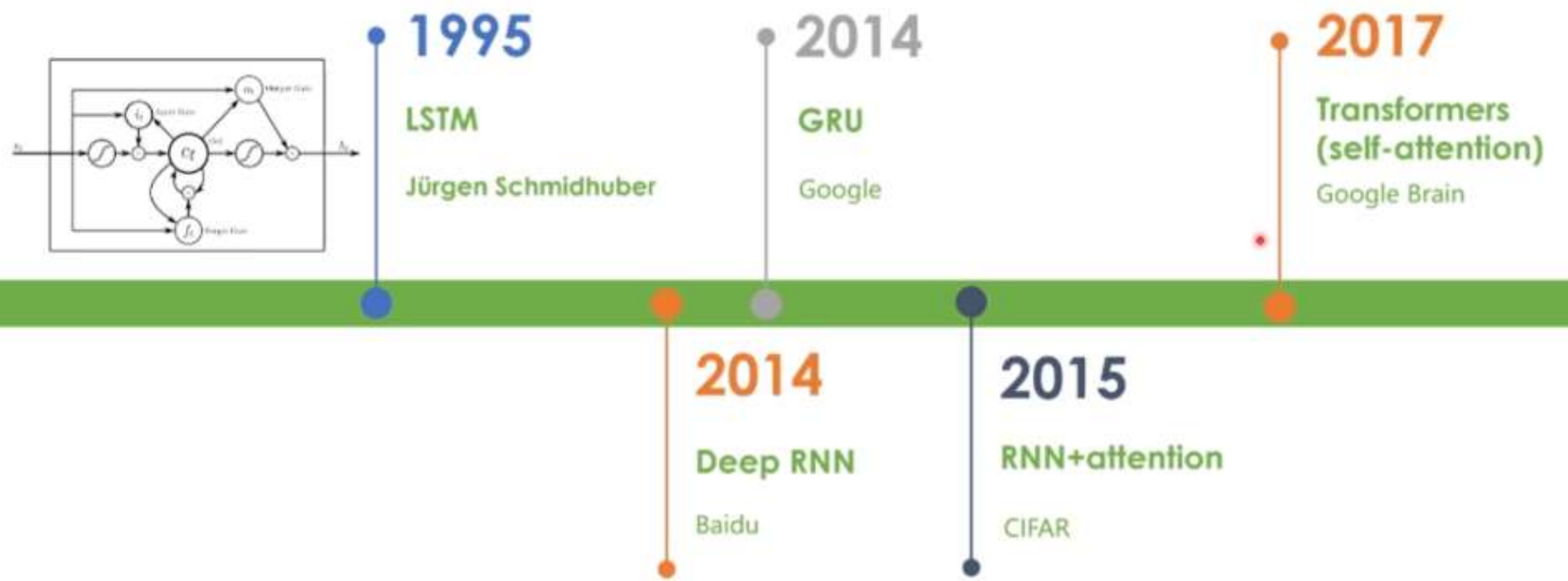
- New large-scale transformer models
- Cross modality: https://openai.com/blog/dall-e/



A ConvNet for the 2020s. Ramesh et al., “Zero-Shot Text-to-Image Generation”, ICML 2021

Vision Transformers

- Recall the model evolution in NLP or sequential data

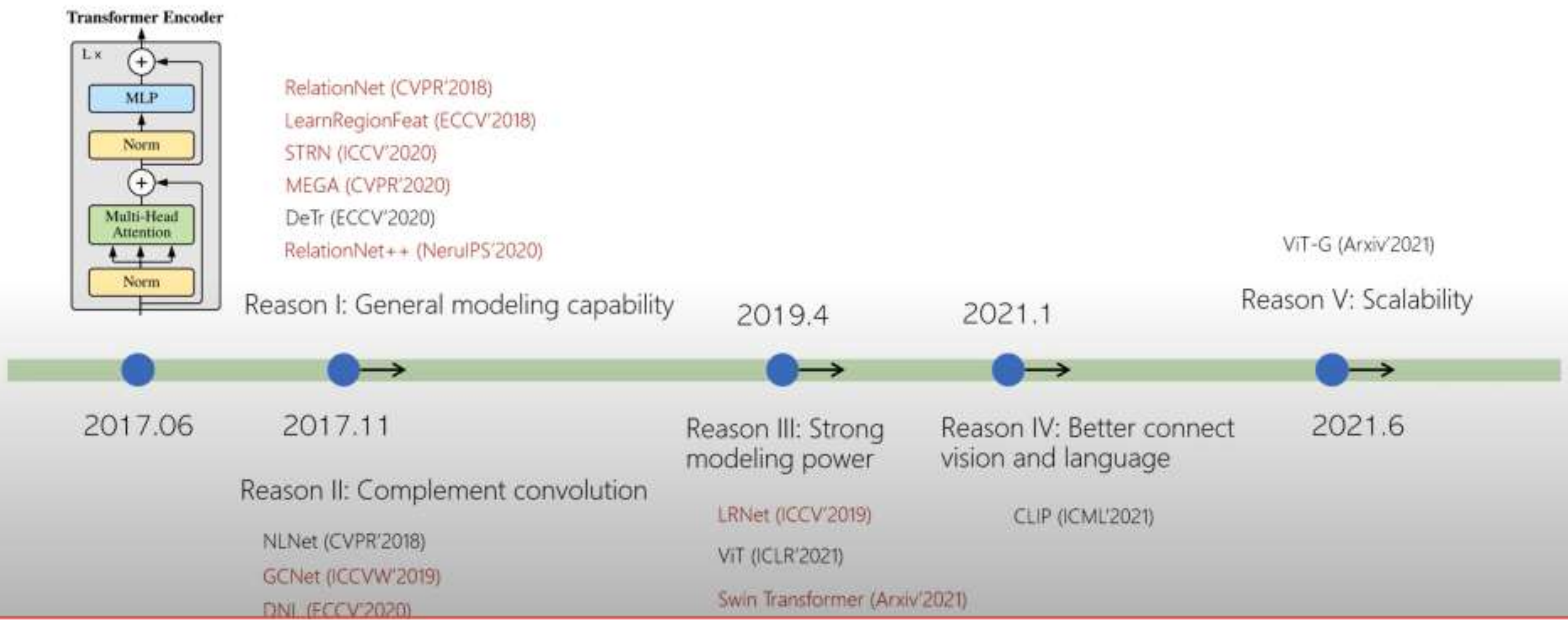


Vision Transformers

- Can NLP/CV share the same basic modules?

Vision Transformers

- Still unleash the power of Transformer in CV



Summary

- Attention
  - Recall RNNs in Vision and NLP
  - Attention Mechanism
  - General Attention to Self-attention
  - Positional encoding
  - Self-attention and CNN
- Transformer
  - Multihead Self-Attention
  - Transformer Architecture
  - Vision Transformer
- Next time:
  - Transformer application