

ITCS 6156/8156 Fall 2023 Machine Learning

Attention & Transformers

Instructor: Hongfei Xue

Email: hongfei.xue@charlotte.edu

Class Meeting: Mon & Wed, 4:00 PM – 5:15 PM, CHHS 376

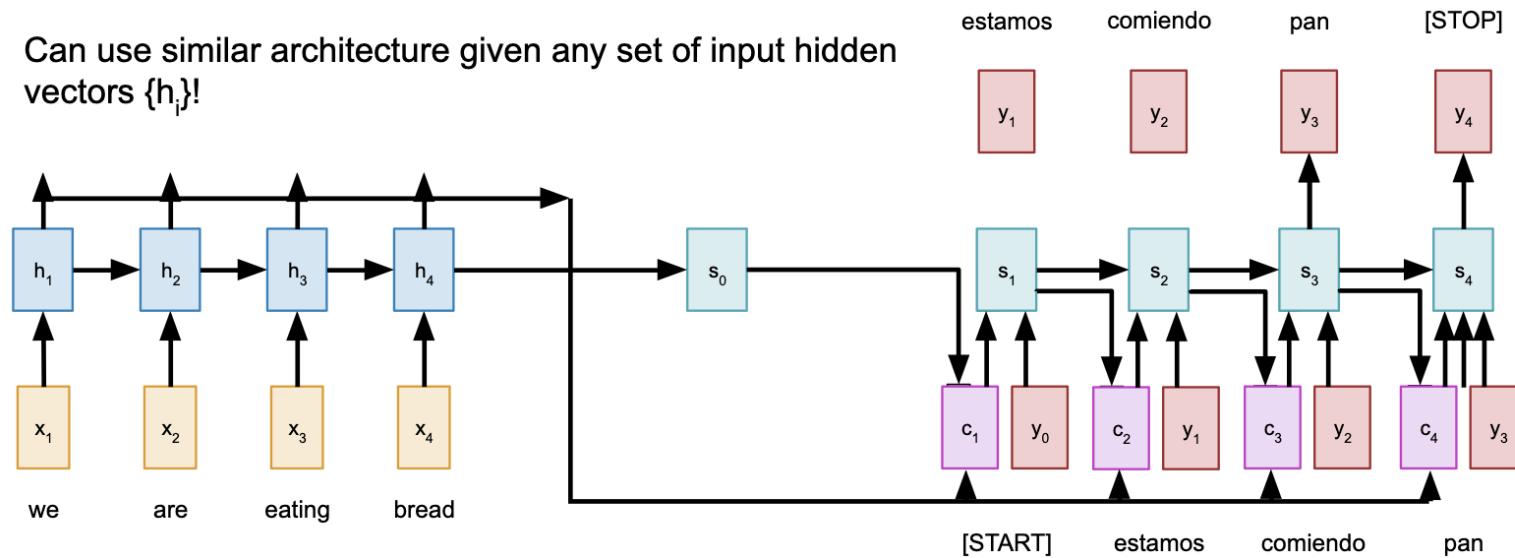


Some content in the slides is based on Dr. Ruohan Gao's lectures

Sequence to Sequence with RNNs and Attention

The decoder doesn't use the fact that h_i form an ordered sequence – it just treats them as an unordered set $\{h_i\}$

Can use similar architecture given any set of input hidden vectors $\{h_i\}$!

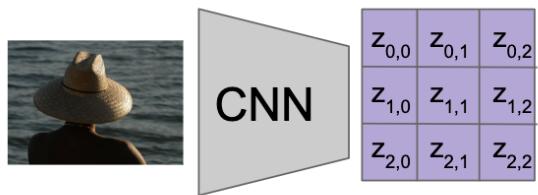


Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Image Captioning using spatial features

Input: Image I

Output: Sequence $\mathbf{y} = y_1, y_2, \dots, y_T$



Extract spatial
features from a
pretrained CNN

Features:
 $H \times W \times D$

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

Image Captioning using spatial features

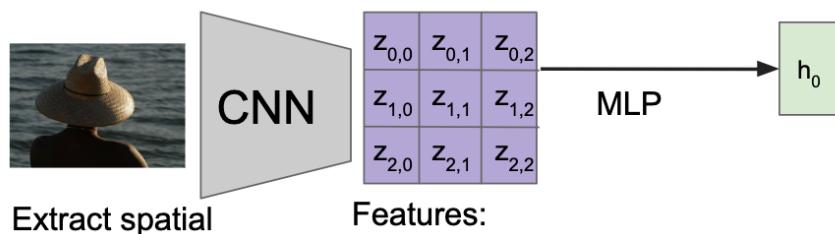
Input: Image I

Output: Sequence $\mathbf{y} = y_1, y_2, \dots, y_T$

Encoder: $h_0 = f_w(\mathbf{z})$

where \mathbf{z} is spatial CNN features

$f_w(\cdot)$ is an MLP



Extract spatial
features from a
pretrained CNN

Features:
 $H \times W \times D$

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

Image Captioning using spatial features

Input: Image I

Output: Sequence $\mathbf{y} = y_1, y_2, \dots, y_T$

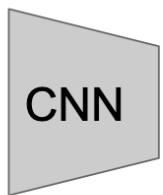
Decoder: $y_t = g_v(y_{t-1}, h_{t-1}, c)$

where context vector c is often $c = h_0$

Encoder: $h_0 = f_w(\mathbf{z})$

where \mathbf{z} is spatial CNN features

$f_w(\cdot)$ is an MLP

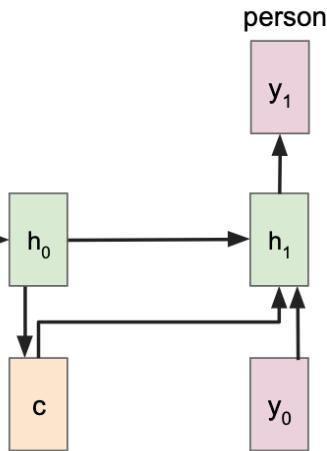


$z_{0,0}$	$z_{0,1}$	$z_{0,2}$
$z_{1,0}$	$z_{1,1}$	$z_{1,2}$
$z_{2,0}$	$z_{2,1}$	$z_{2,2}$

MLP

Extract spatial
features from a
pretrained CNN

Features:
 $H \times W \times D$



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

[START]

Image Captioning using spatial features

Input: Image I

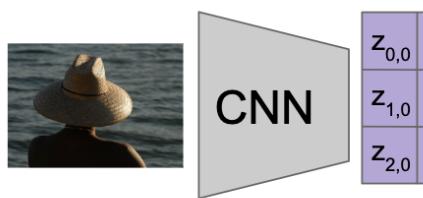
Output: Sequence $\mathbf{y} = y_1, y_2, \dots, y_T$

Decoder: $y_t = g_v(y_{t-1}, h_{t-1}, c)$
where context vector c is often $c = h_0$

Encoder: $h_0 = f_w(\mathbf{z})$

where \mathbf{z} is spatial CNN features

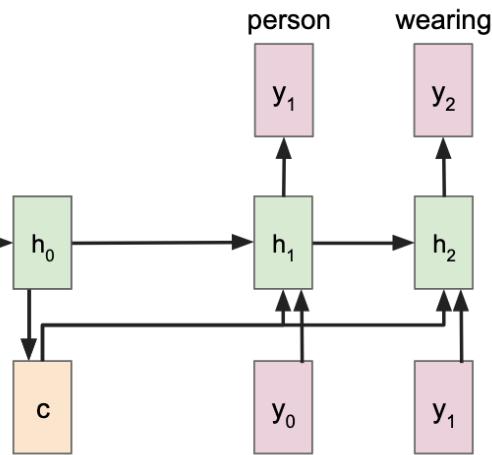
$f_w(\cdot)$ is an MLP



Features:
 $H \times W \times D$

$z_{0,0}$	$z_{0,1}$	$z_{0,2}$
$z_{1,0}$	$z_{1,1}$	$z_{1,2}$
$z_{2,0}$	$z_{2,1}$	$z_{2,2}$

MLP



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

[START] person

Image Captioning using spatial features

Input: Image I

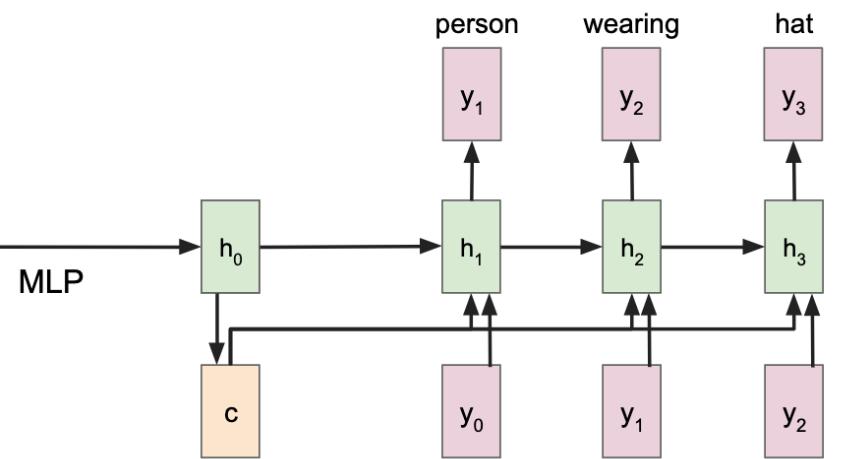
Output: Sequence $\mathbf{y} = y_1, y_2, \dots, y_T$

Decoder: $y_t = g_v(y_{t-1}, h_{t-1}, c)$
where context vector c is often $c = h_0$

Encoder: $h_0 = f_w(\mathbf{z})$

where \mathbf{z} is spatial CNN features

$f_w(\cdot)$ is an MLP



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

Image Captioning using spatial features

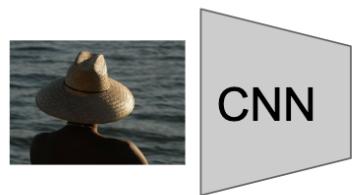
Input: Image I

Output: Sequence $\mathbf{y} = y_1, y_2, \dots, y_T$

Encoder: $h_0 = f_w(\mathbf{z})$

where \mathbf{z} is spatial CNN features

$f_w(\cdot)$ is an MLP



$z_{0,0}$	$z_{0,1}$	$z_{0,2}$
$z_{1,0}$	$z_{1,1}$	$z_{1,2}$
$z_{2,0}$	$z_{2,1}$	$z_{2,2}$

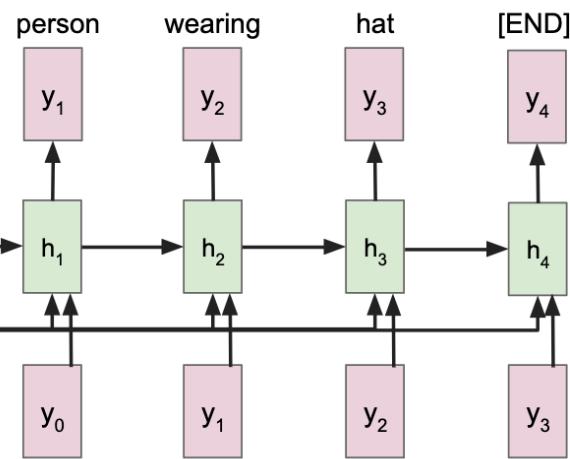
Features:
 $H \times W \times D$

MLP



c

Decoder: $y_t = g_v(y_{t-1}, h_{t-1}, c)$
where context vector c is often $c = h_0$



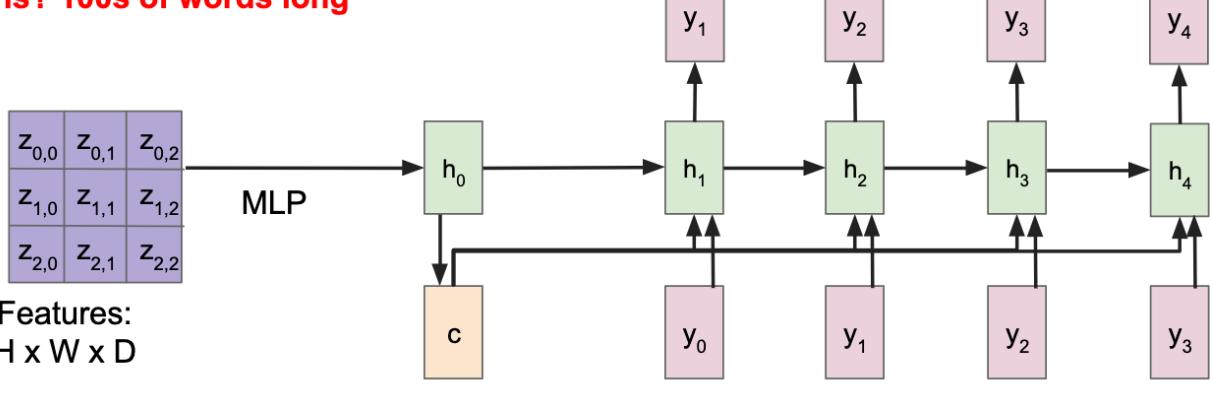
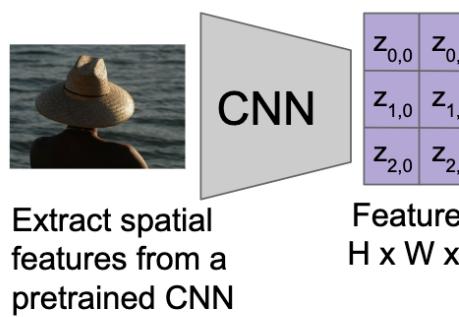
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning using spatial features

Problem: Input is "bottlenecked" through c

- Model needs to encode everything it wants to say within c

This is a problem if we want to generate really long descriptions? 100s of words long



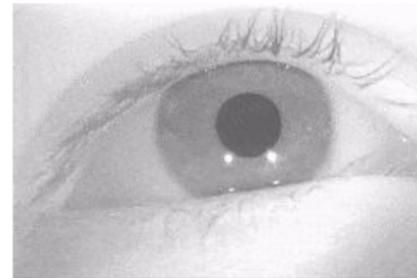
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

[START] person wearing hat

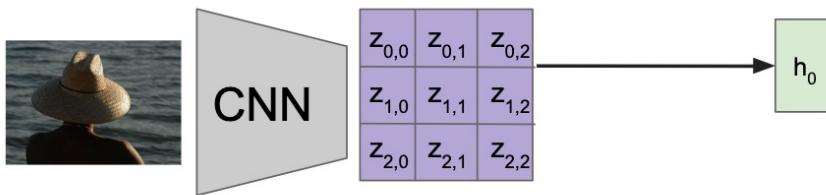
Image Captioning with RNNs and Attention

Attention idea: New context vector at every time step.

Each context vector will attend to different image regions



Attention Saccades in humans



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

Image Captioning with RNNs and Attention

Compute alignments scores (scalars):

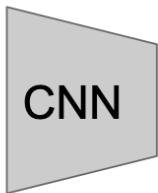
$$e_{t,i,j} = f_{att}(h_{t-1}, z_{i,j})$$

$f_{att}(\cdot)$ is an MLP

Alignment scores:

$H \times W$

$e_{1,0,0}$	$e_{1,0,1}$	$e_{1,0,2}$
$e_{1,1,0}$	$e_{1,1,1}$	$e_{1,1,2}$
$e_{1,2,0}$	$e_{1,2,1}$	$e_{1,2,2}$



Extract spatial features from a pretrained CNN

$z_{0,0}$	$z_{0,1}$	$z_{0,2}$
$z_{1,0}$	$z_{1,1}$	$z_{1,2}$
$z_{2,0}$	$z_{2,1}$	$z_{2,2}$

Features:
 $H \times W \times D$

h_0

Alignment scores:

$H \times W$

$e_{1,0,0}$	$e_{1,0,1}$	$e_{1,0,2}$
$e_{1,1,0}$	$e_{1,1,1}$	$e_{1,1,2}$
$e_{1,2,0}$	$e_{1,2,1}$	$e_{1,2,2}$

h_0

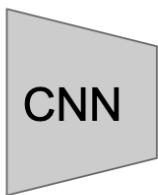
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with RNNs and Attention

Compute alignments scores (scalars):

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

$f_{att}(\cdot)$ is an MLP



Extract spatial features from a pretrained CNN

Alignment scores:
 $H \times W$

$e_{1,0,0}$	$e_{1,0,1}$	$e_{1,0,2}$
$e_{1,1,0}$	$e_{1,1,1}$	$e_{1,1,2}$
$e_{1,2,0}$	$e_{1,2,1}$	$e_{1,2,2}$

Attention:
 $H \times W$

$a_{1,0,0}$	$a_{1,0,1}$	$a_{1,0,2}$
$a_{1,1,0}$	$a_{1,1,1}$	$a_{1,1,2}$
$a_{1,2,0}$	$a_{1,2,1}$	$a_{1,2,2}$

Normalize to get attention weights:

$$a_{t,:,:} = \text{softmax}(e_{t,:,:})$$

$0 < a_{t,i,j} < 1$,
attention values sum to 1

Features:
 $H \times W \times D$

$z_{0,0}$	$z_{0,1}$	$z_{0,2}$
$z_{1,0}$	$z_{1,1}$	$z_{1,2}$
$z_{2,0}$	$z_{2,1}$	$z_{2,2}$



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

Image Captioning with RNNs and Attention

Compute alignments scores (scalars):

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

$f_{att}(\cdot)$ is an MLP

Alignment scores:

H x W

$e_{1,0,0}$	$e_{1,0,1}$	$e_{1,0,2}$
$e_{1,1,0}$	$e_{1,1,1}$	$e_{1,1,2}$
$e_{1,2,0}$	$e_{1,2,1}$	$e_{1,2,2}$

Attention:

H x W

$a_{1,0,0}$	$a_{1,0,1}$	$a_{1,0,2}$
$a_{1,1,0}$	$a_{1,1,1}$	$a_{1,1,2}$
$a_{1,2,0}$	$a_{1,2,1}$	$a_{1,2,2}$

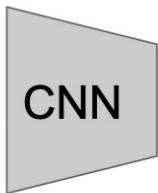
Normalize to get attention weights:

$$a_{t,:,:} = \text{softmax}(e_{t,:,:})$$

$0 < a_{t,i,j} < 1$,
attention values sum to 1

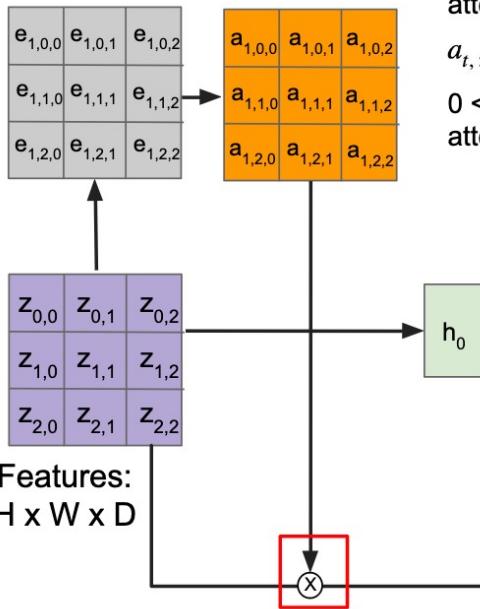
Compute context vector:

$$c_t = \sum_{i,j} a_{t,i,j} z_{t,i,j}$$



Extract spatial features from a pretrained CNN

Features:
H x W x D



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

Image Captioning with RNNs and Attention

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

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

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

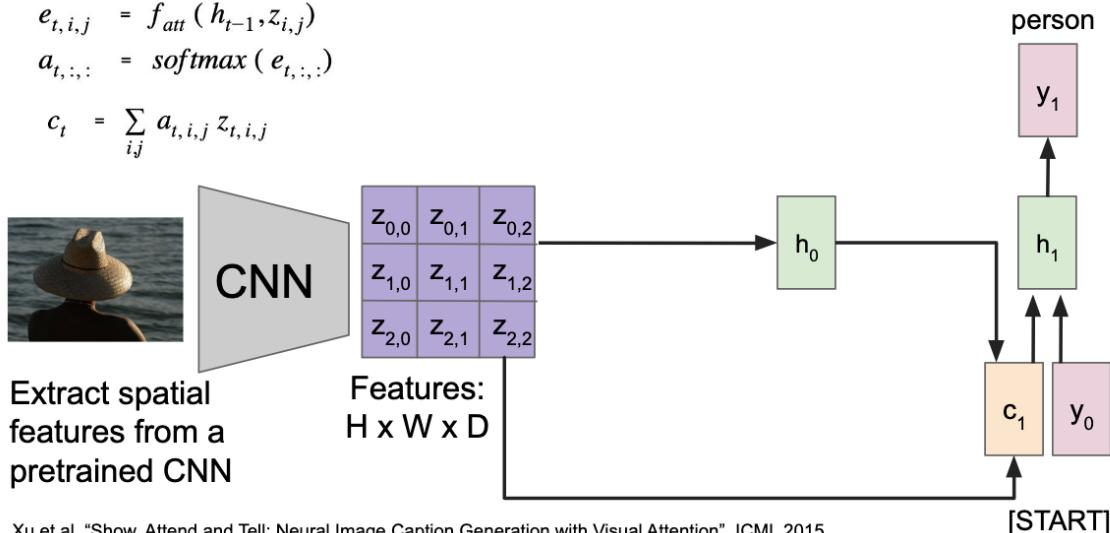


Image Captioning with RNNs and Attention

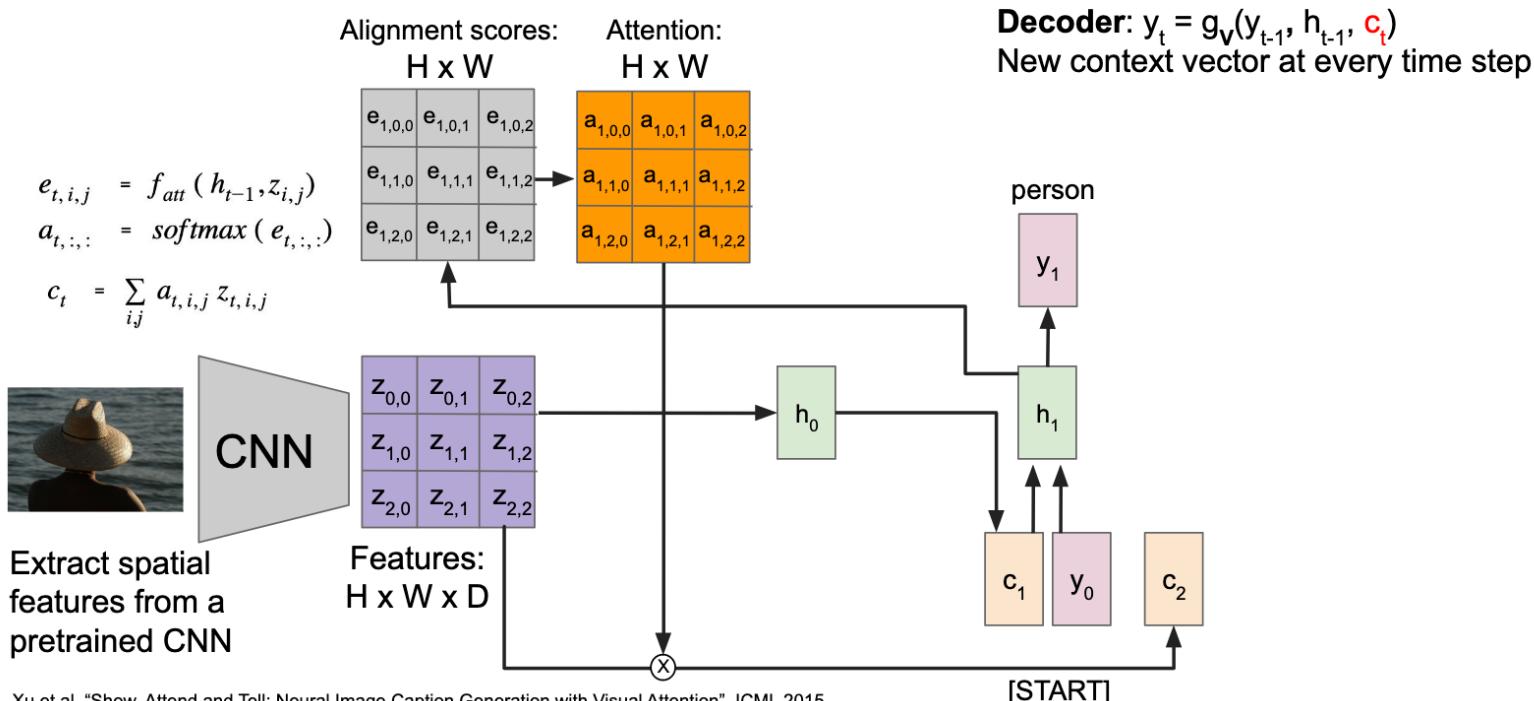
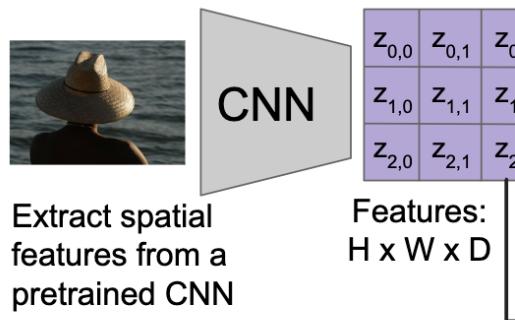


Image Captioning with RNNs and Attention

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

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



Extract spatial
features from a
pretrained CNN

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

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

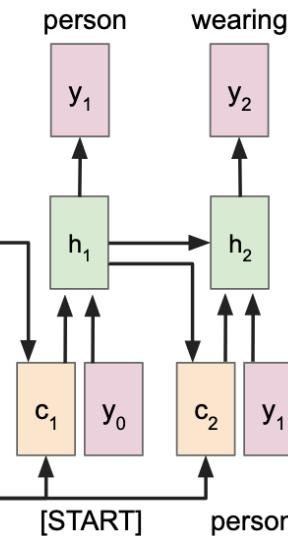
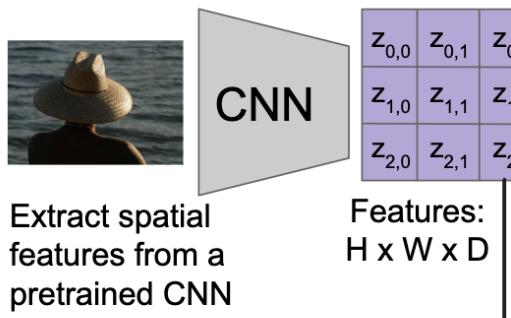


Image Captioning with RNNs and Attention

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

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



Extract spatial
features from a
pretrained CNN

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

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

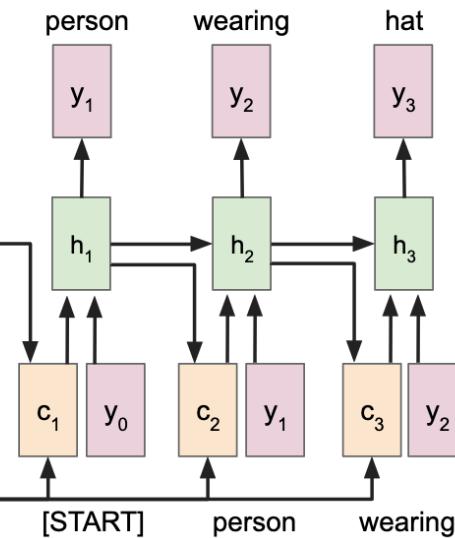
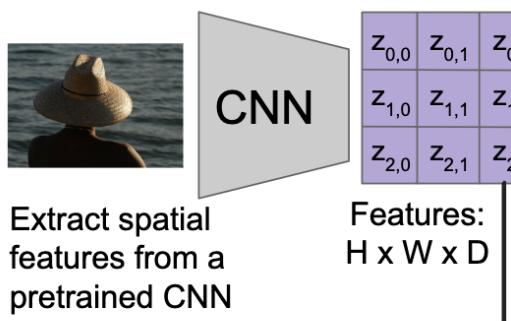


Image Captioning with RNNs and Attention

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

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



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

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

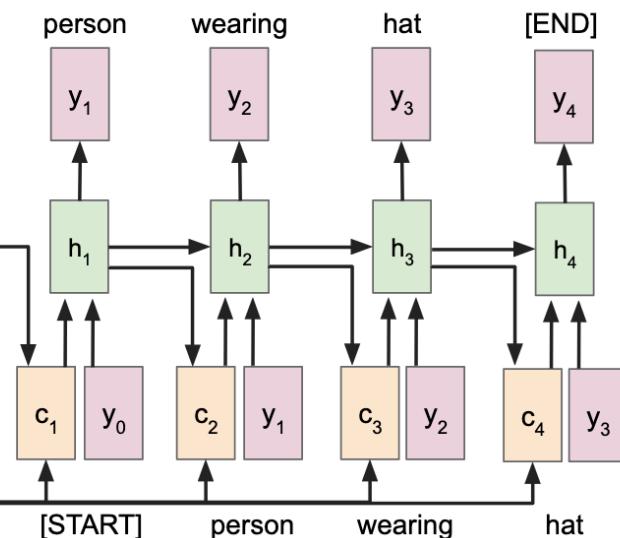
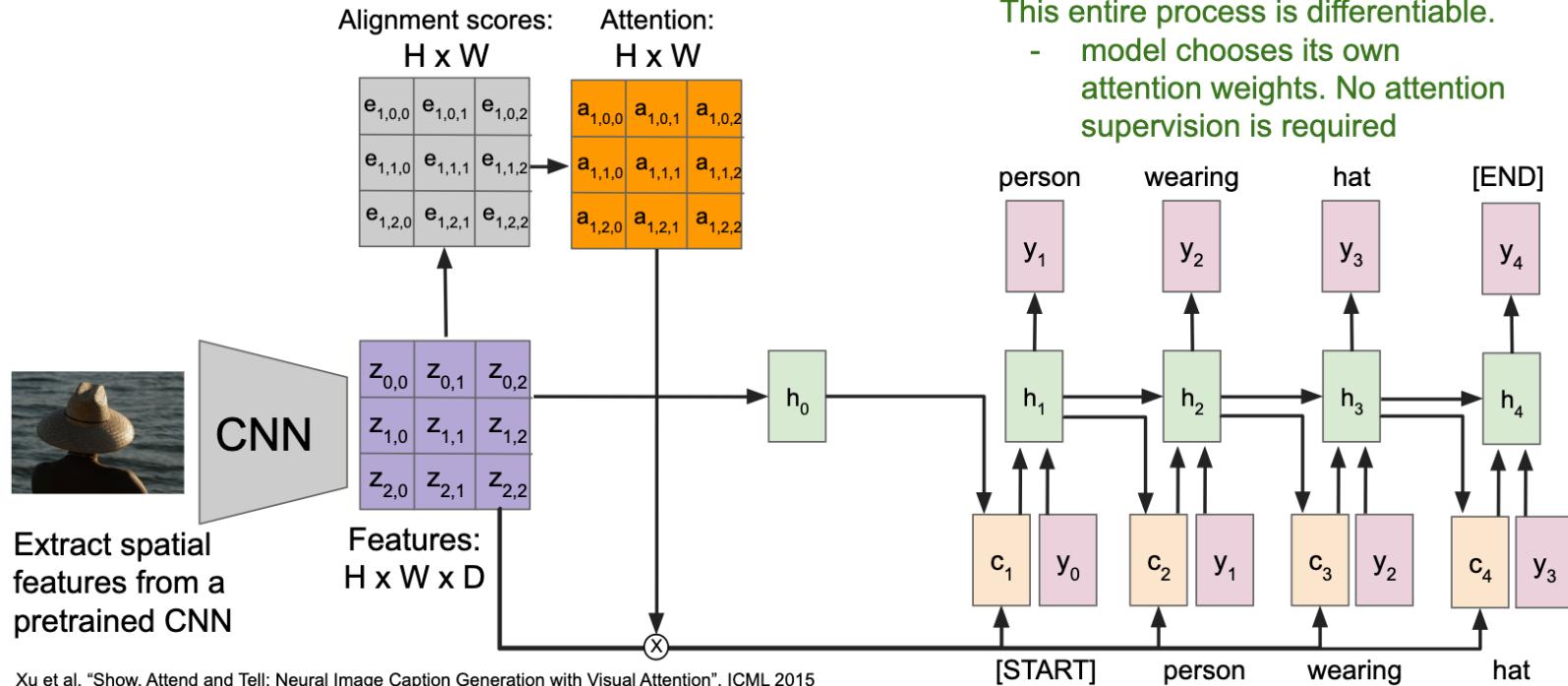


Image Captioning with RNNs and Attention



Attention we just saw in image captioning

Features	$z_{0,0}$	$z_{0,1}$	$z_{0,2}$
	$z_{1,0}$	$z_{1,1}$	$z_{1,2}$
	$z_{2,0}$	$z_{2,1}$	$z_{2,2}$

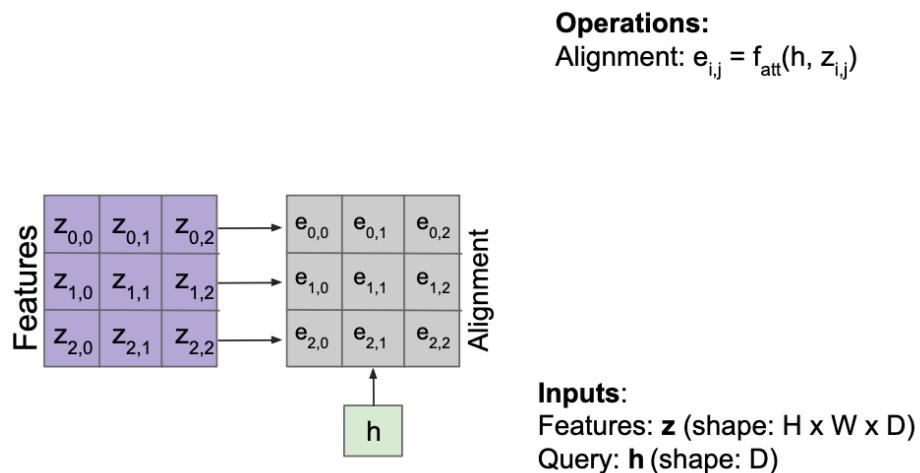
h

Inputs:

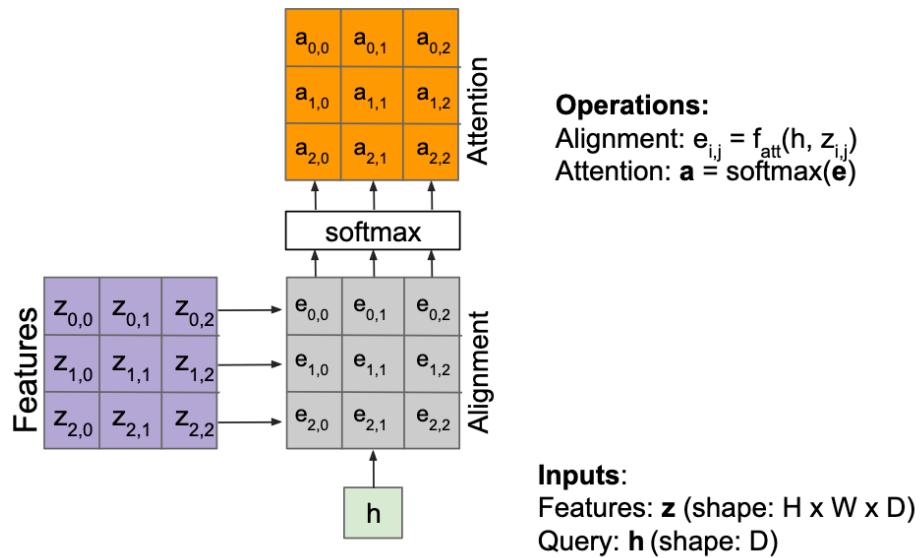
Features: \mathbf{z} (shape: $H \times W \times D$)

Query: \mathbf{h} (shape: D)

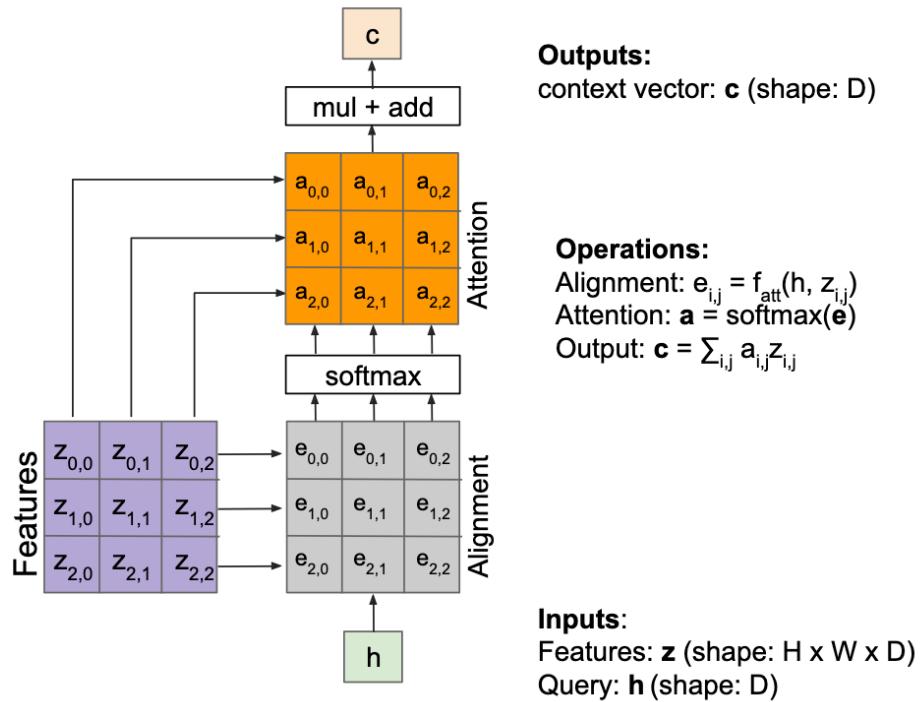
Attention we just saw in image captioning



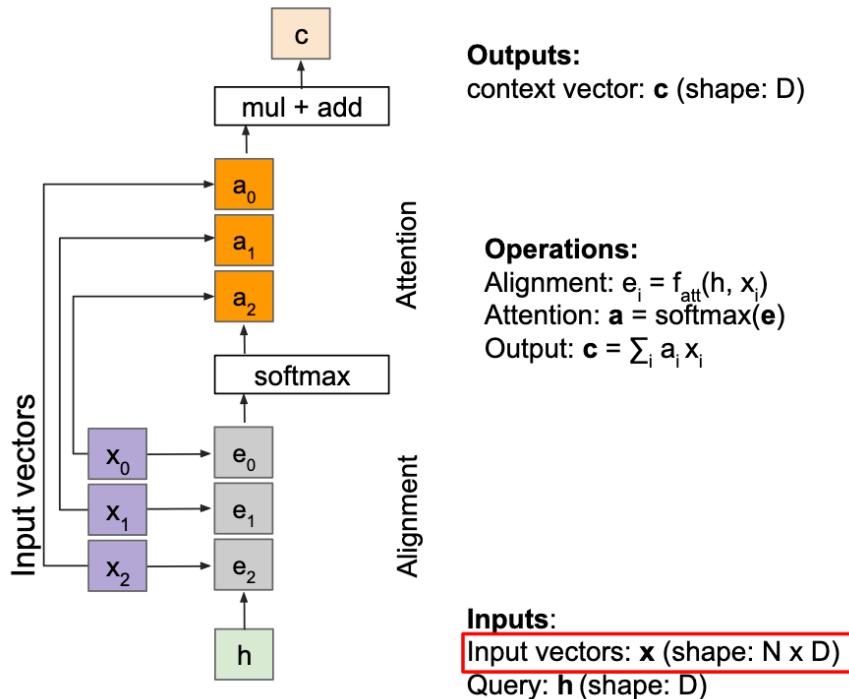
Attention we just saw in image captioning



Attention we just saw in image captioning



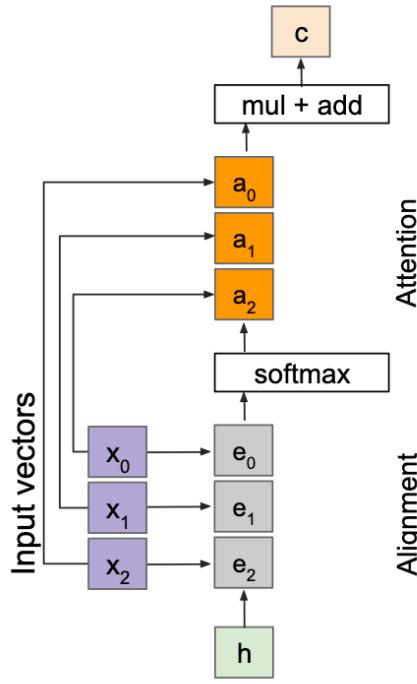
General Attention Layer



Attention operation is **permutation invariant**.

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

General Attention Layer



Outputs:
context vector: c (shape: D)

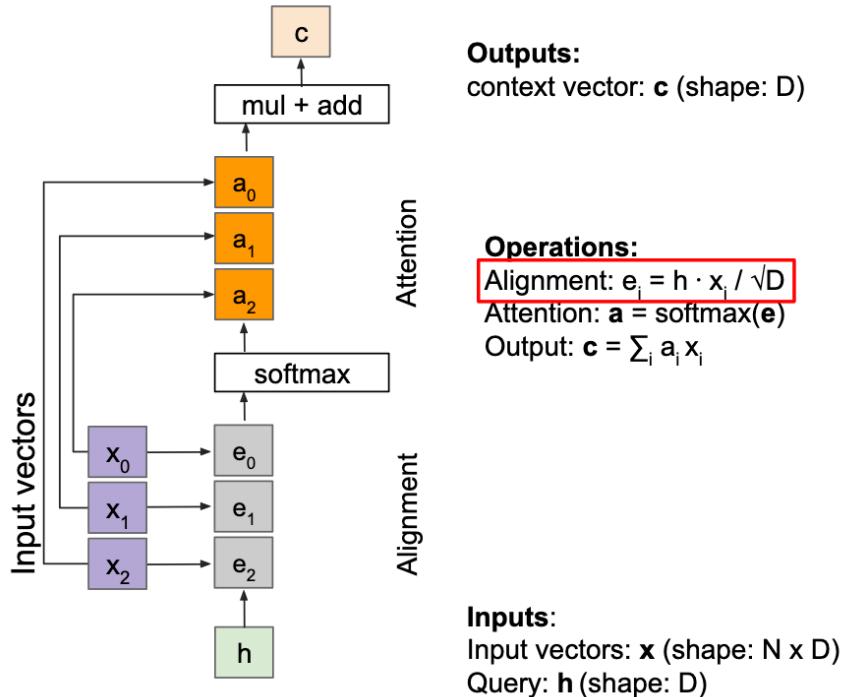
Operations:
Alignment: $e_i = h \cdot x_i$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $\mathbf{c} = \sum_i a_i x_i$

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

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

Inputs:
Input vectors: \mathbf{x} (shape: $N \times D$)
Query: \mathbf{h} (shape: D)

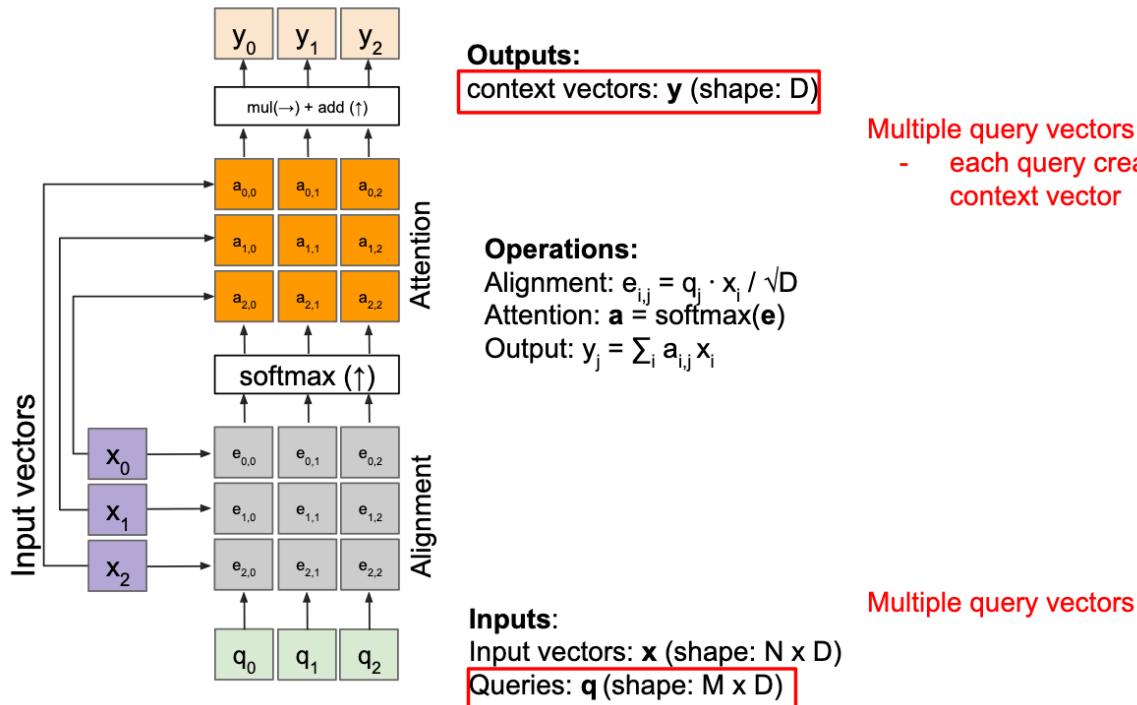
General Attention Layer



Change $f_{\text{att}}(\cdot)$ to a **scaled** simple dot product

- Larger dimensions means more terms in the dot product sum.
- So, the variance of the logits is higher. Large magnitude vectors will produce much higher logits.
- So, the post-softmax distribution has lower-entropy, assuming logits are IID.
- Ultimately, these large magnitude vectors will cause softmax to peak and assign very little weight to all others
- Divide by \sqrt{D} to reduce effect of large magnitude vectors

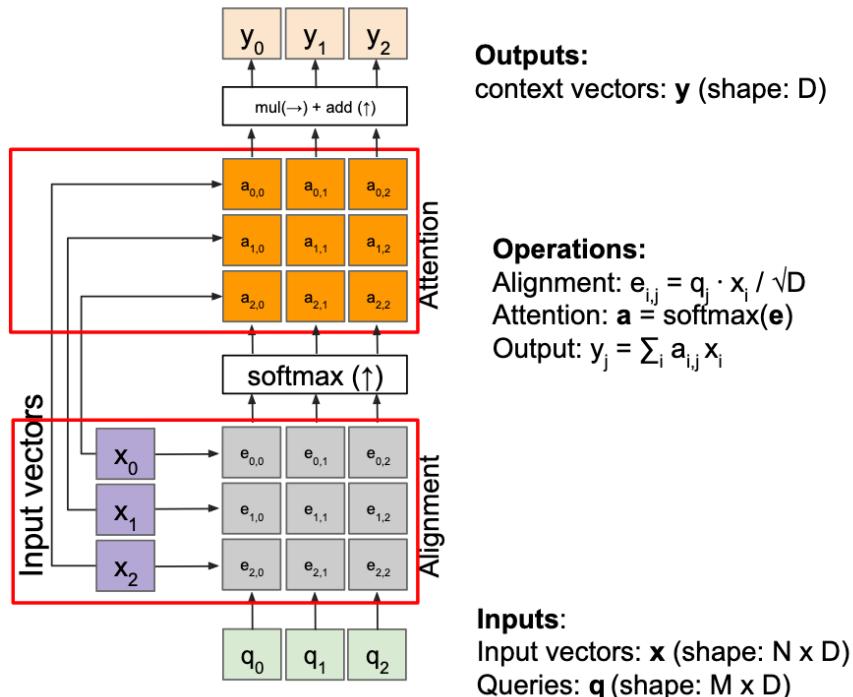
General Attention Layer



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

Multiple query vectors

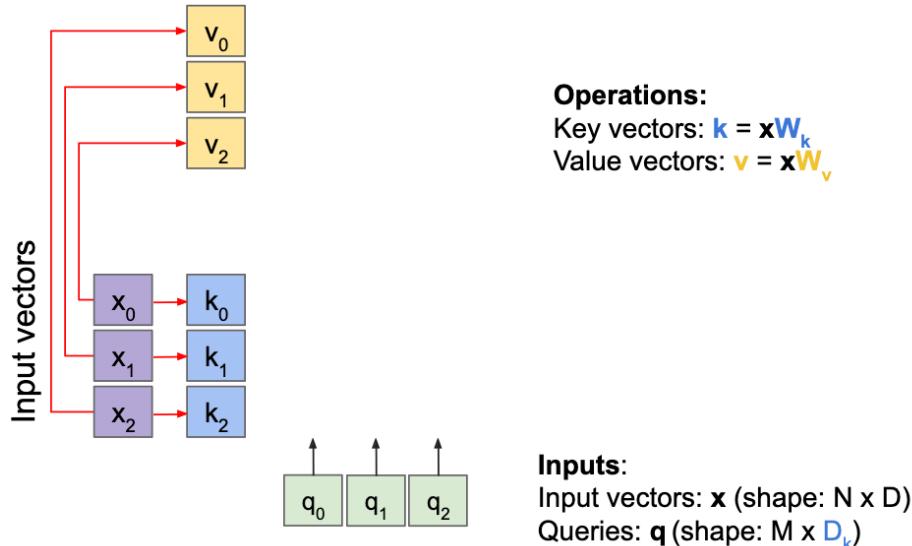
General Attention Layer



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.

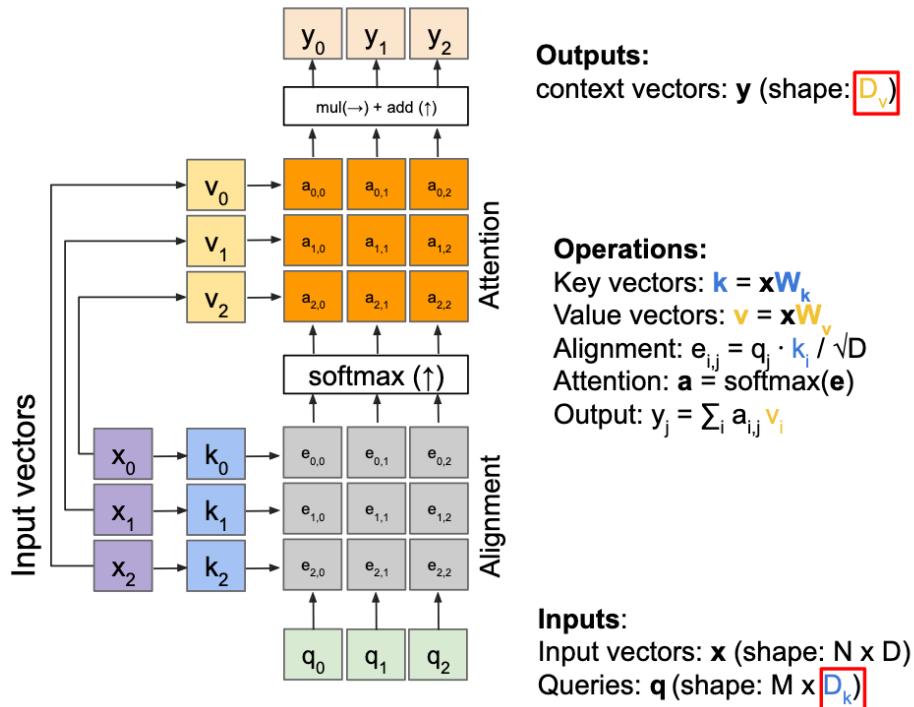
General Attention Layer



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

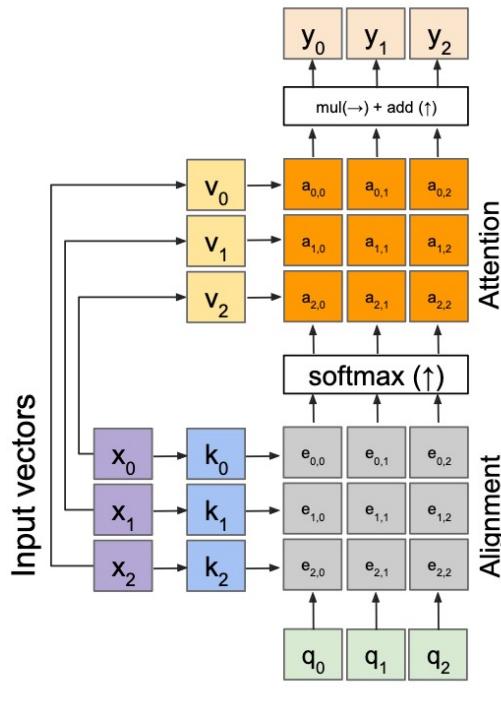


The input and output dimensions can now change depending on the key and value FC layers

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



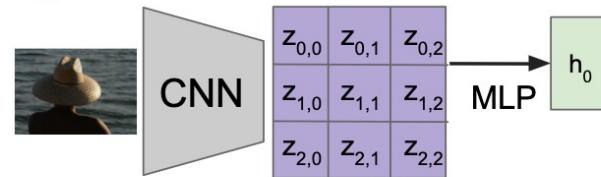
Outputs:
context vectors: \mathbf{y} (shape: D_v)

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: $y_j = \sum_i a_{i,j} \mathbf{v}_i$

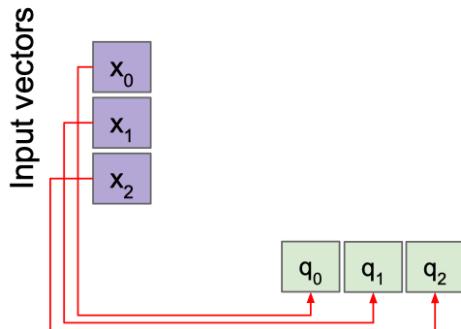
Inputs:
Input vectors: \mathbf{x} (shape: $N \times D$)
Queries: \mathbf{q} (shape: $M \times D_k$)

Recall that the query vector was a function of the input vectors

Encoder: $h_0 = f_w(\mathbf{z})$
where \mathbf{z} is spatial CNN features
 $f_w(\cdot)$ is an MLP



Self Attention Layer



Operations:

Key vectors: $k = xW_k$
Value vectors: $v = xW_v$
Query vectors: $q = xW_q$
Alignment: $e_{i,j} = q_i \cdot k_j / \sqrt{D}$
Attention: $a = \text{softmax}(e)$
Output: $y_j = \sum_i a_{i,j} v_i$

We can calculate the query vectors from the input vectors, therefore, defining a "self-attention" layer.

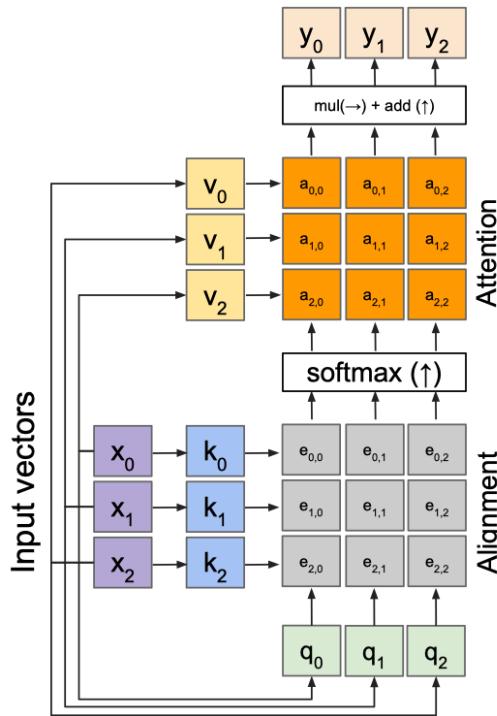
Instead, query vectors are calculated using a FC layer.

Inputs:

Input vectors: x (shape: $N \times D$)
Queries: q (shape: $M \times D_q$)

No input query vectors anymore

Self Attention Layer

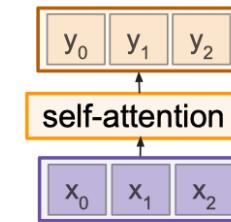
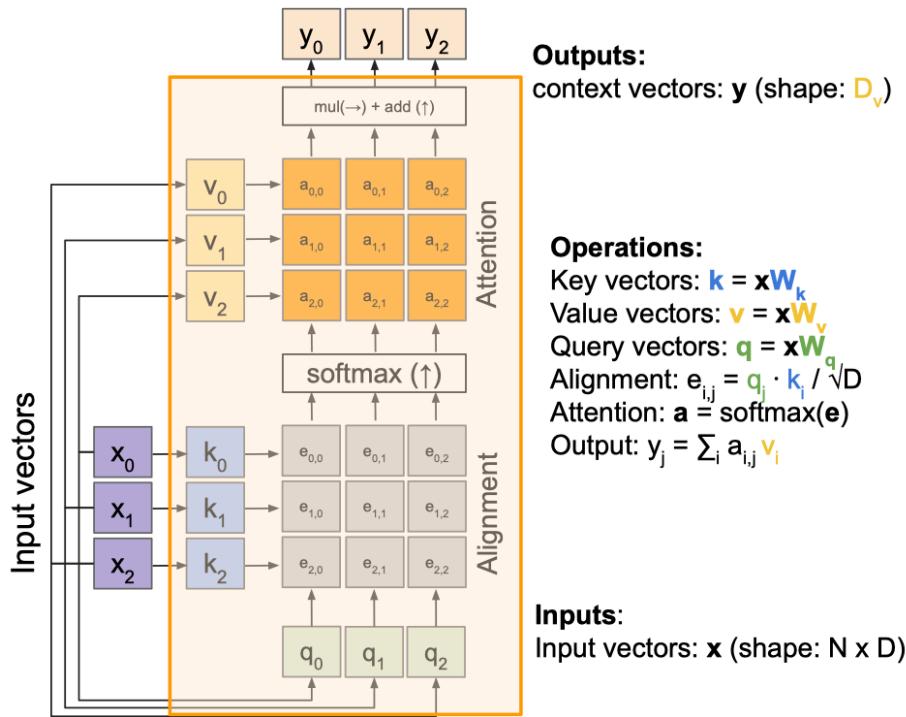


Outputs:
context vectors: \mathbf{y} (shape: D_y)

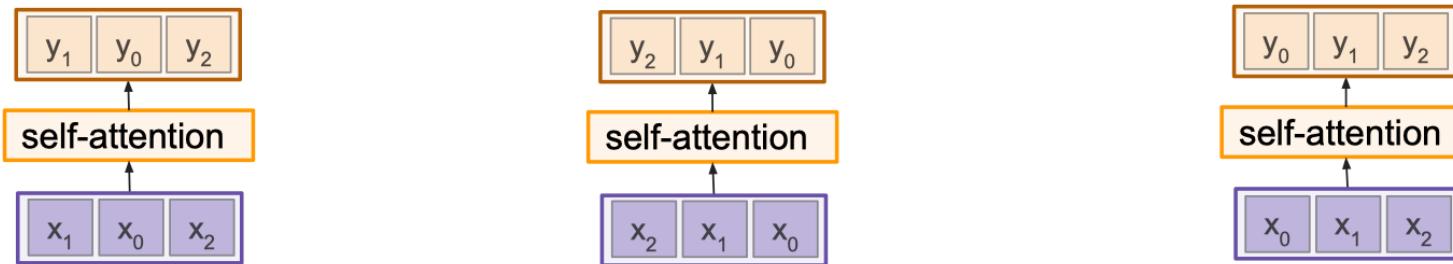
Operations:
 Key vectors: $\mathbf{k} = \mathbf{x}W_k$
 Value vectors: $\mathbf{v} = \mathbf{x}W_v$
 Query vectors: $\mathbf{q} = \mathbf{x}W_q$
 Alignment: $\mathbf{e}_{i,j} = \mathbf{q}_j \cdot \mathbf{k}_i / \sqrt{D}$
 Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
 Output: $\mathbf{y}_j = \sum_i \mathbf{a}_{i,j} \mathbf{v}_i$

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

Self Attention Layer – Attends Over Sets of Inputs



Self Attention Layer – Attends Over Sets of Inputs

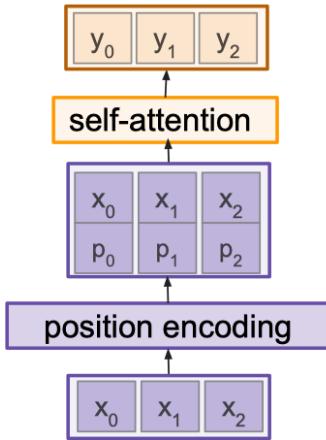


Permutation equivariant

Self-attention layer doesn't care about the orders of the inputs!

Problem: How can we encode ordered sequences like language or spatially ordered image features?

Positional Encoding



Concatenate/add special positional encoding p_j to each input vector x_j

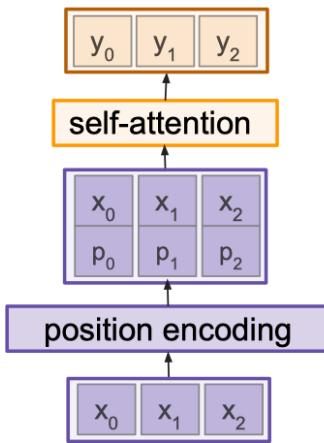
We use a function $pos: N \rightarrow \mathbb{R}^d$ to process the position j of the vector into a d -dimensional vector

So, $p_j = pos(j)$

Desiderata of $pos(\cdot)$:

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



Concatenate special positional encoding p_j to each input vector x_j

We use a function $pos: N \rightarrow \mathbb{R}^d$ to process the position j of the vector into a d -dimensional vector

So, $p_j = pos(j)$

Options for $pos(\cdot)$

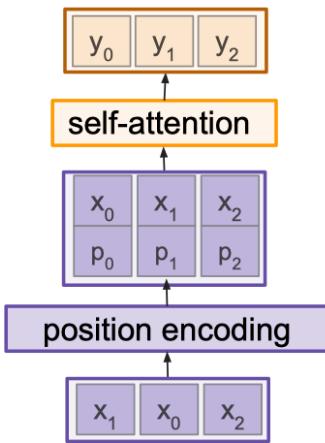
1. Learn a lookup table:
 - o Learn parameters to use for $pos(t)$ for $t \in [0, T]$
 - o Lookup table contains $T \times d$ parameters.

Desiderata of $pos(\cdot)$:

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**.

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

Positional Encoding



Concatenate special positional encoding p_j to each input vector x_j

We use a function $pos: N \rightarrow \mathbb{R}^d$ to process the position j of the vector into a d -dimensional vector

So, $p_j = pos(j)$

Options for $pos(\cdot)$

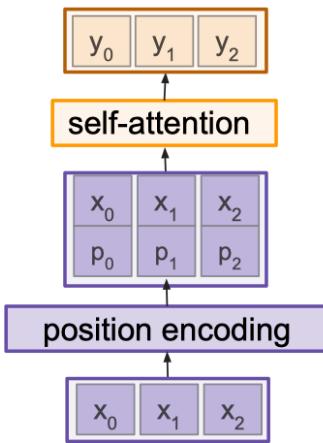
1. Learn a lookup table:
 - o Learn parameters to use for $pos(t)$ for $t \in [0, T]$
 - o Lookup table contains $T \times d$ parameters.
2. Design a fixed function with the desiderata

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

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

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

Positional Encoding



Concatenate special positional encoding p_j to each input vector x_j

We use a function $pos: N \rightarrow \mathbb{R}^d$ to process the position j of the vector into a d -dimensional vector

So, $p_j = pos(j)$

Options for $pos(\cdot)$

1. Learn a lookup table:
 - o Learn parameters to use for $pos(t)$ for $t \in [0, T]$
 - o Lookup table contains $T \times d$ parameters.
2. Design a fixed function with the desiderata

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

Intuition:

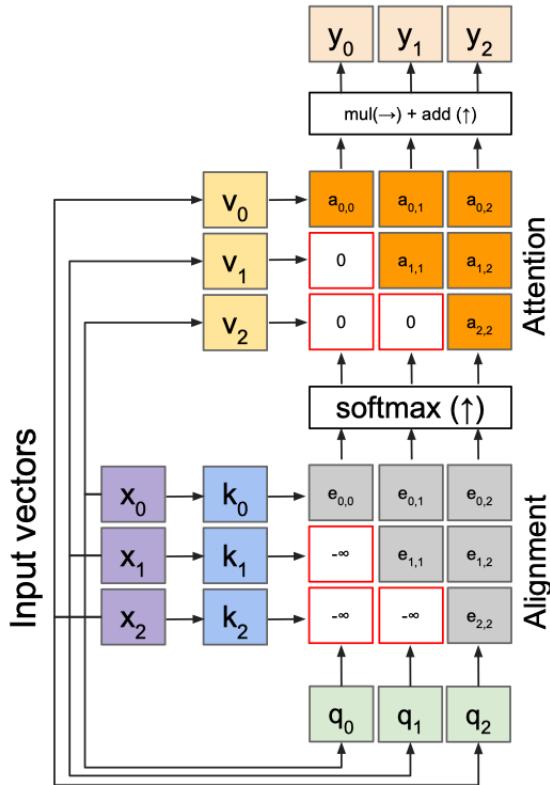
0 :	0 0 0 0	8 :	1 0 0 0
1 :	0 0 0 1	9 :	1 0 0 1
2 :	0 0 1 0	10 :	1 0 1 0
3 :	0 0 1 1	11 :	1 0 1 1
4 :	0 1 0 0	12 :	1 1 0 0
5 :	0 1 0 1	13 :	1 1 0 1
6 :	0 1 1 0	14 :	1 1 1 0
7 :	0 1 1 1	15 :	1 1 1 1

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

[image source](#)

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

Masked Self-attention Layer



Outputs:

context vectors: \mathbf{y} (shape: D_v)

Operations:

Key vectors: $\mathbf{k} = \mathbf{x}\mathbf{W}_k$

Value vectors: $\mathbf{v} = \mathbf{x}\mathbf{W}_v$

Query vectors: $\mathbf{q} = \mathbf{x}\mathbf{W}_q$

Alignment: $\mathbf{e}_{i,j} = \mathbf{q}_j \cdot \mathbf{k}_i / \sqrt{D}$

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

Output: $\mathbf{y}_j = \sum_i \mathbf{a}_{i,j} \mathbf{v}_i$

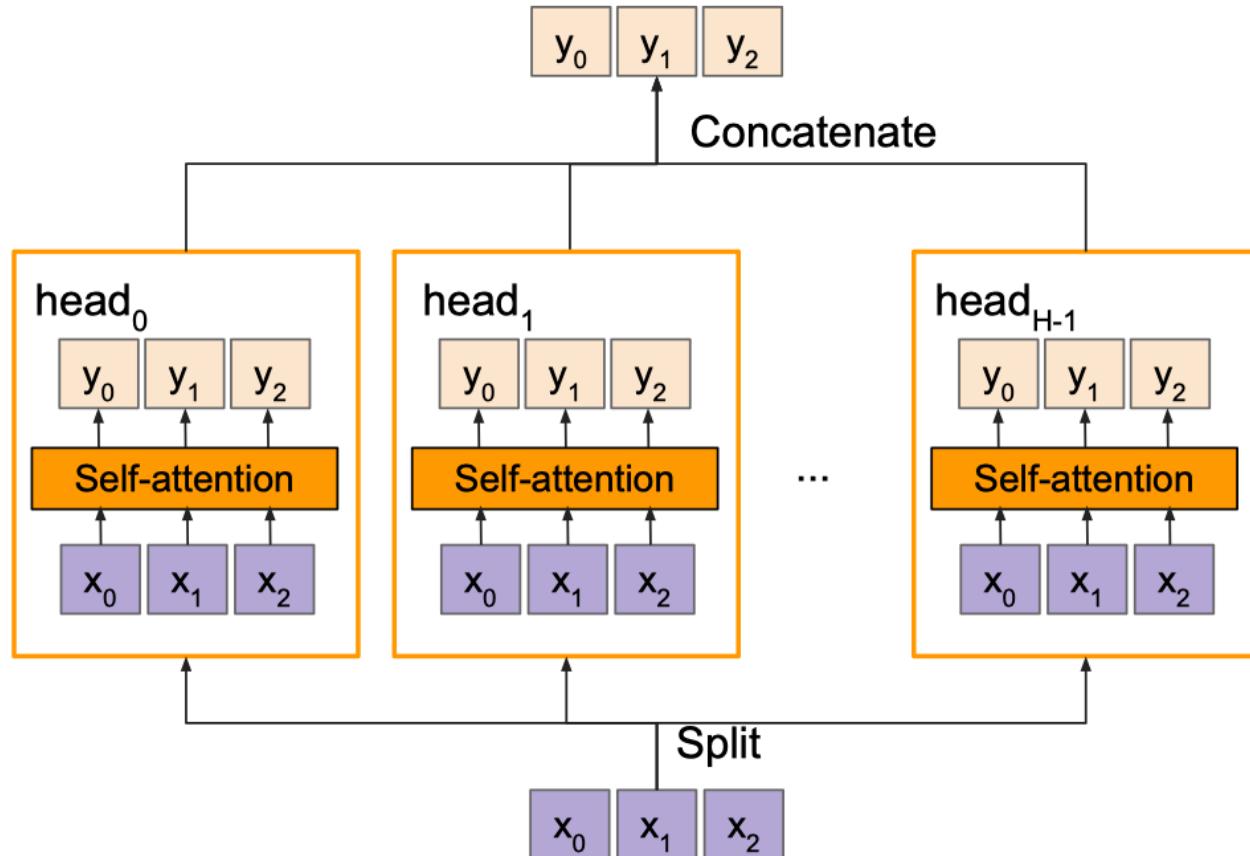
- Prevent vectors from looking at future vectors.
- Manually set alignment scores to $-\infty$

Inputs:

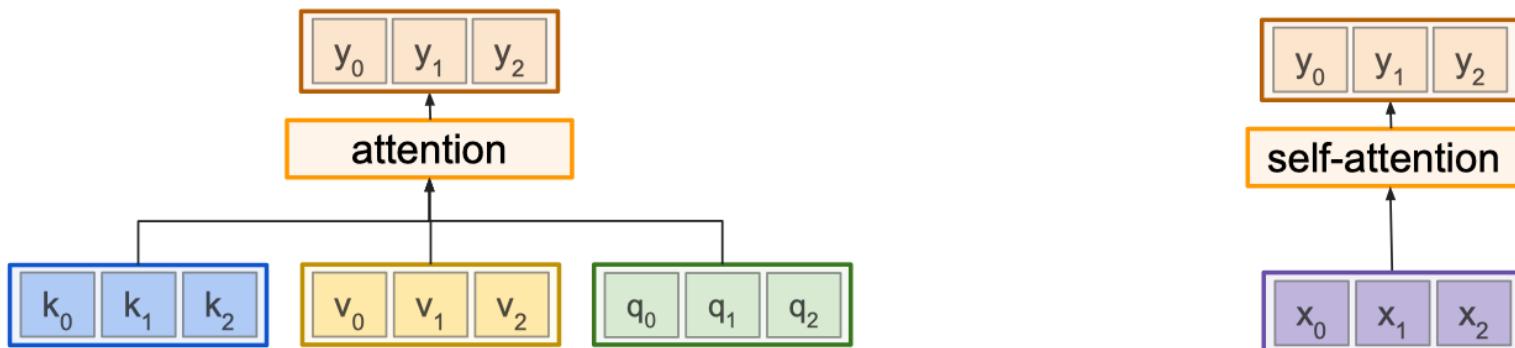
Input vectors: \mathbf{x} (shape: $N \times D$)

Multi-head Self-attention Layer

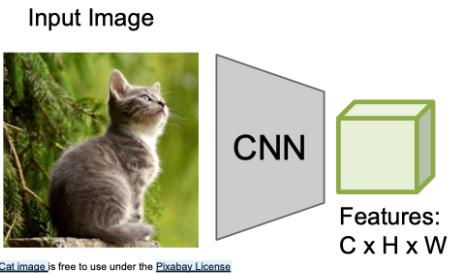
- Multiple self-attention heads in parallel



General Attention Versus Self-attention



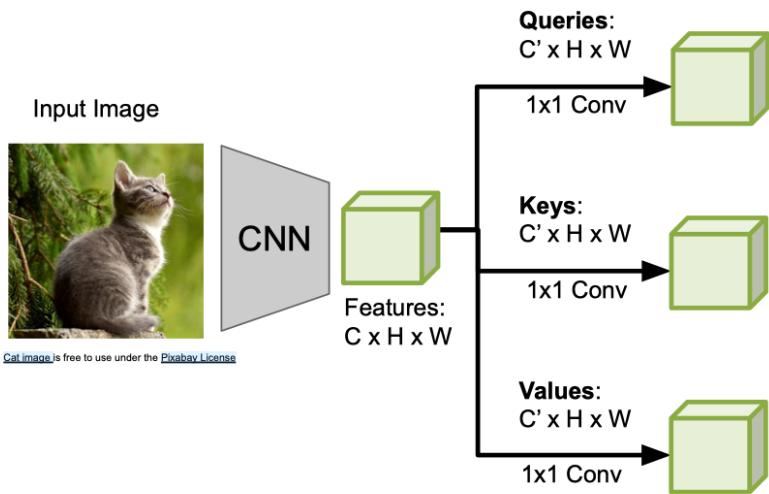
Example: CNN with Self-Attention



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

Slide credit: Justin Johnson

Example: CNN with Self-Attention

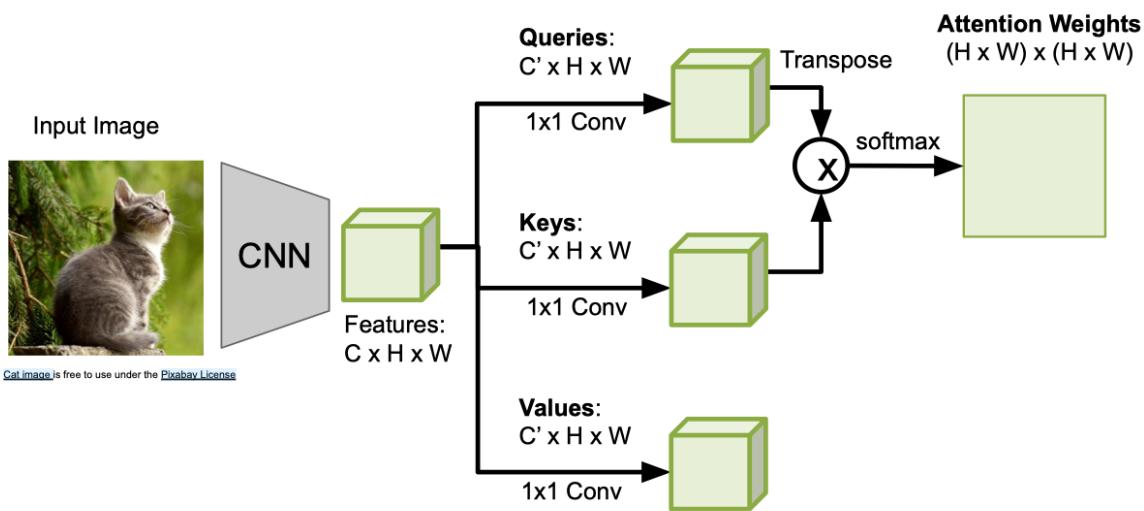


Cat image is free to use under the [Pixabay License](#).

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

Slide credit: Justin Johnson

Example: CNN with Self-Attention

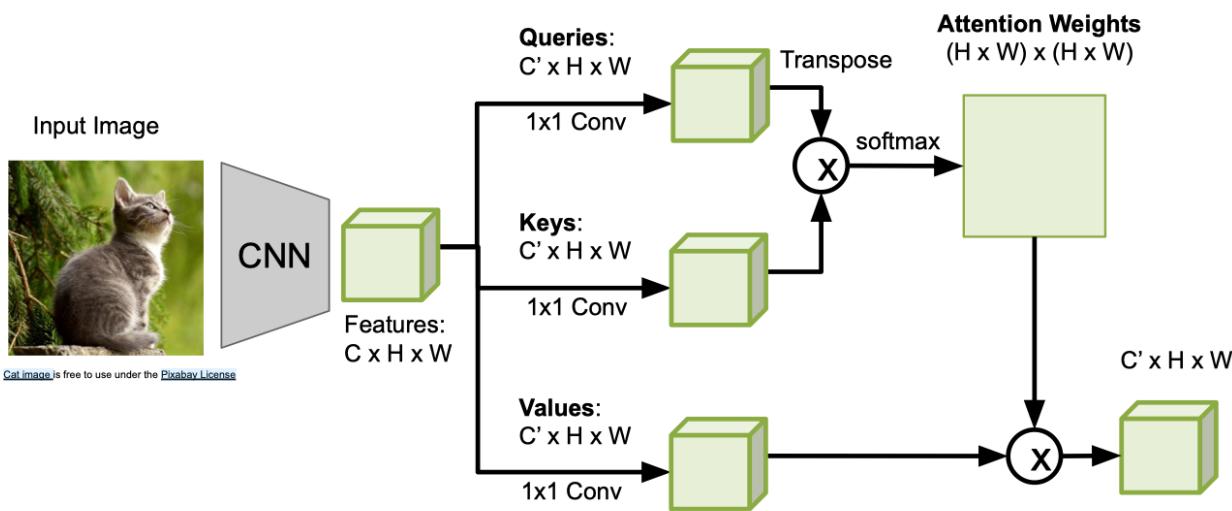


Cat image is free to use under the Pixabay License

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

Slide credit: Justin Johnson

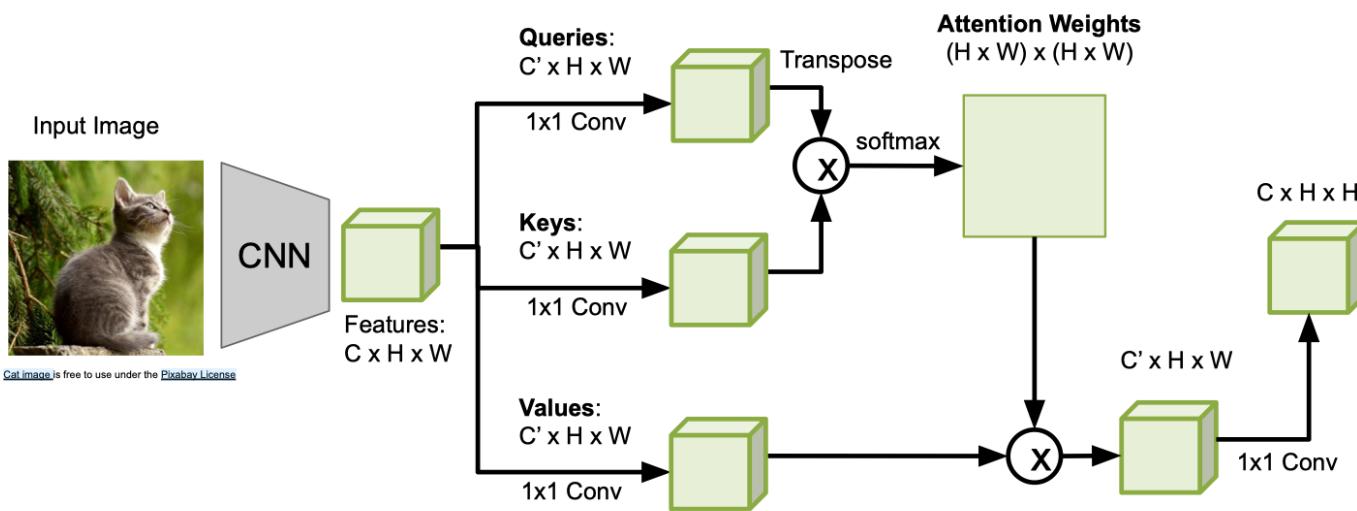
Example: CNN with Self-Attention



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

Slide credit: Justin Johnson

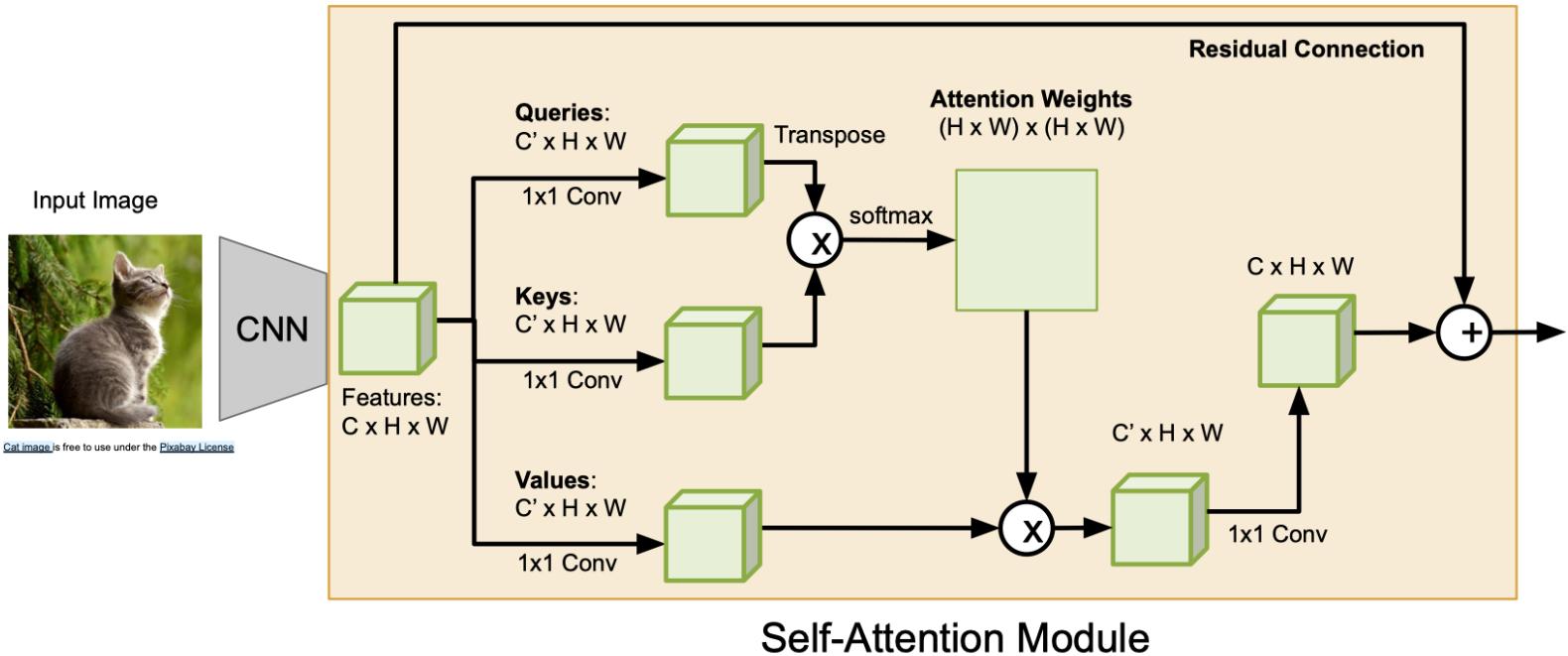
Example: CNN with Self-Attention



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

Slide credit: Justin Johnson

Example: CNN with Self-Attention



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

Slide credit: Justin Johnson

Transformer

Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

Lukasz Kaiser*
Google Brain
lukasz.kaiser@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

"ImageNet Moment for Natural Language Processing"

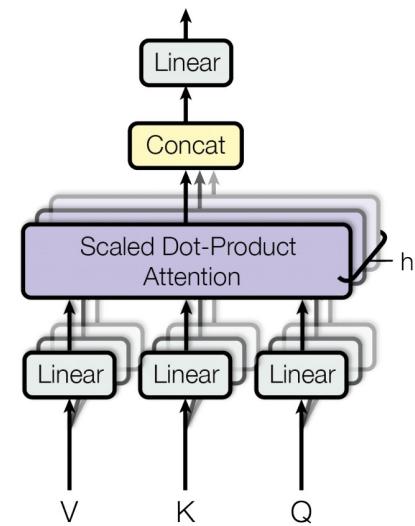
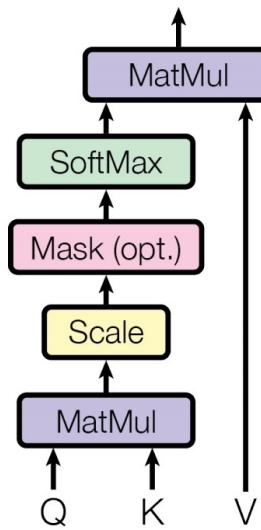
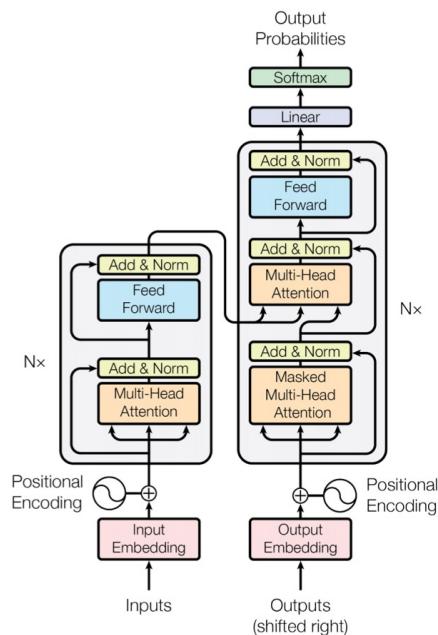
Pretraining:

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



Comparing RNNs to Transformer

RNNs

(+) LSTMs work reasonably well for long sequences.

(-) Expects an ordered sequences of inputs

(-) Sequential computation: subsequent hidden states can only be computed after the previous ones are done.

Transformer:

(+) Good at long sequences. Each attention calculation looks at all inputs.

(+) Can operate over unordered sets or ordered sequences with positional encodings.

(+) Parallel computation: All alignment and attention scores for all inputs can be done in parallel.

(-) Requires a lot of memory: $N \times M$ alignment and attention scalers need to be calculated and stored for a single self-attention head. (but GPUs are getting bigger and better)

Image Captioning using Transformers

Input: Image I

Output: Sequence $\mathbf{y} = y_1, y_2, \dots, y_T$

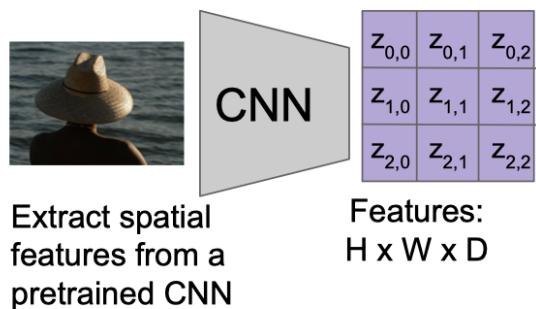


Image Captioning using Transformers

Input: Image I

Output: Sequence $\mathbf{y} = y_1, y_2, \dots, y_T$

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

where \mathbf{z} is spatial CNN features

$T_w(\cdot)$ is the transformer encoder

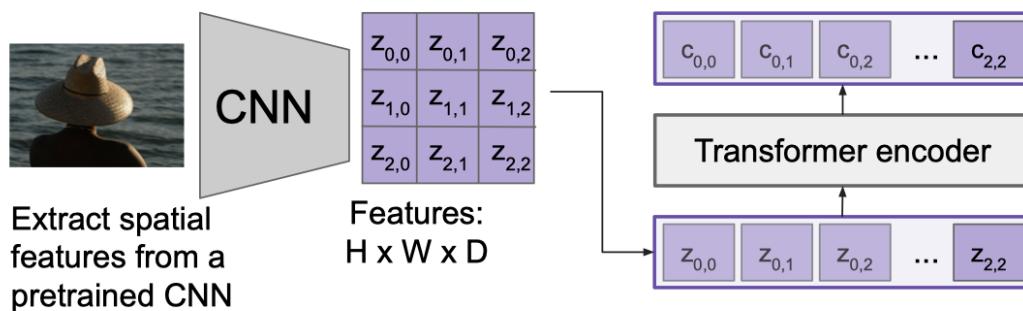


Image Captioning using Transformers

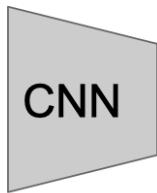
Input: Image I

Output: Sequence $\mathbf{y} = y_1, y_2, \dots, y_T$

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

where \mathbf{z} is spatial CNN features

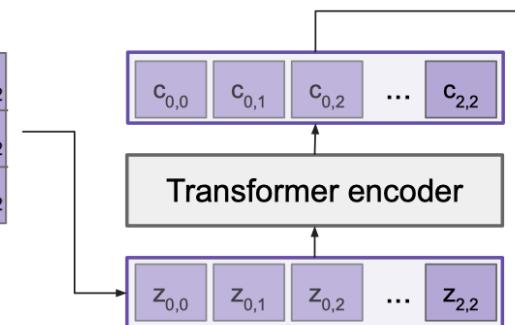
$T_w(\cdot)$ is the transformer encoder



$z_{0,0}$	$z_{0,1}$	$z_{0,2}$
$z_{1,0}$	$z_{1,1}$	$z_{1,2}$
$z_{2,0}$	$z_{2,1}$	$z_{2,2}$

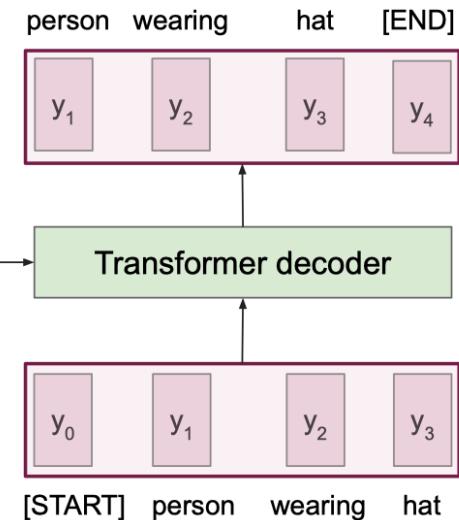
Features:
 $H \times W \times D$

Extract spatial
features from a
pretrained CNN

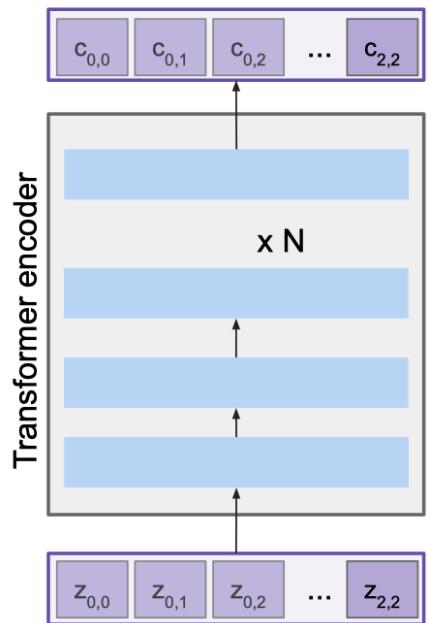


Decoder: $y_t = T_D(y_{0:t-1}, \mathbf{c})$

where $T_D(\cdot)$ is the transformer decoder



The Transformer Encoder Block

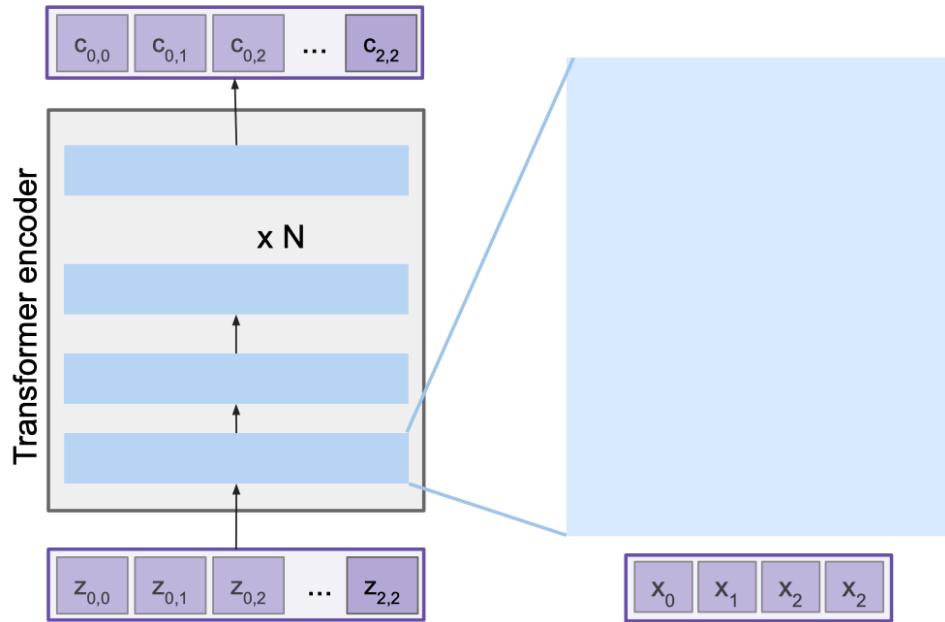


Made up of N encoder blocks.

In vaswani et al. $N = 6$, $D_q = 512$

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

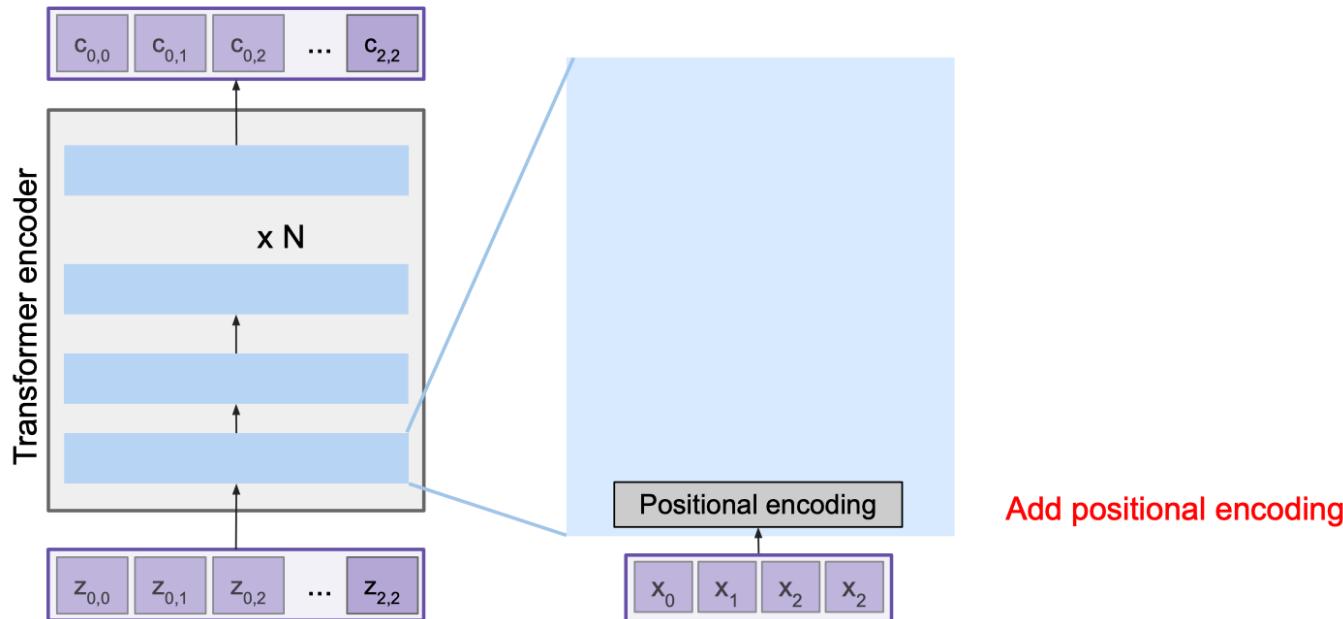
The Transformer Encoder Block



Let's dive into one encoder block

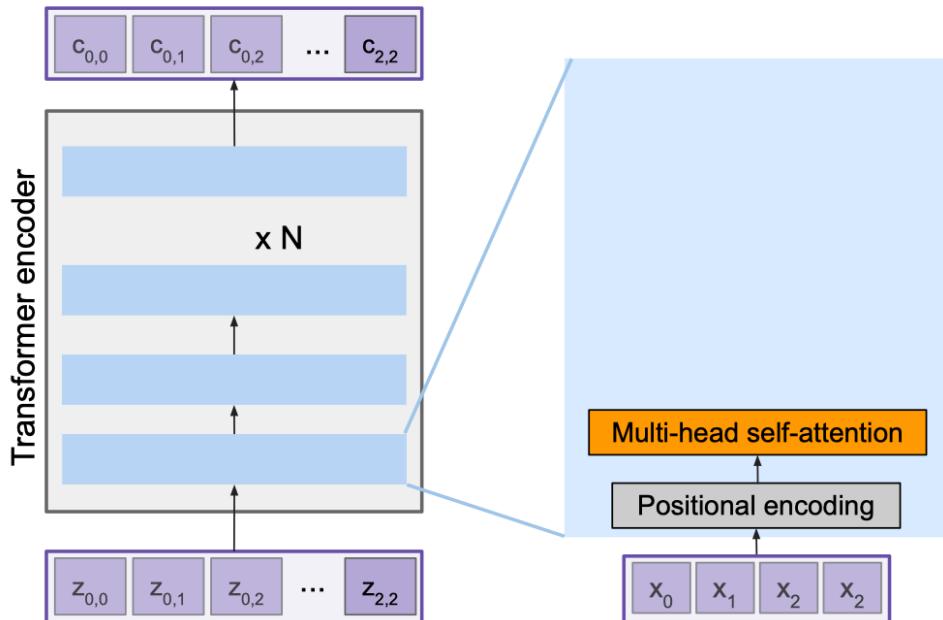
Vaswani et al, "Attention is all you need", NeurIPS 2017

The Transformer Encoder Block



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

The Transformer Encoder Block

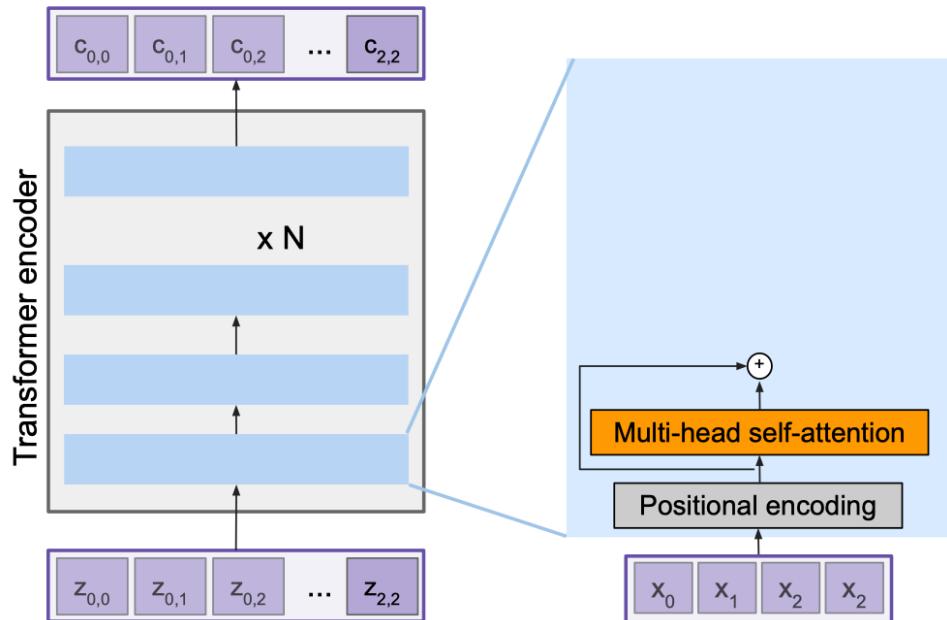


Attention attends over all the vectors

Add positional encoding

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

The Transformer Encoder Block



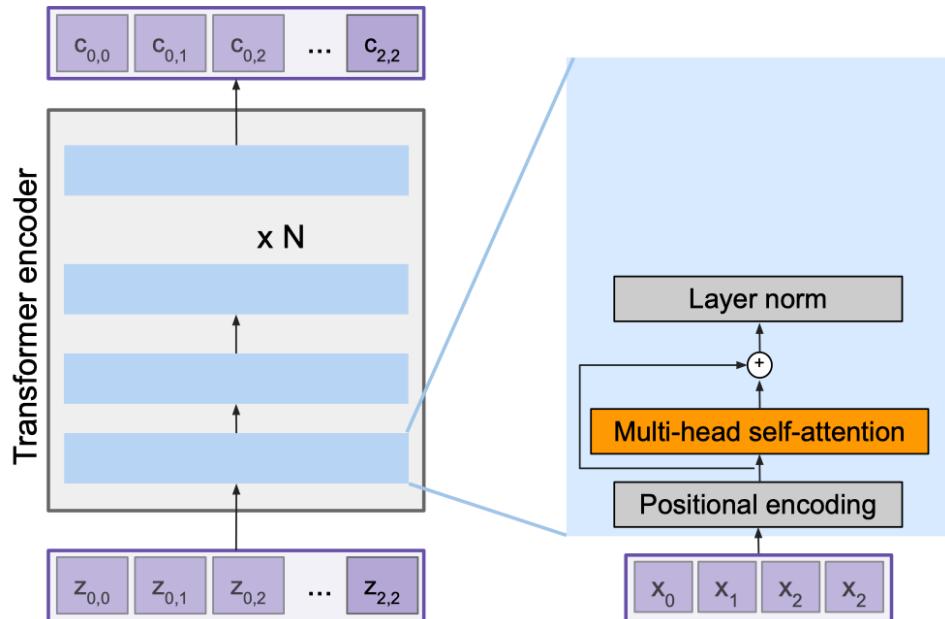
Residual connection

Attention attends over all the vectors

Add positional encoding

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

The Transformer Encoder Block



LayerNorm over each vector individually

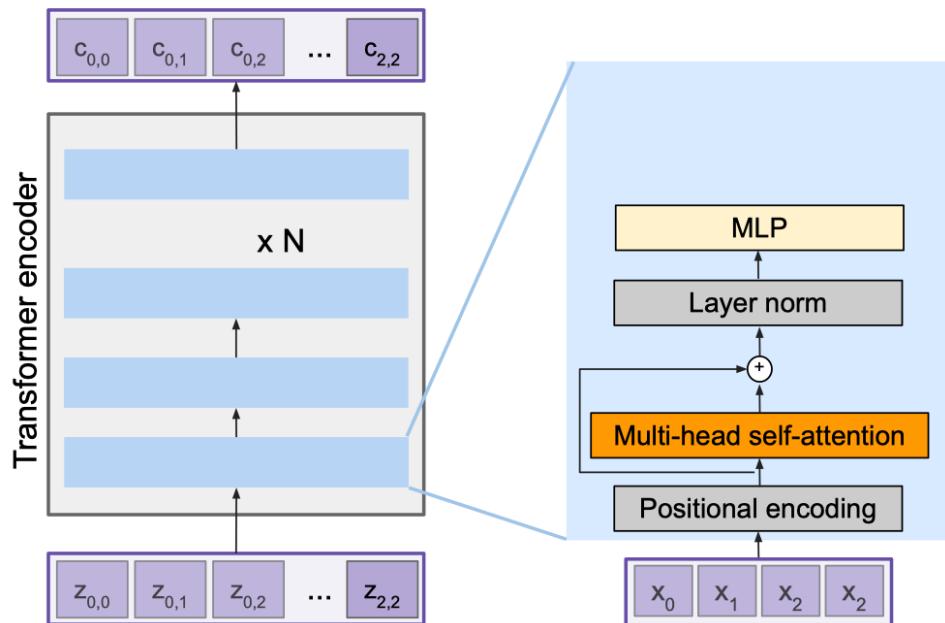
Residual connection

Attention attends over all the vectors

Add positional encoding

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

The Transformer Encoder Block



MLP over each vector individually

LayerNorm over each vector individually

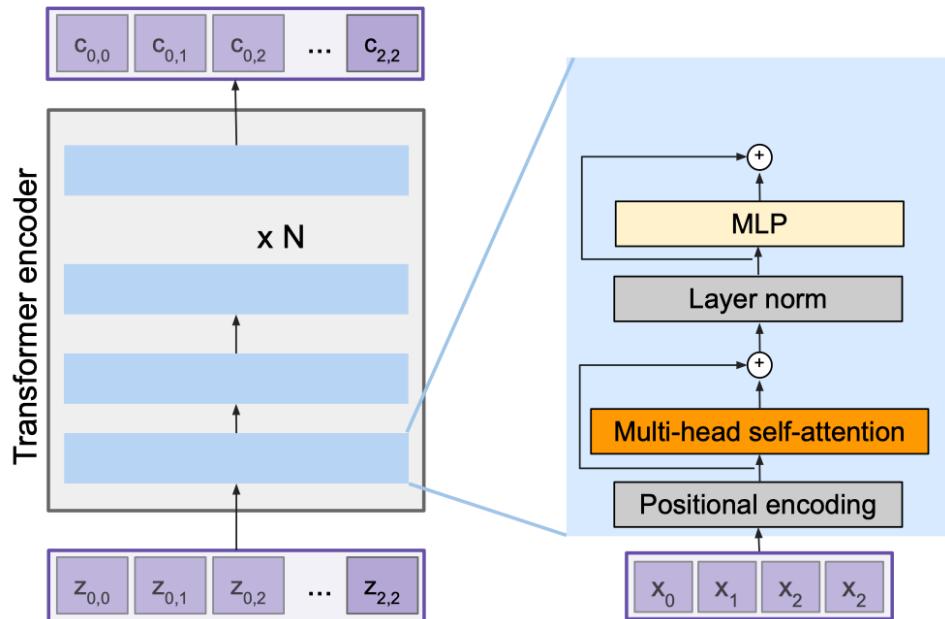
Residual connection

Attention attends over all the vectors

Add positional encoding

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

The Transformer Encoder Block



Residual connection

MLP over each vector individually

LayerNorm over each vector individually

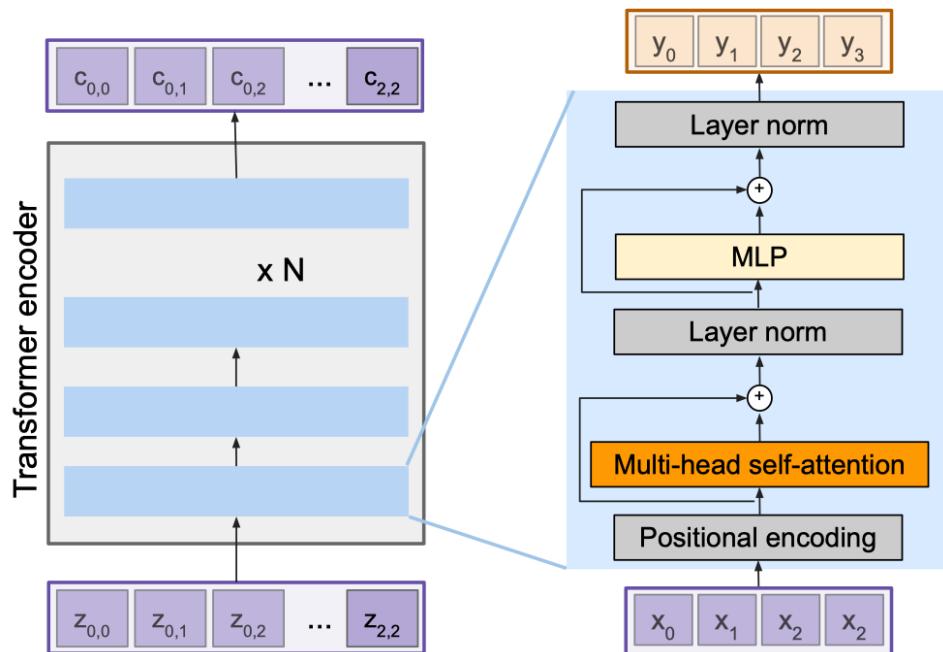
Residual connection

Attention attends over all the vectors

Add positional encoding

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

The Transformer Encoder Block



Transformer Encoder Block:

Inputs: Set of vectors \mathbf{x}

Outputs: Set of vectors \mathbf{y}

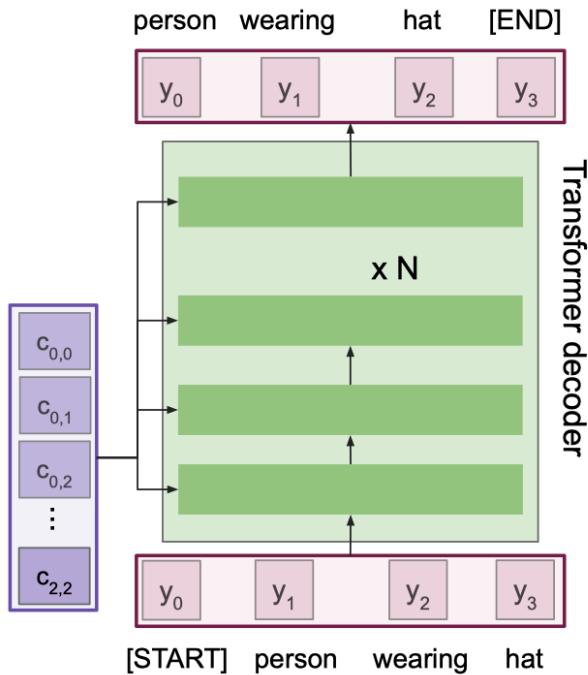
Self-attention is the only interaction between vectors.

Layer norm and MLP operate independently per vector.

Highly scalable, highly parallelizable, but high memory usage.

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

The Transformer Decoder Block

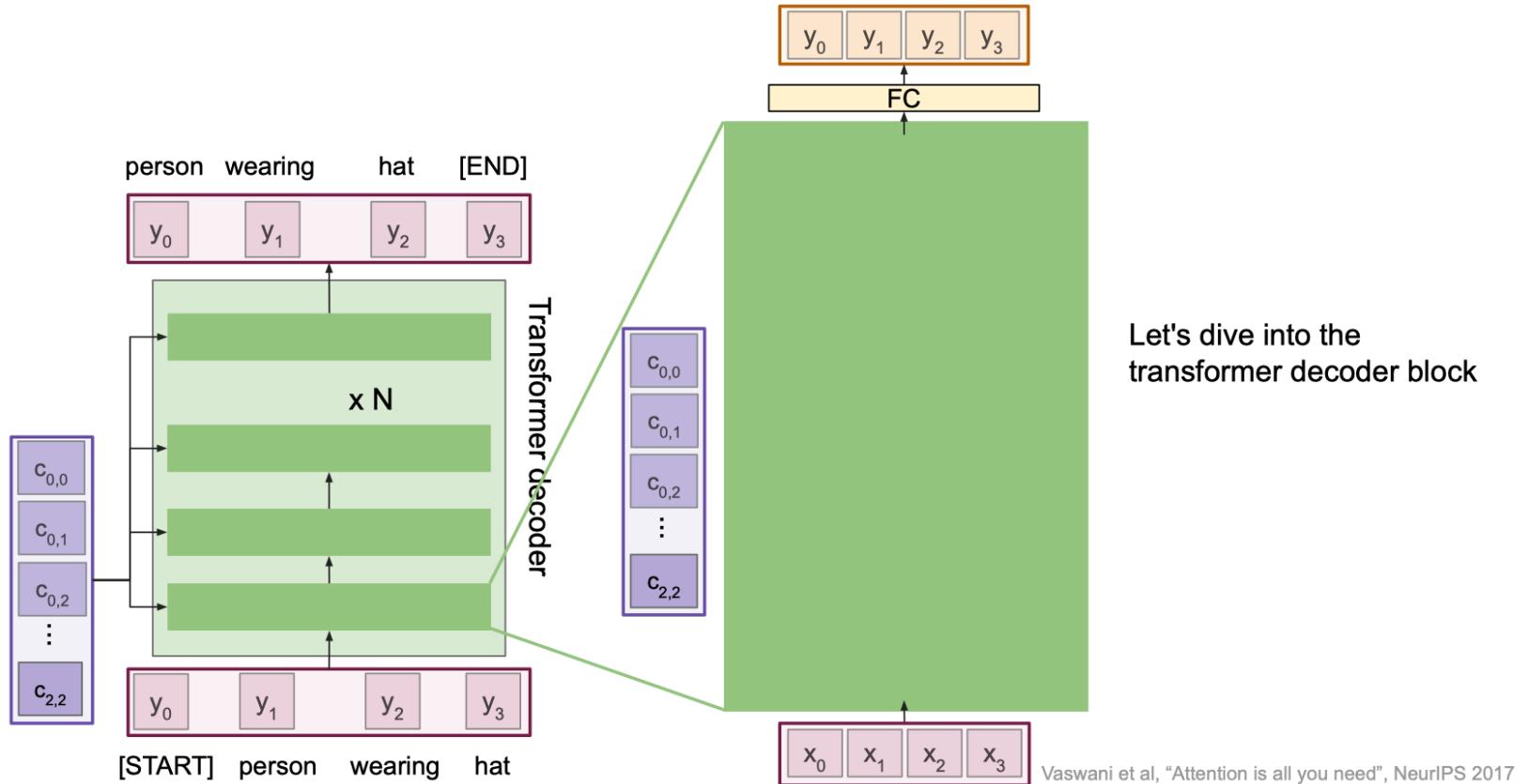


Made up of N decoder blocks.

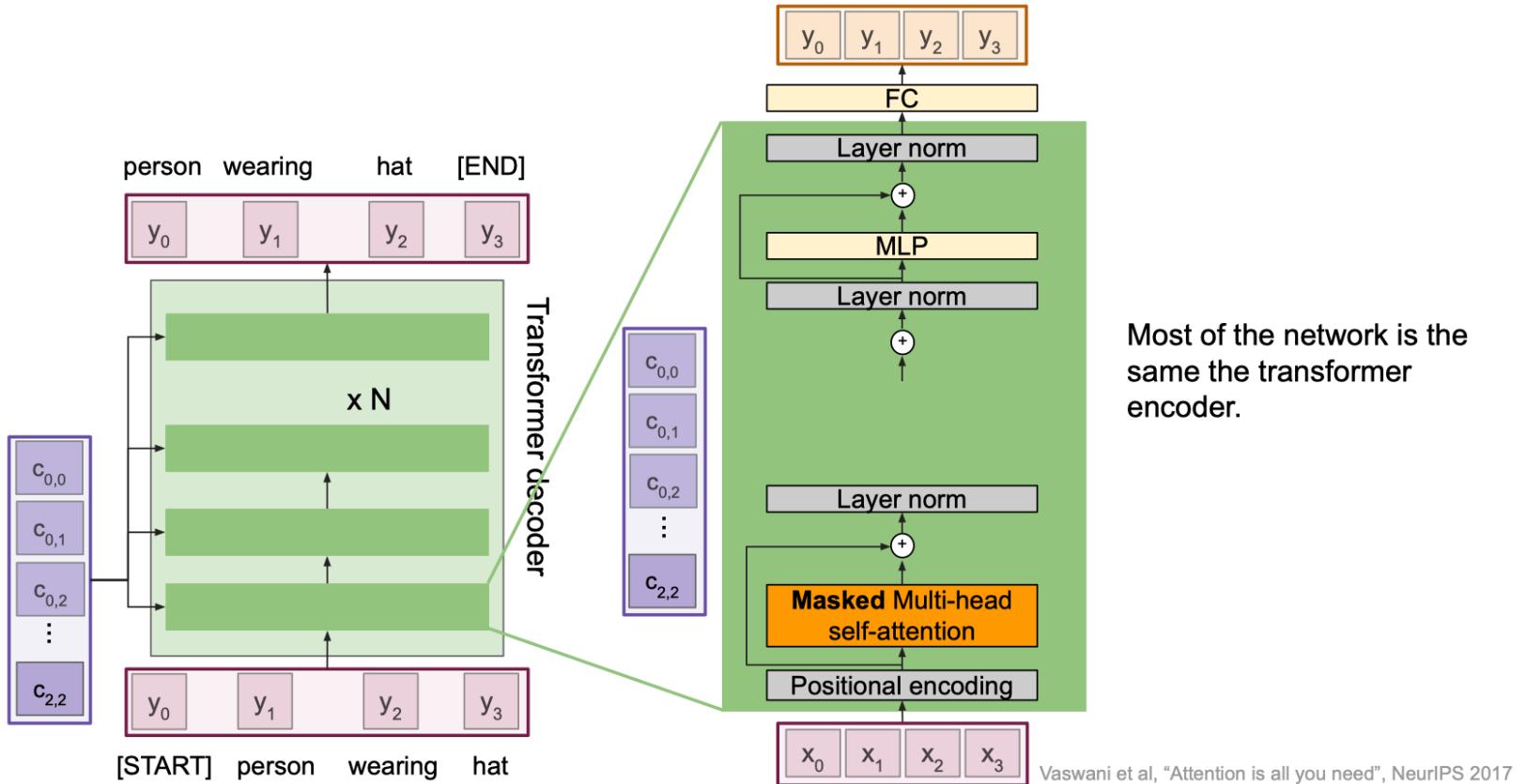
In vaswani et al. $N = 6, D_q = 512$

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

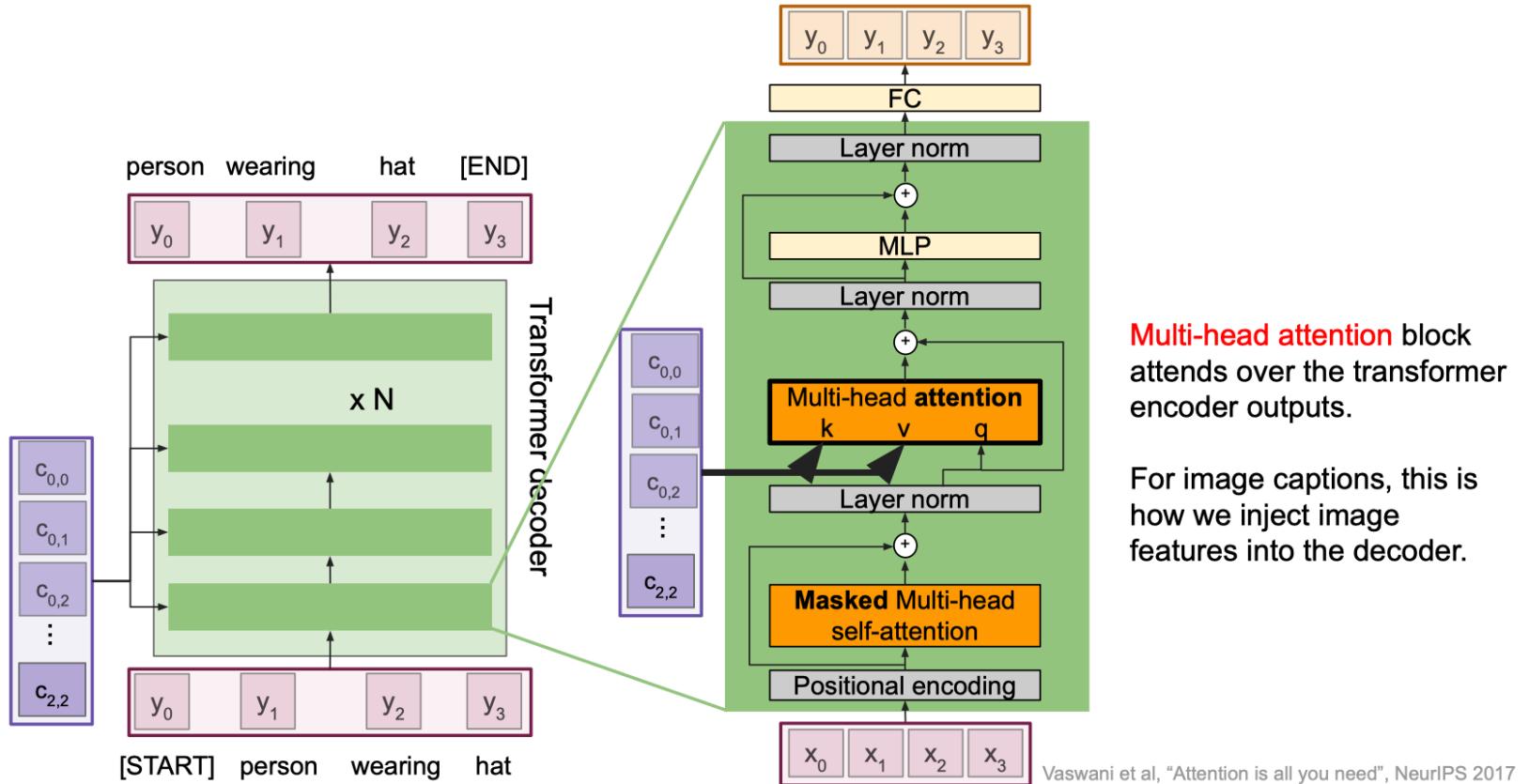
The Transformer Decoder Block



The Transformer Decoder Block



The Transformer Decoder Block



The Transformer Decoder Block

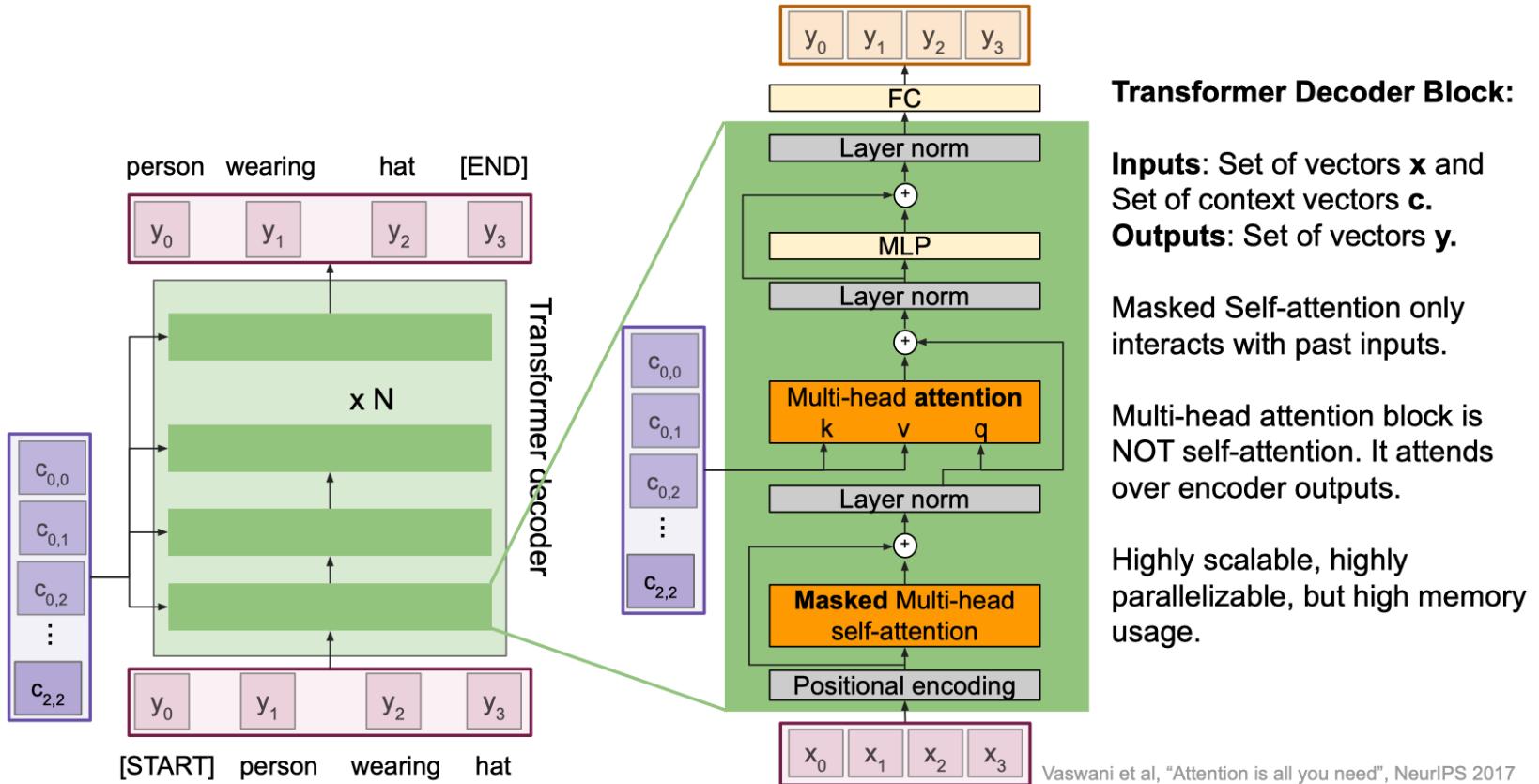


Image Captioning using Transformers

- No recurrence at all

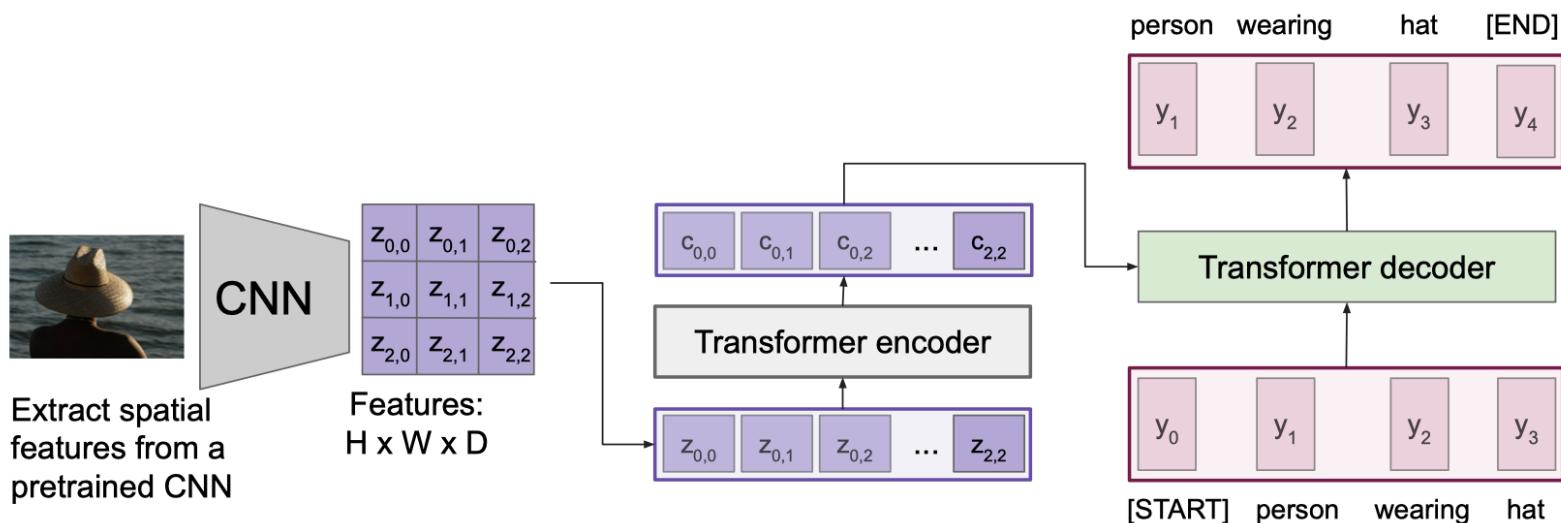


Image Captioning using Transformers

- Perhaps we don't need convolutions at all?

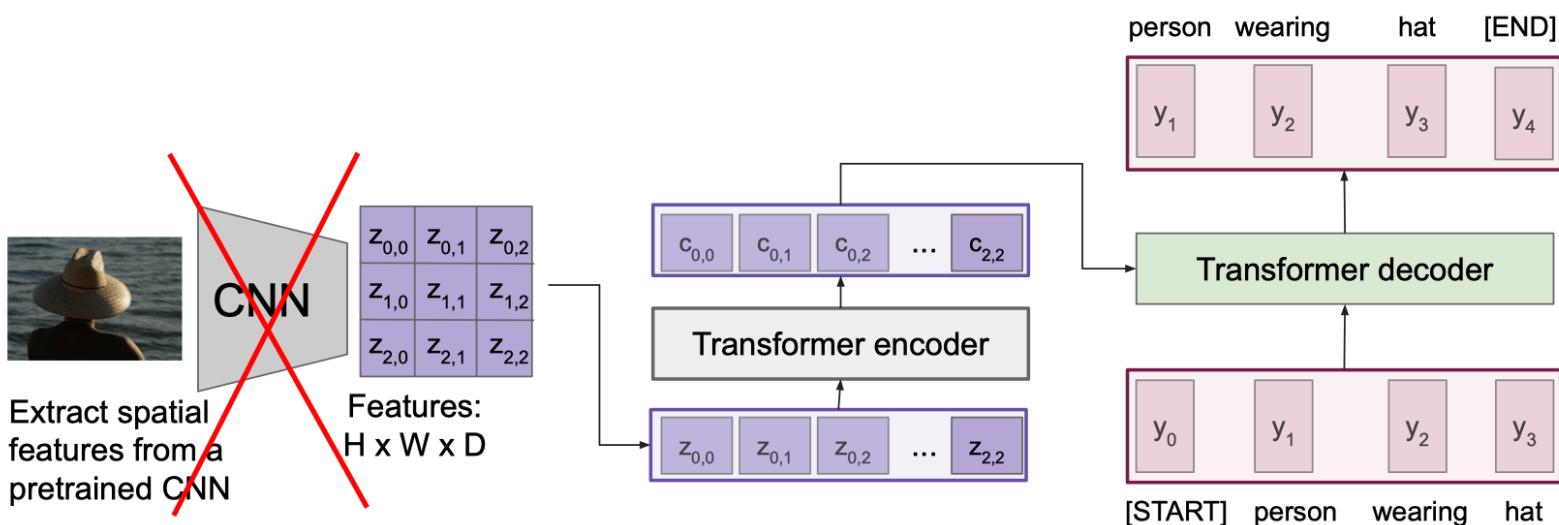
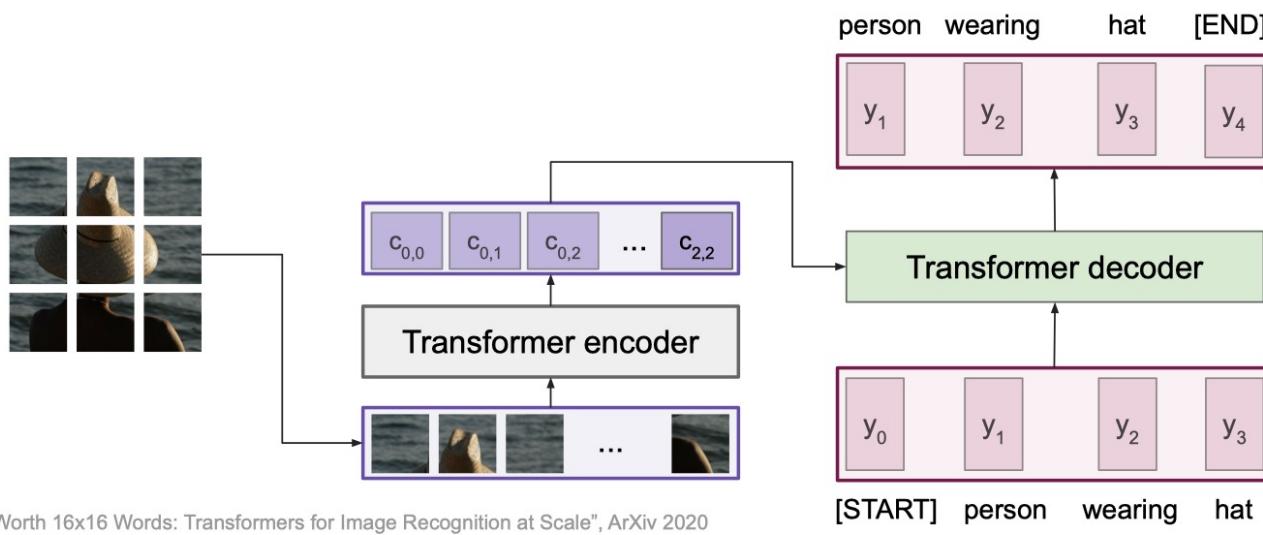


Image Captioning using ONLY Transformers

- **Transformers from pixels to language**



Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ArXiv 2020
[Colab link](#) to an implementation of vision transformers

Vision Transformers vs. ResNets

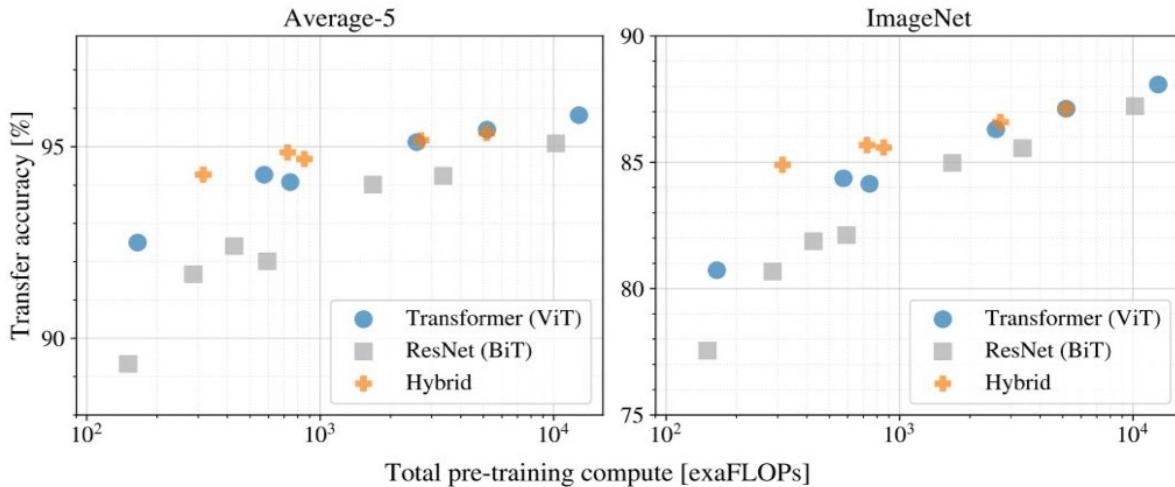
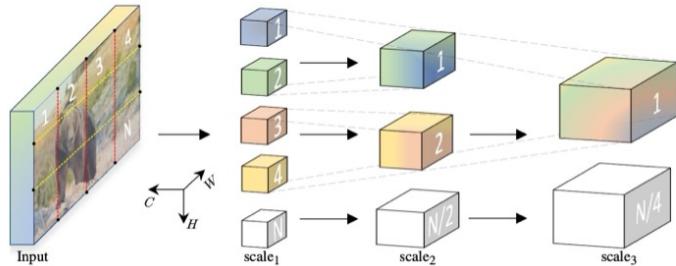


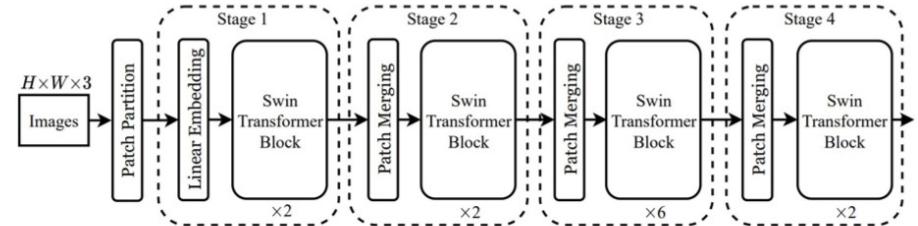
Figure 5: Performance versus cost for different architectures: Vision Transformers, ResNets, and hybrids. Vision Transformers generally outperform ResNets with the same computational budget. Hybrids improve upon pure Transformers for smaller model sizes, but the gap vanishes for larger models.

Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ArXiv 2020
[Colab link](#) to an implementation of vision transformers

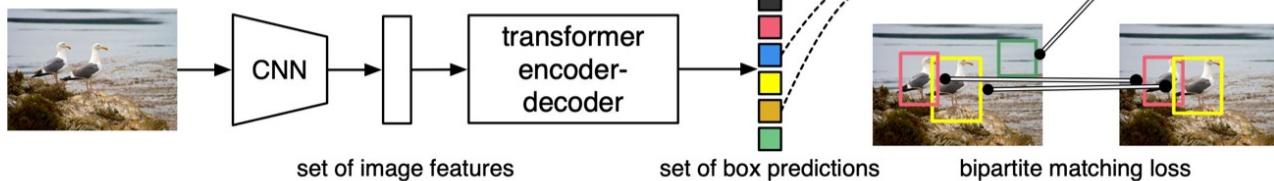
Vision Transformers



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

Summary

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

Questions?