

# Lecture 12:

# Attention and Transformers

# Administrative: Project Proposals

- Please have one member of your group fill out this [form](#) (linked on Ed) **ASAP**.
  - ~5 groups haven't filled this out yet
  - cannot assign your group a TA without this
  - should take 60 seconds to fill out

# Administrative: Assignment 3

Due 5/11 11:59pm

- Normalization Layers,
- Dropout,
- CNNs

# Administrative: Fridays

This Friday

**Scaling Laws**

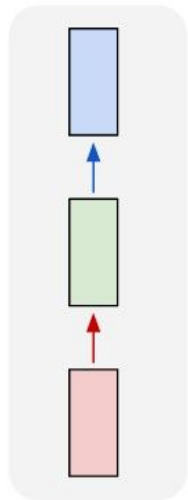
# Administrative: Exam

During lecture on May 20th

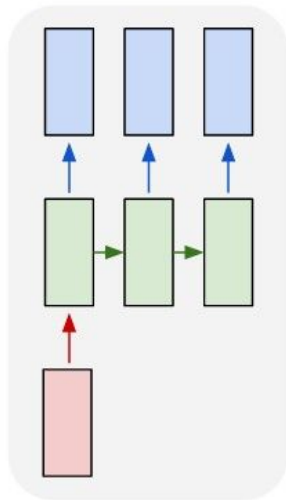
Refer to the [exam archive](#) for practice materials

# Last Time: Recurrent Neural Networks

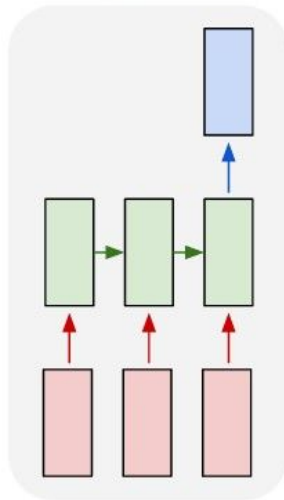
one to one



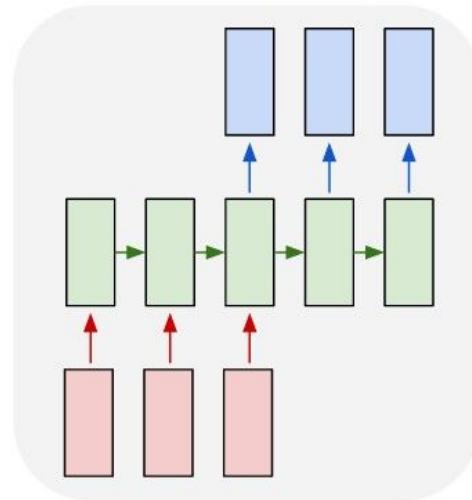
one to many



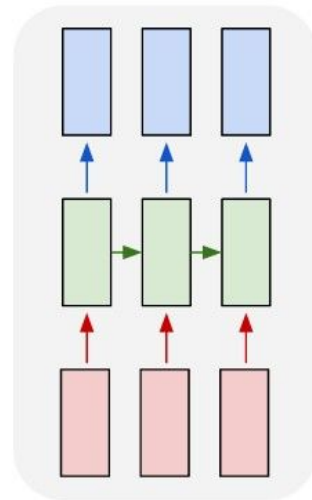
many to one



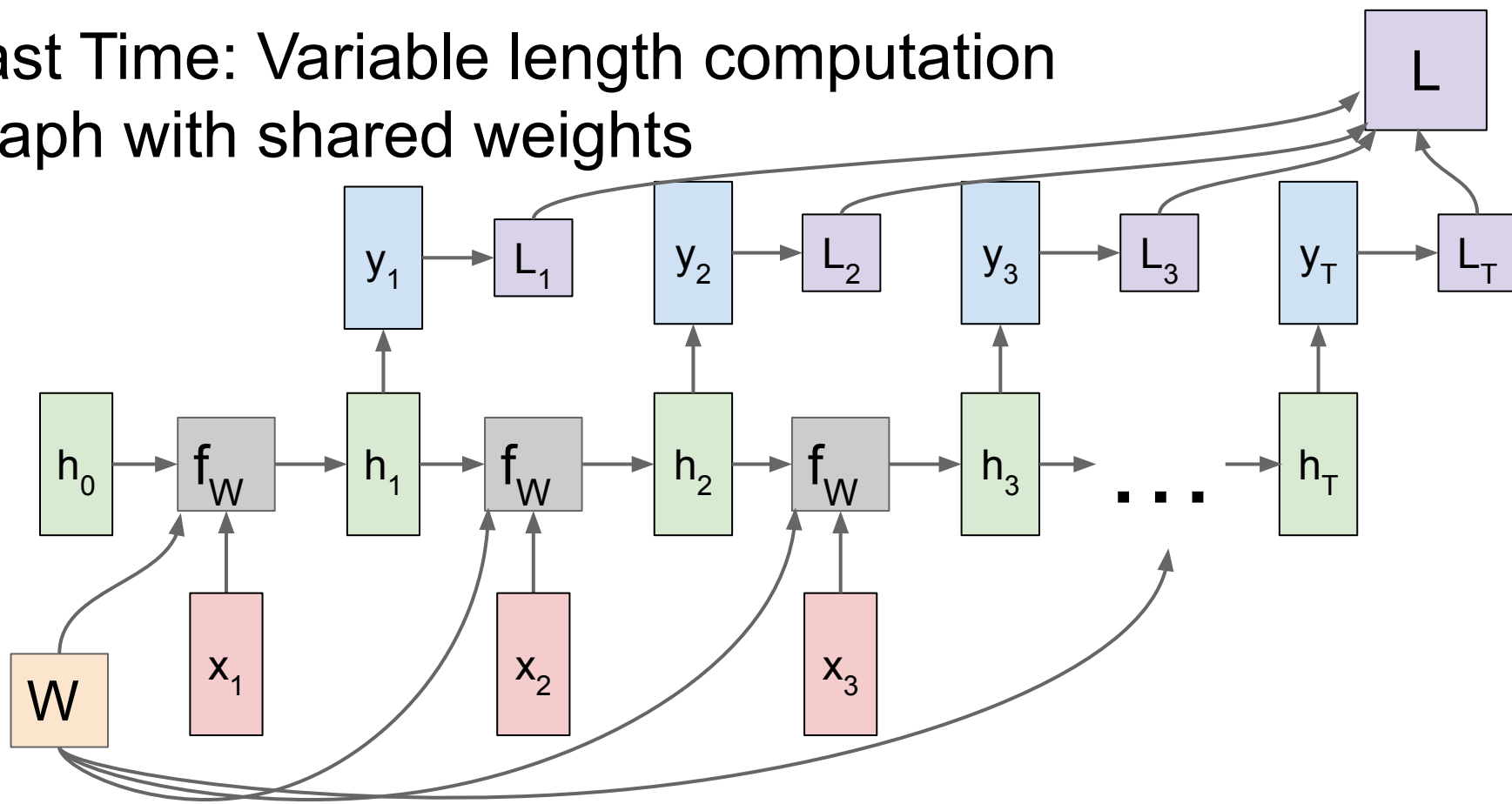
many to many



many to many



# Last Time: Variable length computation graph with shared weights



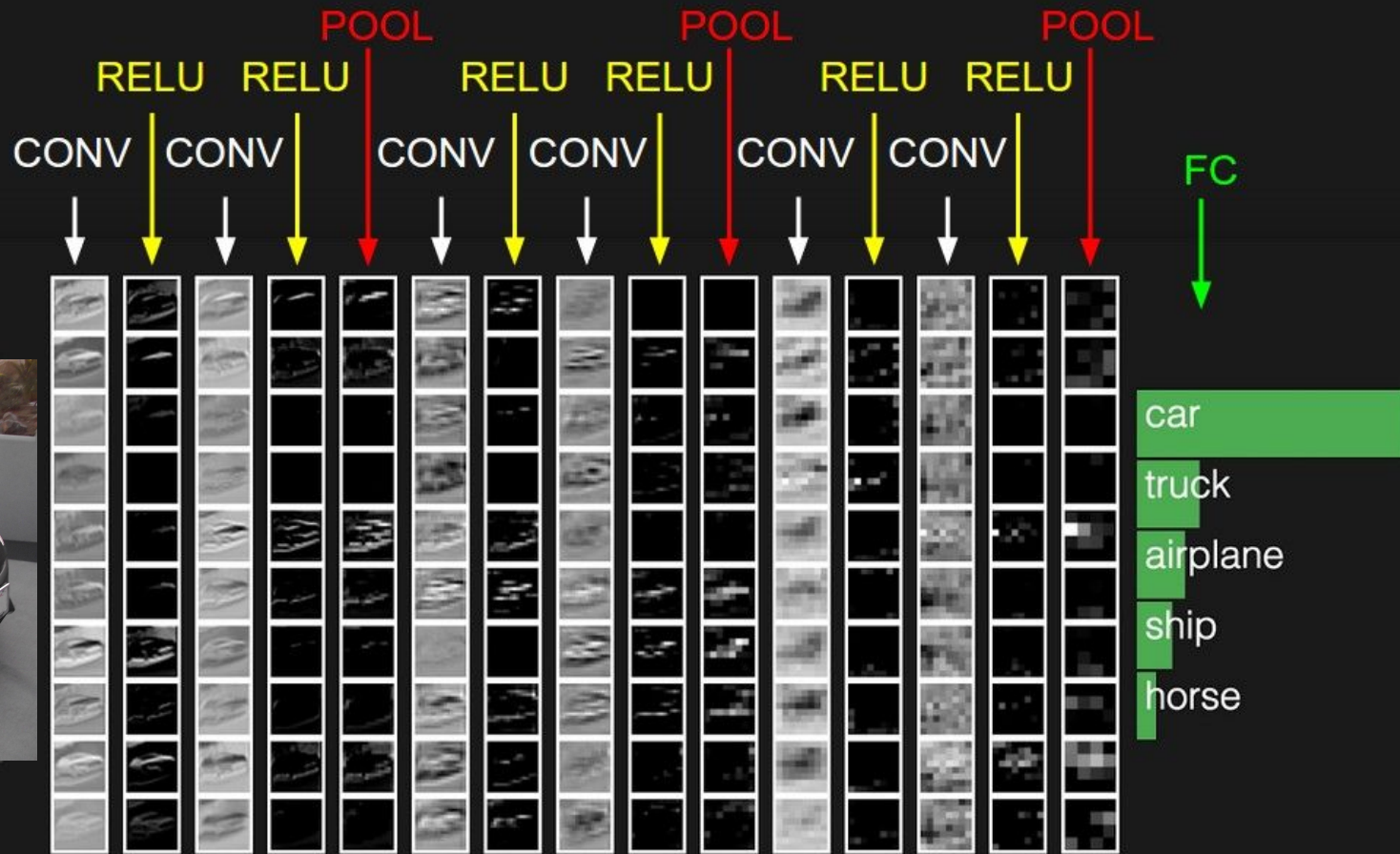
# Today's Agenda:

- **Attention with RNNs**
  - In Computer Vision
  - In NLP
- **General Attention Layer**
  - Self-attention
  - Positional encoding
  - Masked attention
  - Multi-head attention
- **Transformers**

# Today's Agenda:

- **Attention with RNNs**
  - In Computer Vision
  - In NLP
- **General Attention Layer**
  - Self-attention
  - Positional encoding
  - Masked attention
  - Multi-head attention
- **Transformers**

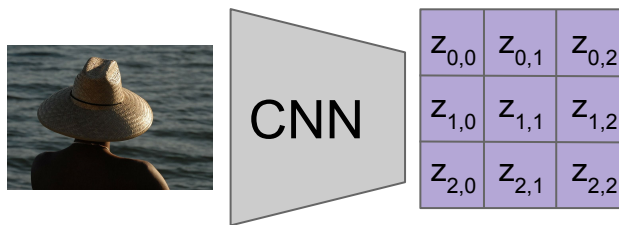
What the vision models we are currently look like:



# 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

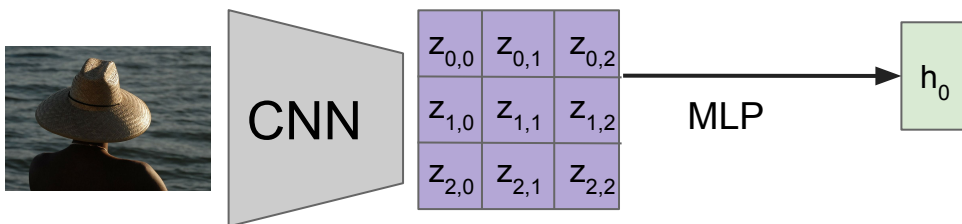
**Input:** Image  $I$

**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  $\mathbf{y} = y_1, y_2, \dots, y_T$

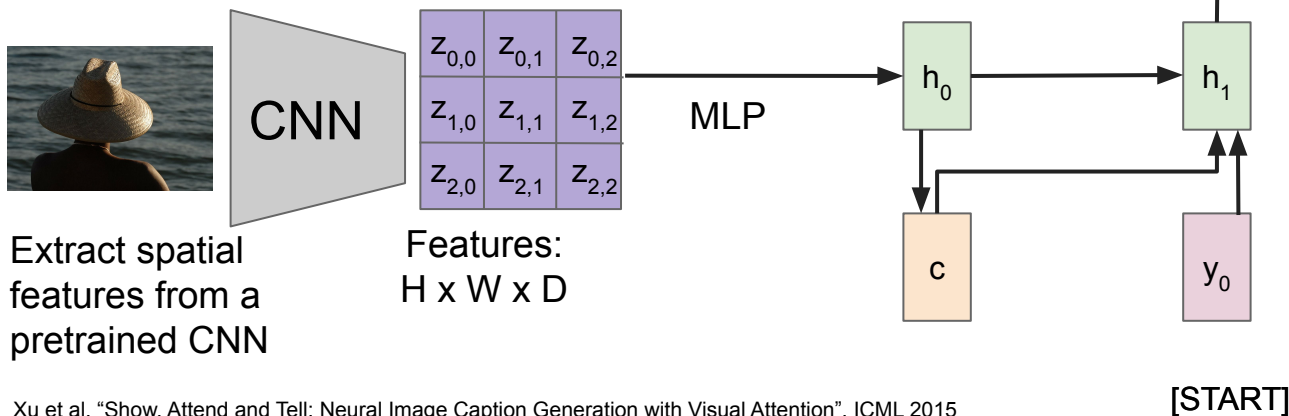
**Encoder:**  $h_0 = f_w(\mathbf{z})$

where  $\mathbf{z}$  is spatial CNN features

$f_w(\cdot)$  is an MLP

**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

**Input:** Image  $I$

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

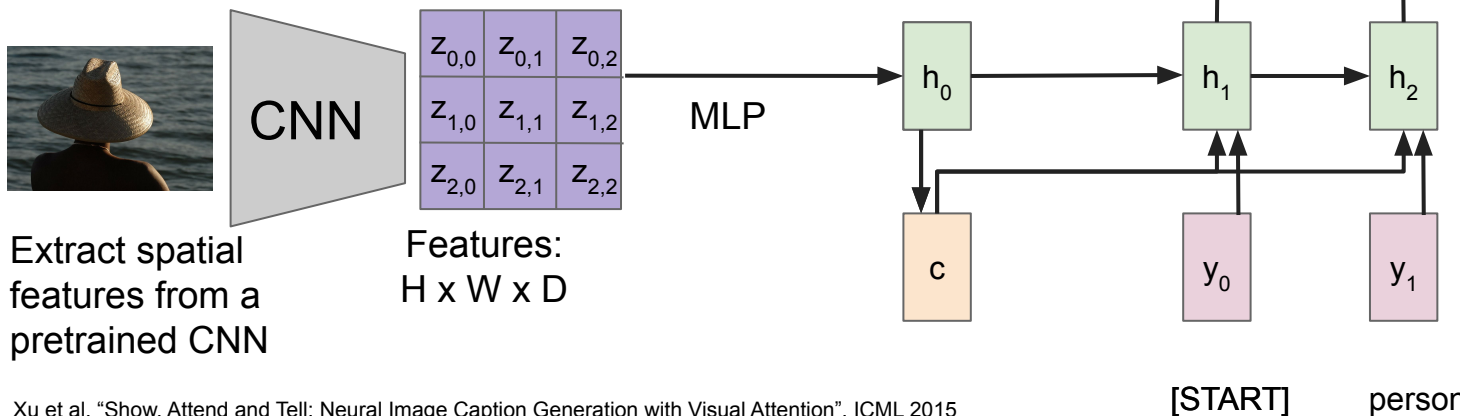
**Encoder:**  $h_0 = f_w(\mathbf{z})$

where  $\mathbf{z}$  is spatial CNN features

$f_w(\cdot)$  is an MLP

**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

**Input:** Image  $I$

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

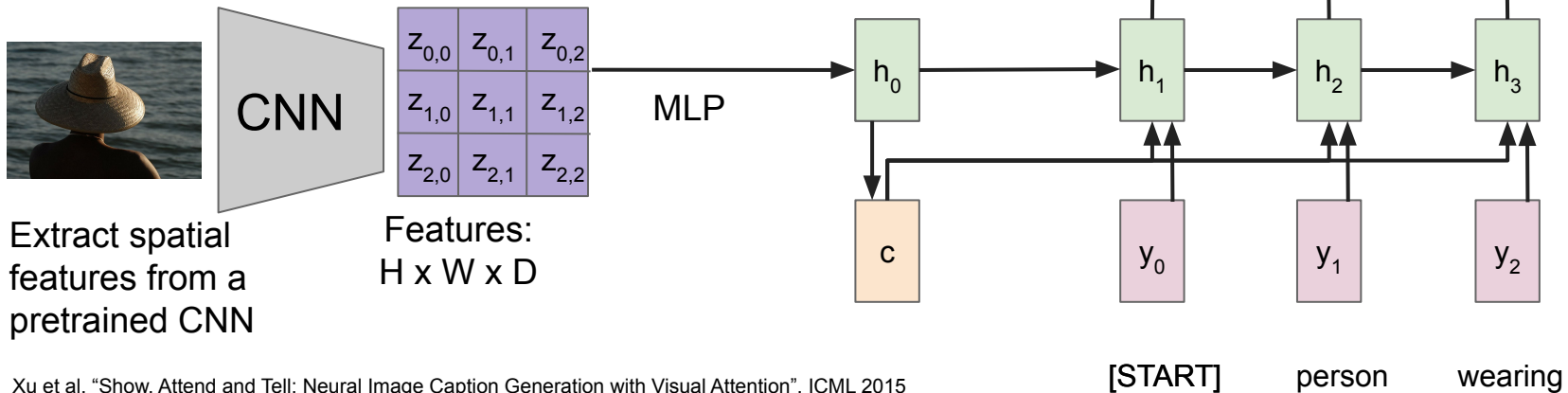
**Encoder:**  $h_0 = f_w(\mathbf{z})$

where  $\mathbf{z}$  is spatial CNN features

$f_w(\cdot)$  is an MLP

**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

**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

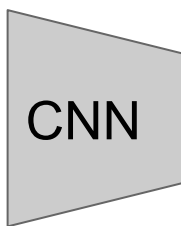
$f_w(\cdot)$  is an MLP

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

where context vector  $c$  is often  $c = h_0$



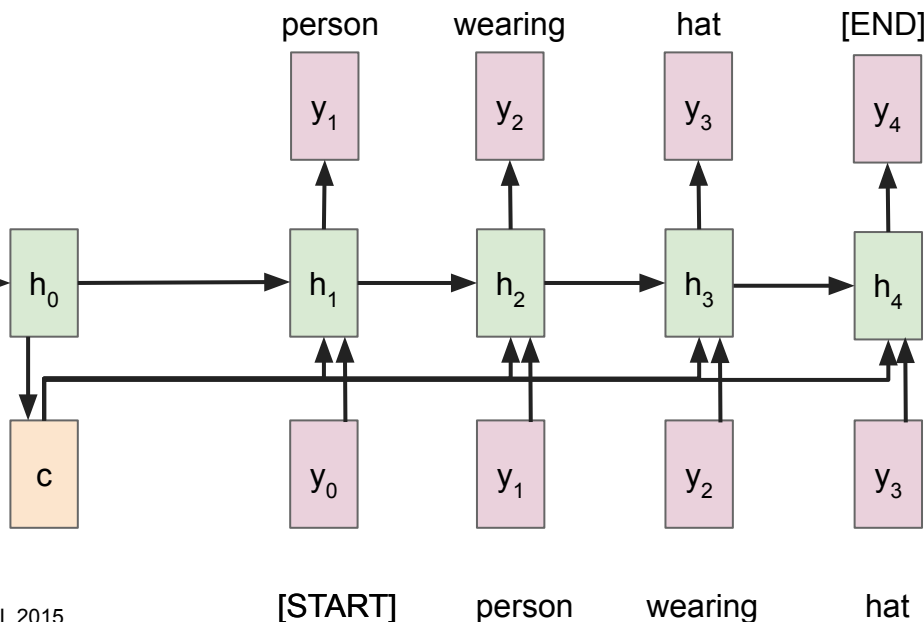
Extract spatial features from a pretrained CNN



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

Features:  
 $H \times W \times D$

MLP

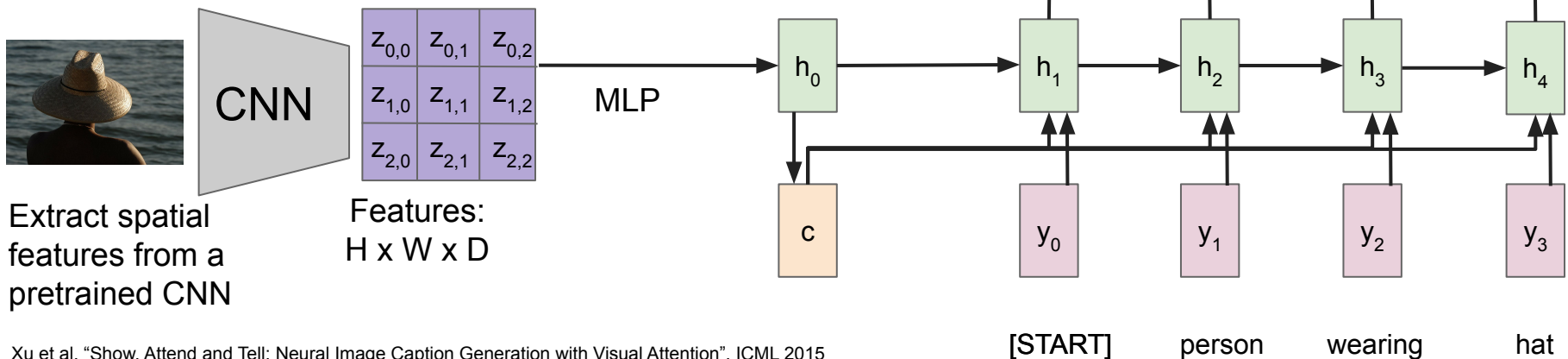


# Image Captioning using spatial features

**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**



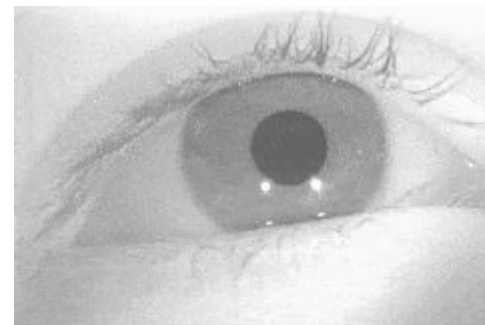
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Image Captioning with RNNs & Attention

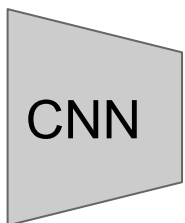
Attention idea: New context vector at every time step.

Each context vector will attend to different image regions

[gif source](#)

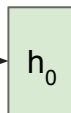


Attention Saccades in humans



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

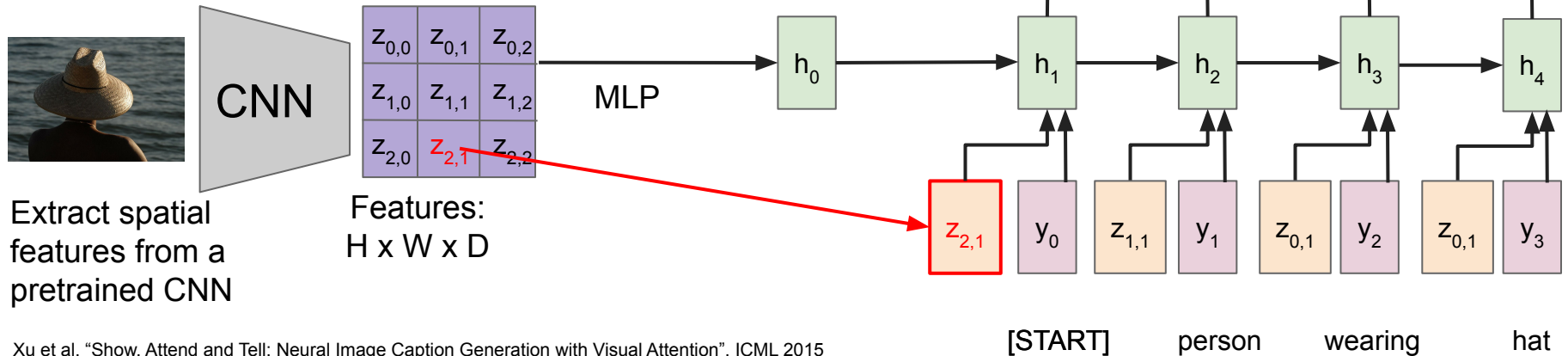
Features:  
 $H \times W \times D$



Extract spatial features from a pretrained CNN

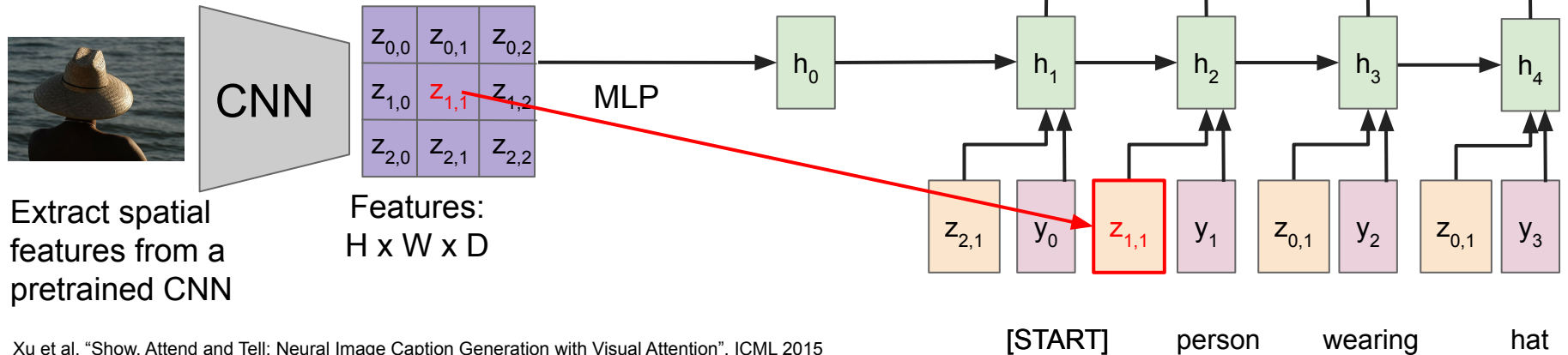
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Ideally what we want is for the model to look at different regions when generating each word



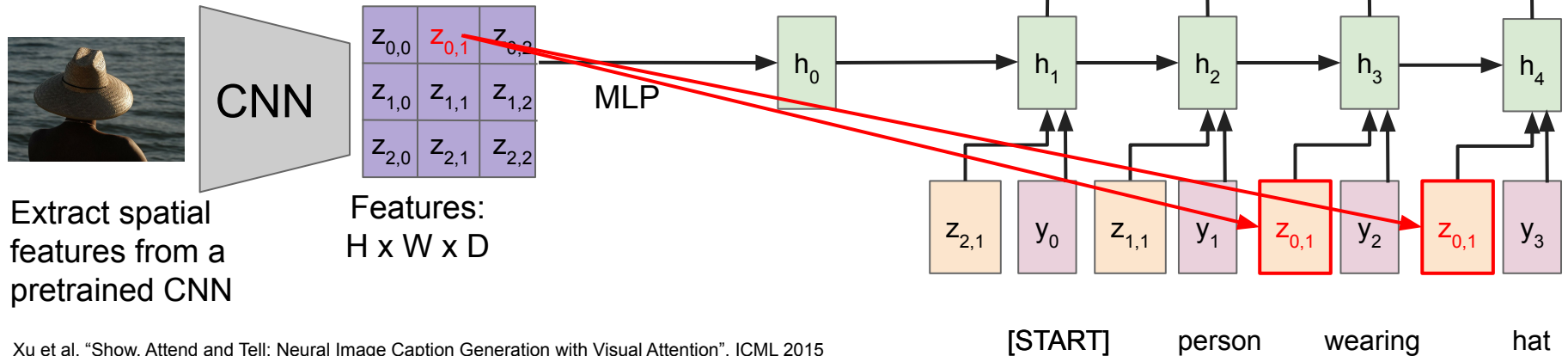
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Ideally what we want is for the model to look at different regions when generating each word



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

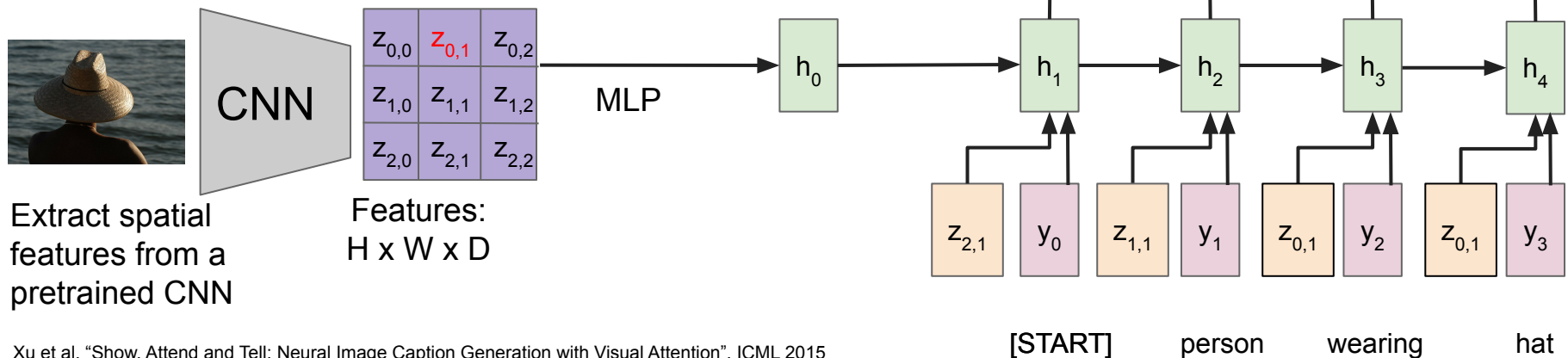
# Ideally what we want is for the model to look at different regions when generating each word



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

# How do we design a differentiable process that “**attends**” to different input regions?

Why differentiable? So that we can use backprop!



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

# Image Captioning with RNNs & 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

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



CNN

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 x W x D

$h_0$

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

# Image Captioning with RNNs & 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



CNN

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 x W x D

$h_0$

# Image Captioning with RNNs & 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:

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



CNN

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 x W x D

$h_0$



$c_1$

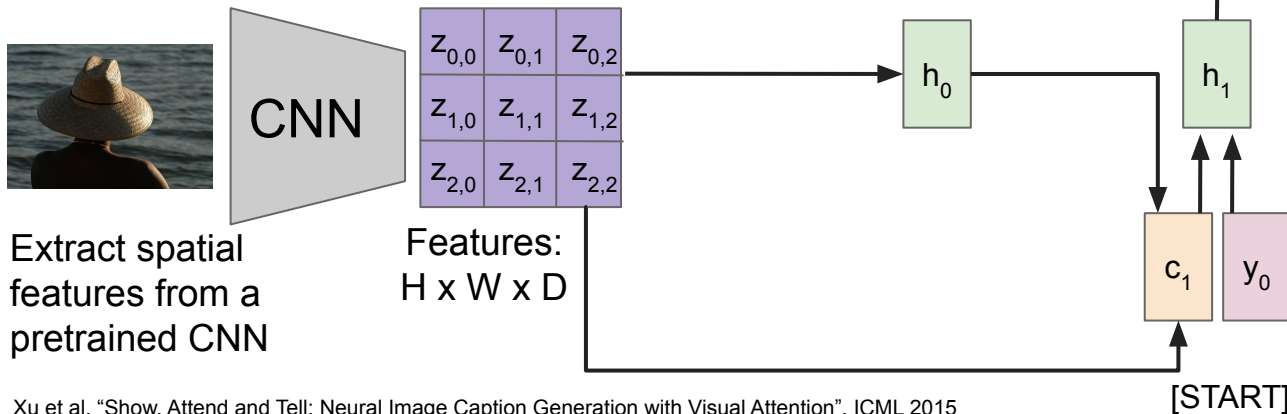
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Image Captioning with RNNs & Attention

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

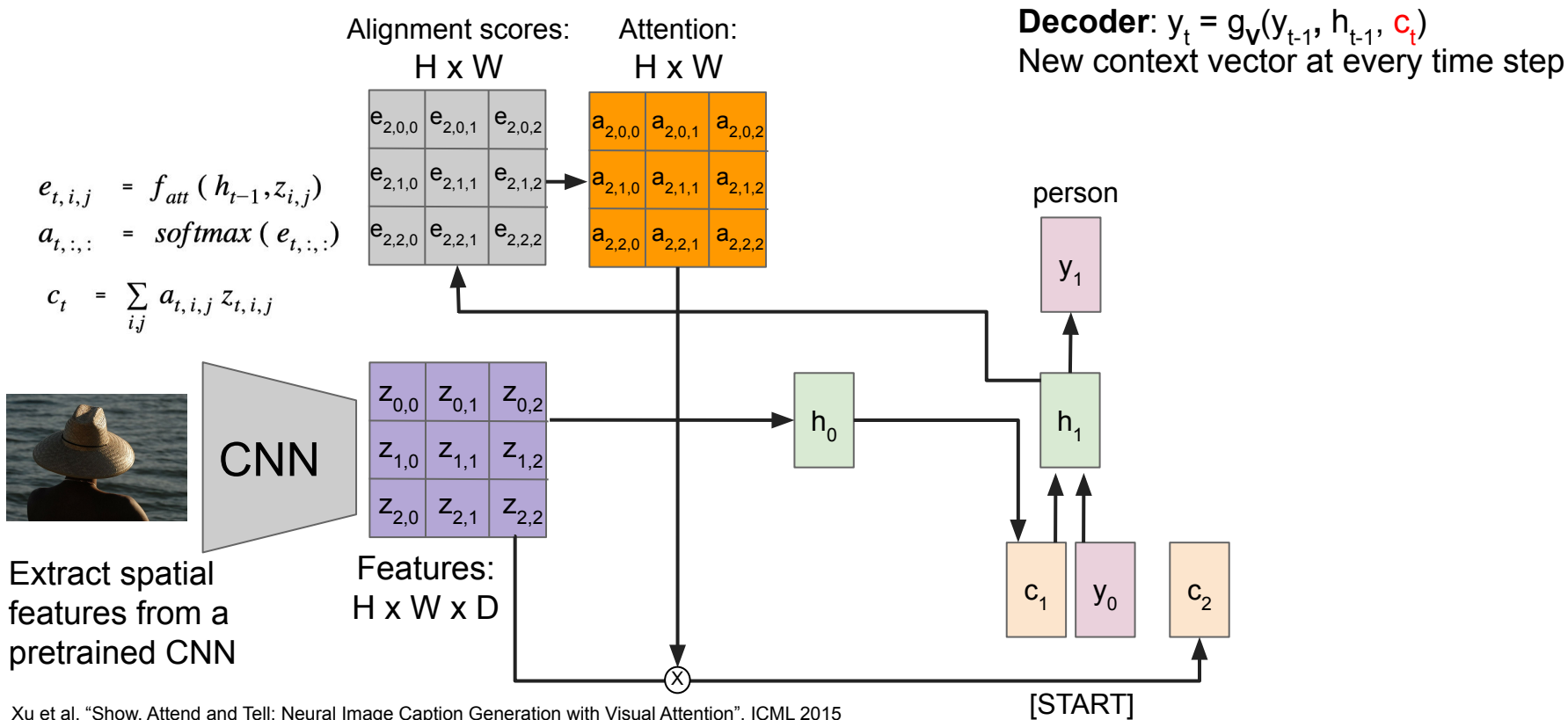
**Decoder:**  $y_t = g_v(y_{t-1}, h_{t-1}, c_t)$   
New context vector at every time step

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



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

# Image Captioning with RNNs & Attention



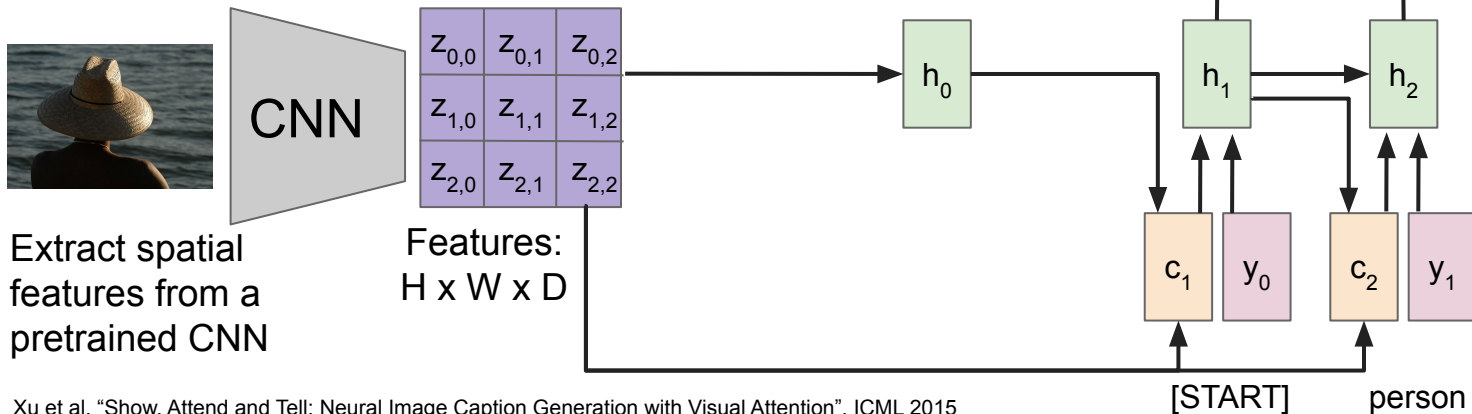
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Image Captioning with RNNs & Attention

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

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



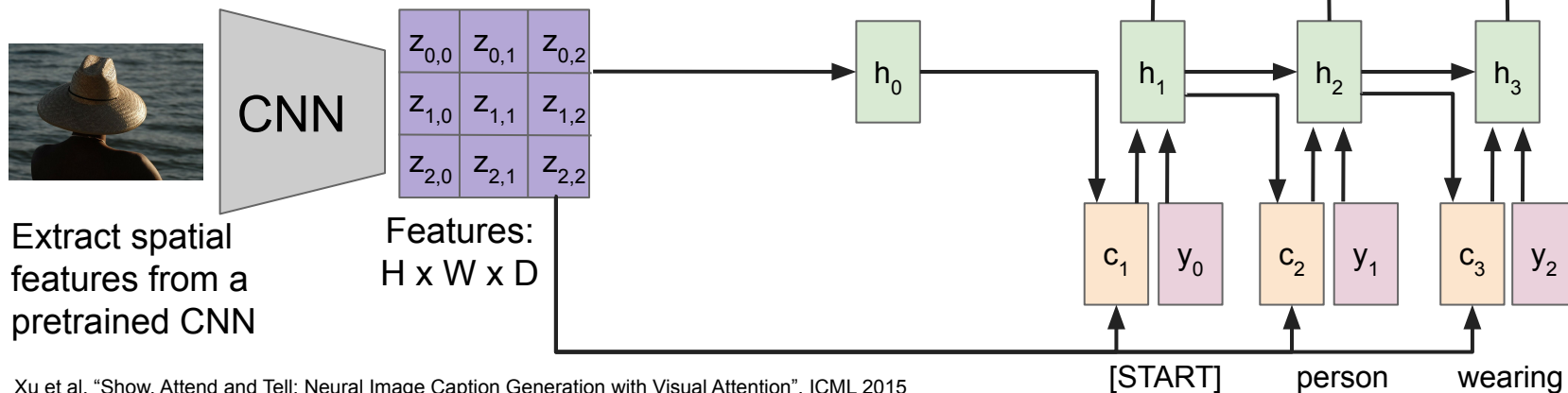
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Image Captioning with RNNs & Attention

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

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

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



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

# Image Captioning with RNNs & Attention

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

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



CNN

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 x W x D

$h_0$

person

wearing

hat

[END]

$y_1$

$y_2$

$y_3$

$y_4$

$h_1$

$h_2$

$h_3$

$h_4$

$c_1$

$y_0$

$c_2$

$y_1$

$c_3$

$y_2$

$c_4$

$y_3$

[START]

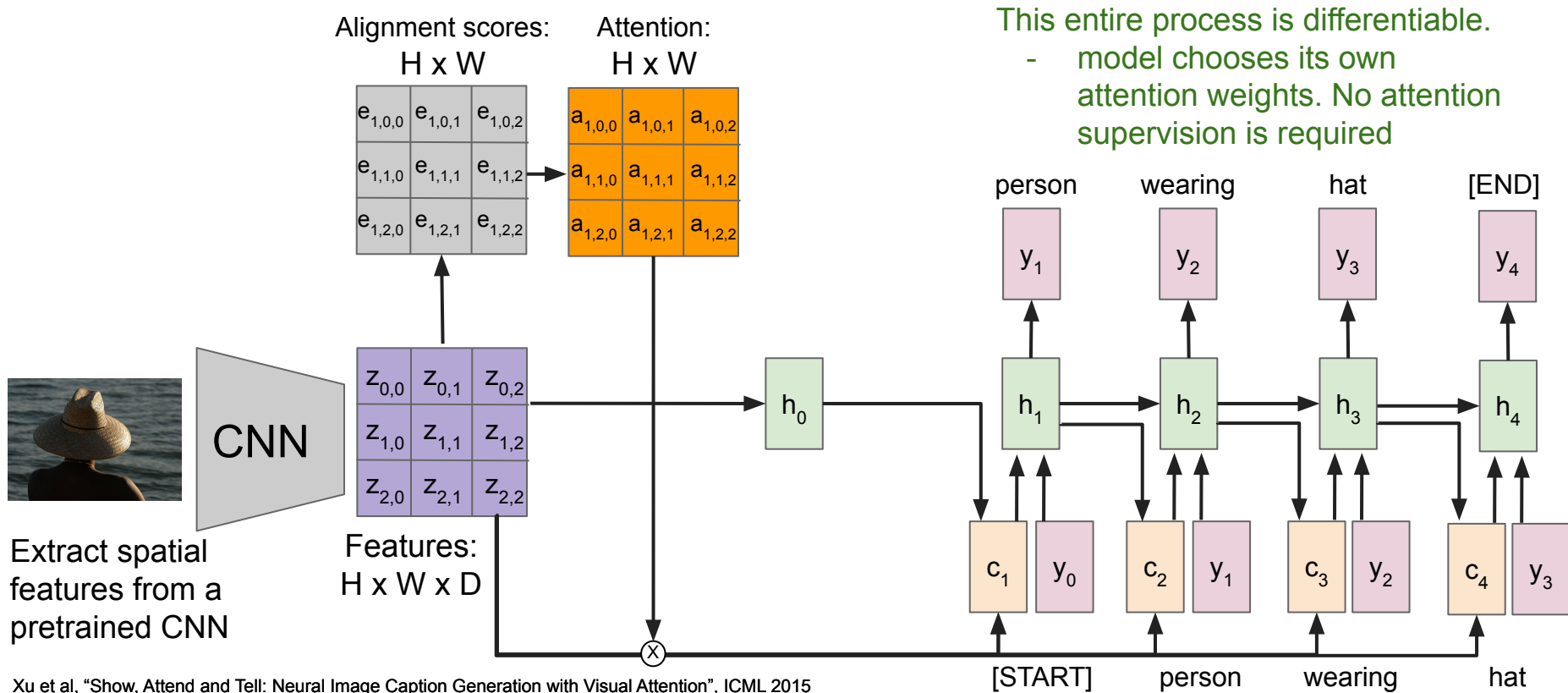
person

wearing

hat

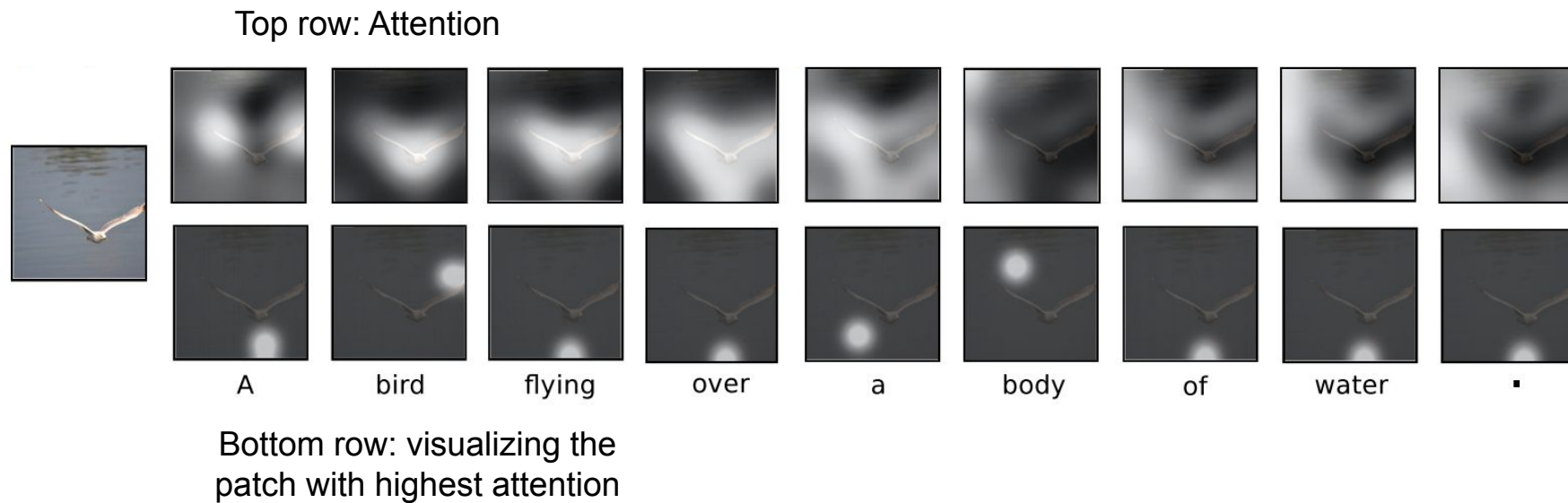
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Image Captioning with RNNs & Attention



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

# Image Captioning with Attention



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

Figure copyright Kelvin Xu, Jimmy Lei Ba, Jamie Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio, 2015. Reproduced with permission.

# Image Captioning with Attention



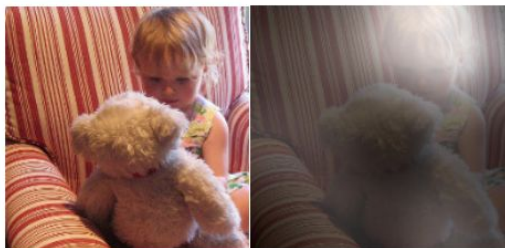
A woman is throwing a frisbee in a park.



A dog is standing on a hardwood floor.



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



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



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



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

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

Figure copyright Kelvin Xu, Jimmy Lei Ba, Jamie Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio, 2015. Reproduced with permission.

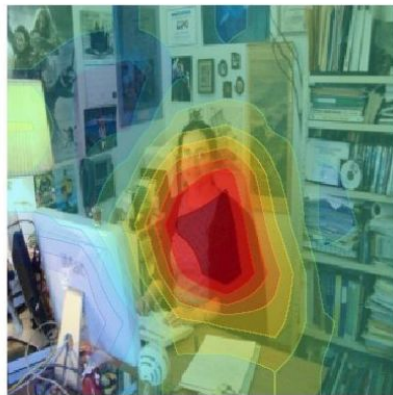
# Attention can detect Gender Bias

Wrong



Baseline:  
*A **man** sitting at a desk with  
a laptop computer.*

Right for the Right  
Reasons



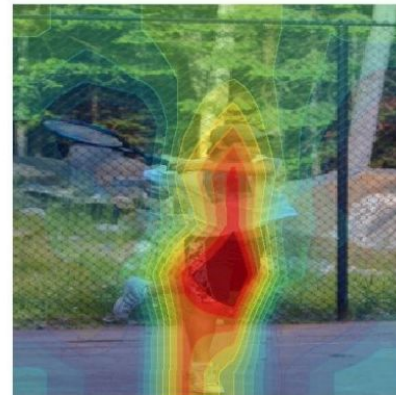
Our Model:  
*A **woman** sitting in front of a  
laptop computer.*

Right for the Wrong  
Reasons



Baseline:  
*A **man** holding a tennis  
racquet on a tennis court.*

Right for the Right  
Reasons

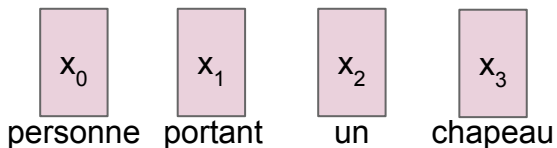


Our Model:  
*A **man** holding a tennis  
racquet on a tennis court.*

# Similar tasks in NLP - Language translation example

**Input:** Sequence  $\mathbf{x} = x_1, x_2, \dots, x_T$

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



# Similar tasks in NLP - Language translation example

**Input:** Sequence  $\mathbf{x} = x_1, x_2, \dots, x_T$

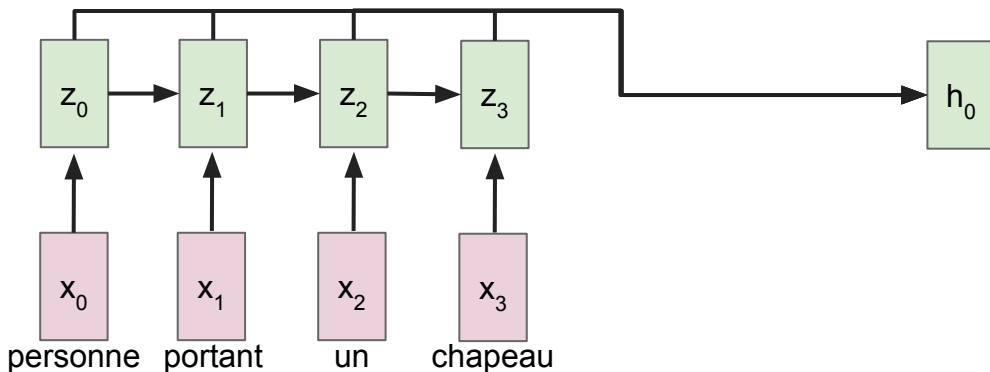
**Output:** Sequence  $\mathbf{y} = y_1, y_2, \dots, y_T$

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

where  $z_t = \text{RNN}(x_t, u_{t-1})$

$f_w(\cdot)$  is MLP

$u$  is the hidden RNN state



# Similar tasks in NLP - Language translation example

**Input:** Sequence  $\mathbf{x} = x_1, x_2, \dots, x_T$

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

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

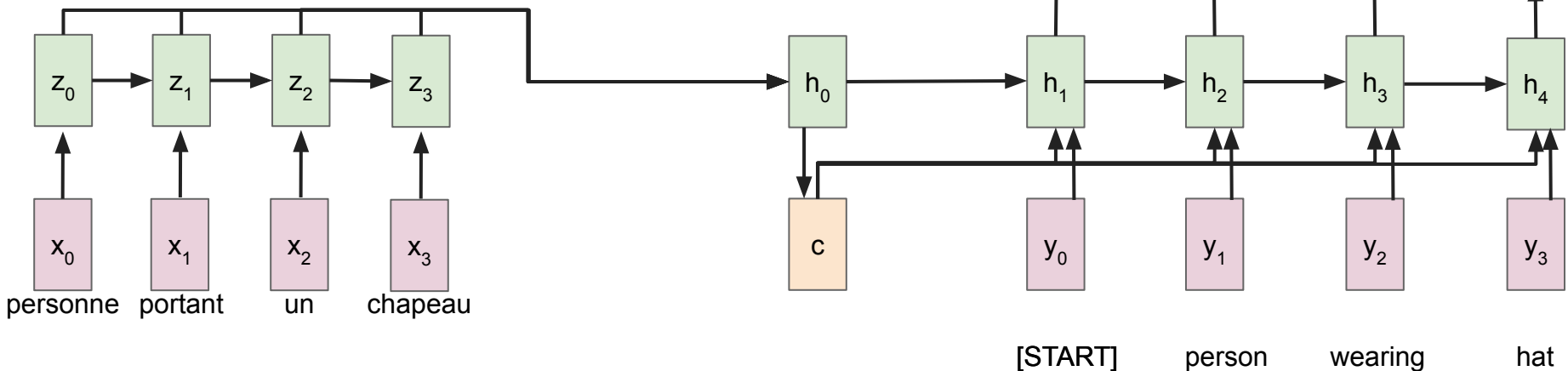
where context vector  $c$  is often  $c = h_0$

**Encoder:**  $h_0 = f_w(z)$

where  $z_t = \text{RNN}(x_t, u_{t-1})$

$f_w(\cdot)$  is MLP

$u$  is the hidden RNN state

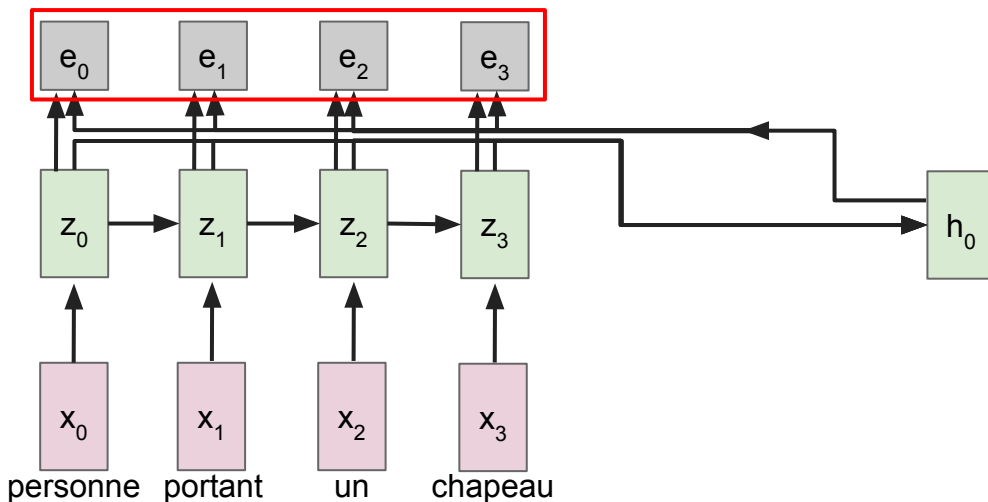


# Attention in NLP - Language translation example

Compute alignments  
scores (scalars):

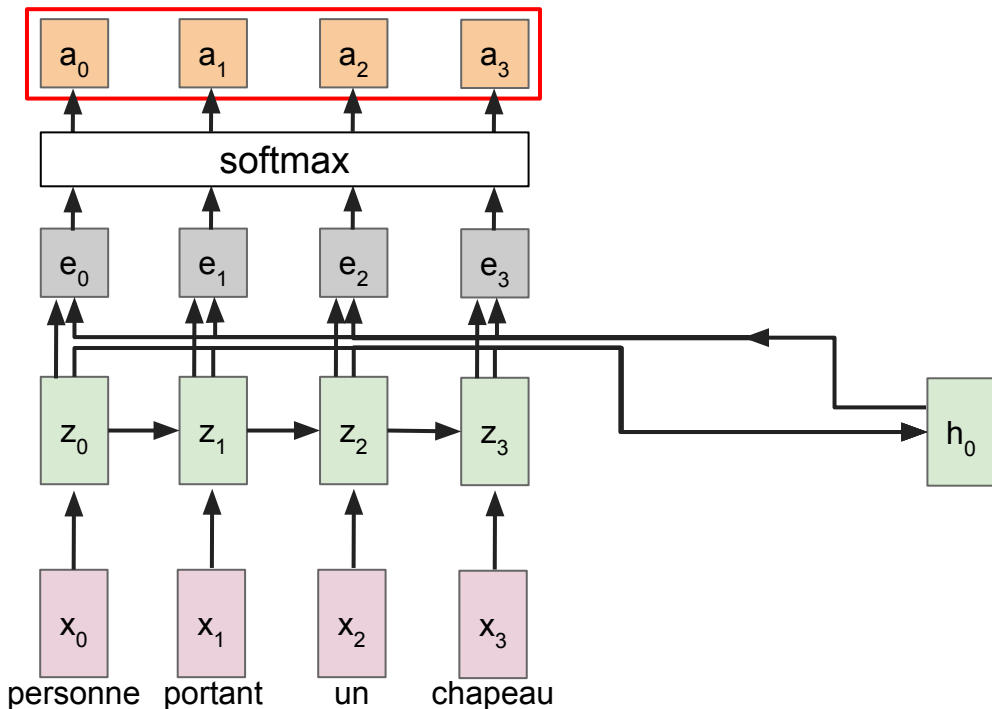
$$e_{t,i} = f_{att}(h_{t-1}, z_i)$$

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



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

# Attention in NLP - Language translation example



Compute alignments scores (scalars):

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

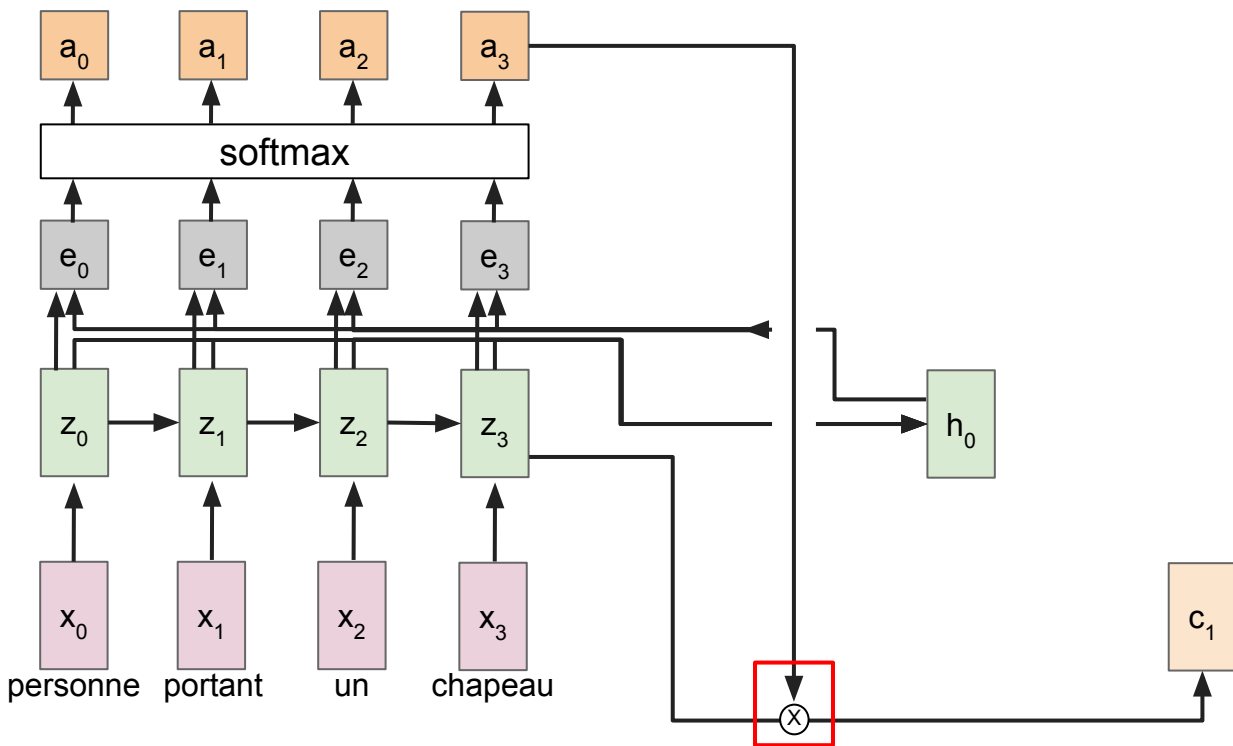
$f_{att}(\cdot)$  is an MLP

Normalize to get attention weights:

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

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

# Attention in NLP - Language translation example



Compute alignments scores (scalars):

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

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

Normalize to get attention weights:

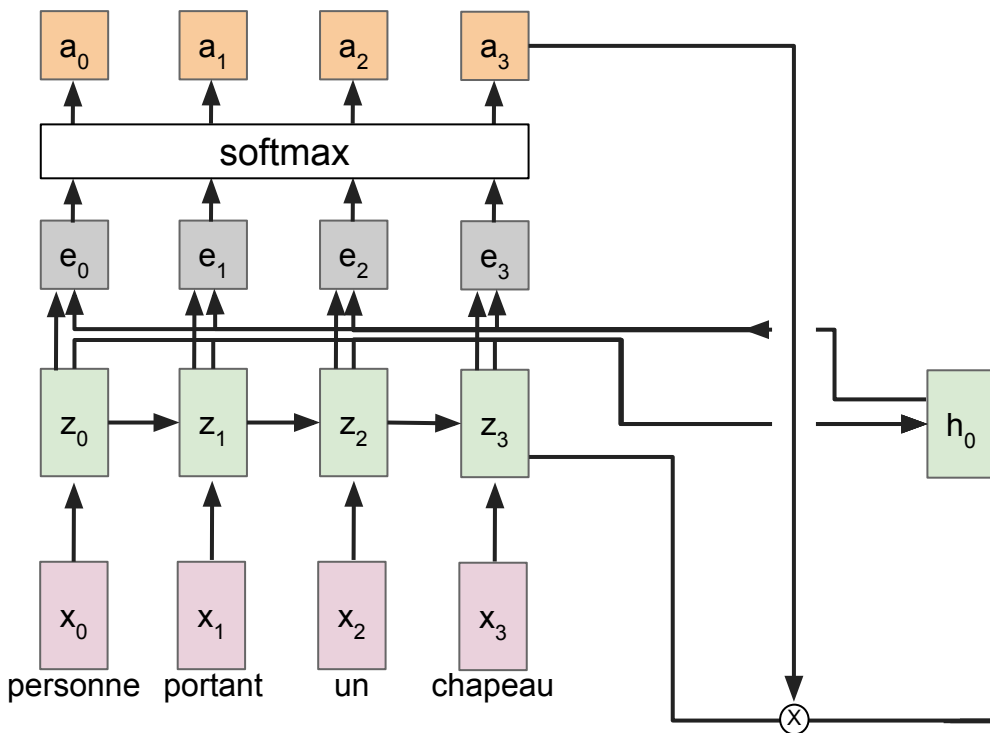
$$a_{t,:} = \text{softmax}(e_{t,:})$$

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

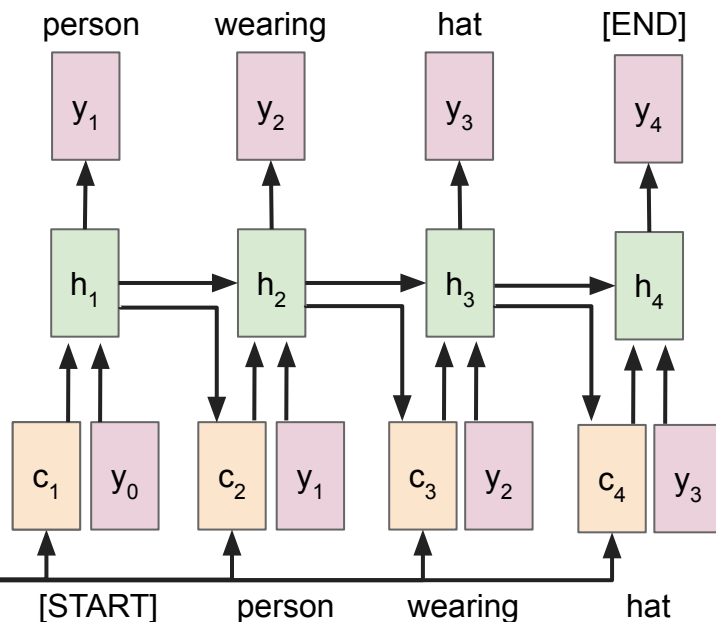
Compute context vector:

$$c_t = \sum_i a_{t,i} z_{t,i}$$

# Attention in NLP - Language translation example



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



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

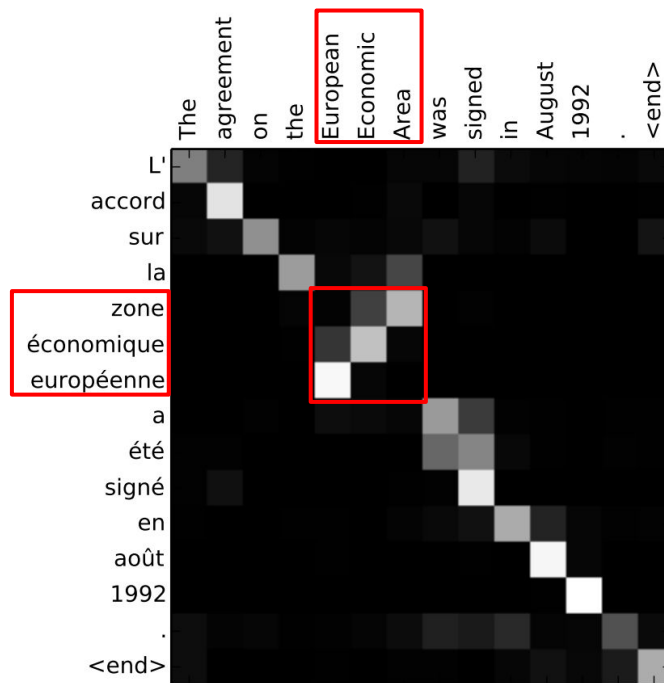
# Similar visualization of attention weights

English to French translation example:

**Input:** "The agreement on the **European Economic Area** was signed in August 1992."

**Output:** "L'accord sur la **zone économique européenne** a été signé en août 1992."

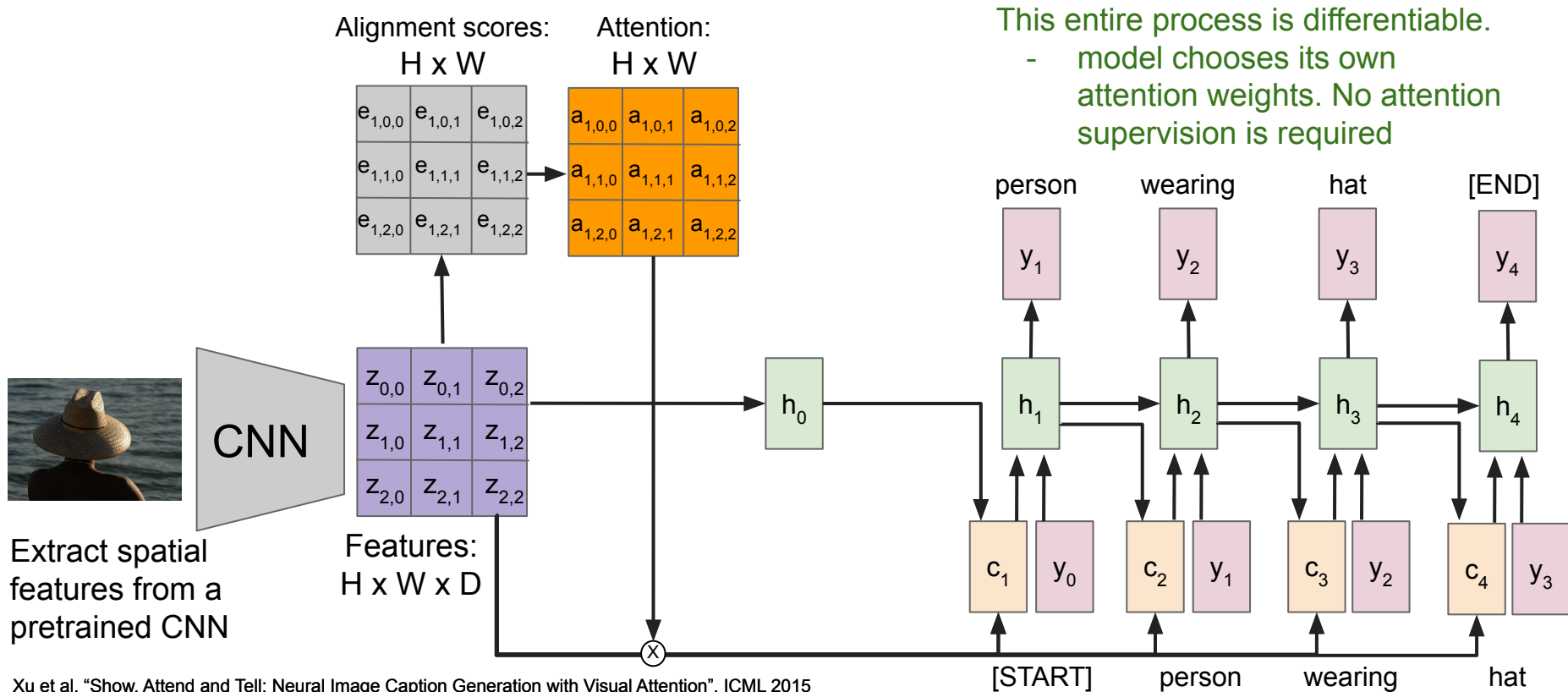
Without any attention supervision, model learns different word orderings for different languages



# Today's Agenda:

- **Attention with RNNs**
  - In Computer Vision
  - In NLP
- **General Attention Layer**
  - Self-attention
  - Positional encoding
  - Masked attention
  - Multi-head attention
- **Transformers**

# Image Captioning with RNNs & Attention



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

# 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:

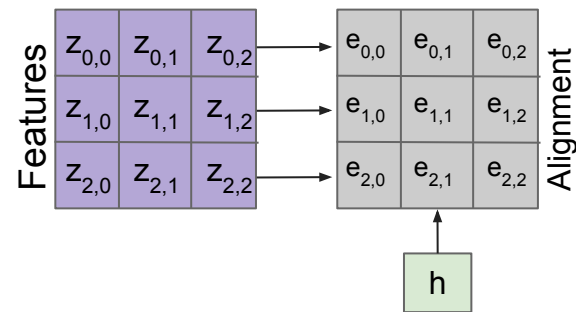
Features:  $\mathbf{z}$  (shape:  $H \times W \times D$ )

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

# Attention we just saw in image captioning

## Operations:

Alignment:  $e_{i,j} = f_{\text{att}}(h, z_{i,j})$

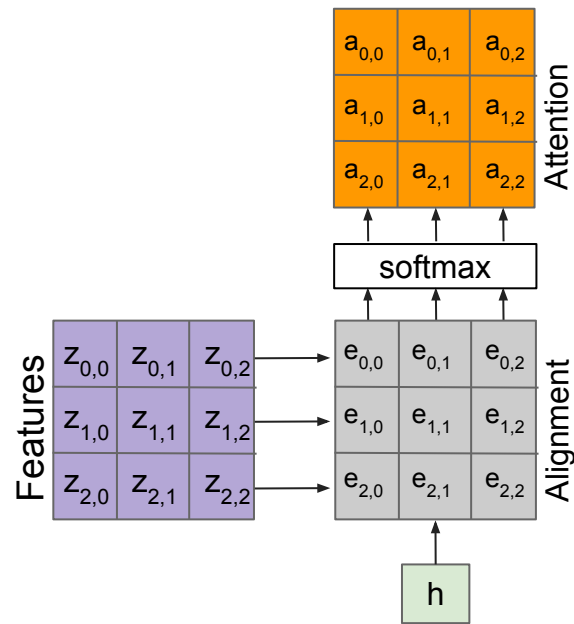


## Inputs:

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

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

# Attention we just saw in image captioning



## Operations:

Alignment:  $e_{i,j} = f_{\text{att}}(h, z_{i,j})$

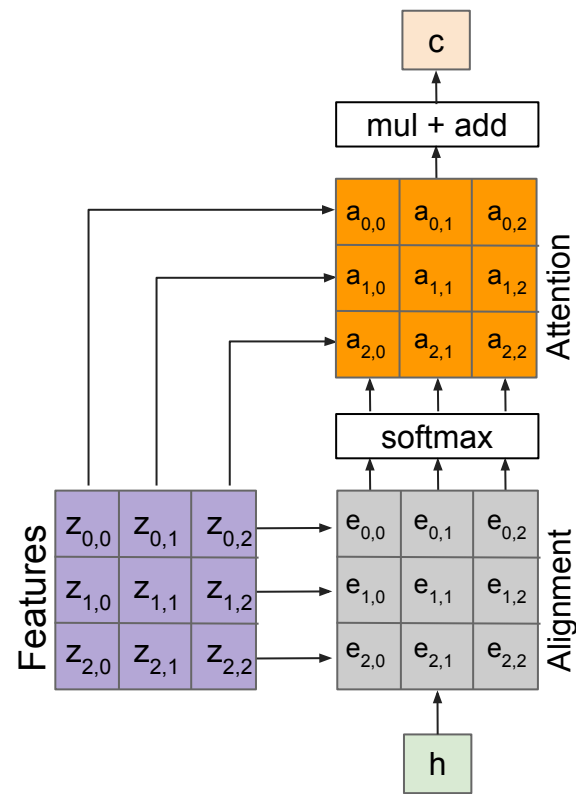
Attention:  $\mathbf{a} = \text{softmax}(\mathbf{e})$

## Inputs:

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

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

# Attention we just saw in image captioning



## Outputs:

context vector:  $c$  (shape: D)

## Operations:

Alignment:  $e_{i,j} = f_{\text{att}}(h, z_{i,j})$

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

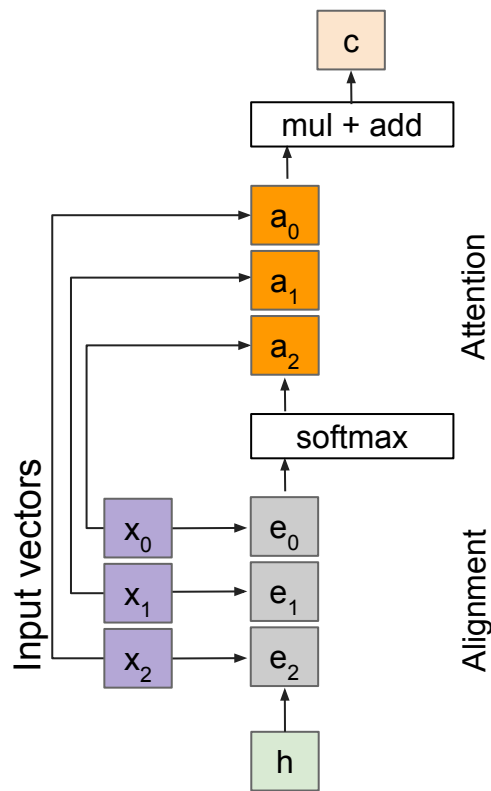
Output:  $c = \sum_{i,j} a_{i,j} z_{i,j}$

## Inputs:

Features:  $z$  (shape: H x W x D)

Query:  $h$  (shape: D)

# General attention layer



## Outputs:

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

## Operations:

Alignment:  $e_i = f_{\text{att}}(h, x_i)$

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

Output:  $\mathbf{c} = \sum_i \mathbf{a}_i x_i$

## Inputs:

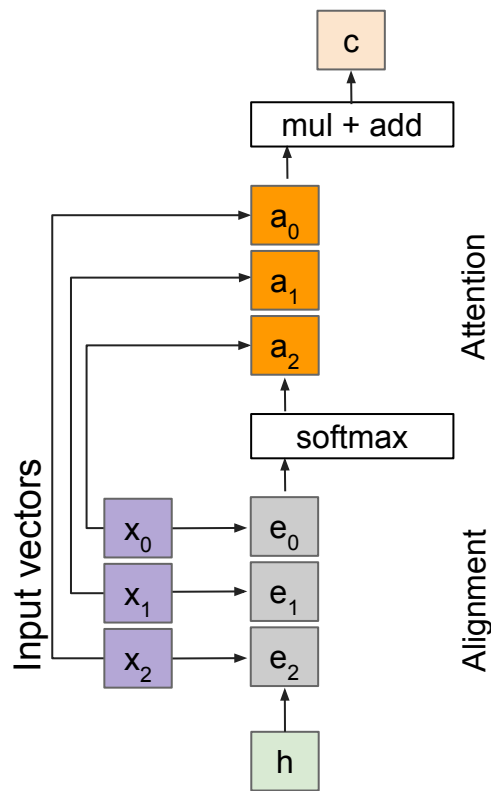
Input vectors: **x** (shape: N x D)

Query: **h** (shape: D)

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:  $\mathbf{c}$  (shape:  $D$ )

## Operations:

Alignment:  $\mathbf{e}_i = \mathbf{h} \cdot \mathbf{x}_i$

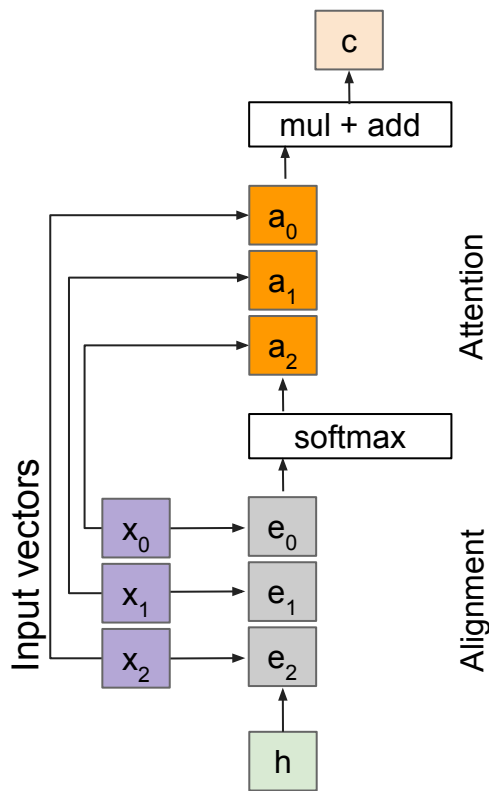
Attention:  $\mathbf{a} = \text{softmax}(\mathbf{e})$

Output:  $\mathbf{c} = \sum_i \mathbf{a}_i \mathbf{x}_i$

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

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

# General attention layer



## Outputs:

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

## Operations:

Alignment:  $e_i = h \cdot x_i / \sqrt{D}$

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

Output:  $\mathbf{c} = \sum_i a_i x_i$

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

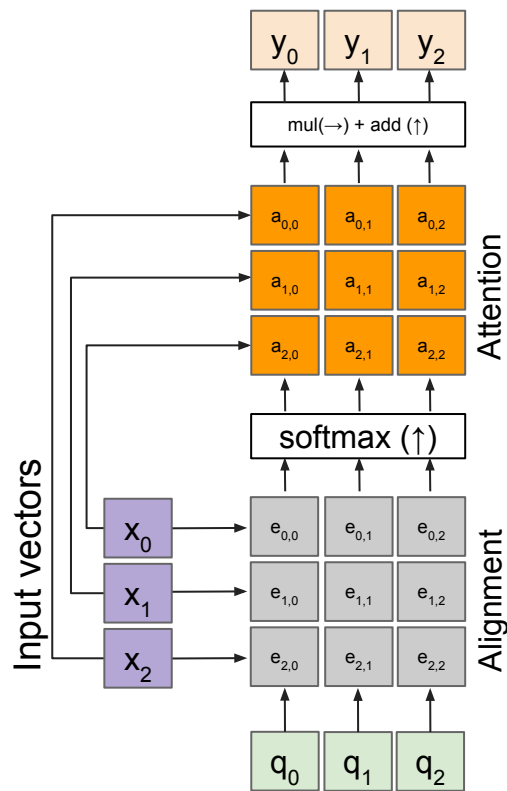
- High dimensionality means more terms in the dot product sum.
- Large magnitude vectors will cause softmax to peak and assign very little weight to all others
- Dividing by  $\sqrt{D}$  will reduce effect of large magnitude vectors

## Inputs:

Input vectors: **x** (shape: N x D)

Query: **h** (shape: D)

# General attention layer



## Outputs:

context vectors:  $\mathbf{y}$  (shape:  $M \times D$ )

Multiple query vectors

- each query creates a new output context vector

## Operations:

Alignment:  $e_{i,j} = q_j \cdot x_i / \sqrt{D}$

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

Output:  $y_j = \sum_i a_{i,j} x_i$

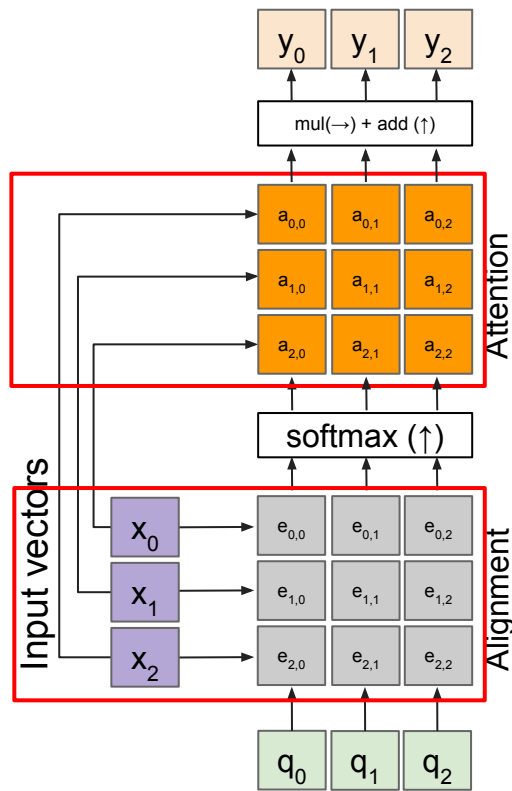
## Inputs:

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

Queries:  $\mathbf{q}$  (shape:  $M \times D$ )

Multiple query vectors

# General attention layer



## Outputs:

context vectors:  $\mathbf{y}$  (shape:  $M \times D$ )

## Operations:

Alignment:  $e_{i,j} = q_j \cdot x_i / \sqrt{D}$

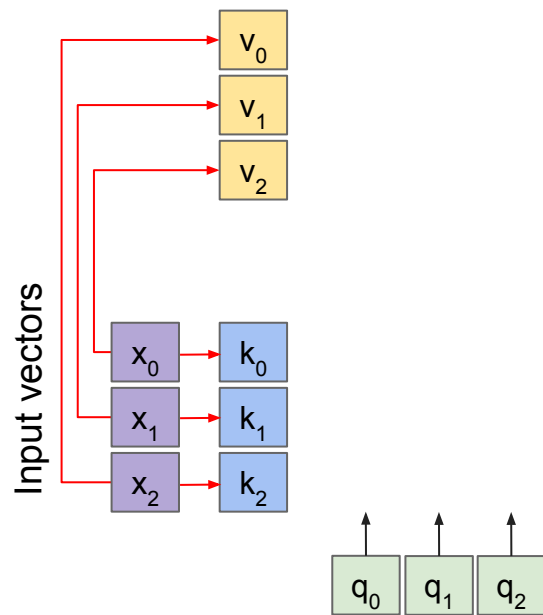
Attention:  $\mathbf{a} = \text{softmax}(\mathbf{e})$

Output:  $y_j = \sum_i a_{i,j} x_i$

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



## Operations:

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

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

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

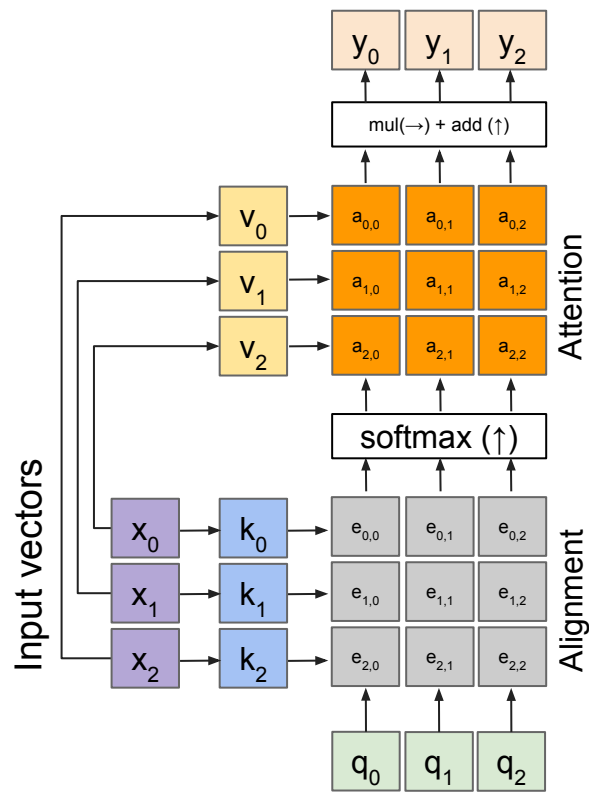
- We can add more expressivity to the layer by adding a different FC layer before each of the two steps.

## Inputs:

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

Queries:  $\mathbf{q}$  (shape:  $M \times D_k$ )

# General attention layer



## Outputs:

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

## Operations:

Key vectors:  $\mathbf{k} = \mathbf{x}\mathbf{W}_k$   
Value vectors:  $\mathbf{v} = \mathbf{x}\mathbf{W}_v$   
Alignment:  $e_{i,j} = q_i \cdot k_j / \sqrt{D}$   
Attention:  $\mathbf{a} = \text{softmax}(\mathbf{e})$   
Output:  $y_j = \sum_i a_{i,j} v_i$

## Inputs:

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

Queries:  $\mathbf{q}$  (shape:  $M \times D_k$ )

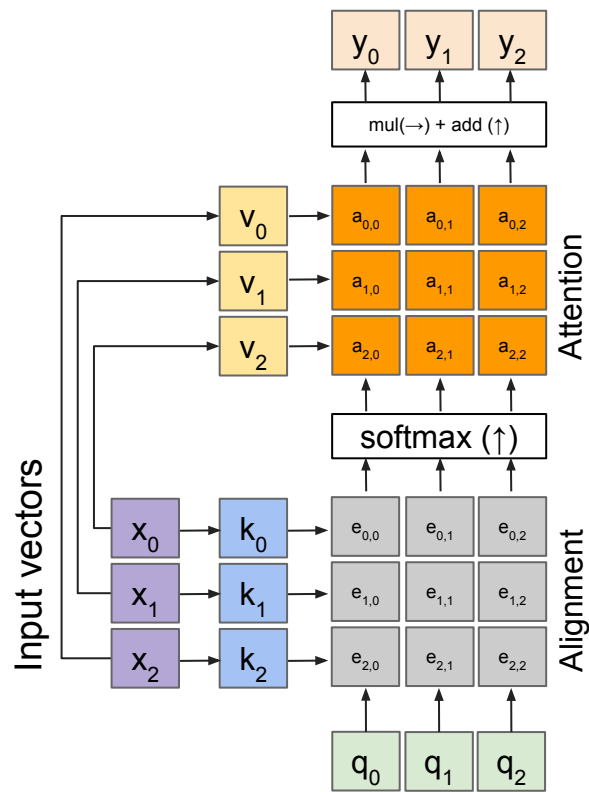
The input and output dimensions can now change depending on the key and value FC layers

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.

Deriving self-attention from this general attention layer

# General attention layer



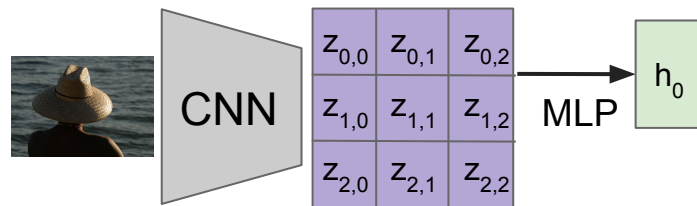
**Outputs:**  
context vectors:  $\mathbf{y}$  (shape:  $D_v$ )

**Operations:**  
Key vectors:  $\mathbf{k} = \mathbf{x}\mathbf{W}_k$   
Value vectors:  $\mathbf{v} = \mathbf{x}\mathbf{W}_v$   
Alignment:  $e_{i,j} = q_i \cdot k_j / \sqrt{D}$   
Attention:  $\mathbf{a} = \text{softmax}(\mathbf{e})$   
Output:  $y_j = \sum_i a_{i,j} v_i$

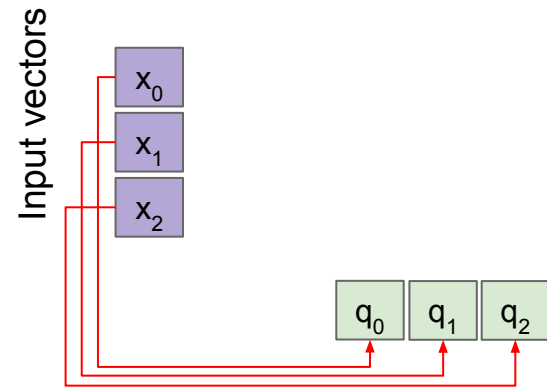
**Inputs:**  
Input vectors:  $\mathbf{x}$  (shape:  $N \times D$ )  
Queries:  $\mathbf{q}$  (shape:  $M \times D_k$ )

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

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

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

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

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

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.

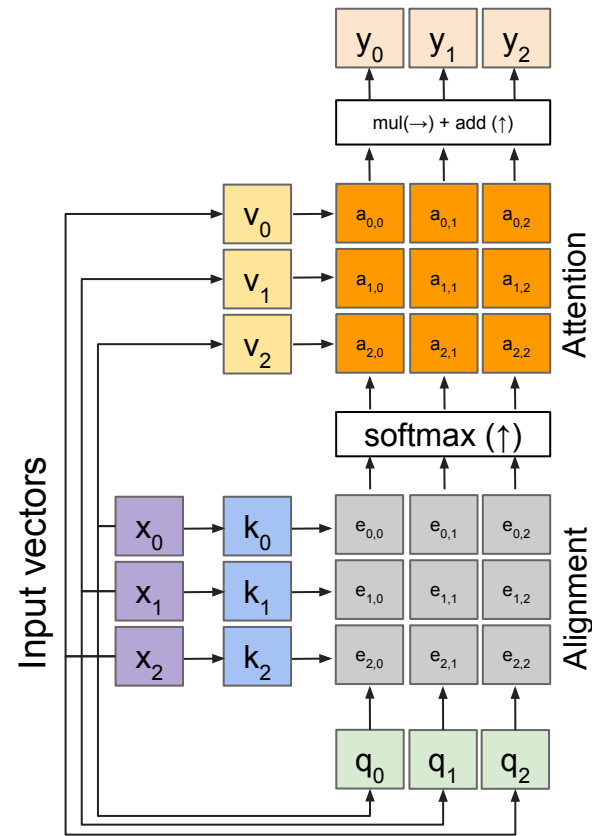
No input query vectors anymore

## Inputs:

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

Queries:  $\mathbf{q}$  (shape:  $M \times D_k$ )

# 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:  $e_{i,j} = \mathbf{q}_i \cdot \mathbf{k}_j / \sqrt{D}$

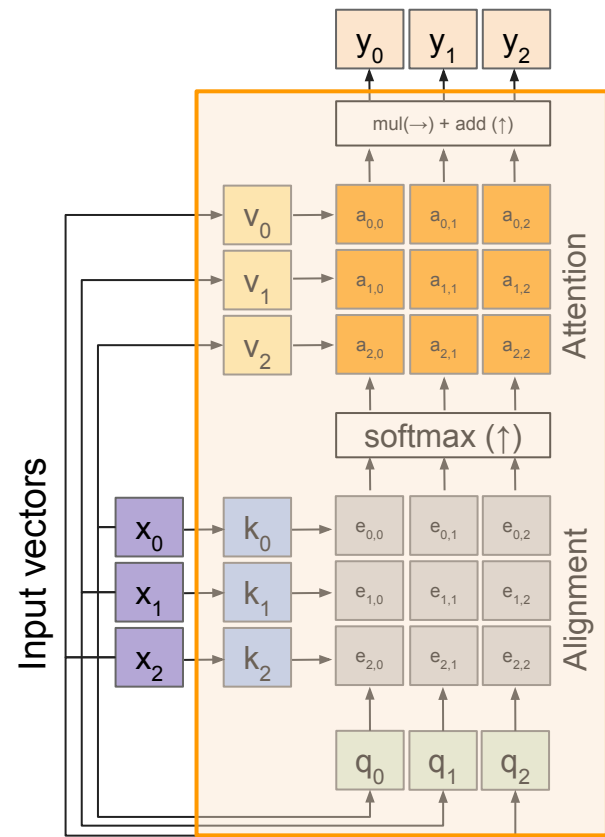
Attention:  $\mathbf{a} = \text{softmax}(\mathbf{e})$

Output:  $y_j = \sum_i a_{i,j} v_i$

## Inputs:

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

# Self attention layer - attends over sets of inputs



## 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:  $\mathbf{q} = \mathbf{x}\mathbf{W}_q$

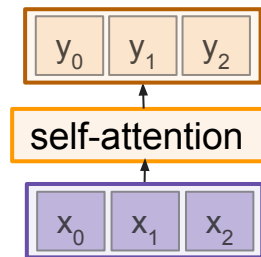
Alignment:  $e_{i,j} = \mathbf{q}_i \cdot \mathbf{k}_j / \sqrt{D_k}$

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

Output:  $y_j = \sum_i a_{i,j} v_i$

## Inputs:

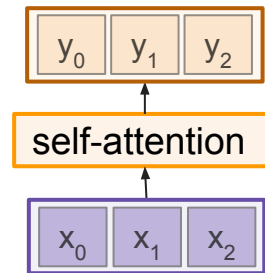
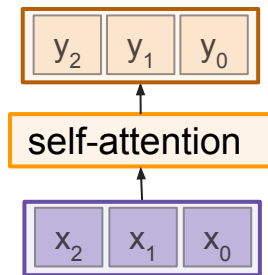
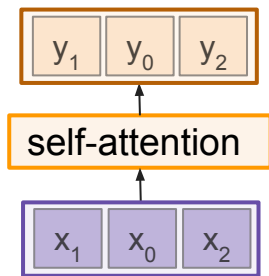
Input vectors:  $\mathbf{x}$  (shape:  $N \times D$ )



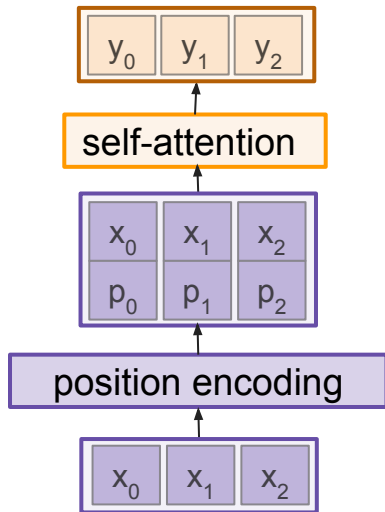
# Self attention layer - attends over sets of inputs

Permutation invariant

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



# Positional encoding



Concatenate special positional encoding  $p_j$  to each input vector  $x_j$

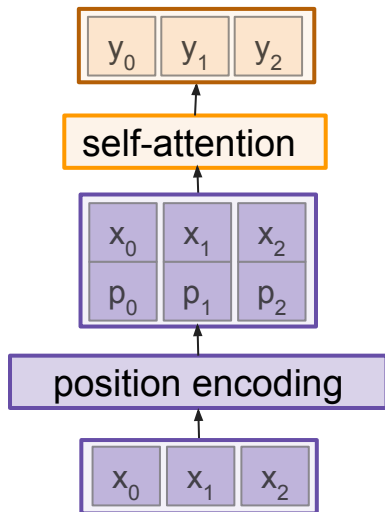
We use a function  $pos: \mathbb{N} \rightarrow \mathbb{R}^d$  to process the position  $j$  of the vector into a  $d$ -dimensional vector

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

So,  $p_j = pos(j)$

Options for  $pos(\cdot)$

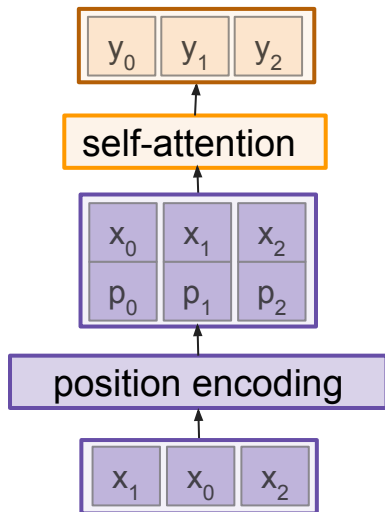
1. Learn a lookup table:
  - Learn parameters to use for  $pos(t)$  for  $t \in [0, T)$
  - Lookup table contains  $T \times d$  parameters.

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

So,  $p_j = pos(j)$

Options for  $pos(\cdot)$

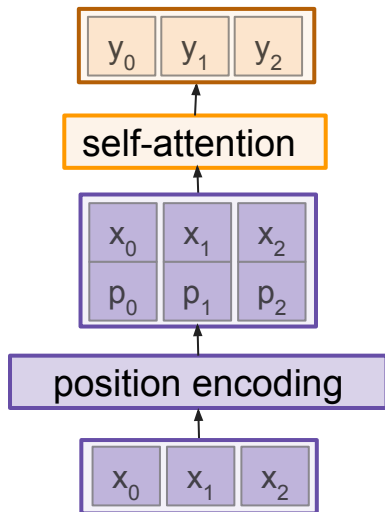
1. Learn a lookup table:
  - Learn parameters to use for  $pos(t)$  for  $t \in [0, T]$
  - Lookup table contains  $T \times d$  parameters.
2. Design a fixed function with the desiderata
  -

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

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

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  $pos: \mathbb{N} \rightarrow \mathbb{R}^d$  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:
  - Learn parameters to use for  $pos(t)$  for  $t \in [0, T)$
  - 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:

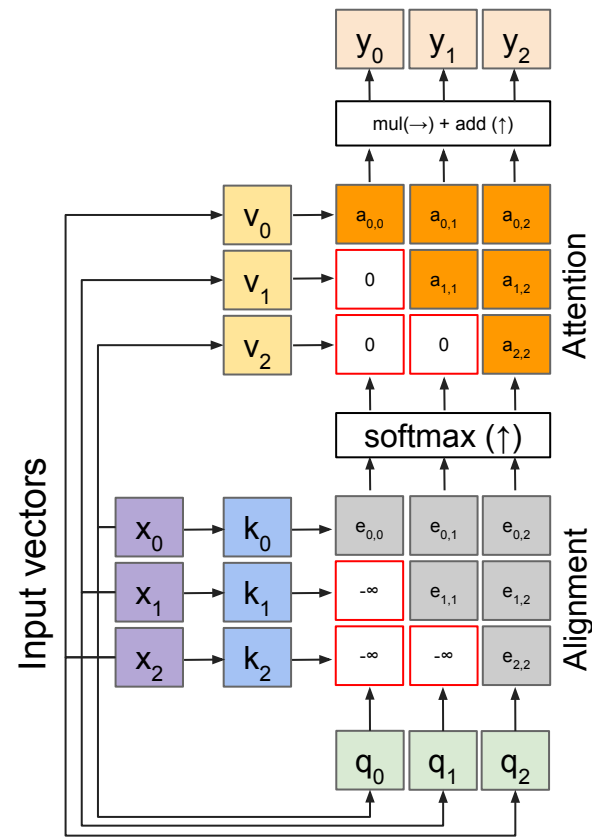
|      |   |   |   |   |   |
|------|---|---|---|---|---|
| 0 :  | 0 | 0 | 0 | 0 | 0 |
| 1 :  | 0 | 0 | 0 | 1 | 0 |
| 2 :  | 0 | 0 | 1 | 0 | 0 |
| 3 :  | 0 | 0 | 1 | 1 | 0 |
| 4 :  | 0 | 1 | 0 | 0 | 0 |
| 5 :  | 0 | 1 | 0 | 1 | 0 |
| 6 :  | 0 | 1 | 1 | 0 | 0 |
| 7 :  | 0 | 1 | 1 | 1 | 0 |
| 8 :  | 1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

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

[image source](#)

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

# Causal 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:  $e_{i,j} = q_j \cdot k_i / \sqrt{D}$

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

Output:  $y_j = \sum_i a_{i,j} v_i$

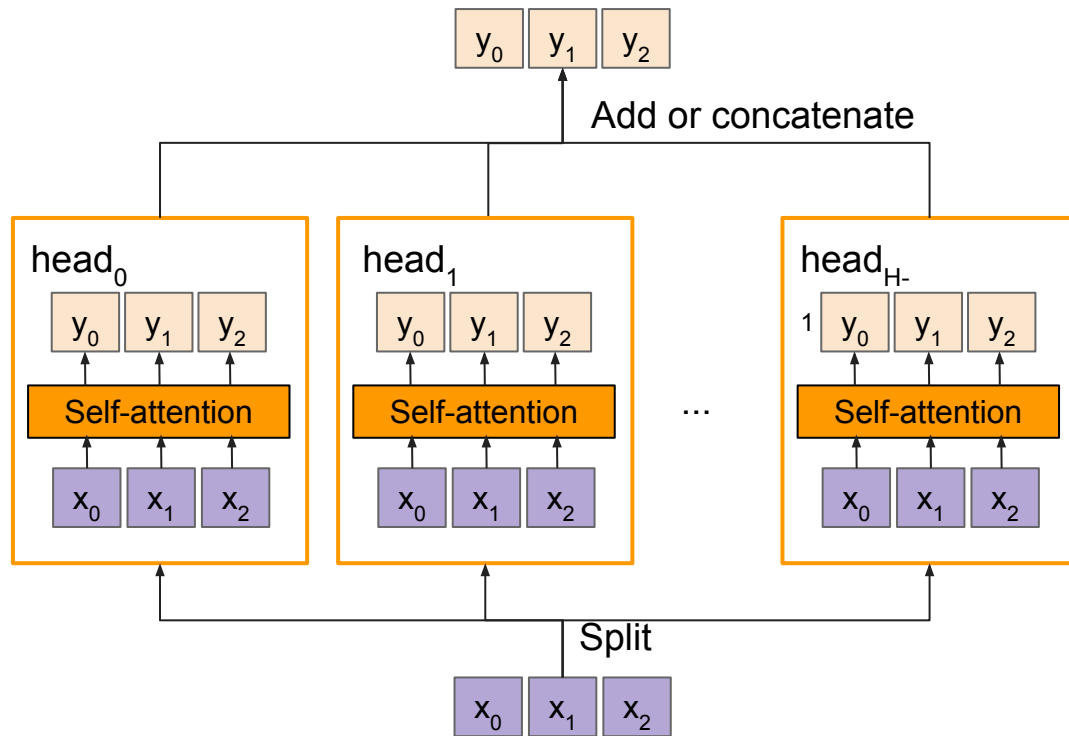
**Inputs:**

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

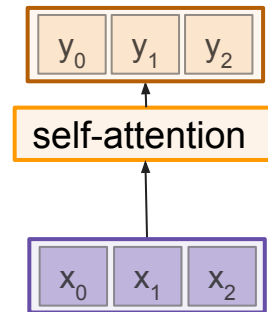
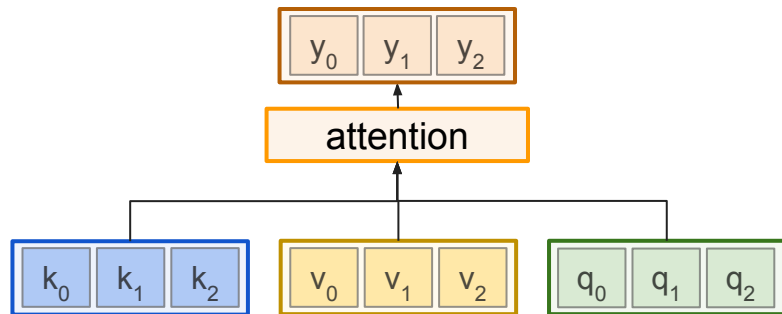
- Prevent vectors from looking at future vectors.
- Manually set alignment scores to -infinity

# Multi-head self-attention layer

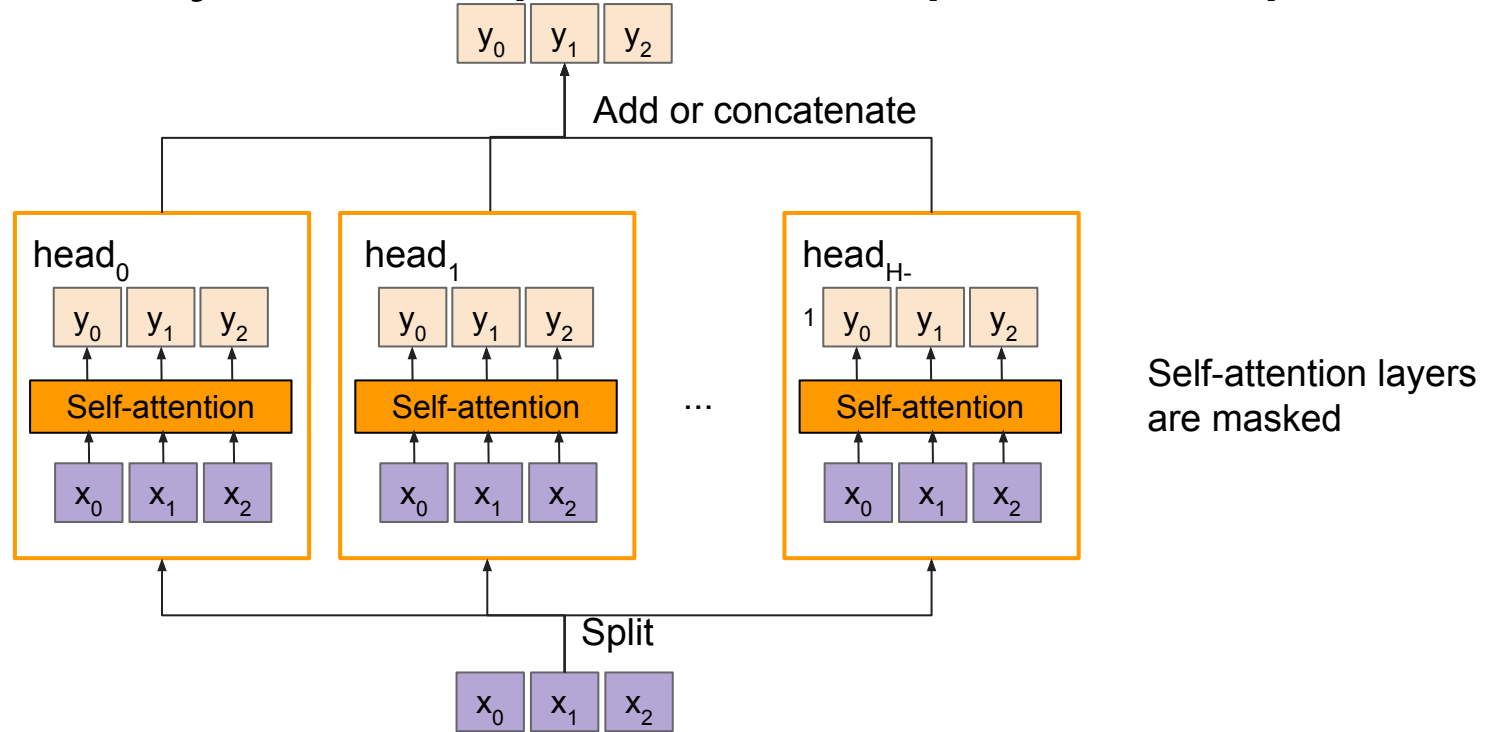
- Multiple self-attention heads in parallel



# General attention versus self-attention



# Attention layers can process sequential inputs



# Comparing RNNs to masked multi-headed attention

## 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.

## Masked multi-headed attention:

- (+) 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 scalars need to be calculated and stored for a single self-attention head. (but GPUs are getting bigger and better)

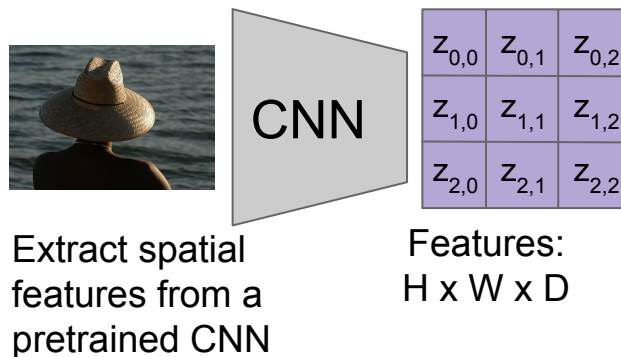
# Today's Agenda:

- **Attention with RNNs**
  - In Computer Vision
  - In NLP
- **General Attention Layer**
  - Self-attention
  - Positional encoding
  - Masked attention
  - Multi-head attention
- **Transformers**

# Image Captioning using transformers

**Input:** Image  $I$

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



# Image Captioning using transformers

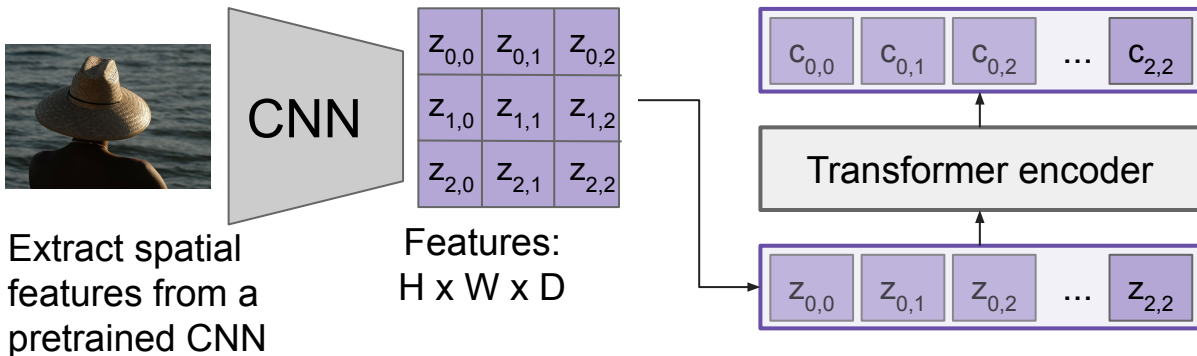
**Input:** Image  $I$

**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  $\mathbf{y} = y_1, y_2, \dots, y_T$

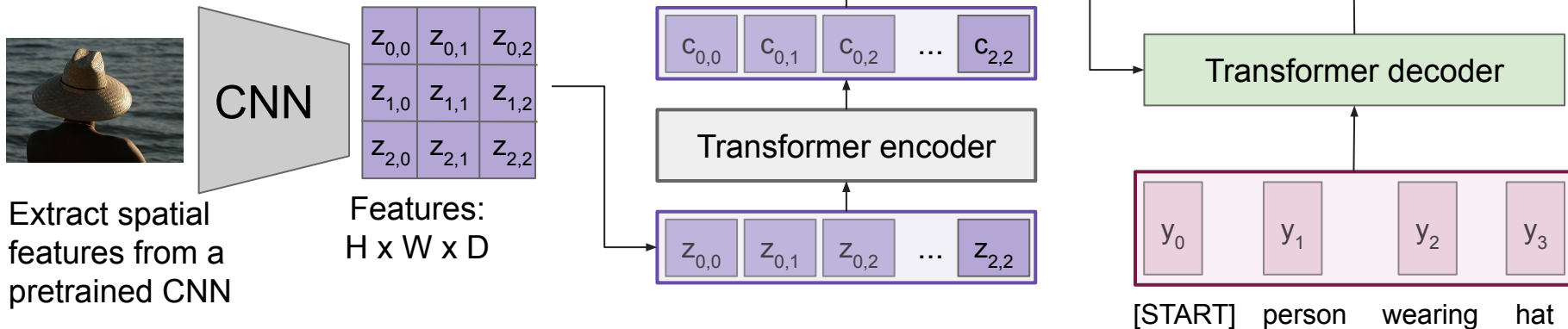
**Encoder:**  $\mathbf{c} = T_w(\mathbf{z})$

where  $\mathbf{z}$  is spatial CNN features

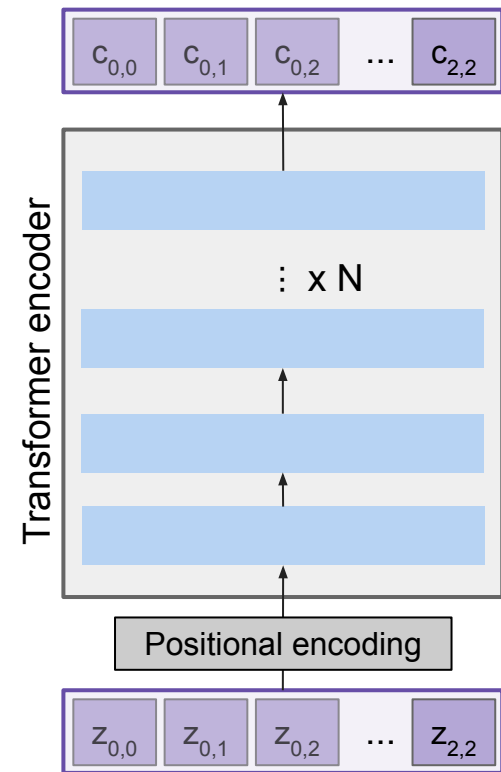
$T_w(\cdot)$  is the transformer encoder

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

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



# The Transformer encoder block

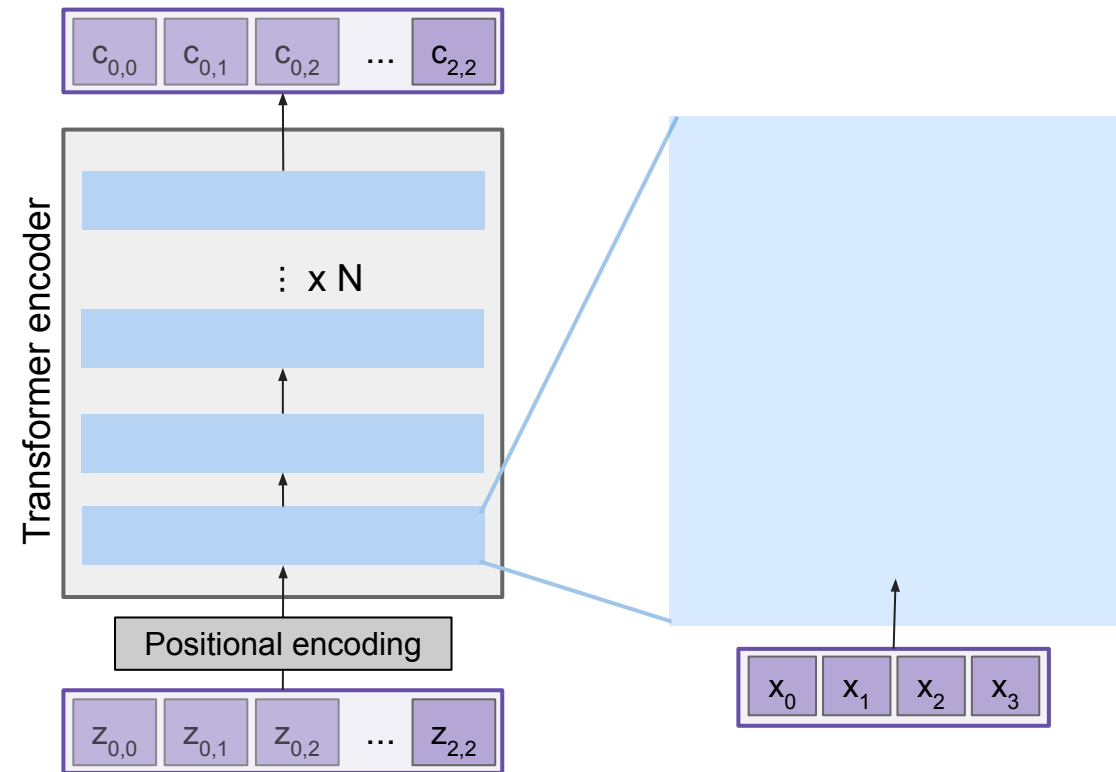


Made up of N encoder blocks.

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

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

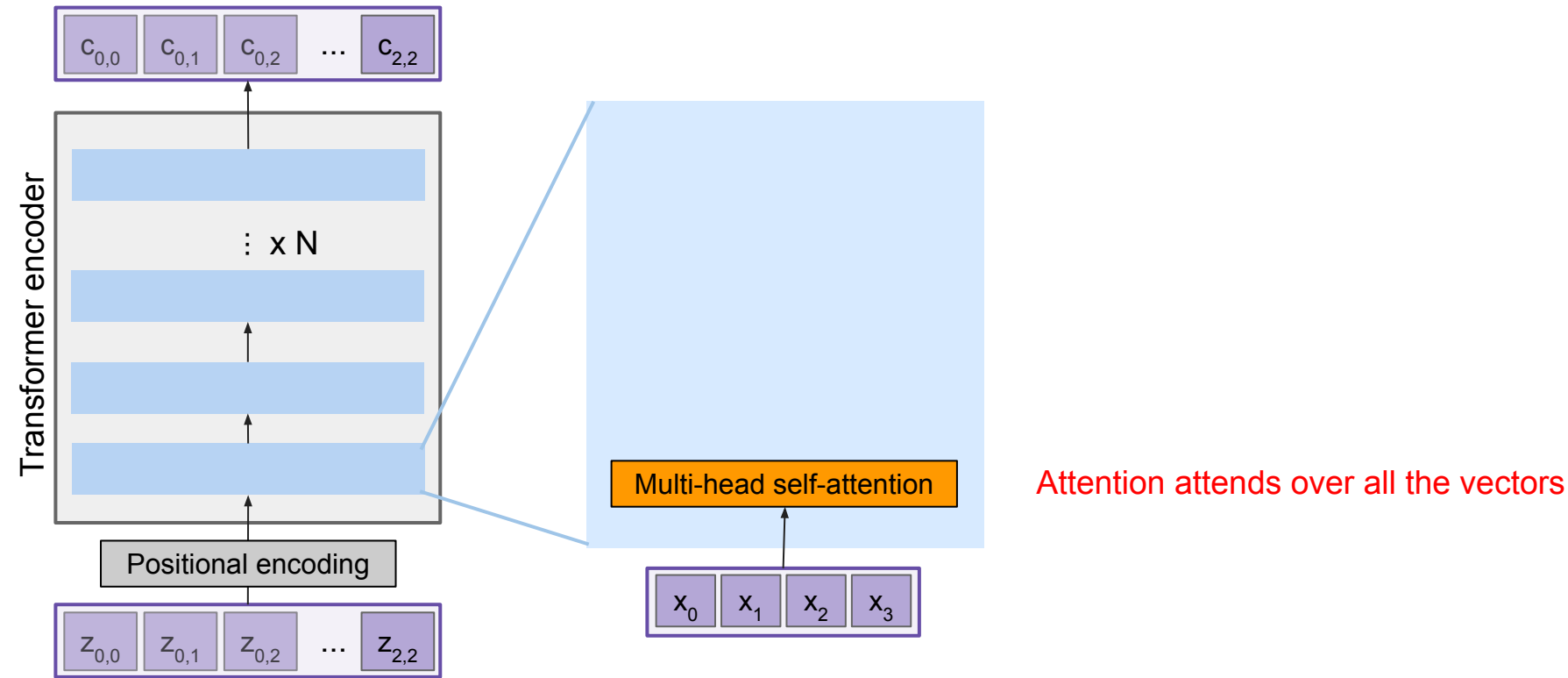
# The Transformer encoder block



Let's dive into one encoder block

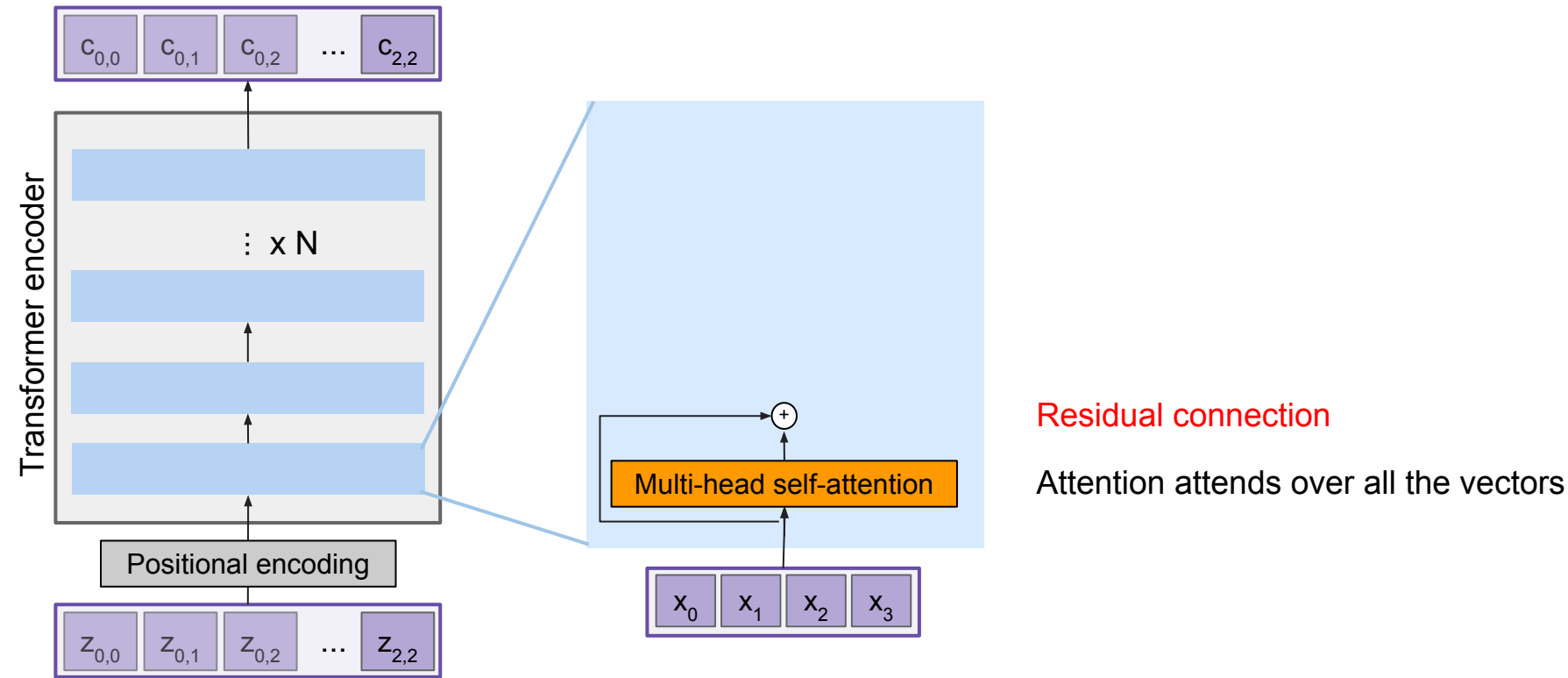
Vaswani et al, "Attention is all you need", NeurIPS 2017

# The Transformer encoder block



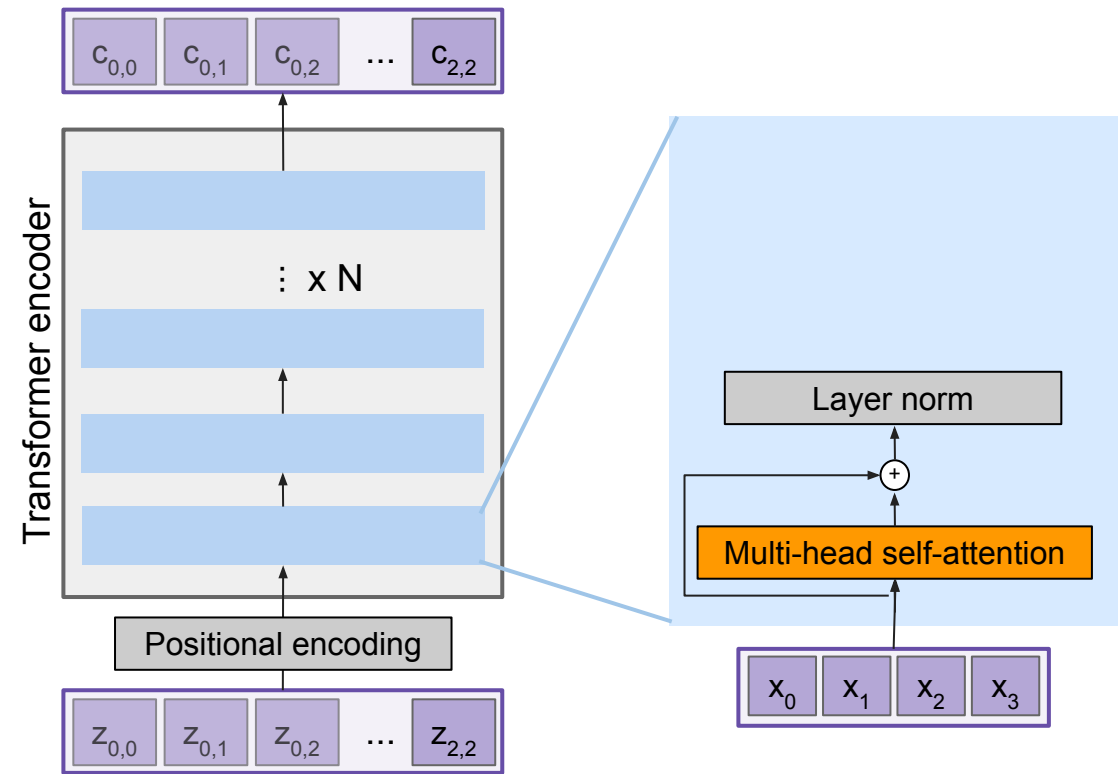
Vaswani et al, "Attention is all you need", NeurIPS 2017

# The Transformer encoder block



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

# The Transformer encoder block



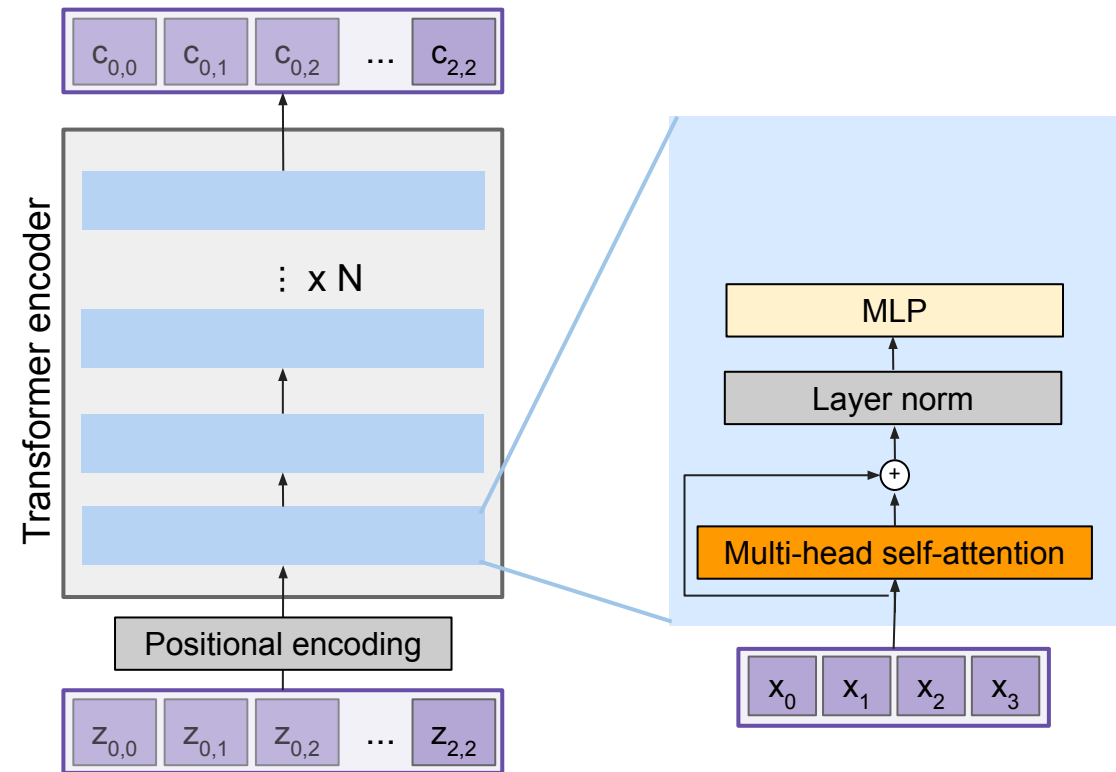
LayerNorm over each vector individually

Residual connection

Attention attends over all the vectors

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

# The Transformer encoder block



MLP (2 layers with GeLU) over each vector individually

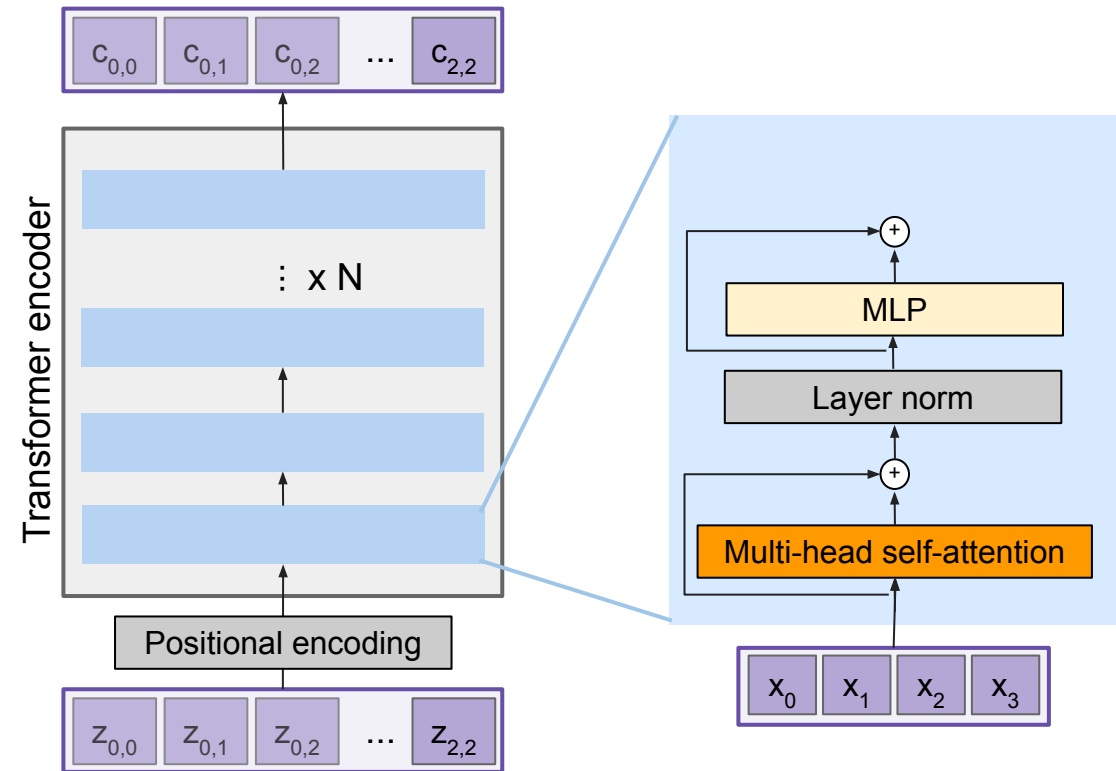
LayerNorm over each vector individually

Residual connection

Attention attends over all the vectors

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

# The Transformer encoder block



Residual connection

MLP (2 layers with GeLU) over each vector individually

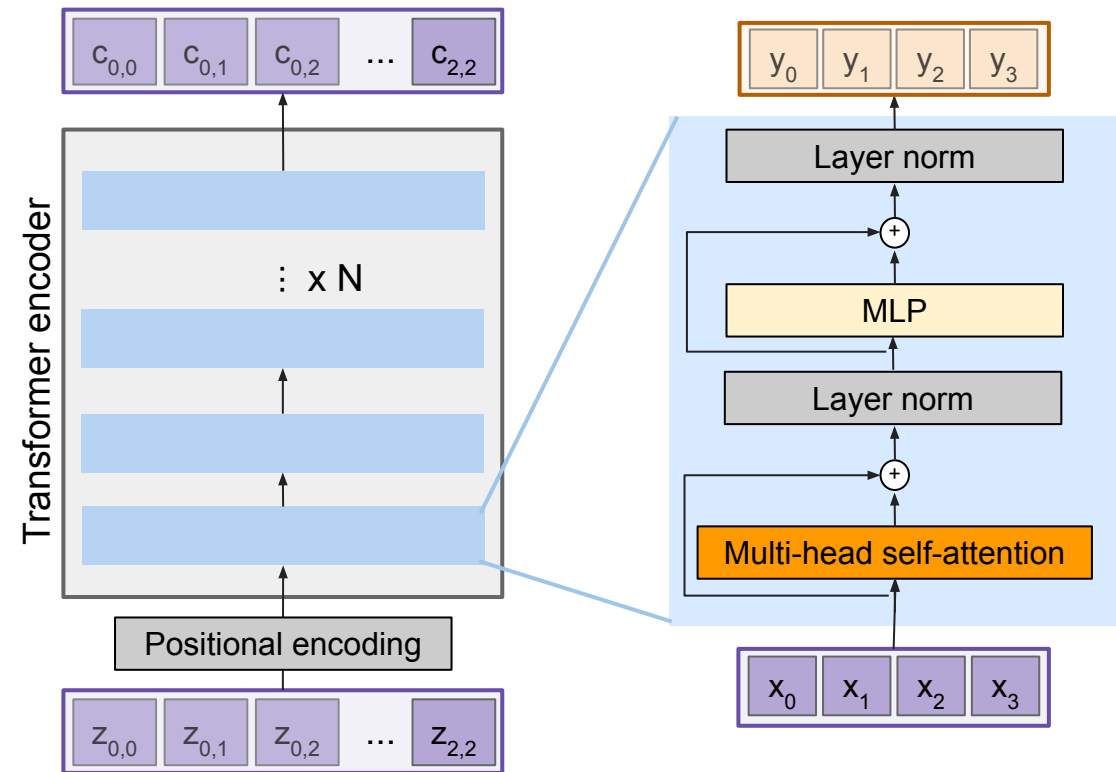
LayerNorm over each vector individually

Residual connection

Attention attends over all the vectors

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.

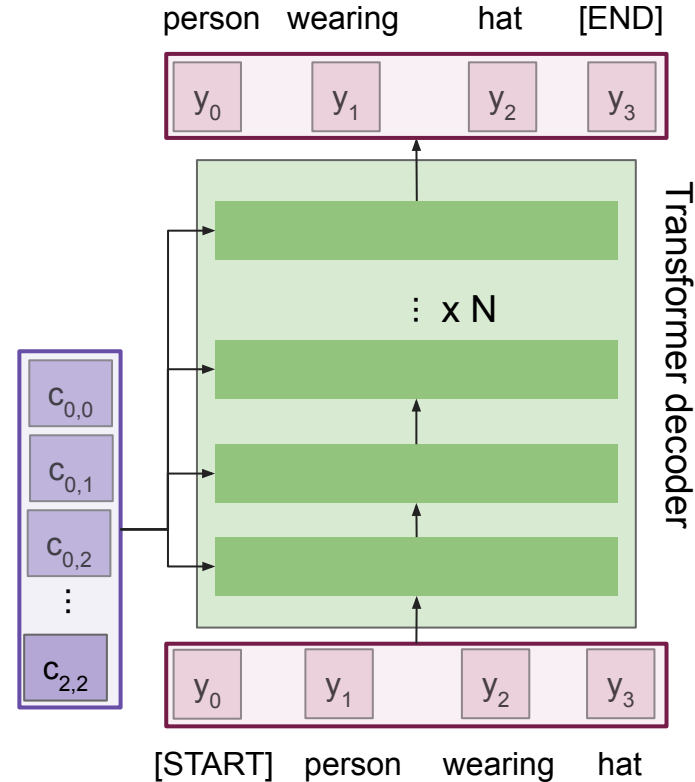
Layer norm and MLP operate independently per vector.

Highly scalable, highly parallelizable, but high memory usage.

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

# The Transformer

## Decoder block



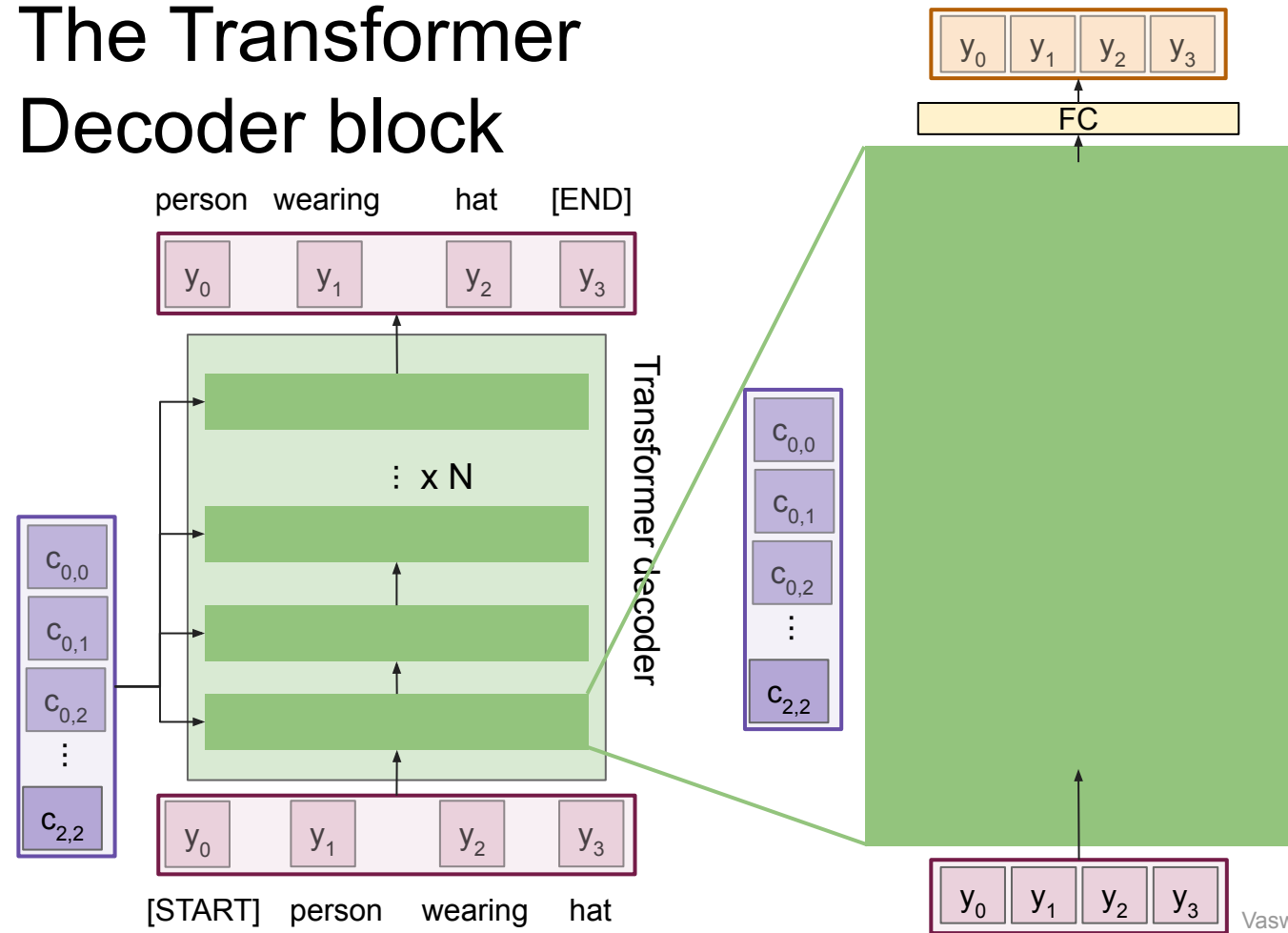
Made up of  $N$  decoder blocks.

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

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

# The Transformer

## Decoder block

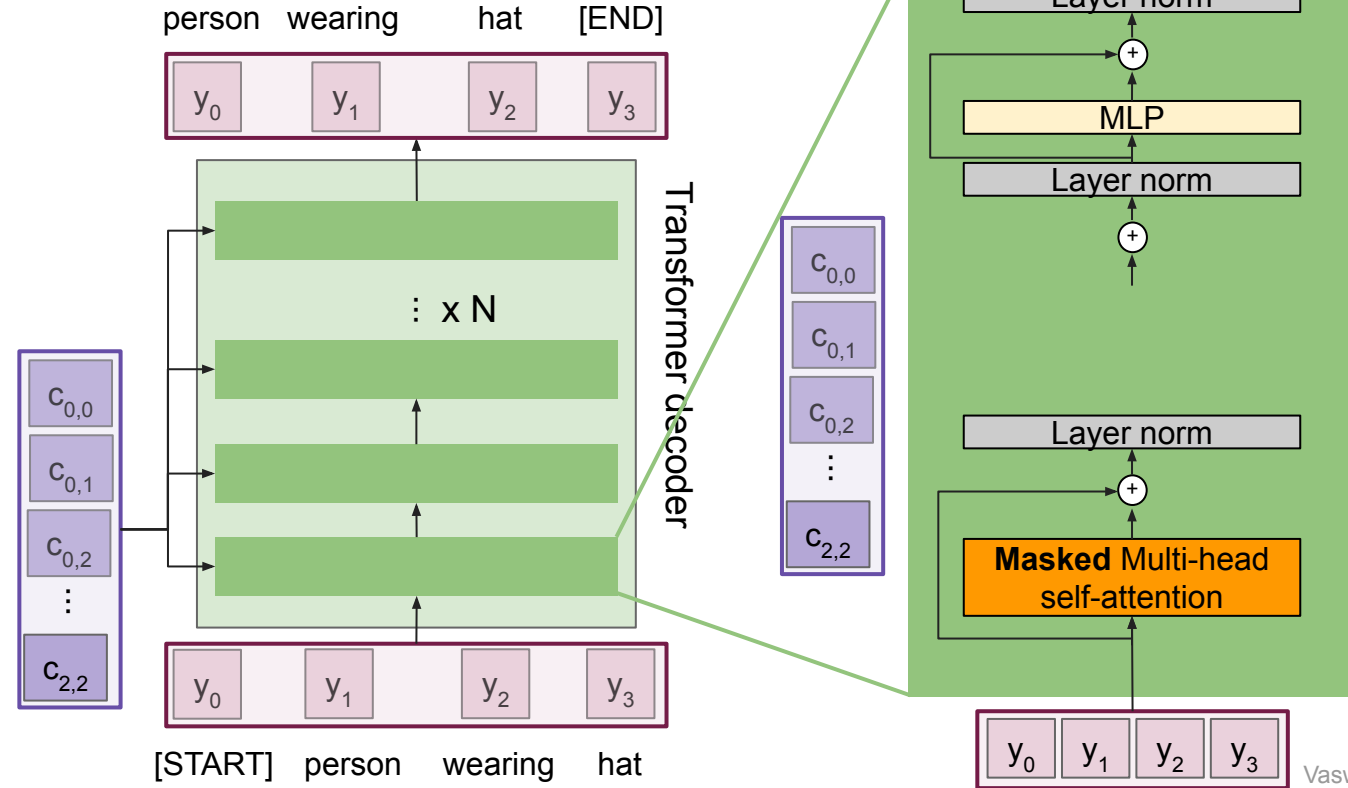


Let's dive into the  
transformer decoder block

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

# The Transformer

## Decoder block

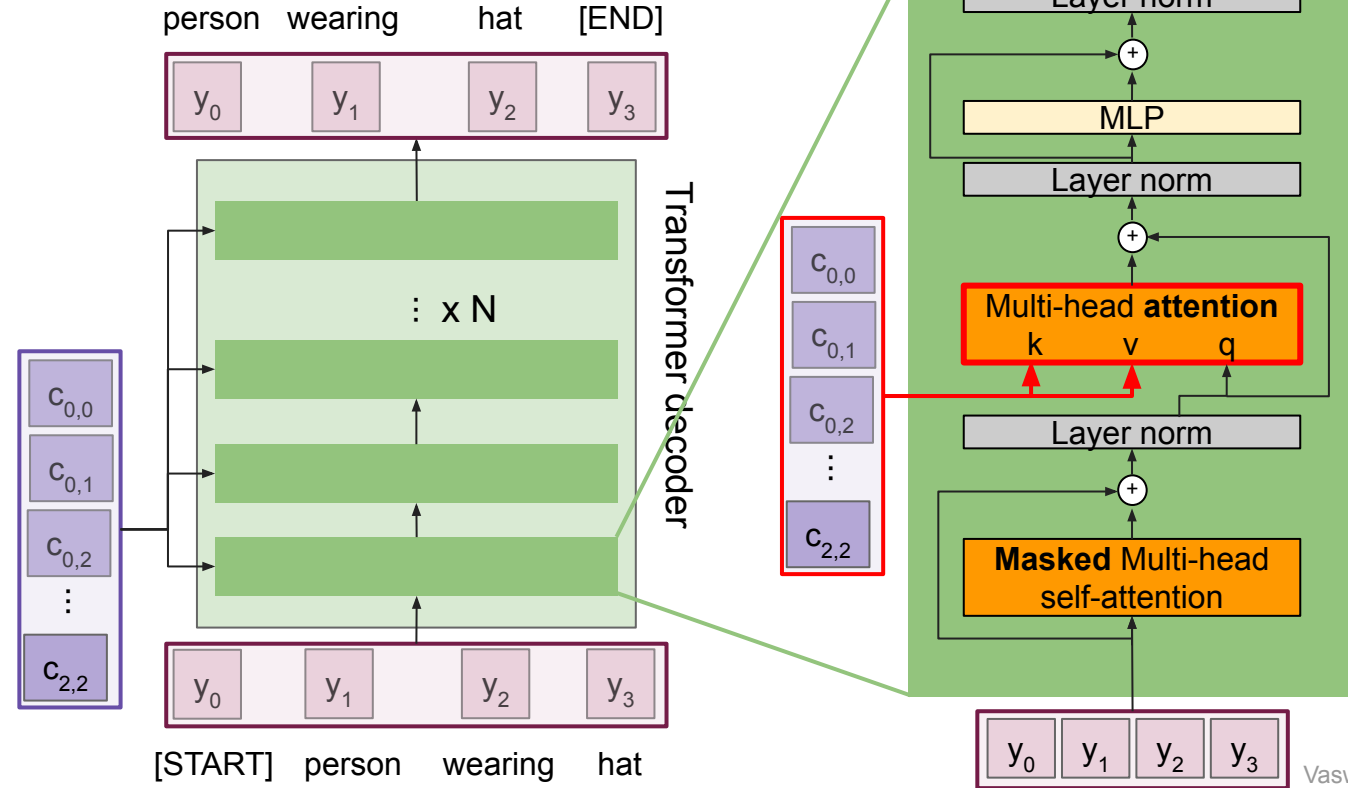


Most of the network is the same the transformer encoder.

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

# The Transformer

## Decoder block



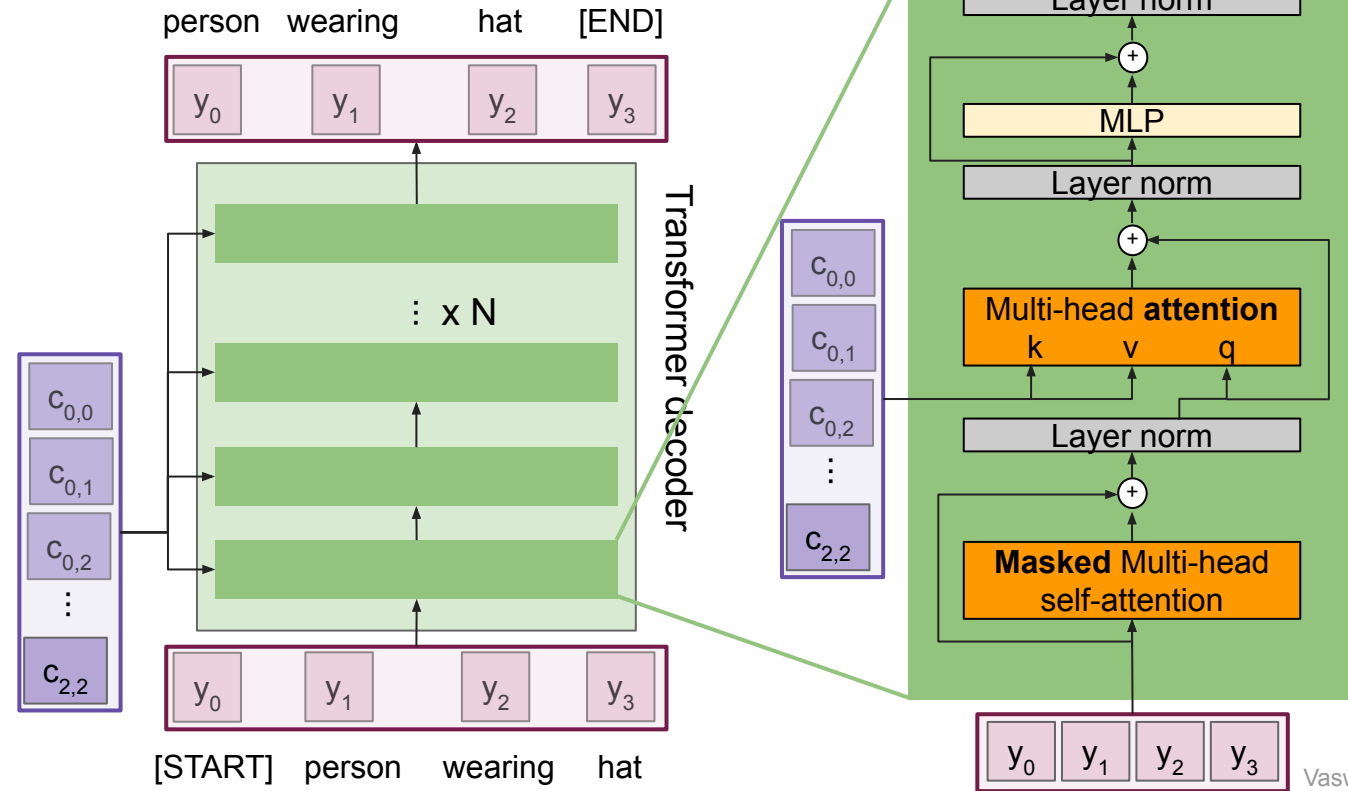
**Multi-head attention** block attends over the transformer encoder outputs.

For image captions, this is how we inject image features into the decoder.

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

# The Transformer

## Decoder block



### Transformer Decoder Block:

**Inputs:** Set of vectors  $\mathbf{x}$  and Set of context vectors  $\mathbf{c}$ .  
**Outputs:** Set of vectors  $\mathbf{y}$ .

Masked Self-attention only interacts with past inputs.

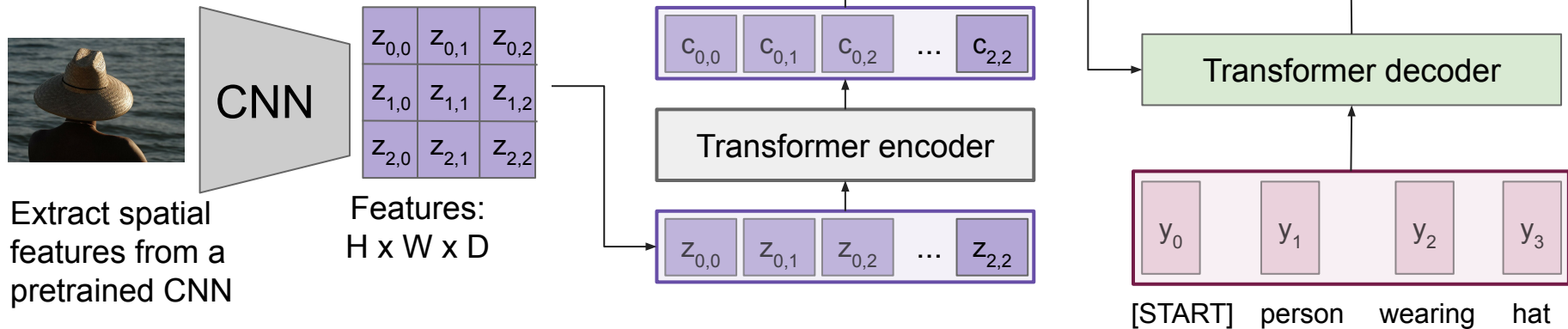
Multi-head attention block is NOT self-attention. It attends over encoder outputs.

Highly scalable, highly parallelizable, but high memory usage.

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

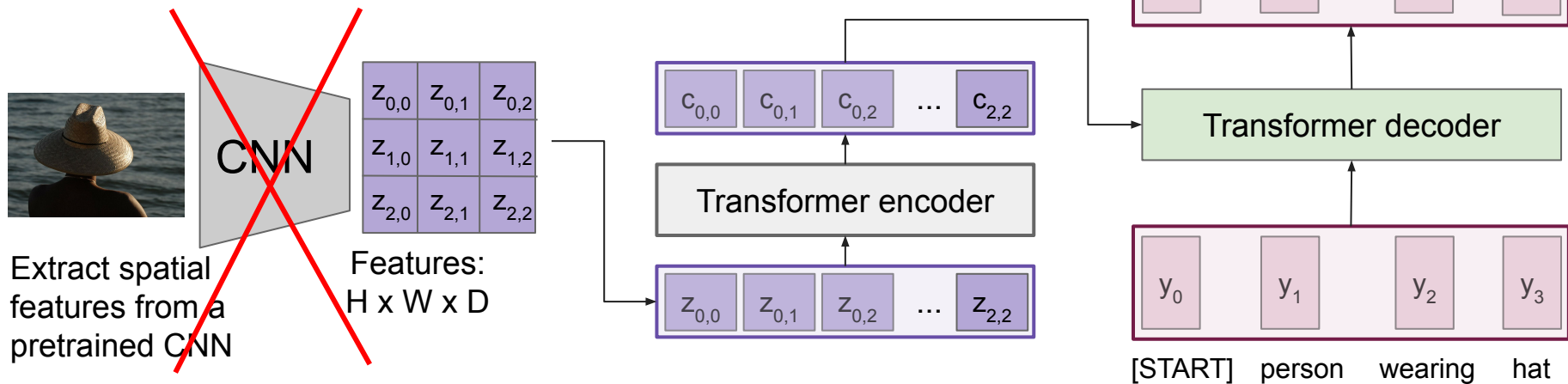
# Image Captioning using transformers

- No recurrence at all



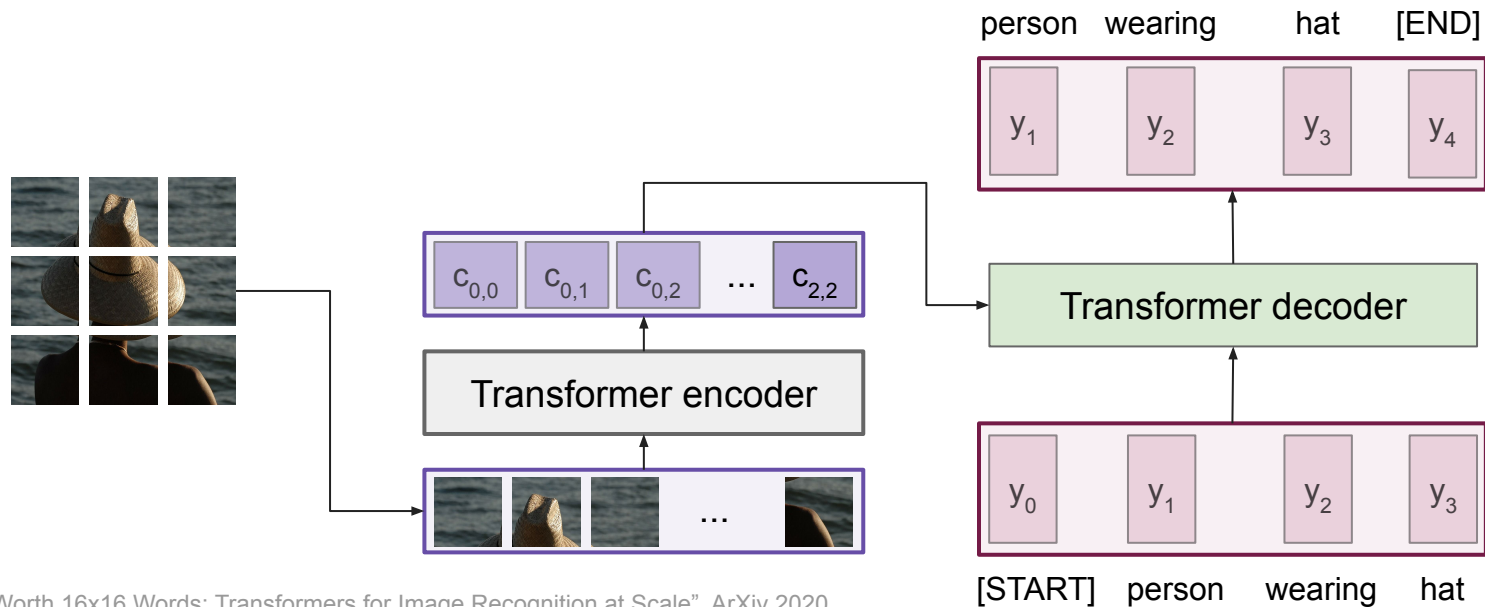
# Image Captioning using transformers

- 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

# Image Captioning using **ONLY** transformers

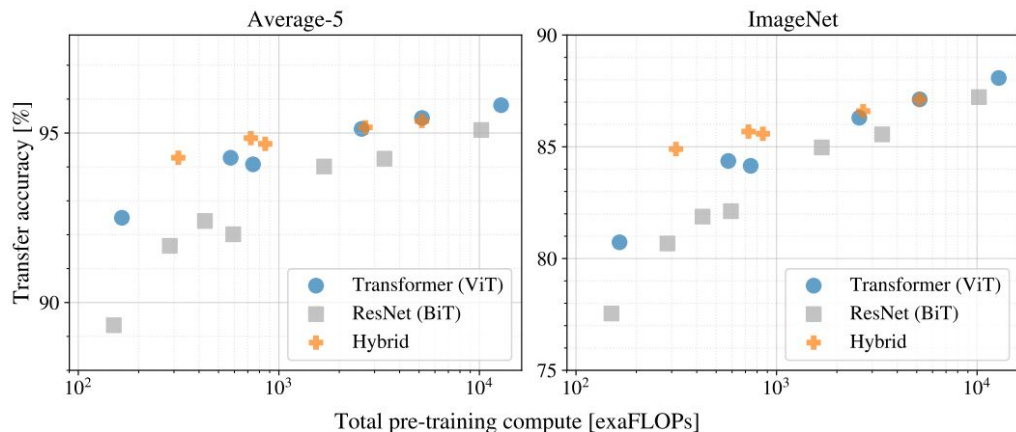


Figure 5: Performance versus cost for different architectures: Vision Transformers, ResNets, and hybrids. Vision Transformers generally outperform ResNets with the same computational budget. Hybrids improve upon pure Transformers for smaller model sizes, but the gap vanishes for larger models.

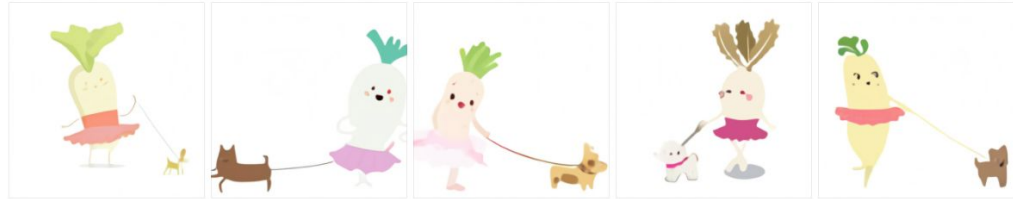
Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ArXiv 2020  
[Colab link](#) to an implementation of vision transformers

# New large-scale transformer models

TEXT PROMPT

an illustration of a baby daikon radish in a tutu walking a dog

AI-GENERATED IMAGES



[Edit prompt or view more images ↓](#)

TEXT PROMPT

an armchair in the shape of an avocado [...]

AI-GENERATED IMAGES



[Edit prompt or view more images ↓](#)

[link](#) to more examples

# Transformers today

| 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) |

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

# Transformers today

| 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    | 119M   | 13GB | ?                  |
| BERT-Large        | 24     | 1024  | 16    | 340M   | 13GB | ?                  |

Devlin et al, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", EMNLP 2018

# Transformers today

| 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    | 119M   | 13GB  | ?                       |
| BERT-Large        | 24     | 1024  | 16    | 340M   | 13GB  | ?                       |
| XLNet-Large       | 24     | 1024  | 16    | ~340M  | 126GB | 512x TPU-v3 (2.5 days)  |
| RoBERTa           | 24     | 1024  | 16    | 355M   | 160GB | 1024x V100 GPUs (1 day) |

Yang et al, "XLNet: Generalized Autoregressive Pretraining for Language Understanding", 2019 (Google)

Liu et al, "RoBERTa: A Robustly Optimized BERT Pretraining Approach", 2019 (Meta)

# Transformers today

| 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    | 119M   | 13GB  | ?                       |
| BERT-Large        | 24     | 1024  | 16    | 340M   | 13GB  | ?                       |
| XLNet-Large       | 24     | 1024  | 16    | ~340M  | 126GB | 512x TPU-v3 (2.5 days)  |
| RoBERTa           | 24     | 1024  | 16    | 355M   | 160GB | 1024x V100 GPUs (1 day) |
| GPT-2             | 48     | 1600  | ?     | 1.5B   | 40GB  | ?                       |

Radford et al, "Language models are unsupervised multitask learners", 2019 (OpenAI)

# Transformers today

| 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    | 119M   | 13GB  | ?                       |
| BERT-Large        | 24     | 1024  | 16    | 340M   | 13GB  | ?                       |
| XLNet-Large       | 24     | 1024  | 16    | ~340M  | 126GB | 512x TPU-v3 (2.5 days)  |
| RoBERTa           | 24     | 1024  | 16    | 355M   | 160GB | 1024x V100 GPUs (1 day) |
| GPT-2             | 48     | 1600  | ?     | 1.5B   | 40GB  | ?                       |
| Megatron-LM       | 72     | 3072  | 32    | 8.3B   | 174GB | 512x V100 GPU (9 days)  |

Shoeybi et al. "Megatron-Lm: Training multi-billion parameter language models using model parallelism." 2019. (Google)

# Transformers today

| 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    | 119M   | 13GB  | ?                       |
| BERT-Large        | 24     | 1024  | 16    | 340M   | 13GB  | ?                       |
| XLNet-Large       | 24     | 1024  | 16    | ~340M  | 126GB | 512x TPU-v3 (2.5 days)  |
| RoBERTa           | 24     | 1024  | 16    | 355M   | 160GB | 1024x V100 GPUs (1 day) |
| GPT-2             | 48     | 1600  | ?     | 1.5B   | 40GB  | ?                       |
| Megatron-LM       | 72     | 3072  | 32    | 8.3B   | 174GB | 512x V100 GPU (9 days)  |
| Turing-NLG        | 78     | 4256  | 28    | 17B    | ?     | 256x V100 GPUs          |

Microsoft, "Turing-NLG: A 17-billion parameter language model by Microsoft", 2020

# Transformers today

| 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    | 119M   | 13GB  | ?                       |
| BERT-Large        | 24     | 1024  | 16    | 340M   | 13GB  | ?                       |
| XLNet-Large       | 24     | 1024  | 16    | ~340M  | 126GB | 512x TPU-v3 (2.5 days)  |
| RoBERTa           | 24     | 1024  | 16    | 355M   | 160GB | 1024x V100 GPUs (1 day) |
| GPT-2             | 48     | 1600  | ?     | 1.5B   | 40GB  | ?                       |
| Megatron-LM       | 72     | 3072  | 32    | 8.3B   | 174GB | 512x V100 GPU (9 days)  |
| Turing-NLG        | 78     | 4256  | 28    | 17B    | ?     | 256x V100 GPUs          |
| GPT-3             | 96     | 12288 | 96    | 175B   | 694GB | ?                       |

Brown et al, "Language Models are Few-Shot Learners", NeurIPS 2020

# Transformers today

Rae et al, "Scaling Language Models: Methods, Analysis, & Insights from Training Gopher", arXiv 2021 (Google)

| 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    | 119M   | 13GB    | ?                       |
| BERT-Large        | 24     | 1024  | 16    | 340M   | 13GB    | ?                       |
| XLNet-Large       | 24     | 1024  | 16    | ~340M  | 126GB   | 512x TPU-v3 (2.5 days)  |
| RoBERTa           | 24     | 1024  | 16    | 355M   | 160GB   | 1024x V100 GPUs (1 day) |
| GPT-2             | 48     | 1600  | ?     | 1.5B   | 40GB    | ?                       |
| Megatron-LM       | 72     | 3072  | 32    | 8.3B   | 174GB   | 512x V100 GPU (9 days)  |
| Turing-NLG        | 78     | 4256  | 28    | 17B    | ?       | 256x V100 GPUs          |
| GPT-3             | 96     | 12288 | 96    | 175B   | 694GB   | ?                       |
| Gopher            | 80     | 16384 | 128   | 280B   | 10.55TB | 4096x TPU-v3 (38 days)  |

| 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    | 119M   | 13GB    | ?                       |
| BERT-Large        | 24     | 1024  | 16    | 340M   | 13GB    | ?                       |
| XLNet-Large       | 24     | 1024  | 16    | ~340M  | 126GB   | 512x TPU-v3 (2.5 days)  |
| RoBERTa           | 24     | 1024  | 16    | 355M   | 160GB   | 1024x V100 GPUs (1 day) |
| GPT-2             | 48     | 1600  | ?     | 1.5B   | 40GB    | ?                       |
| Megatron-LM       | 72     | 3072  | 32    | 8.3B   | 174GB   | 512x V100 GPU (9 days)  |
| Turing-NLG        | 78     | 4256  | 28    | 17B    | ?       | 256x V100 GPUs          |
| GPT-3             | 96     | 12288 | 96    | 175B   | 694GB   | ?                       |
| Gopher            | 80     | 16384 | 128   | 280B   | 10.55TB | 4096x TPU-v3 (38 days)  |
| GPT-4             | ?      | ?     | ?     | 1.8T   | ?       | ?                       |

# 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.
  - It is highly **scalable** and highly **parallelizable**
  - **Faster** training, **larger** models, **better** performance across vision and language tasks
  - They are quickly replacing RNNs, LSTMs, and may even replace convolutions.

Next time: Modern architectures