# Lecture 11:
# Attention and Transformers

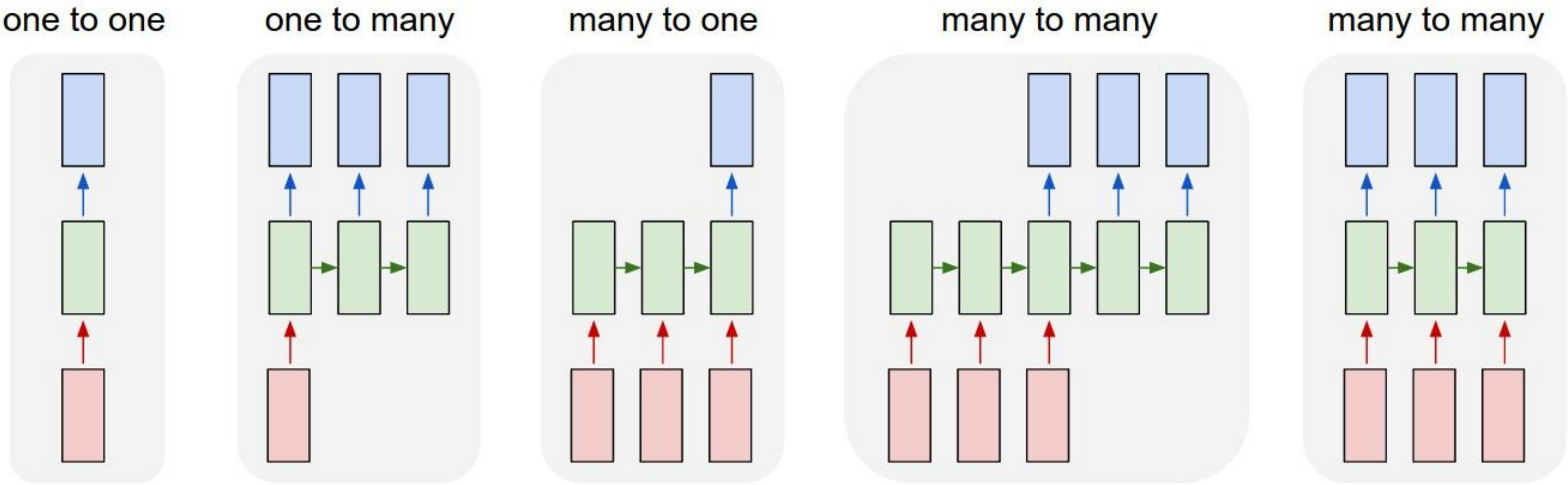
# Administrative: Midterm

- Midterm was this Tuesday
- We will be grading this week and you should have grades by next week.

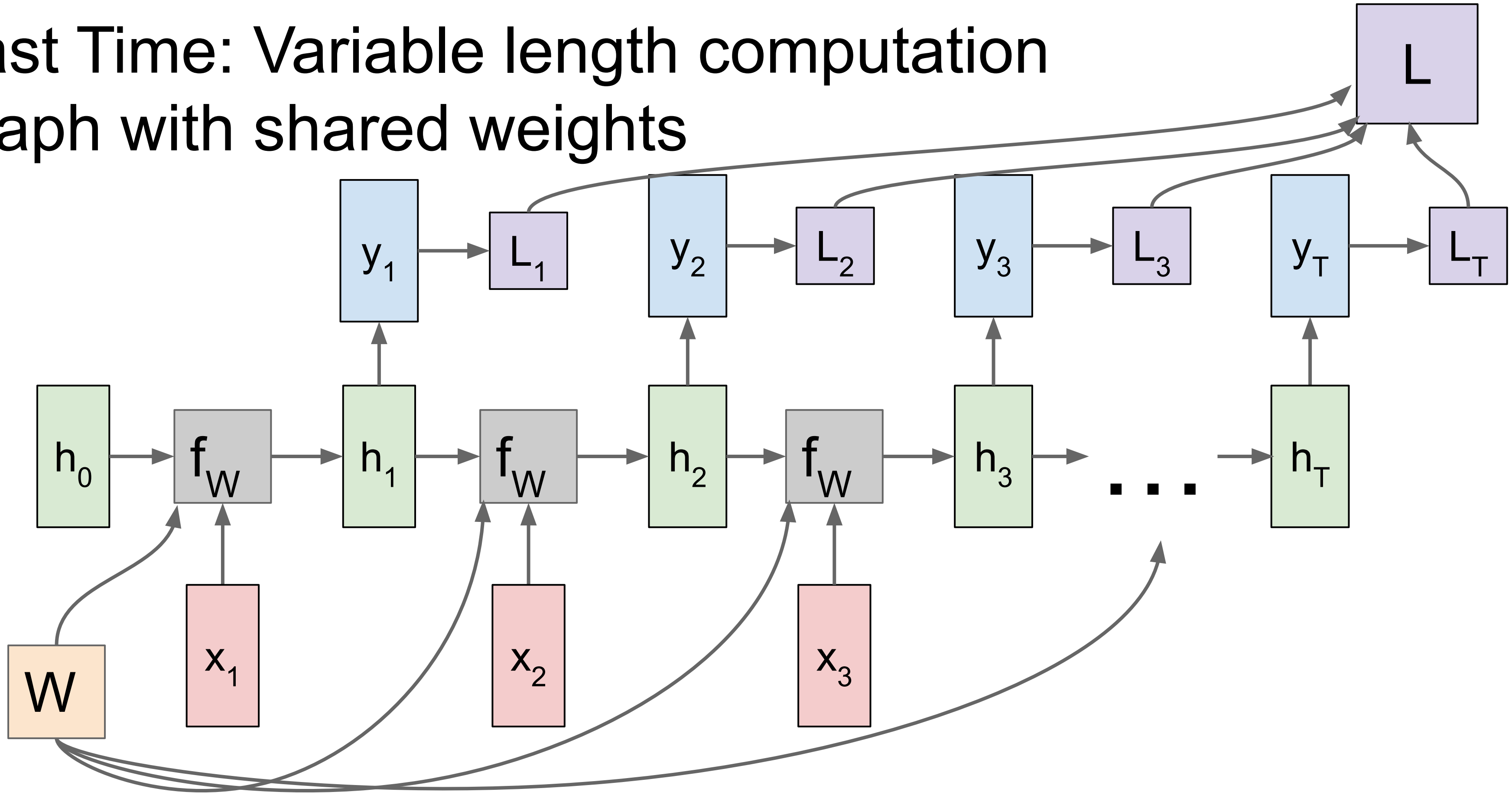
# Administrative: Assignment 3

- A3 is due Friday May 25th, 11:59pm
    - Lots of applications of ConvNets
    - Also contains an extra credit notebook, which is worth an additional 5% of the A3 grade.
    - Extra credit will not be used when curving the class grades.

# Last Time: Recurrent Neural Networks



# Last Time: Variable length computation graph with shared weights



# Let's jump to lecture 10 - slide 43

# Today's Agenda:

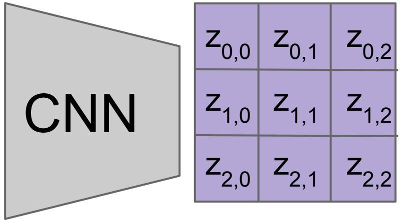
- **Attention with RNNs**
  - In Computer Vision
  - In NLP
- **General Attention Layer**
  - Self-attention
  - Positional encoding
  - Masked attention
  - Multi-head attention
- **Transformers**

# Today's Agenda:

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

# Image Captioning using spatial features

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




Extract spatial features from a pretrained CNN

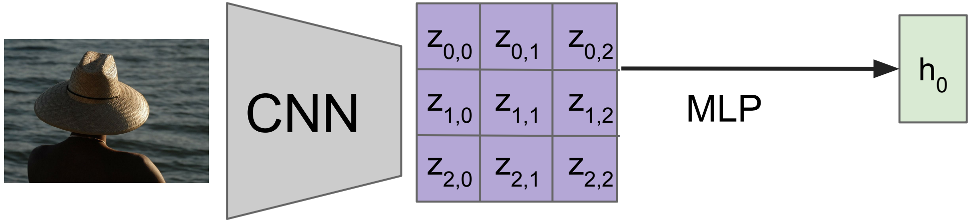
Features: H x W x D

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

# Image Captioning using spatial features

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

**Encoder**: $h_0 = f_{\mathbf{W}}(\mathbf{z})$
where **z** is spatial CNN features
$f_{\mathbf{W}}(.)$ is an MLP




Extract spatial features from a pretrained CNN

Features: H x W x D

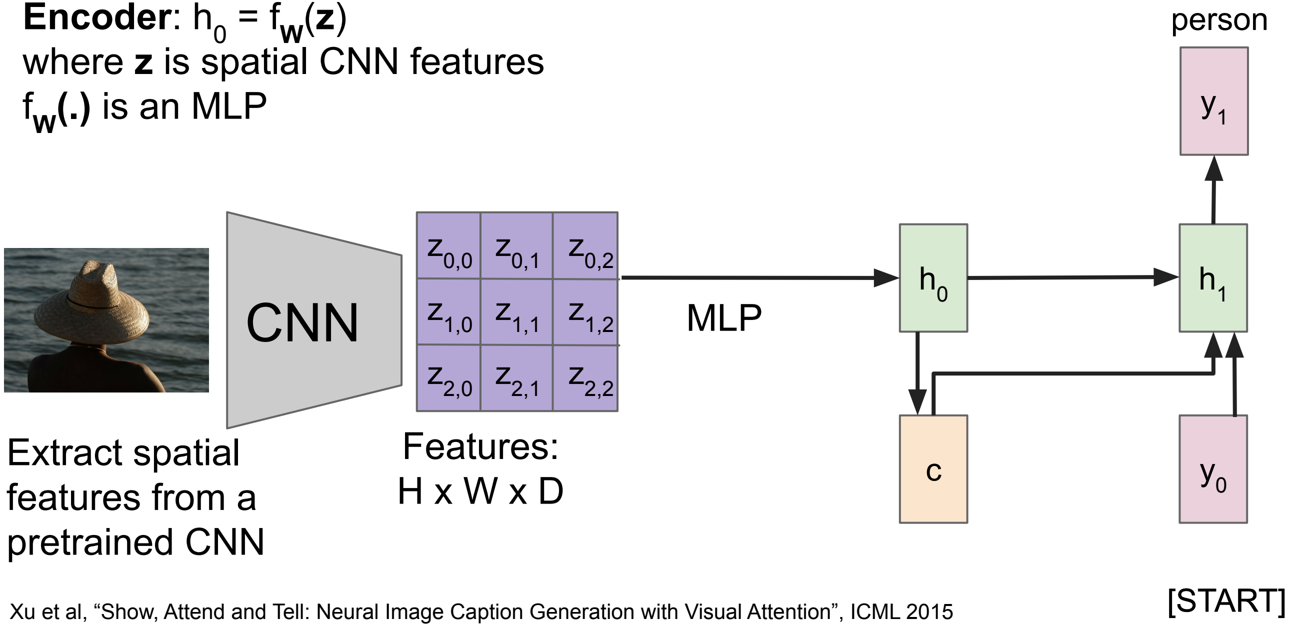
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Image Captioning using spatial features

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

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

**Encoder**: $h_0 = f_{\mathbf{W}}(\mathbf{z})$
where **z** is spatial CNN features
$f_{\mathbf{W}}(.)$ is an MLP




Extract spatial features from a pretrained CNN

Features: H x W x D

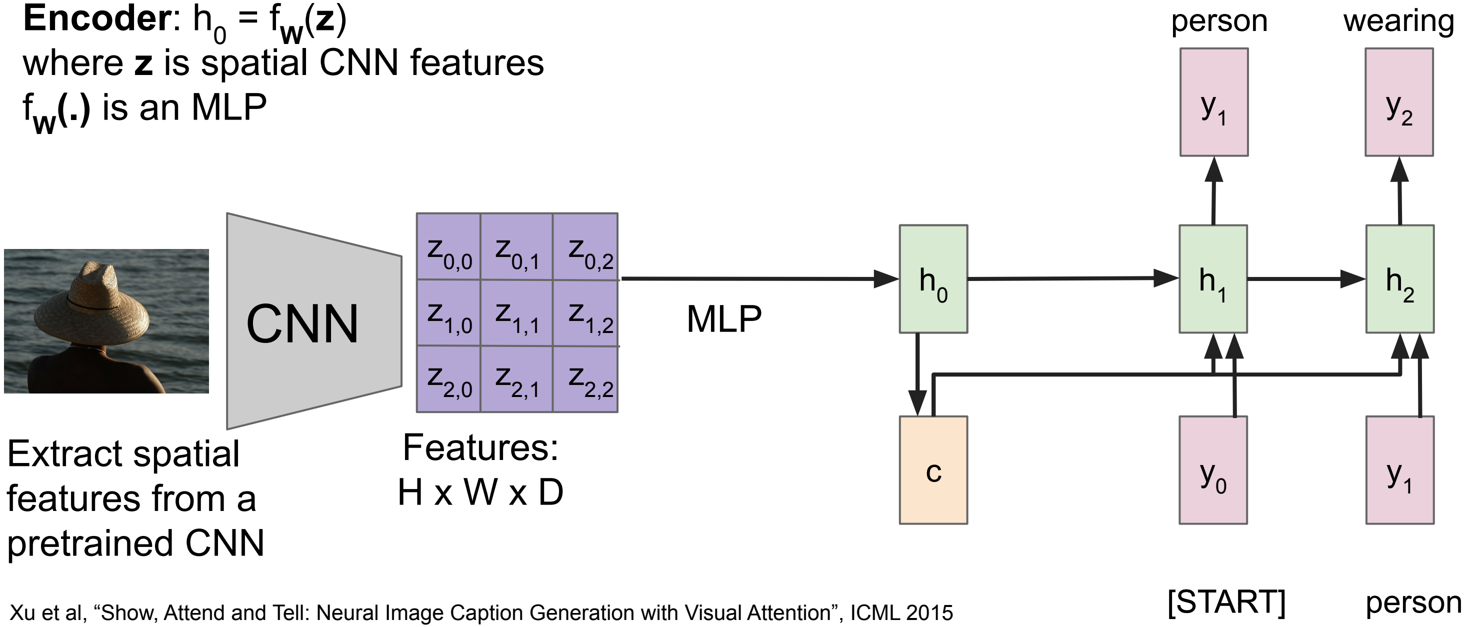
Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

# Image Captioning using spatial features

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

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

**Encoder**: $h_0 = f_{\mathbf{W}}(\mathbf{z})$
where $\mathbf{z}$ is spatial CNN features
$f_{\mathbf{W}}(.)$ is an MLP




Extract spatial features from a pretrained CNN

Features: H x W x D

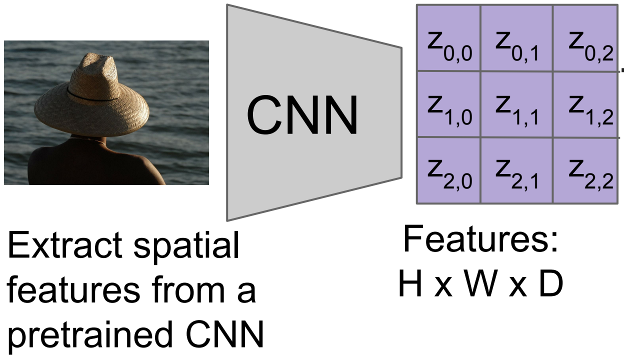
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

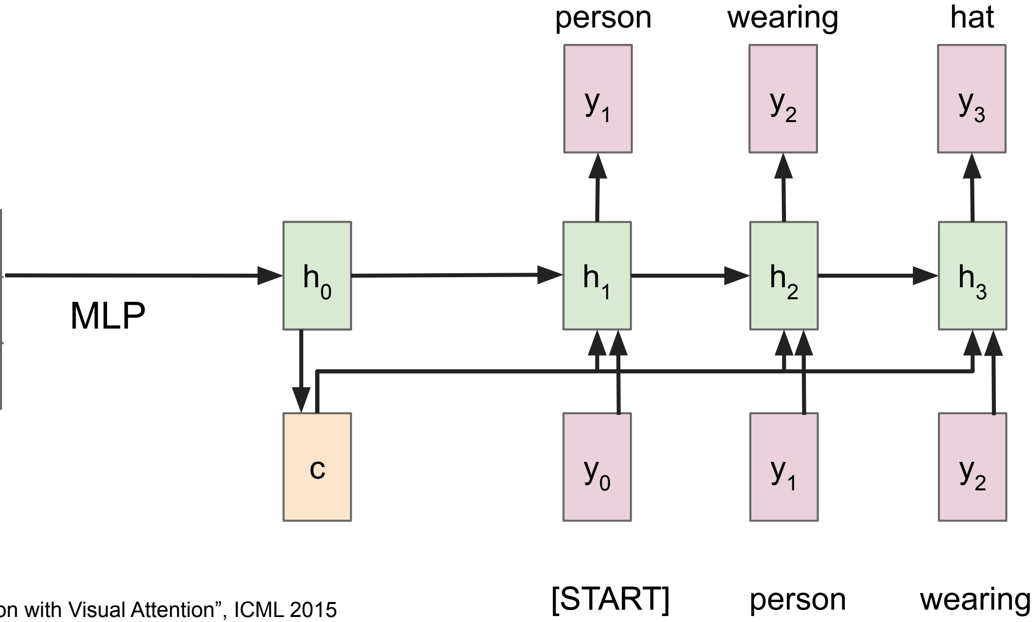
# Image Captioning using spatial features

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

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

**Encoder**: $h_0 = f_{\mathbf{W}}(\mathbf{z})$
where **z** is spatial CNN features
$f_{\mathbf{W}}(.)$ is an MLP



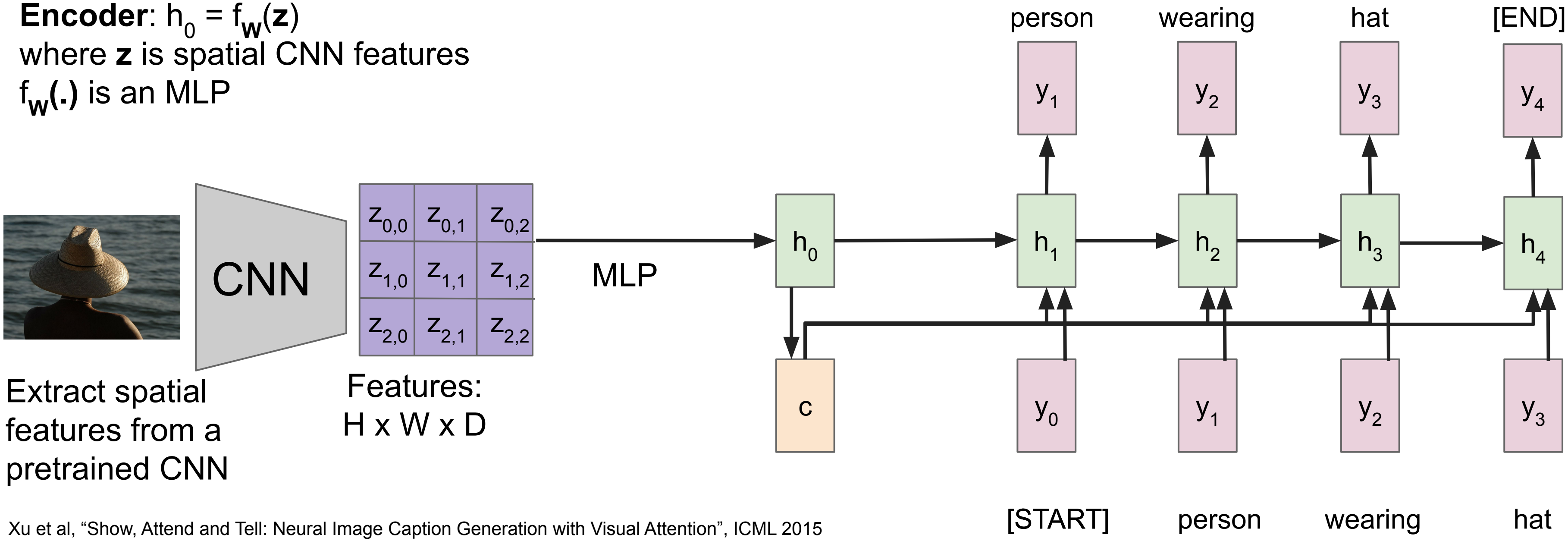

Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

# Image Captioning using spatial features

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

**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 **z** is spatial CNN features
$f_W(.)$ is an MLP




Extract spatial features from a pretrained CNN

Features: H x W x D

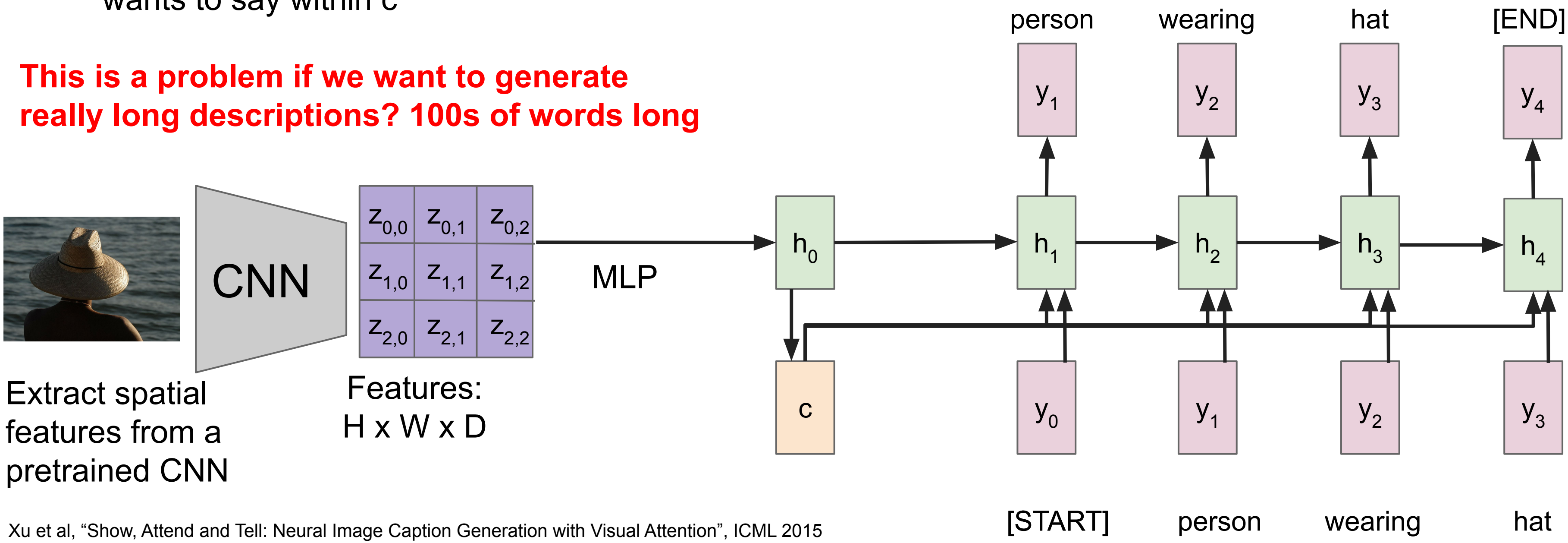
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Image Captioning using spatial features

**Problem: Input is "bottlenecked" through c**

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

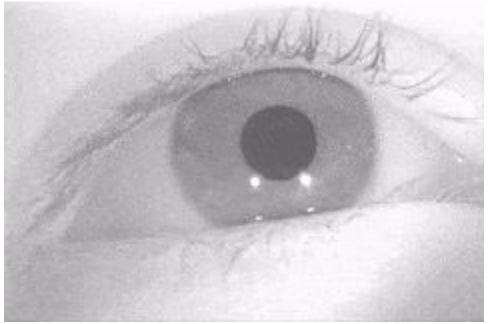
**This is a problem if we want to generate really long descriptions? 100s of words long**




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

# Image Captioning with RNNs & Attention

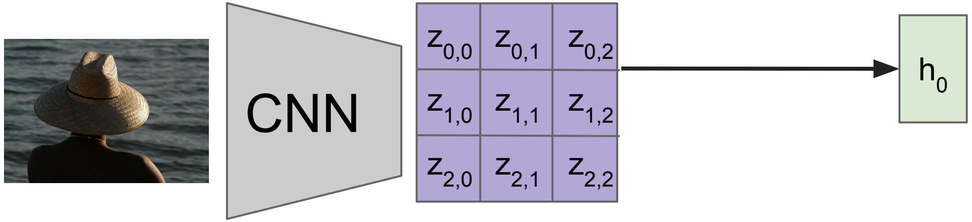
gif source



Attention Saccades in humans

**Attention idea: New context vector at every time step.**

**Each context vector will attend to different image regions**




Extract spatial features from a pretrained CNN

Features: H x W x D

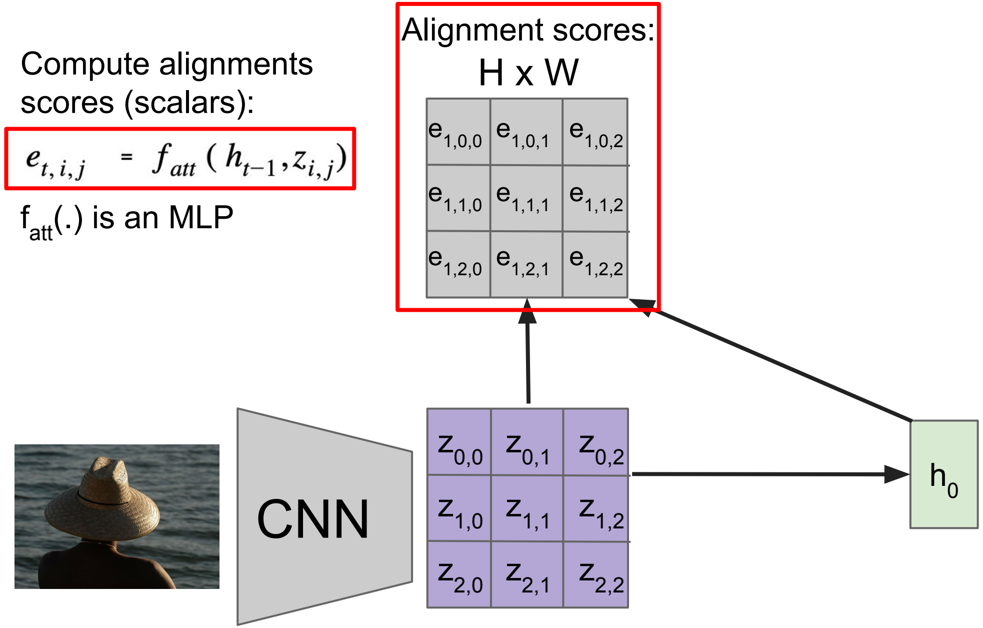
Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

# Image Captioning with RNNs & Attention

Compute alignments scores (scalars):

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

$f_{att}(.)$ is an MLP




Extract spatial features from a pretrained CNN

Features:
H x W x D

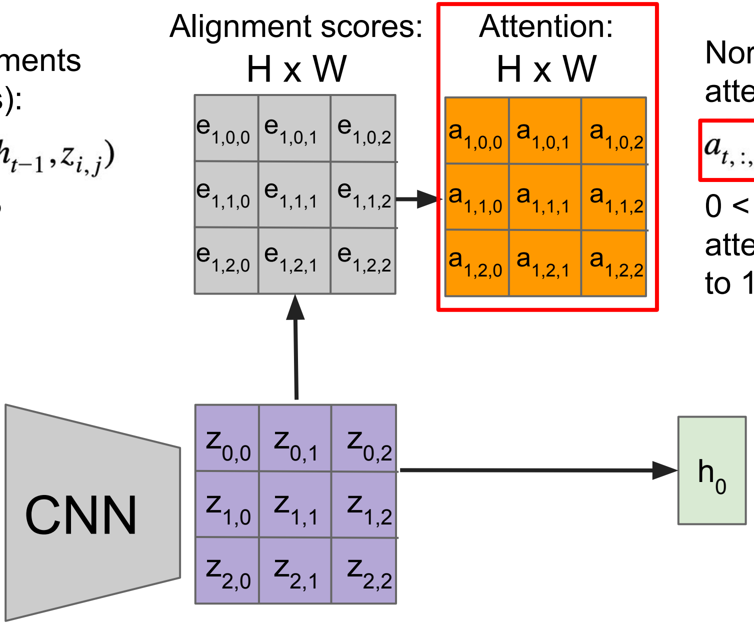
Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

# Image Captioning with RNNs & Attention

Compute alignments scores (scalars):

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

$f_{att}(.)$ is an MLP




Normalize to get attention weights:

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

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

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 & Attention

Compute alignments scores (scalars):

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

$f_{att}(.)$ is an MLP

Alignment scores: H x W

Attention: H x W

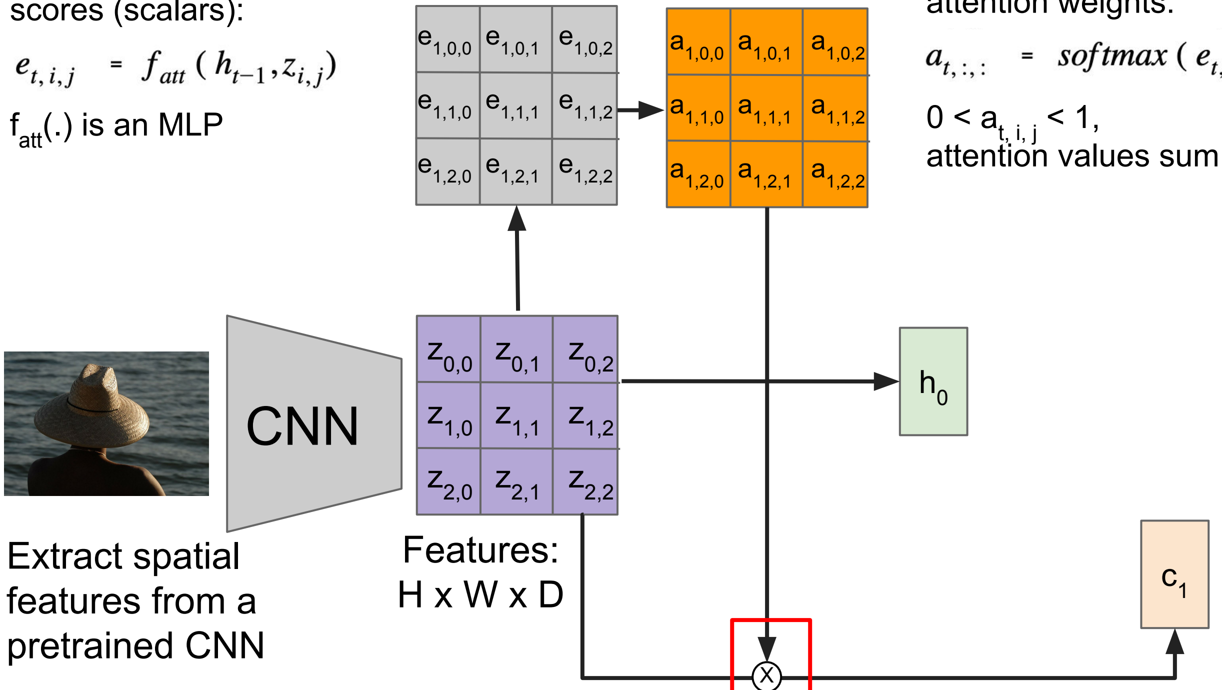
Normalize to get attention weights:

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

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

Compute context vector:

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




Extract spatial features from a pretrained CNN

Features: H x W x D

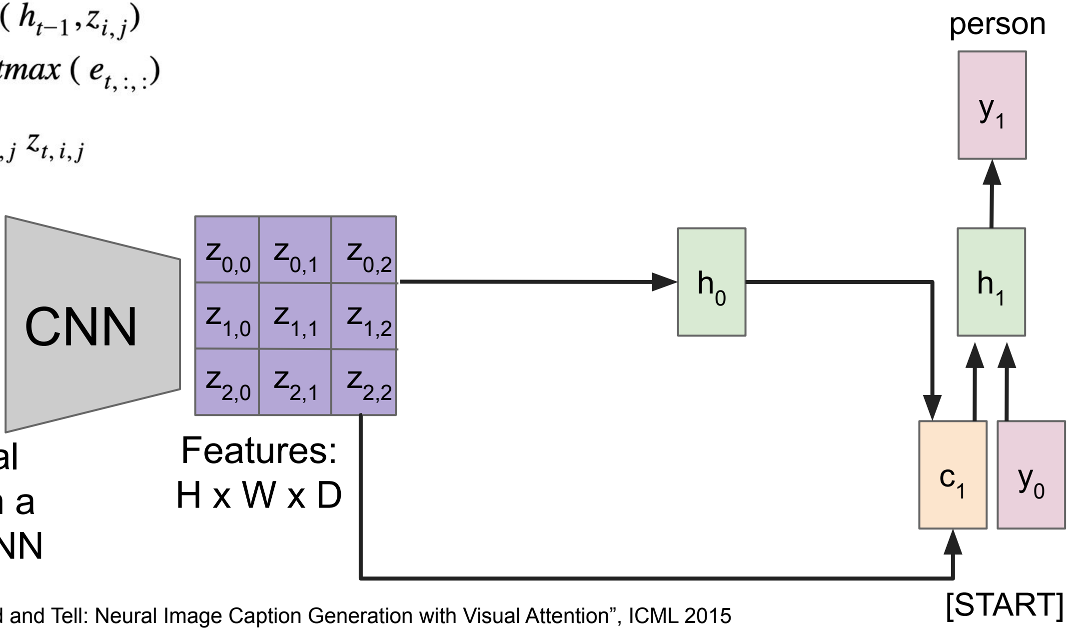
Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

# Image Captioning with RNNs & 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

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



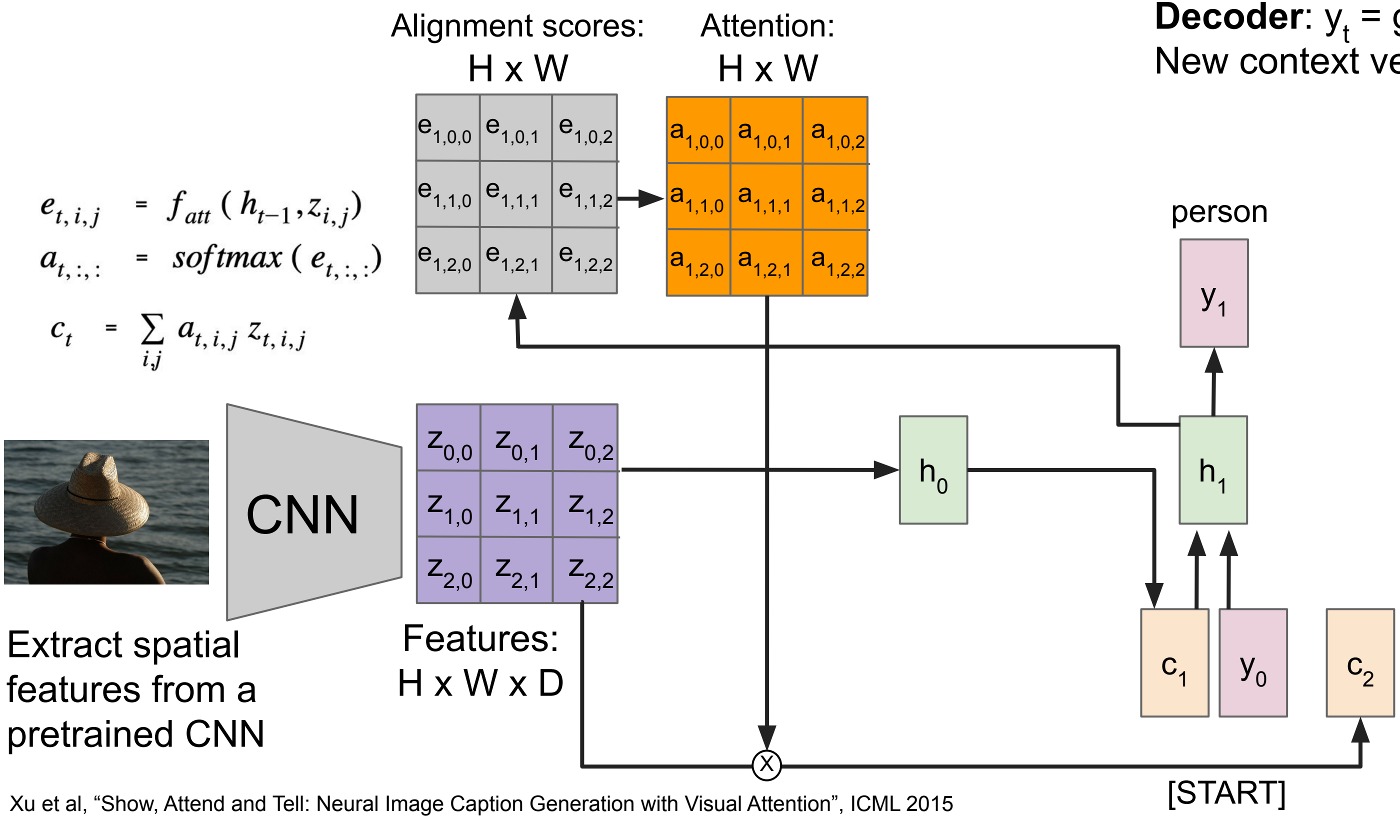

Extract spatial features from a pretrained CNN

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

# Image Captioning with RNNs & Attention

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

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



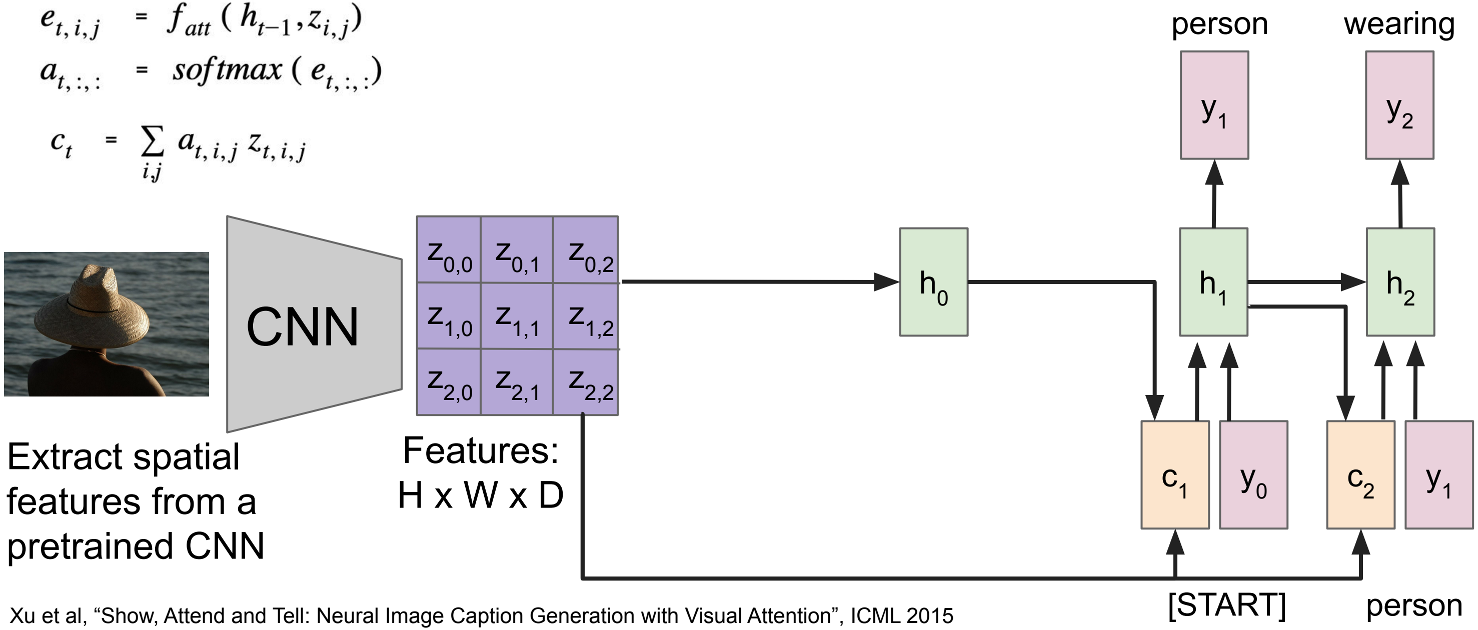

Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

# Image 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

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



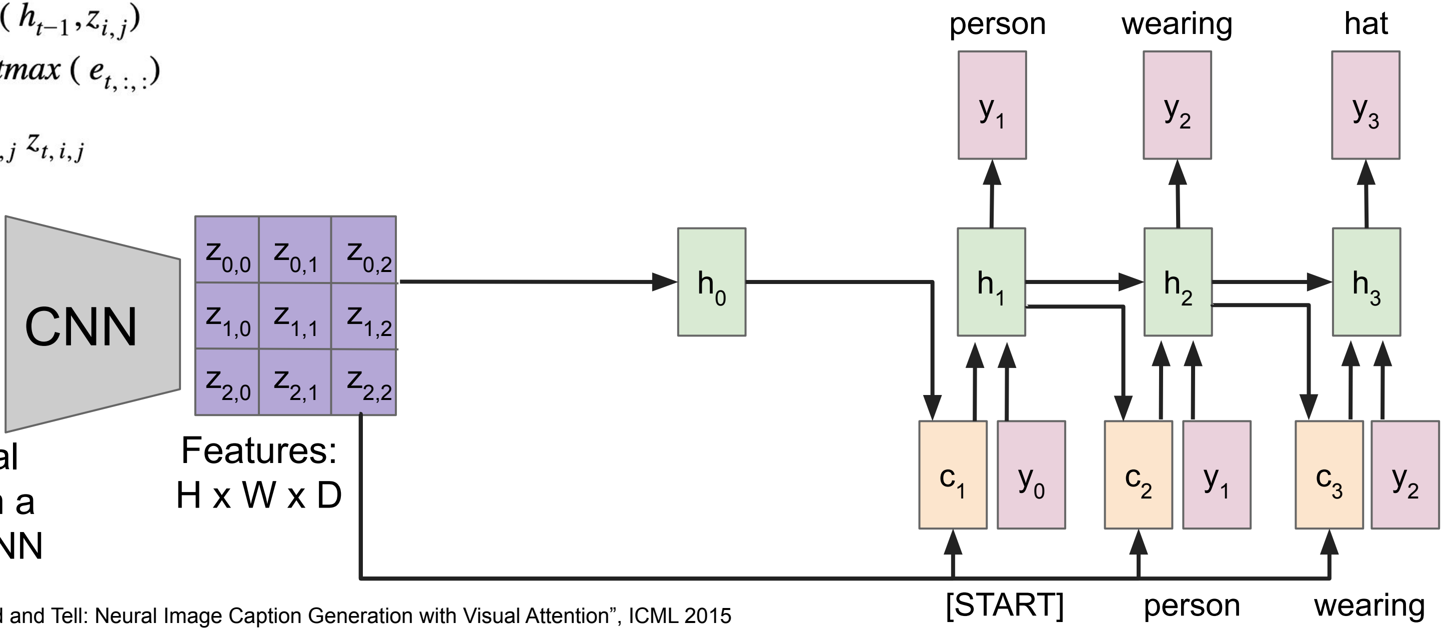

Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

# Image Captioning with RNNs & Attention

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

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

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



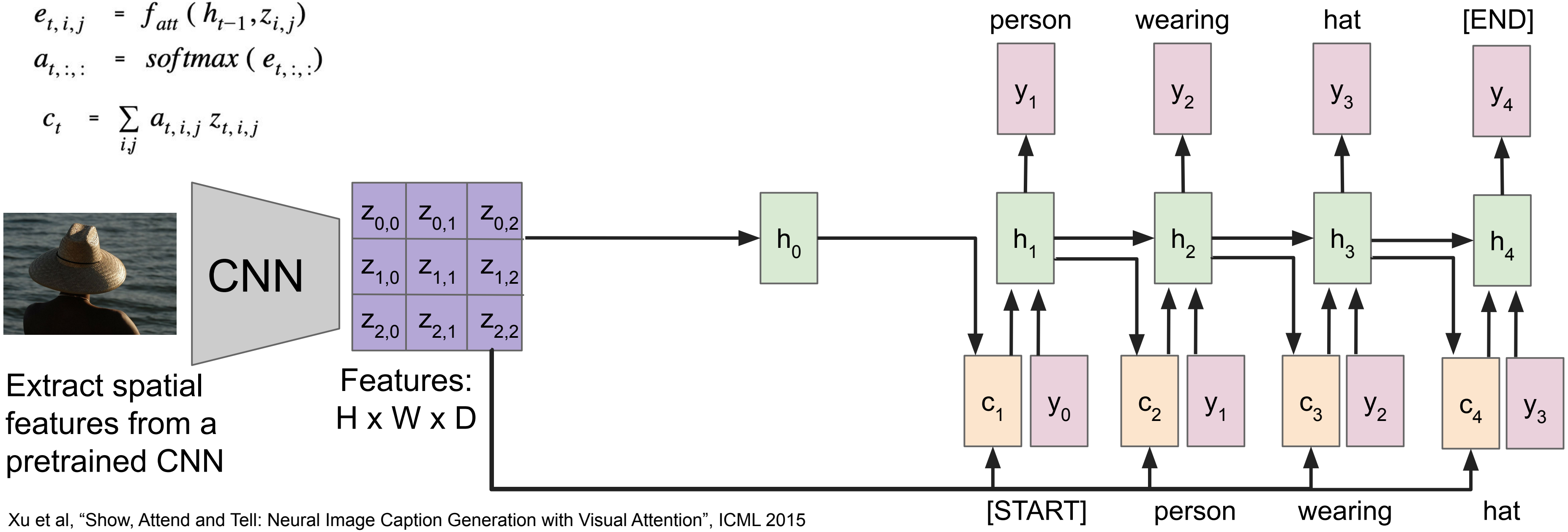

Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

# Image 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

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

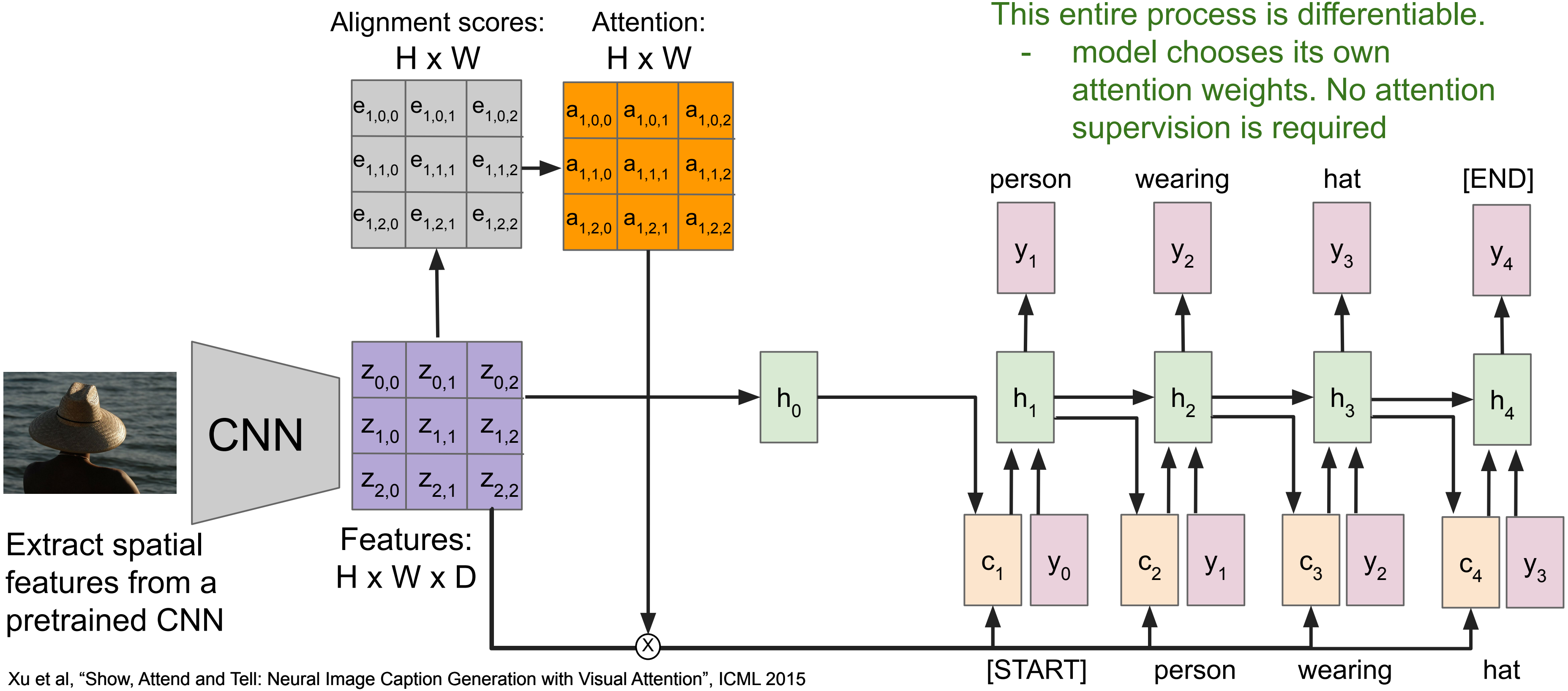



Extract spatial features from a pretrained CNN

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 & Attention




This entire process is differentiable.

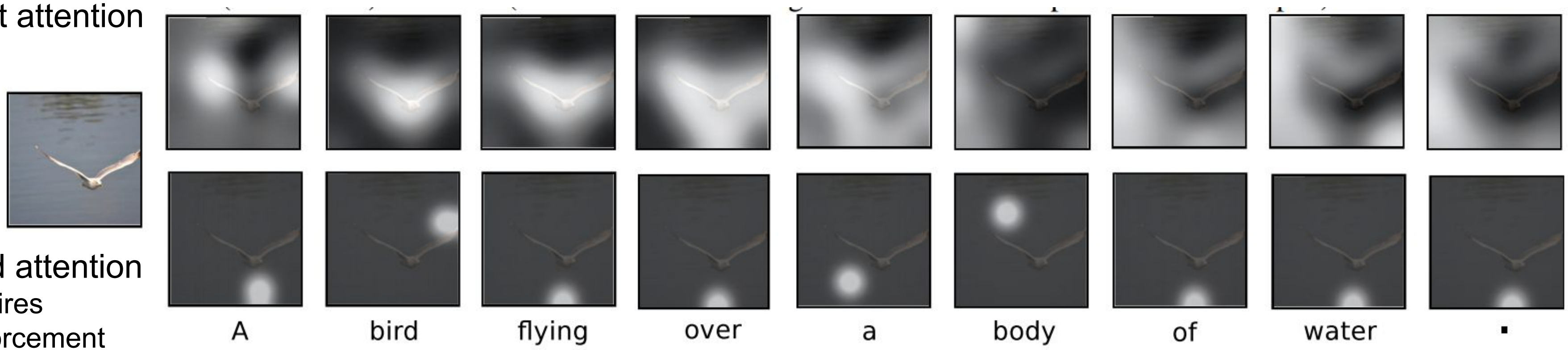
- model chooses its own attention weights. No attention supervision is required

Extract spatial features from a pretrained CNN

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

# Image Captioning with Attention

Soft attention




Hard attention
(requires reinforcement learning)

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

# 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

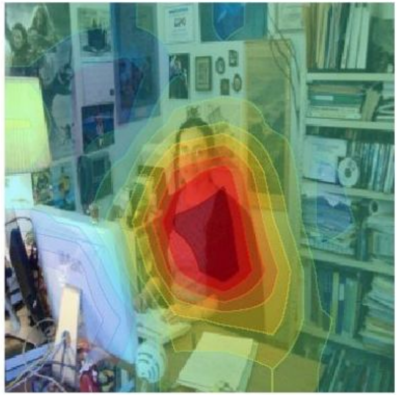
# Attention can detect Gender Bias

All images are [CC0 Public domain](): [dog]().

**Wrong**

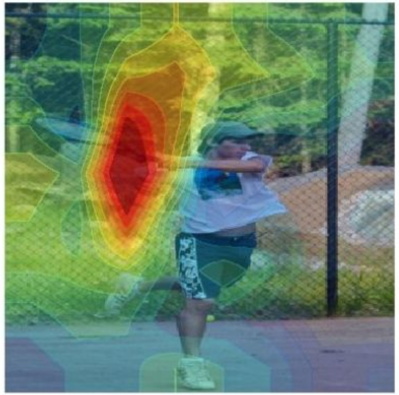
Baseline:
*A **man** sitting at a desk with a laptop computer.*

**Right for the Right Reasons**



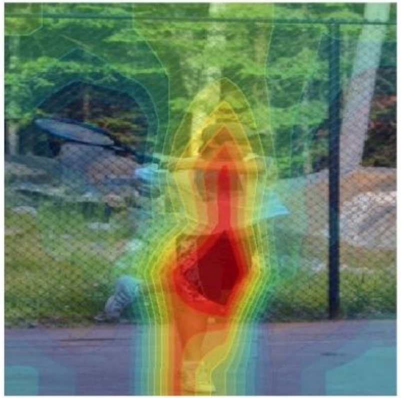
Our Model:
*A **woman** sitting in front of a laptop computer.*

**Right for the Wrong Reasons**



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

**Right for the Right Reasons**

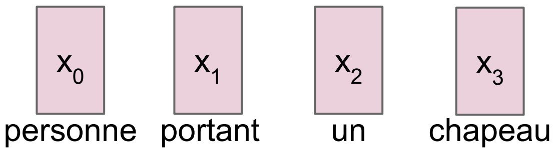


Our Model:
*A **man** holding a tennis racquet on a tennis court.*

Burns et al. "Women also Snowboard: Overcoming Bias in Captioning Models" ECCV 2018

# Similar tasks in NLP - Language translation example

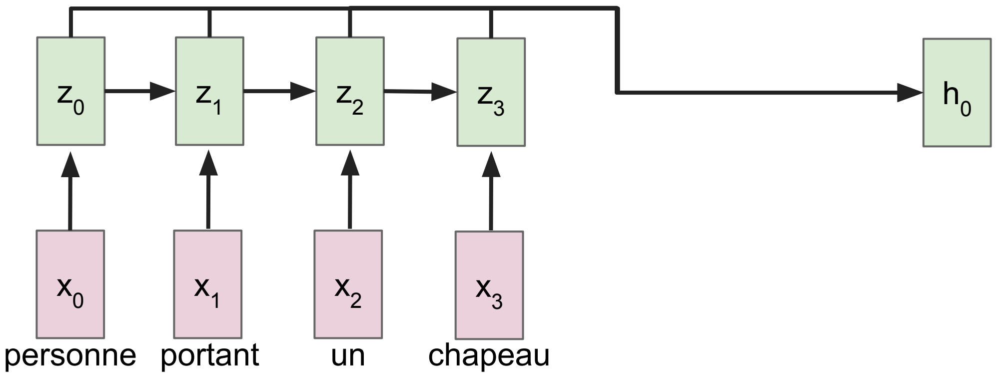
**Input**: Sequence **x** = $x_1, x_2, \ldots, x_T$
**Output:** Sequence **y** = $y_1, y_2, \ldots, y_T$



# Similar tasks in NLP - Language translation example

**Input**: Sequence $\mathbf{x} = x_1, x_2, \ldots, x_T$
**Output:** Sequence $\mathbf{y} = y_1, y_2, \ldots, y_T$

**Encoder**: $h_0 = f_{\mathbf{W}}(\mathbf{z})$
where $z_t = \text{RNN}(x_t, u_{t-1})$
$f_{\mathbf{W}}(.)$ is MLP
u is the hidden RNN state

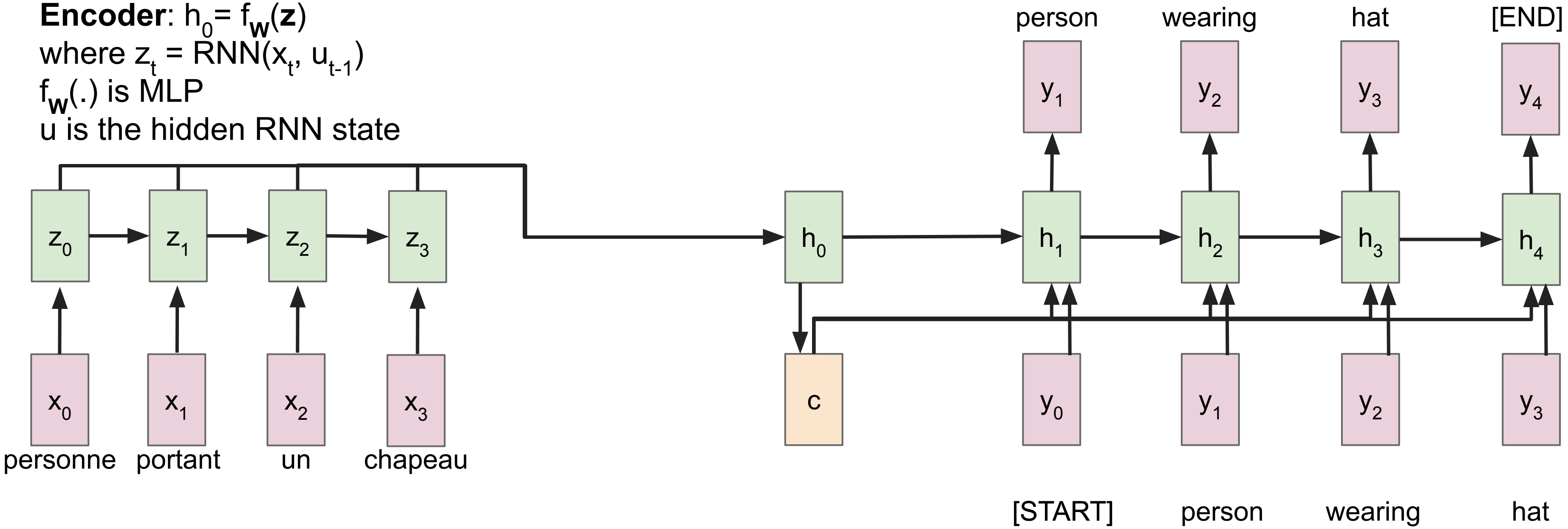


# Similar tasks in NLP - Language translation example

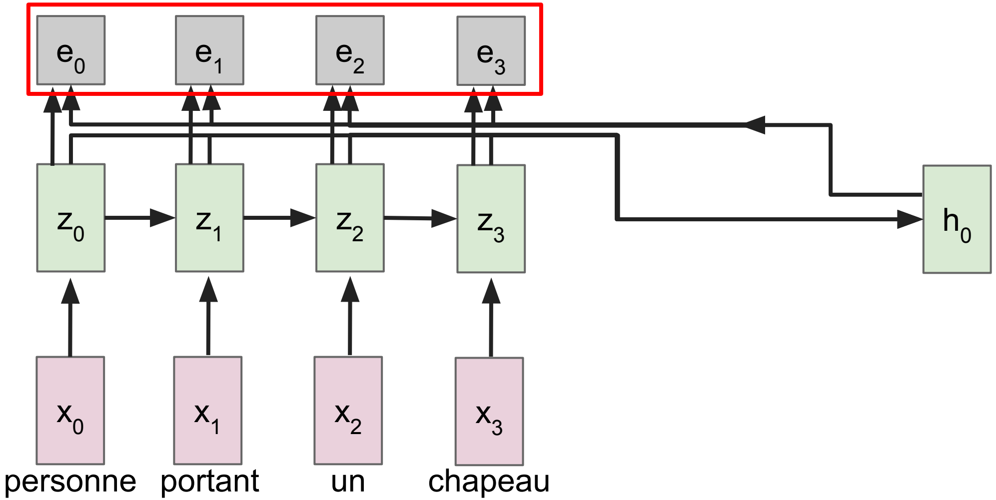
**Input**: Sequence $\mathbf{x} = x_1, x_2, \ldots, x_T$
**Output:** Sequence $\mathbf{y} = y_1, y_2, \ldots, y_T$

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

**Encoder**: $h_0 = f_{\mathbf{W}}(\mathbf{z})$
where $z_t = \mathrm{RNN}(x_t, u_{t-1})$
$f_{\mathbf{W}}(.)$ is MLP
u is the hidden RNN state



# Attention in NLP - Language translation example



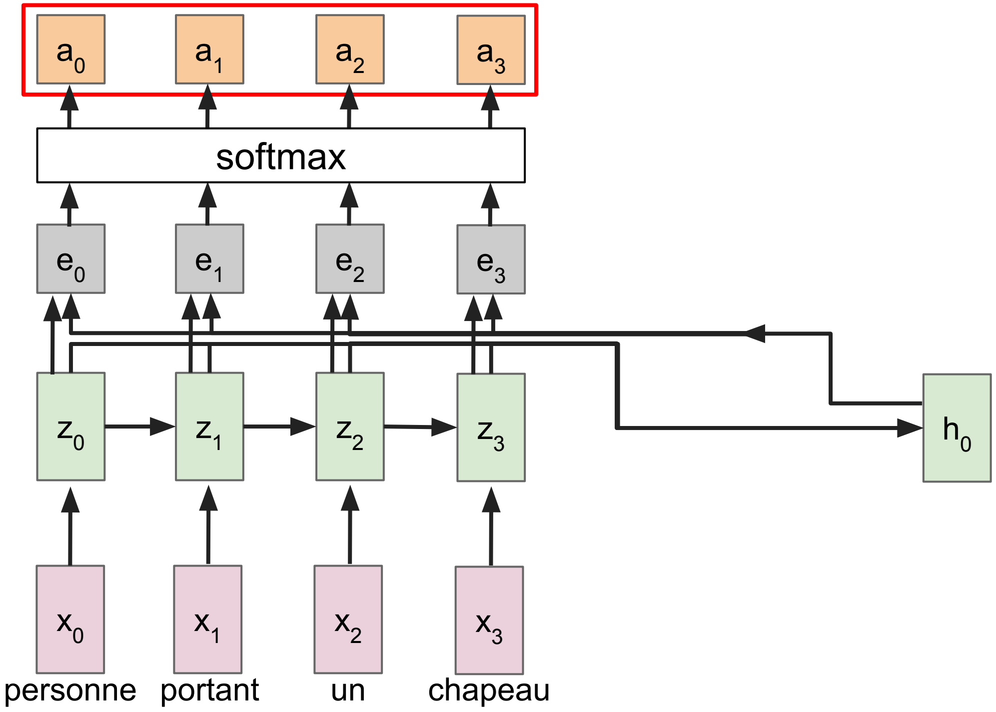

Compute alignments scores (scalars):

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

$f_{att}(.)$ 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):

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

$f_{att}(.)$ is an MLP

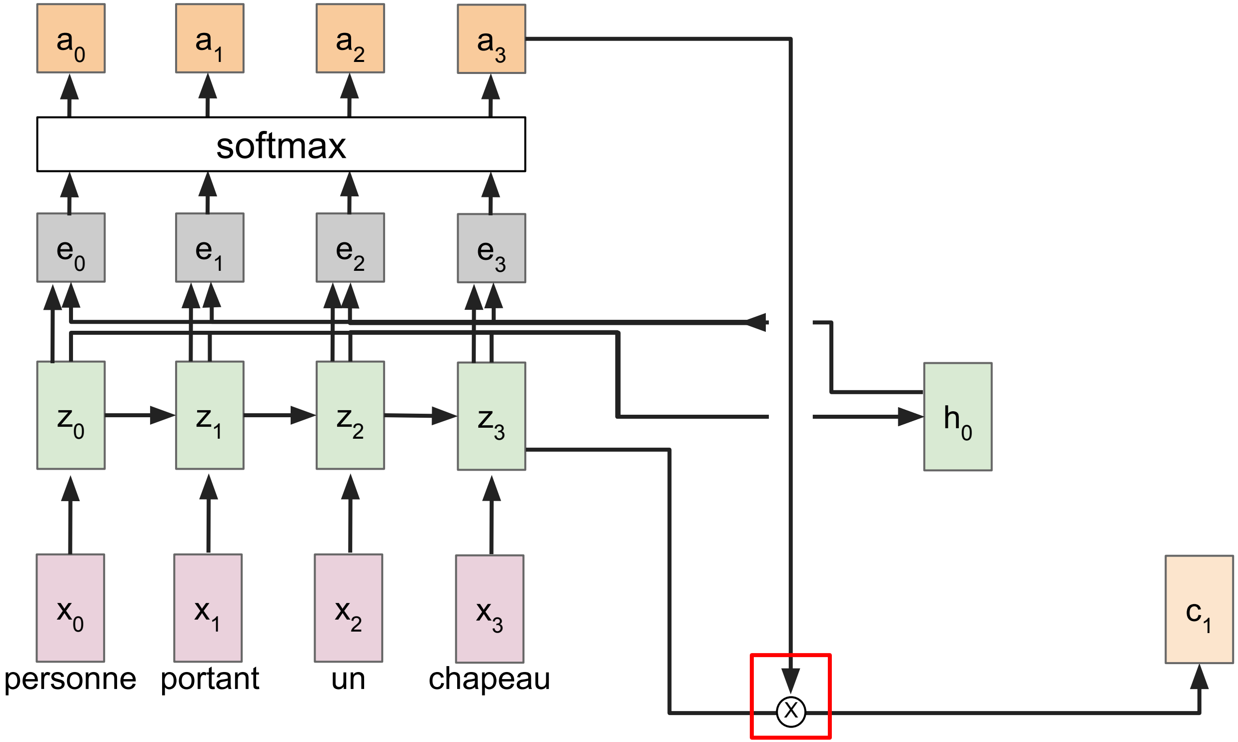
Normalize to get attention weights:

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

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

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):

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

$f_{att}(.)$ is an MLP

Normalize to get attention weights:

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

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

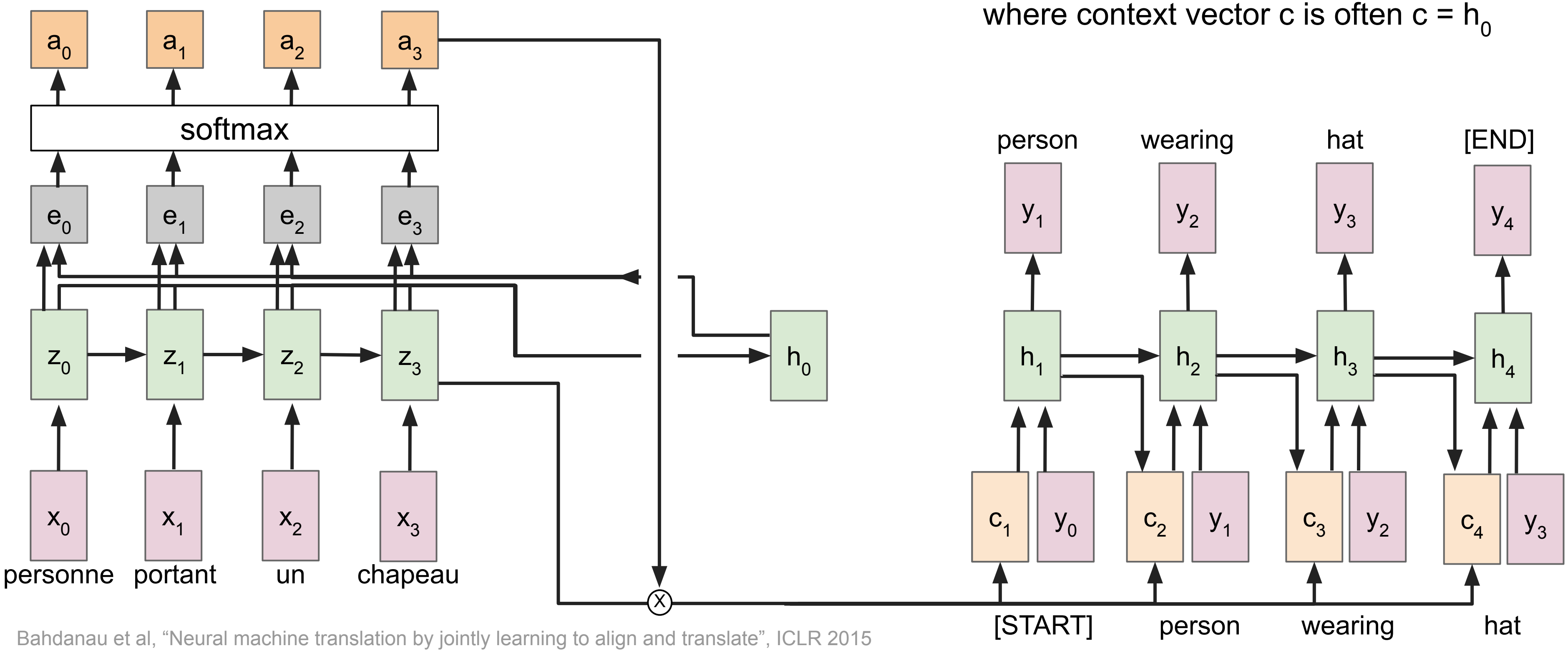
Compute context vector:

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

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

# 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

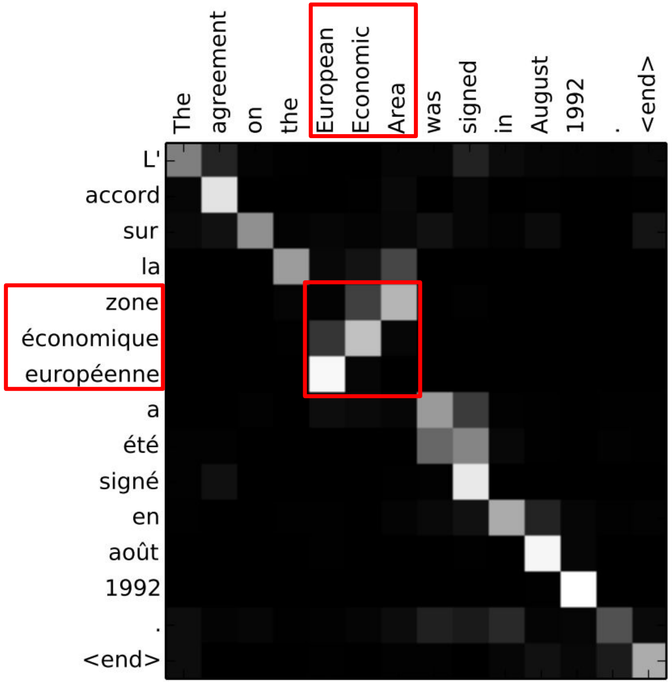
# Similar visualization of attention weights

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

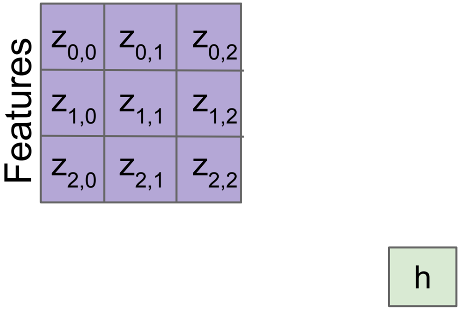



Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

# Today's Agenda:

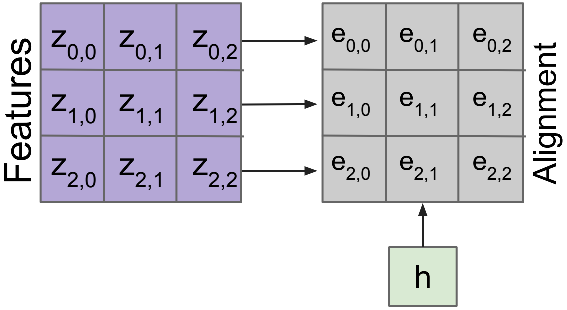
- **Attention with RNNs**
  - In Computer Vision
  - In NLP
- **General Attention Layer**
  - Self-attention
  - Positional encoding
  - Masked attention
  - Multi-head attention
- **Transformers**

# Attention we just saw in image captioning




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

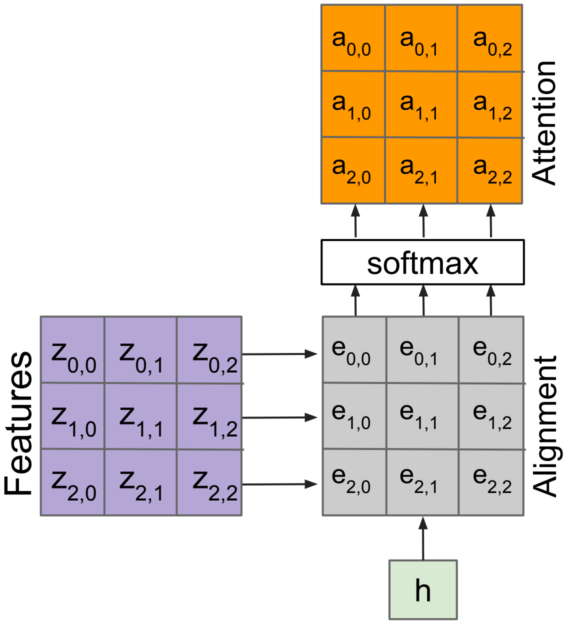
# Attention we just saw in image captioning




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

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

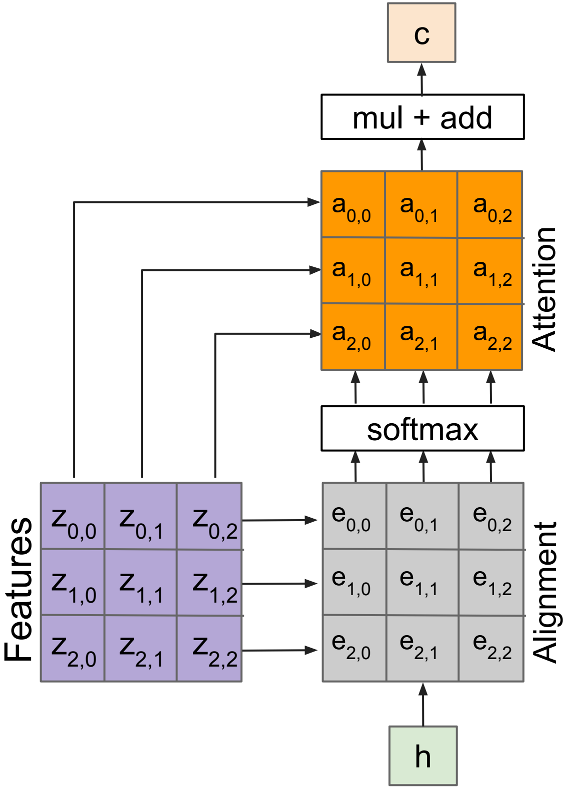
# Attention we just saw in image captioning




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

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

# Attention we just saw in image captioning

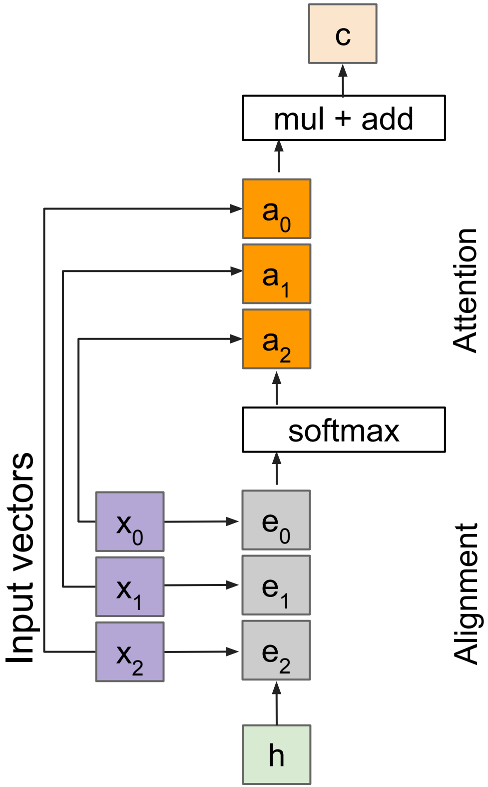



**Outputs:**
context vector: **c** (shape: D)

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

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

# General attention layer



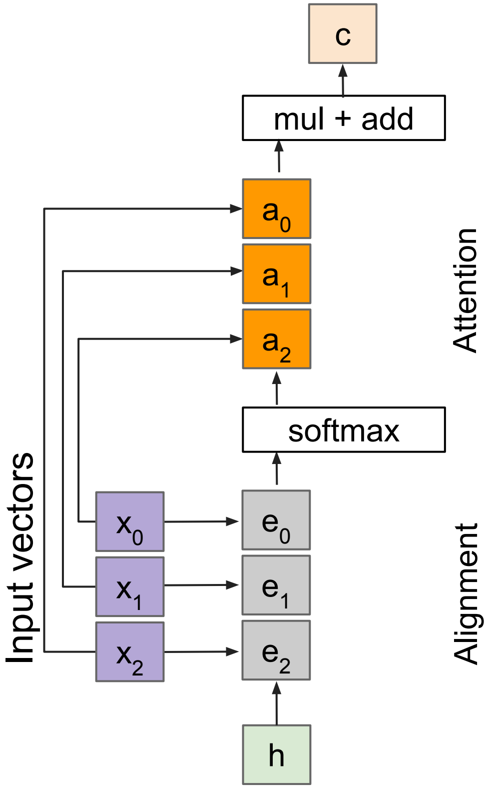

**Outputs:**
context vector: **c** (shape: D)

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

**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 x W = N into N vectors

# General attention layer



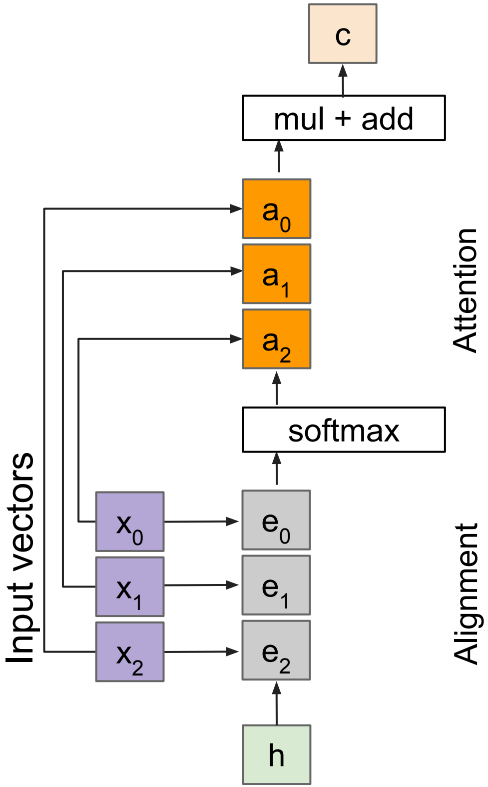

**Outputs:**
context vector: **c** (shape: D)

**Operations:**
Alignment: $e_i = h \cdot x_i$
Attention: **a** = softmax(**e**)
Output: **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: **x** (shape: N x D)
Query: **h** (shape: D)

# General attention layer




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

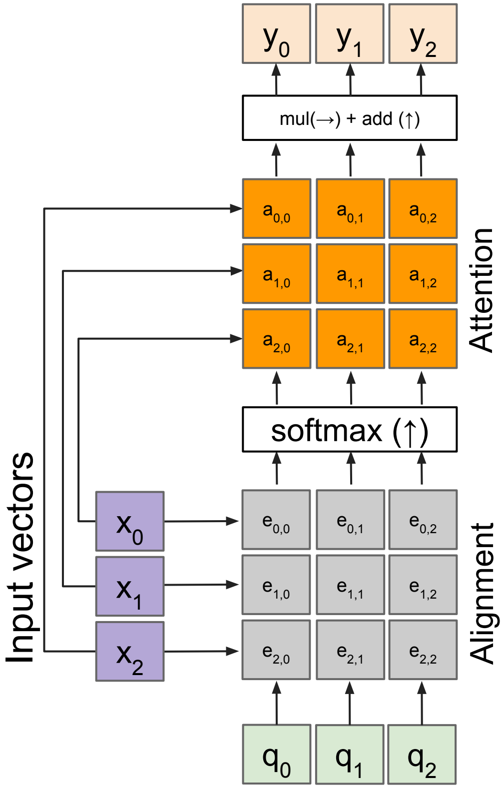
**Operations:**
Alignment: $\boxed{e_i = h \cdot x_i \,/\, \sqrt{D}}$
Attention: **a** = softmax(**e**)
Output: **c** = $\sum_i a_i x_i$

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

Change $f_{att}(.)$ 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 √D to reduce effect of large magnitude vectors

# General attention layer




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

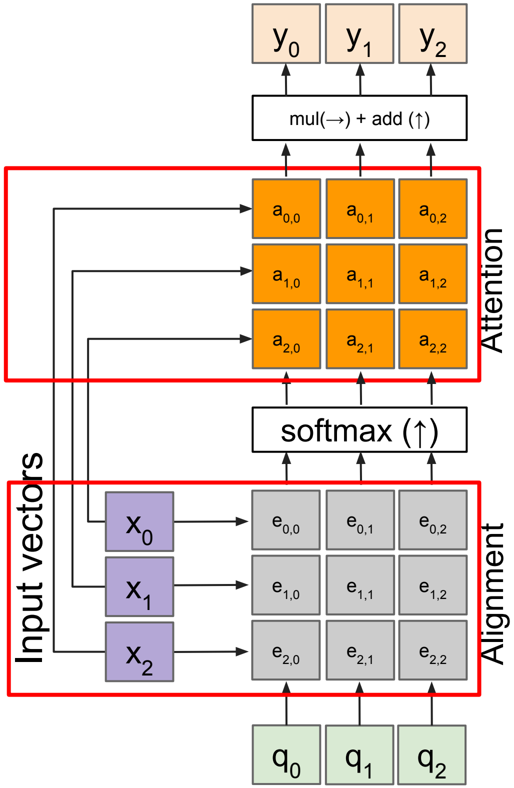
Multiple query vectors
- each query creates a new output context vector

**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: **x** (shape: N x D)
Queries: **q** (shape: M x D)

Multiple query vectors

# General attention layer




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

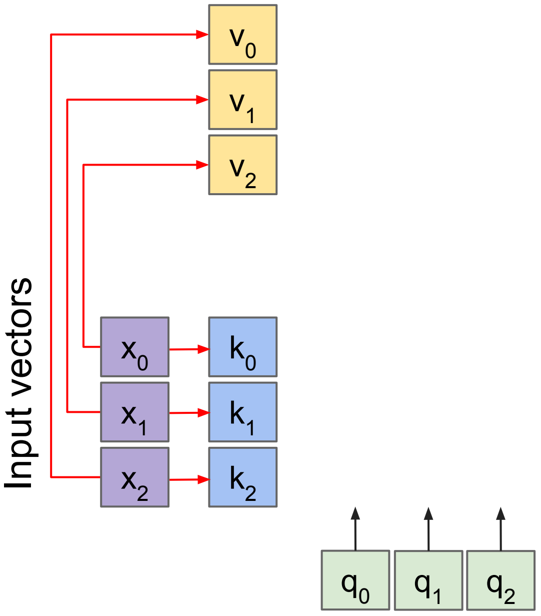
**Operations:**
Alignment: $e_{i,j} = q_j \cdot x_i / \sqrt{D}$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $y_j = \sum_i a_{i,j} x_i$

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

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



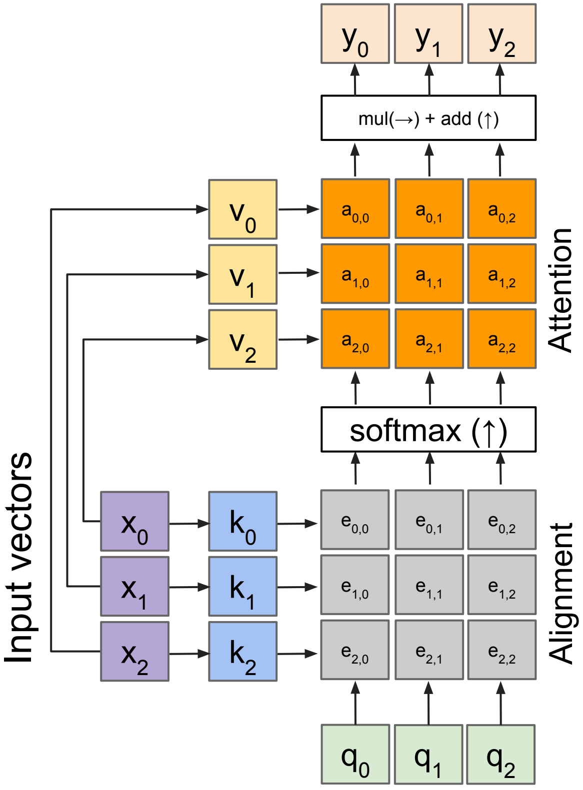

**Operations:**
Key vectors: $\mathbf{k} = \mathbf{x}\mathbf{W}_k$
Value vectors: $\mathbf{v} = \mathbf{x}\mathbf{W}_v$

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

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: **y** (shape: $\boxed{D_v}$)

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

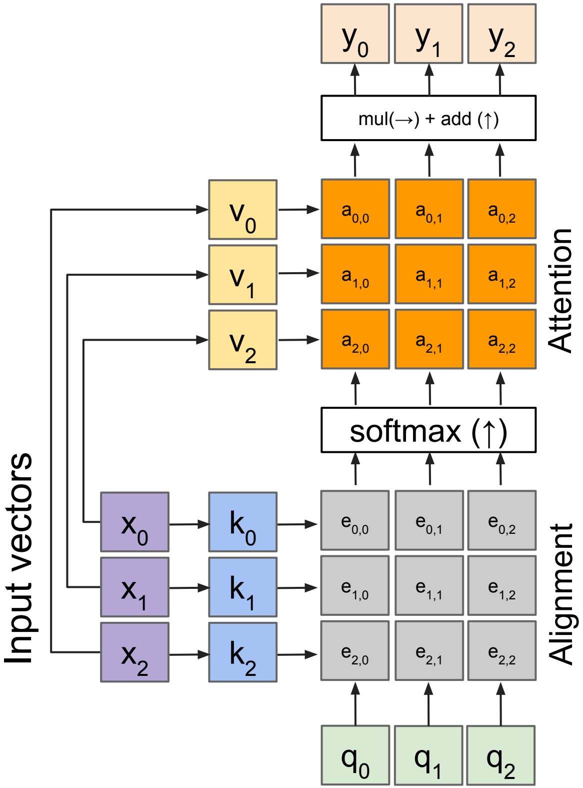
Notice that the input vectors are used for both the alignment as well as the attention calculations.

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

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

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

# General attention layer



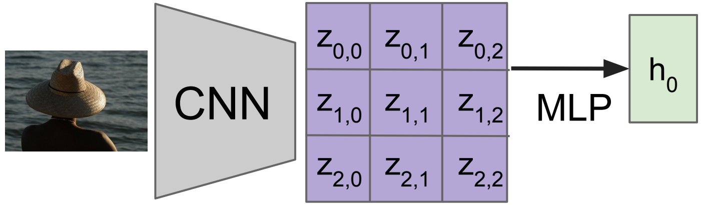

**Outputs:**
context vectors: **y** (shape: $D_v$)

**Operations:**
Key vectors: **k** = **xW**$_k$
Value vectors: **v** = **xW**$_v$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: **a** = softmax(**e**)
Output: $y_j = \sum_i a_{i,j} v_i$

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

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

**Encoder**: $h_0 = f_W(\mathbf{z})$
where **z** is spatial CNN features
$f_W(.)$ is an MLP



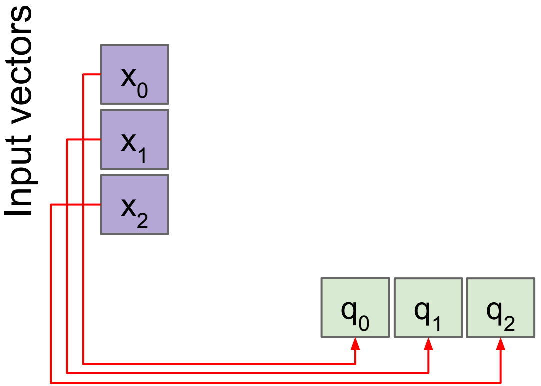
# Self attention layer

We can calculate the query vectors from the input vectors, therefore, defining a "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: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $y_j = \sum_i a_{i,j} v_i$

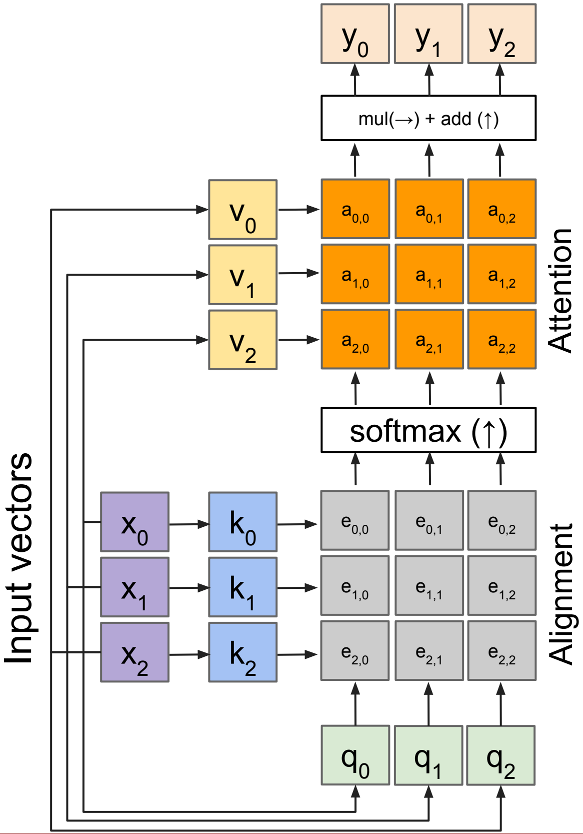
Instead, query vectors are calculated using a FC layer.

No input query vectors anymore




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

# Self attention layer

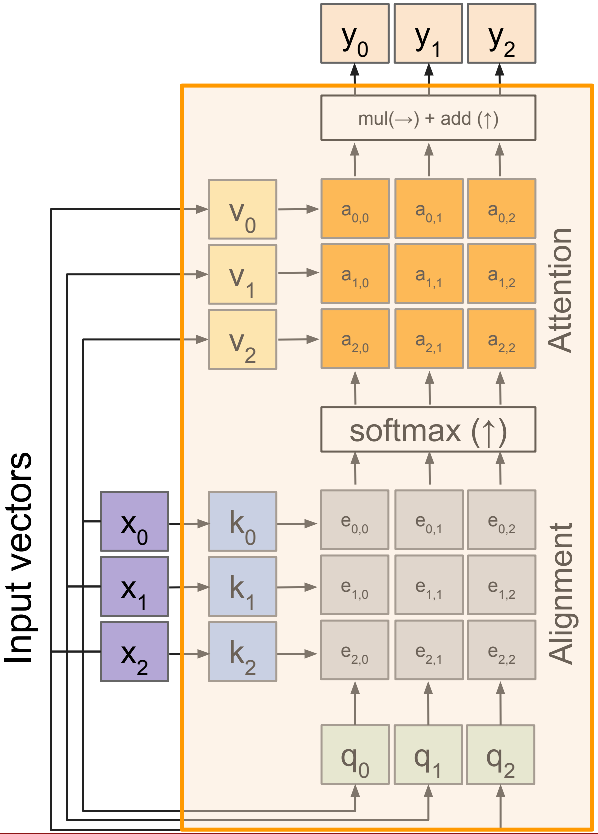



**Outputs:**
context vectors: **y** (shape: $D_v$)

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

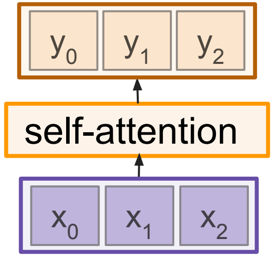
**Inputs**:
Input vectors: **x** (shape: N x D)

# Self attention layer - attends over sets of inputs




**Outputs:**
context vectors: **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: $y_j = \sum_i a_{i,j} v_i$

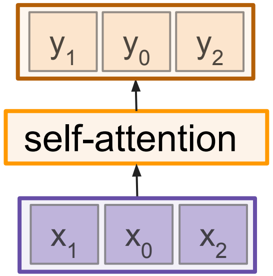


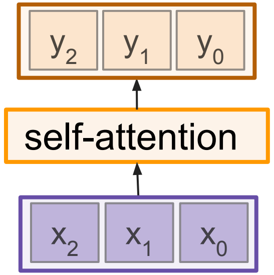

**Inputs**:
Input vectors: **x** (shape: N x D)

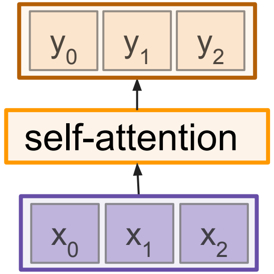
# Self attention layer - attends over sets of inputs

Permutation invariant

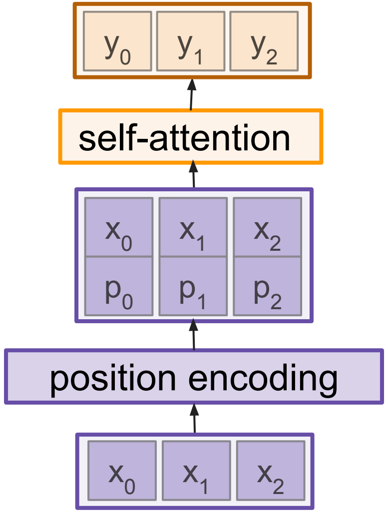
**Problem:** how can we encode ordered sequences like language or spatially ordered image features?



# Positional encoding




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

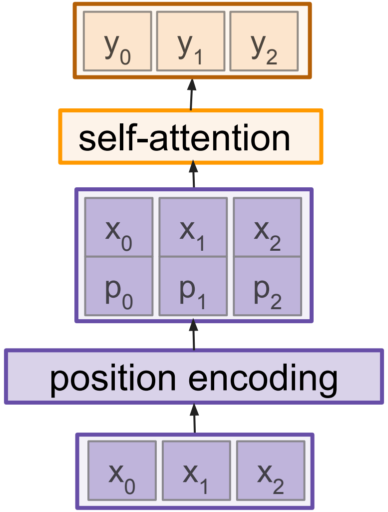
We use a function *pos*: $N \rightarrow R^d$ to process the position j of the vector into a d-dimensional vector

So, $p_j = pos(j)$

Desiderata of *pos*(.) :

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

# Positional encoding




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

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

So, $p_j = pos(j)$

Options for *pos*(.)

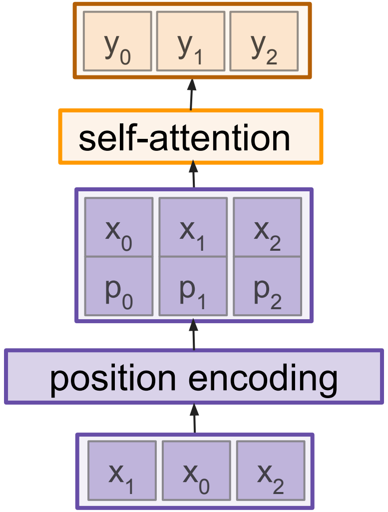
1. Learn a lookup table:
    - Learn parameters to use for *pos*(t) for $t \in [0, T)$
    - Lookup table contains T x d parameters.

Desiderata of *pos*(.) :

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

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*: $N \rightarrow R^d$ to process the position $j$ of the vector into a d-dimensional vector

So, $p_j$ = *pos*(j)

Options for *pos*(.)

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

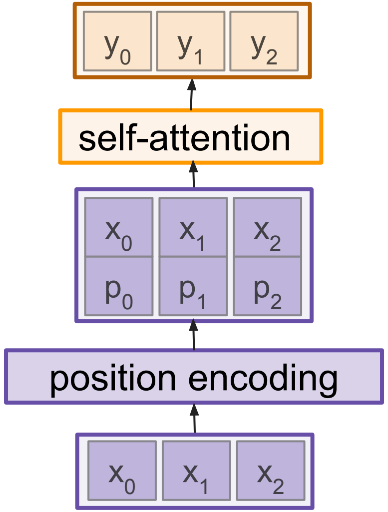
2. Design a fixed function with the desiderata
   -

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

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

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*: $N \rightarrow R^d$ to process the position $j$ of the vector into a d-dimensional vector

So, $p_j = pos(j)$

Options for *pos*(.)

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

2. Design a fixed function with the desiderata
   - 

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

Intuition:

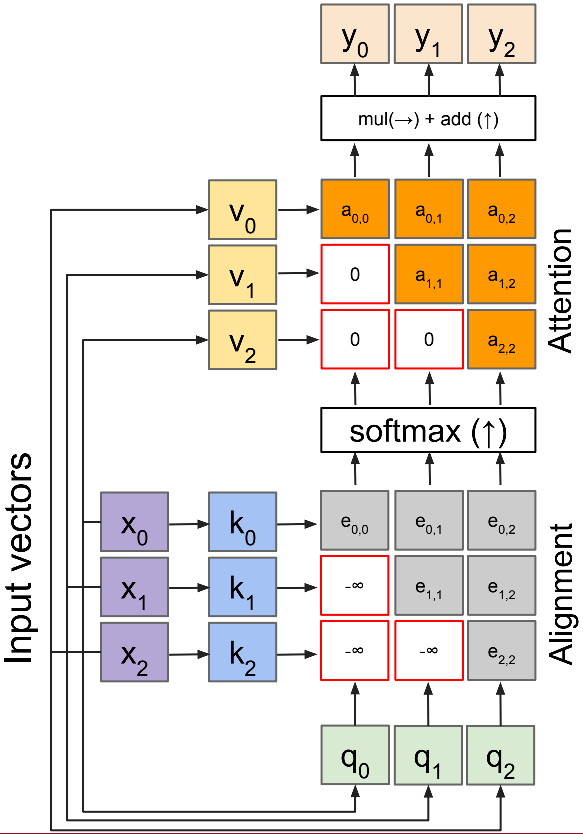
| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 : | 0 | 0 | 0 | 0 | 8 : | 1 | 0 | 0 | 0 |
| 1 : | 0 | 0 | 0 | 1 | 9 : | 1 | 0 | 0 | 1 |
| 2 : | 0 | 0 | 1 | 0 | 10 : | 1 | 0 | 1 | 0 |
| 3 : | 0 | 0 | 1 | 1 | 11 : | 1 | 0 | 1 | 1 |
| 4 : | 0 | 1 | 0 | 0 | 12 : | 1 | 1 | 0 | 0 |
| 5 : | 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: **y** (shape: $D_v$)

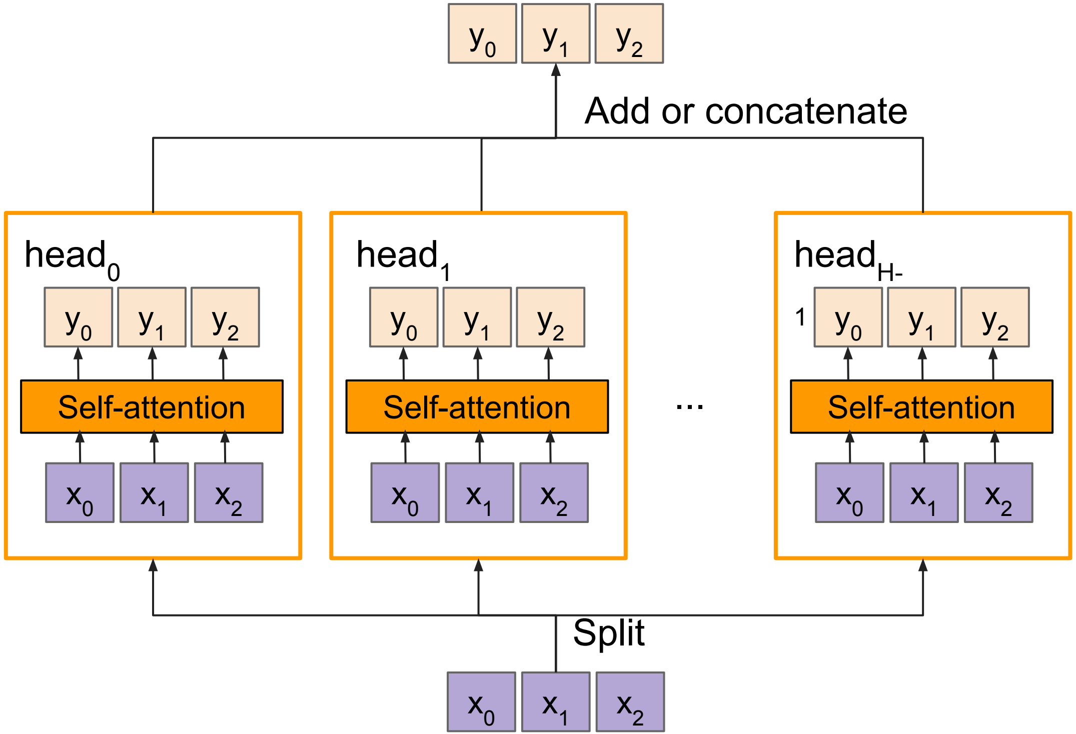
**Operations:**
Key vectors: $\mathbf{k} = \mathbf{x}\mathbf{W}_k$
Value vectors: $\mathbf{v} = \mathbf{x}\mathbf{W}_v$
Query vectors: $\mathbf{q} = \mathbf{x}\mathbf{W}_q$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $y_j = \sum_i a_{i,j}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

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

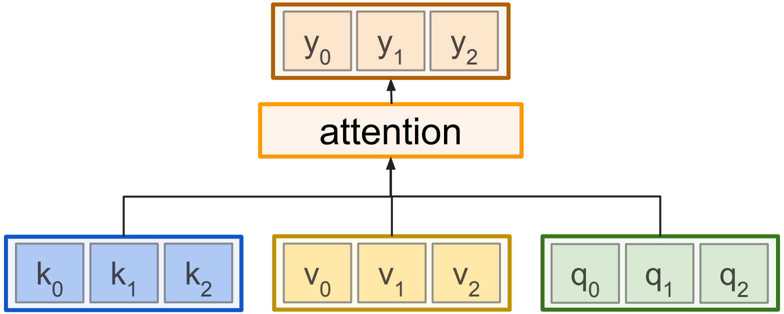
- Prevent vectors from looking at future vectors.
- Manually set alignment scores to -infinity

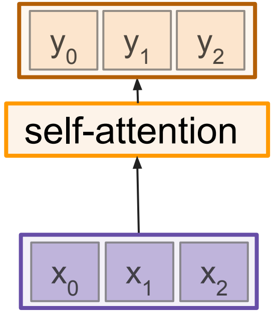
# Multi-head self attention layer

- Multiple self-attention heads in parallel



# General attention versus self-attention



# Comparing RNNs to Transformers

**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.

**Transformers:**

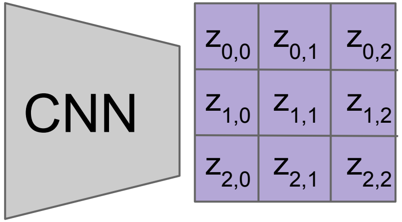
(+) Good at long sequences. Each attention calculation looks at all inputs.
(+) Can operate over unordered sets or ordered sequences with positional encodings.
(+) Parallel computation: All alignment and attention scores for all inputs can be done in parallel.
(-) Requires a lot of memory: N x M alignment and attention scalers 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 $\mathbf{y} = y_1, y_2, \ldots, y_T$



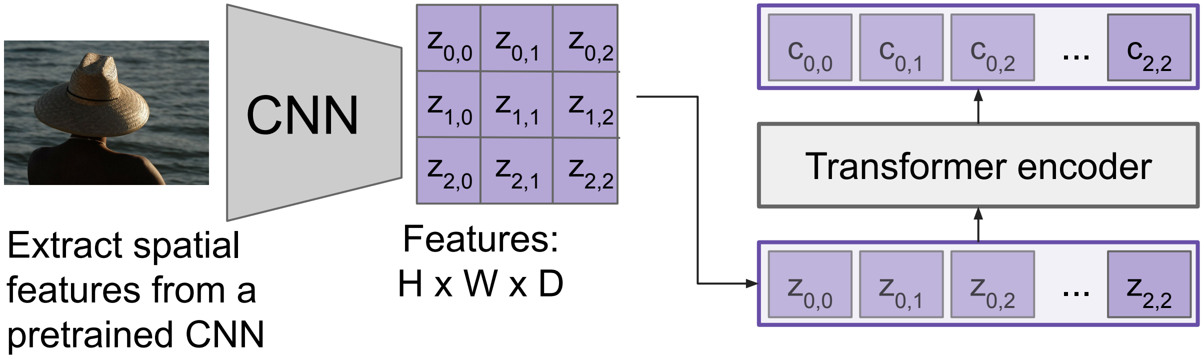

Extract spatial features from a pretrained CNN

Features: H x W x D

# Image Captioning using transformers

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

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

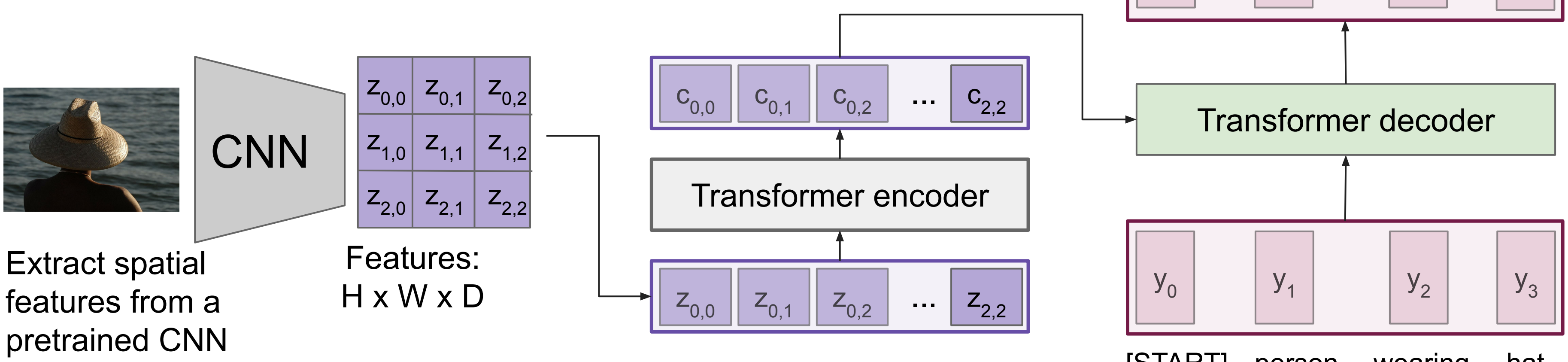


# Image Captioning using transformers

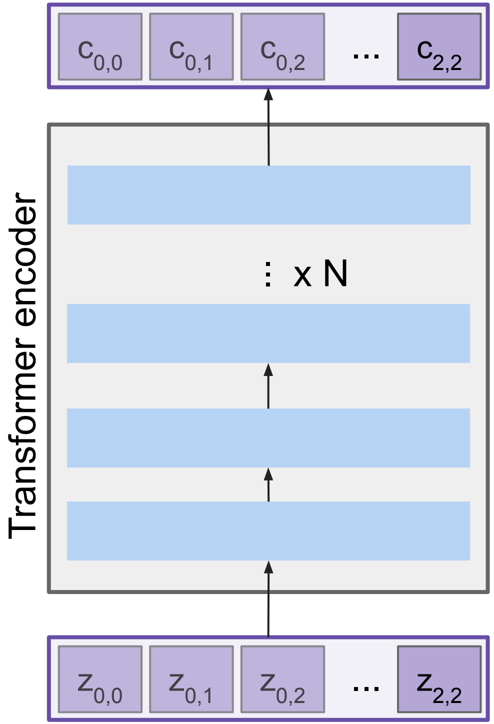
**Input**: Image **I**
**Output:** Sequence $\mathbf{y} = y_1, y_2, \ldots, y_T$

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

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



# The Transformer encoder block

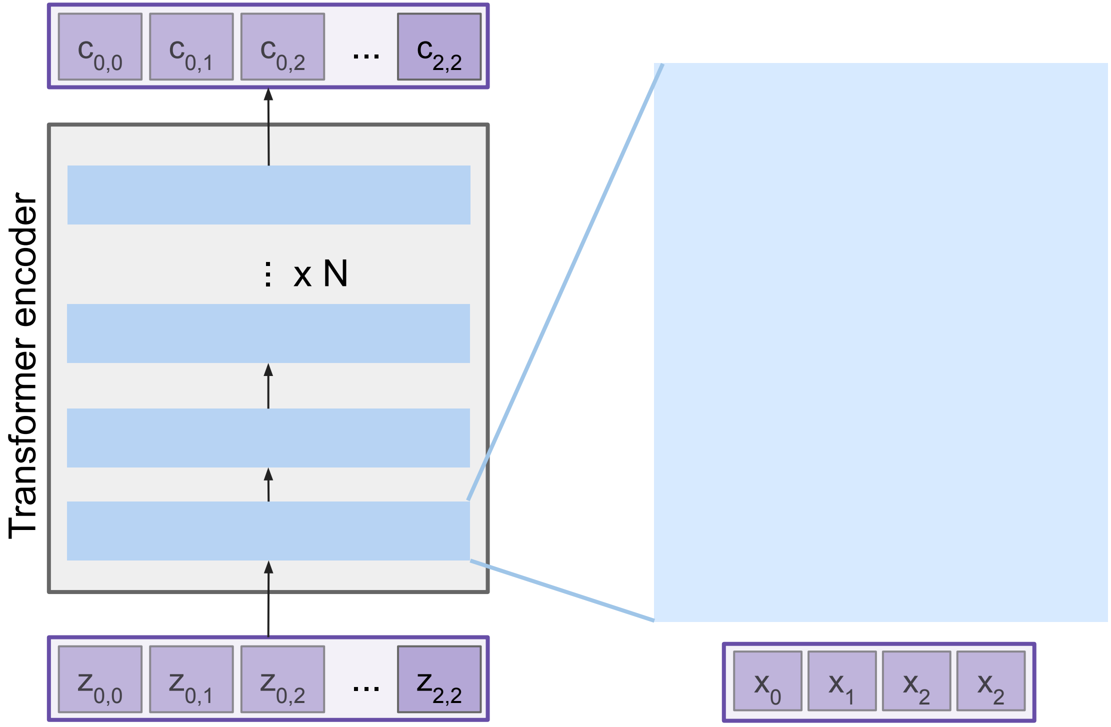



Made up of N encoder blocks.

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

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

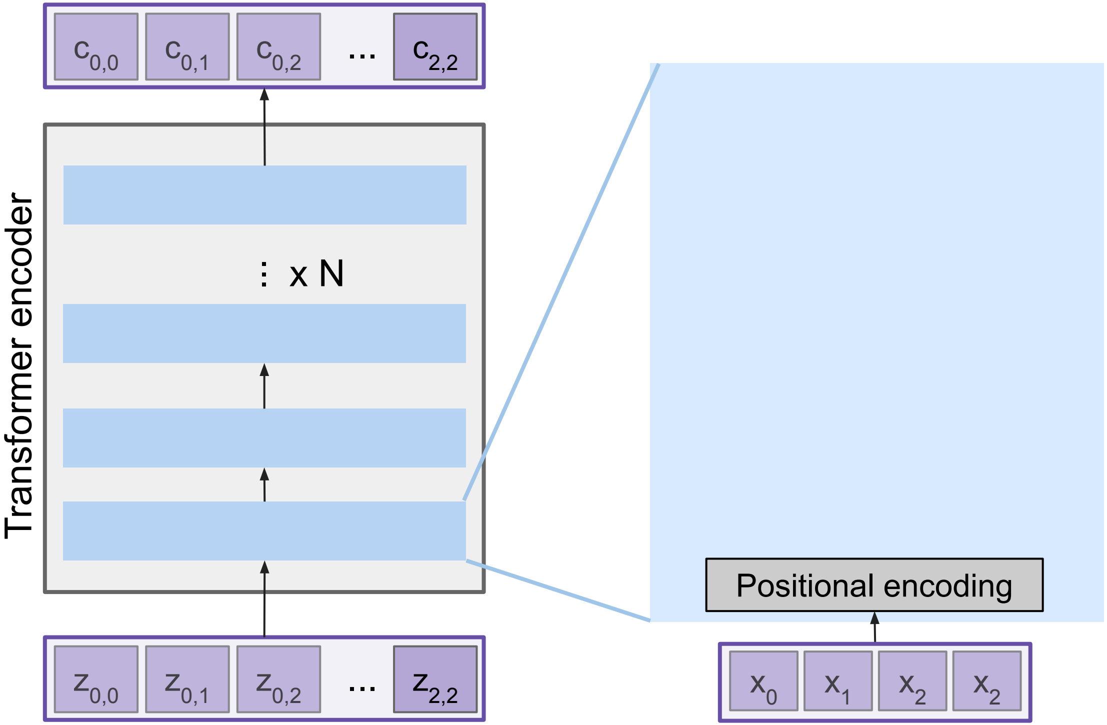
# The Transformer encoder block




Let's dive into one encoder block

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

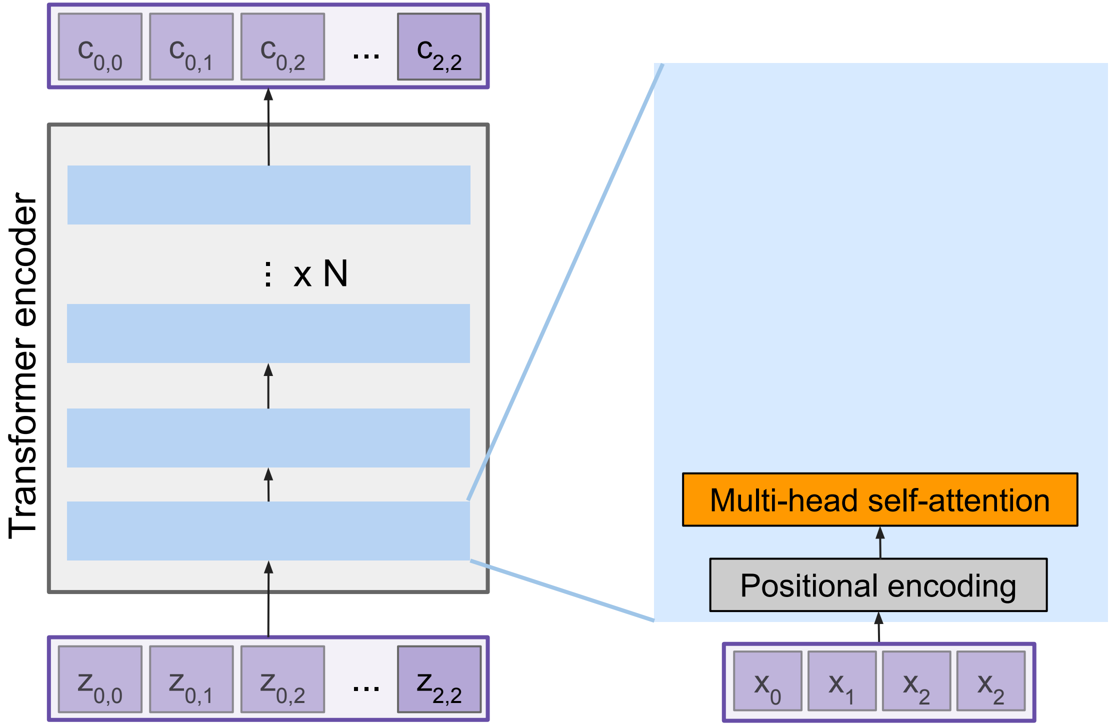
# The Transformer encoder block




Add positional encoding

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

# The Transformer encoder block

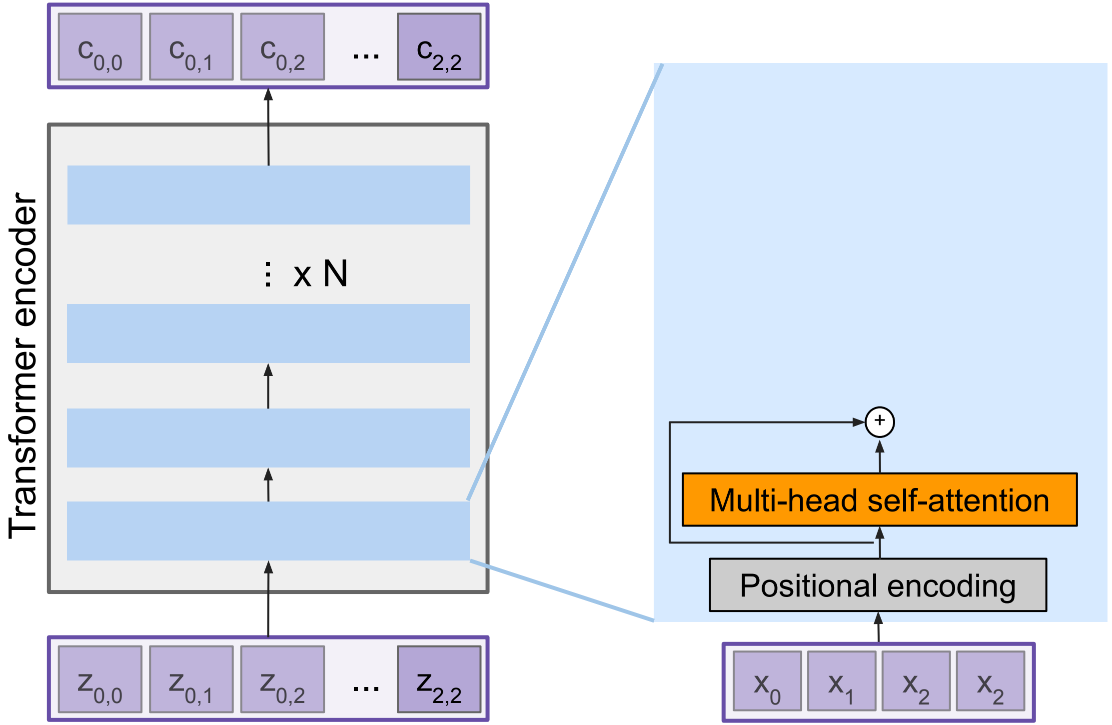



Attention attends over all the vectors

Add positional encoding

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

# The Transformer encoder block



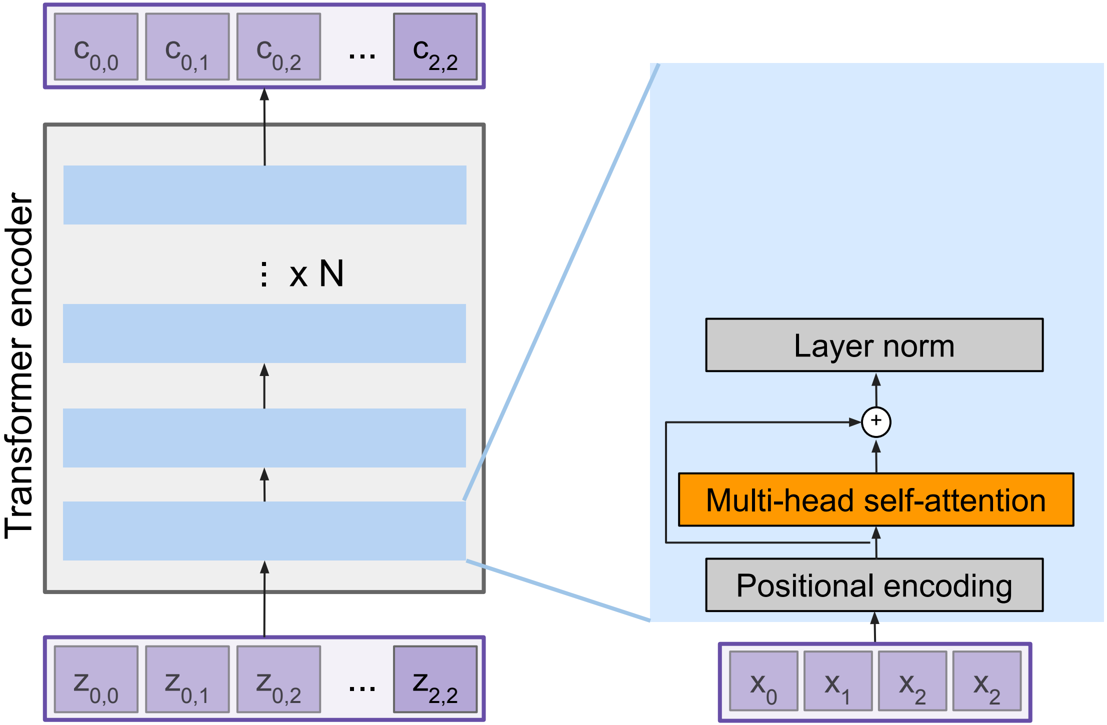

Residual connection

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

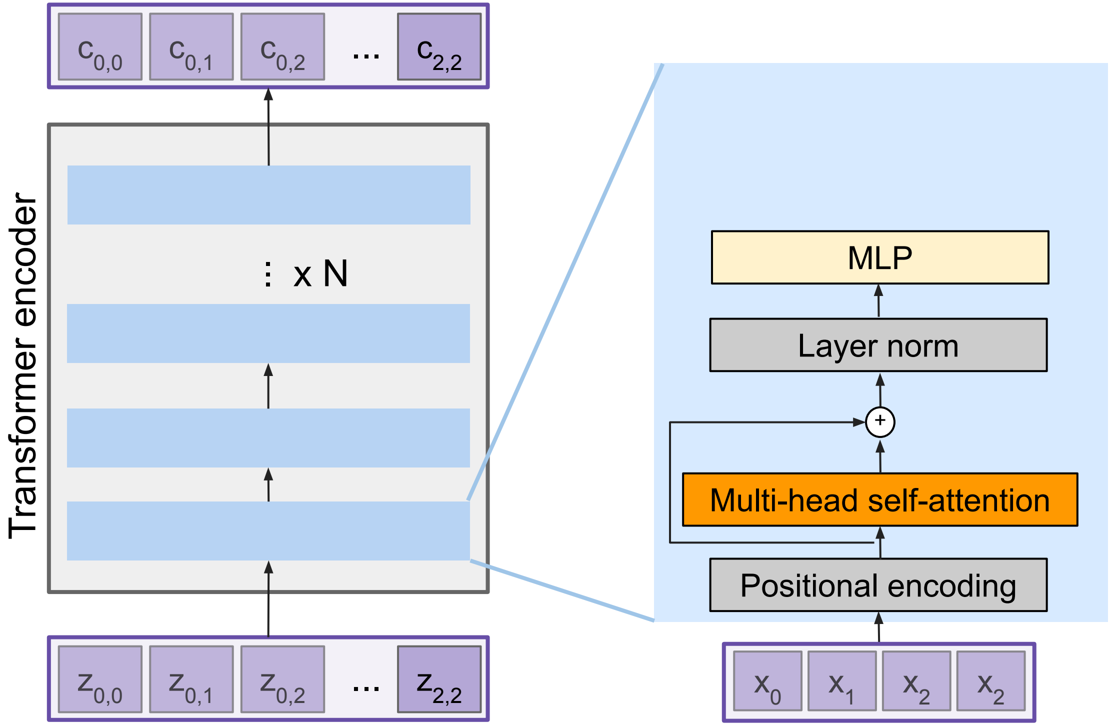
Residual connection

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

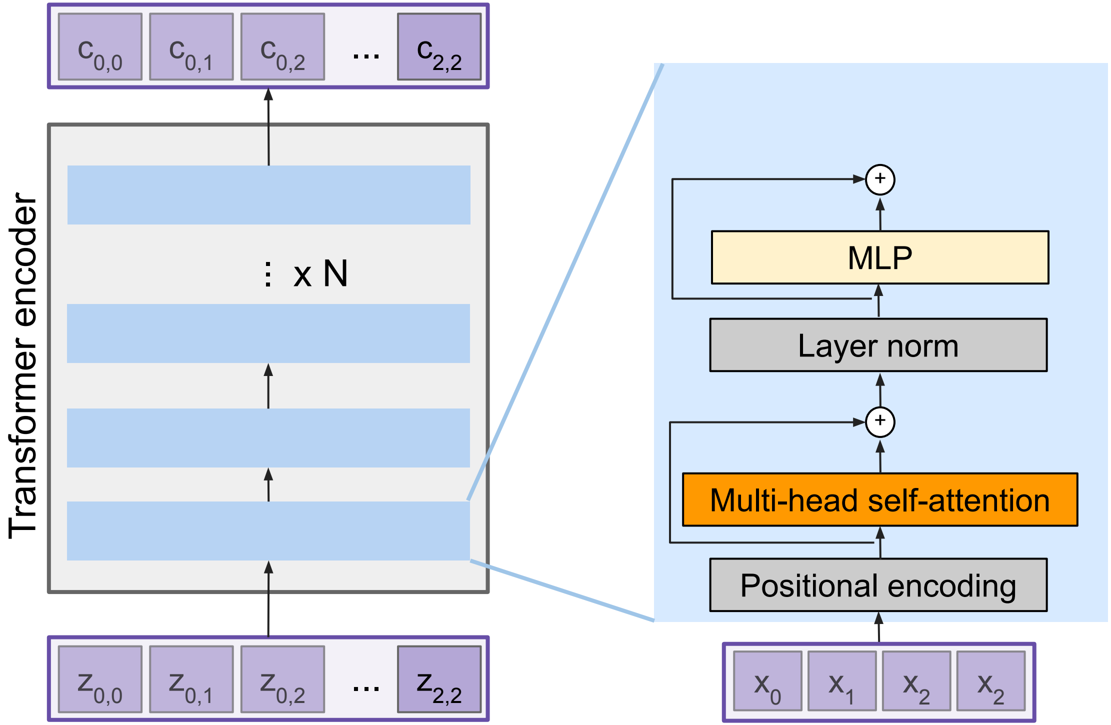
Residual connection

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

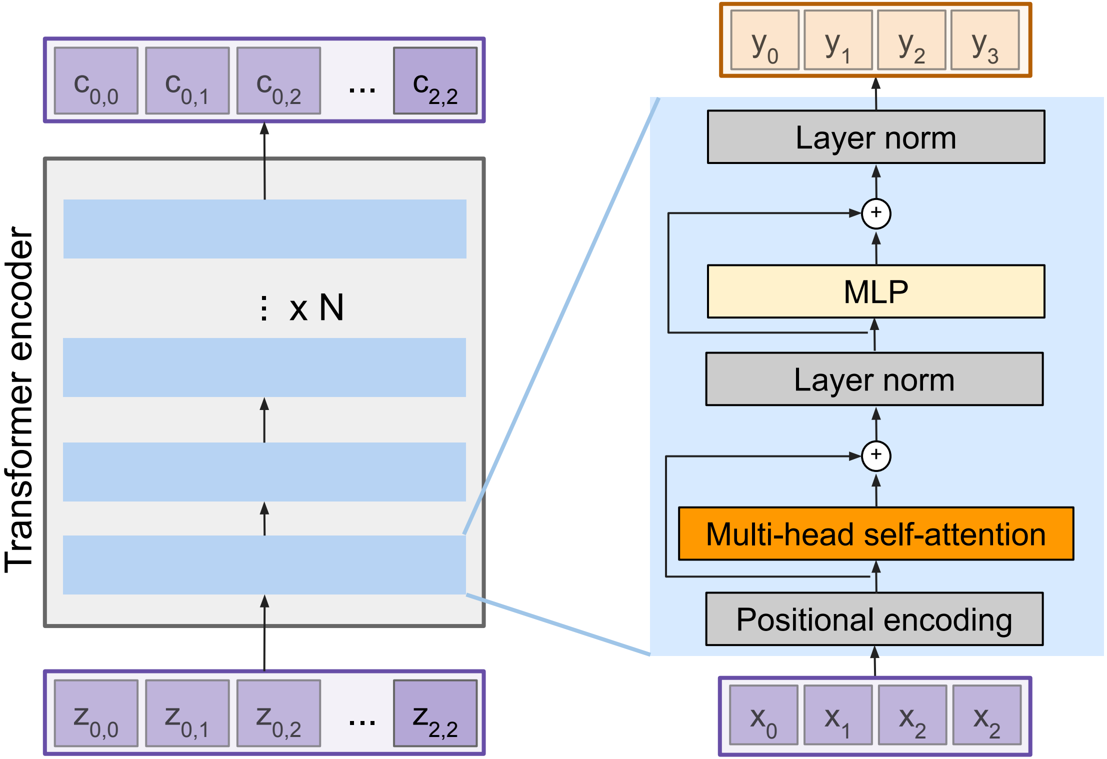
Residual connection

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

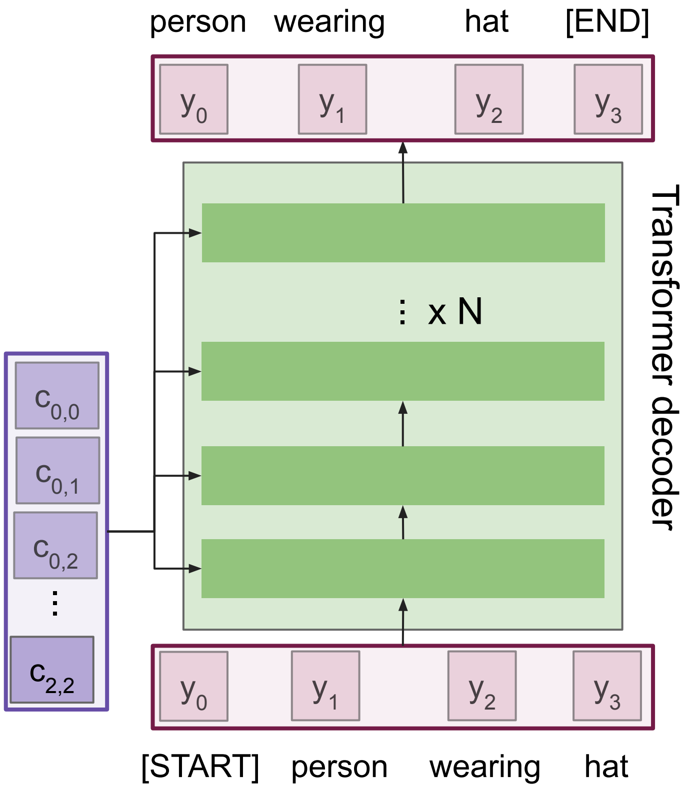
Self-attention is the only interaction between vectors.

Layer norm and MLP operate independently per vector.

Highly scalable, highly parallelizable, but high memory usage.

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

# The Transformer Decoder block

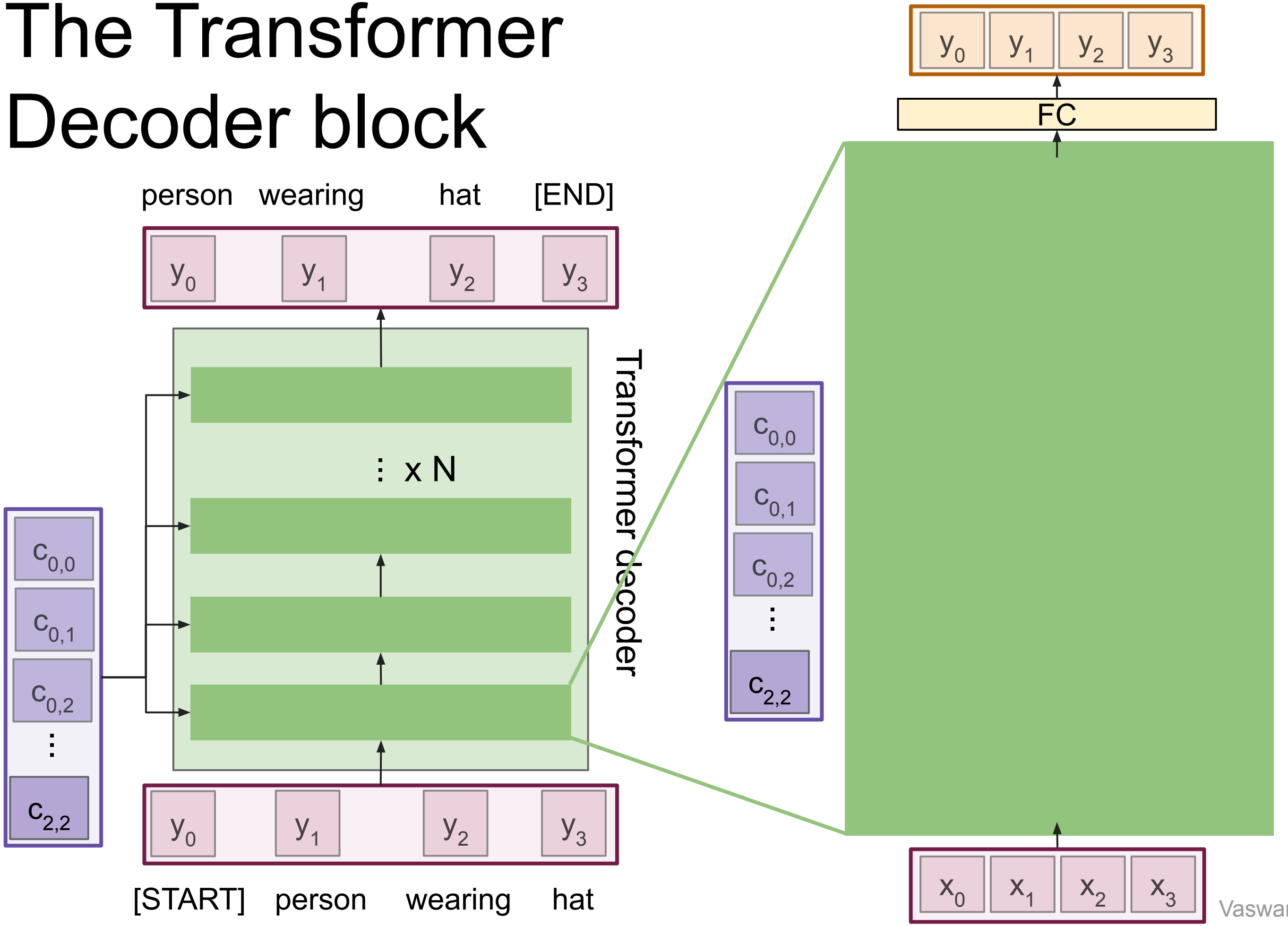



Made up of N decoder blocks.

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

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

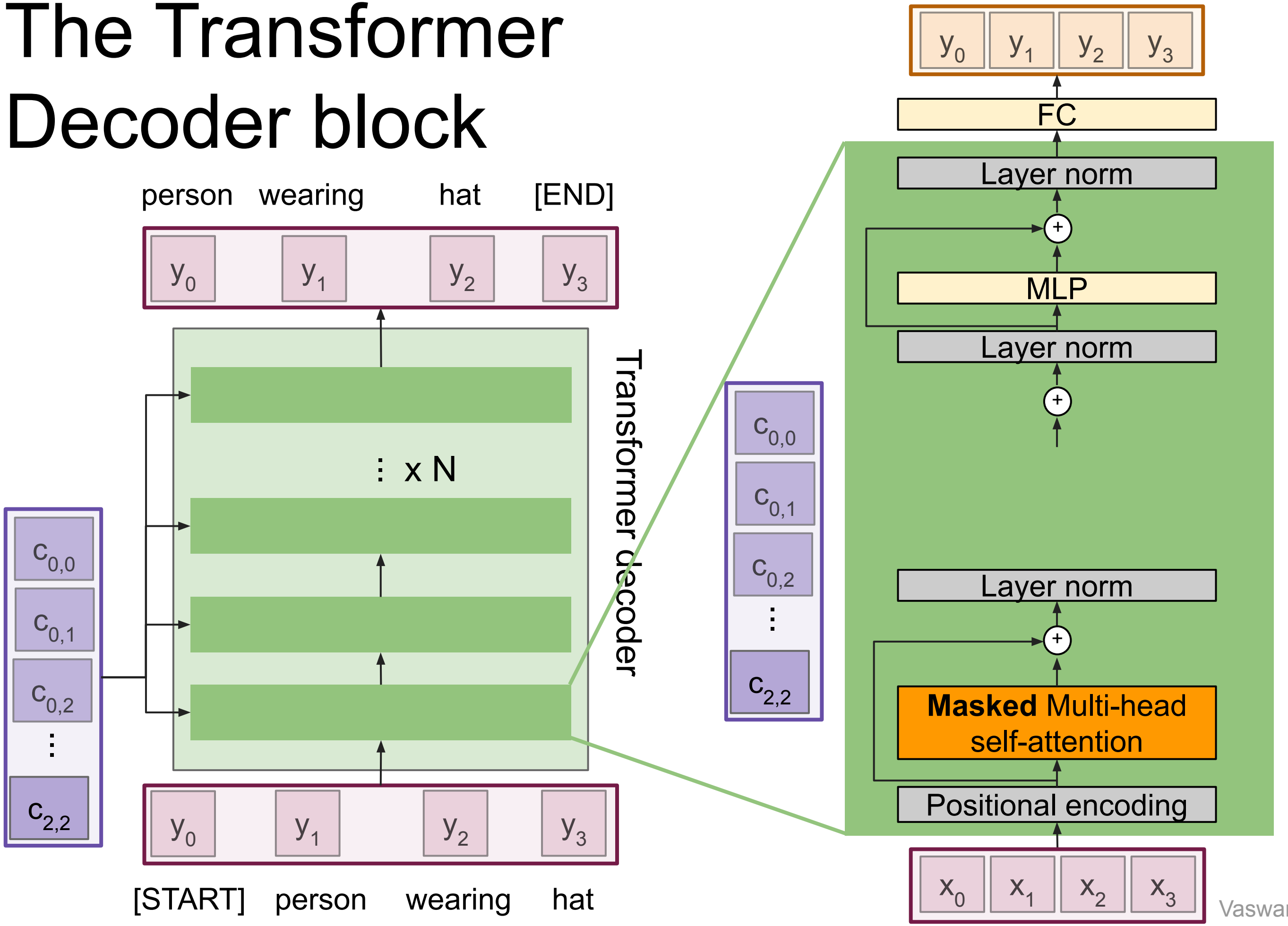
# The Transformer Decoder block




Let's dive into the transformer decoder block

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

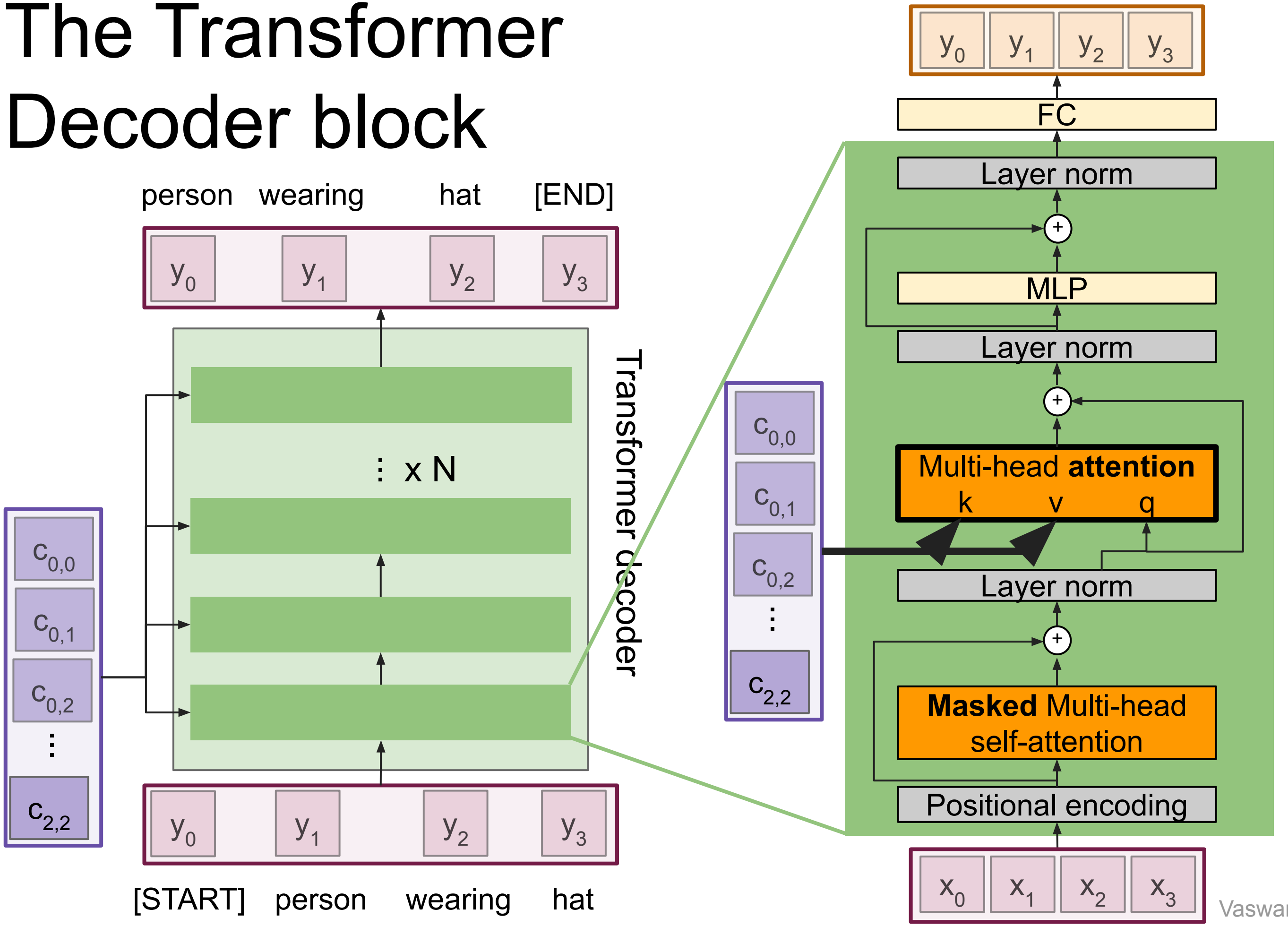
# The Transformer Decoder block




Most of the network is the same the transformer encoder.

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

# The Transformer Decoder block

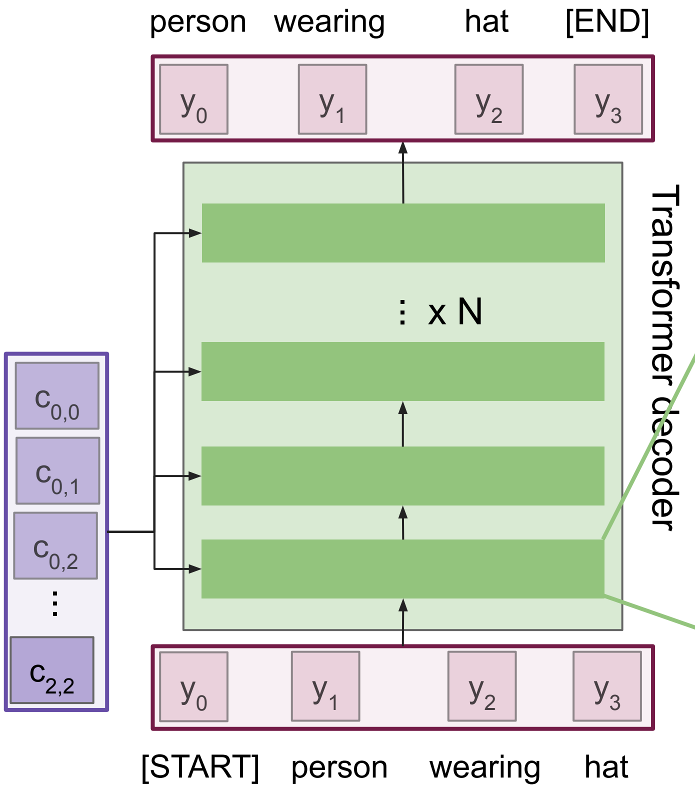


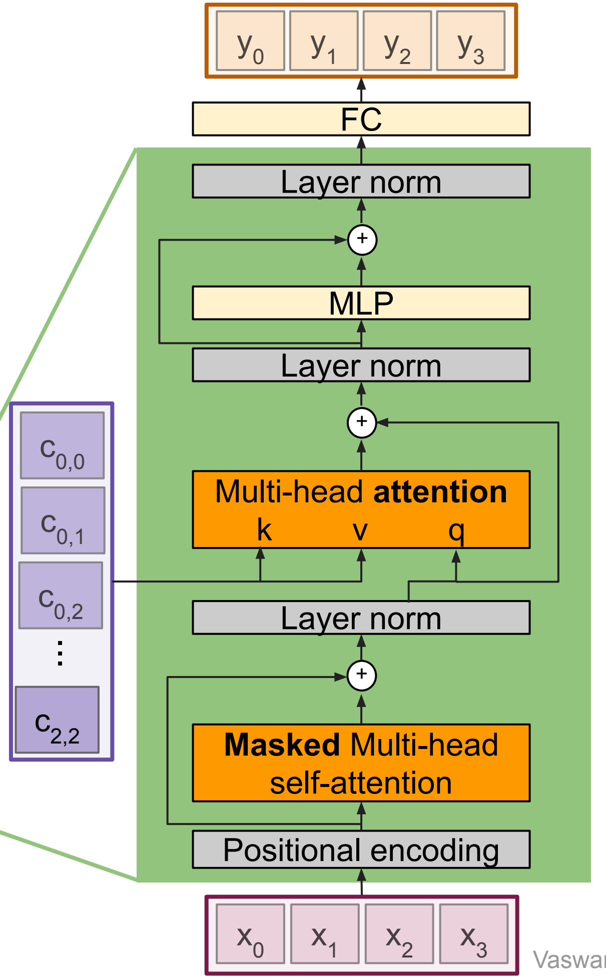

Multi-head attention block attends over the transformer encoder outputs.

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

Masked Self-attention only interacts with past inputs.

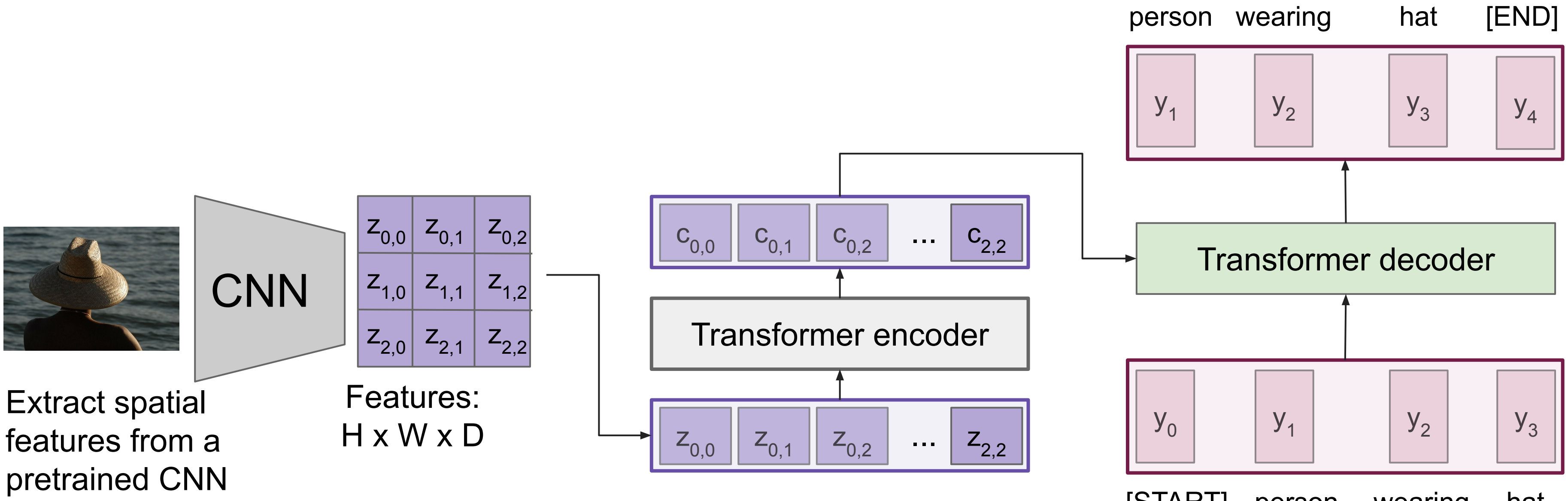
Multi-head attention block is NOT self-attention. It attends over encoder outputs.

Highly scalable, highly parallelizable, but high memory usage.

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

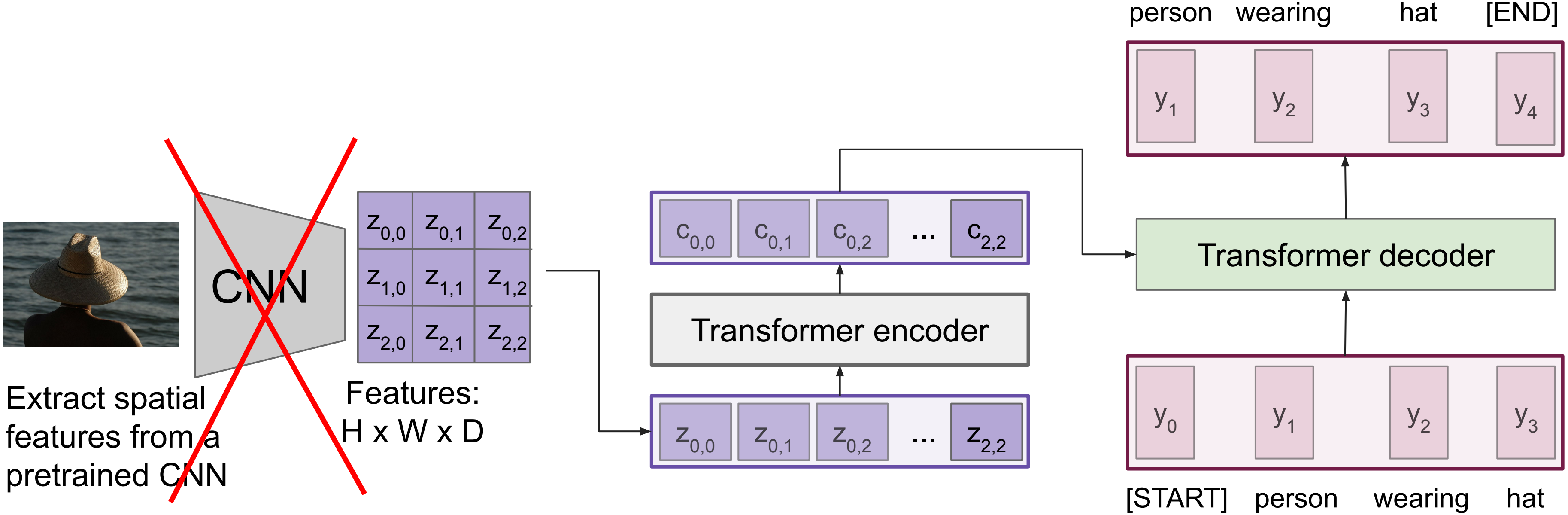
# Image Captioning using transformers

- **No recurrence at all**



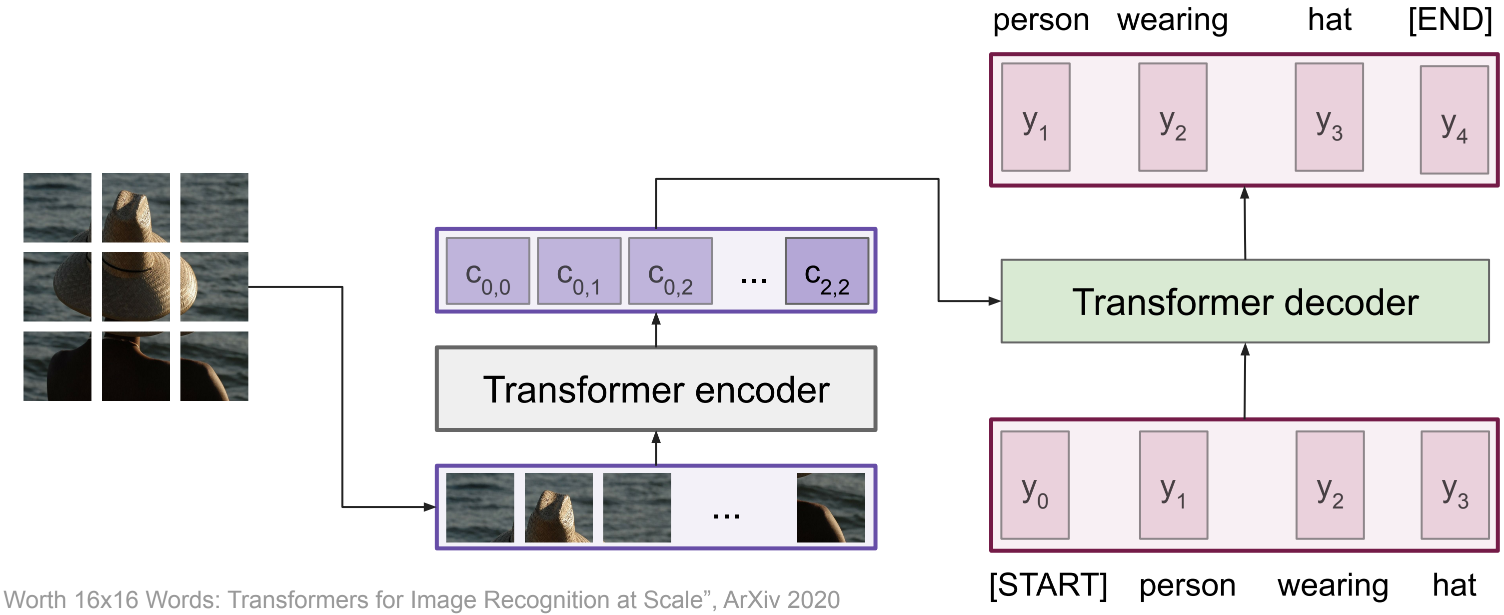
# Image Captioning using transformers

- **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

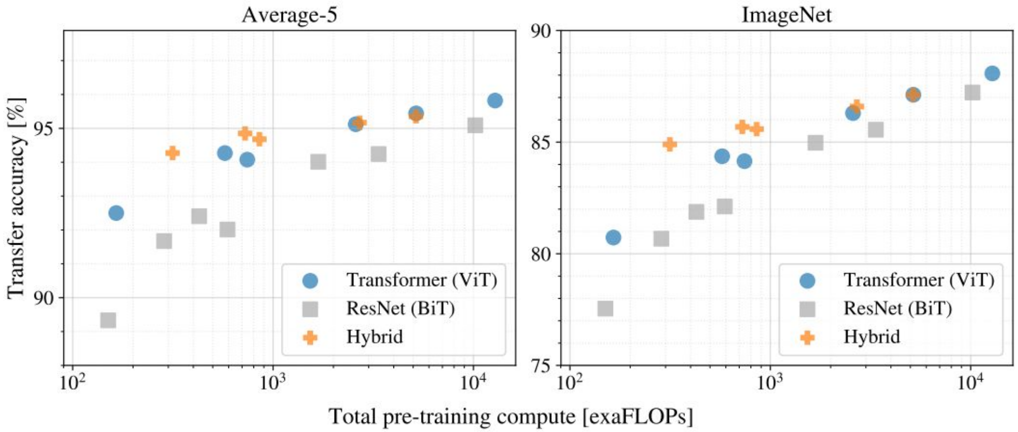



Figure 5: Performance versus cost for different architectures: Vision Transformers, ResNets, and hybrids. Vision Transformers generally outperform ResNets with the same computational budget. Hybrids improve upon pure Transformers for smaller model sizes, but the gap vanishes for larger models.

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


[link](#) to more examples

# 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: Unsupervised learning
VAEs and GANs