Lecture 11:
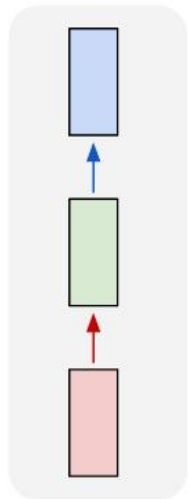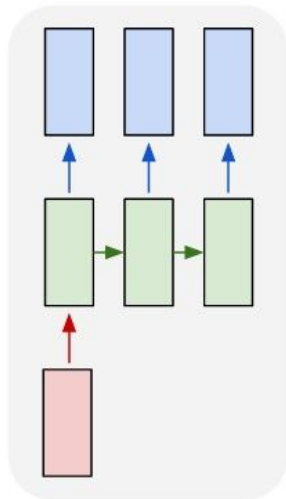# Attention and Transformers

Last Time: Recurrent Neural Networks
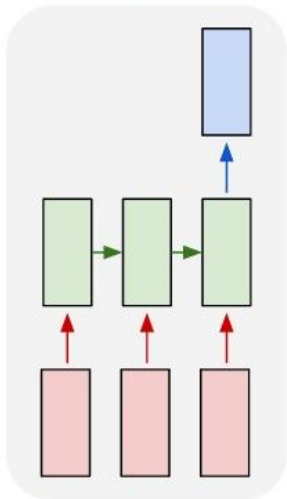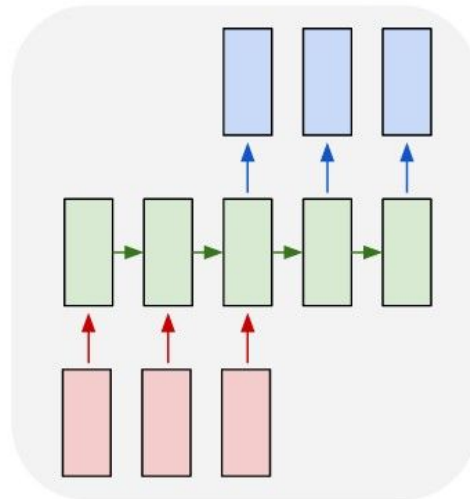
Today's Agenda:

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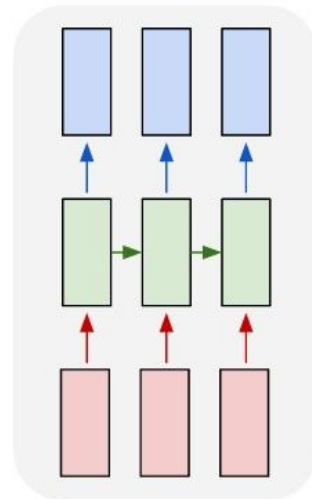- **Transformers**

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

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 $y = y_1, y_2,..., y_T$

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

person

y_1

CNN

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

MLP

h_0

h_1

c

y_0

Extract spatial features from a pretrained CNN

Features: H x W x D

[START]

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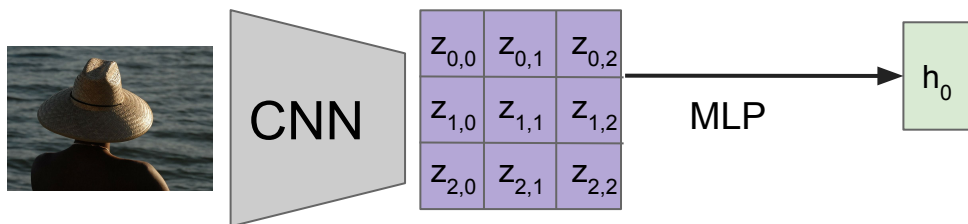
Image Captioning using spatial features

Input: Image I
Output: Sequence $y = y_1, y_2,..., y_T$

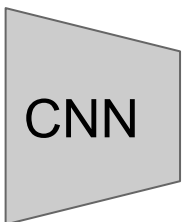**Decoder**: $y_t = g_V(y_{t-1}, h_{t-1}, c)$
where context vector c is often c = h_0

Encoder: $h_0 = f_W(z)$
where z is spatial CNN features
$f_W(.)$ is an MLP

Extract spatial features from a pretrained CNN

Features: H x W x D

person   wearing

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

[START]   person

Image Captioning using spatial features

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

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

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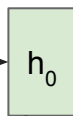where $\mathbf{z}$ is spatial CNN features
$f_{\mathbf{W}}(.)$ is an MLP

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 $y = y_1, y_2,..., y_T$

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

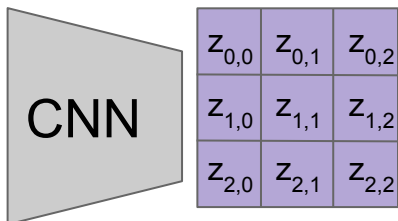**Encoder**: $h_0 = f_W(z)$
where z is spatial CNN features
$f_W(.)$ is an MLP

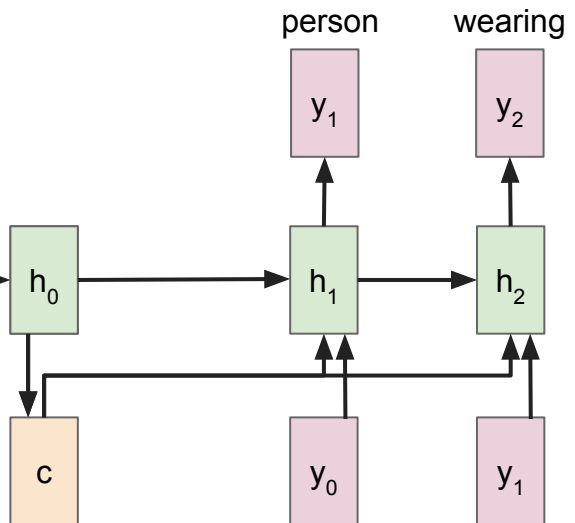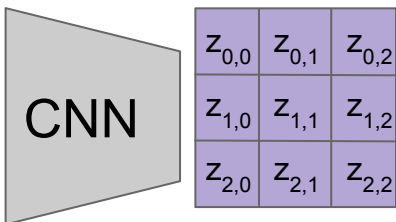Extract spatial features from a pretrained CNN

CNN

Features:
$H \times W \times D$

MLP

person   wearing   hat   [END]

[START]   person   wearing   hat

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

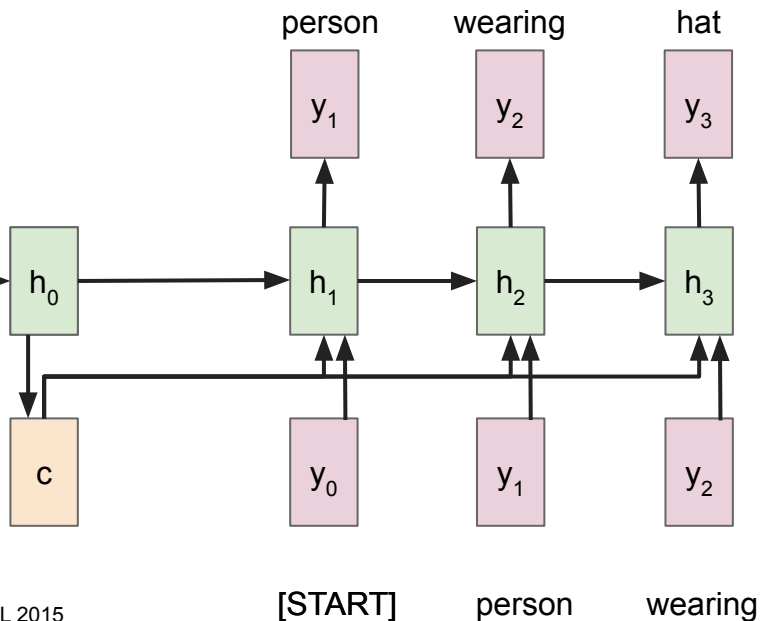**This is a problem if we want to generate really long descriptions? 100s of words long**

Extract spatial features from a pretrained CNN
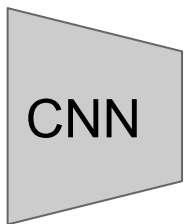
Features: H x W x D
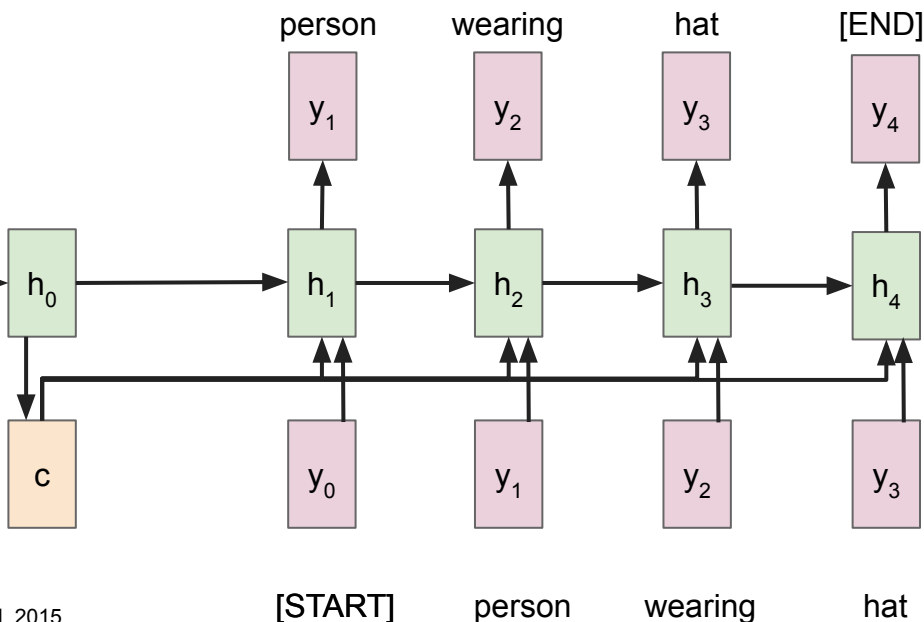
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with RNNs & Attention

Attention idea: New context vector at every time step.

Each context vector will attend to different image regions

Attention Saccades in humans

CNN

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

h_0

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

Alignment scores:
H x W

Compute alignments scores (scalars):

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

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

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

CNN

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

h_0

Extract spatial features from a pretrained CNN

Features:
H x W x D
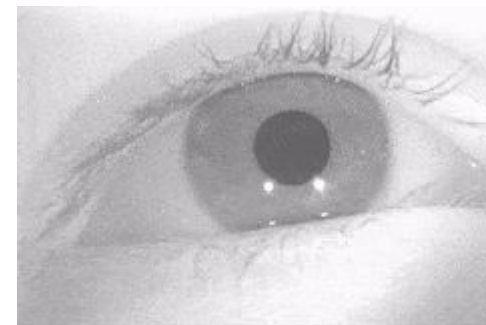
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

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

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

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:

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

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

CNN

Features:
H x W x D

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

h_0

Extract spatial
features from a
pretrained CNN

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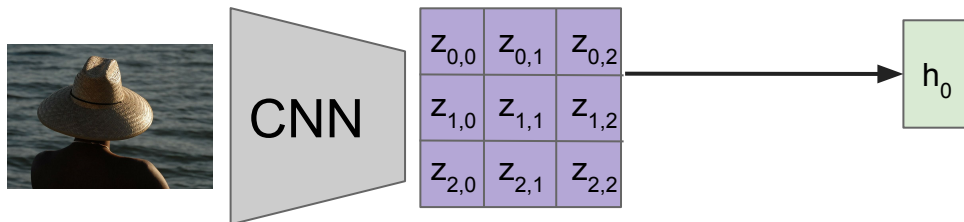
Image Captioning with RNNs & Attention

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

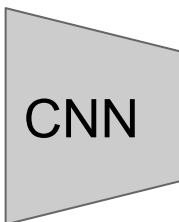| $e_{1,0,0}$ | $e_{1,0,1}$ | $e_{1,0,2}$ |
| $e_{1,1,0}$ | $e_{1,1,1}$ | $e_{1,1,2}$ |
| $e_{1,2,0}$ | $e_{1,2,1}$ | $e_{1,2,2}$ |

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:

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

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

Compute context vector:

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

CNN

Features: H x W x D

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

h_0

c_1

$\otimes$

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

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

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

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

person

Extract spatial features from a pretrained CNN
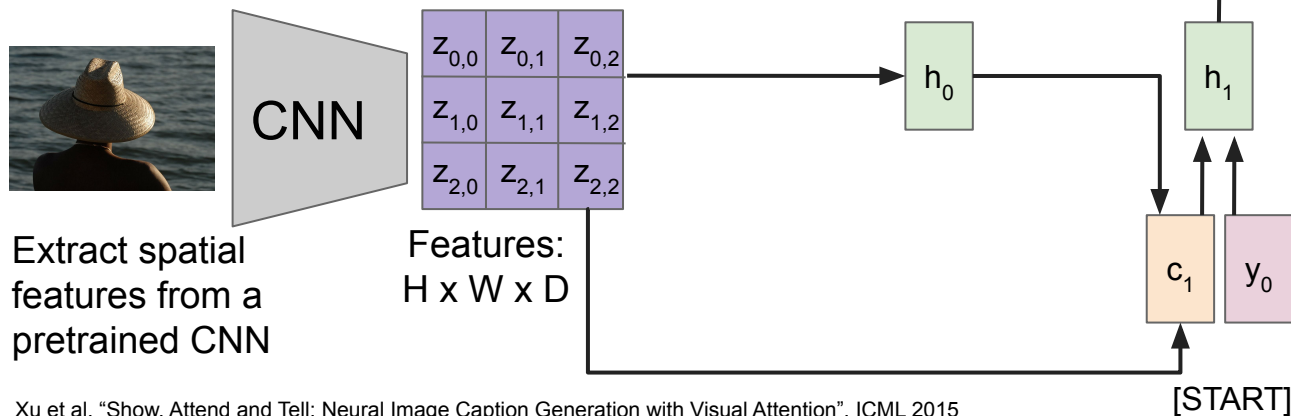
Features: H x W x D

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

Image Captioning with RNNs & Attention

Alignment scores:
H x W

Attention:
H x W

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

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

person

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

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

y_1

CNN

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

h_0

h_1

Extract spatial
features from a
pretrained CNN

Features:
H x W x D

c_1 y_0 c_2

[START]

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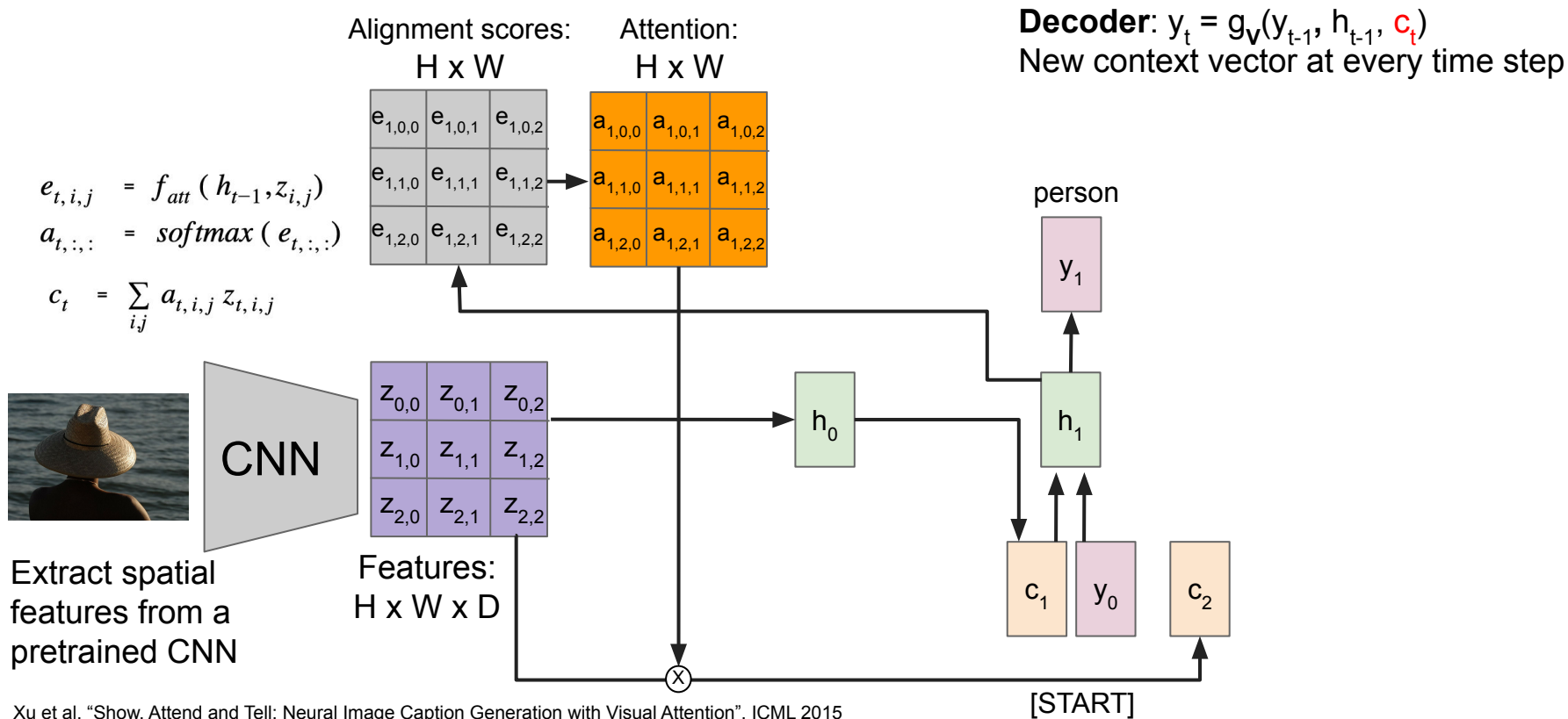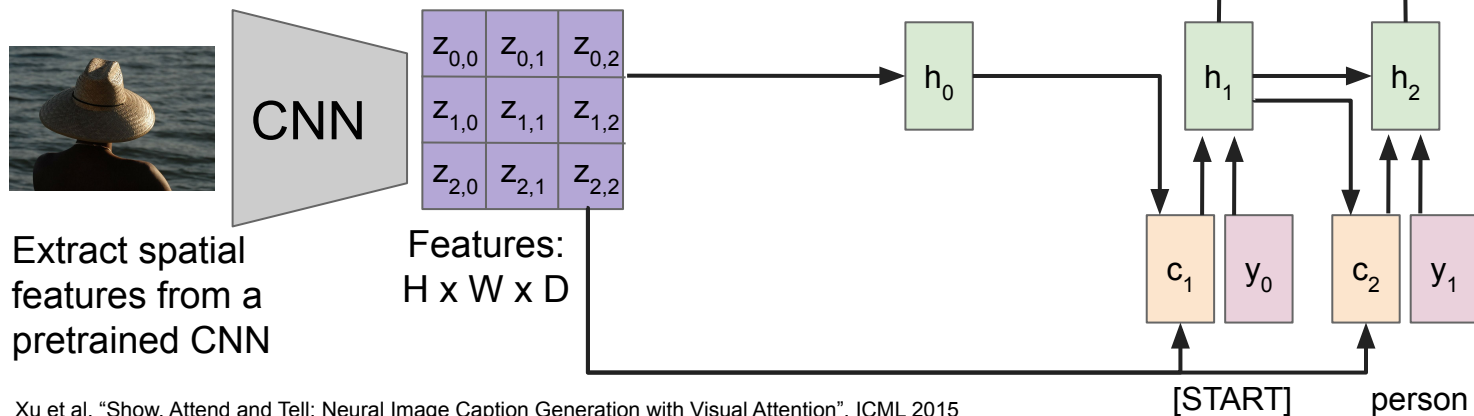
Decoder: $y_t = g_v(y_{t-1}, h_{t-1}, c_t)$
New context vector at every time step

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

Extract spatial features from a pretrained CNN
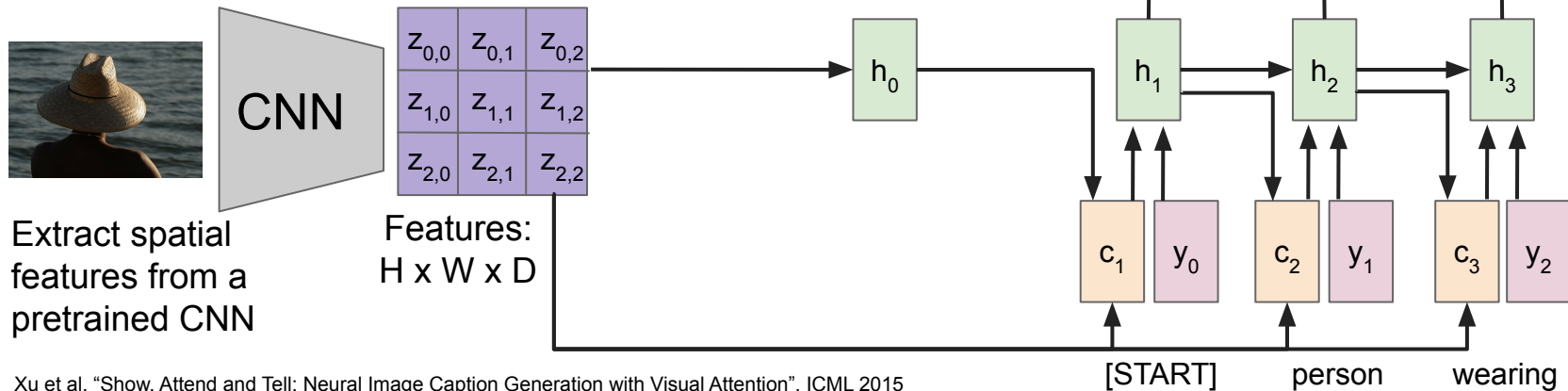
Features: H x W x D

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

Image Captioning with RNNs & Attention

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
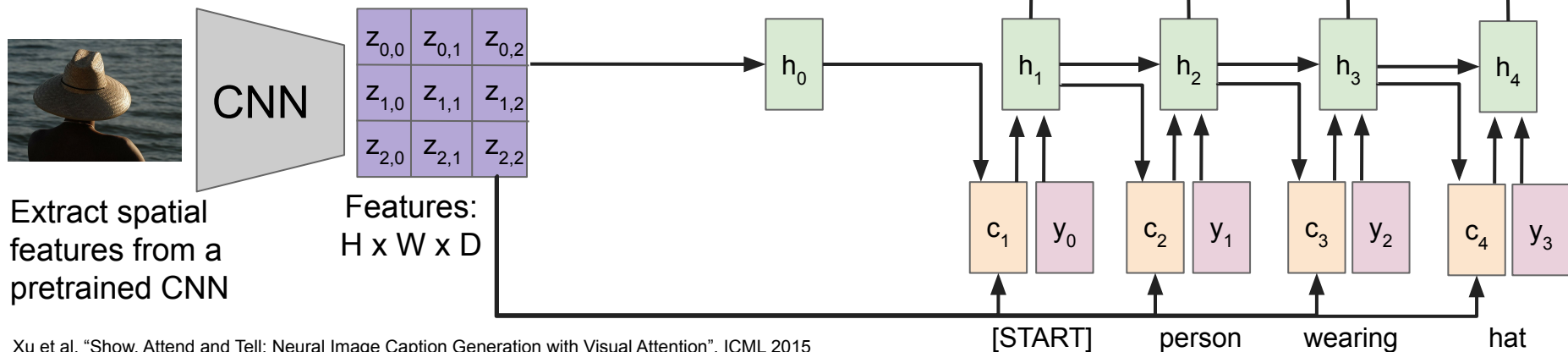
Features: H x W x D

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

Image Captioning with RNNs & Attention

Each timestep of decoder uses a different context vector that looks at different parts of the input image
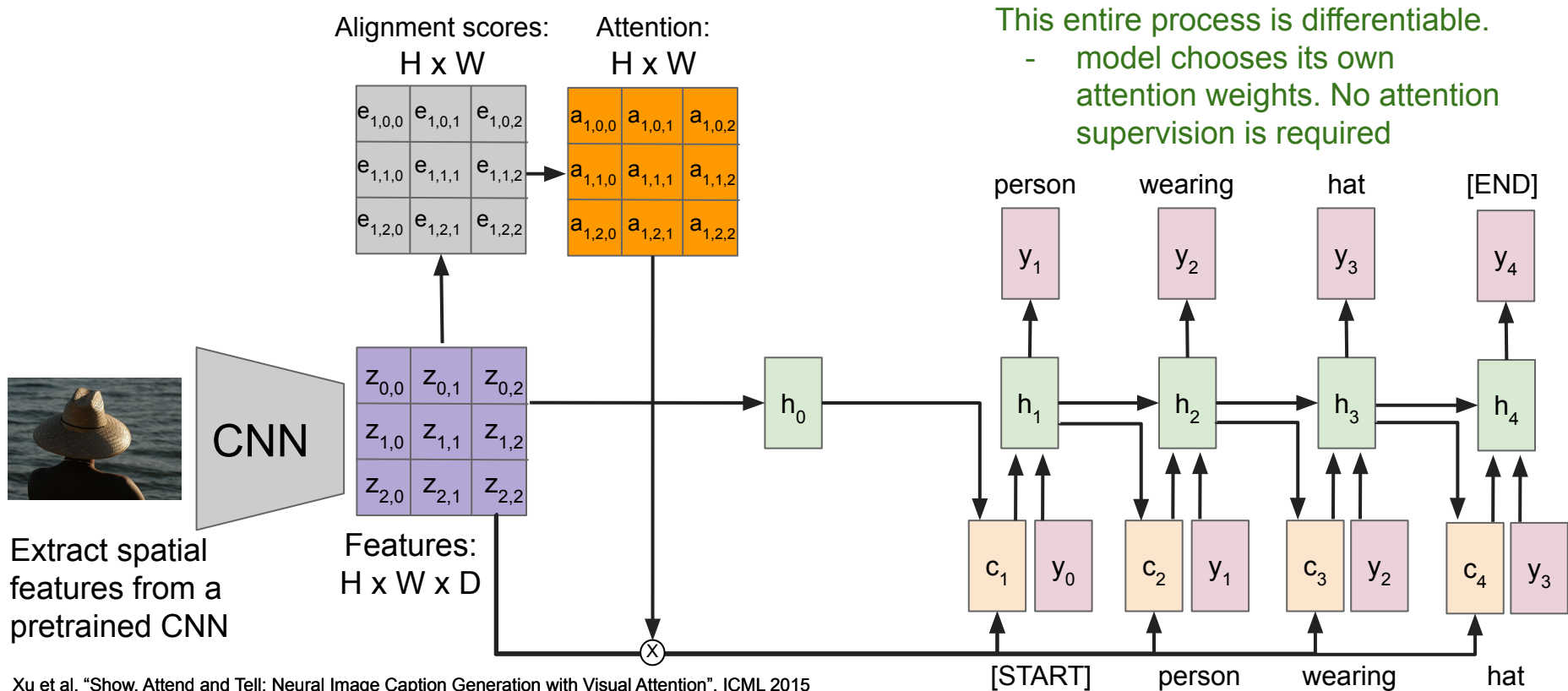
Decoder: $y_t = g_\mathbf{v}(y_{t-1}, h_{t-1}, c_t)$
New context vector at every time step

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

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

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

Extract spatial features from a pretrained CNN

Features: H x W x D
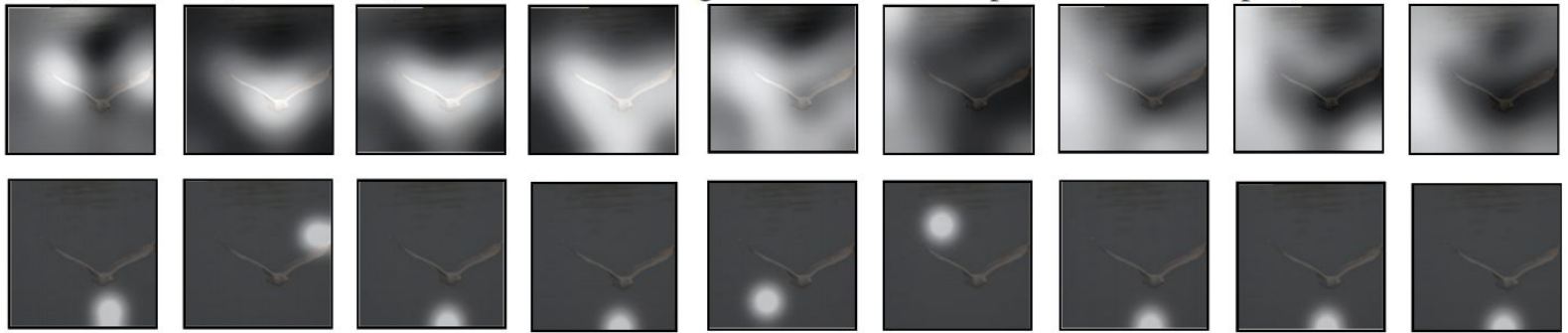
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with RNNs & Attention

Alignment scores:
H x W

Attention:
H x W

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

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

This entire process is differentiable.
- model chooses its own attention weights. No attention supervision is required

person   wearing   hat   [END]

y_1 y_2 y_3 y_4

CNN

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

h_0 h_1 h_2 h_3 h_4

Extract spatial features from a pretrained CNN

Features:
H x W x D

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 Attention

Soft attention

Hard attention
(requires
reinforcement
learning)

A   bird   flying   over   a   body   of   water   .

Image Captioning with Attention
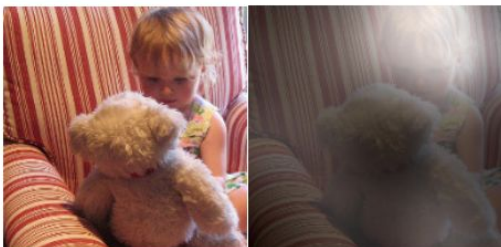

A woman is throwing a <u>frisbee</u> in a park.

A <u>dog</u> is standing on a hardwood floor.

A <u>stop</u> sign is on a road with a mountain in the background.

A little <u>girl</u> sitting on a bed with a teddy bear.

A group of <u>people</u> sitting on a boat in the water.

A giraffe standing in a forest with <u>trees</u> 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 Benchio, 2015. Reproduced with permission.

Attention can detect Gender Bias

Wrong

Right for the Right Reasons

Right for the Wrong Reasons

Right for the Right Reasons

Baseline:
A **man** sitting at a desk with a laptop computer.
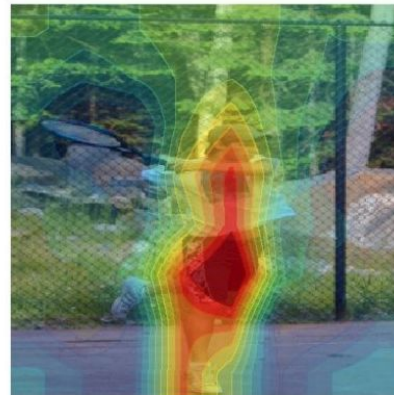
Our Model:
A **woman** sitting in front of a laptop computer.

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

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
Figures from Burns et al, copyright 2018. Reproduced with permission.

Similar tasks in NLP - Language translation example

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

x_0	x_1	x_2	x_3
personne	portant	un	chapeau
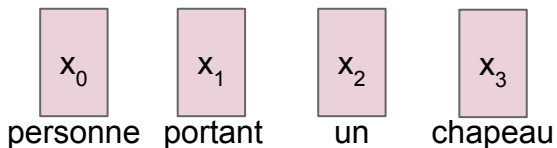
Similar tasks in NLP - Language translation example

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

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
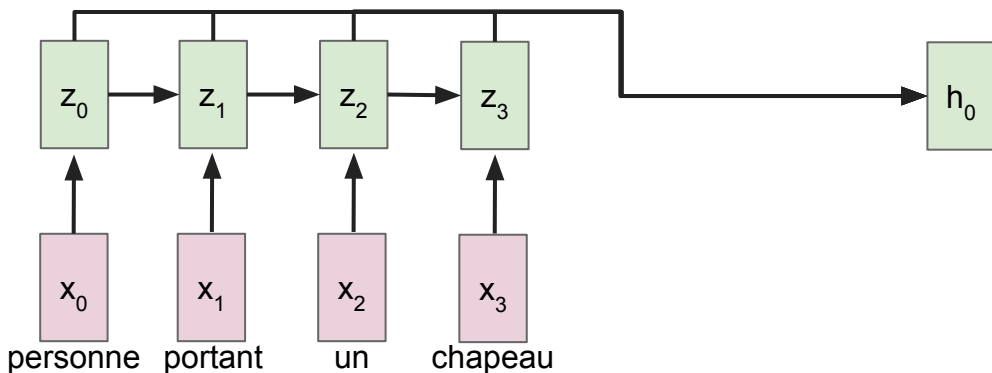
Similar tasks in NLP - Language translation example

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

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 $z_t = RNN(x_t, u_{t-1})$
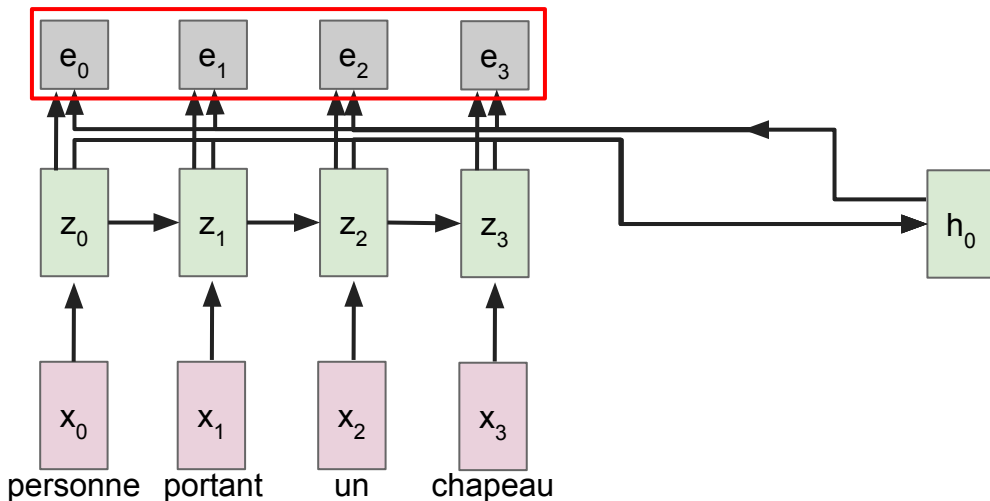$f_{\mathbf{W}}(.)$ is MLP
u is the hidden RNN state
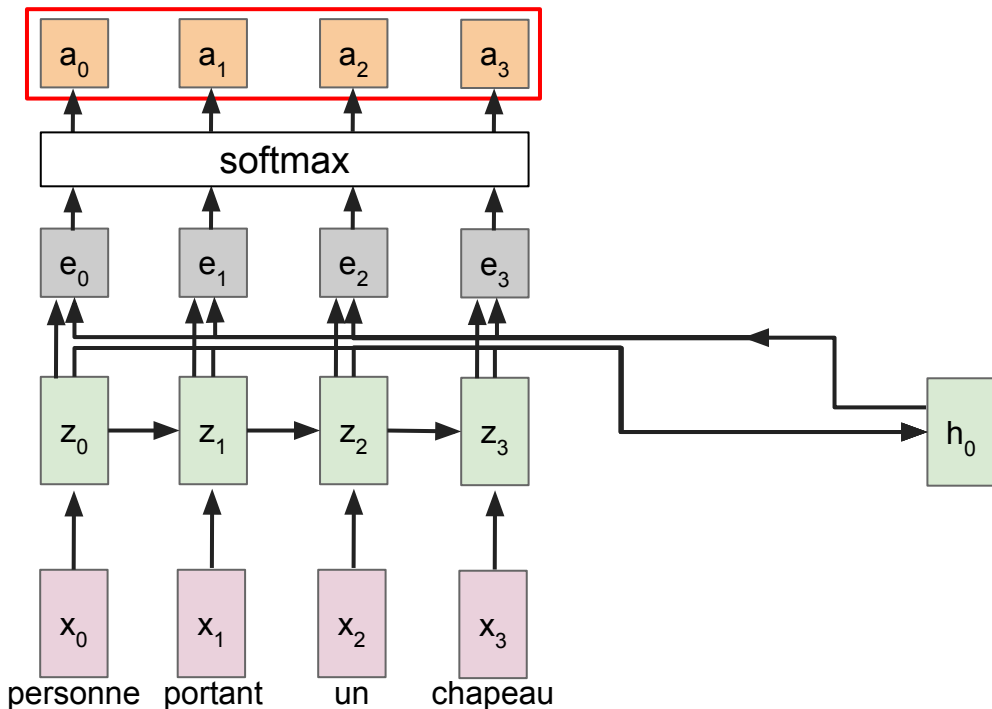
Attention in NLP - Language translation example

Compute alignments scores (scalars):

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

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

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

Attention in NLP - Language translation example

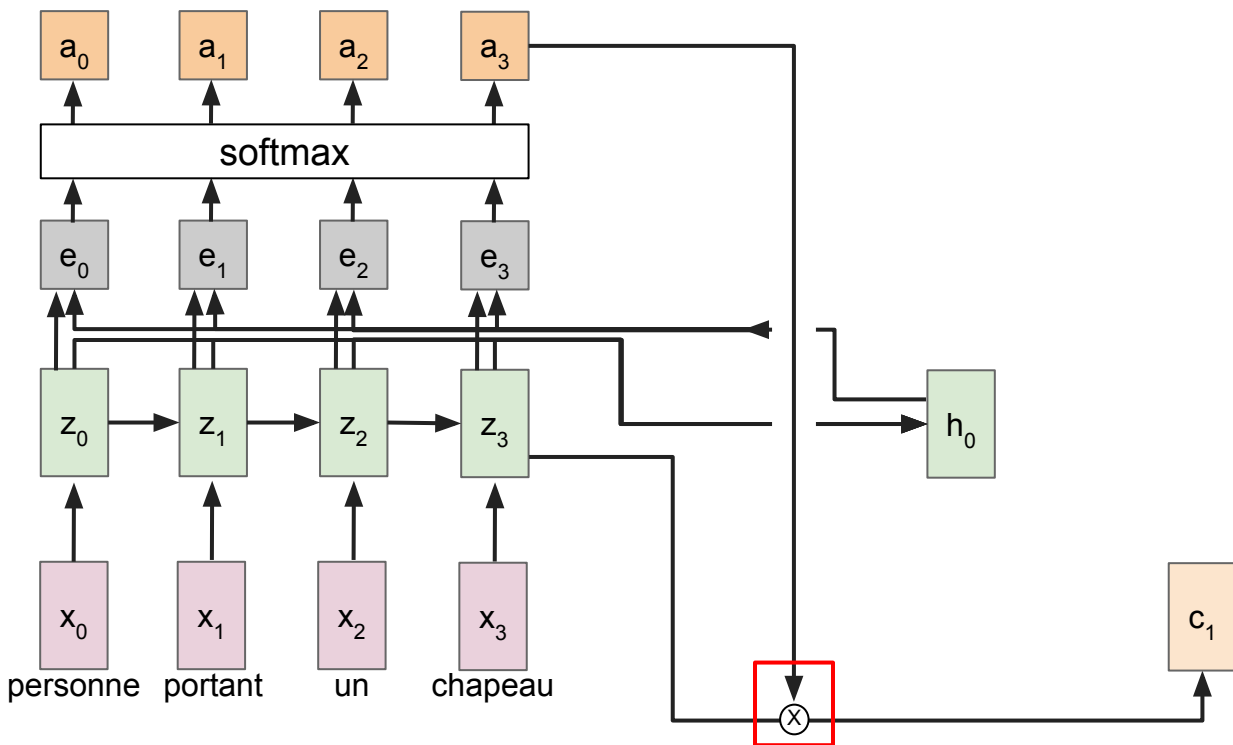Compute alignments scores (scalars):

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

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

Normalize to get attention weights:

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

$0 < a_{t,i,j} < 1$,
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:
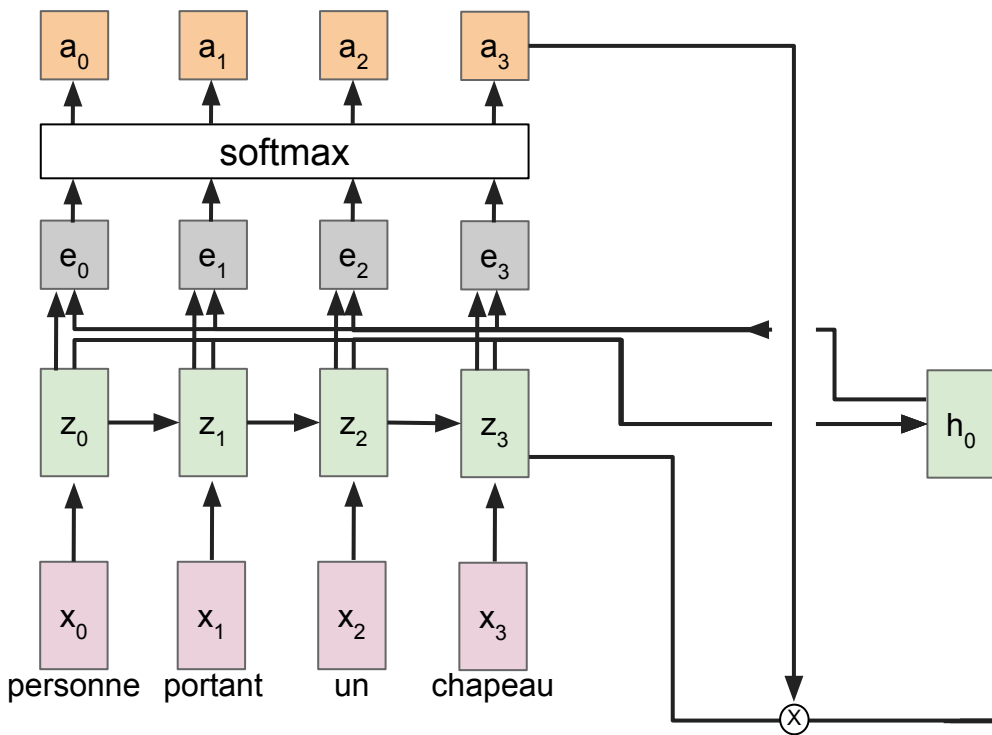
$$a_{t,:} = softmax(e_{t,:})$$
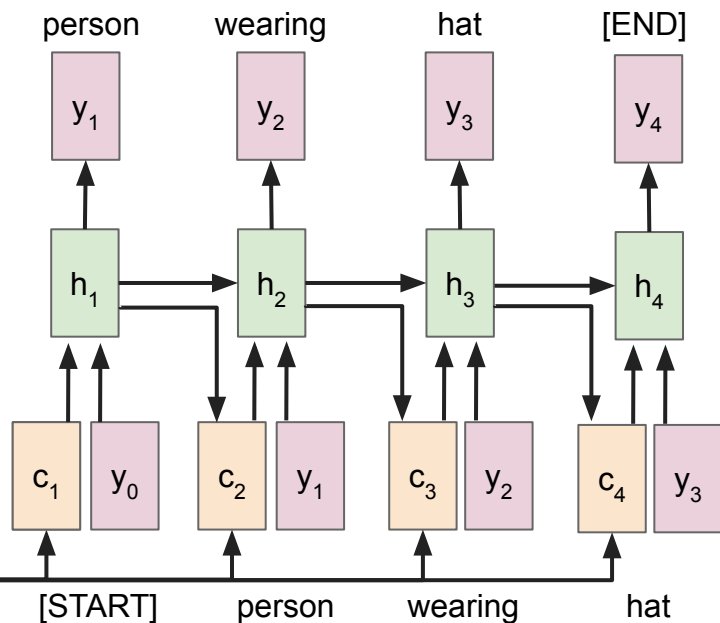
$0 < a_{t,i,j} < 1$, attention values sum to 1

Compute context vector:

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

personne   portant   un   chapeau

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 $c = h_0$

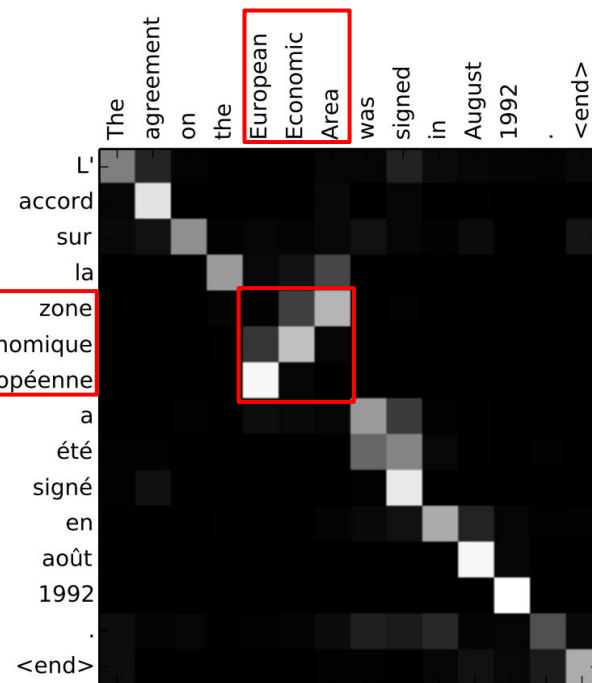Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

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

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

Features

h

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

Attention we just saw in image captioning
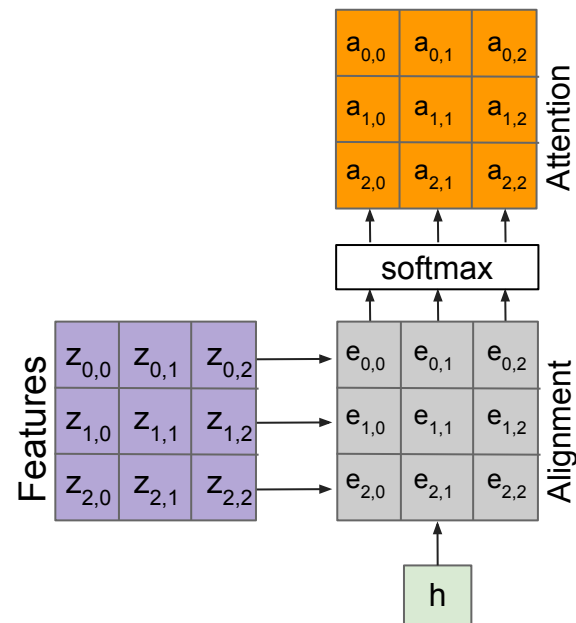
Operations:
Alignment: $e_{i,j} = f_{att}(h, z_{i,j})$

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

Features — Alignment

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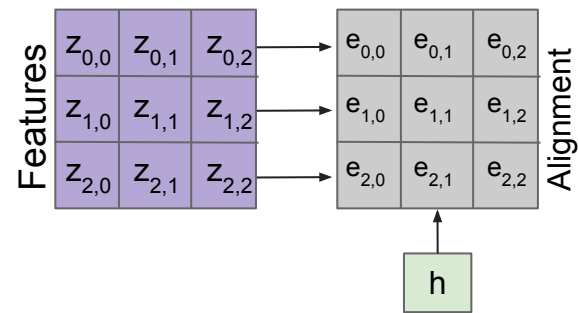
Inputs:
Features: **z** (shape: H x W x D)
Query: **h** (shape: D)

Attention we just saw in image captioning

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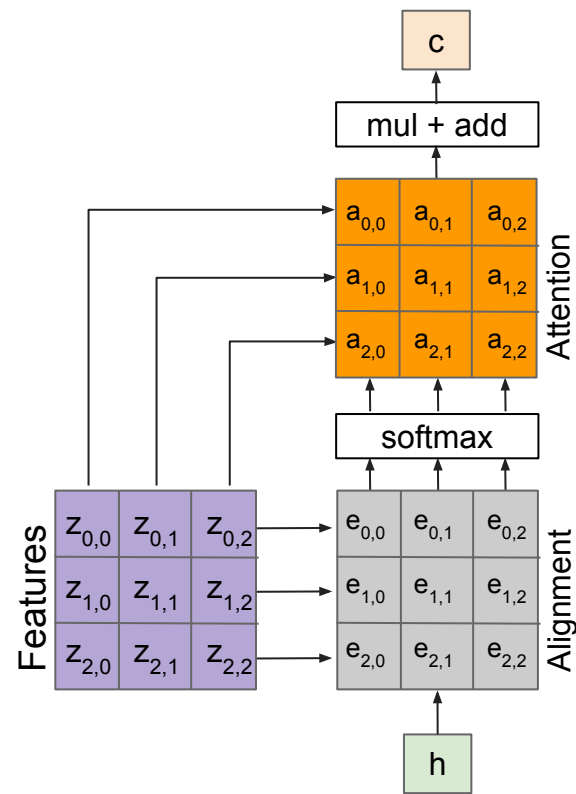Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$

Inputs:
Features: $\mathbf{z}$ (shape: H x W x D)
Query: $\mathbf{h}$ (shape: D)
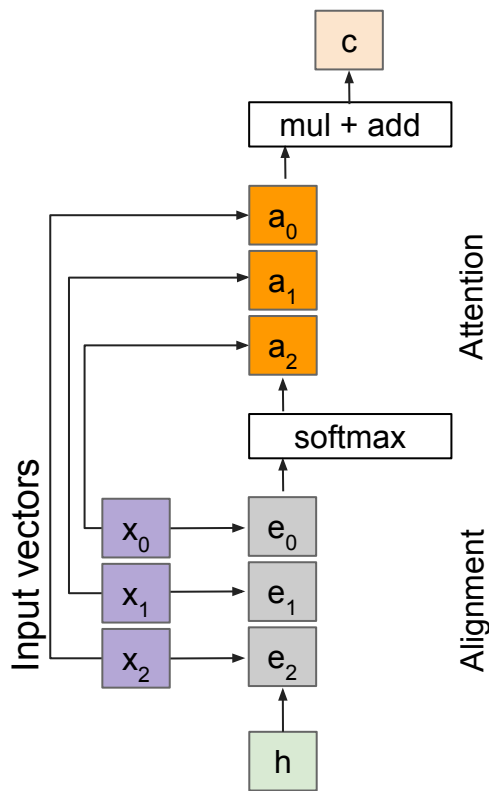
Attention we just saw in image captioning
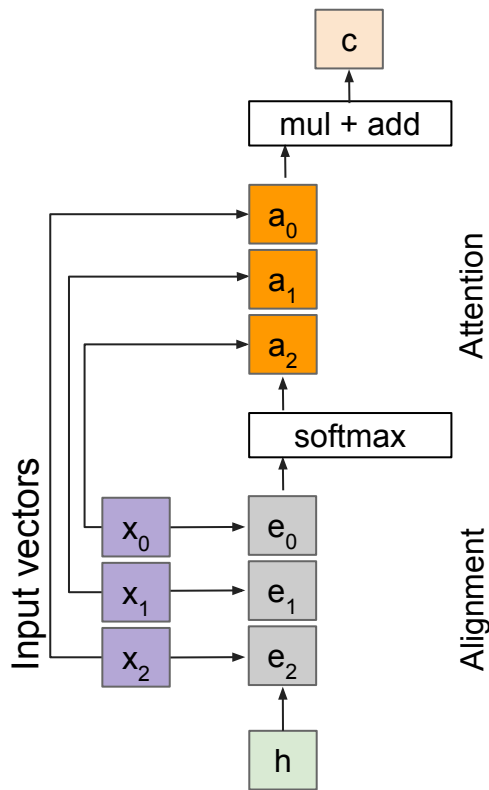

Outputs:
context vector: $\mathbf{c}$ (shape: D)

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: $\mathbf{z}$ (shape: H x W x D)
Query: $\mathbf{h}$ (shape: D)

General attention layer

c

mul + add

a_0

a_1

a_2

Attention

softmax

Input vectors

x_0 → e_0

x_1 → e_1

x_2 → e_2

Alignment

h

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

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

Attention operation is **permutation invariant.**
- Doesn't care about ordering of the features
- Stretch H x W = N into N vectors

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

General attention layer

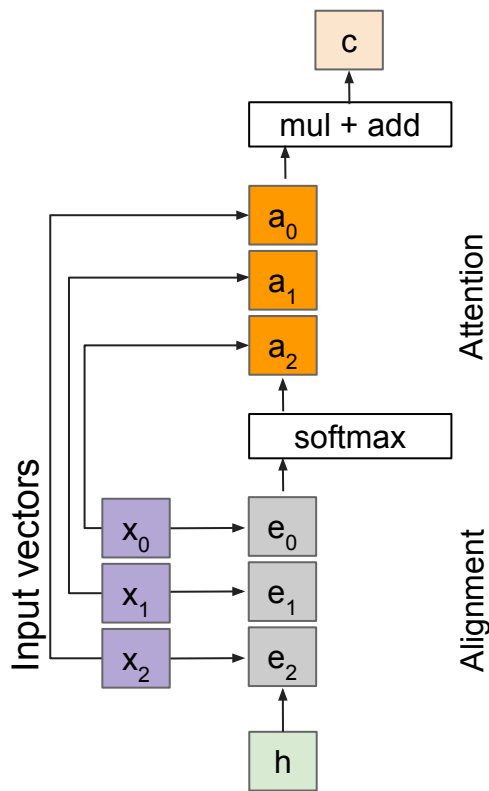**Outputs:**
context vector: $\mathbf{c}$ (shape: D)

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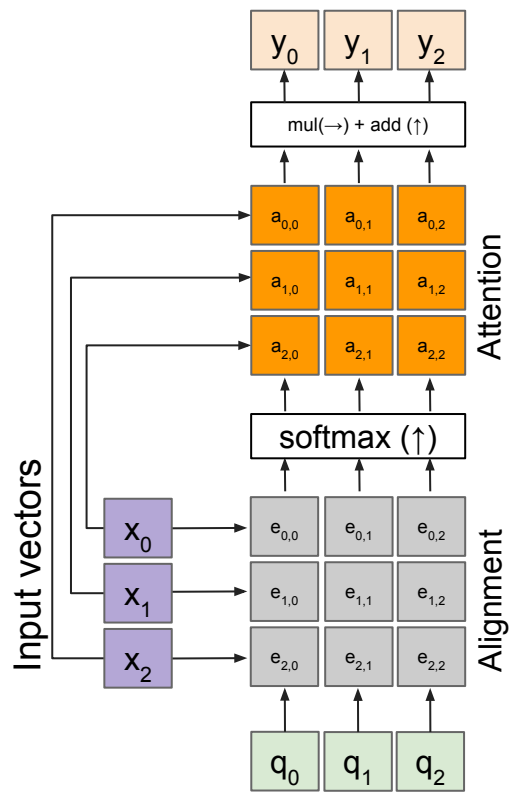Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $\mathbf{c} = \sum_i a_i x_i$

Change $f_{att}(.)$ to a simple dot product
- only works well with key & value transformation trick (will mention in a few slides)

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

General attention layer

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

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

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 $\sqrt{D}$ to reduce effect of large magnitude vectors

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

General attention layer

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

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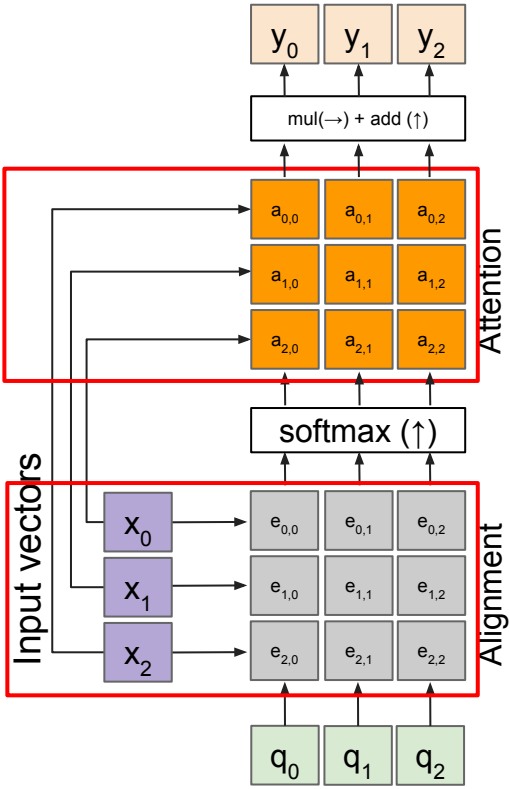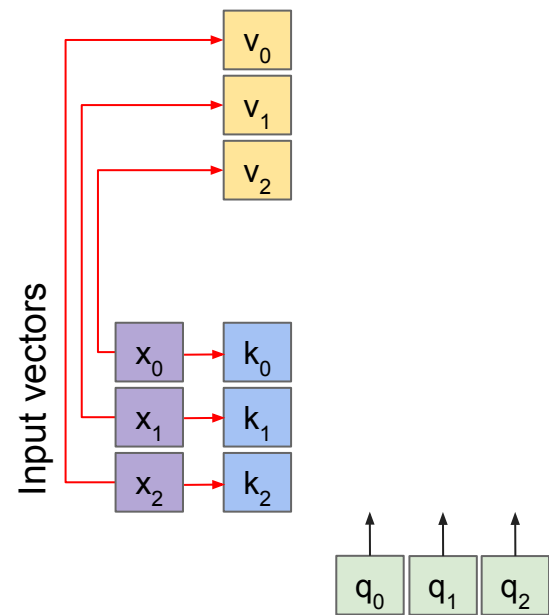- each query creates a new output context vector

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

Multiple query vectors

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

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

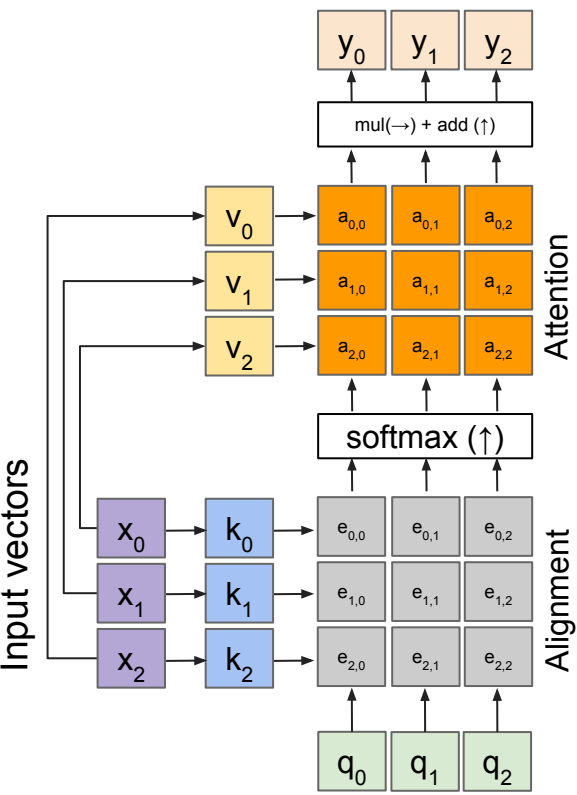Notice that the input vectors are used for both the alignment as well as the attention calculations.
- We can add more expressivity to the layer by adding a different FC layer before each of the two steps.

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

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: $\mathbf{x}$ (shape: N x D)
Queries: $\mathbf{q}$ (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: D_v)

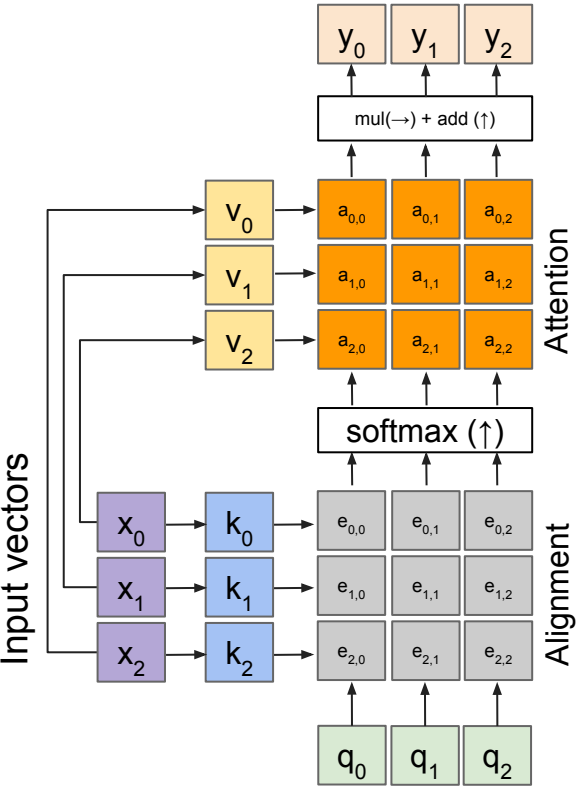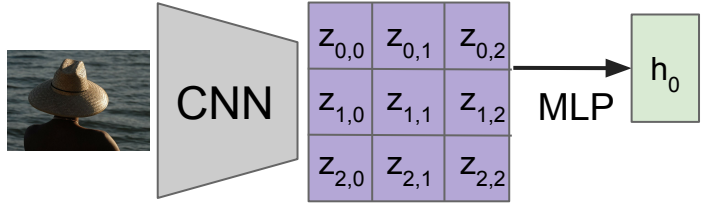**Operations:**
Key vectors: $\mathbf{k} = \mathbf{x}W_k$
Value vectors: $\mathbf{v} = \mathbf{x}W_v$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $y_j = \sum_i a_{i,j} v_i$

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

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

Notice that the input vectors are used for both the alignment as well as the attention calculations.
-   We can add more expressivity to the layer by adding a different FC layer before each of the two steps.

General attention layer

y_0 y_1 y_2

mul($\rightarrow$) + add ($\uparrow$)

v_0 v_1 v_2

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

Attention

softmax ($\uparrow$)

x_0 x_1 x_2 (Input vectors)

k_0 k_1 k_2

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

Alignment

q_0 q_1 q_2

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

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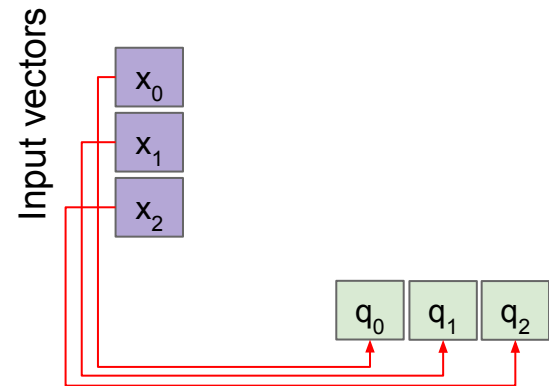Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $y_j = \sum_i a_{i,j} v_i$

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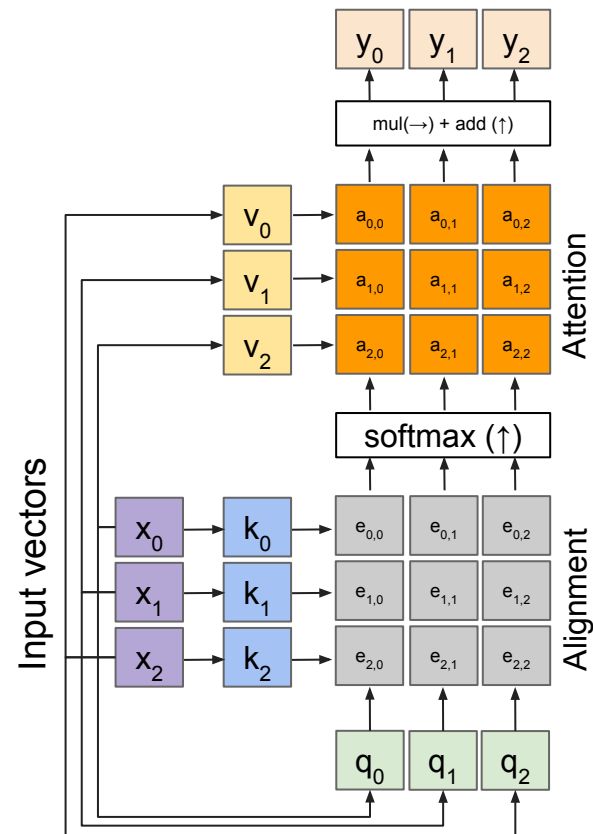Input vectors: **x** (shape: N x D)
Queries: **q** (shape: M x D_k)

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

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

CNN

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

MLP

h_0

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

Input vectors

| x_0 |
| x_1 |
| x_2 |

| q_0 | q_1 | q_2 |

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

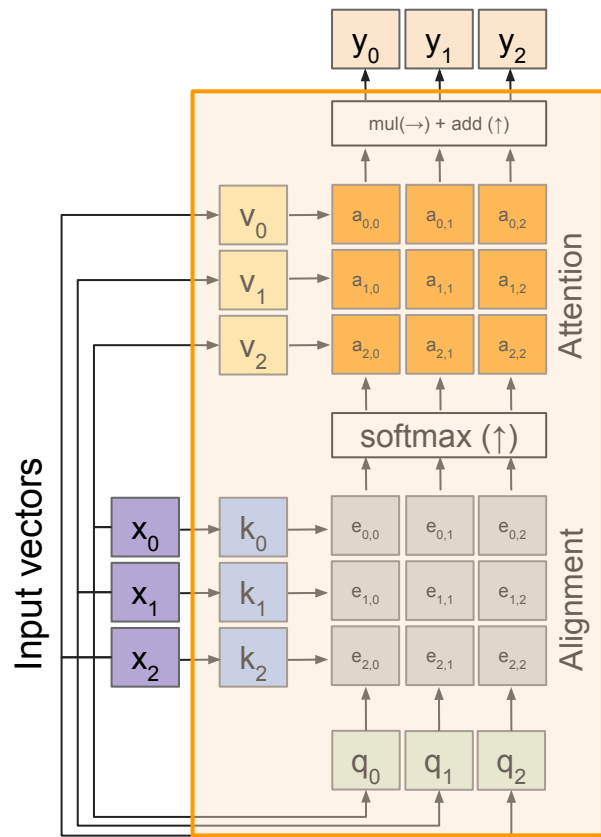# Self attention layer

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

Operations:
Key vectors: $\mathbf{k} = \mathbf{xW_k}$
Value vectors: $\mathbf{v} = \mathbf{xW_v}$
Query vectors: $\mathbf{q} = \mathbf{xW_q}$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $y_j = \sum_i a_{i,j} \, v_i$
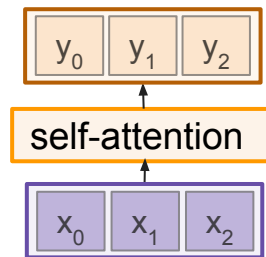
Inputs:
Input vectors: $\mathbf{x}$ (shape: N x D)

Self attention layer - attends over sets of inputs

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

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

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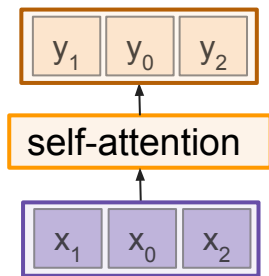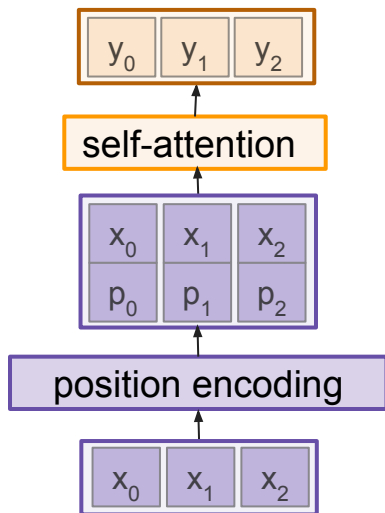Input vectors: **x** (shape: N x D)

Self attention layer - attends over sets of inputs

Permutation invariant

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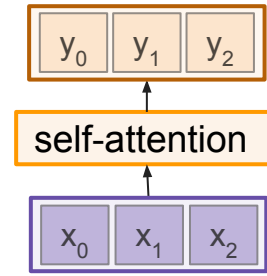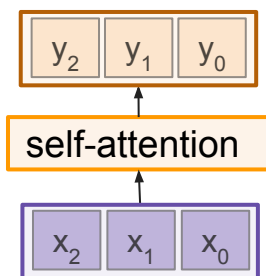
Positional encoding

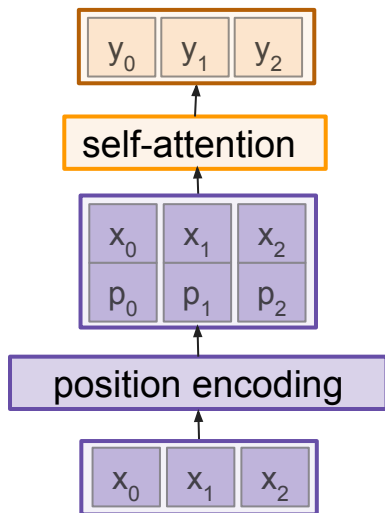Concatenate special positional encoding p_j to each input vector x_j

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 →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 ε [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**.

Positional encoding

Concatenate special positional encoding p_j to each input vector x_j

We use a function pos: $N \to R^d$ to process the position j of the vector into a d-dimensional vector
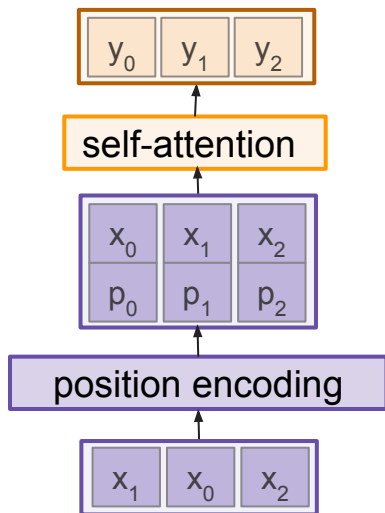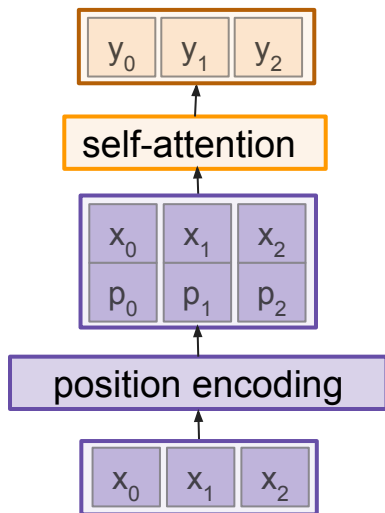
So, $p_j = pos(j)$

Options for $pos(.)$

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

2. Design a fixed function with the desiderata
   -
$$p(t) = \begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}_d$$
   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
   -

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

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

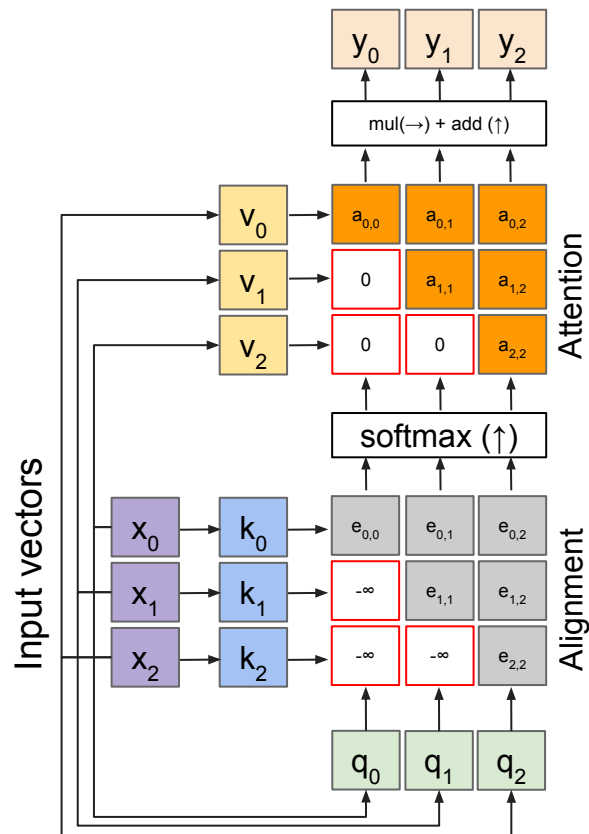Intuition:

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

image source

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

Masked self-attention layer

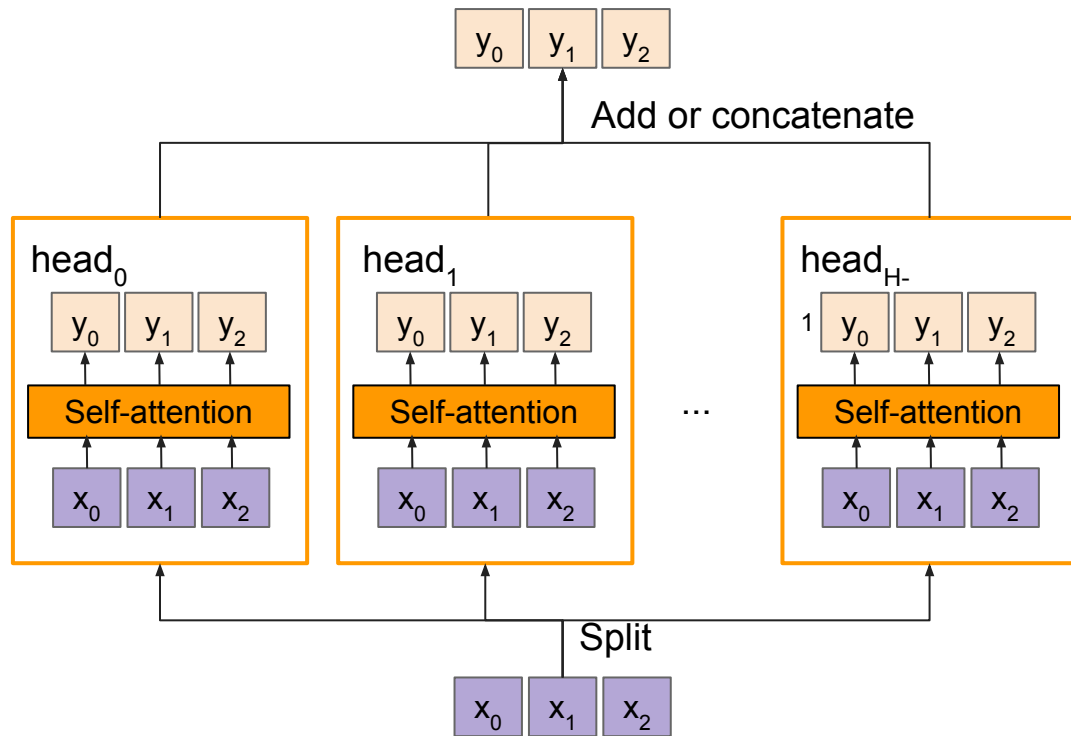**Outputs:**
context vectors: $\mathbf{y}$ (shape: D_v)

Operations:
Key vectors: $\mathbf{k} = \mathbf{x}W_k$
Value vectors: $\mathbf{v} = \mathbf{x}W_v$
Query vectors: $\mathbf{q} = \mathbf{x}W_q$
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:
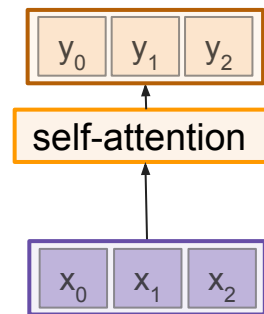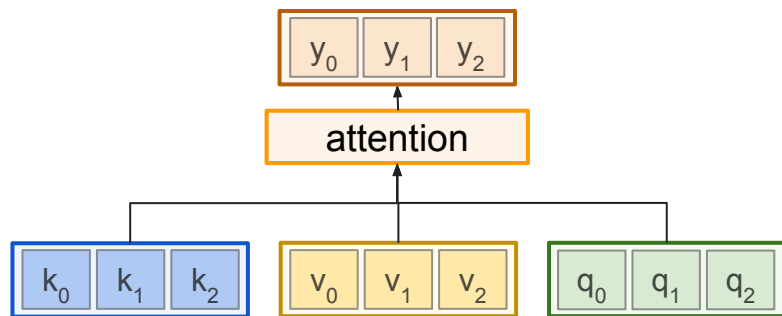Input vectors: $\mathbf{x}$ (shape: N x D)

- 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)
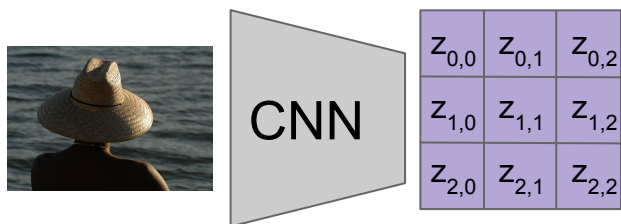
Today's Agenda:

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

Extract spatial features from a pretrained CNN

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

Image Captioning using transformers

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

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

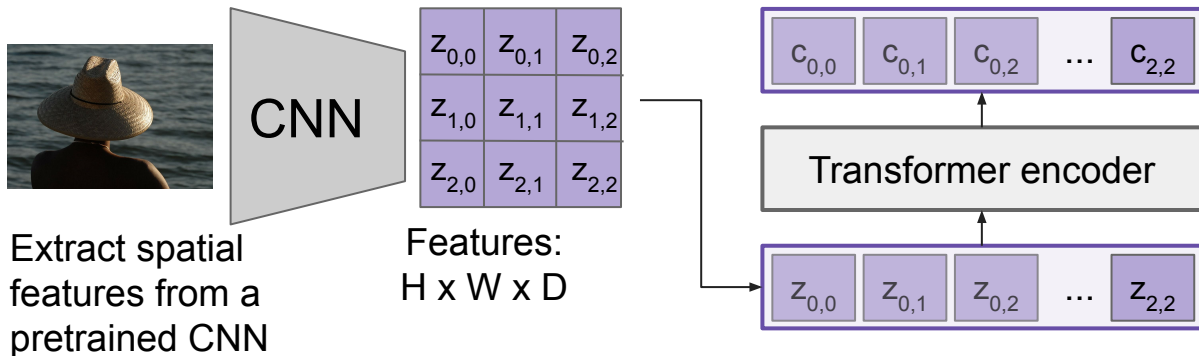Extract spatial features from a pretrained CNN

CNN

Features: H x W x D

Image Captioning using transformers

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

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

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

Extract spatial features from a pretrained CNN

Features:
H x W x D
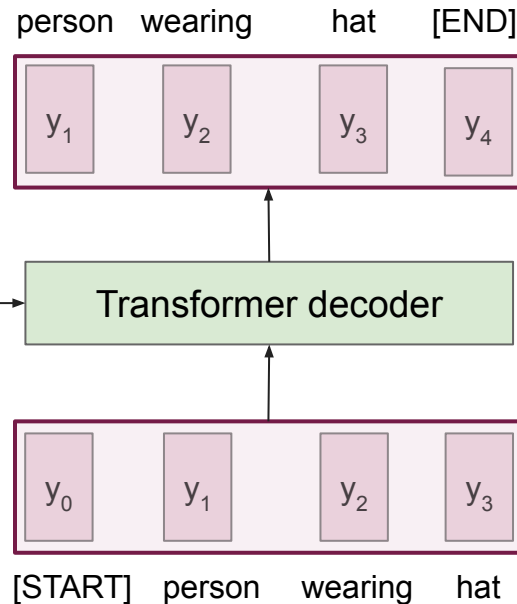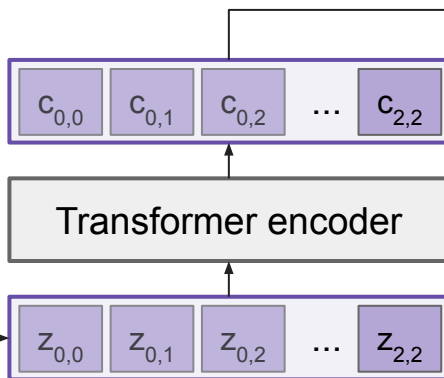
The Transformer encoder block

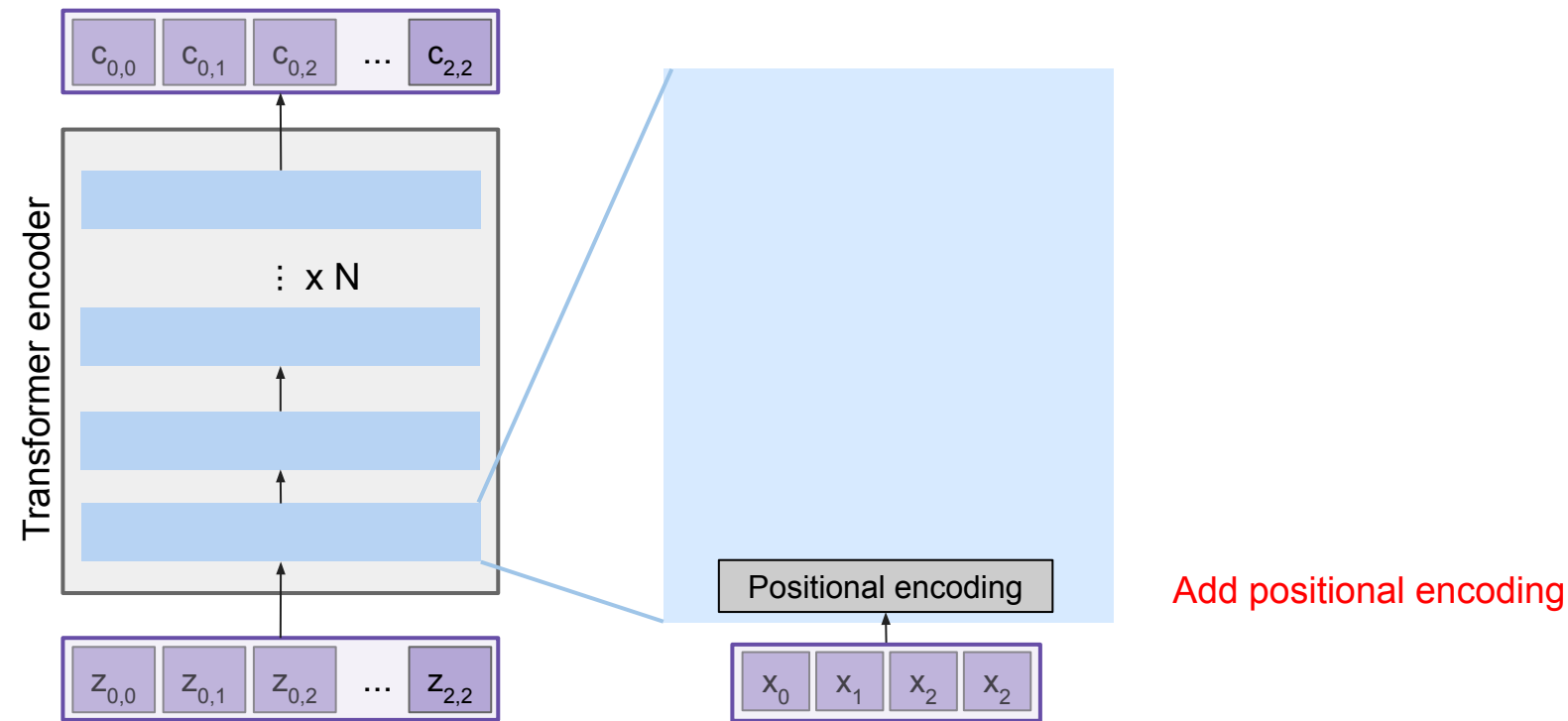
Made up of N encoder blocks.

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

The Transformer encoder block

| $c_{0,0}$ | $c_{0,1}$ | $c_{0,2}$ | ... | $c_{2,2}$ |

Transformer encoder

: x N

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

| x_0 | x_1 | x_2 | x_2 |

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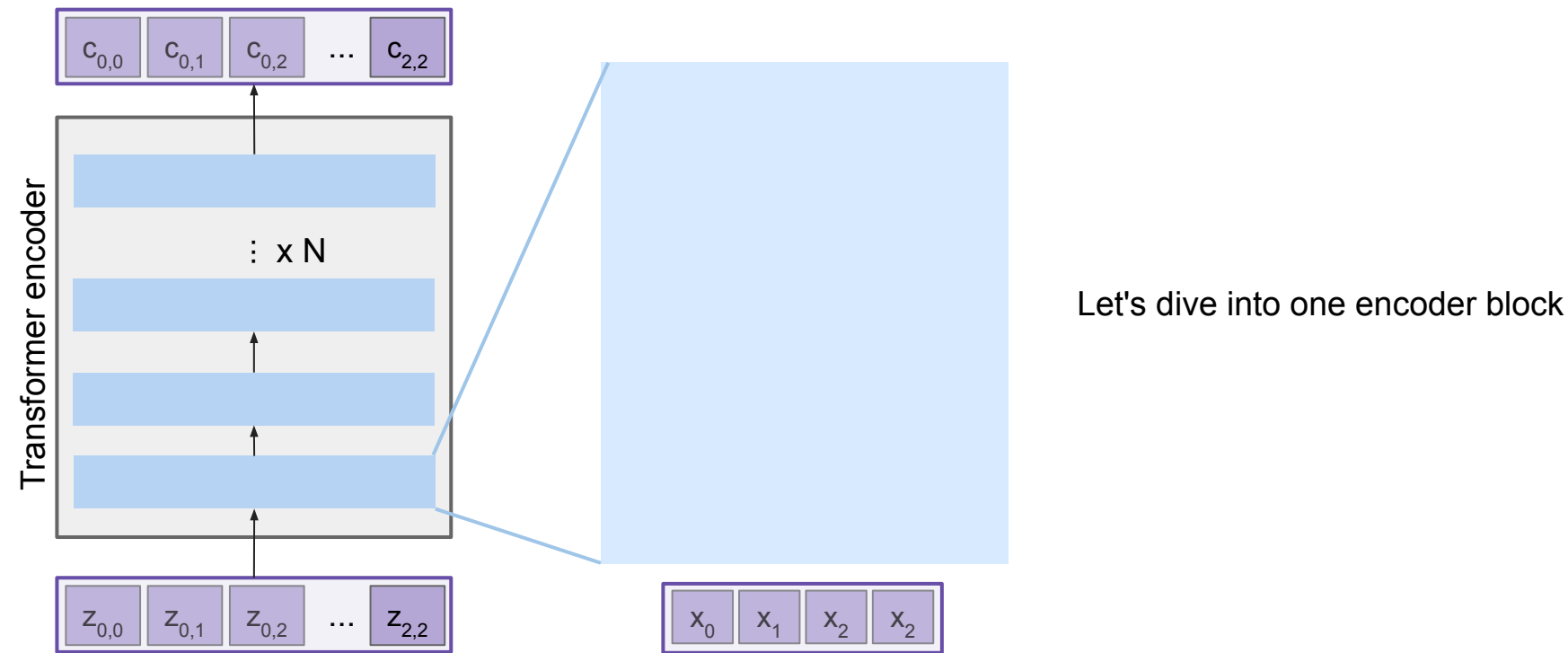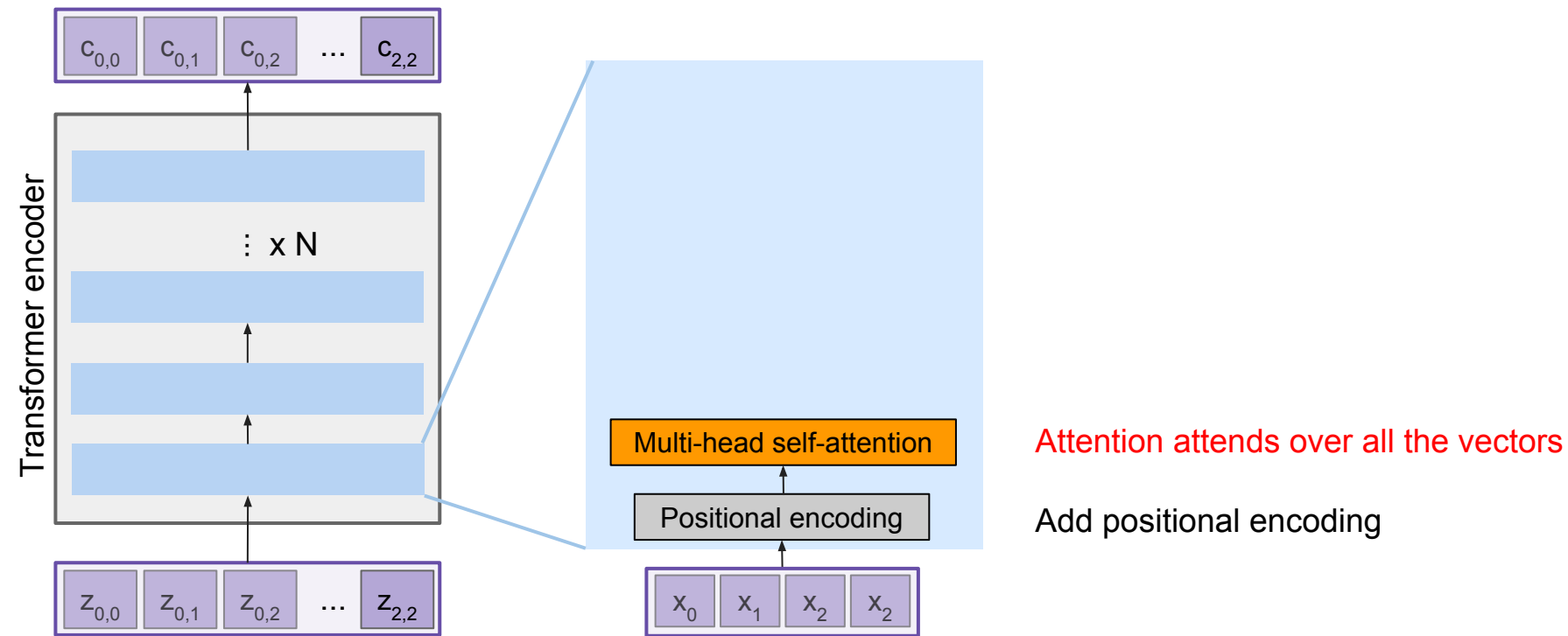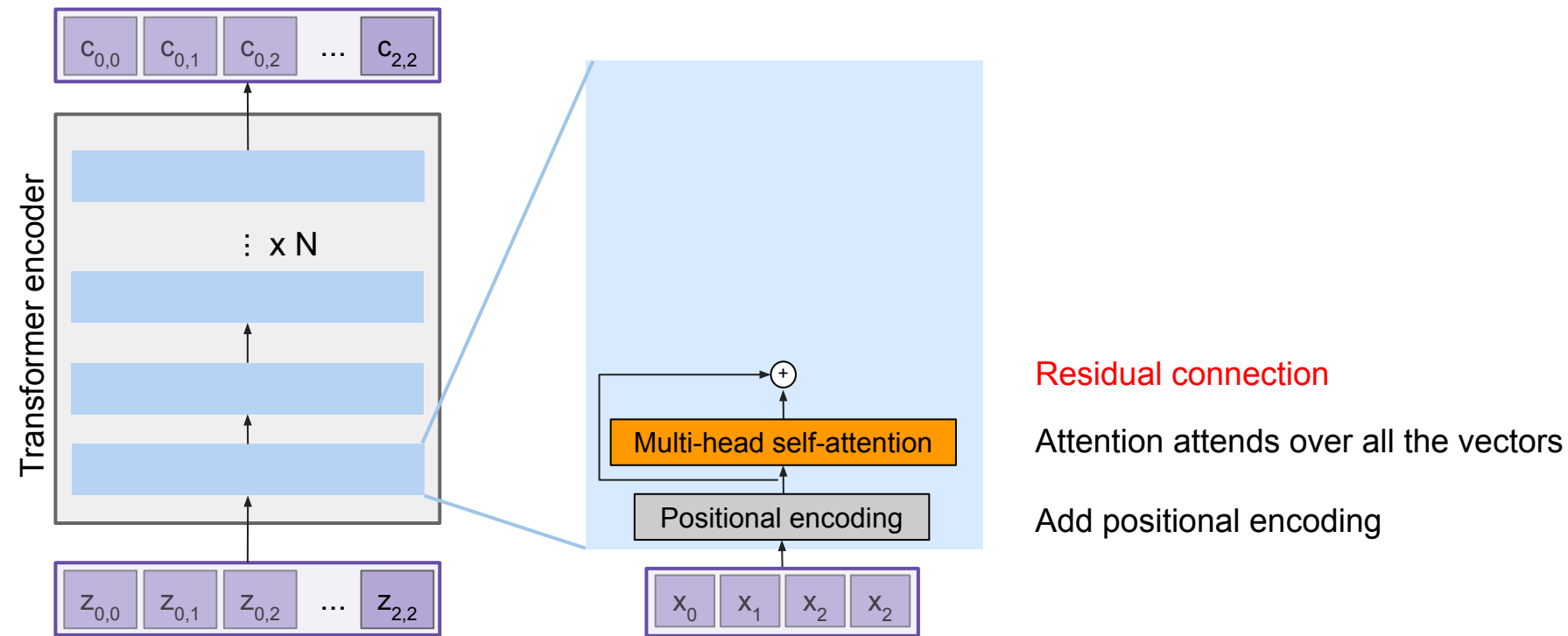
Vaswani et al, "Attention is all you need", NeurIPS 2017

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

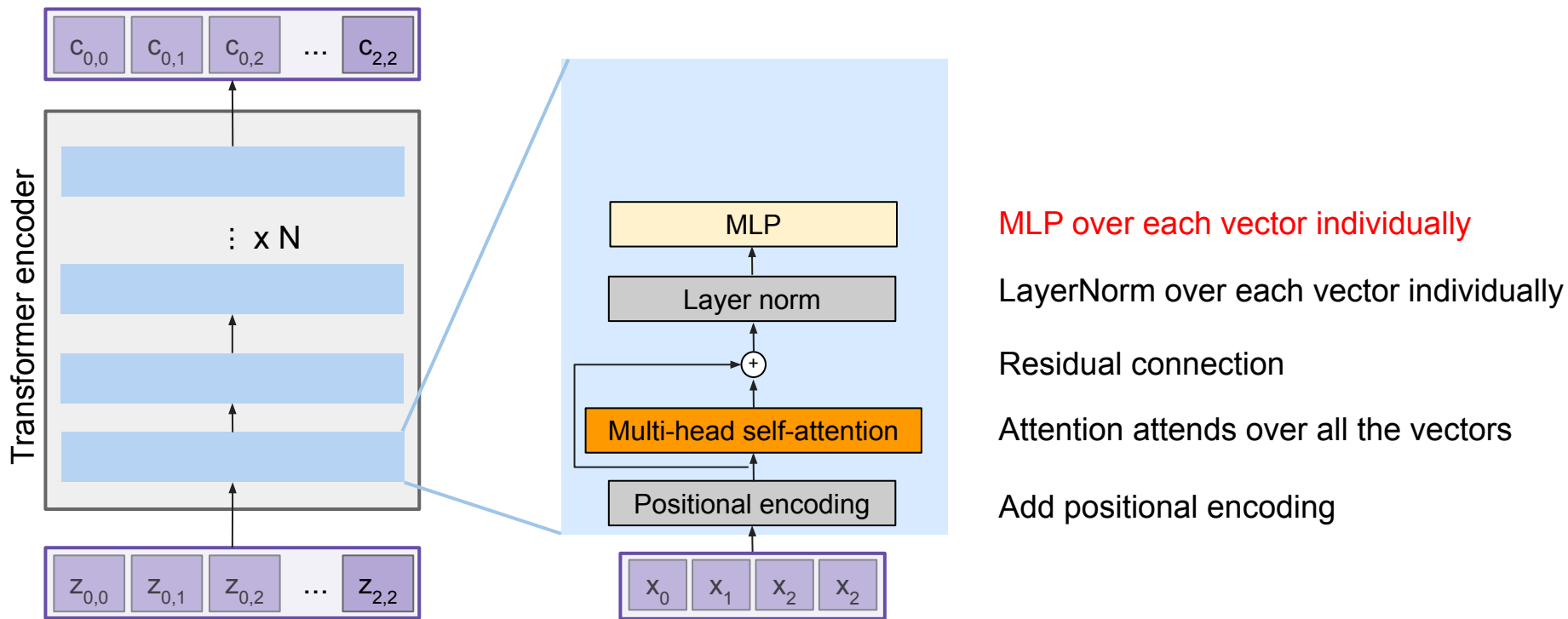LayerNorm over each vector individually

Residual connection

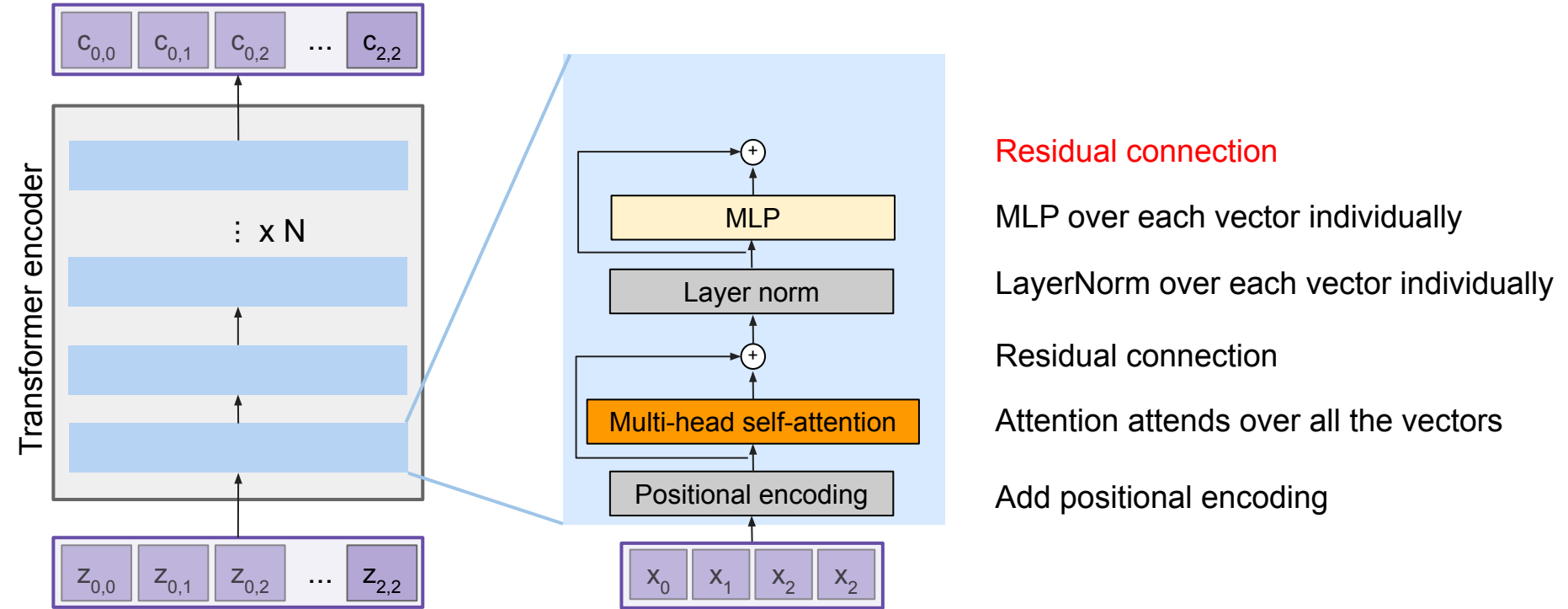Attention attends over all the vectors

Add positional encoding

Vaswani et al, "Attention is all you need", NeurIPS 2017
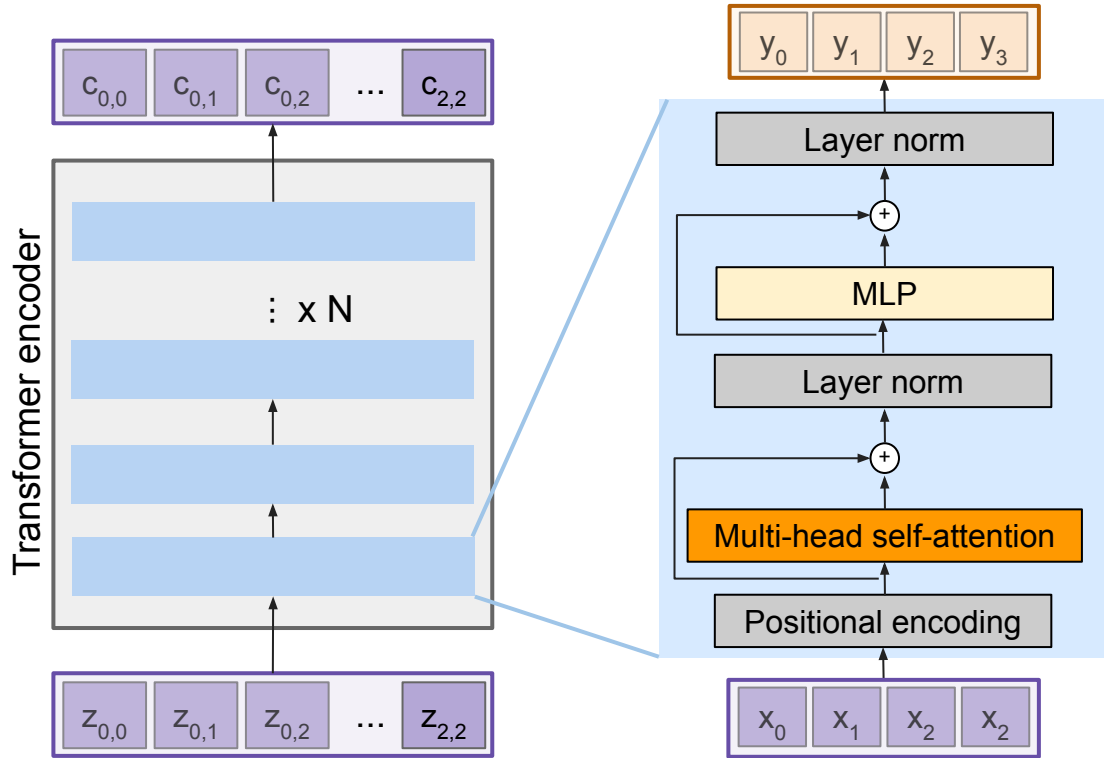
The Transformer encoder block

Transformer Encoder Block:

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

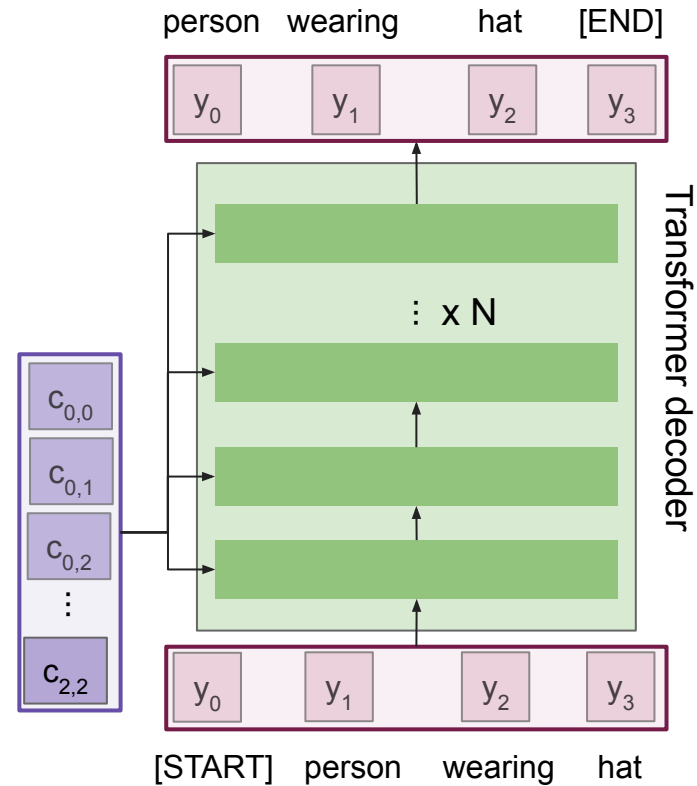Self-attention is the only interaction between vectors.

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

person wearing hat [END]

y_0 y_1 y_2 y_3

Transformer decoder

: x N

$c_{0,0}$

$c_{0,1}$

$c_{0,2}$

:

$c_{2,2}$

y_0 y_1 y_2 y_3

[START] person wearing hat

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
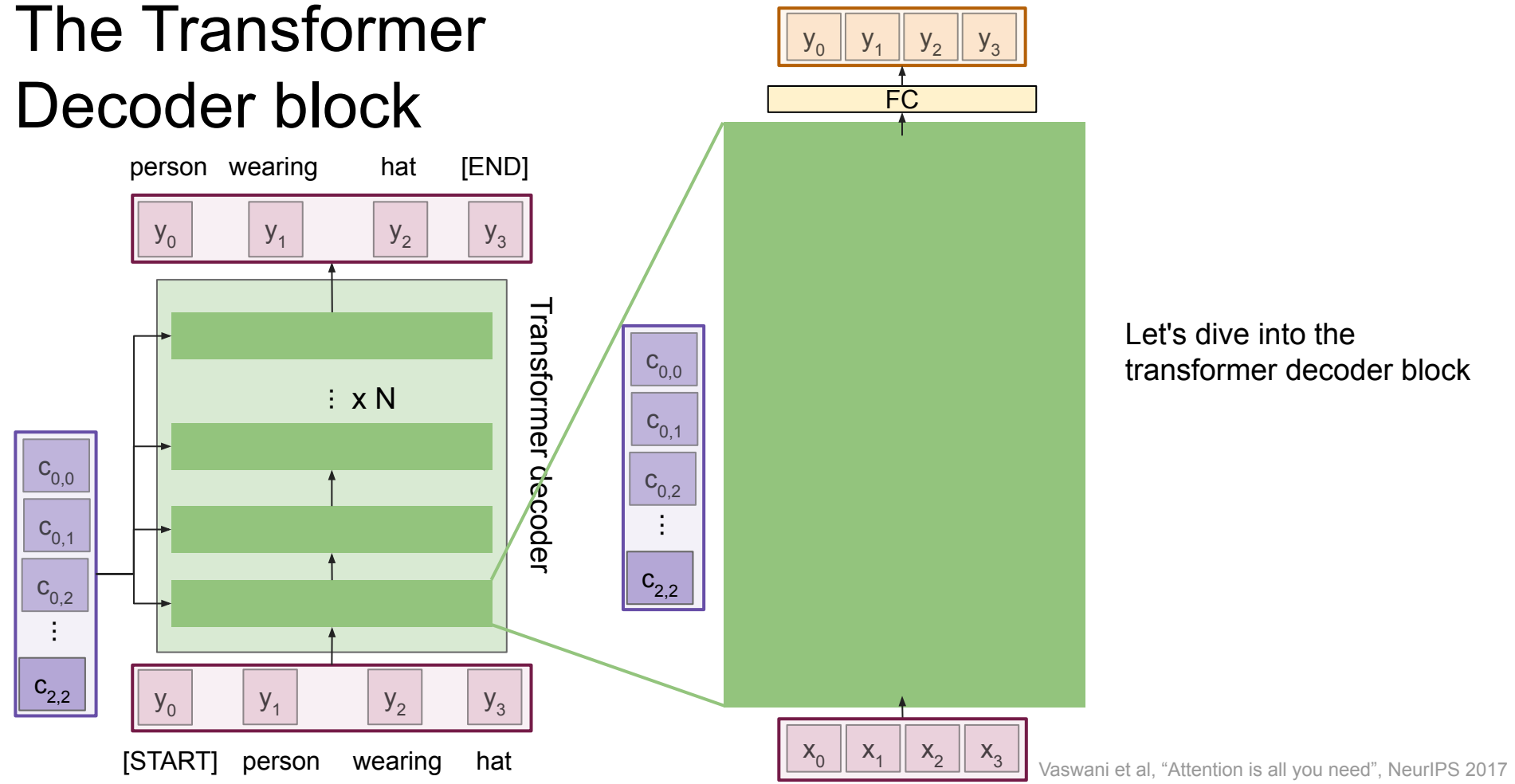
The Transformer Decoder block
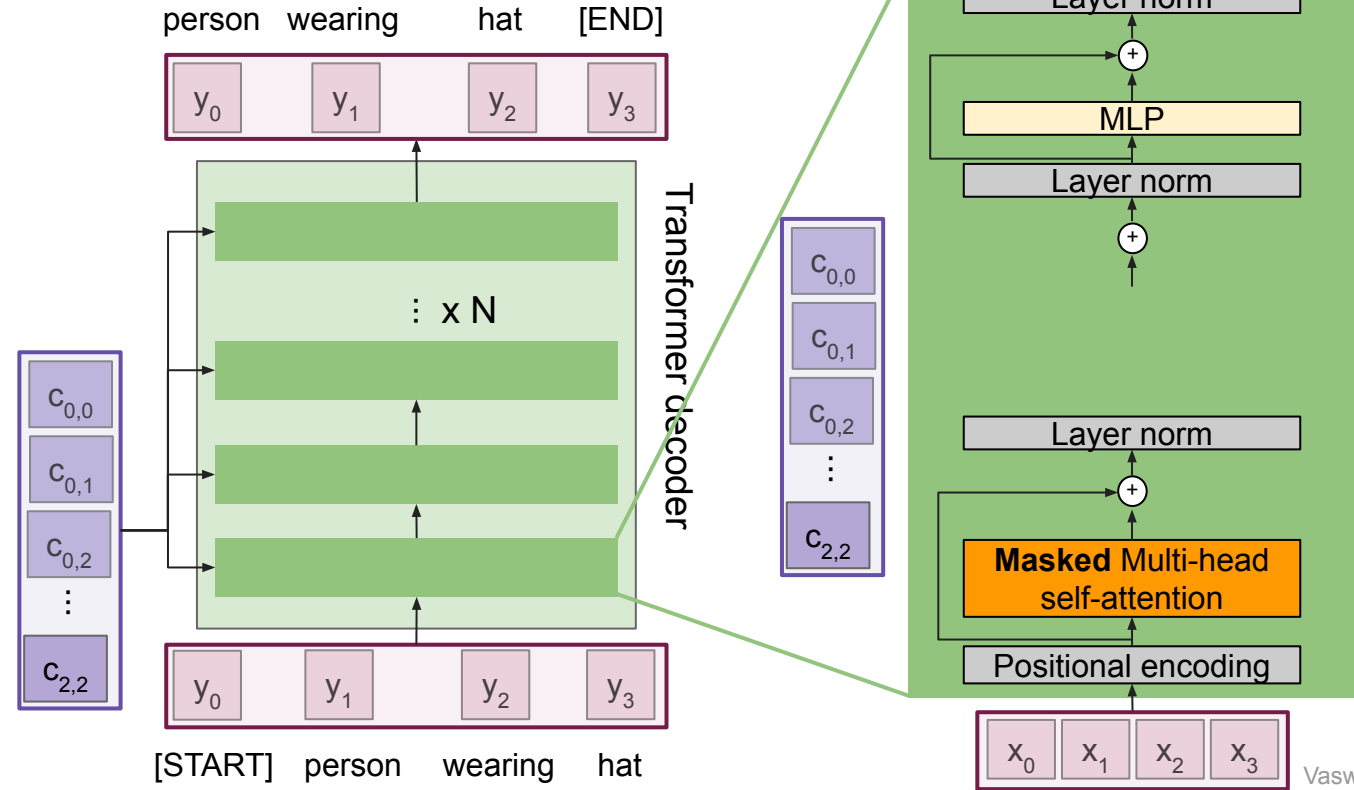
Most of the network is the same the transformer encoder.

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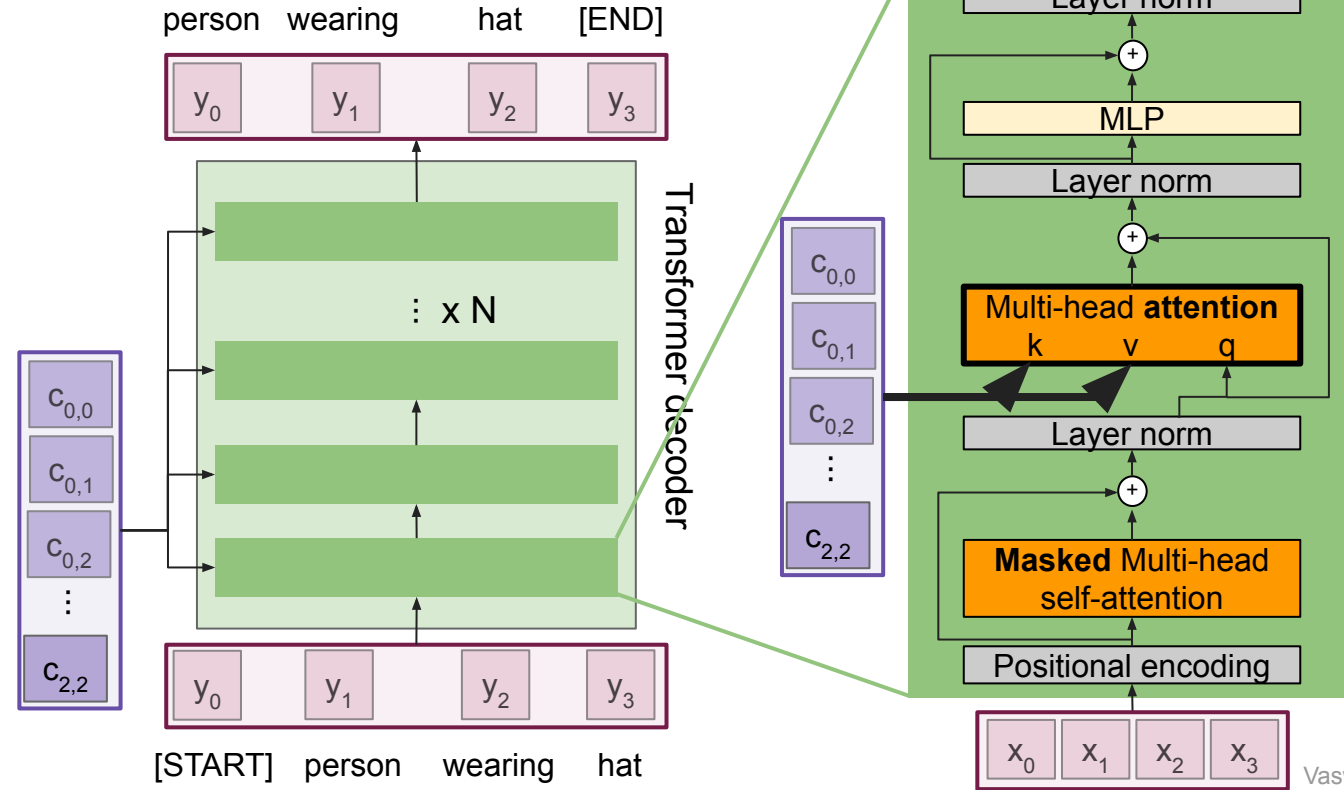
Multi-head attention block attends over the transformer encoder outputs.

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

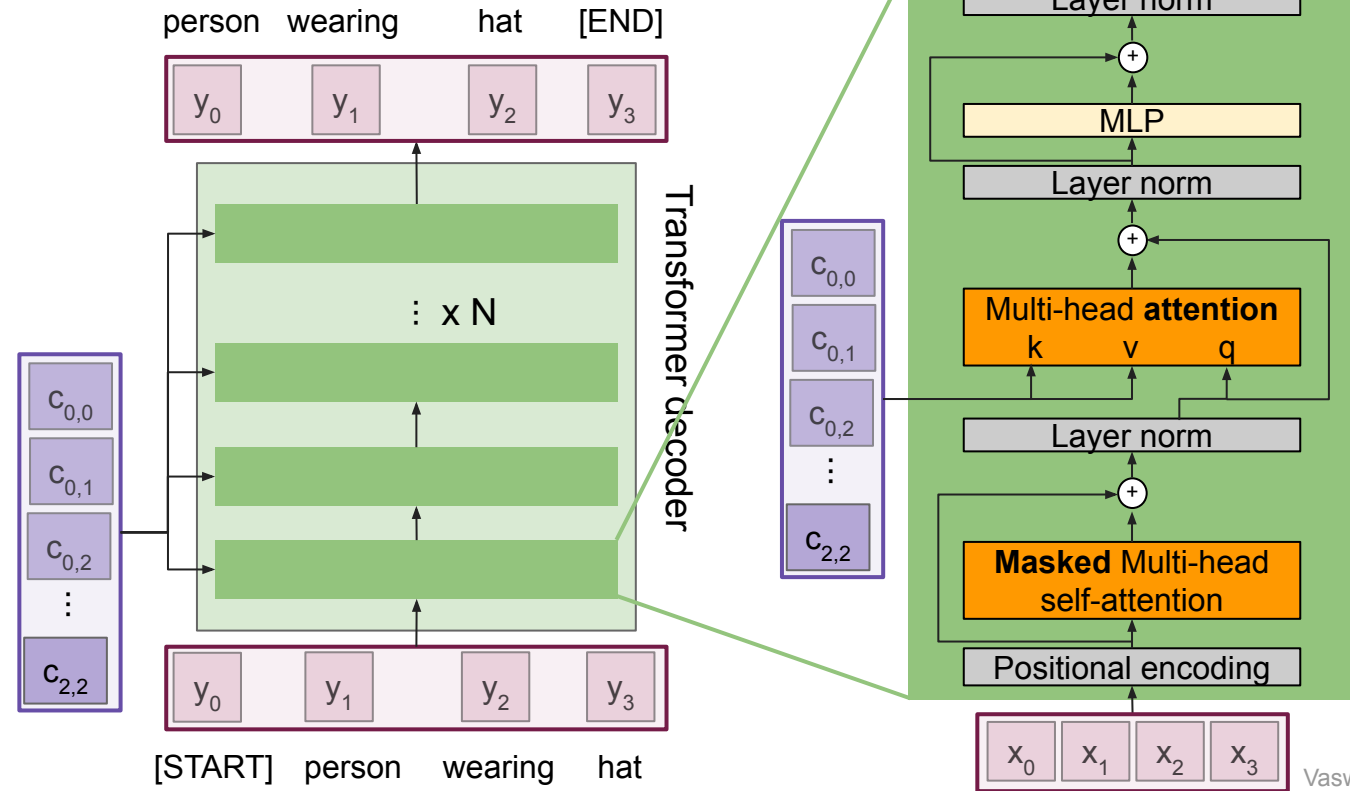person   wearing   hat   [END]

y_0 y_1 y_2 y_3

x N

Transformer decoder

$c_{0,0}$
$c_{0,1}$
$c_{0,2}$
⋮
$c_{2,2}$

y_0 y_1 y_2 y_3

[START]   person   wearing   hat

y_0 y_1 y_2 y_3

FC

Layer norm

+

MLP

Layer norm

+

$c_{0,0}$
$c_{0,1}$
$c_{0,2}$
⋮
$c_{2,2}$

Multi-head **attention**
k   v   q

Layer norm

+

Masked Multi-head self-attention

Positional encoding

x_0 x_1 x_2 x_3

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
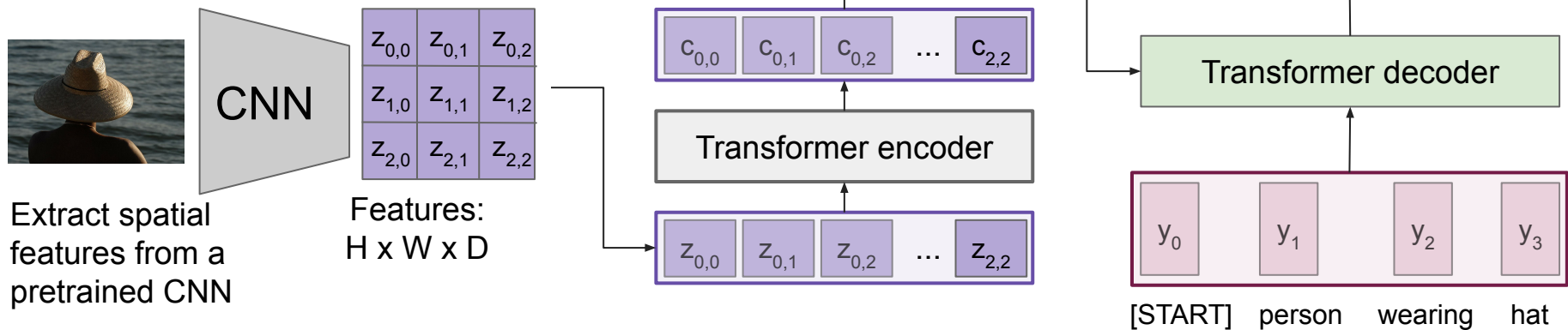
- **No recurrence at all**

Image Captioning using transformers

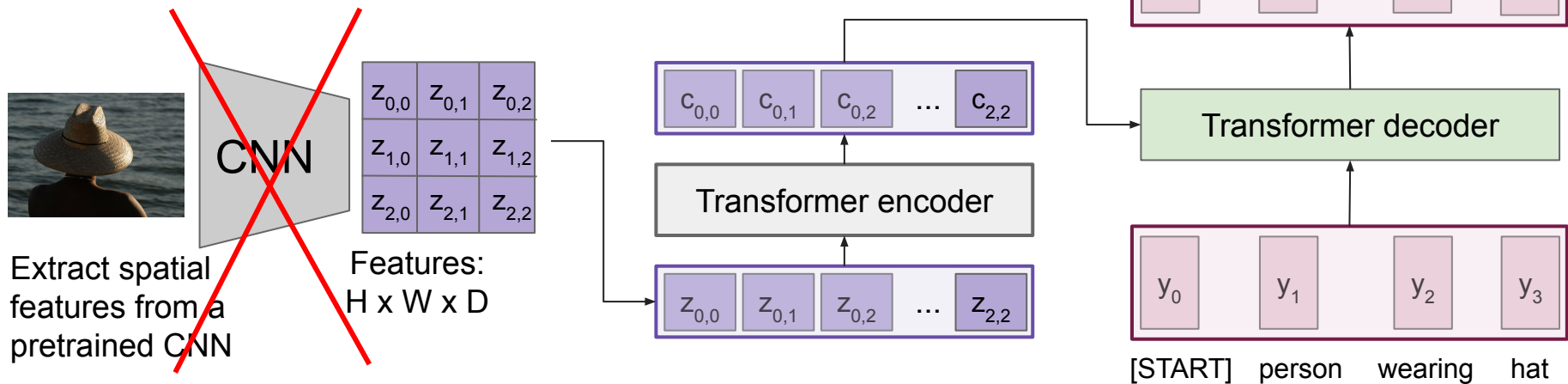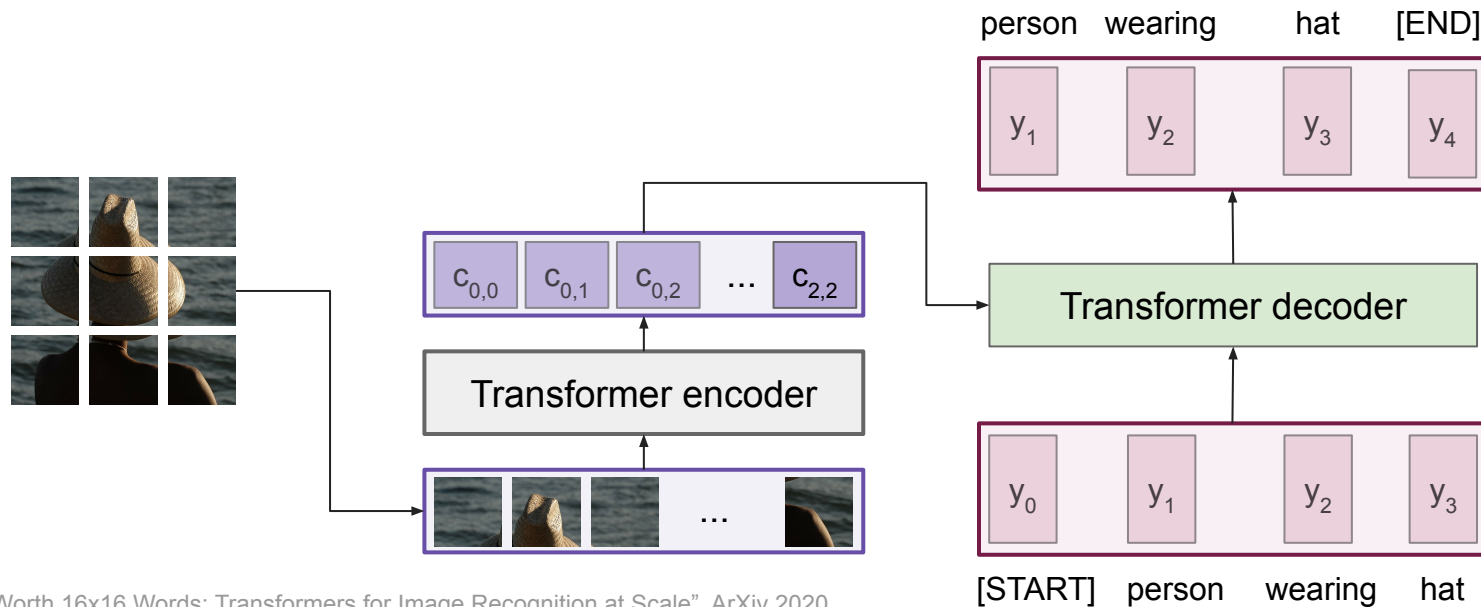- **Perhaps we don't need convolutions at all?**

Image Captioning using ONLY transformers

- **Transformers from pixels to language**

person   wearing   hat   [END]

y_1 y_2 y_3 y_4

$c_{0,0}$ $c_{0,1}$ $c_{0,2}$ … $c_{2,2}$

Transformer encoder

Transformer decoder

y_0 y_1 y_2 y_3

[START]   person   wearing   hat

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
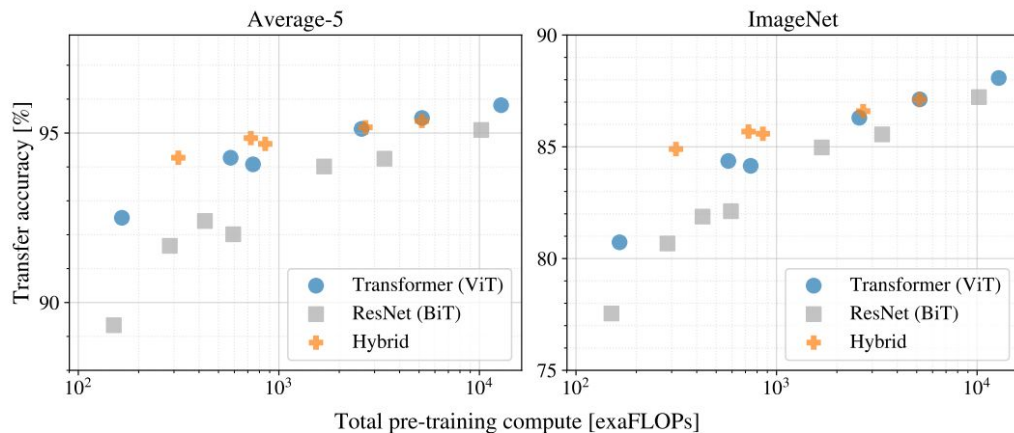
Image Captioning using ONLY transformers

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
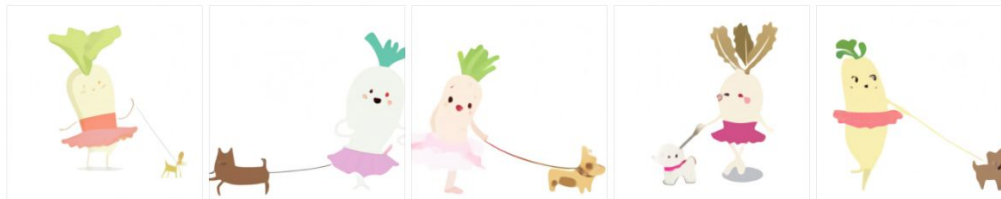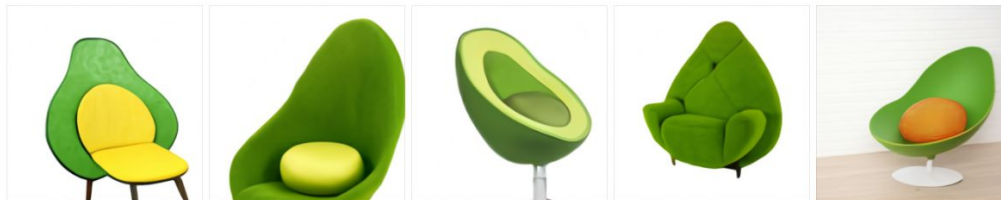
Edit prompt or view more images↓

TEXT PROMPT

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

AI-GENERATED IMAGES

Edit prompt or view more images↓

link to more examples

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