

Lecture 11:

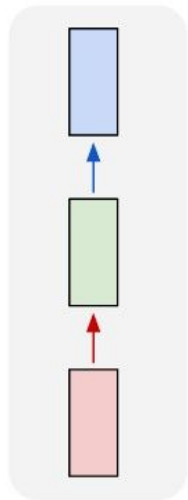
Attention and Transformers

Administrative

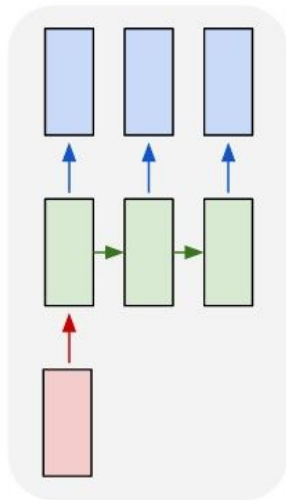
- Project proposal grades released.
Check feedback on GradeScope!
- Project milestone due May 7th Saturday 11:59pm PT
Check Ed and course website for requirements

Last Time: Recurrent Neural Networks

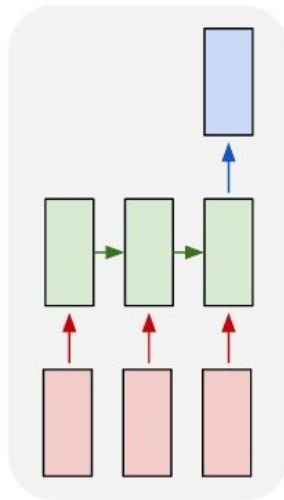
one to one



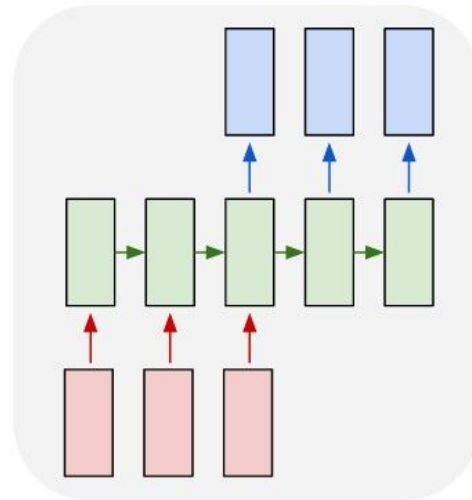
one to many



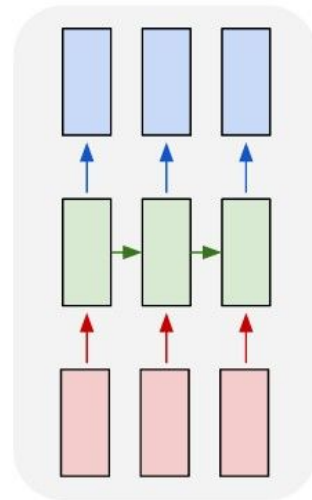
many to one



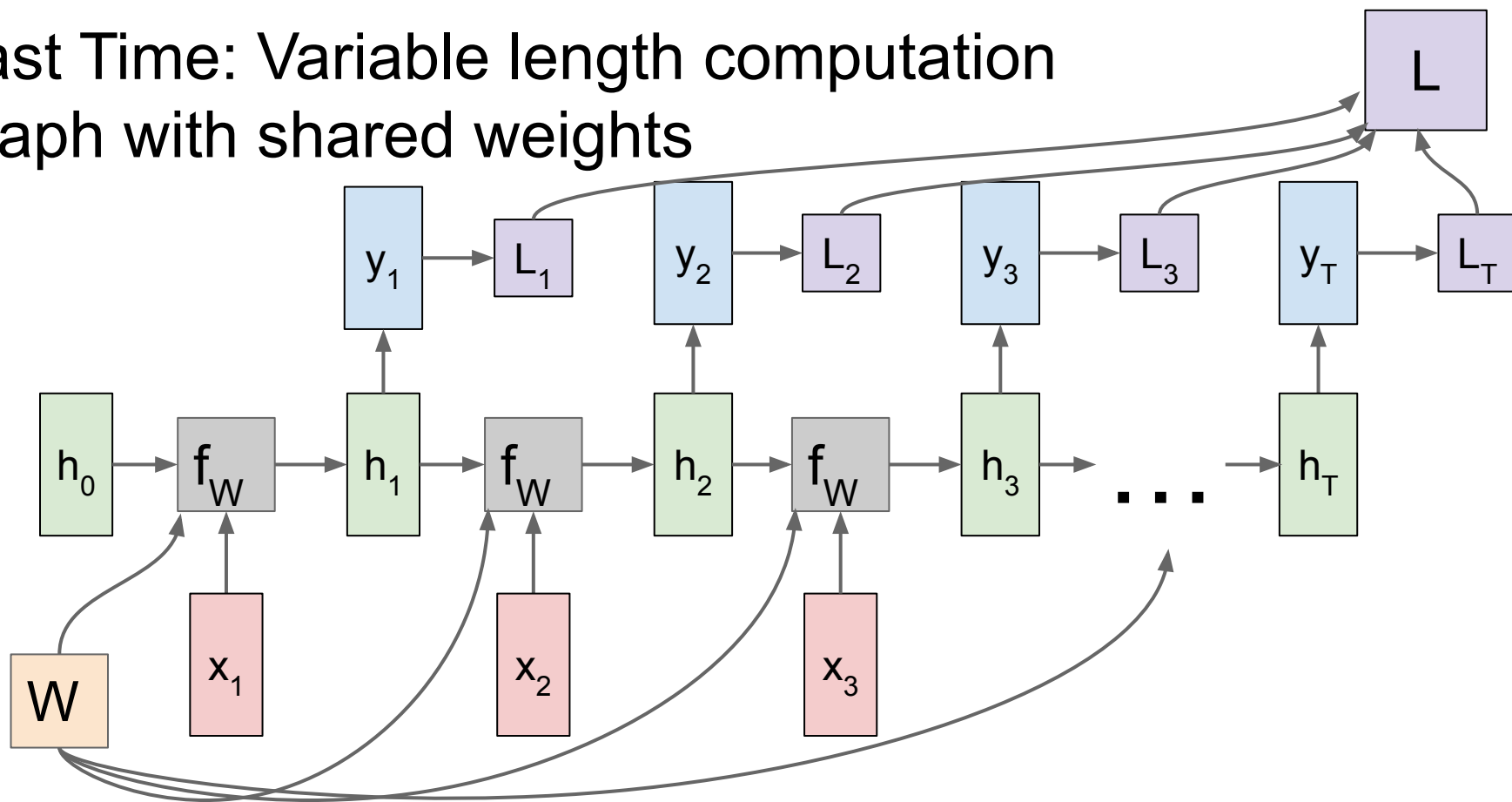
many to many



many to many



Last Time: Variable length computation graph with shared weights

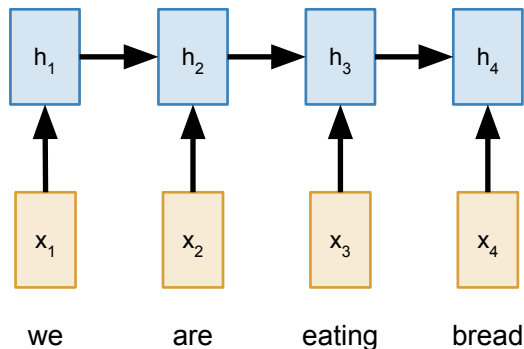


Sequence to Sequence with RNNs

Input: Sequence x_1, \dots, x_T

Output: Sequence y_1, \dots, y_T

Encoder: $h_t = f_W(x_t, h_{t-1})$

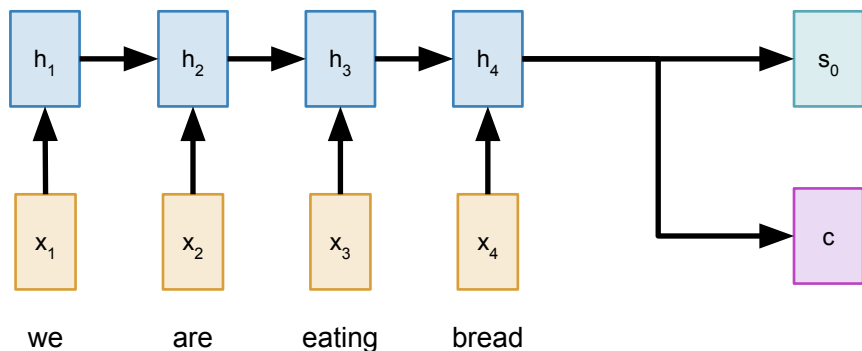


Sequence to Sequence with RNNs

Input: Sequence x_1, \dots, x_T

Output: Sequence y_1, \dots, y_T

Encoder: $h_t = f_W(x_t, h_{t-1})$ From final hidden state predict:
Initial decoder state s_0
Context vector c (often $c=h_T$)



Sequence to Sequence with RNNs

Input: Sequence x_1, \dots, x_T

Output: Sequence y_1, \dots, y_T

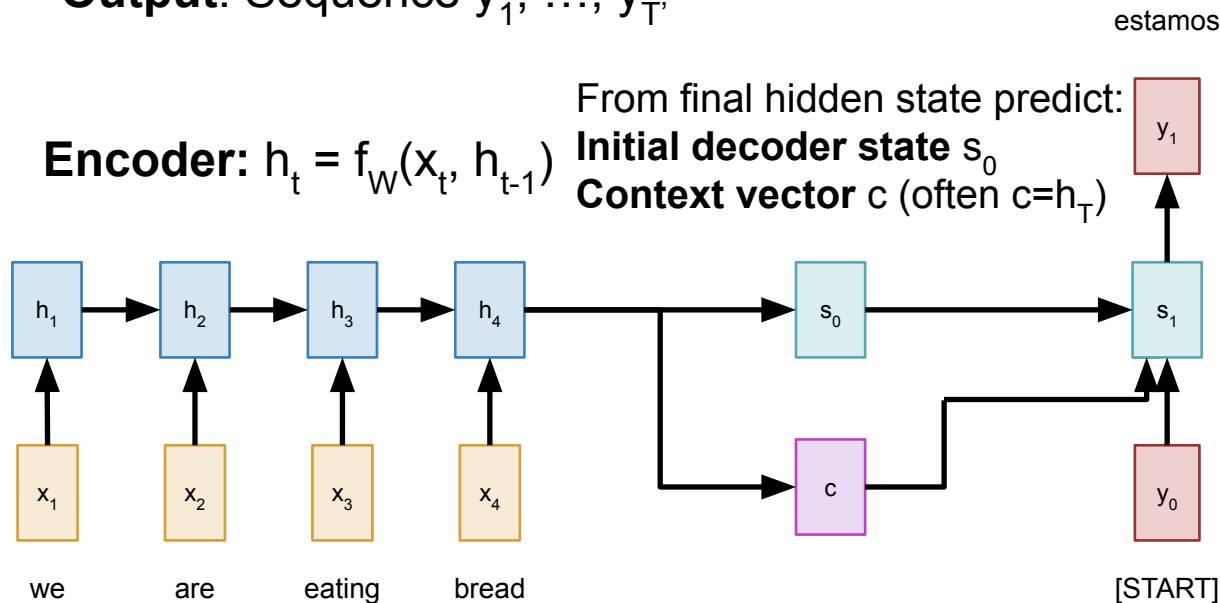
Decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c)$

Encoder: $h_t = f_W(x_t, h_{t-1})$

From final hidden state predict:

Initial decoder state s_0

Context vector c (often $c=h_T$)



Sequence to Sequence with RNNs

Input: Sequence x_1, \dots, x_T

Output: Sequence y_1, \dots, y_T

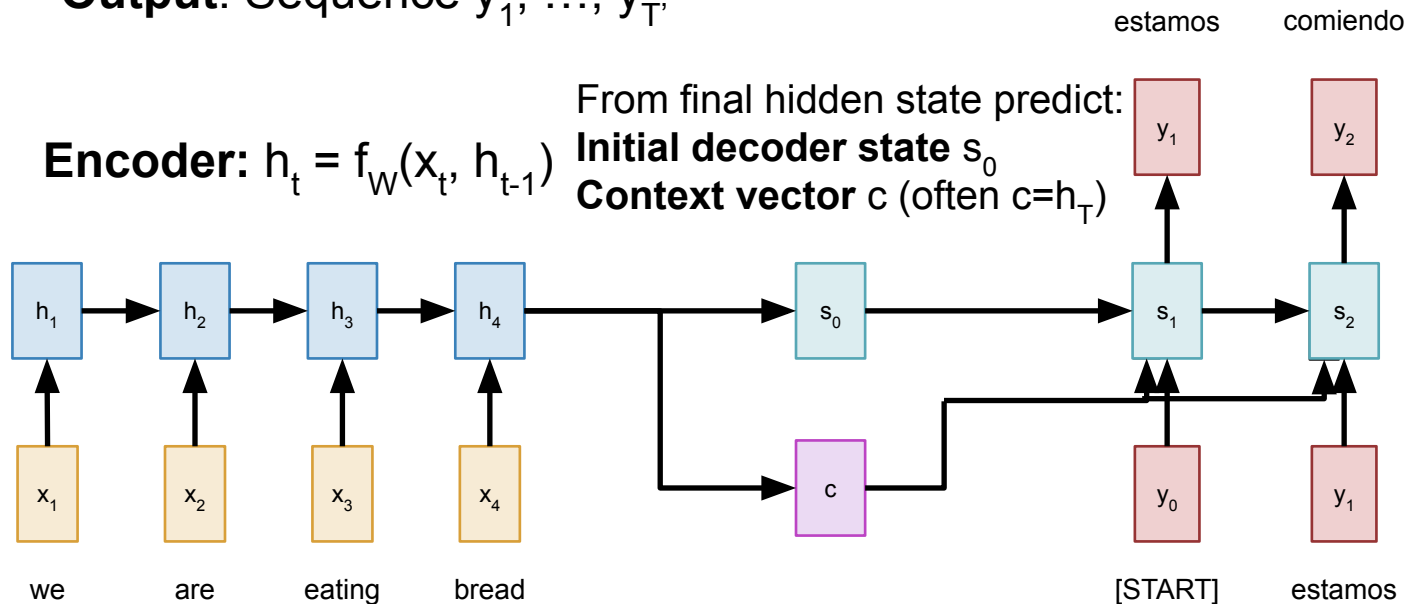
Decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c)$

Encoder: $h_t = f_W(x_t, h_{t-1})$

From final hidden state predict:

Initial decoder state s_0

Context vector c (often $c=h_T$)



Sequence to Sequence with RNNs

Input: Sequence x_1, \dots, x_T

Output: Sequence y_1, \dots, y_T

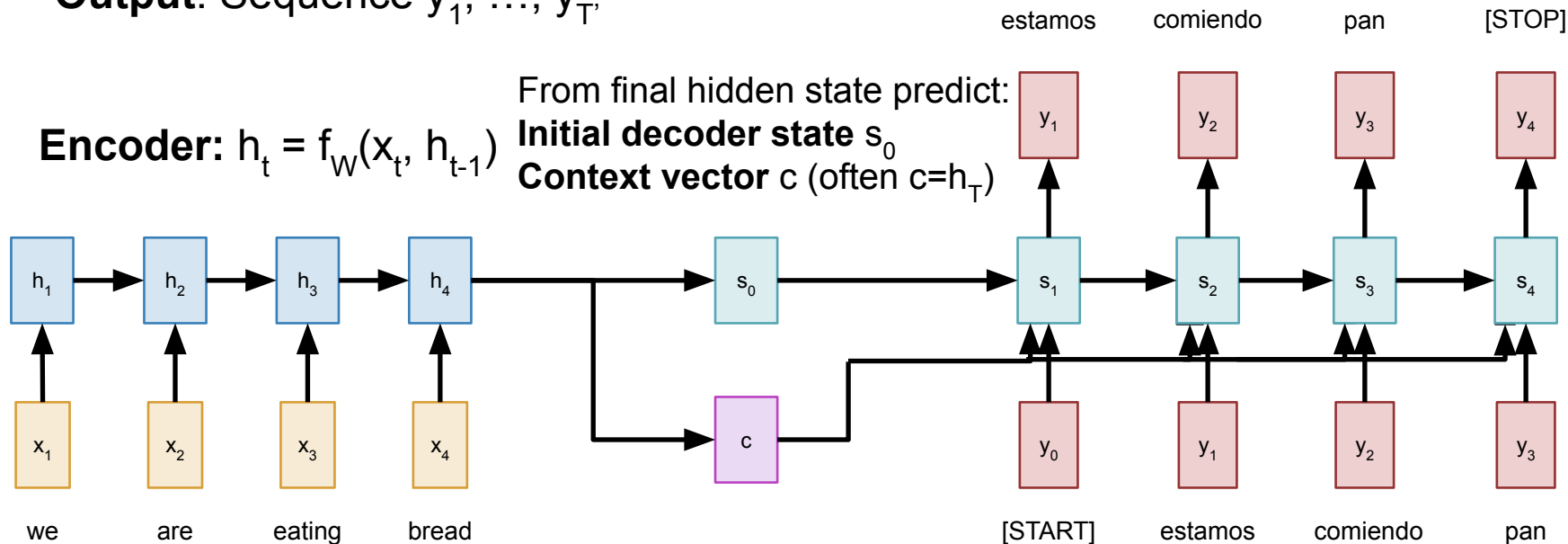
Decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c)$

Encoder: $h_t = f_W(x_t, h_{t-1})$

From final hidden state predict:

Initial decoder state s_0

Context vector c (often $c=h_T$)



Sequence to Sequence with RNNs

Input: Sequence x_1, \dots, x_T

Output: Sequence y_1, \dots, y_T

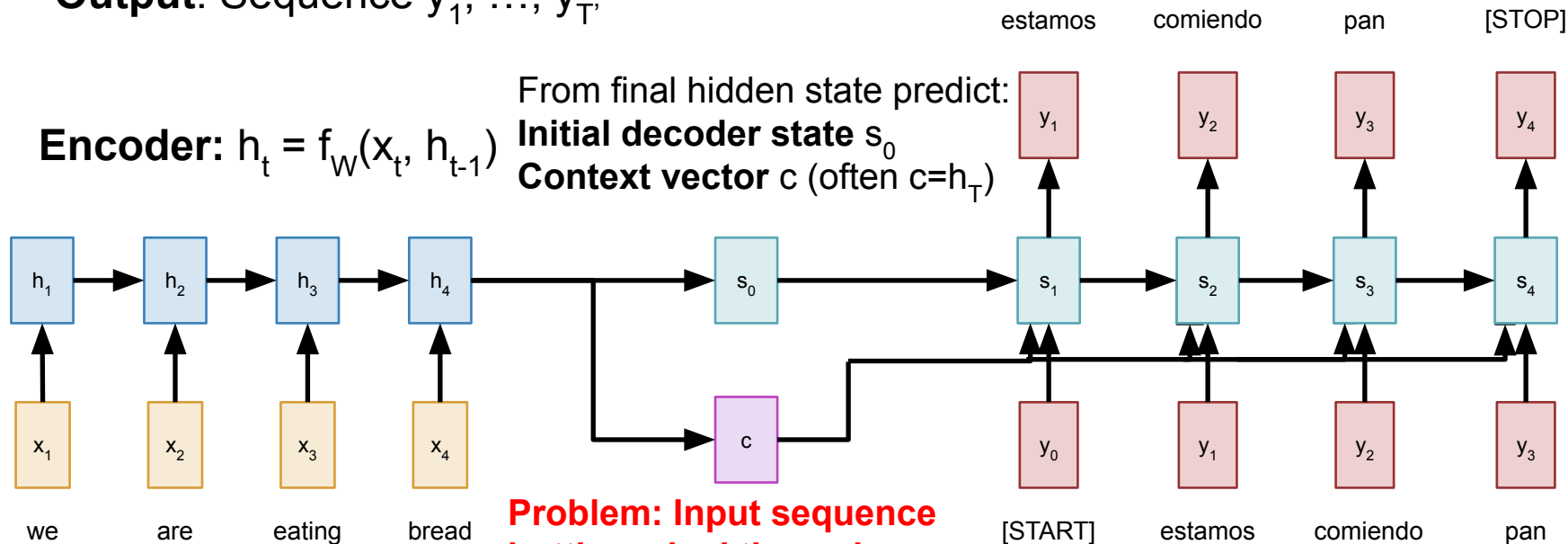
Decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c)$

Encoder: $h_t = f_W(x_t, h_{t-1})$

From final hidden state predict:

Initial decoder state s_0

Context vector c (often $c=h_T$)



Problem: Input sequence bottlenecked through fixed-sized vector. What if

$T=1000?$

Sutskever et al, "Sequence to sequence learning with neural networks", *NeurIPS* 2014

Sequence to Sequence with RNNs

Input: Sequence x_1, \dots, x_T

Output: Sequence y_1, \dots, y_T

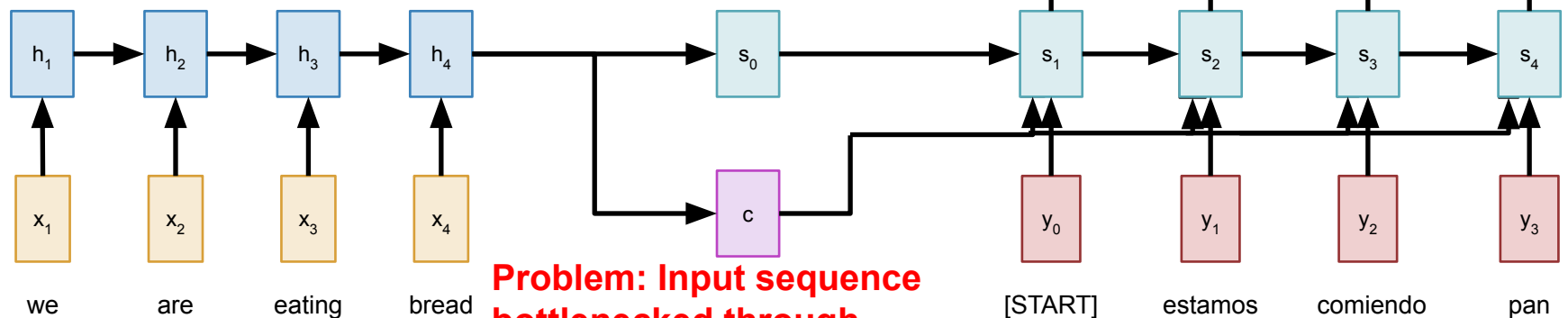
Decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c)$

Encoder: $h_t = f_W(x_t, h_{t-1})$

From final hidden state predict:

Initial decoder state s_0

Context vector c (often $c=h_T$)



Problem: Input sequence bottlenecked through fixed-sized vector. What if $T=1000$?

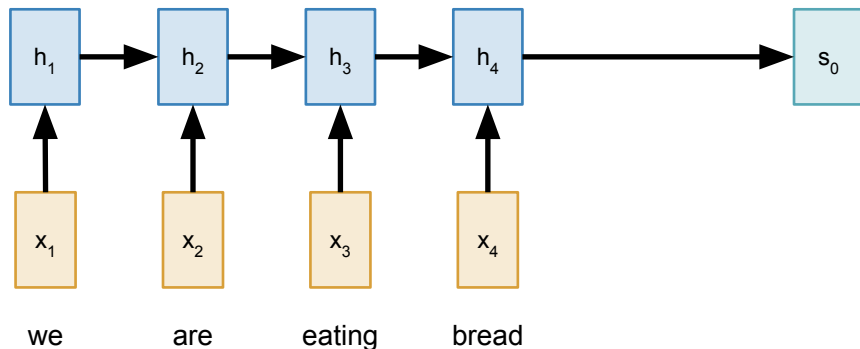
Idea: use new context vector at each step of decoder!

Sequence to Sequence with RNNs and Attention

Input: Sequence x_1, \dots, x_T

Output: Sequence y_1, \dots, y_T

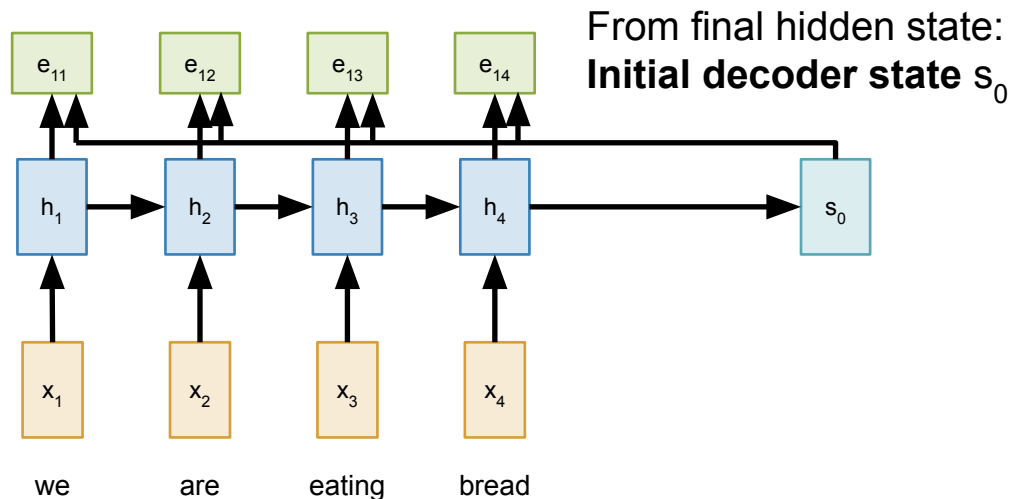
Encoder: $h_t = f_W(x_t, h_{t-1})$ From final hidden state:
Initial decoder state s_0



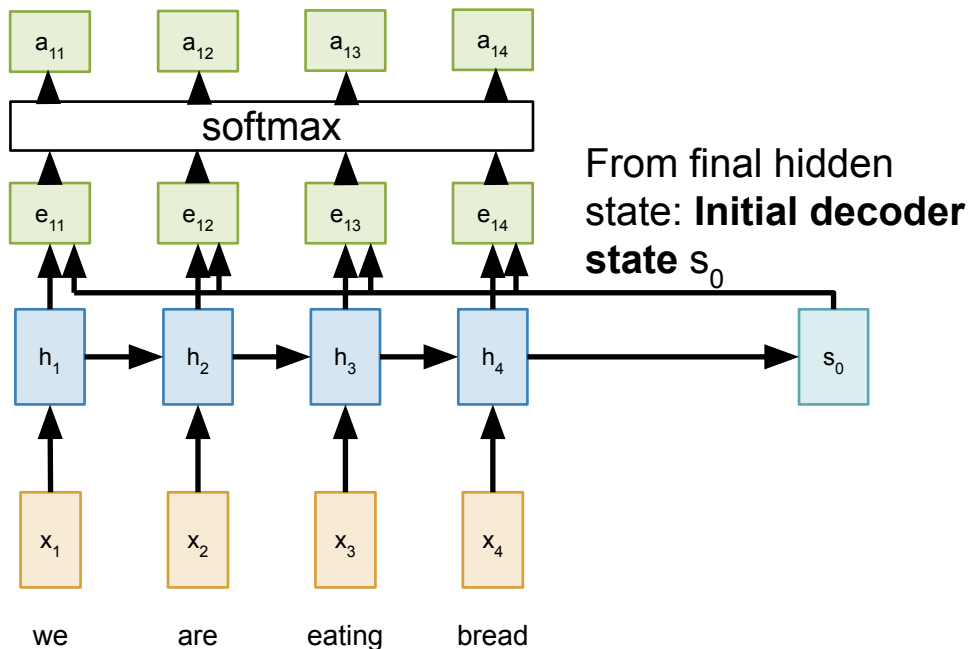
Sequence to Sequence with RNNs and **Attention**

Compute (scalar) **alignment scores**

$$e_{t,i} = f_{\text{att}}(s_{t-1}, h_i) \quad (f_{\text{att}} \text{ is an MLP})$$



Sequence to Sequence with RNNs and **Attention**



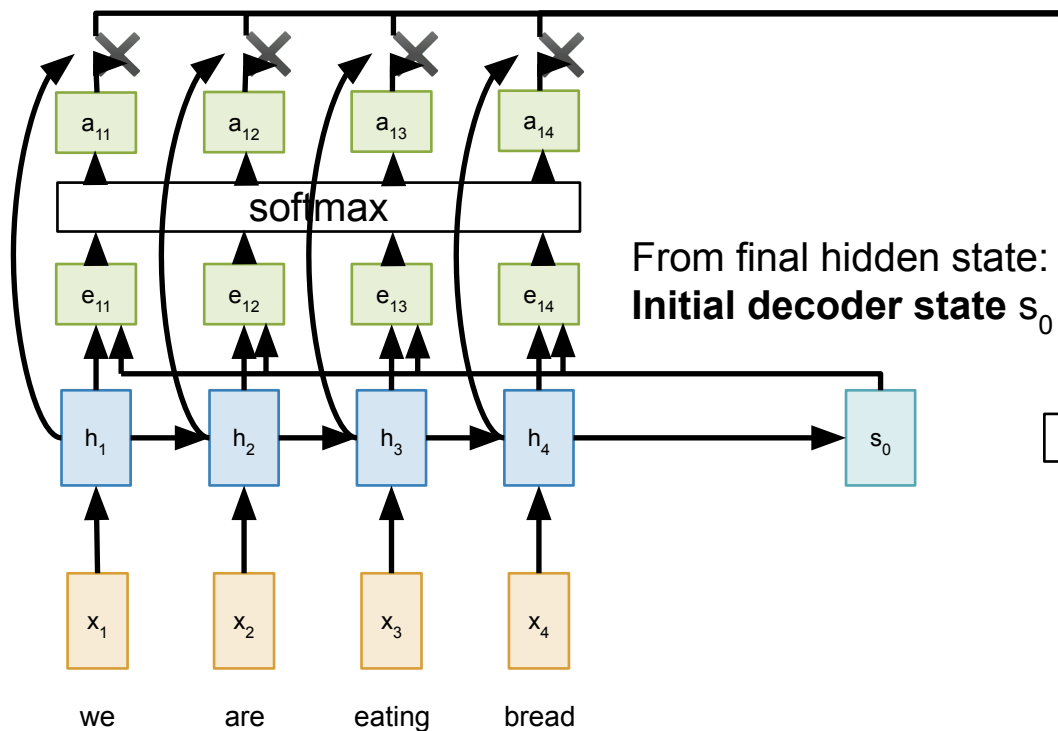
Compute (scalar) **alignment scores**

$$e_{t,i} = f_{\text{att}}(s_{t-1}, h_i) \quad (f_{\text{att}} \text{ is an MLP})$$

Normalize alignment scores
to get **attention weights**

$$0 < a_{t,i} < 1 \quad \sum_i a_{t,i} = 1$$

Sequence to Sequence with RNNs and Attention



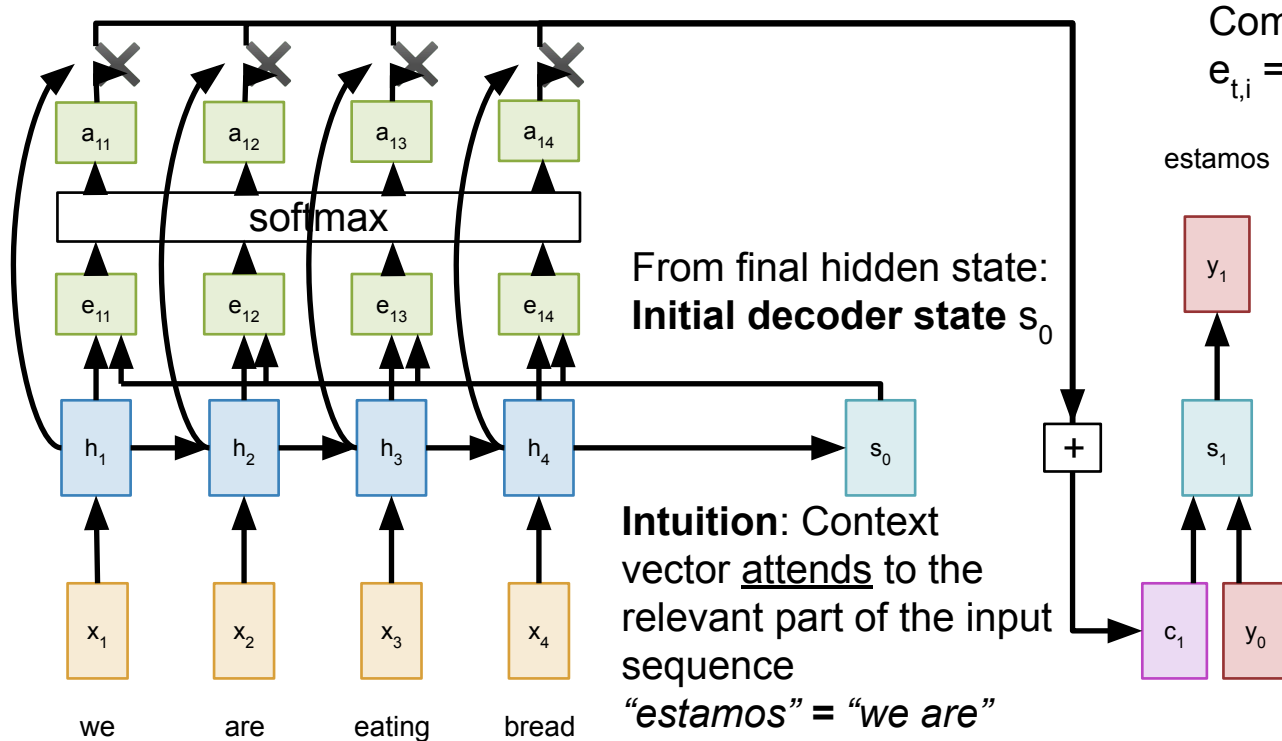
Compute (scalar) **alignment scores**
 $e_{t,i} = f_{\text{att}}(s_{t-1}, h_i)$ (f_{att} is an MLP)

Normalize alignment scores
to get **attention weights**
 $0 < a_{t,i} < 1 \quad \sum_i a_{t,i} = 1$

Compute context vector as
linear combination of hidden
states
 $c_t = \sum_i a_{t,i} h_i$

[START]

Sequence to Sequence with RNNs and Attention



From final hidden state:
Initial decoder state s_0

Intuition: Context vector attends to the relevant part of the input sequence

"estamos" = "we are"
so maybe $a_{11}=a_{12}=0.45$,
 $a_{13}=a_{14}=0.05$

Compute (scalar) **alignment scores**
 $e_{t,i} = f_{\text{att}}(s_{t-1}, h_i)$ (f_{att} is an MLP)

Normalize alignment scores to get **attention weights**
 $0 < a_{t,i} < 1 \quad \sum_i a_{t,i} = 1$

Compute context vector as linear combination of hidden states

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

Use context vector in decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c_t)$

estamos

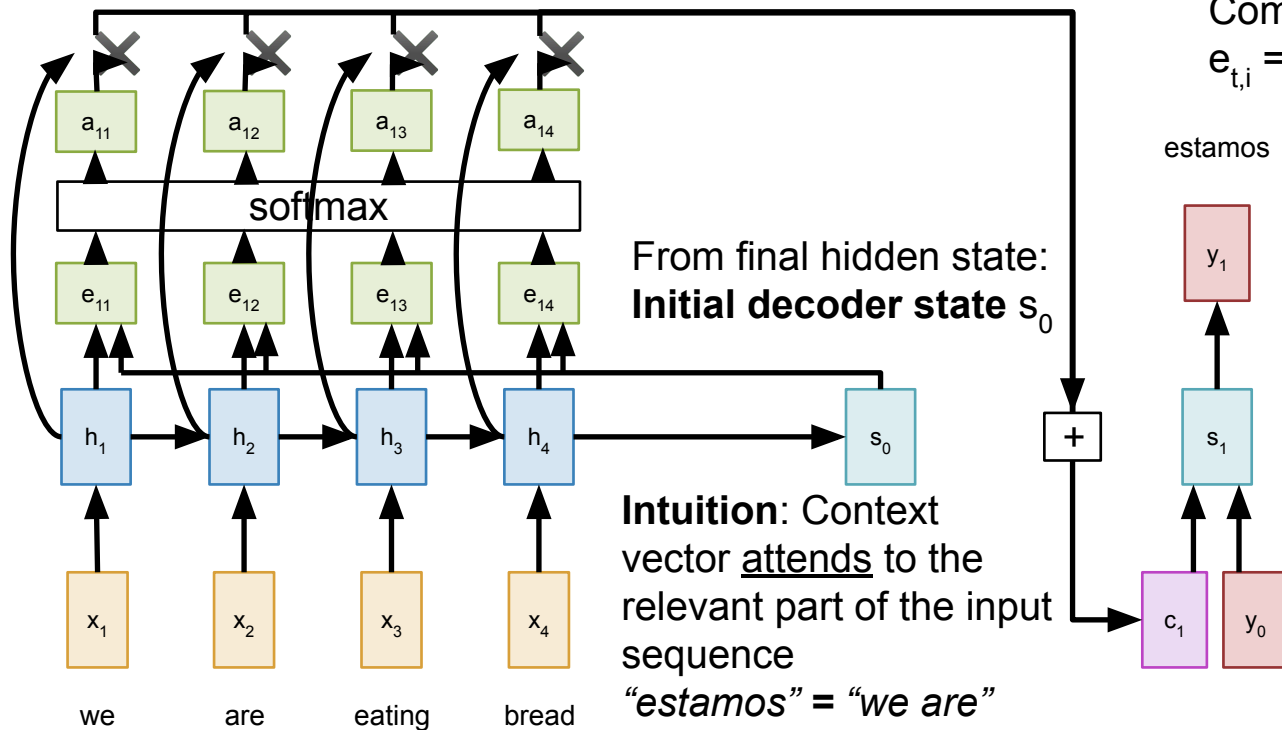
y_1

s_1

c_1 y_0

[START]

Sequence to Sequence with RNNs and Attention



From final hidden state:
Initial decoder state s_0

Intuition: Context vector attends to the relevant part of the input sequence

"estamos" = "we are"
so maybe $a_{11}=a_{12}=0.45$,
 $a_{13}=a_{14}=0.05$

Compute (scalar) **alignment scores**
 $e_{t,i} = f_{\text{att}}(s_{t-1}, h_i)$ (f_{att} is an MLP)

Normalize alignment scores to get **attention weights**

$$0 < a_{t,i} < 1 \quad \sum_i a_{t,i} = 1$$

Compute context vector as linear combination of hidden states

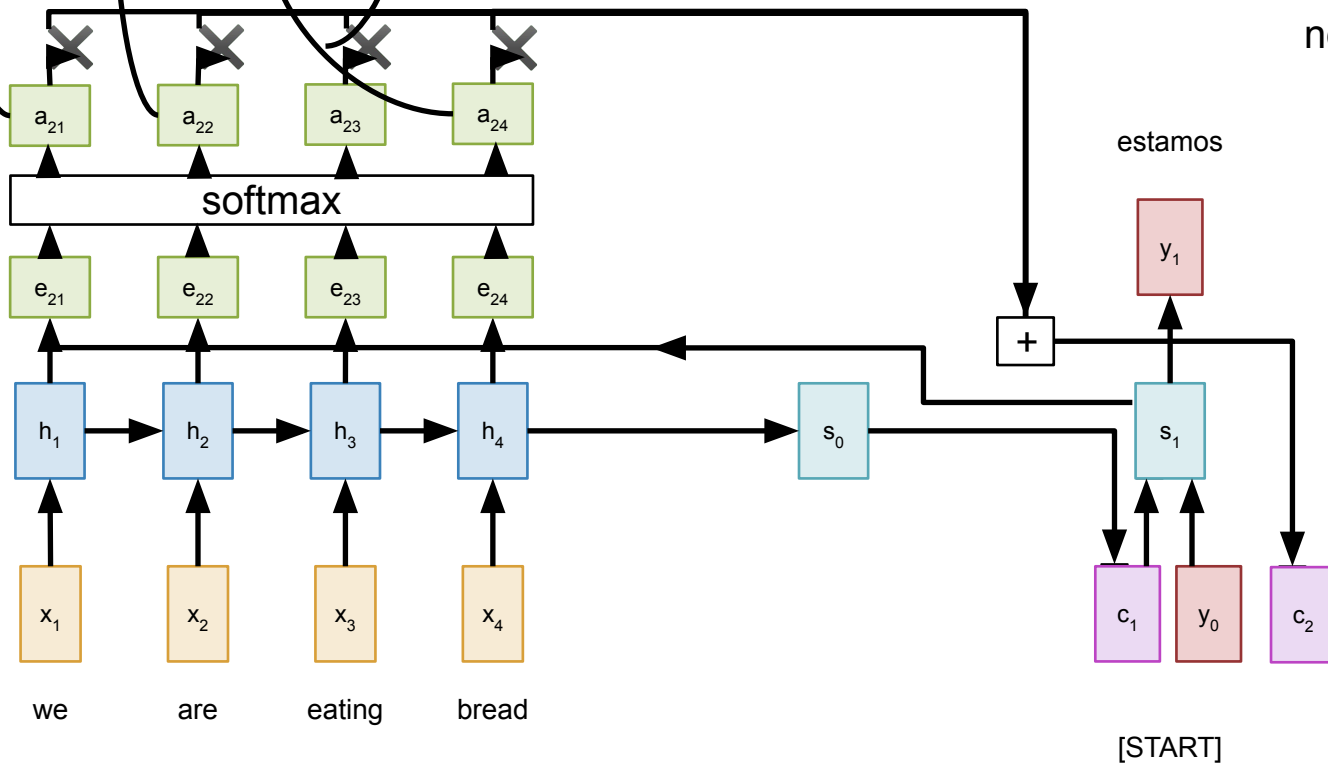
$$c_t = \sum_i a_{t,i} h_i$$

Use context vector in decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c_t)$

This is all differentiable! No supervision on attention weights – backprop through everything

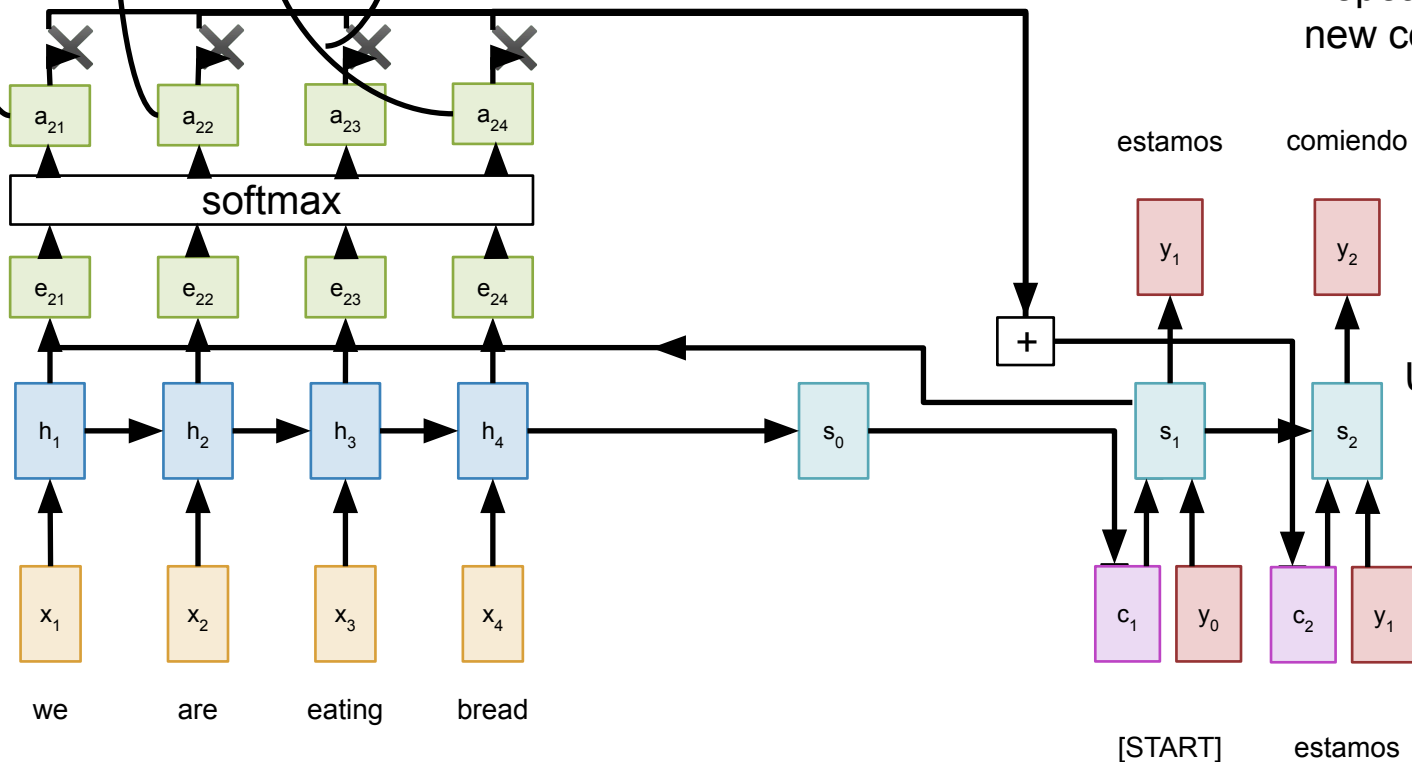
Sequence to Sequence with RNNs and Attention

Repeat: Use s_1 to compute new context vector c_2



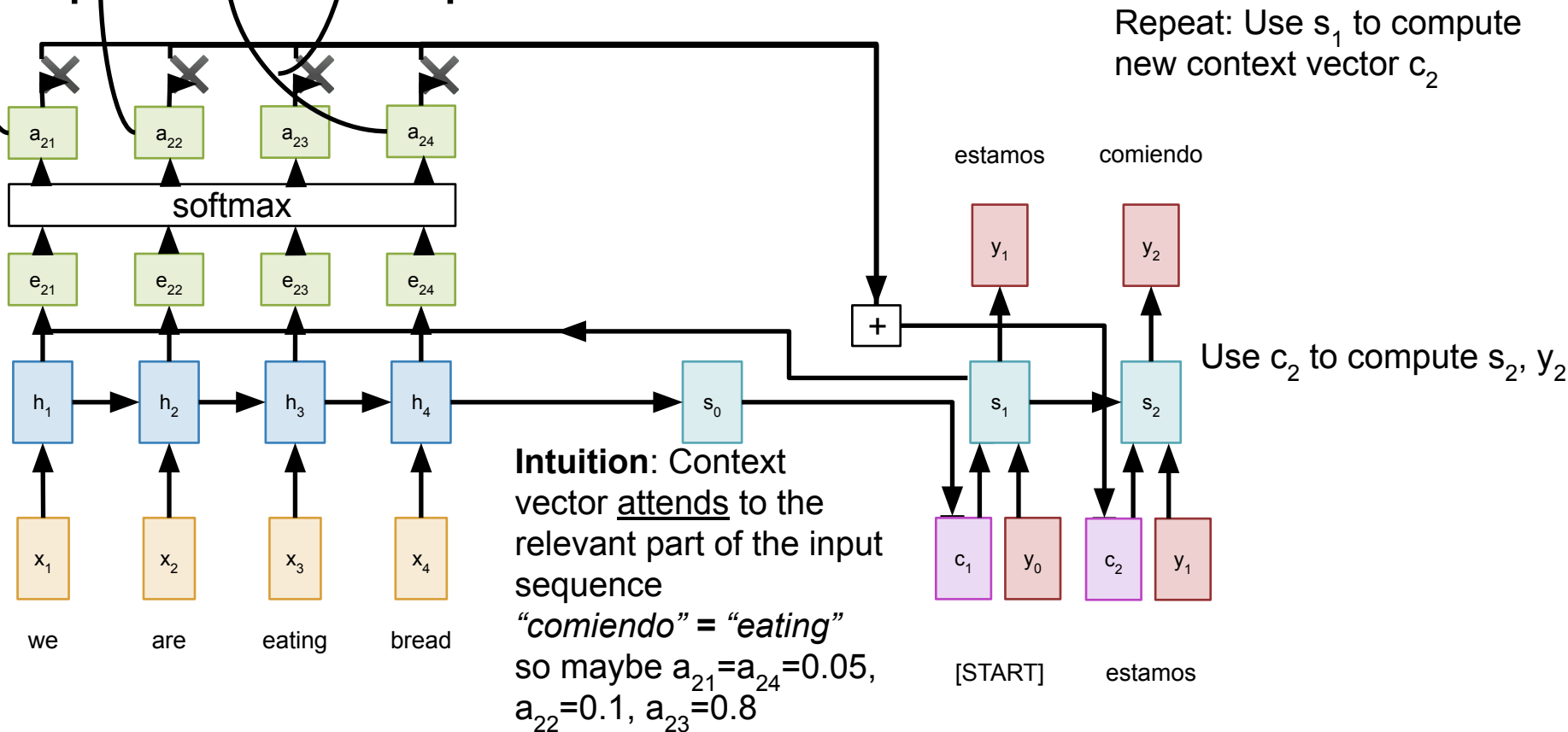
Sequence to Sequence with RNNs and Attention

Repeat: Use s_1 to compute new context vector c_2



Use c_2 to compute s_2, y_2

Sequence to Sequence with RNNs and Attention

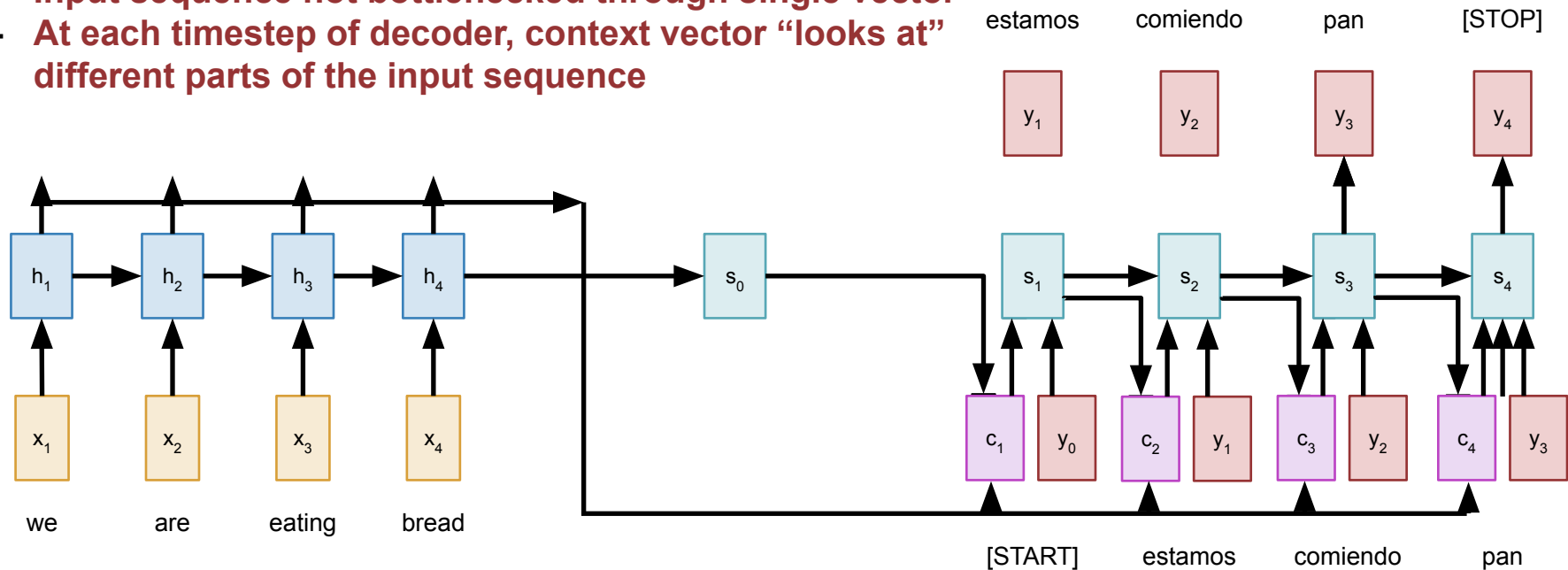


Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Sequence to Sequence with RNNs and Attention

Use a different context vector in each timestep of decoder

- Input sequence not bottlenecked through single vector
- At each timestep of decoder, context vector “looks at” different parts of the input sequence



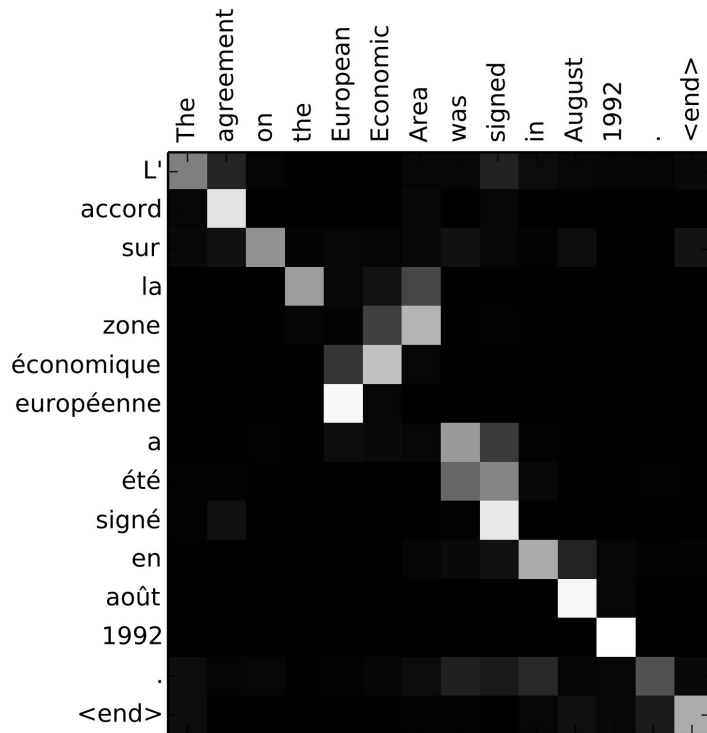
Sequence to Sequence with RNNs and Attention

Example: English to French translation

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.”

Visualize attention weights $a_{t,i}$



Sequence to Sequence with RNNs and Attention

Example: English to French translation

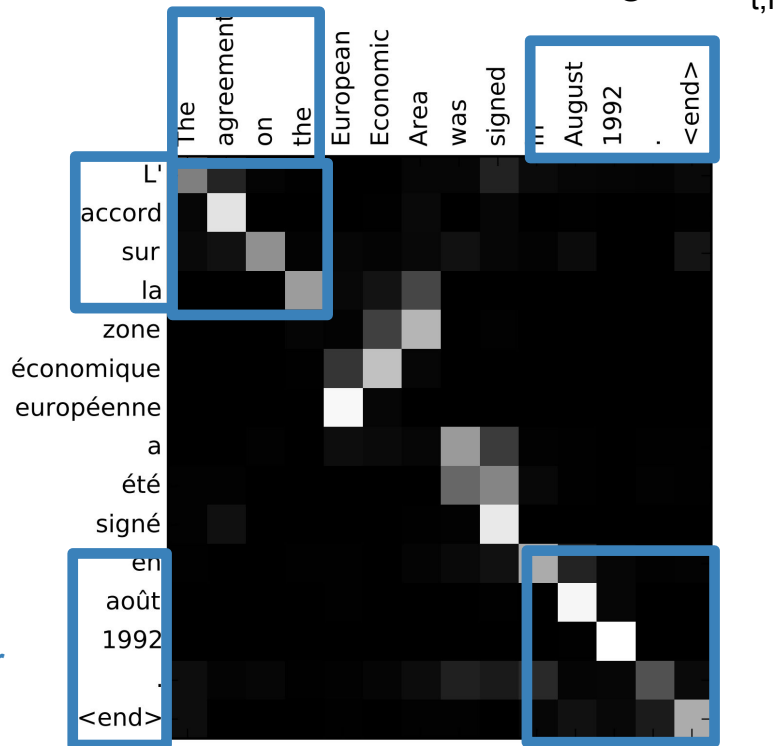
Input: “**The agreement on the** European Economic Area was signed **in August 1992.**”

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

Diagonal attention means words correspond in order

Diagonal attention means words correspond in order

Visualize attention weights $a_{t,i}$



Sequence to Sequence with RNNs and Attention

Example: English to French translation

Input: “The agreement on the European Economic Area was signed in August 1992.”

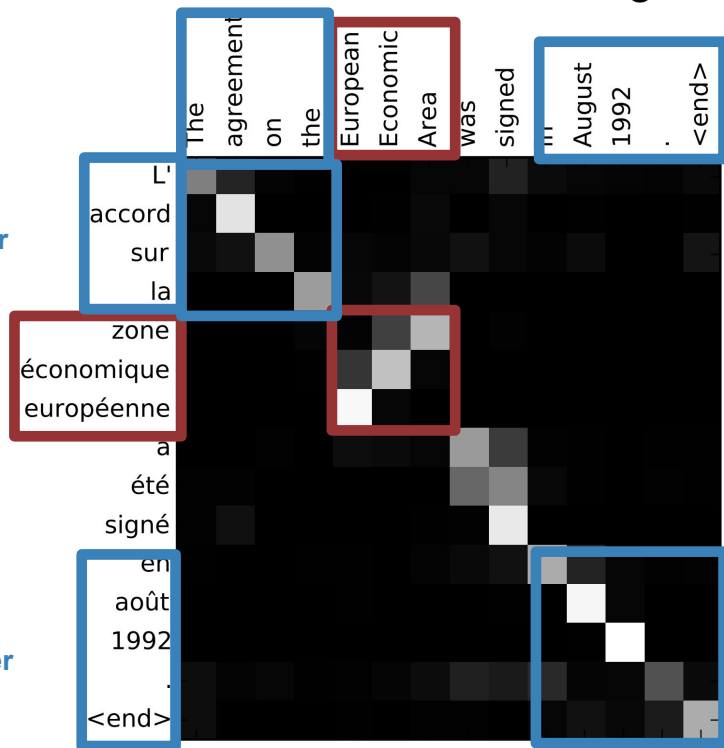
Output: “L'accord sur la zone économique européenne a été signé en août 1992.”

Diagonal attention means words correspond in order

Attention figures out different word orders

Diagonal attention means words correspond in order

Visualize attention weights $a_{t,i}$



Sequence to Sequence with RNNs and Attention

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

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

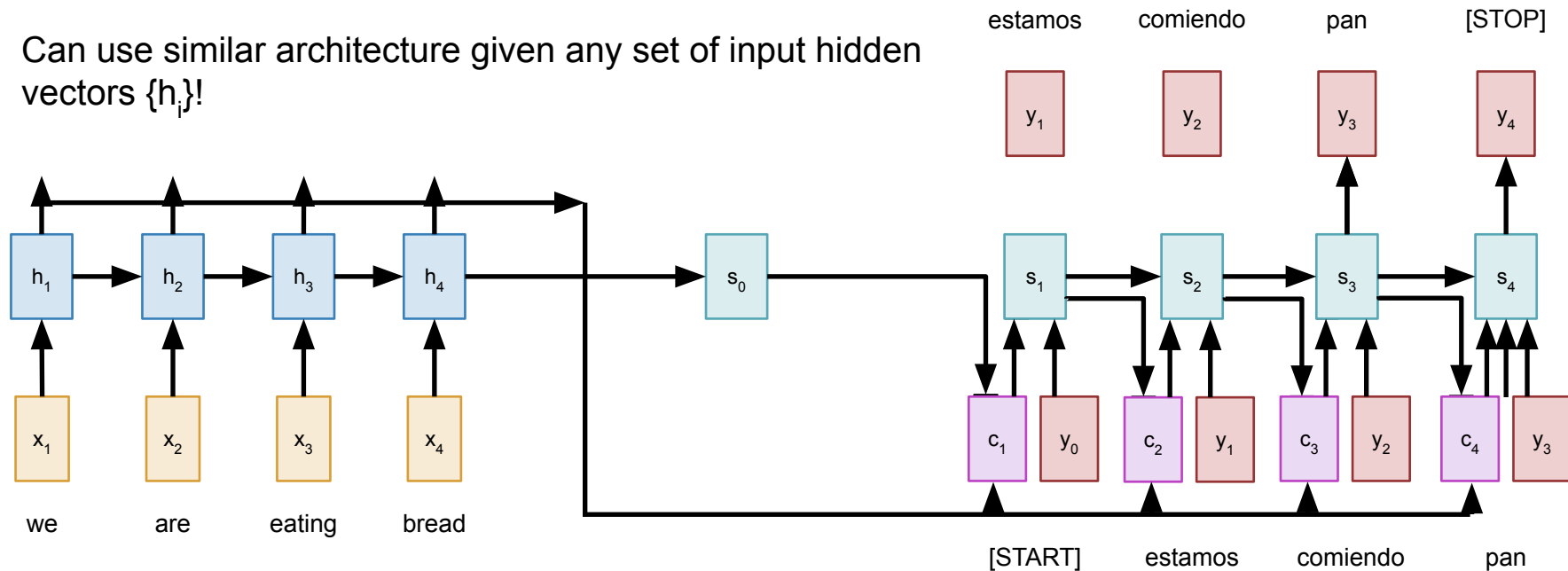
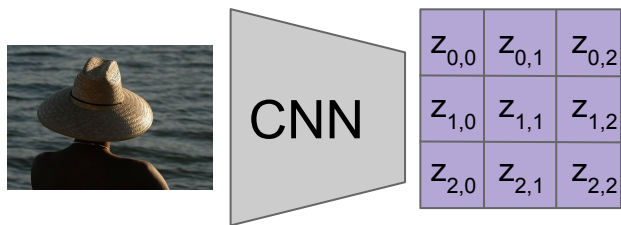


Image Captioning using spatial features

Input: Image I

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



Extract spatial features from a pretrained CNN

Features:
 $H \times W \times D$

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

Image Captioning using spatial features

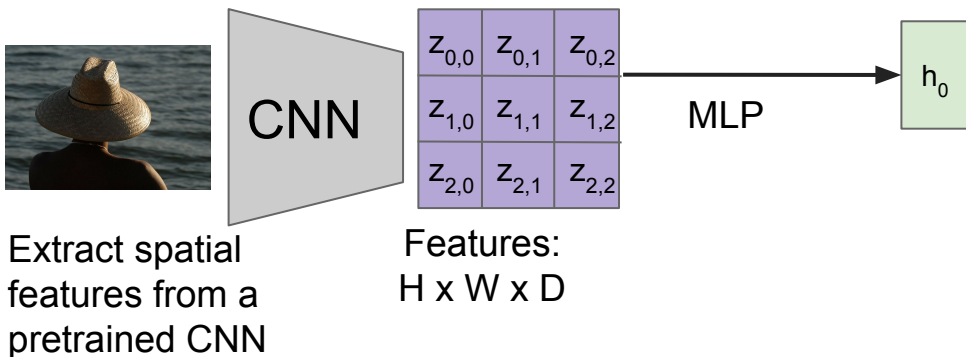
Input: Image I

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

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

where \mathbf{z} is spatial CNN features

$f_w(\cdot)$ is an MLP



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

Image Captioning using spatial features

Input: Image I

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

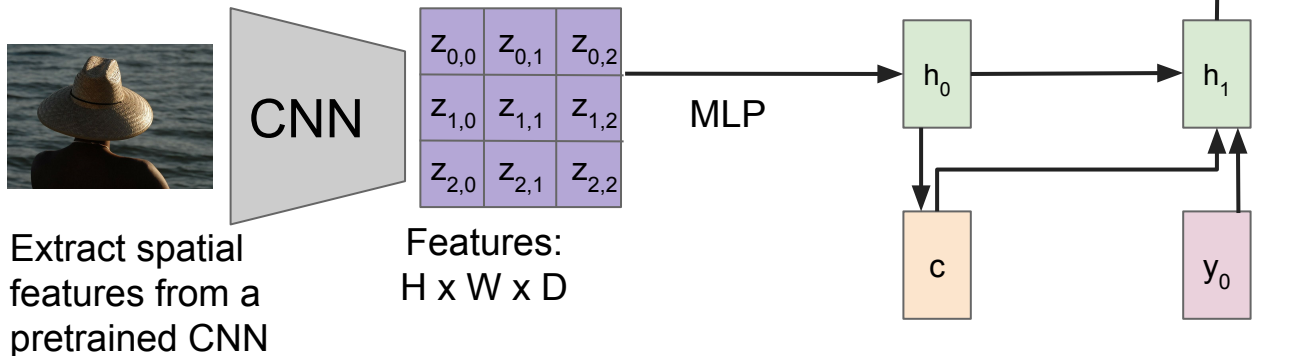
Encoder: $h_0 = f_w(\mathbf{z})$

where \mathbf{z} is spatial CNN features

$f_w(\cdot)$ is an MLP

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

where context vector c is often $c = h_0$



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

Image Captioning using spatial features

Input: Image I

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

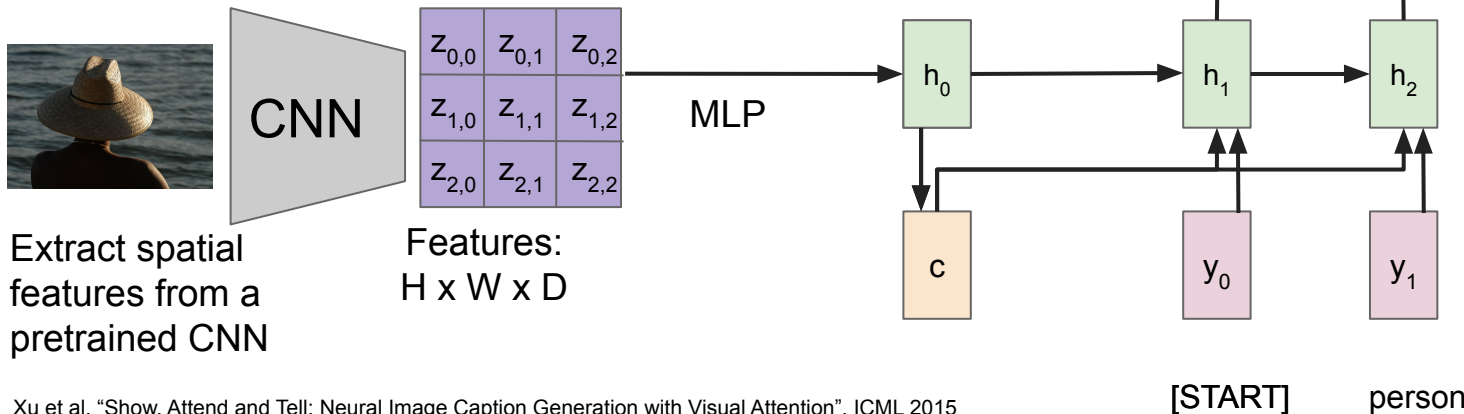
Encoder: $h_0 = f_w(\mathbf{z})$

where \mathbf{z} is spatial CNN features

$f_w(\cdot)$ is an MLP

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

where context vector c is often $c = h_0$



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

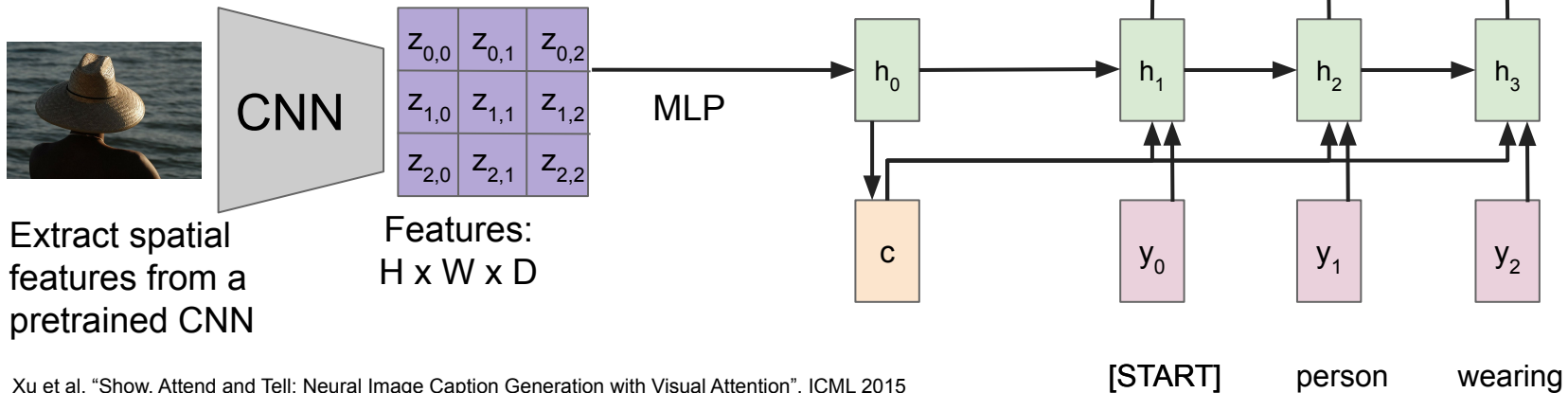
Image Captioning using spatial features

Input: Image I

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

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

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



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

Image Captioning using spatial features

Input: Image I

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

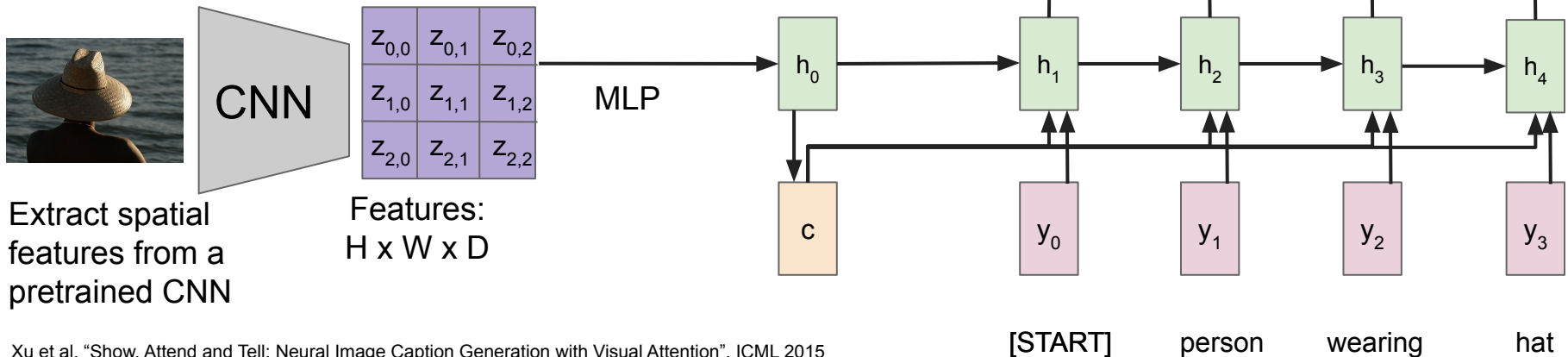
Encoder: $h_0 = f_w(\mathbf{z})$

where \mathbf{z} is spatial CNN features

$f_w(\cdot)$ is an MLP

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

where context vector c is often $c = h_0$



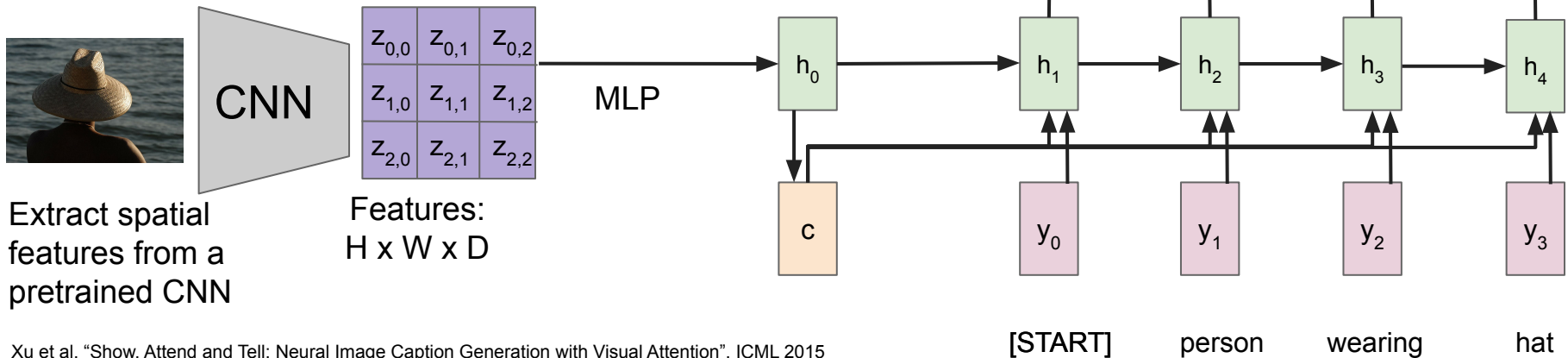
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning using spatial features

Problem: Input is "bottlenecked" through c

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

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



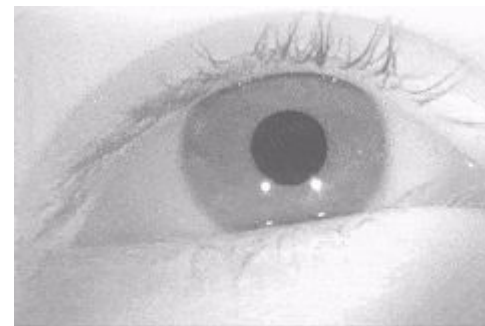
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with RNNs and Attention

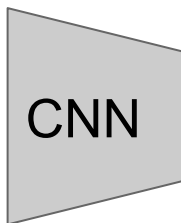
Attention idea: New context vector at every time step.

Each context vector will attend to different image regions

[gif source](#)

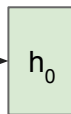


Attention Saccades in humans



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

Features:
 $H \times W \times D$



Extract spatial
features from a
pretrained CNN

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

Image Captioning with RNNs and Attention

Compute alignments scores (scalars):

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

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

Alignment scores:

H x W

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



CNN

Extract spatial features from a pretrained CNN

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

Features:

H x W x D

h_0

Image Captioning with RNNs and Attention

Compute alignments scores (scalars):

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

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

Alignment scores:
H x W

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

Attention:
H x W

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

Normalize to get
attention weights:

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

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



CNN

Extract spatial
features from a
pretrained CNN

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

Features:
H x W x D

h_0

Image Captioning with RNNs and Attention

Compute alignments scores (scalars):

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

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

Alignment scores:
H x W

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

Attention:
H x W

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

Normalize to get
attention weights:

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

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

Compute context vector:

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



CNN

Extract spatial
features from a
pretrained CNN

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

Features:
H x W x D

h_0



c_1

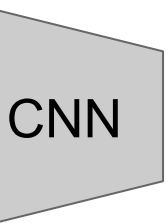
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with RNNs and Attention

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

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

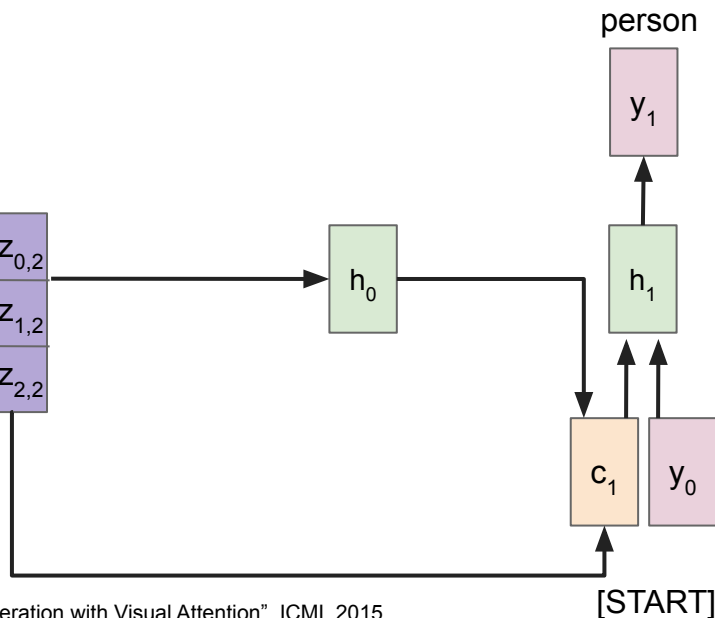
$$e_{t,i,j} = f_{att}(h_{t-1}, z_{i,j})$$
$$a_{t,:,:} = \text{softmax}(e_{t,:,:})$$
$$c_t = \sum_{i,j} a_{t,i,j} z_{i,j}$$



Extract spatial features from a pretrained CNN

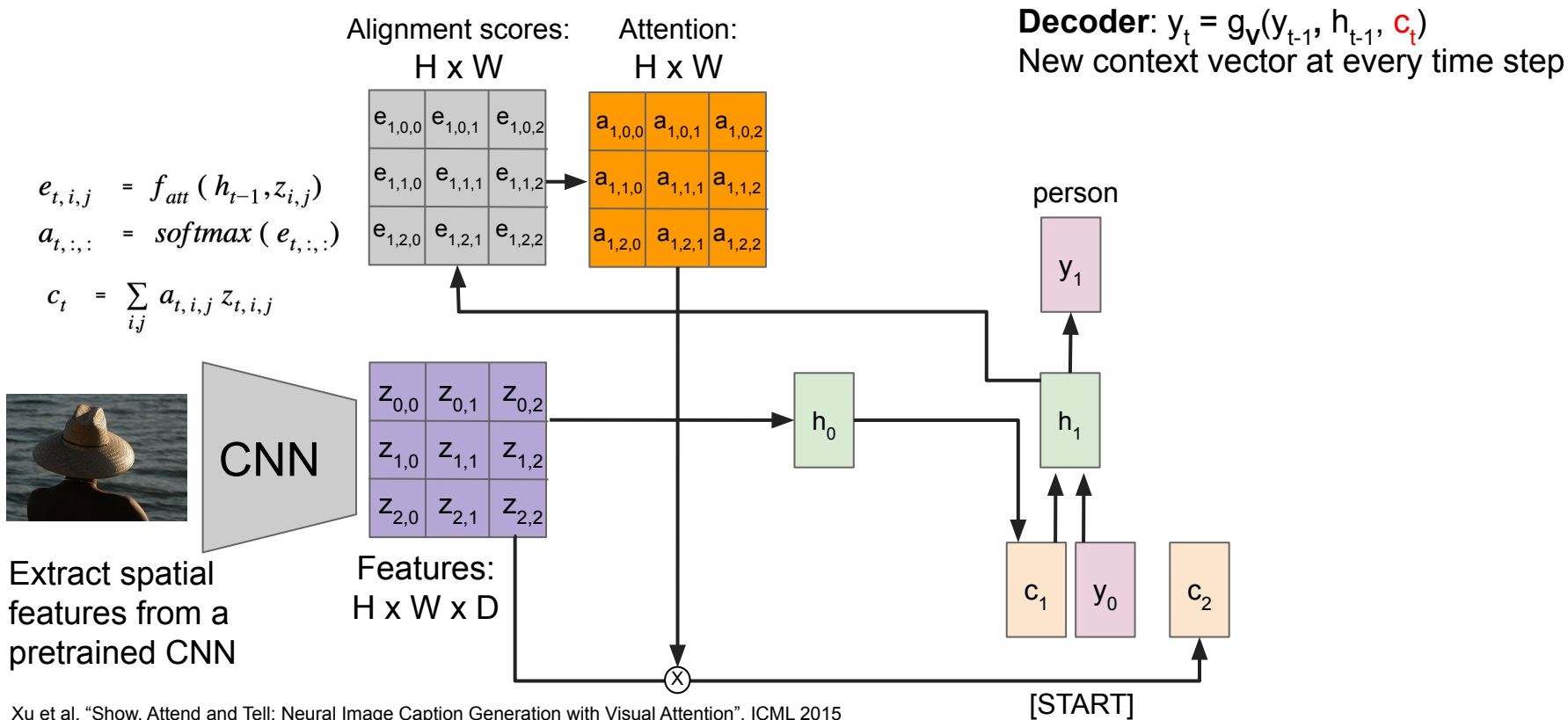
$z_{0,0}$	$z_{0,1}$	$z_{0,2}$
$z_{1,0}$	$z_{1,1}$	$z_{1,2}$
$z_{2,0}$	$z_{2,1}$	$z_{2,2}$

Features:
H x W x D



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

Image Captioning with RNNs and Attention



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

Image Captioning with RNNs and Attention

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

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

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



CNN

Extract spatial features from a pretrained CNN

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

Features:
H x W x D

h_0

person

wearing

y_1

y_2

h_1

h_2

c_1

y_0

c_2

y_1

[START]

person

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

Image Captioning with RNNs and Attention

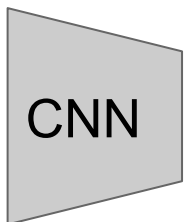
Each timestep of decoder uses a different context vector that looks at different parts of the input image

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

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

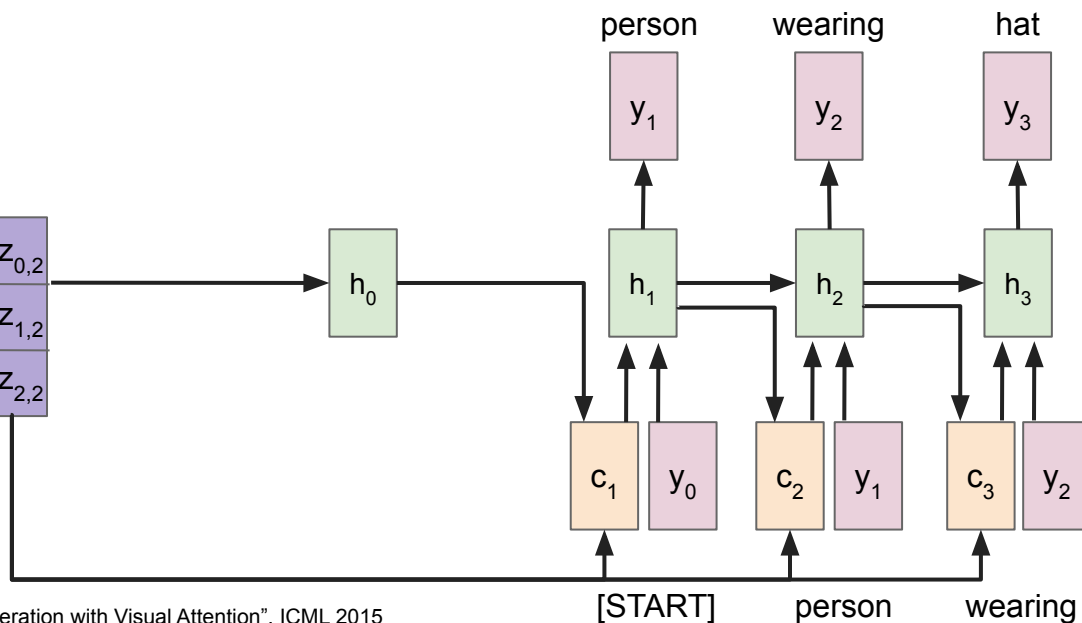


Extract spatial features from a pretrained CNN



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

Features:
H x W x D



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

Image Captioning with RNNs and Attention

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

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



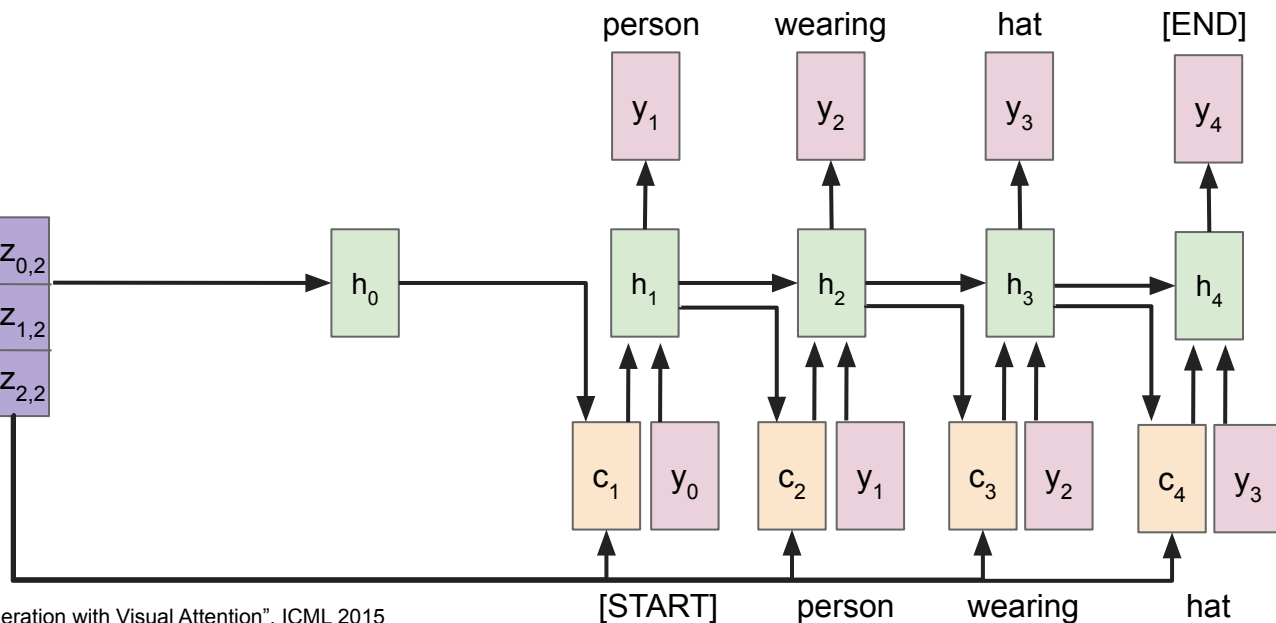
CNN

Extract spatial features from a pretrained CNN

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

Features:
H x W x D

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



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

Image Captioning with RNNs and Attention

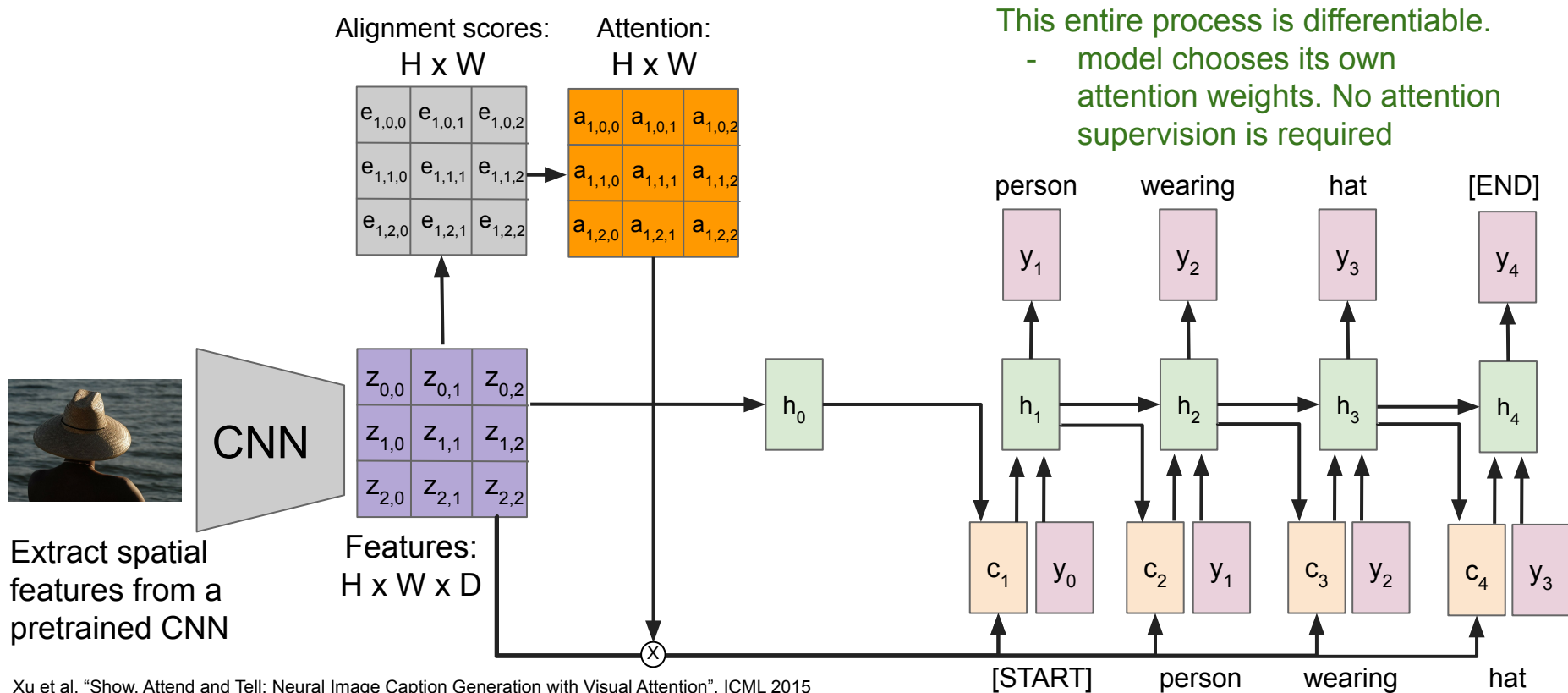
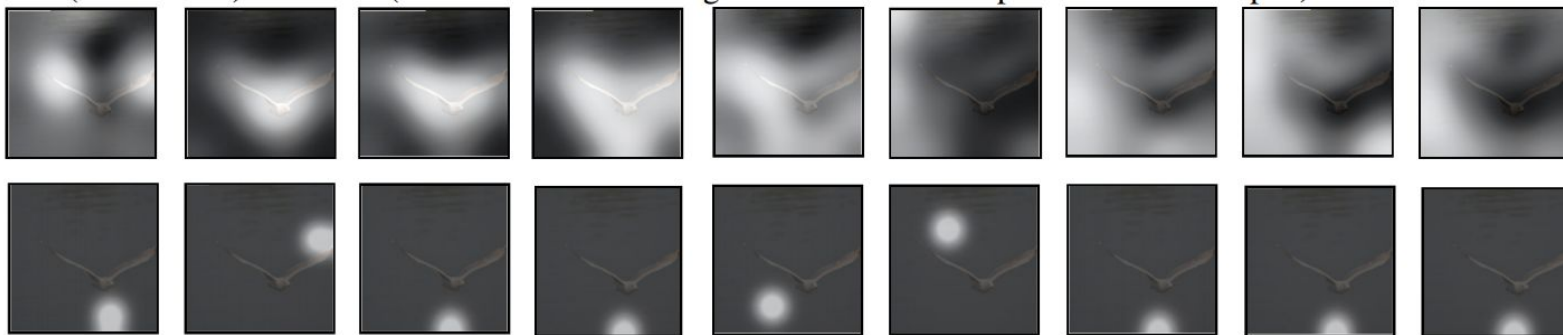


Image Captioning with Attention

Soft attention



A

bird

flying

over

a

body

of

water

▪

Hard attention
(requires
reinforcement
learning)

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

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

Image Captioning with Attention



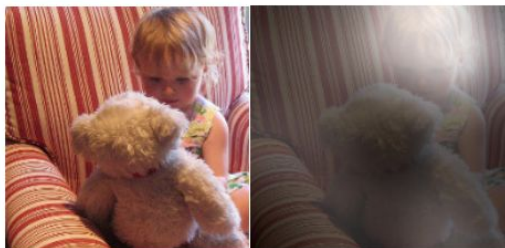
A woman is throwing a frisbee in a park.



A dog is standing on a hardwood floor.



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



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



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

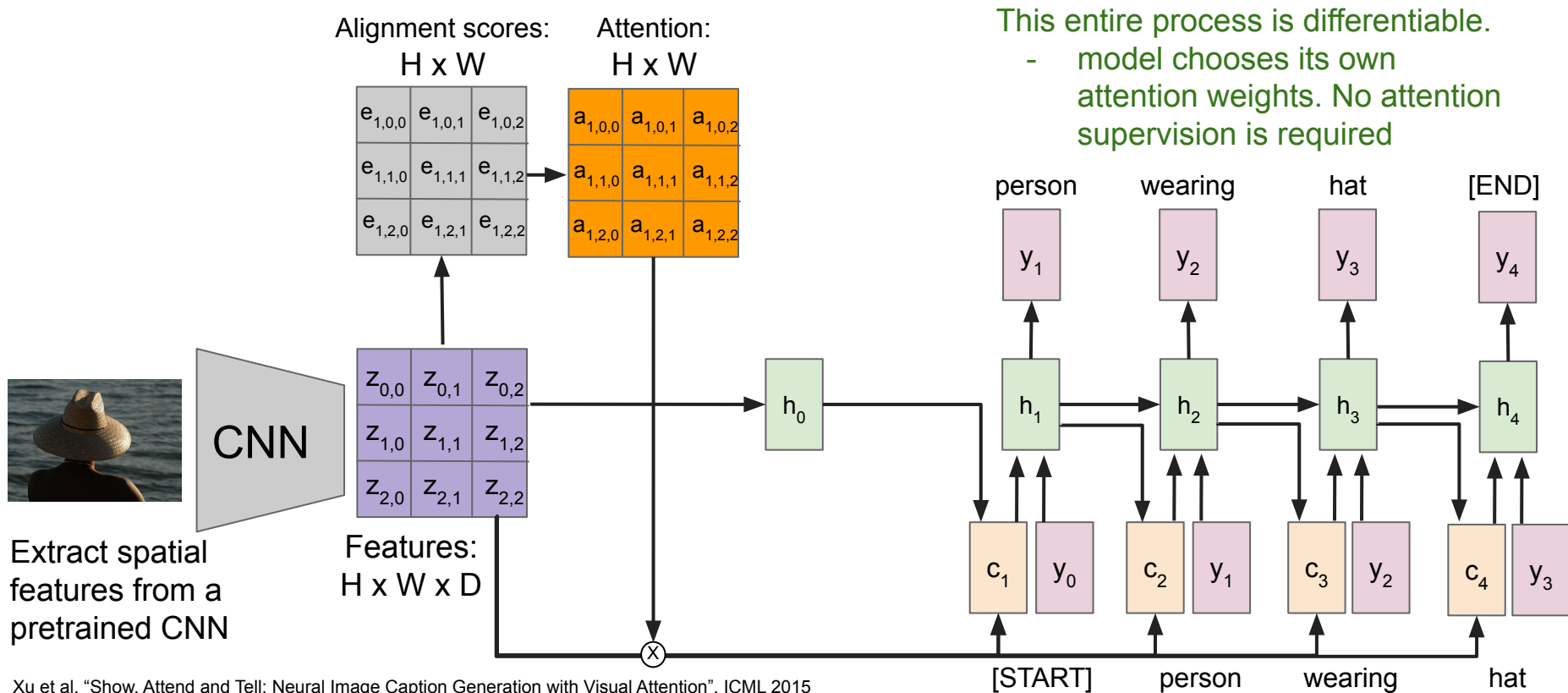


A giraffe standing in a forest with trees in the background.

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

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

Image Captioning with RNNs and Attention



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

Attention we just saw in image captioning

Features

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

h

Inputs:

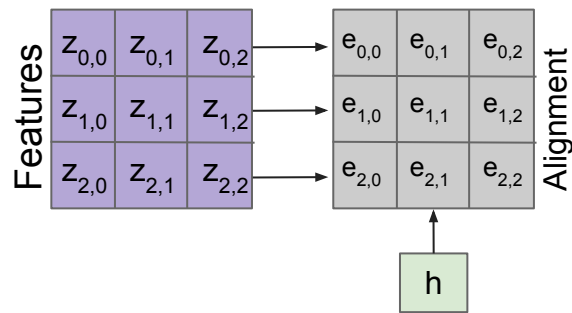
Features: \mathbf{z} (shape: $H \times W \times D$)

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

Attention we just saw in image captioning

Operations:

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

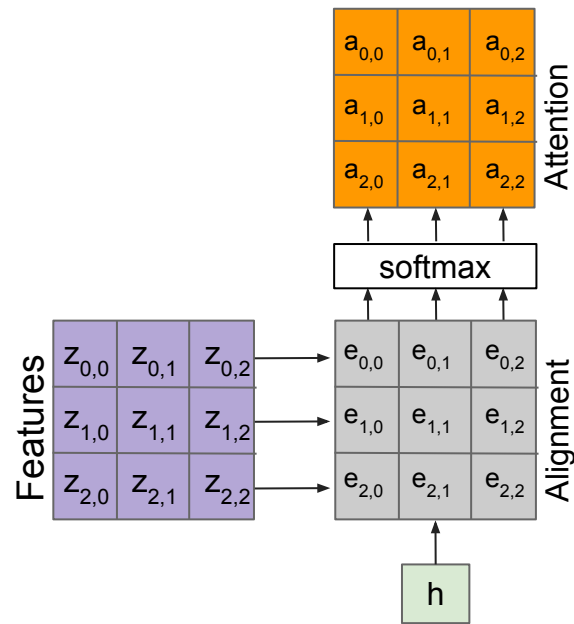


Inputs:

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

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

Attention we just saw in image captioning



Operations:

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

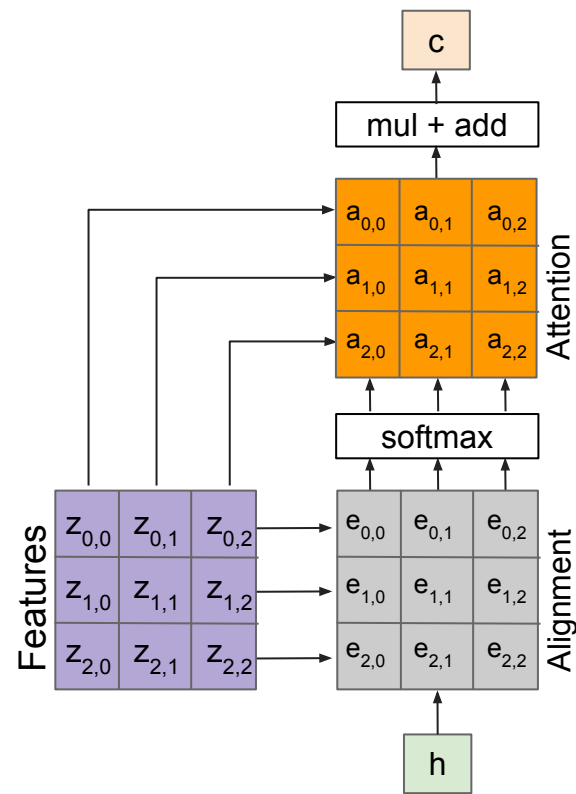
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$

Inputs:

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

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

Attention we just saw in image captioning



Outputs:

context vector: c (shape: D)

Operations:

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

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

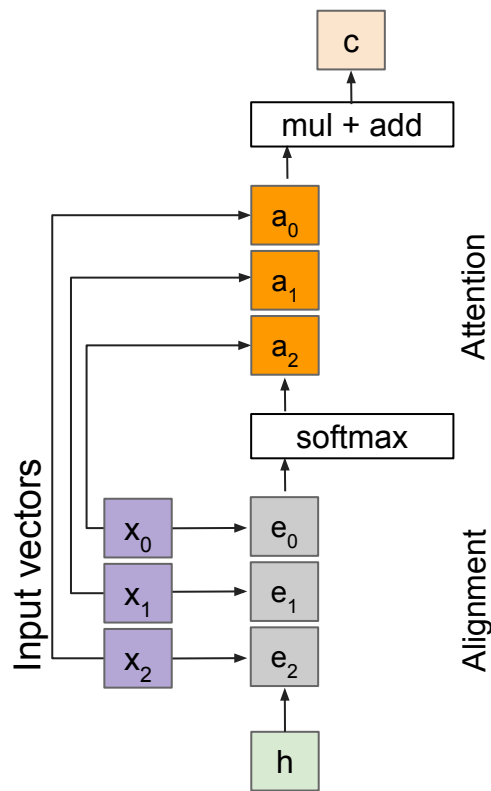
Output: $c = \sum_{i,j} a_{i,j} z_{i,j}$

Inputs:

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

Query: h (shape: D)

General attention layer



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

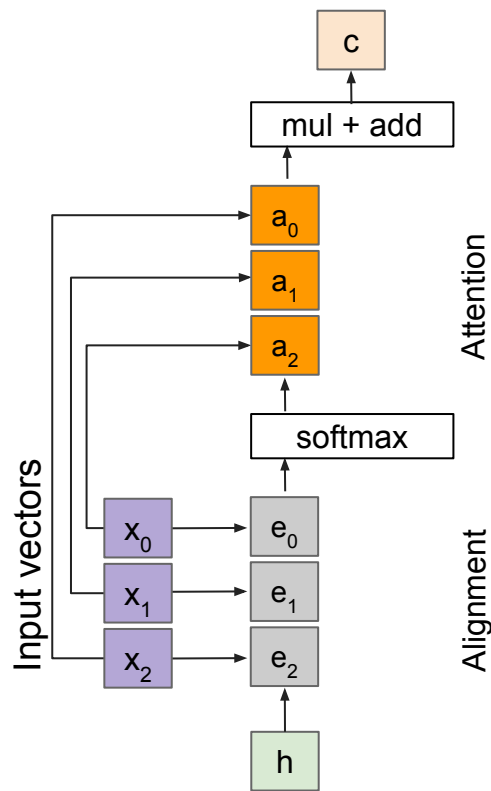
Operations:
Alignment: $e_i = f_{\text{att}}(h, x_i)$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $\mathbf{c} = \sum_i a_i x_i$

Attention operation is **permutation invariant**.

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

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

General attention layer



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

Operations:

Alignment: $e_i = h \cdot x_i$

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

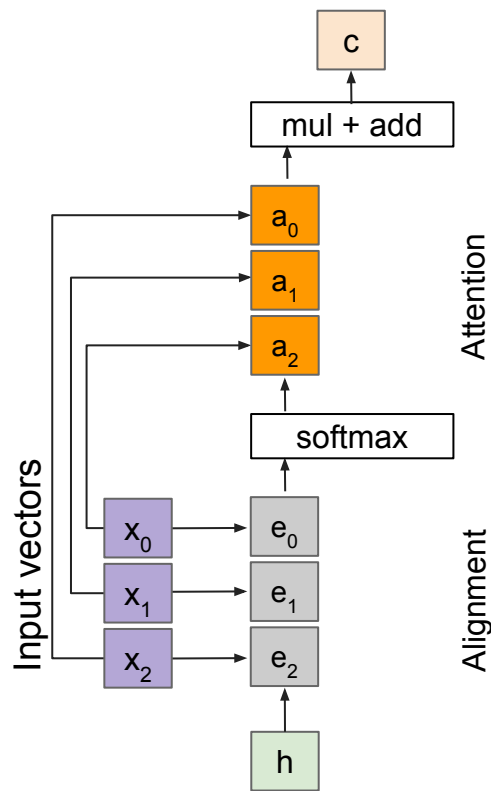
Output: $\mathbf{c} = \sum_i a_i x_i$

Change $f_{\text{att}}(\cdot)$ 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: $\mathbf{a} = \text{softmax}(\mathbf{e})$

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

Inputs:

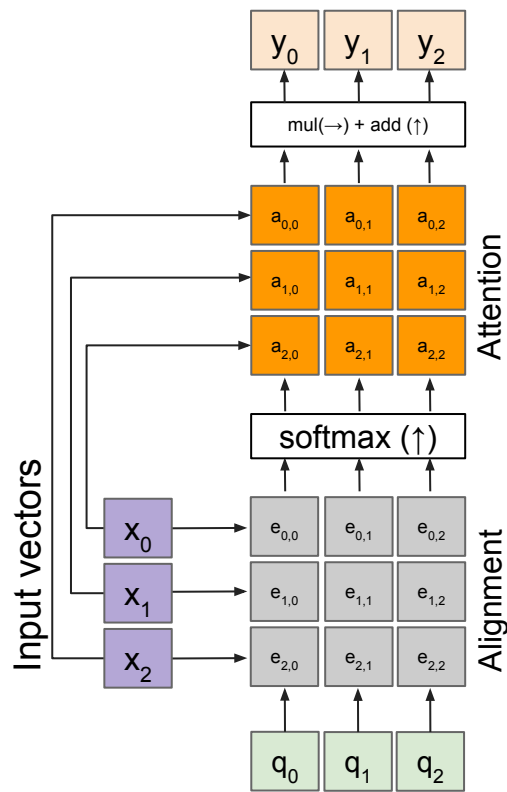
Input vectors: **x** (shape: N x D)

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

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

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

General attention layer



Outputs:

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

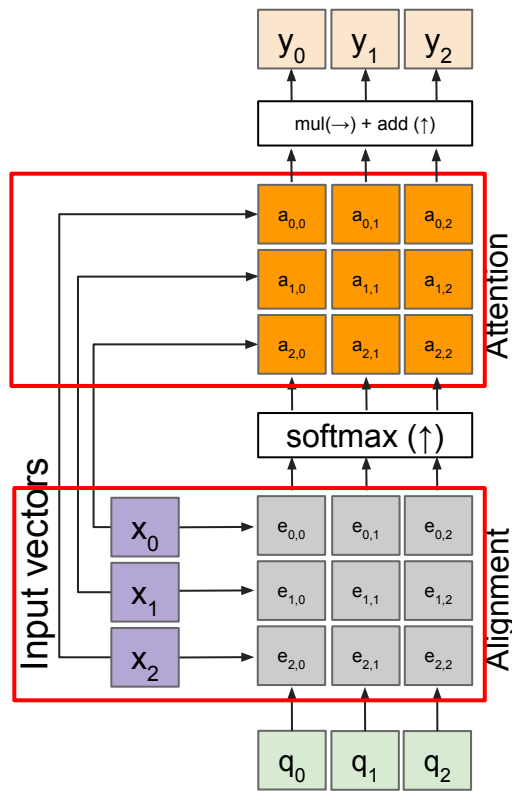
Inputs:

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

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

Multiple query vectors

General attention layer



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

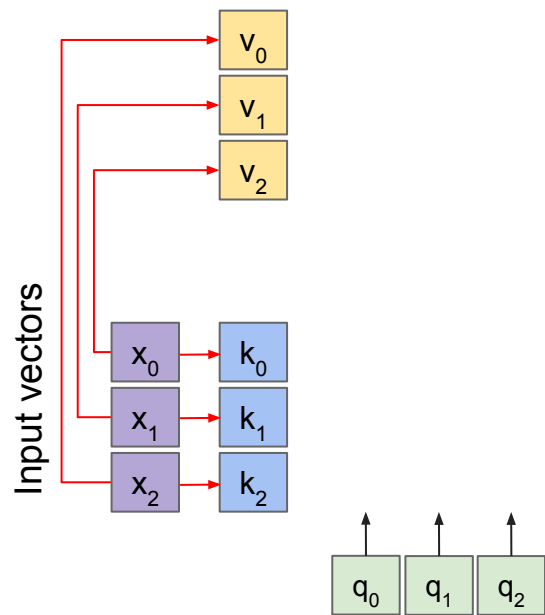
Operations:
Alignment: $e_{i,j} = \mathbf{q}_j \cdot \mathbf{x}_i / \sqrt{D}$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $y_j = \sum_i a_{i,j} x_i$

Inputs:
Input vectors: \mathbf{x} (shape: N x D)
Queries: \mathbf{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: $\mathbf{v} = \mathbf{x}\mathbf{W}_v$

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

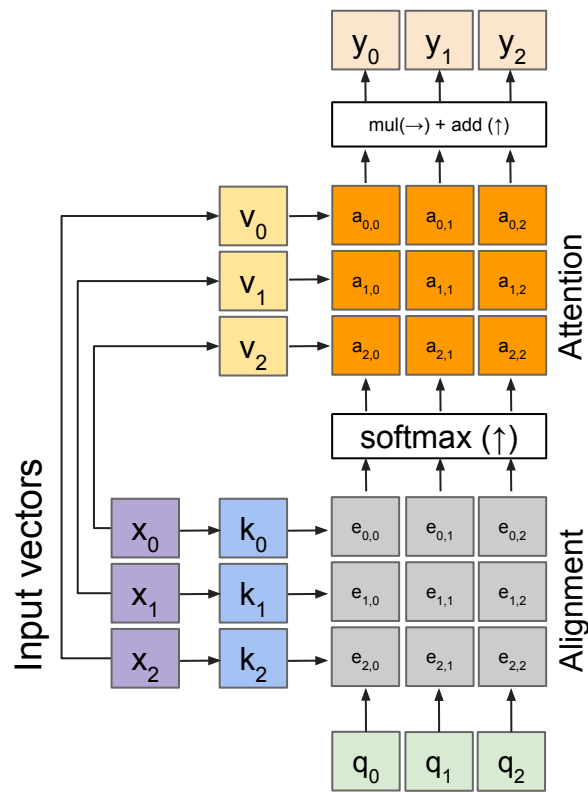
- We can add more expressivity to the layer by adding a different FC layer before each of the two steps.

Inputs:

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

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

General attention layer



Outputs:

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

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

Operations:

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

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

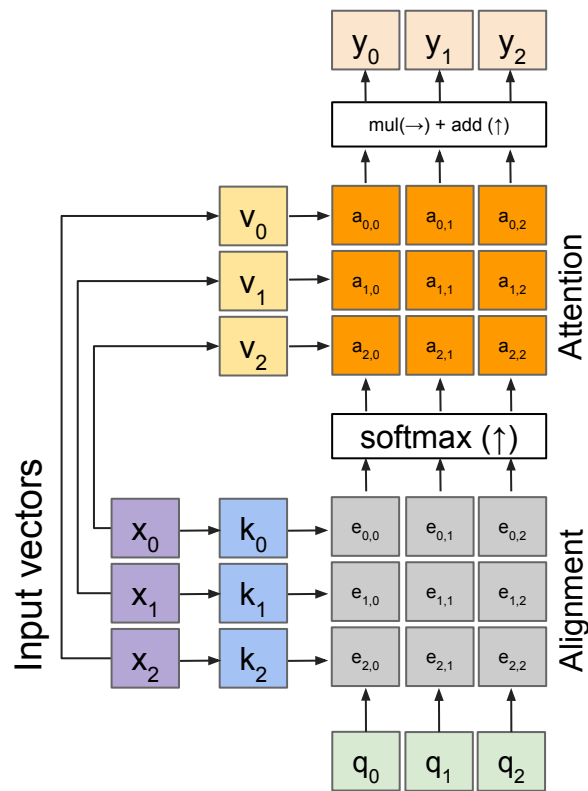
- We can add more expressivity to the layer by adding a different FC layer before each of the two steps.

Inputs:

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

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

General attention layer



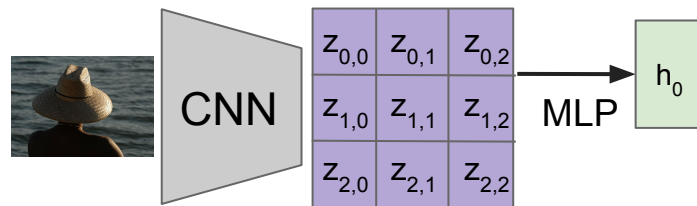
Outputs:
context vectors: \mathbf{y} (shape: D_v)

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

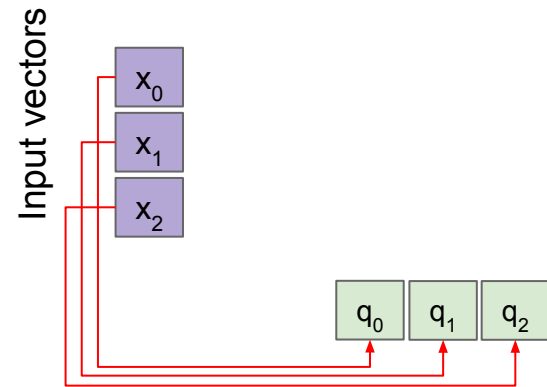
Inputs:
Input vectors: \mathbf{x} (shape: $N \times D$)
Queries: \mathbf{q} (shape: $M \times D_k$)

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

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



Self attention layer



Operations:

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

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

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

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

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

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

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

Instead, query vectors are calculated using a FC layer.

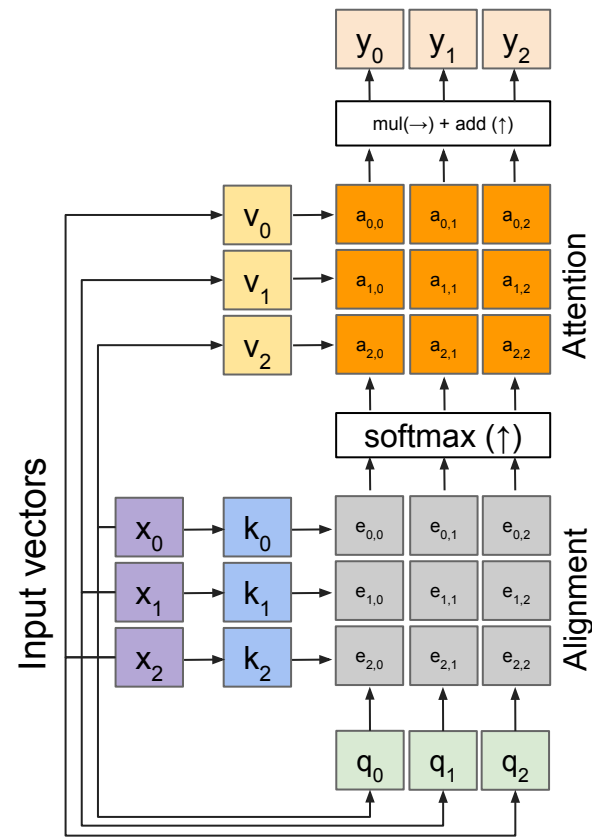
No input query vectors anymore

Inputs:

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

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

Self attention layer



Outputs:

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

Operations:

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

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

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

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

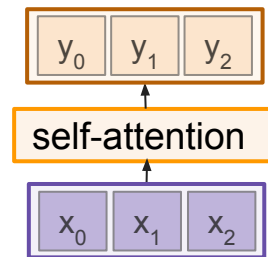
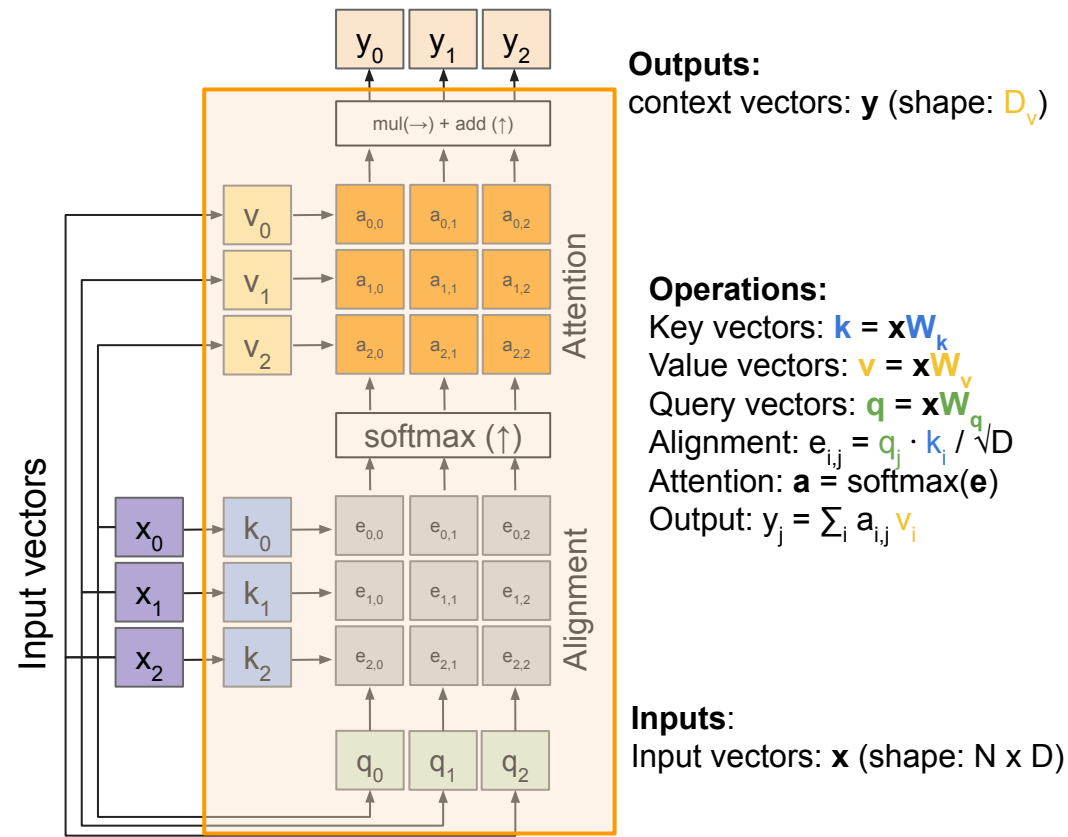
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$

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

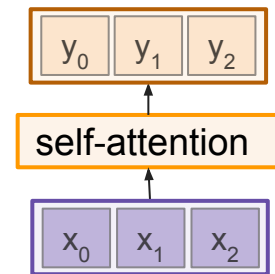
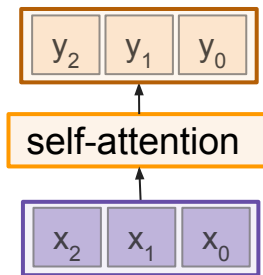
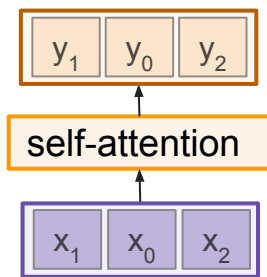
Inputs:

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

Self attention layer - attends over sets of inputs



Self attention layer - attends over sets of inputs

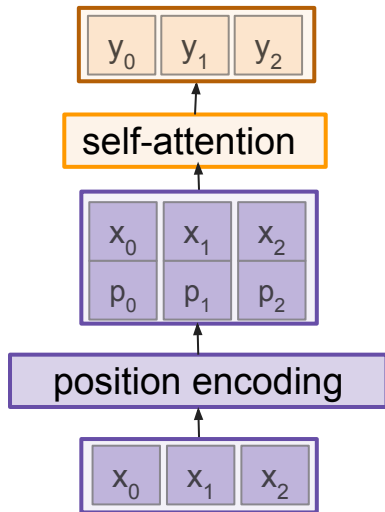


Permutation equivariant

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

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

Positional encoding



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

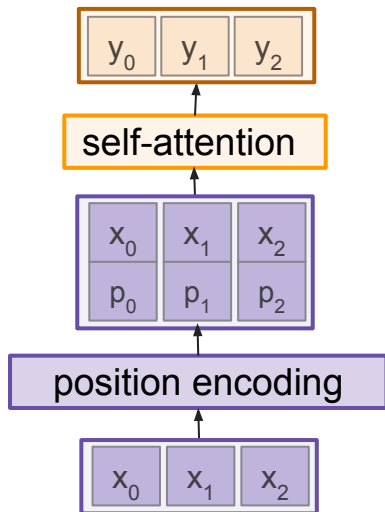
We use a function $pos: \mathbb{N} \rightarrow \mathbb{R}^d$ to process the position j of the vector into a d -dimensional vector

So, $p_j = pos(j)$

Desiderata of $pos(\cdot)$:

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

Positional encoding



Concatenate special positional encoding p_j to each input vector x_j

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

So, $p_j = pos(j)$

Options for $pos(.)$

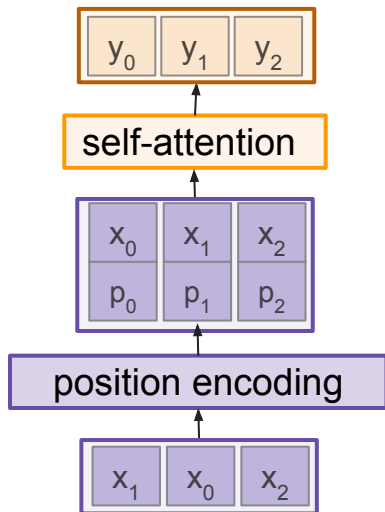
1. Learn a lookup table:
 - Learn parameters to use for $pos(t)$ for $t \in [0, T)$
 - Lookup table contains $T \times d$ parameters.

Desiderata of $pos(.)$:

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

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

Positional encoding



Concatenate special positional encoding p_j to each input vector x_j

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

So, $p_j = pos(j)$

Options for $pos(\cdot)$

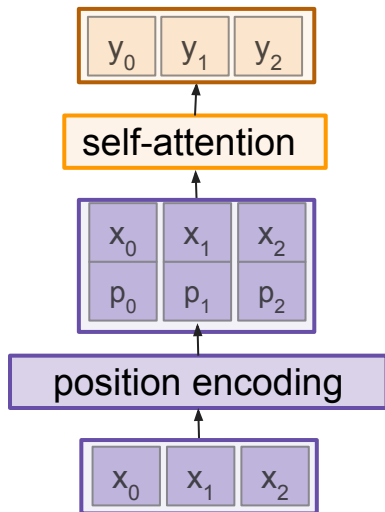
1. Learn a lookup table:
 - Learn parameters to use for $pos(t)$ for $t \in [0, T)$
 - Lookup table contains $T \times d$ parameters.
2. Design a fixed function with the desiderata

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

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

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

Positional encoding



Concatenate special positional encoding p_j to each input vector x_j

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

So, $p_j = pos(j)$

Options for $pos(\cdot)$

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

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

Intuition:

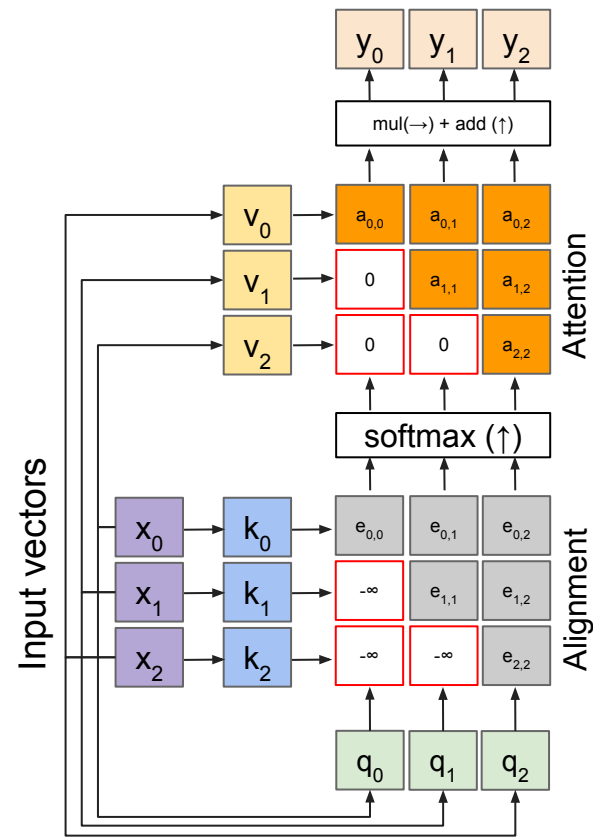
0 :	0	0	0	0	0
1 :	0	0	0	1	0
2 :	0	0	1	0	0
3 :	0	0	1	1	0
4 :	0	1	0	0	0
5 :	0	1	0	1	0
6 :	0	1	1	0	0
7 :	0	1	1	1	0
8 :	1	0	0	0	0
9 :	1	0	0	1	0
10 :	1	0	1	0	0
11 :	1	0	1	1	0
12 :	1	1	0	0	0
13 :	1	1	0	1	0
14 :	1	1	1	0	0
15 :	1	1	1	1	0

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

[image source](#)

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

Masked self-attention layer



Outputs:

context vectors: y (shape: D_v)

Operations:

Key vectors: $k = xW_k$

Value vectors: $v = xW_v$

Query vectors: $q = xW_q$

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

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

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

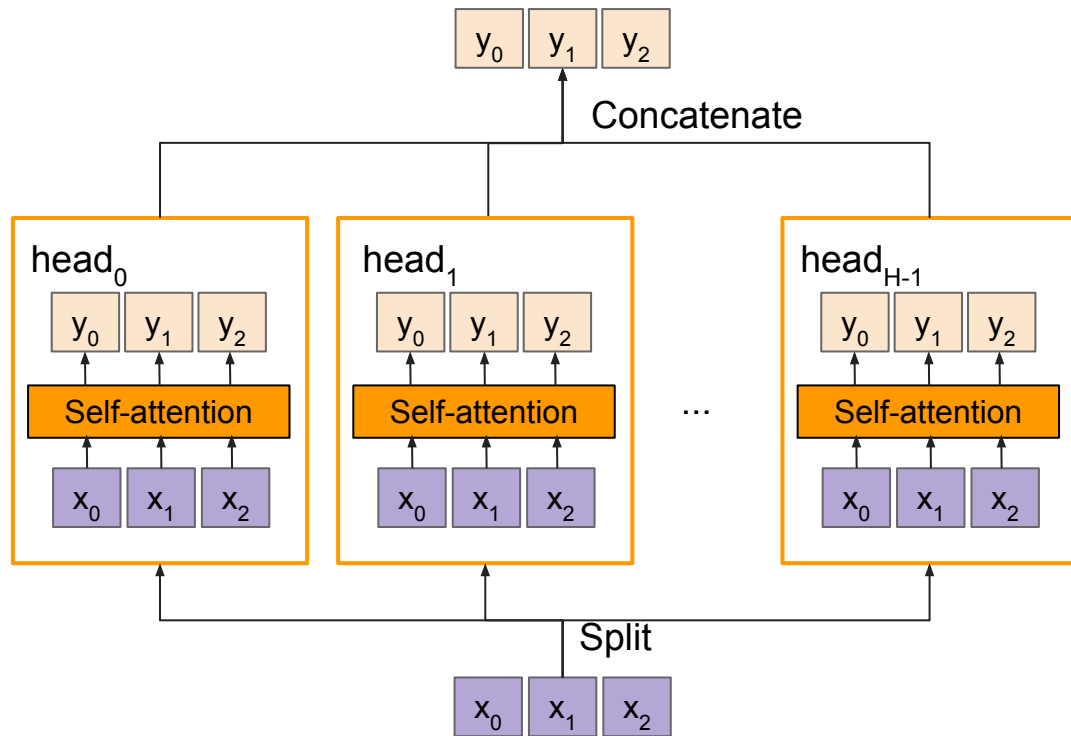
Inputs:

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

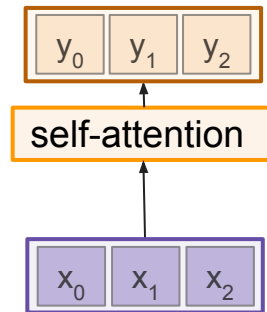
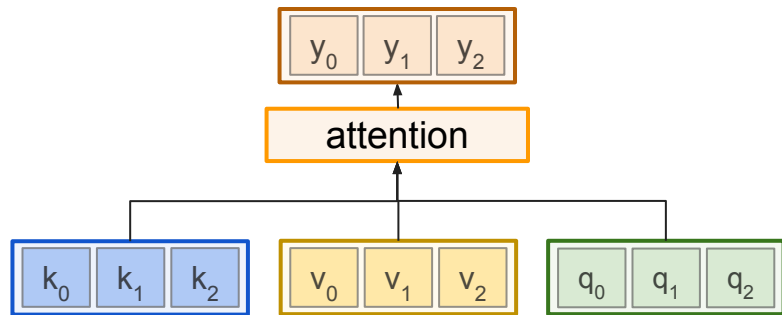
- Prevent vectors from looking at future vectors.
- Manually set alignment scores to -infinity

Multi-head self attention layer

- Multiple self-attention heads in parallel

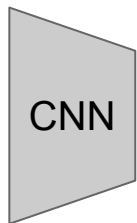
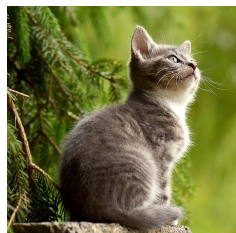


General attention versus self-attention



Example: CNN with Self-Attention

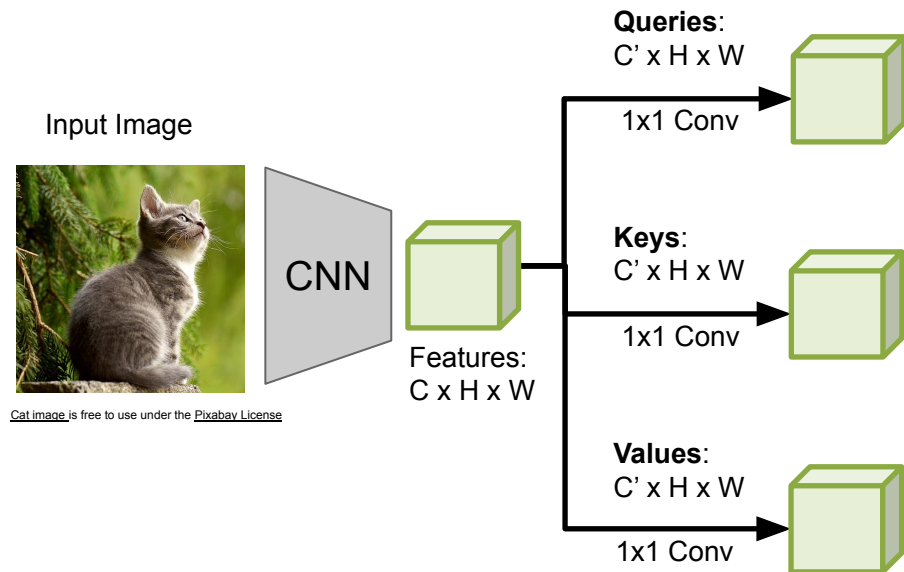
Input Image



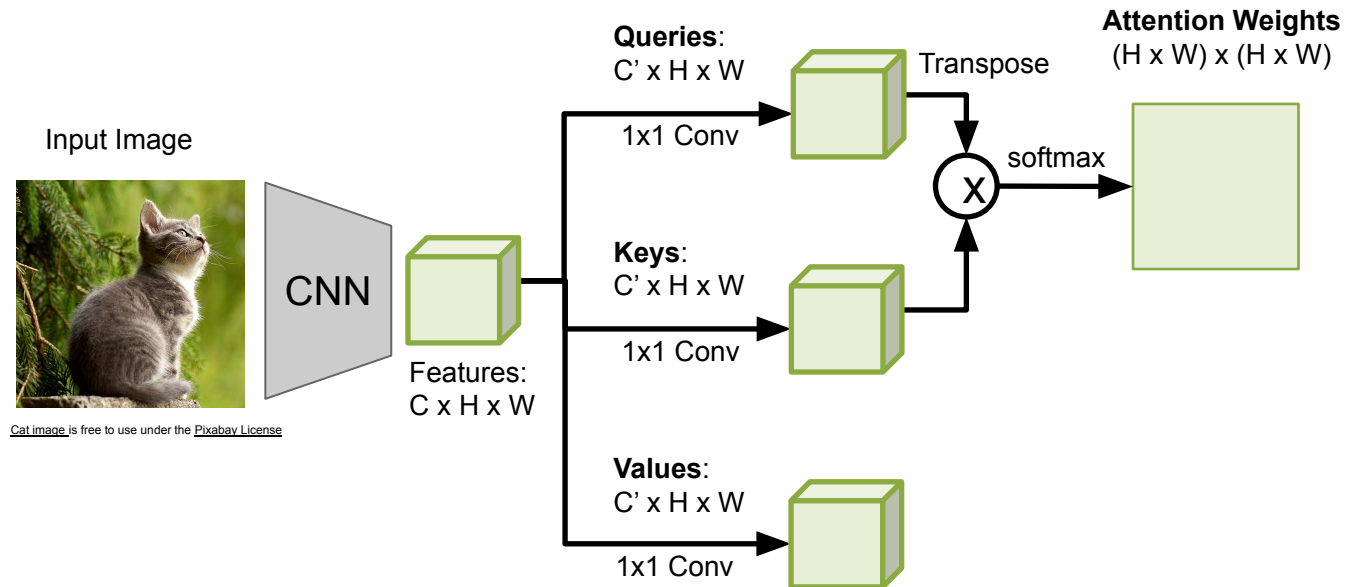
Features:
 $C \times H \times W$

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

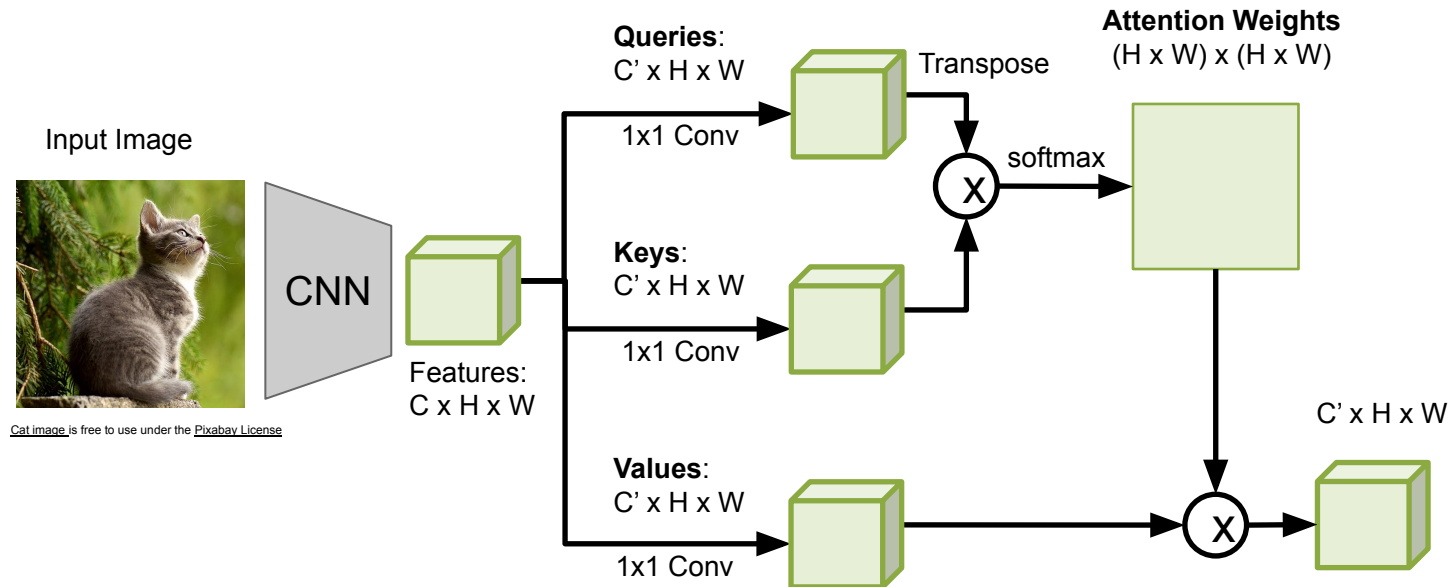
Example: CNN with Self-Attention



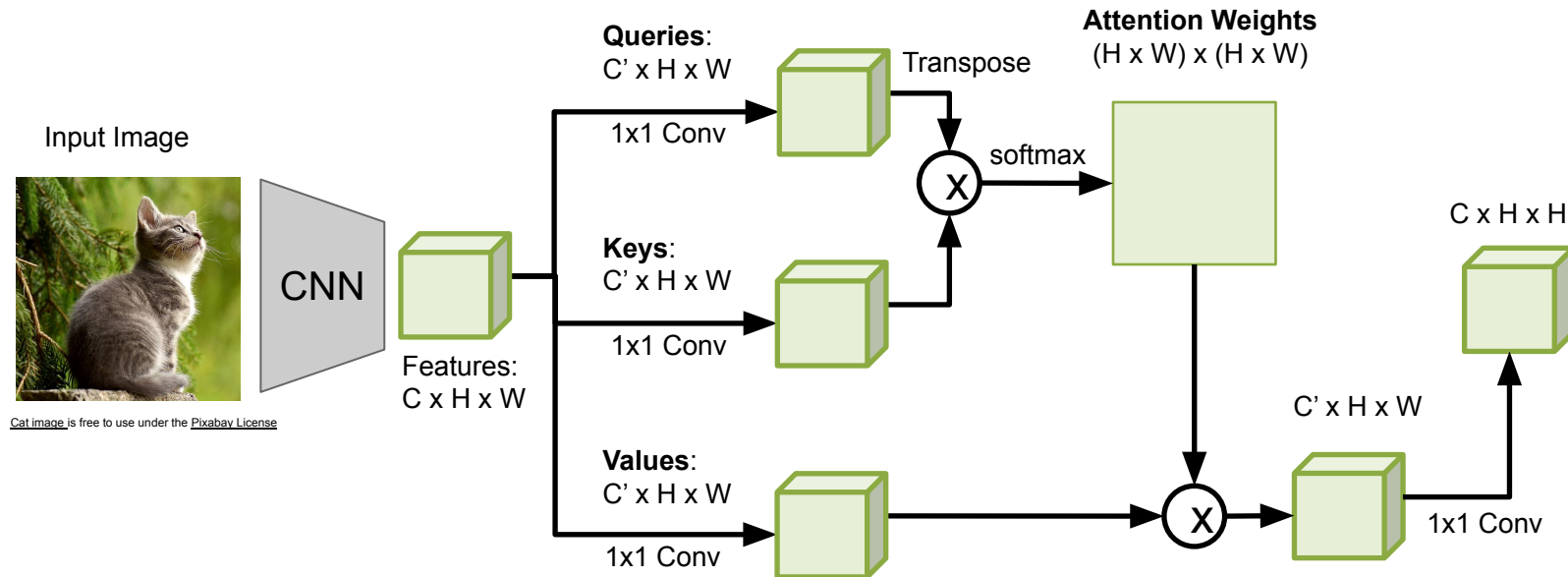
Example: CNN with Self-Attention



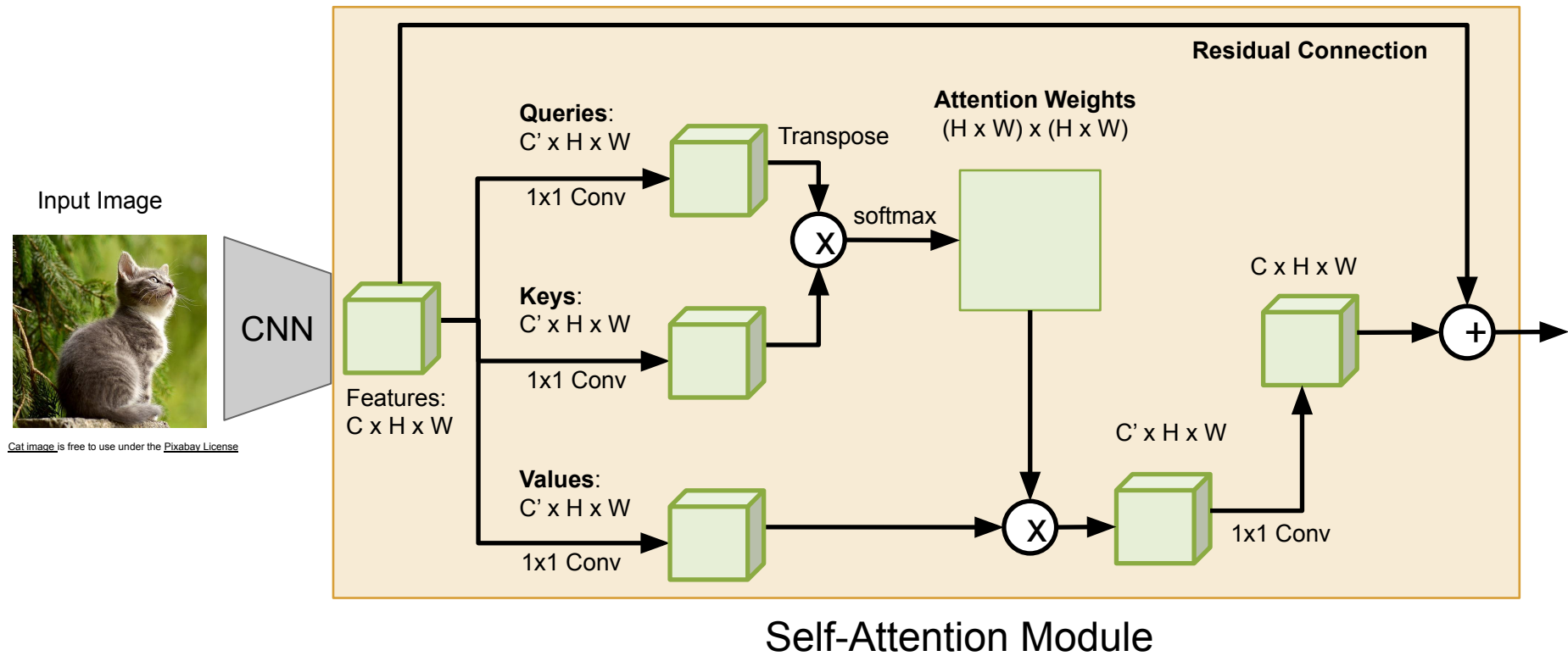
Example: CNN with Self-Attention



Example: CNN with Self-Attention



Example: CNN with Self-Attention



Comparing RNNs to Transformer

RNNs

(+) LSTMs work reasonably well for long sequences.

(-) Expects an ordered sequences of inputs

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

Transformer:

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

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

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

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

Attention Is All You Need

Ashish Vaswani*

Google Brain

avaswani@google.com

Noam Shazeer*

Google Brain

noam@google.com

Niki Parmar*

Google Research

nikip@google.com

Jakob Uszkoreit*

Google Research

usz@google.com

Llion Jones*

Google Research

llion@google.com

Aidan N. Gomez* †

University of Toronto

aidan@cs.toronto.edu

Łukasz Kaiser*

Google Brain

lukaszkaizer@google.com

Illia Polosukhin* †

illia.polosukhin@gmail.com

“ImageNet Moment for Natural Language Processing”

Pretraining:

Download a lot of text from the internet

Train a giant Transformer model for language modeling

Finetuning:

Fine-tune the Transformer on your own NLP task

On the Opportunities and Risks of Foundation Models

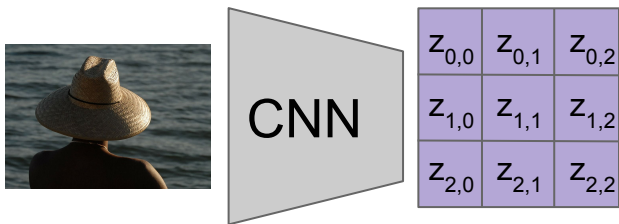
Rishi Bommasani* Drew A. Hudson Ehsan Adeli Russ Altman Simran Arora
Sydney von Arx Michael S. Bernstein Jeannette Bohg Antoine Bosselut Emma Brunskill
Erik Brynjolfsson Shyamal Buch Dallas Card Rodrigo Castellon Niladri Chatterji
Annie Chen Kathleen Creel Jared Quincy Davis Dorottya Demszky Chris Donahue
Moussa Doumbouya Esin Durmus Stefano Ermon John Etchemendy Kawin Ethayarajh
Li Fei-Fei Chelsea Finn Trevor Gale Lauren Gillespie Karan Goel Noah Goodman
Shelby Grossman Neel Guha Tatsunori Hashimoto Peter Henderson John Hewitt
Daniel E. Ho Jenny Hong Kyle Hsu Jing Huang Thomas Icard Saahil Jain
Dan Jurafsky Pratyusha Kalluri Siddharth Karamcheti Geoff Keeling Fereshte Khani
Omar Khattab Pang Wei Koh Mark Krass Ranjay Krishna Rohith Kuditipudi
Ananya Kumar Faisal Ladhak Mina Lee Tony Lee Jure Leskovec Isabelle Levent
Xiang Lisa Li Xuechen Li Tengyu Ma Ali Malik Christopher D. Manning
Suvir Mirchandani Eric Mitchell Zanele Munyikwa Suraj Nair Avanika Narayan
Deepak Narayanan Ben Newman Allen Nie Juan Carlos Nieves Hamed Nilforoshan
Julian Nyarko Giray Ogut Laurel Orr Isabel Papadimitriou Joon Sung Park Chris Piech
Eva Portelance Christopher Potts Aditi Raghunathan Rob Reich Hongyu Ren
Frieda Rong Yusuf Roohani Camilo Ruiz Jack Ryan Christopher Ré Dorsa Sadigh
Shiori Sagawa Keshav Santhanam Andy Shih Krishnan Srinivasan Alex Tamkin
Rohan Taori Armin W. Thomas Florian Tramèr Rose E. Wang William Wang Bohan Wu
Jiajun Wu Yuhuai Wu Sang Michael Xie Michihiro Yasunaga Jiaxuan You Matei Zaharia
Michael Zhang Tianyi Zhang Xikun Zhang Yuhui Zhang Lucia Zheng Kaitlyn Zhou
Percy Liang*¹

Center for Research on Foundation Models (CRFM)
Stanford Institute for Human-Centered Artificial Intelligence (HAI)
Stanford University

Image Captioning using Transformers

Input: Image I

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



Extract spatial
features from a
pretrained CNN

Features:
 $H \times W \times D$

Image Captioning using Transformers

Input: Image I

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

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

where \mathbf{z} is spatial CNN features

$T_w(\cdot)$ is the transformer encoder

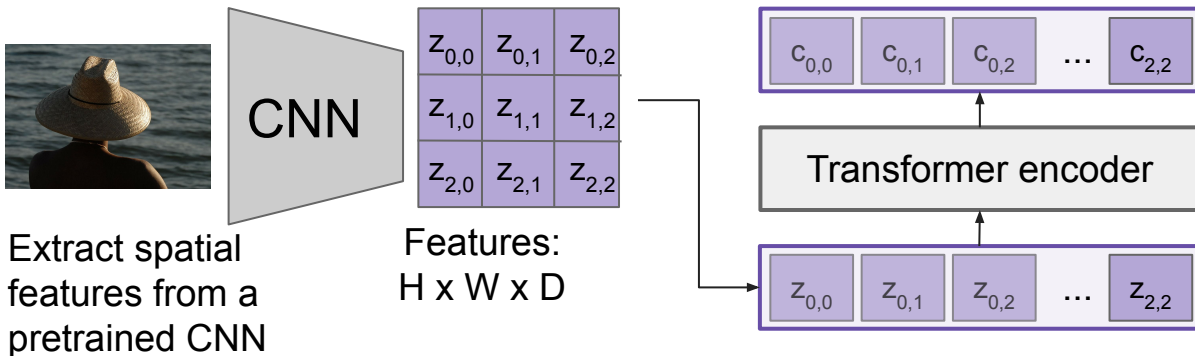


Image Captioning using Transformers

Input: Image I

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

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

where \mathbf{z} is spatial CNN features

$T_w(\cdot)$ is the transformer encoder

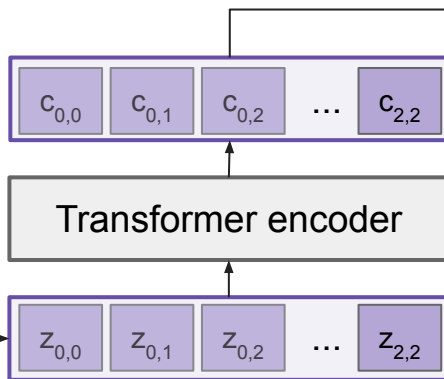


CNN

Extract spatial features from a pretrained CNN

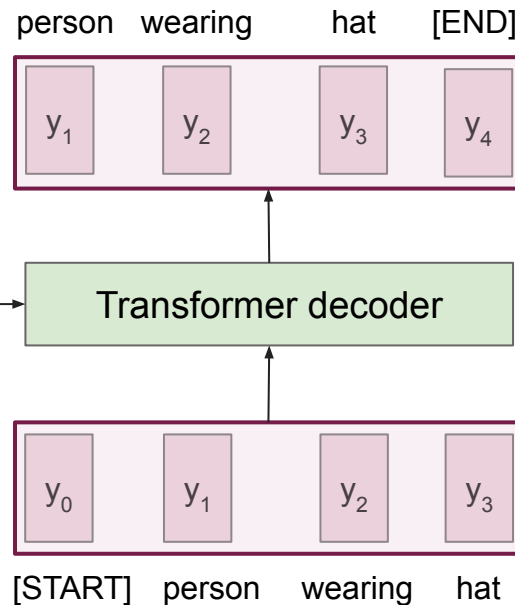
$z_{0,0}$	$z_{0,1}$	$z_{0,2}$
$z_{1,0}$	$z_{1,1}$	$z_{1,2}$
$z_{2,0}$	$z_{2,1}$	$z_{2,2}$

Features:
 $H \times W \times D$

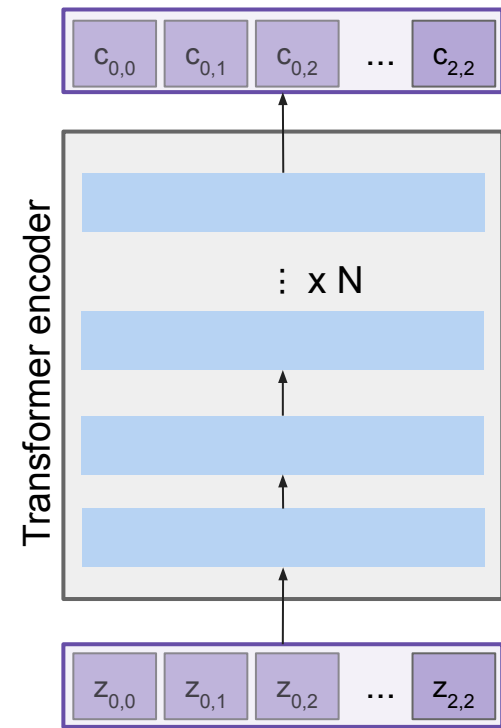


Decoder: $y_t = T_d(y_{0:t-1}, \mathbf{c})$

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



The Transformer encoder block

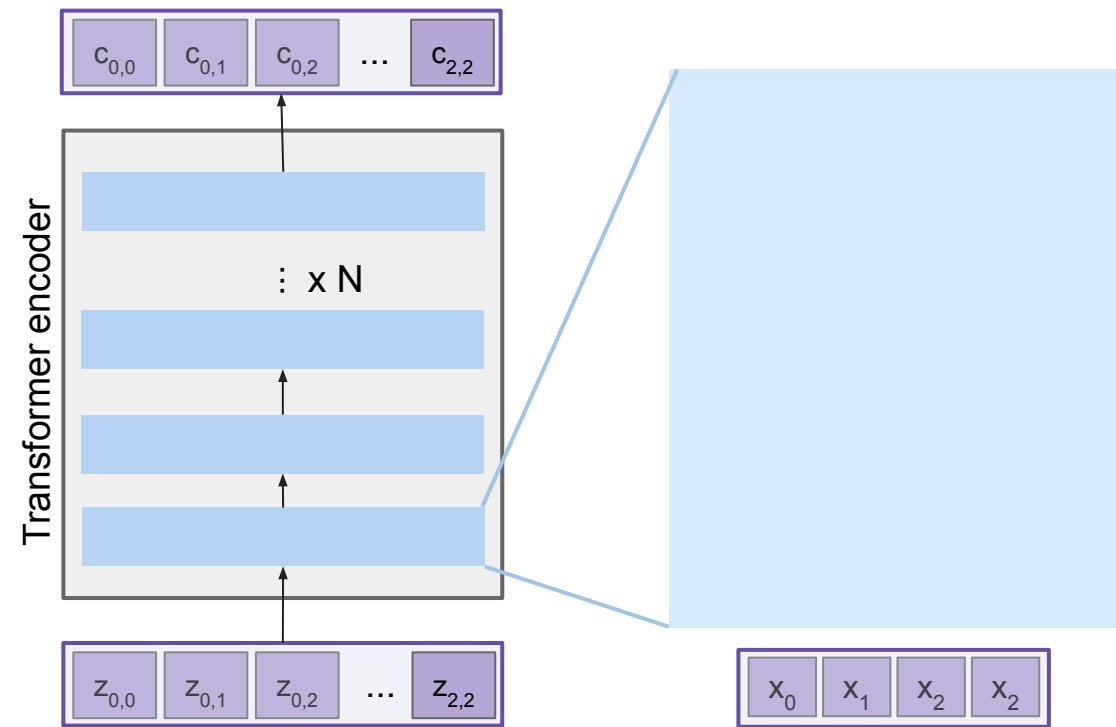


Made up of N encoder blocks.

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

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

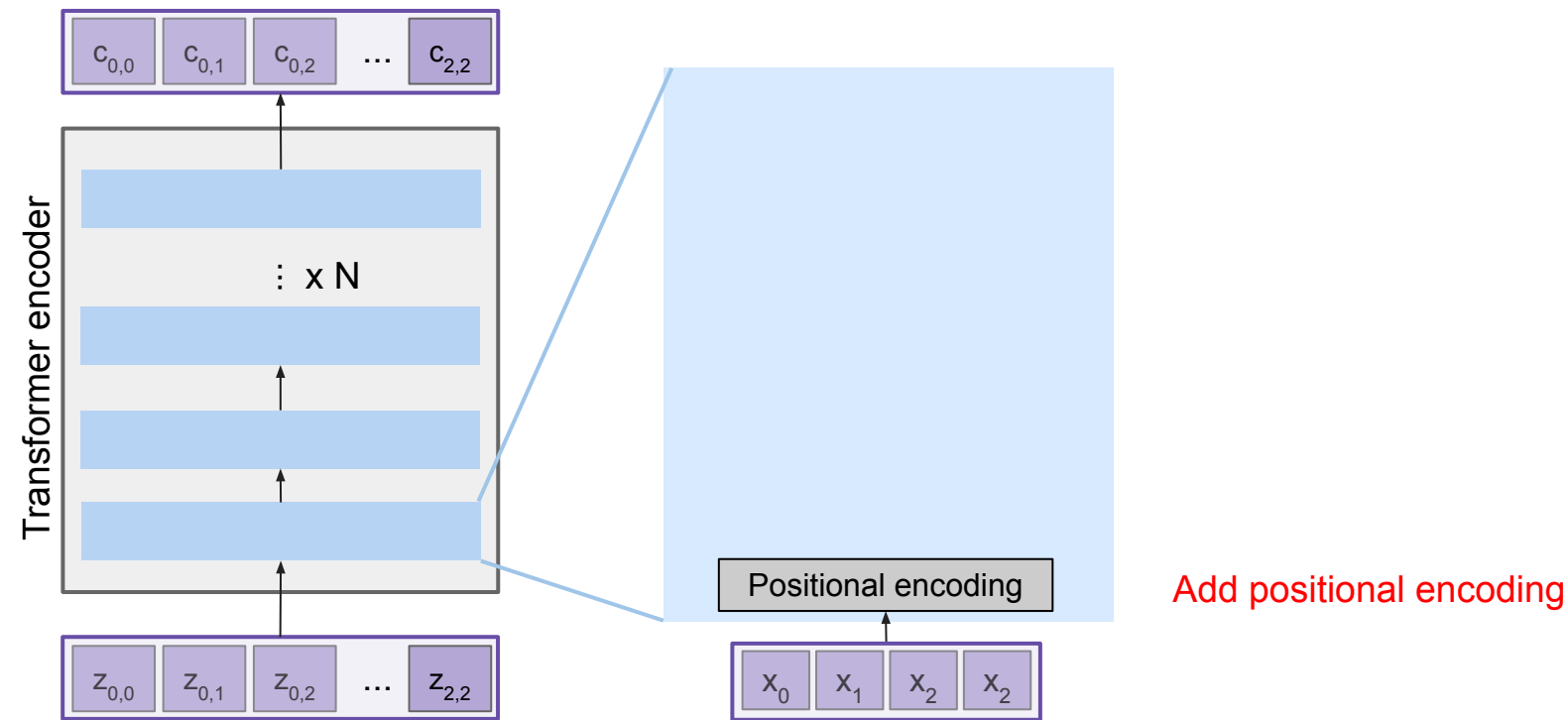
The Transformer encoder block



Let's dive into one encoder block

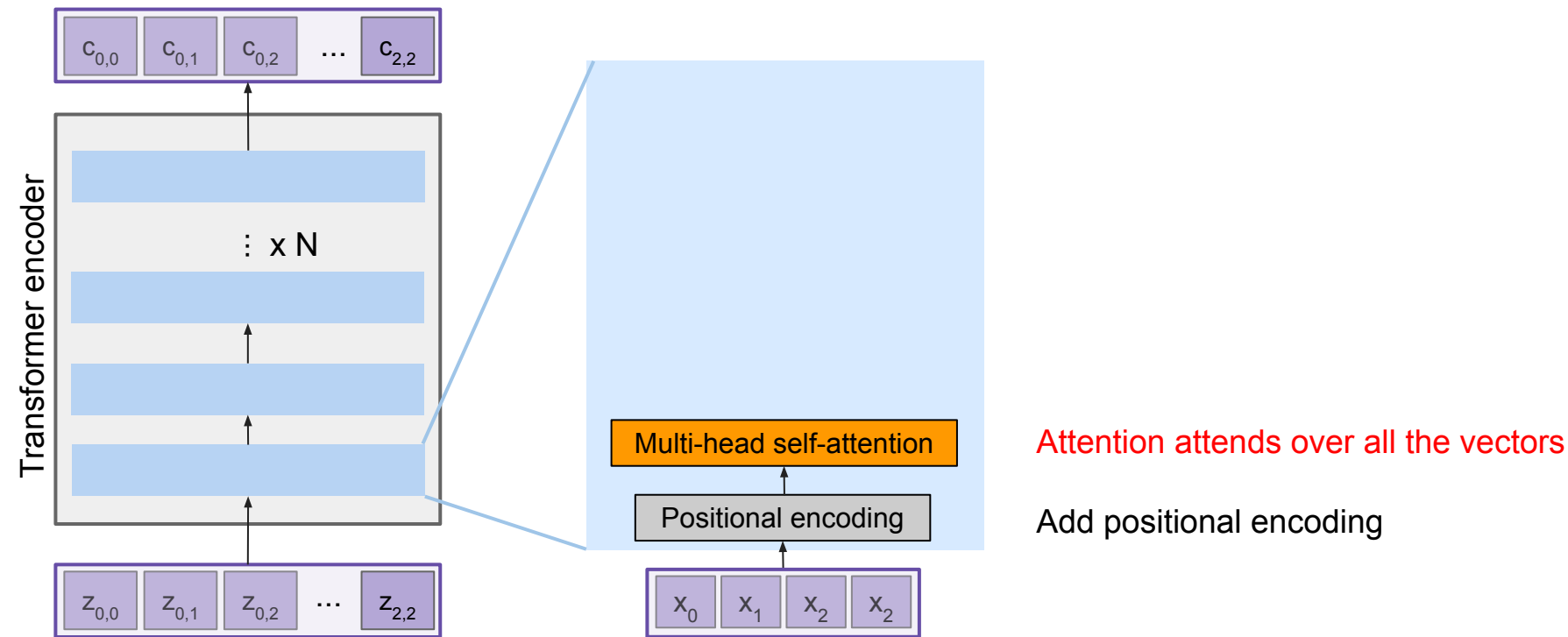
Vaswani et al, "Attention is all you need", NeurIPS 2017

The Transformer encoder block



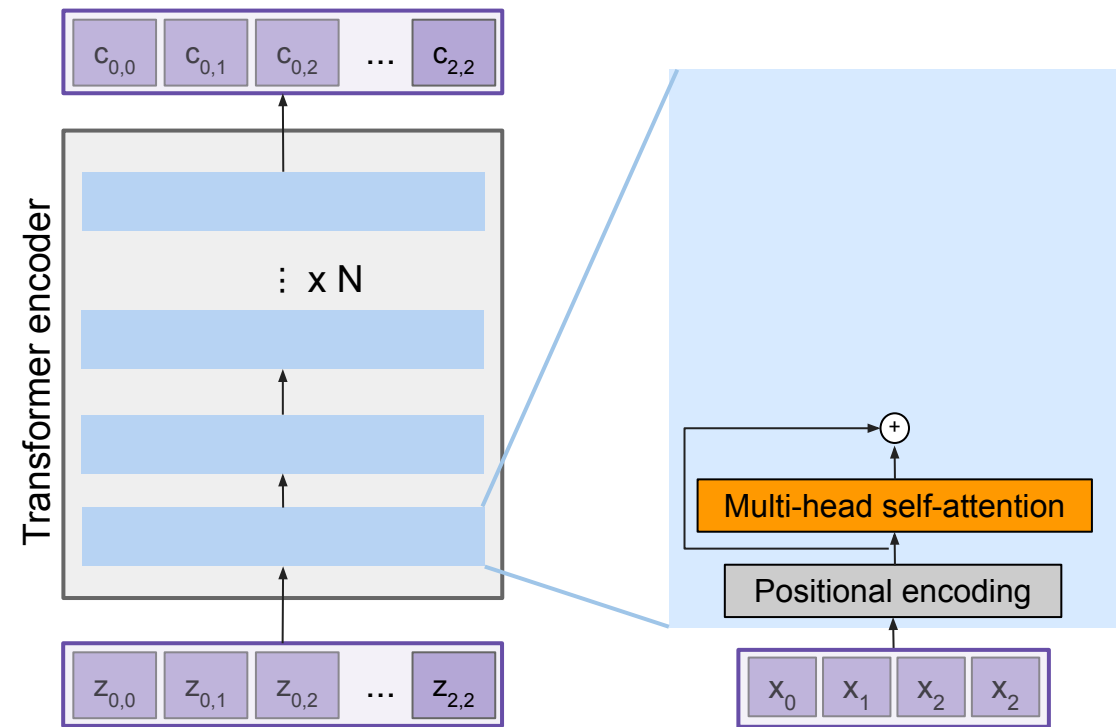
Vaswani et al, "Attention is all you need", NeurIPS 2017

The Transformer encoder block



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

The Transformer encoder block



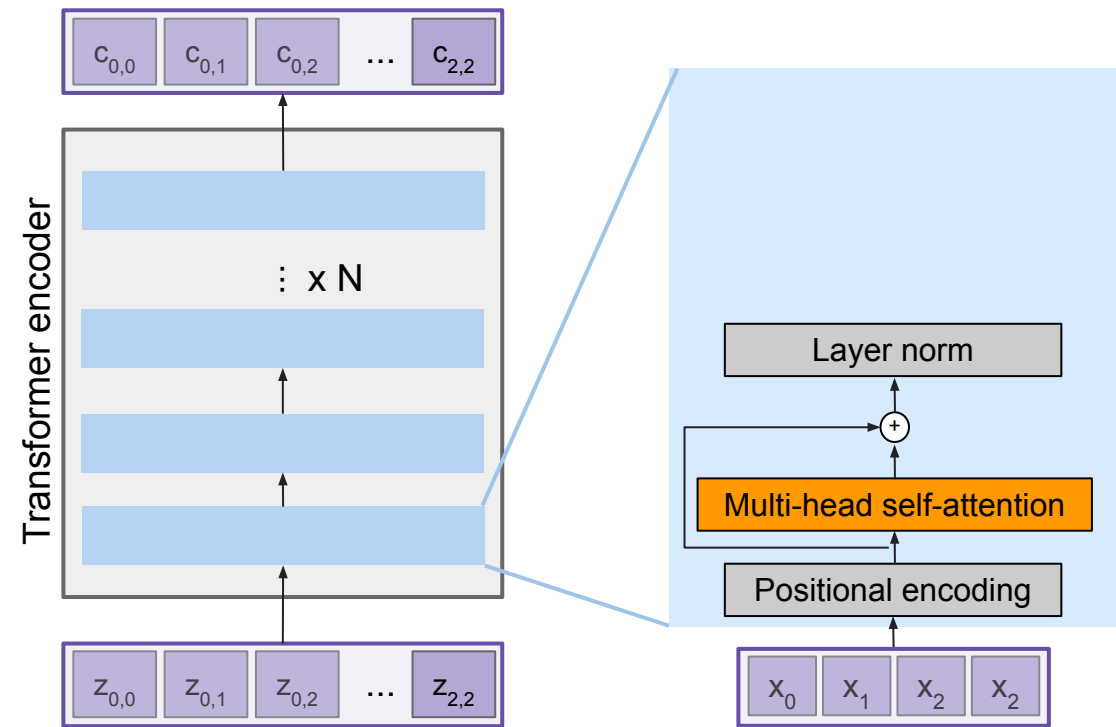
Residual connection

Attention attends over all the vectors

Add positional encoding

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

The Transformer encoder block



LayerNorm over each vector individually

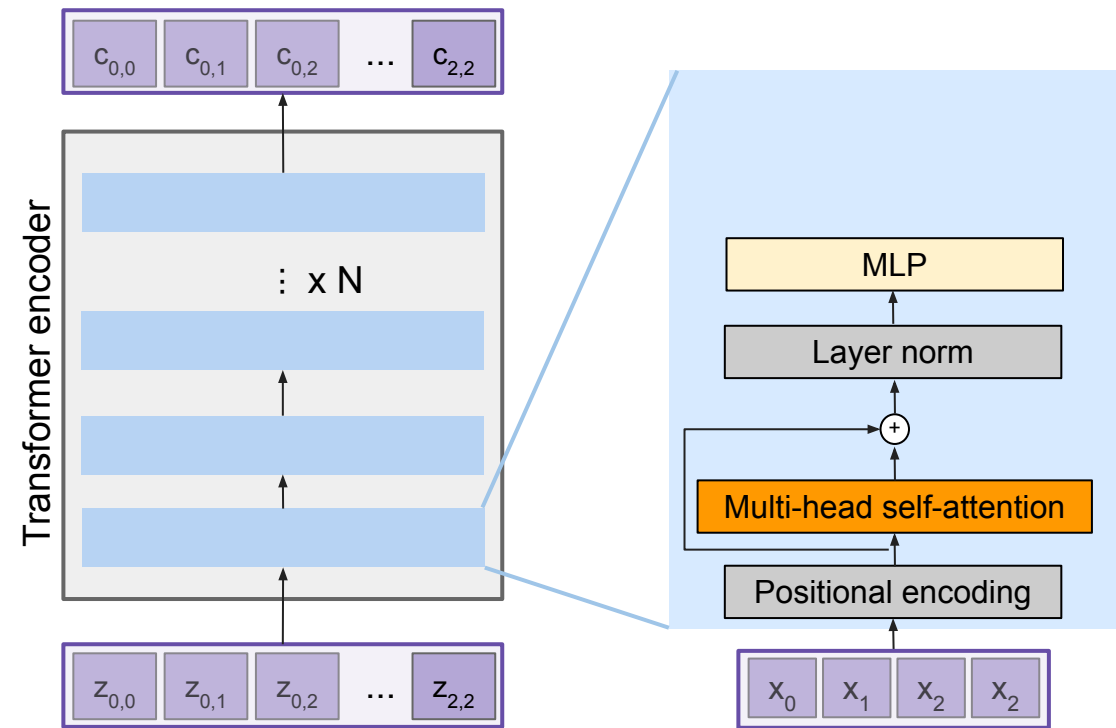
Residual connection

Attention attends over all the vectors

Add positional encoding

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

The Transformer encoder block



MLP over each vector individually

LayerNorm over each vector individually

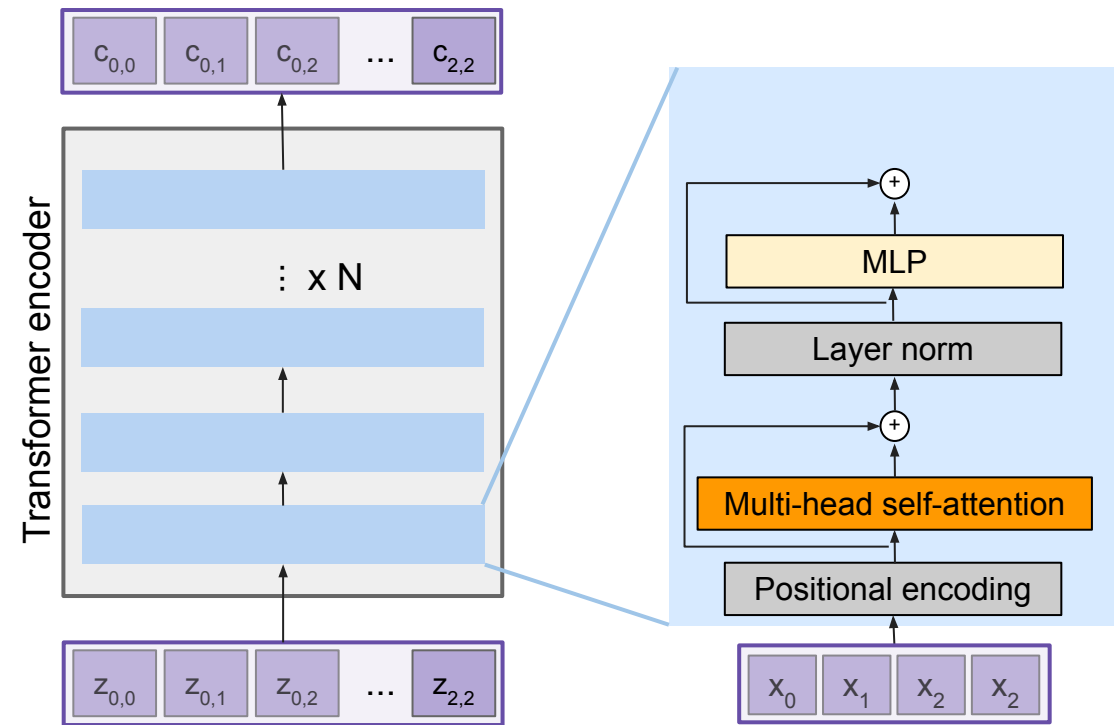
Residual connection

Attention attends over all the vectors

Add positional encoding

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

The Transformer encoder block



Residual connection

MLP over each vector individually

LayerNorm over each vector individually

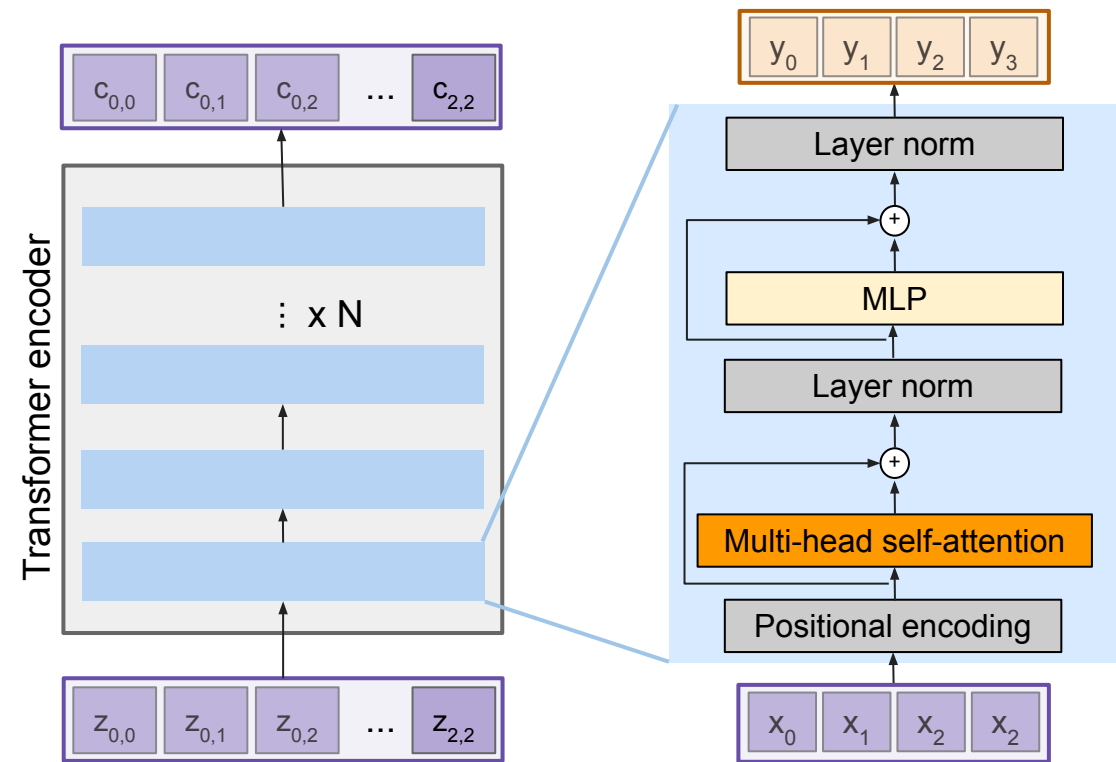
Residual connection

Attention attends over all the vectors

Add positional encoding

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

The Transformer encoder block



Transformer Encoder Block:

Inputs: Set of vectors \mathbf{x}

Outputs: Set of vectors \mathbf{y}

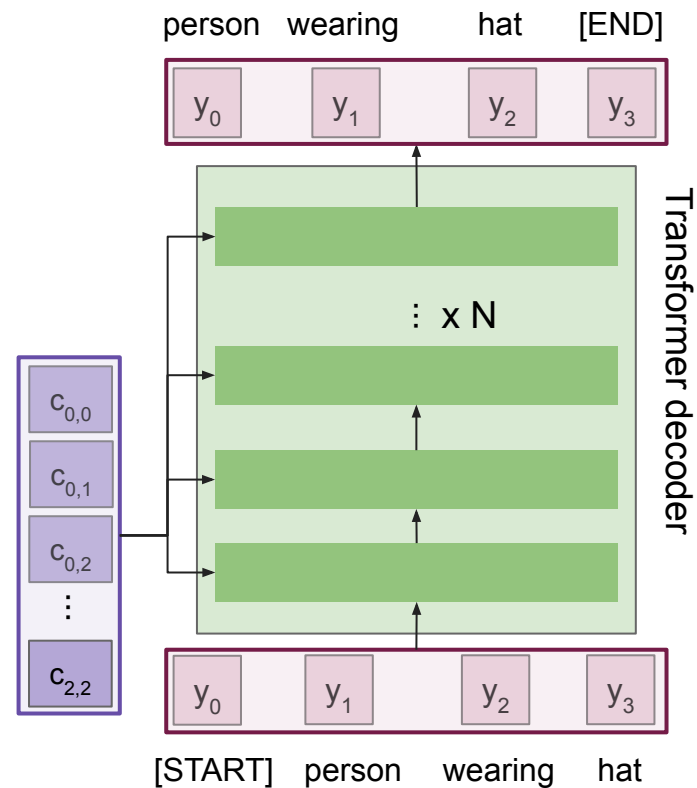
Self-attention is the only interaction between vectors.

Layer norm and MLP operate independently per vector.

Highly scalable, highly parallelizable, but high memory usage.

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

The Transformer decoder block

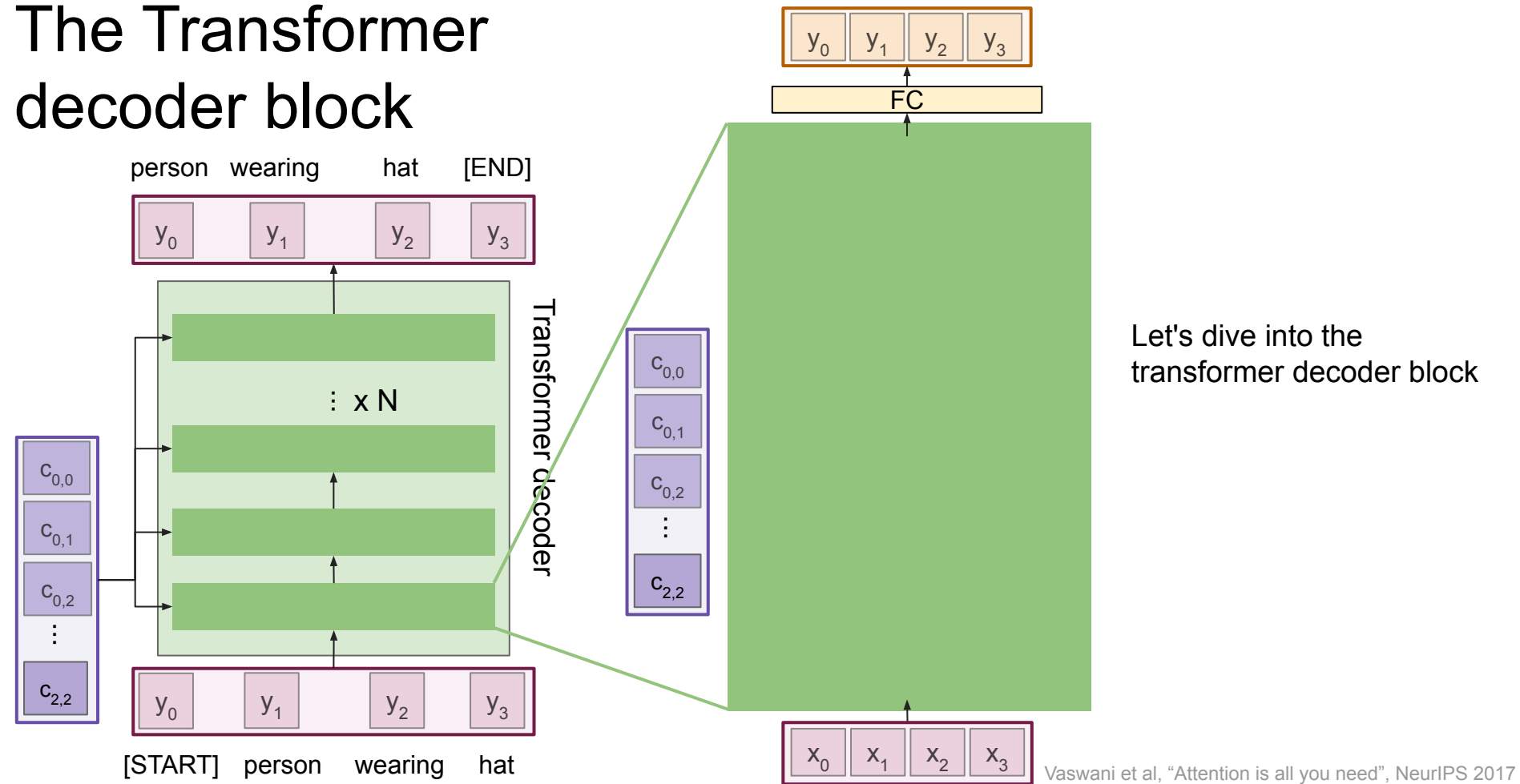


Made up of N decoder blocks.

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

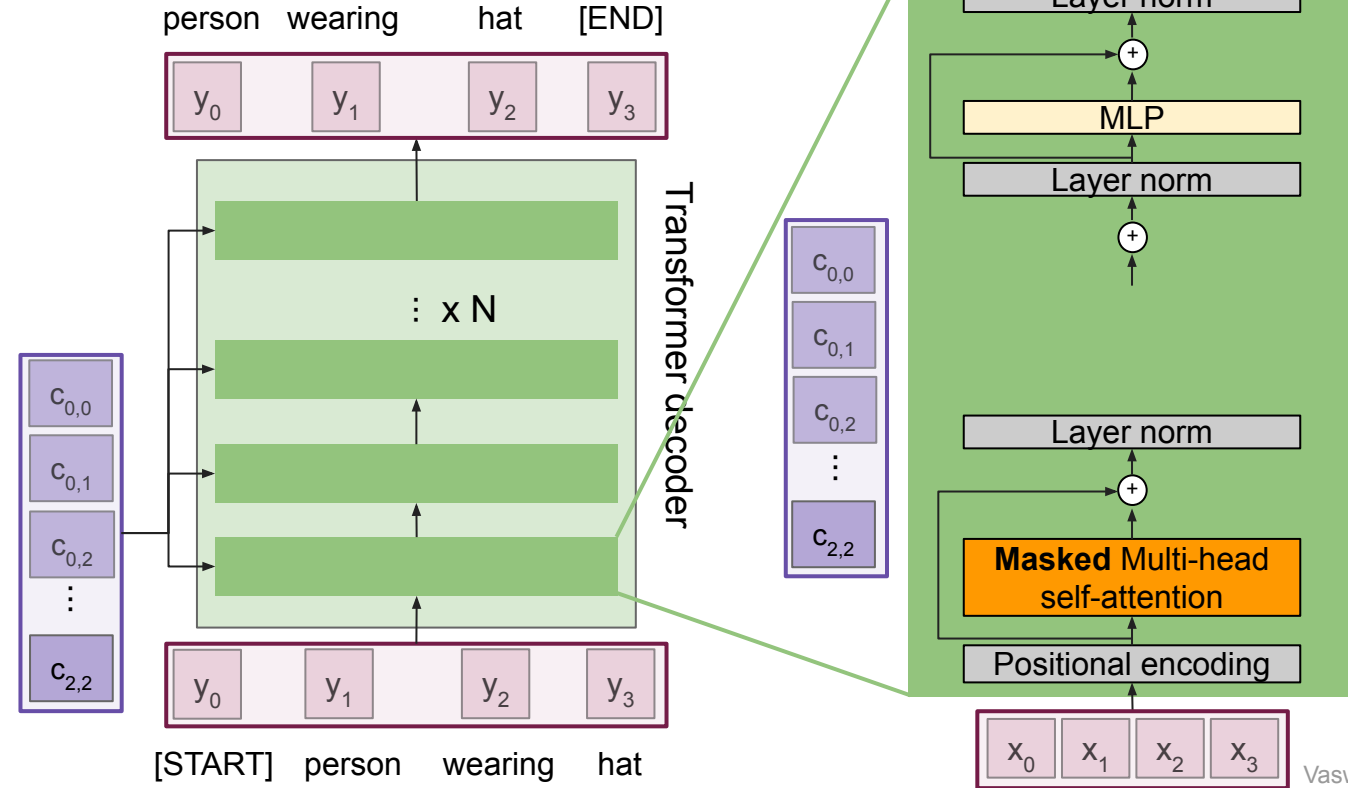
Vaswani et al, "Attention is all you need", NeurIPS 2017

The Transformer decoder block



The Transformer

Decoder block

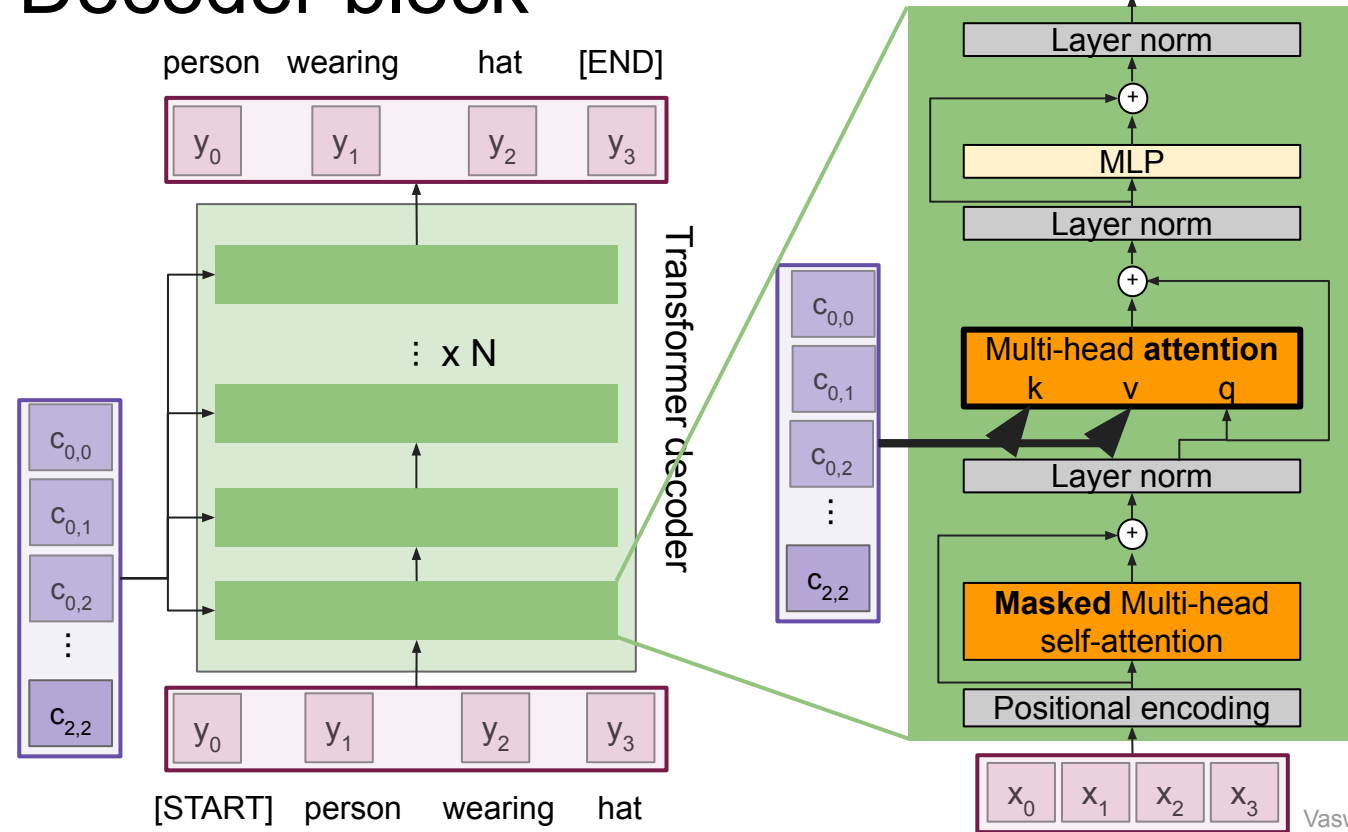


Most of the network is the same the transformer encoder.

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

The Transformer

Decoder block



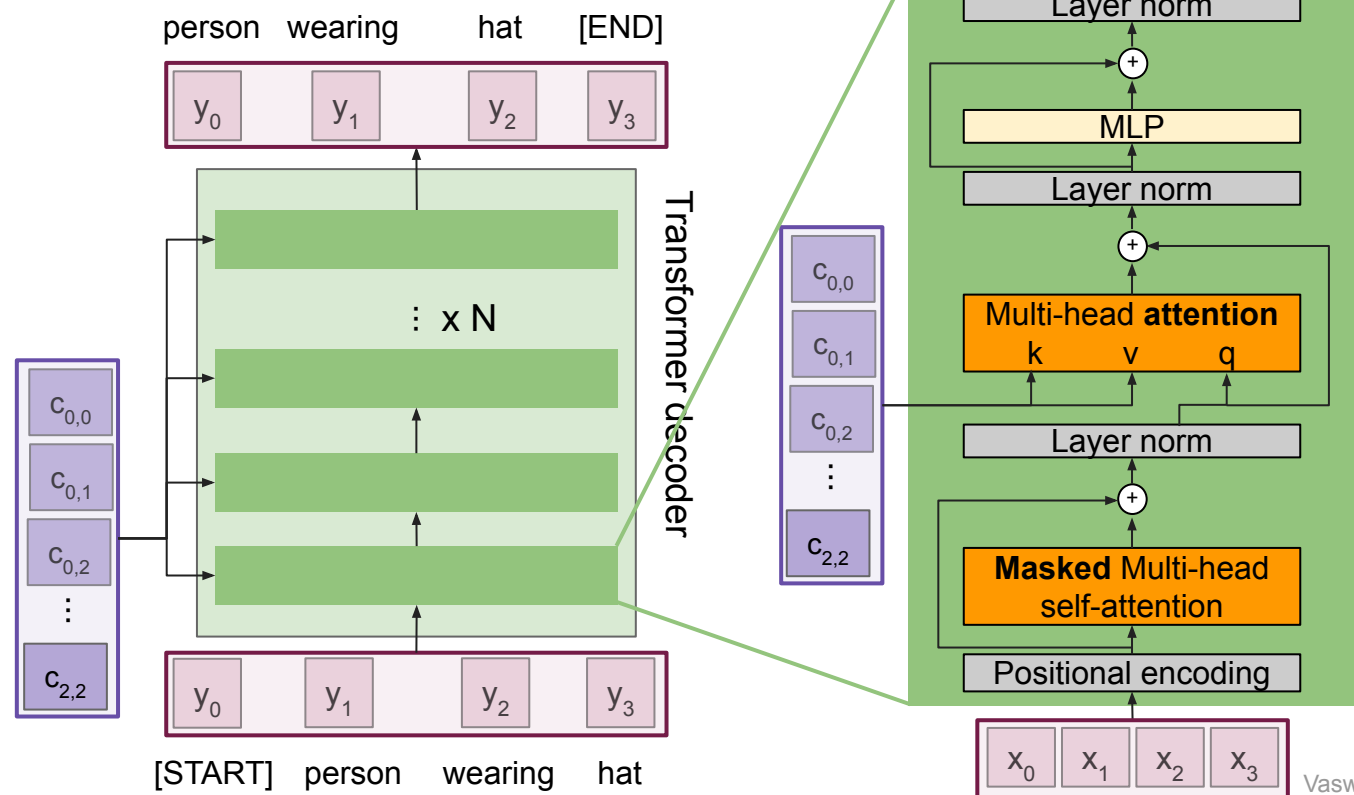
Multi-head attention block attends over the transformer encoder outputs.

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

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

The Transformer

Decoder block



Transformer Decoder Block:

Inputs: Set of vectors \mathbf{x} and Set of context vectors \mathbf{c} .

Outputs: Set of vectors \mathbf{y} .

Masked Self-attention only interacts with past inputs.

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

Highly scalable, highly parallelizable, but high memory usage.

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

Image Captioning using transformers

- No recurrence at all

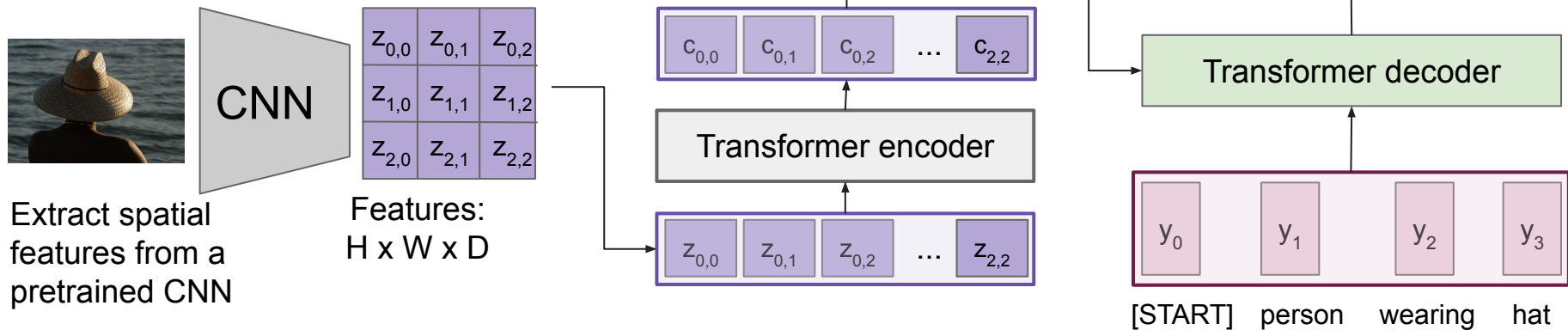


Image Captioning using transformers

- Perhaps we don't need convolutions at all?

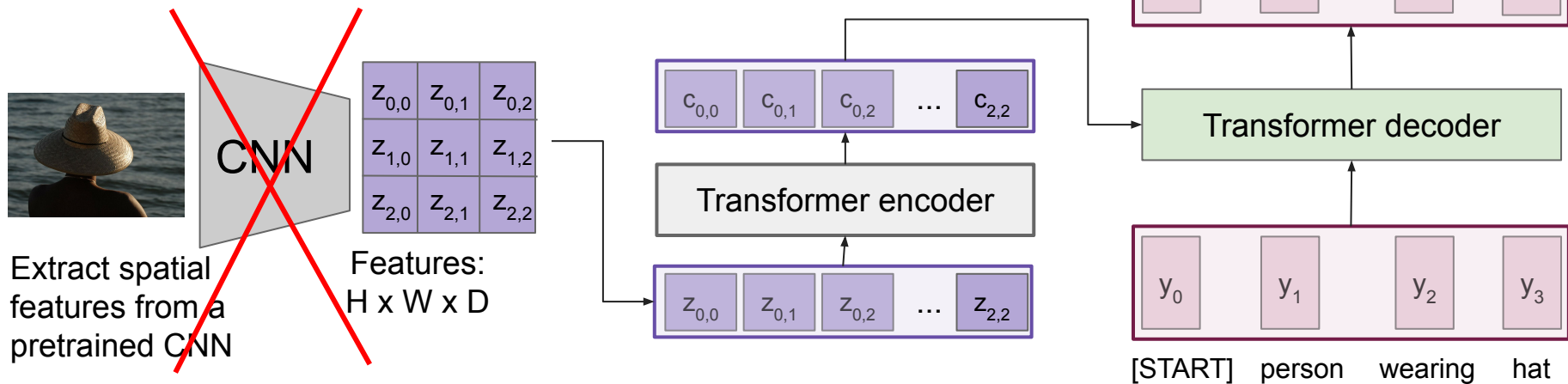
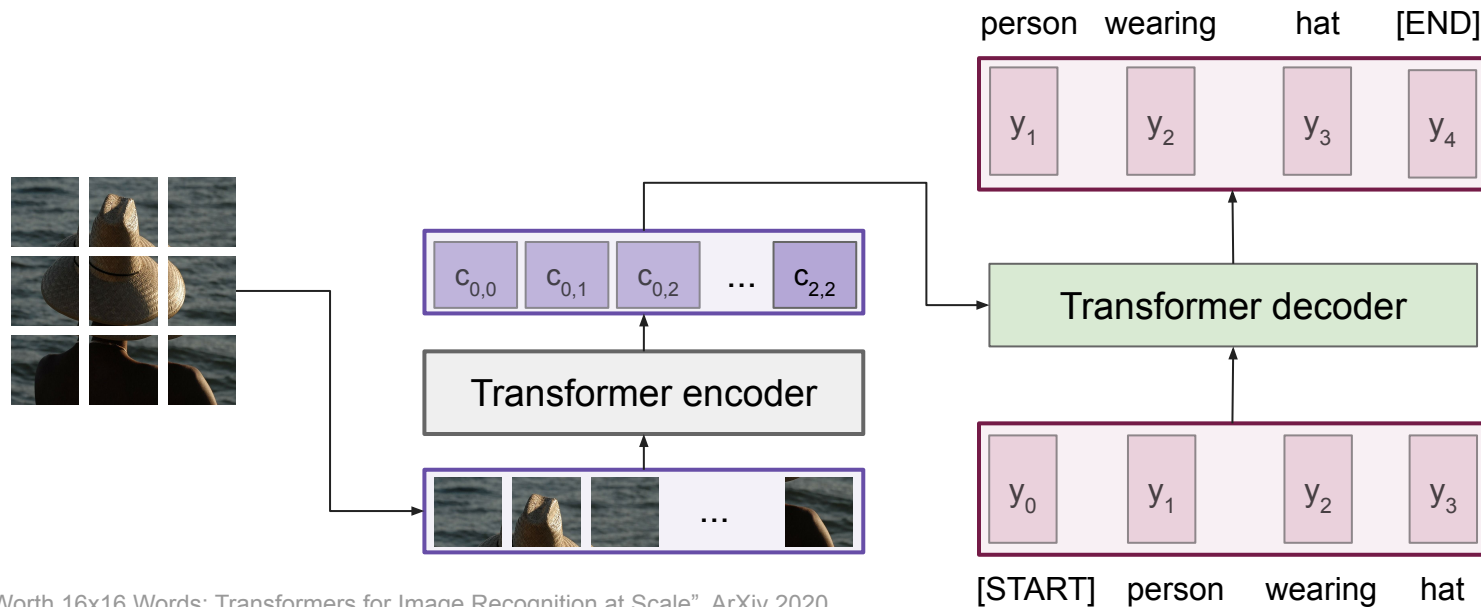


Image Captioning using **ONLY** transformers

- Transformers from pixels to language



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

Vision Transformers vs. ResNets

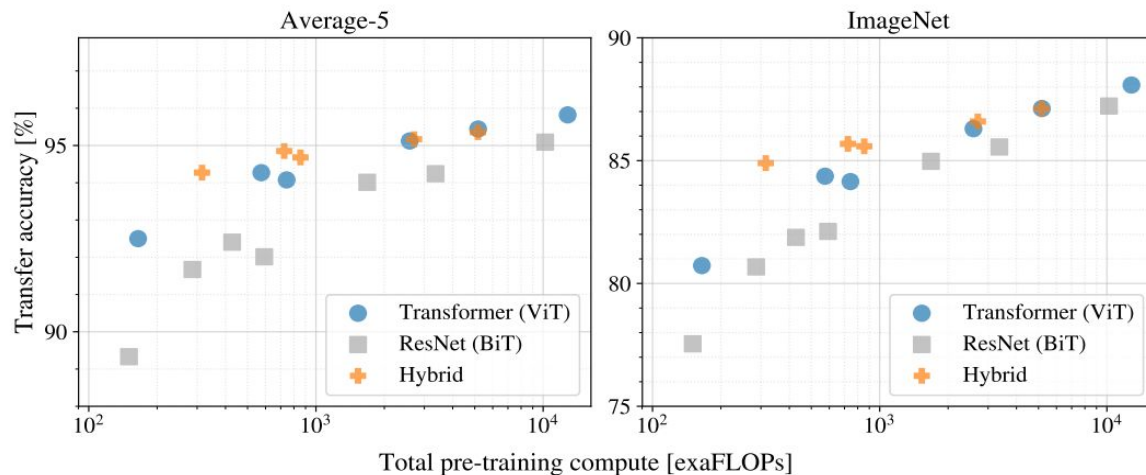
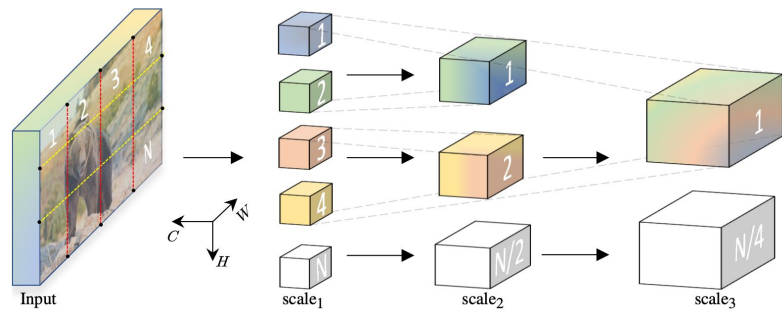


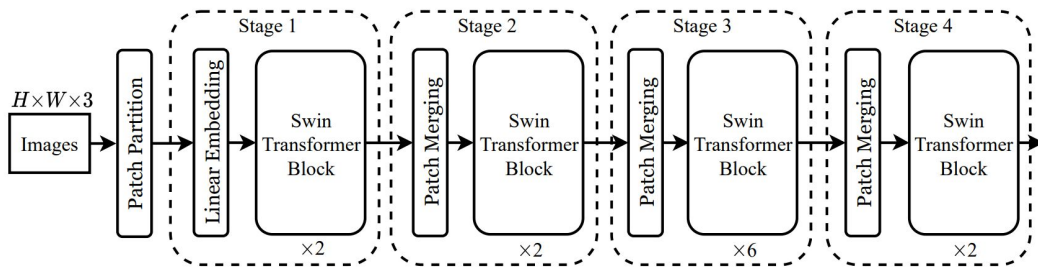
Figure 5: Performance versus cost for different architectures: Vision Transformers, ResNets, and hybrids. Vision Transformers generally outperform ResNets with the same computational budget. Hybrids improve upon pure Transformers for smaller model sizes, but the gap vanishes for larger models.

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

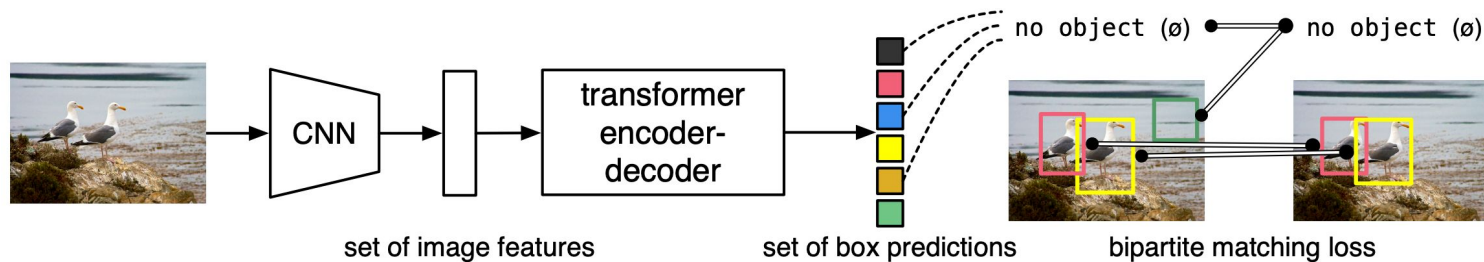
Vision Transformers



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

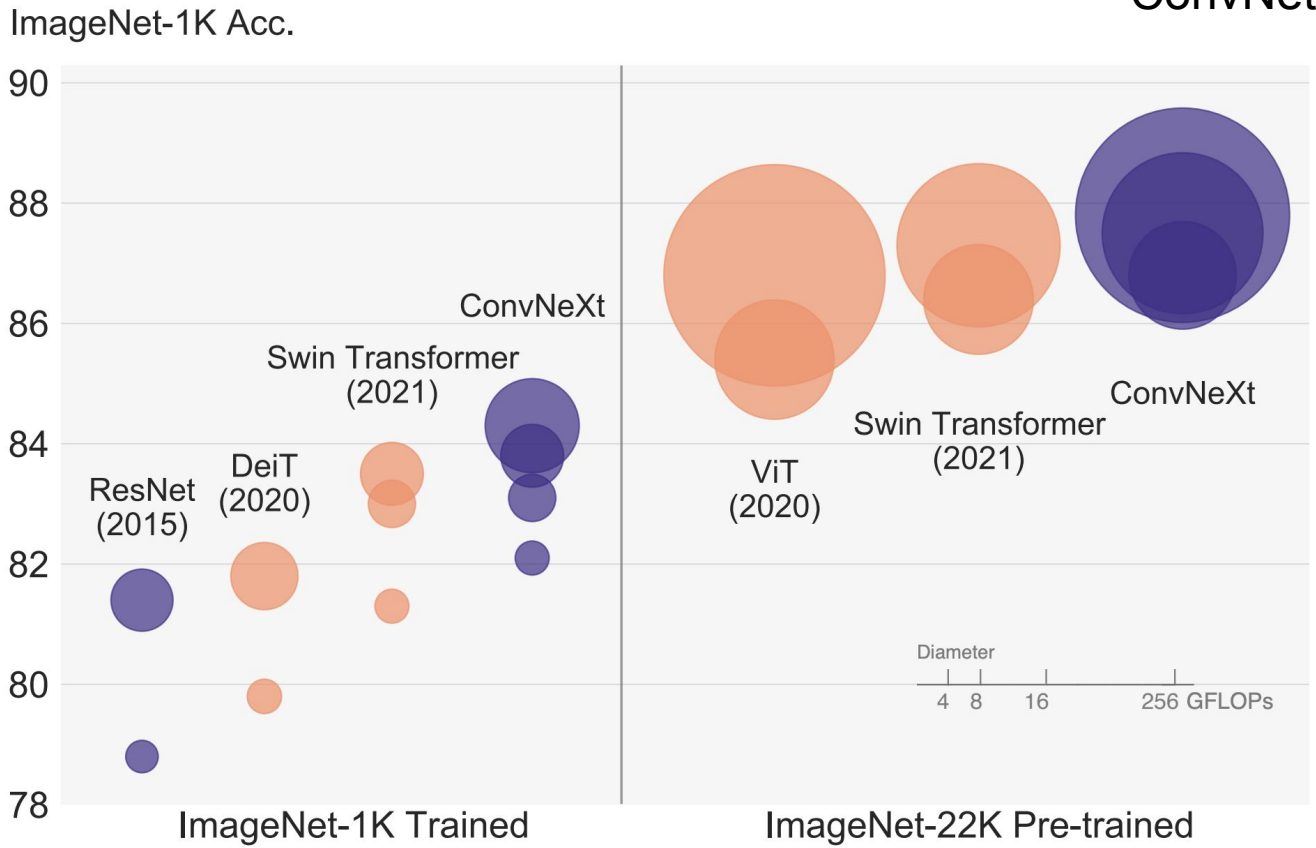


Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021



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

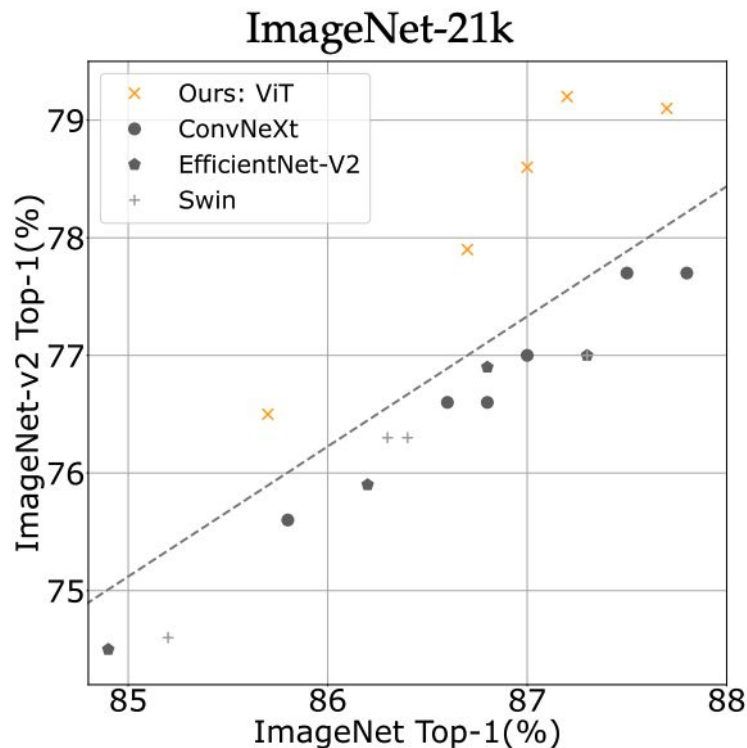
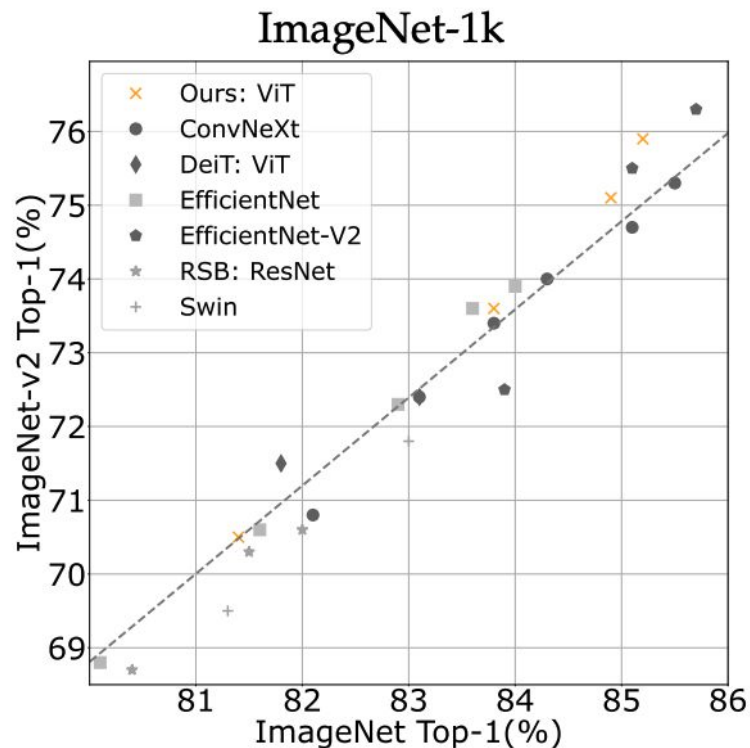
ConvNets strike back!



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

DeiT III: Revenge of the ViT

Hugo Touvron^{*,†} Matthieu Cord[†] Hervé Jégou^{*}



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: Video Understanding