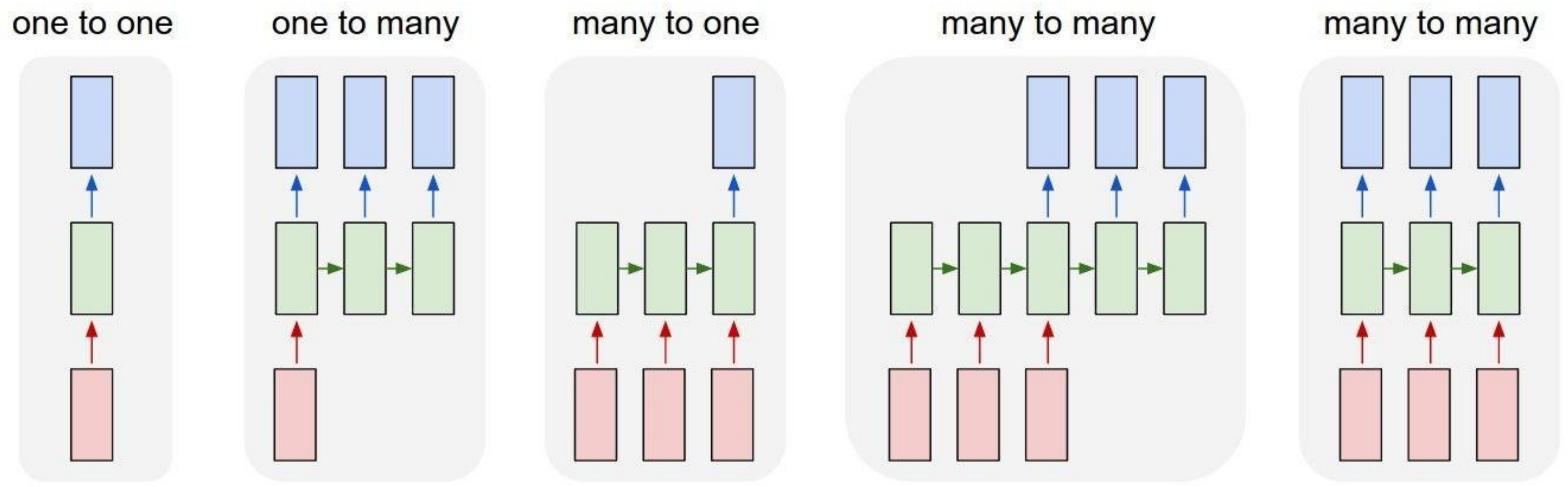
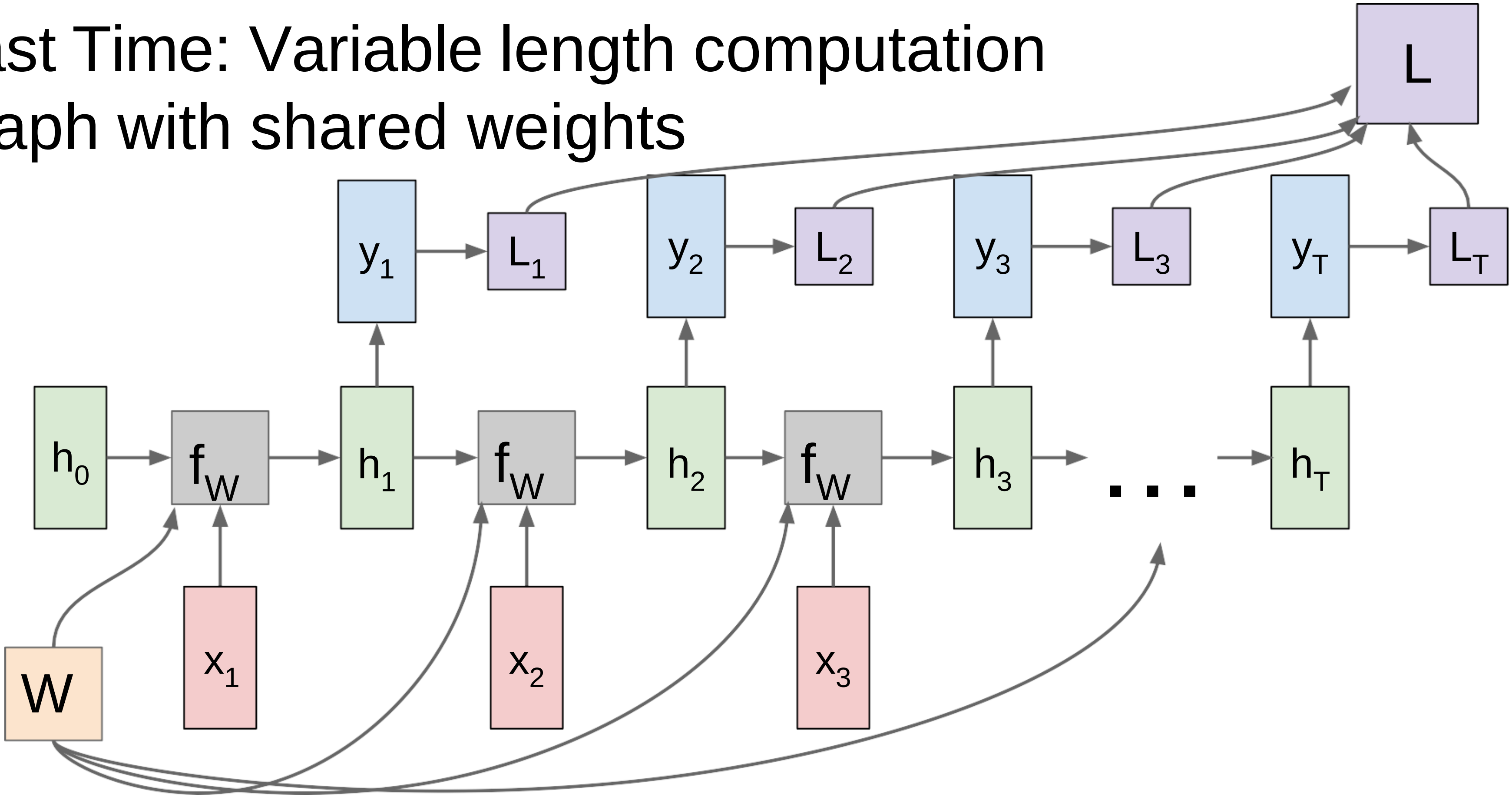
Lecture 9:
Attention and Transformers

Credit: CS231N at Stanford University

Last Time: Recurrent Neural Networks



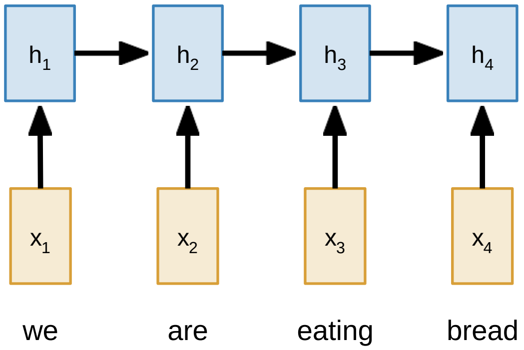
Last Time: Variable length computation graph with shared weights



Sequence to Sequence with RNNs

Input: Sequence $x_1, \dots x_T$
Output: Sequence $y_1, \dots, y_{T'}$

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



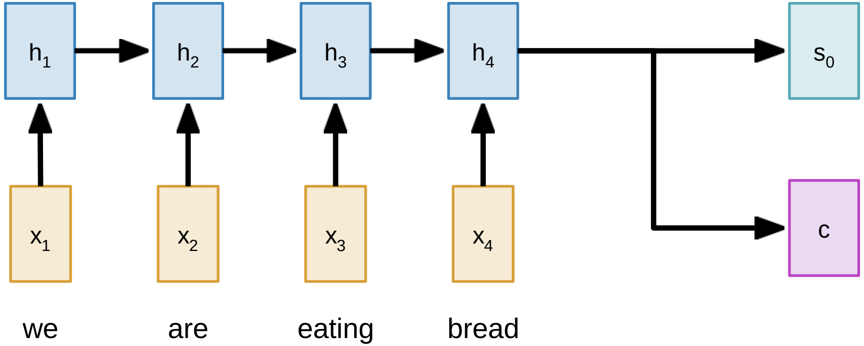
Sutskever et al, "Sequence to sequence learning with neural networks", NeurIPS 2014

Sequence to Sequence with RNNs

Input: Sequence $x_1, \ldots x_T$
Output: Sequence $y_1, \ldots, y_{T'}$

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

From final hidden state predict:
Initial decoder state s_0
Context vector c (often $c=h_T$)



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

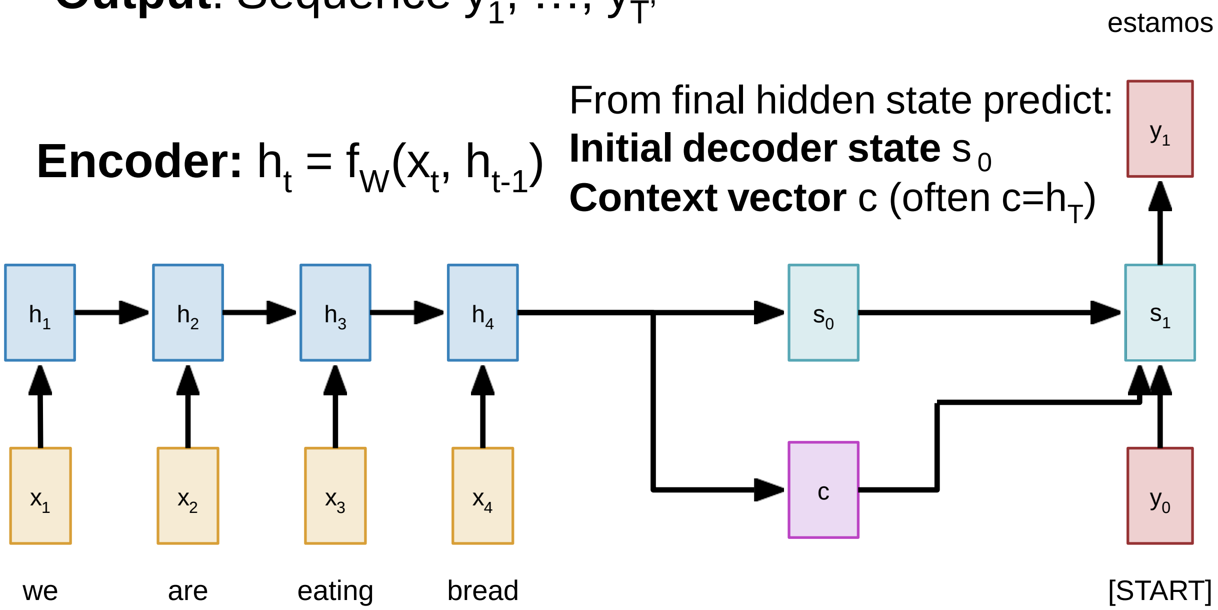
Sequence to Sequence with RNNs

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 $c=h_T$)



Sutskever et al, “Sequence to sequence learning with neural networks”, NeurIPS 2014

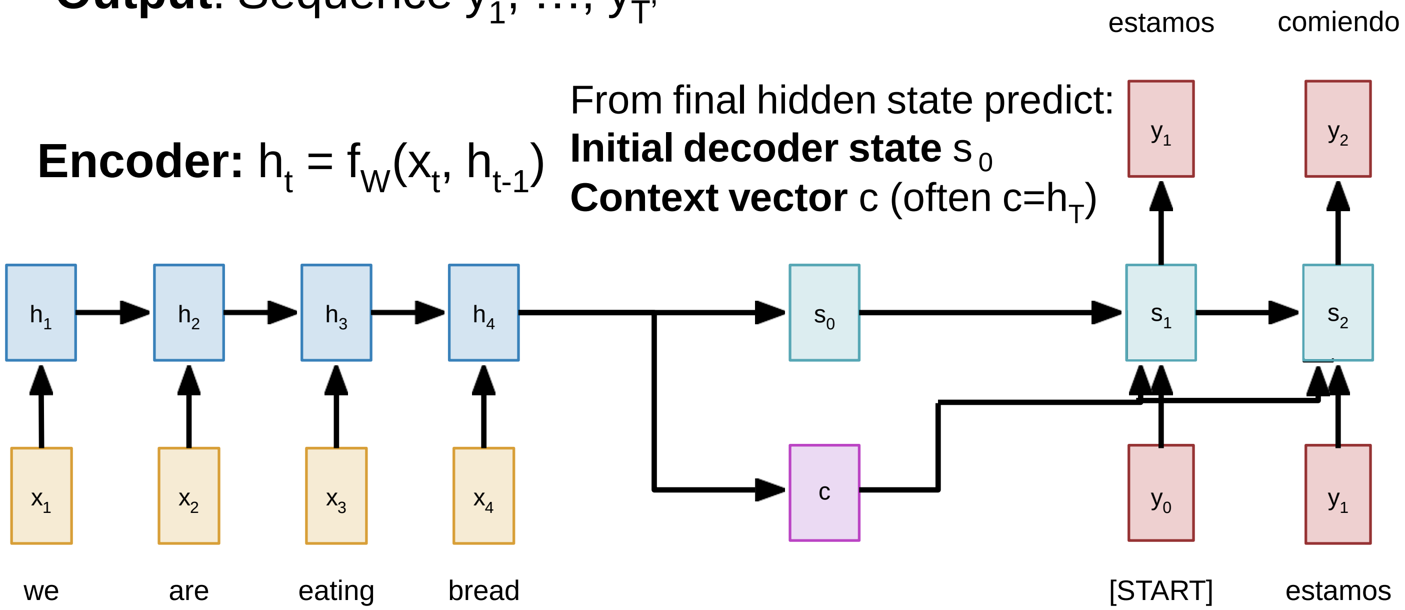
Sequence to Sequence with RNNs

Input: Sequence $x_1, \ldots x_T$
Output: Sequence $y_1, \ldots, y_{T'}$

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 $c=h_T$)



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

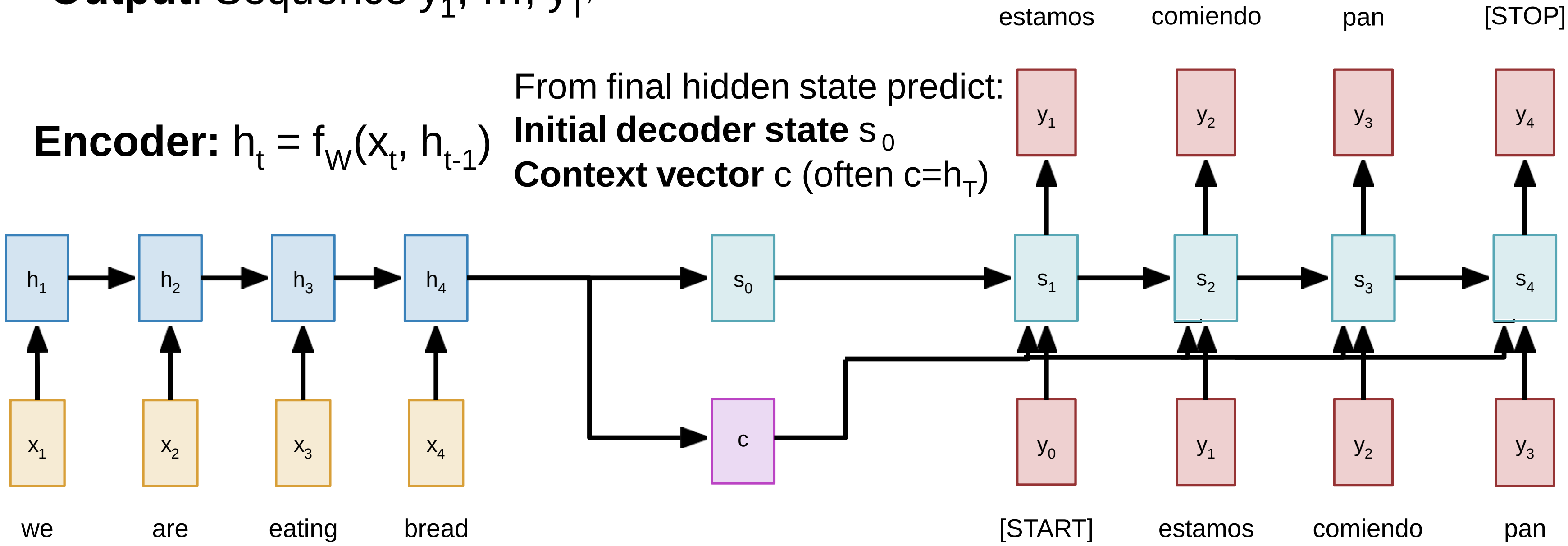
Sequence to Sequence with RNNs

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 $c=h_T$)



Sutskever et al, “Sequence to sequence learning with neural networks”, NeurIPS 2014

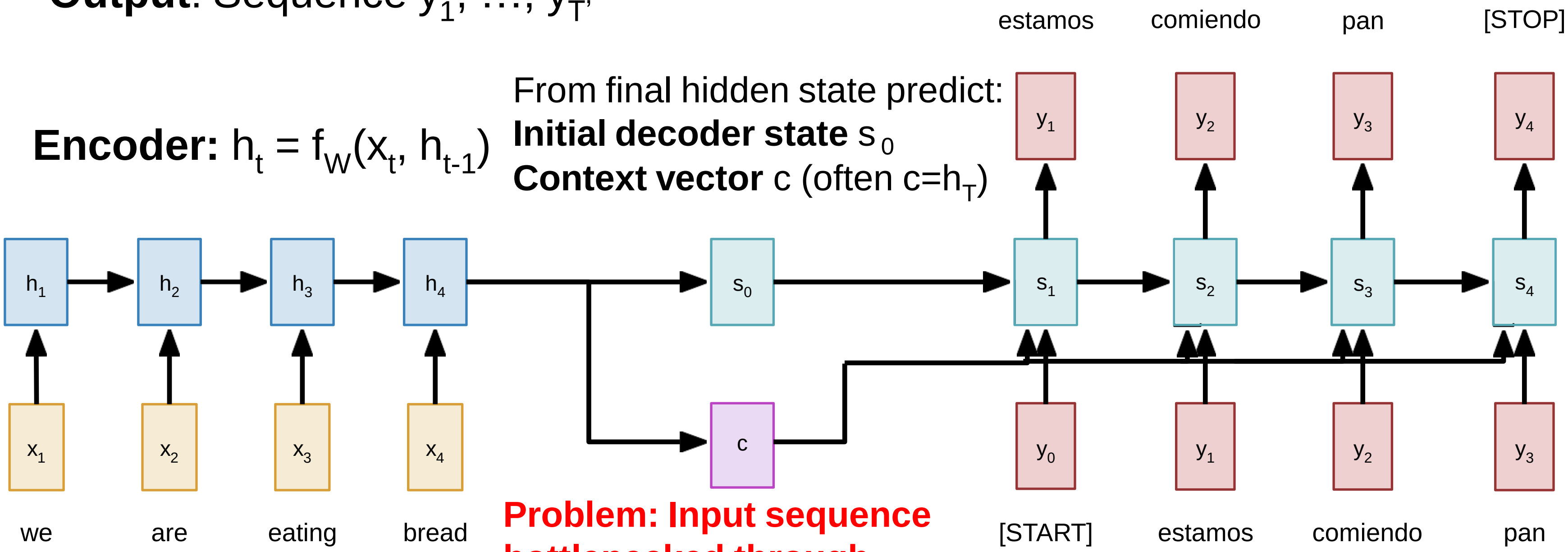
Sequence to Sequence with RNNs

Input: Sequence $x_1, \ldots x_T$
Output: Sequence $y_1, \ldots, y_{T'}$

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

From final hidden state predict:
Initial decoder state s_0
Context vector c (often $c=h_T$)

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



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

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

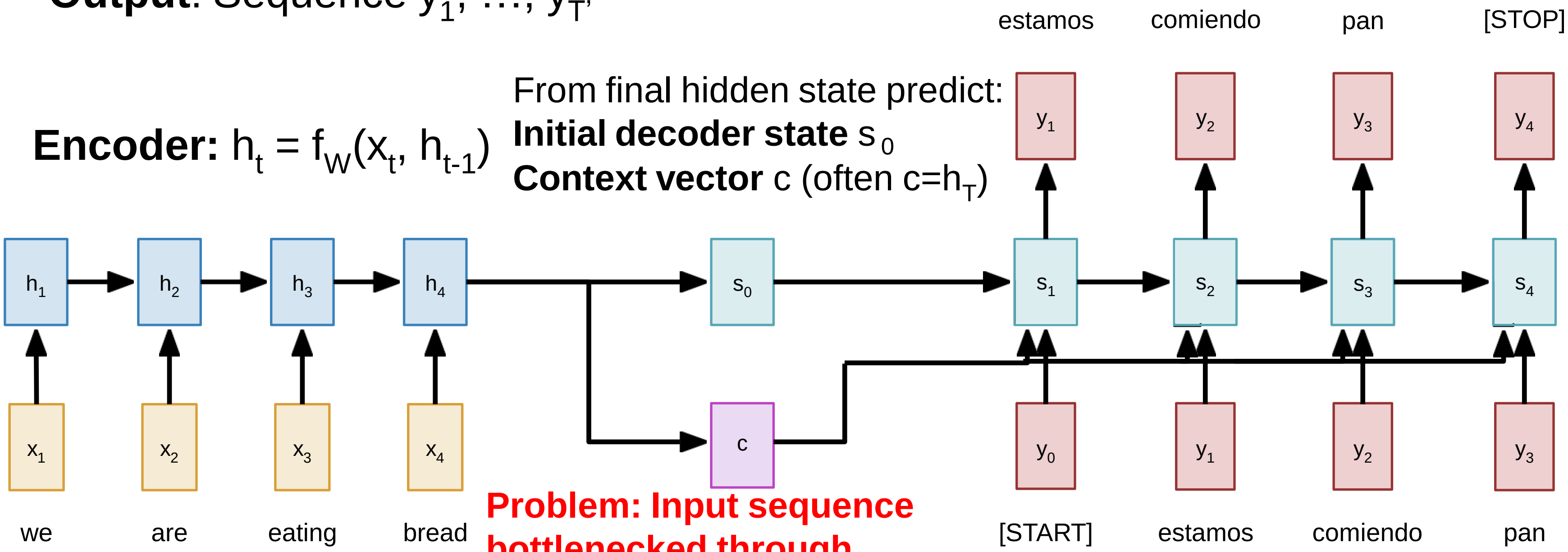
Sequence to Sequence with RNNs

Input: Sequence $x_1, \ldots x_T$
Output: Sequence $y_1, \ldots, y_{T'}$

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!

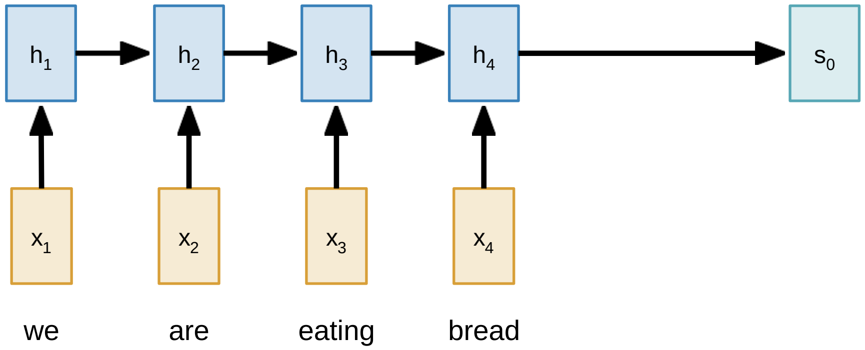
Sutskever et al, "Sequence to sequence learning with neural networks", NeurIPS 2014

Sequence to Sequence with RNNs and Attention

Input: Sequence $x_1, \ldots x_T$
Output: Sequence $y_1, \ldots, y_{T'}$

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

From final hidden state:
Initial decoder state s_0

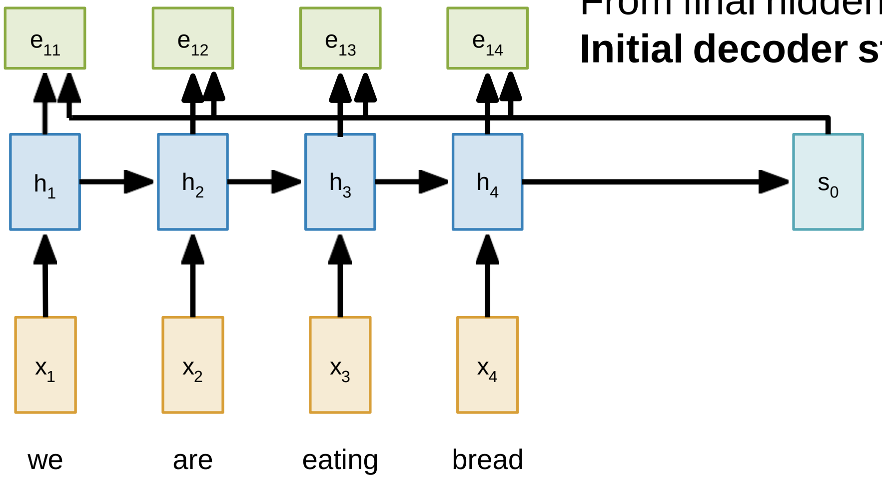


Bahdanau et al, “Neural machine translation by jointly learning to align and translate”, ICLR 2015

Sequence to Sequence with RNNs and Attention

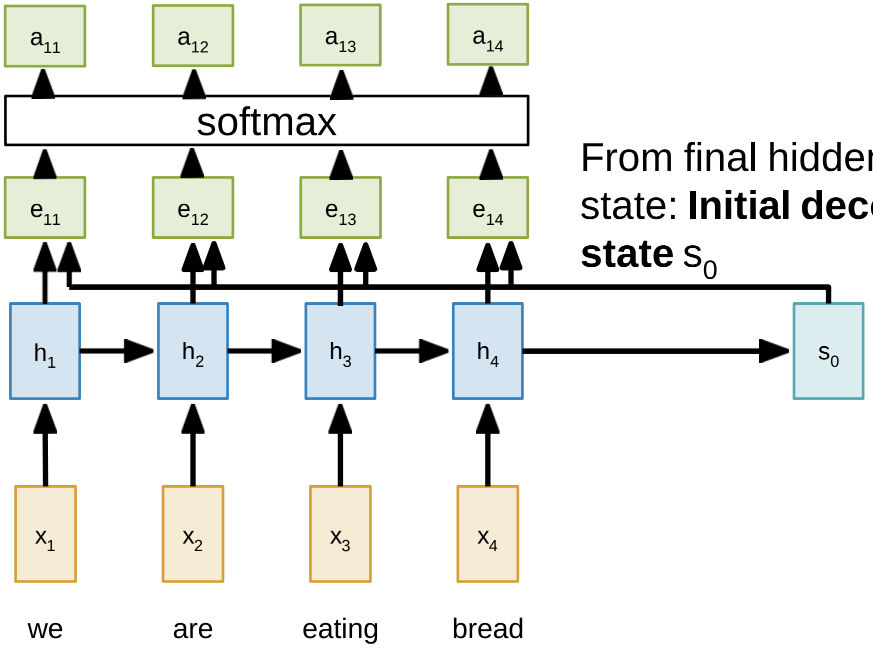
Compute (scalar) **alignment scores**
$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)

From final hidden state:
Initial decoder state s_0



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

Sequence to Sequence with RNNs and Attention



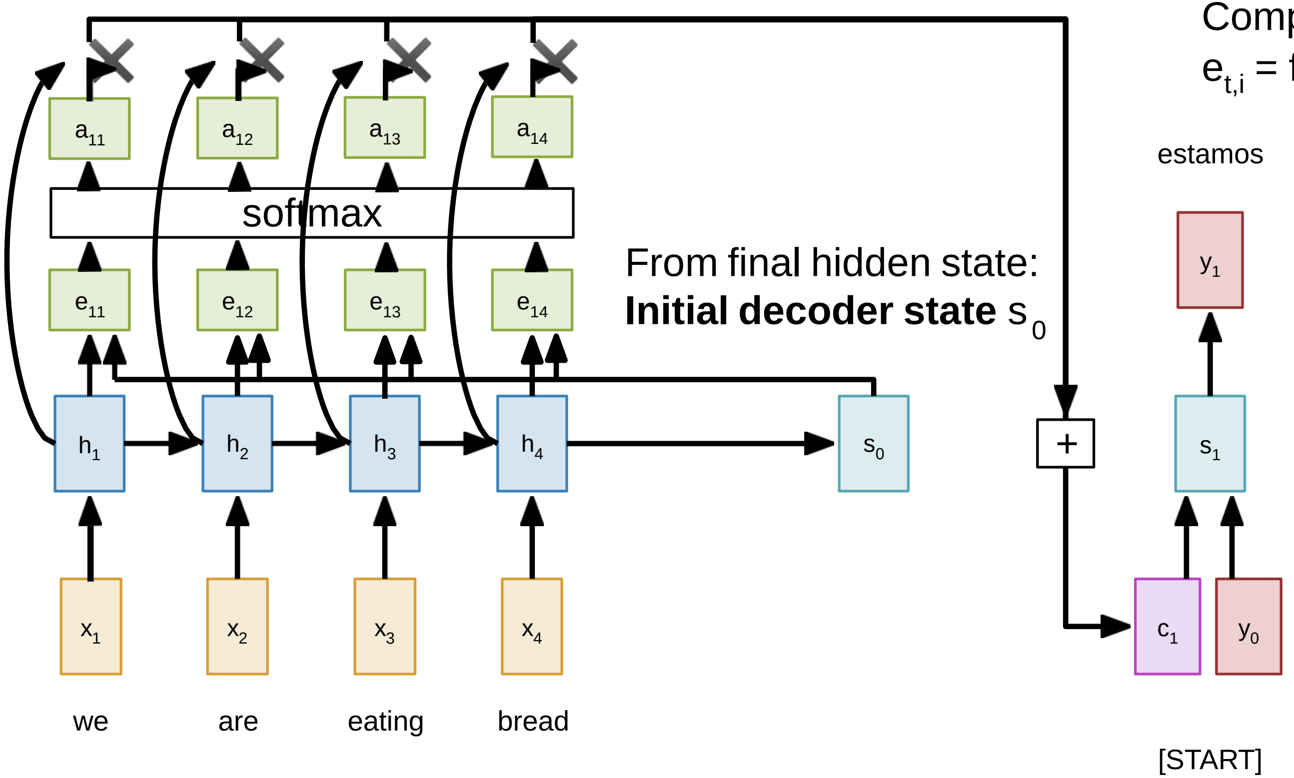
From final hidden state: **Initial decoder state** s_0

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

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

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

Sequence to Sequence with RNNs and Attention



Compute (scalar) **alignment scores**

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

Normalize alignment scores to get **attention weights**

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

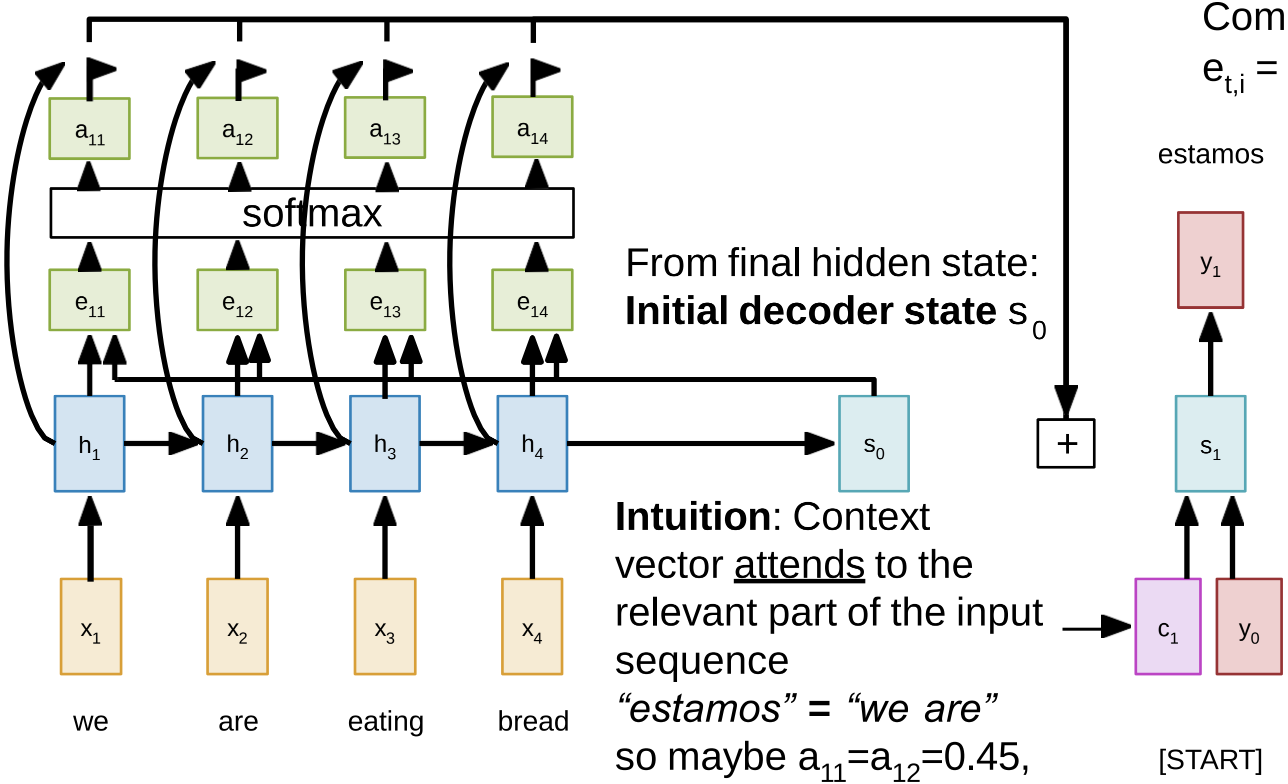
Compute context vector as linear combination of hidden states

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

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

Sequence to Sequence with RNNs and Attention

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



From final hidden state: **Initial decoder state** s_0

Intuition: Context vector <u>attends</u> to the relevant part of the input sequence
"estamos" **=** *"we are"*
so maybe $a_{11}=a_{12}=0.45$, $a_{13}=a_{14}=0.05$

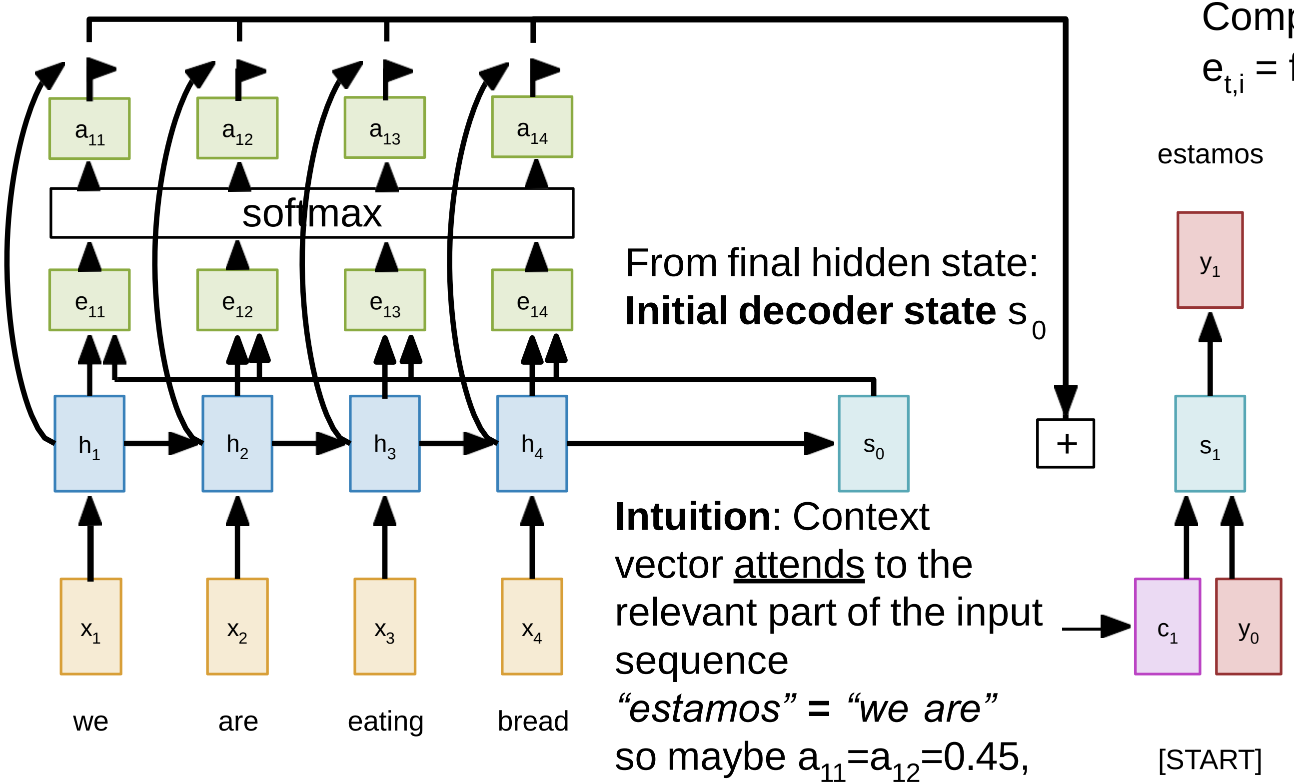
Normalize alignment scores to get **attention weights**
$0 < a_{t,i} < 1$ $\sum_i a_{t,i} = 1$

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

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

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

Sequence to Sequence with RNNs and Attention



From final hidden state: **Initial decoder state** s_0

Intuition: Context vector <u>attends</u> 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**

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

Normalize alignment scores to get **attention weights**

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

Compute context vector as linear combination of hidden states

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

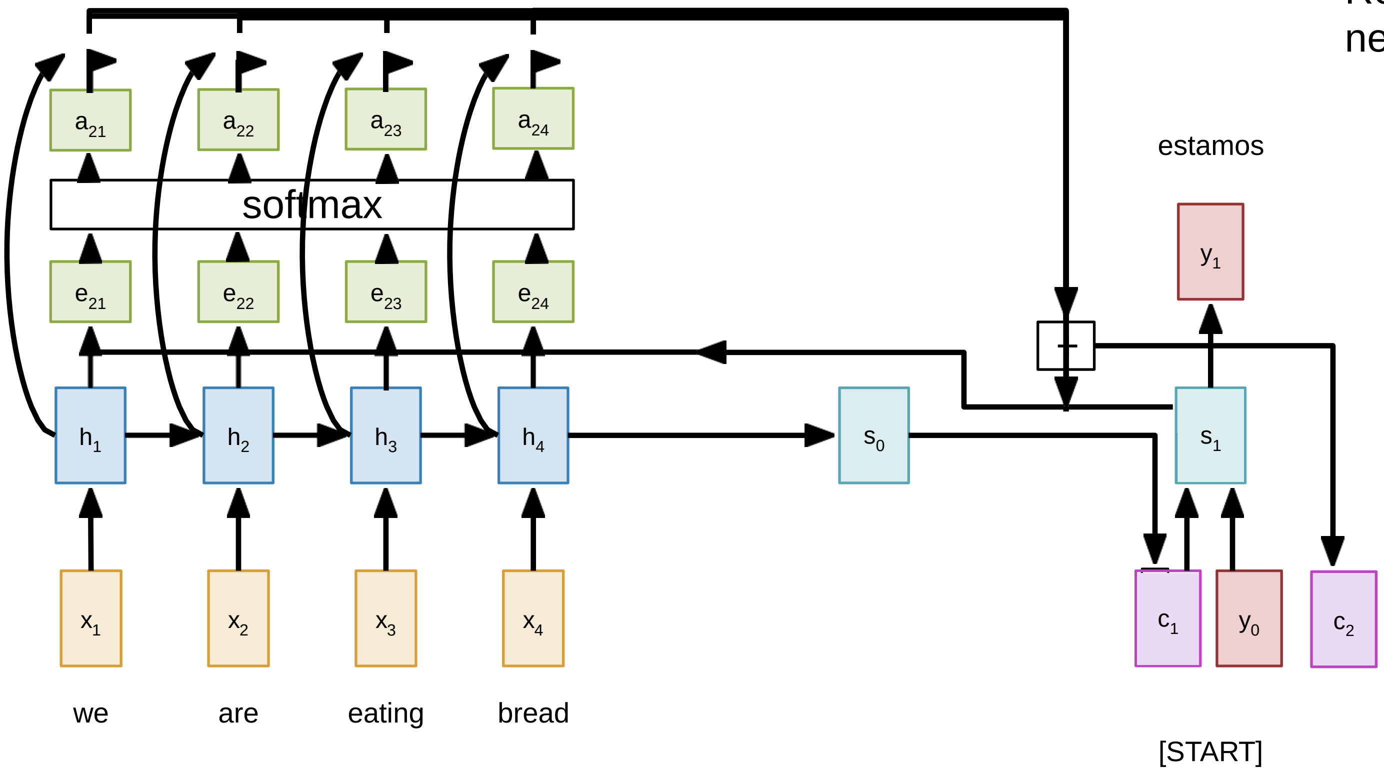
Use context vector in decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c_t)$

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

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

Sequence to Sequence with RNNs and Attention

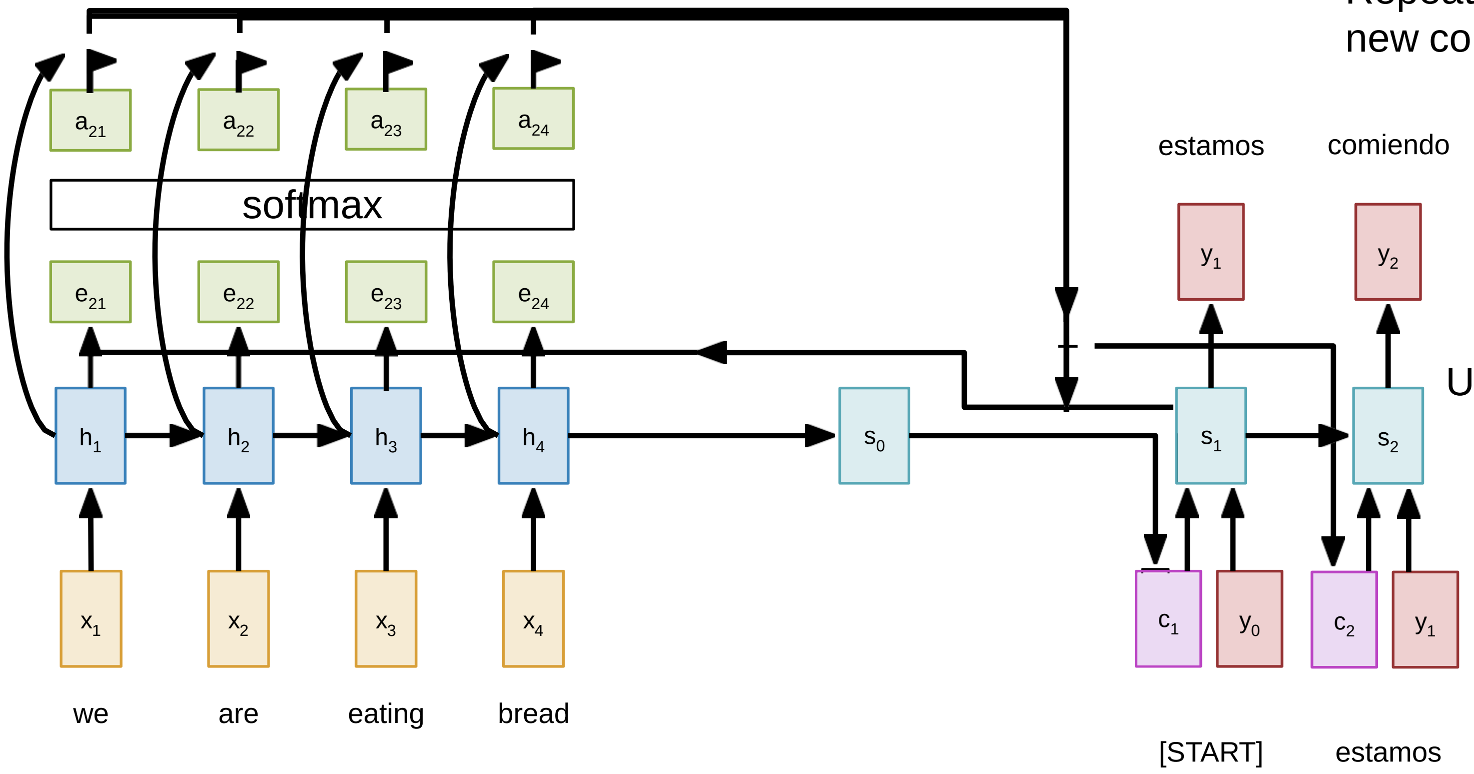
Repeat: Use s_1 to compute new context vector c_2



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

Sequence to Sequence with RNNs and Attention

Repeat: Use s_1 to compute new context vector c_2

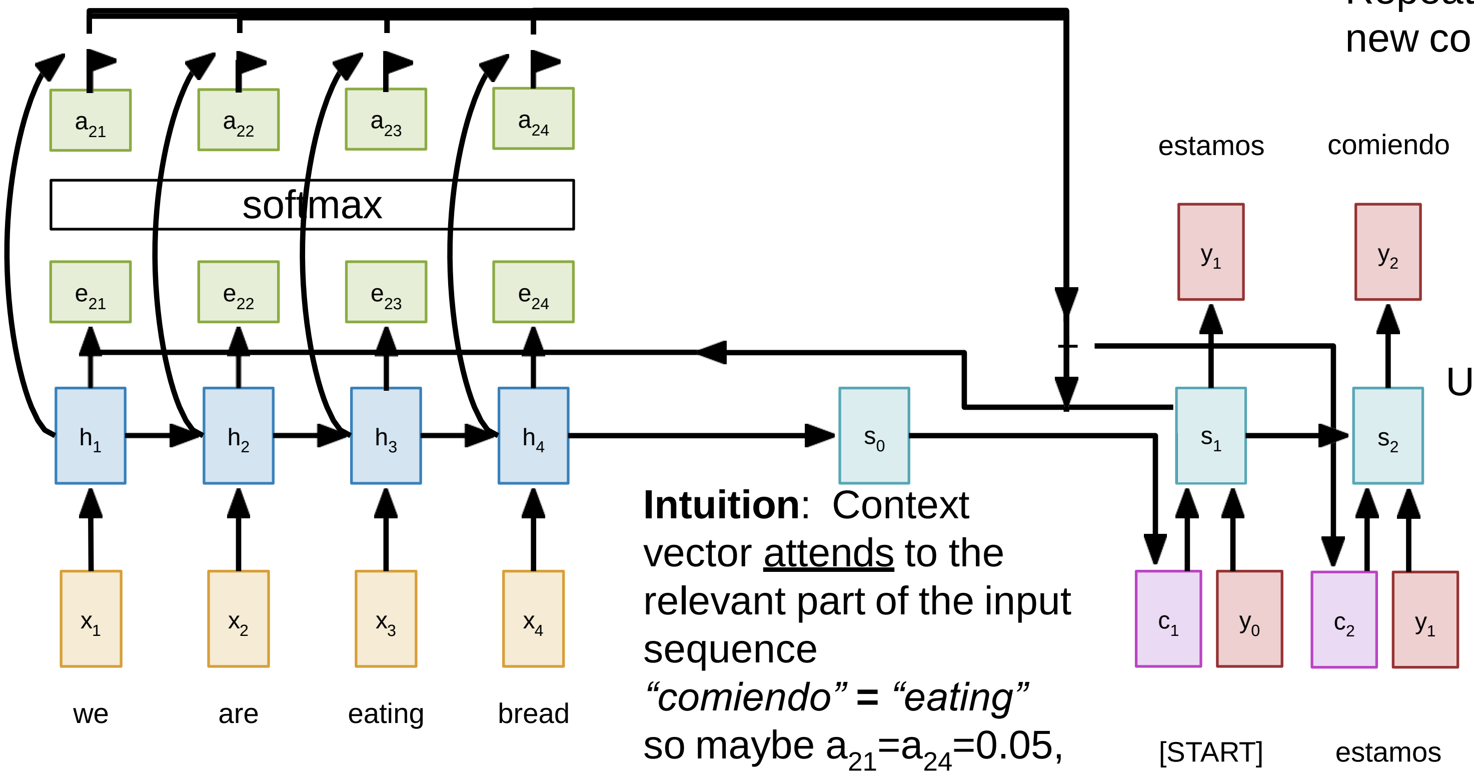


Use c_2 to compute s_2, y_2

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

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

Intuition: Context vector <u>attends</u> to the relevant part of the input sequence

"comiendo" **=** *"eating"*

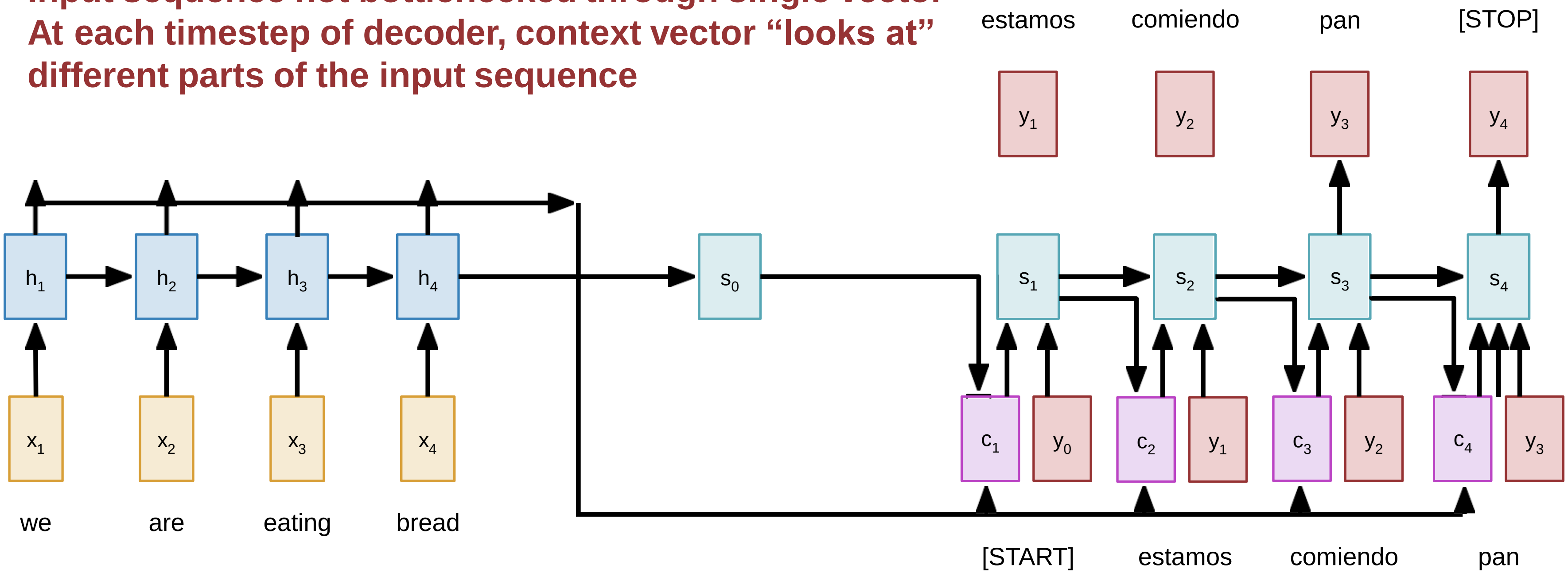
so maybe $a_{21}=a_{24}=0.05$, $a_{22}=0.1$, $a_{23}=0.8$

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**



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

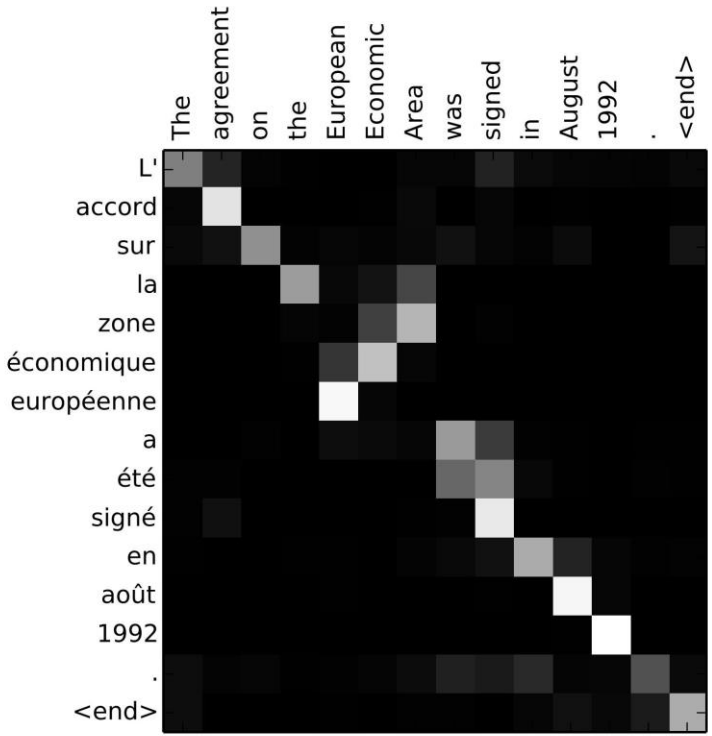
Sequence to Sequence with RNNs and Attention

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 $a_{t,i}$



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

Sequence to Sequence with RNNs and Attention

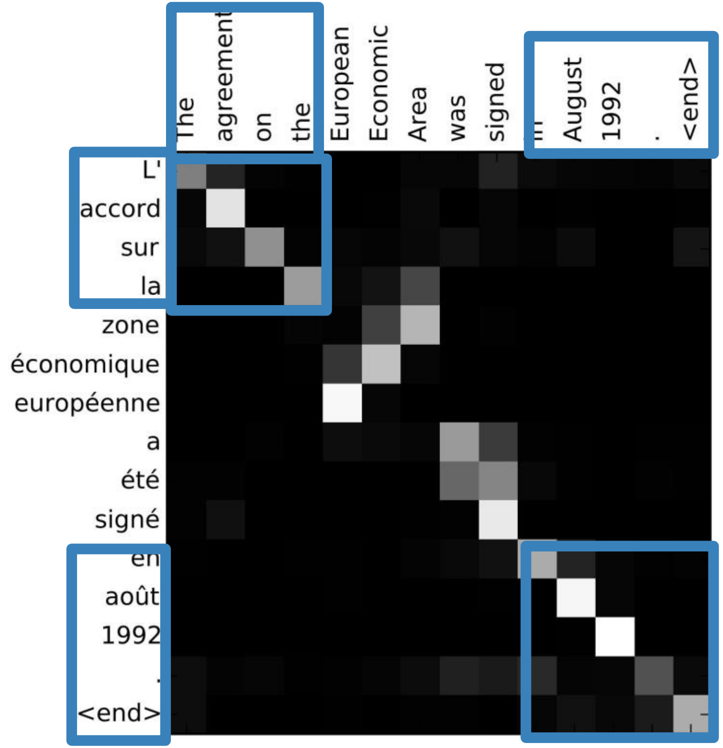
Example: English to French translation

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 $a_{t,i}$

Diagonal attention means words correspond in order



Diagonal attention means words correspond in order

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

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**."

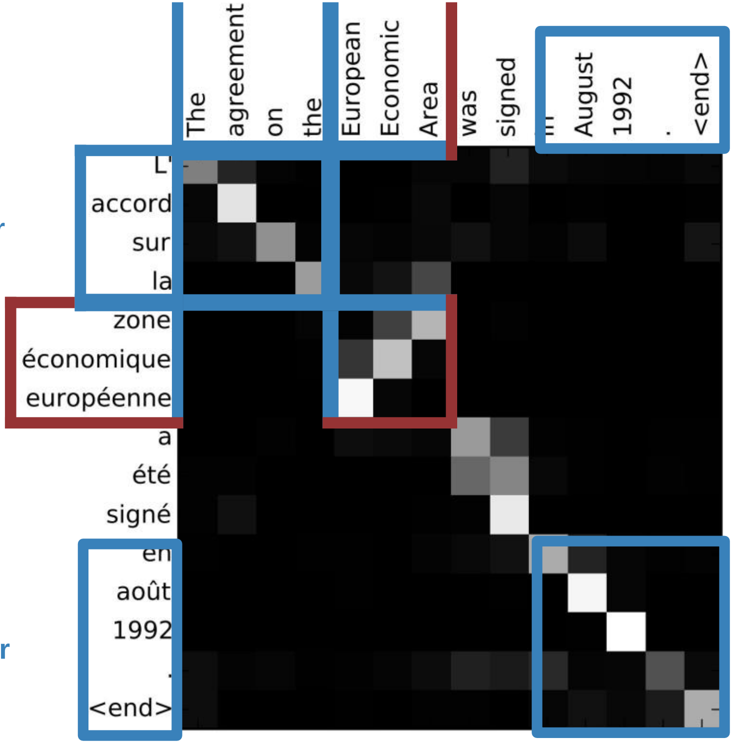
Output: "**L'accord sur la zone économique européenne** a été signé **en août 1992**."

Visualize attention weights $a_{t,i}$

Diagonal attention means words correspond in order

Attention figures out different word orders

Diagonal attention means words correspond in order

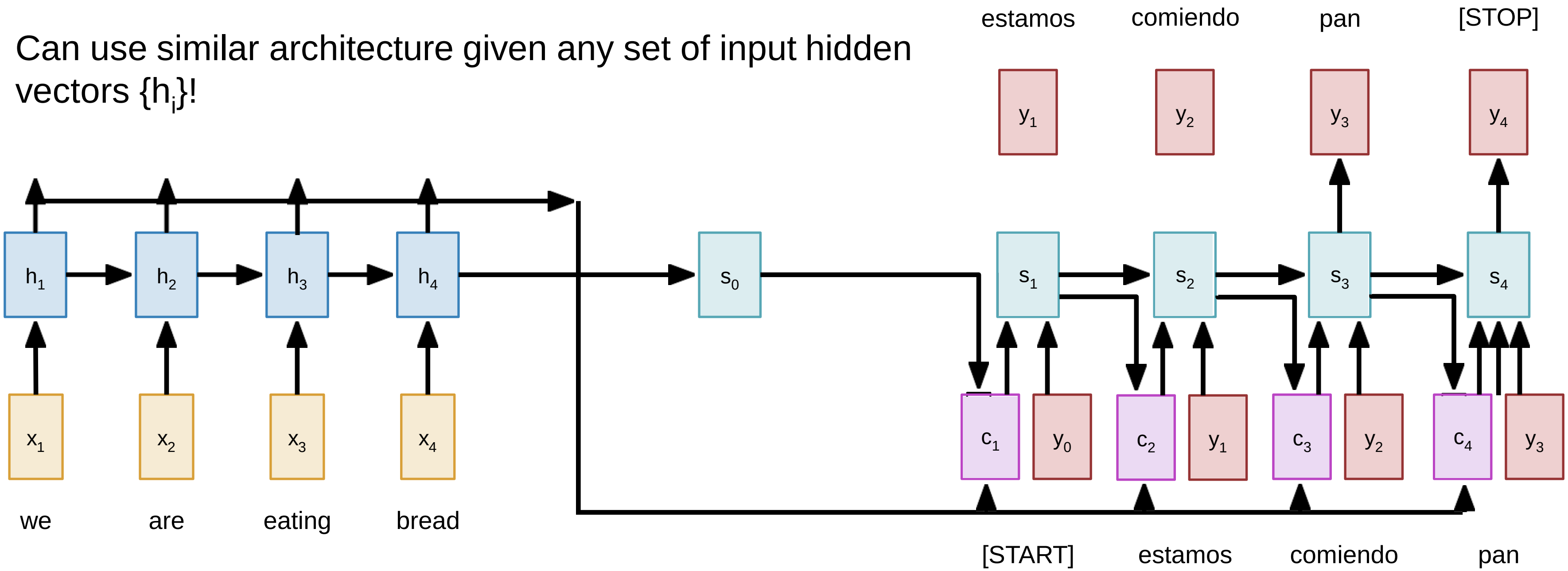


Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Sequence to Sequence with RNNs and Attention

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

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

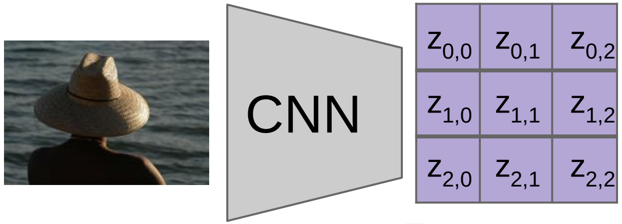


Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Image Captioning using spatial features

Input: Image **I**

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



Extract spatial features from a pretrained CNN

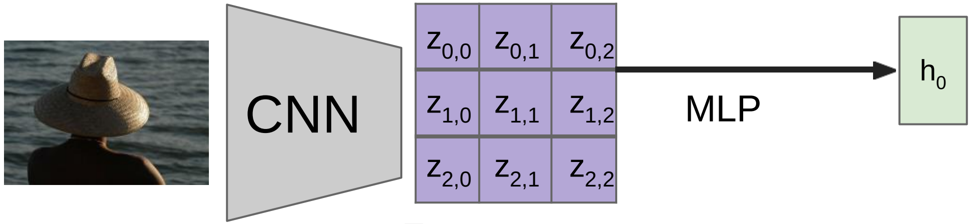
Features: H x W x D

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

Image Captioning using spatial features

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

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



Extract spatial features from a pretrained CNN

Features: $H \times W \times D$

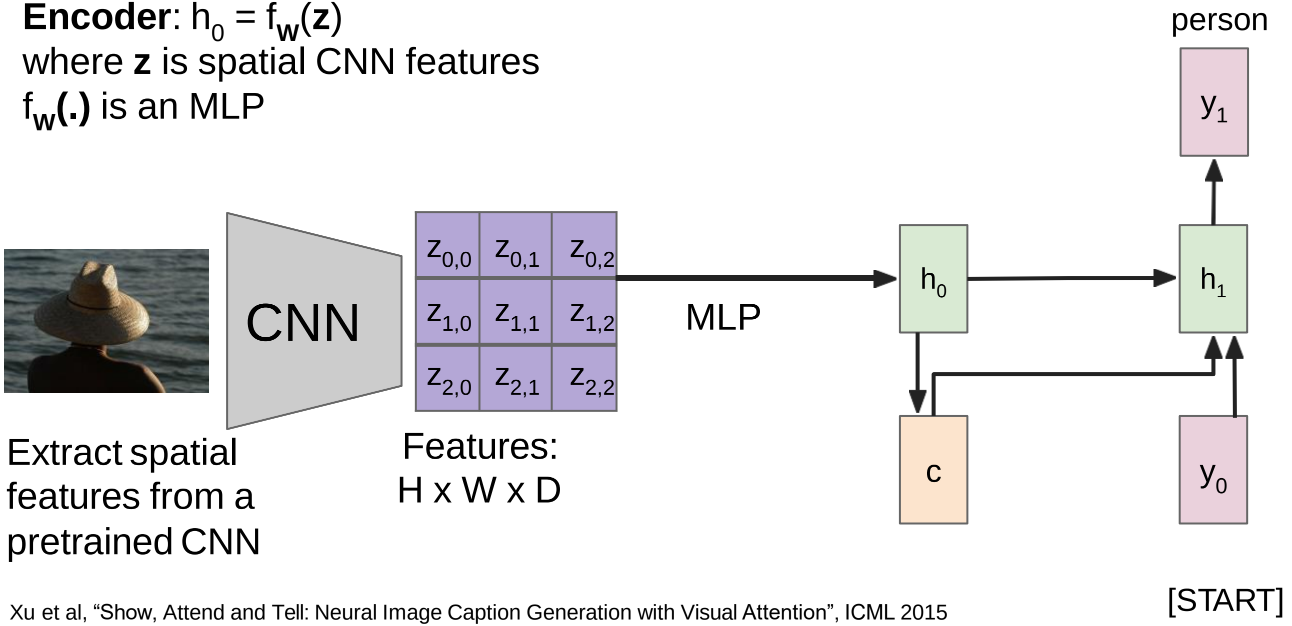
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning using spatial features

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

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

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



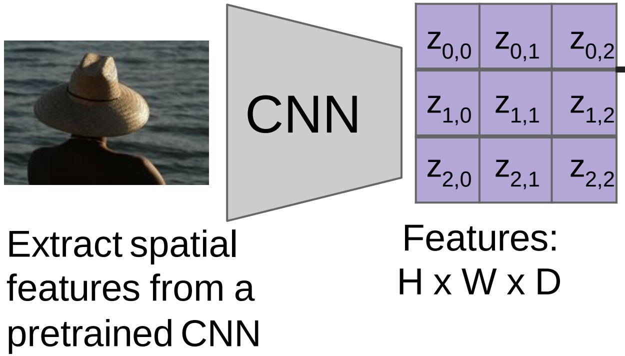
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

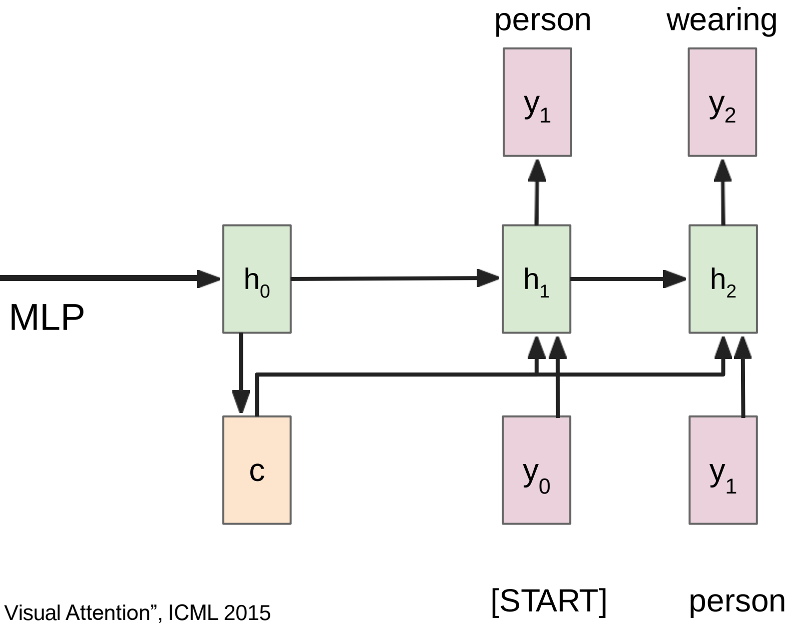
Image Captioning using spatial features

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

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

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





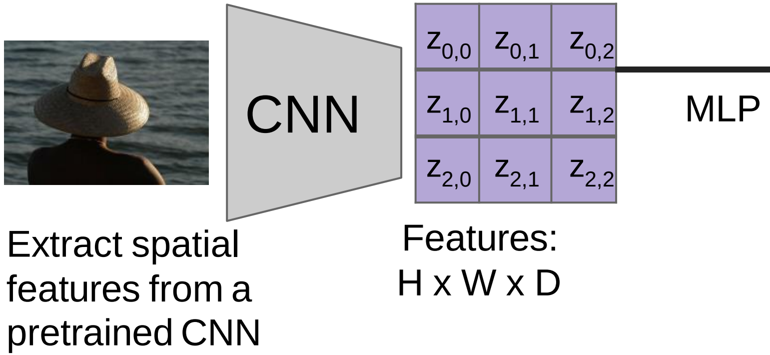
Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

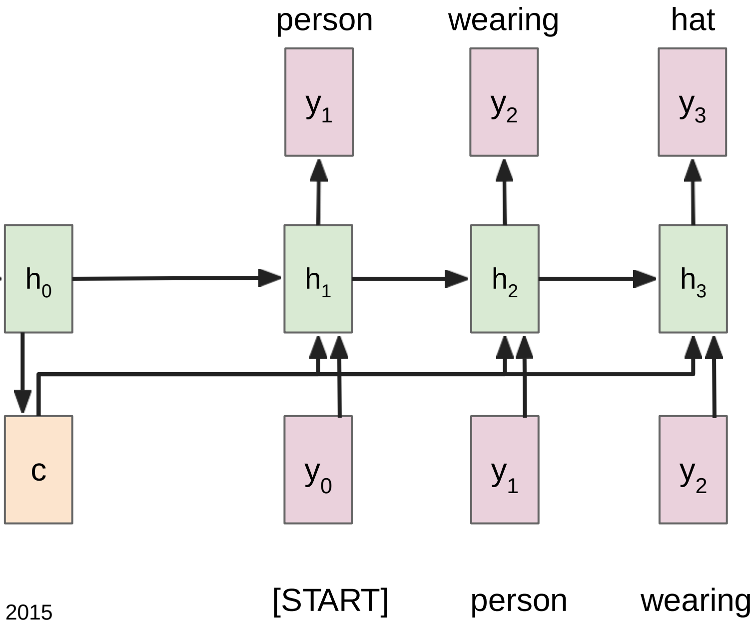
Image Captioning using spatial features

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

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

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





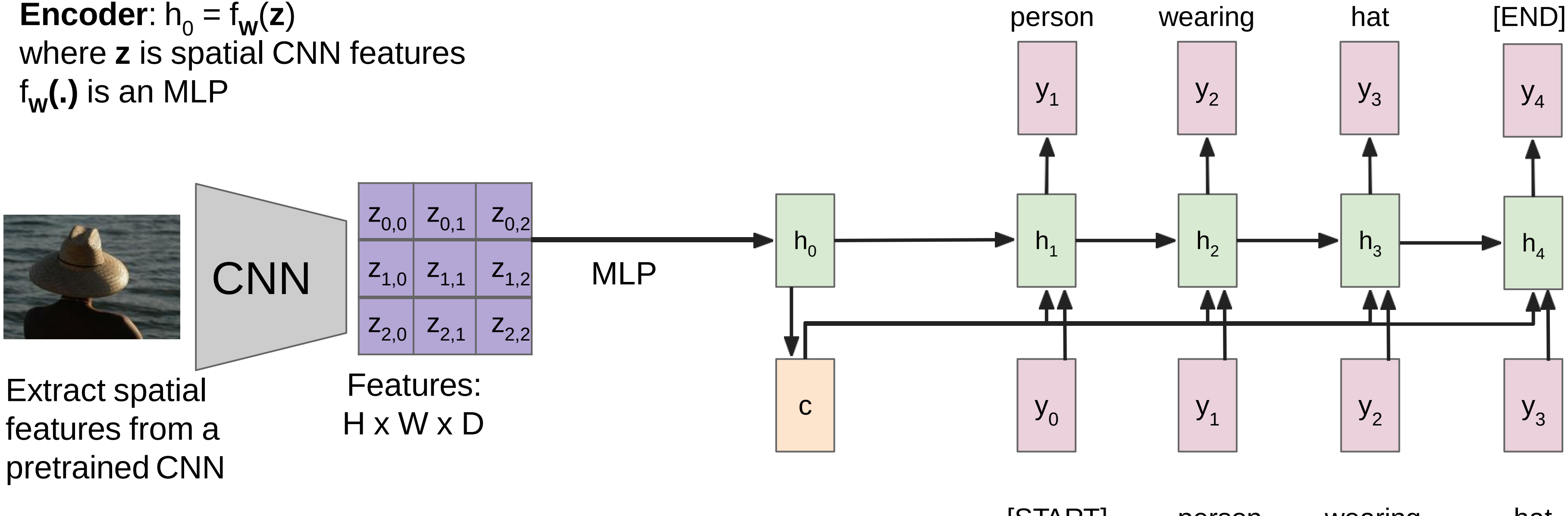
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning using spatial features

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

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

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



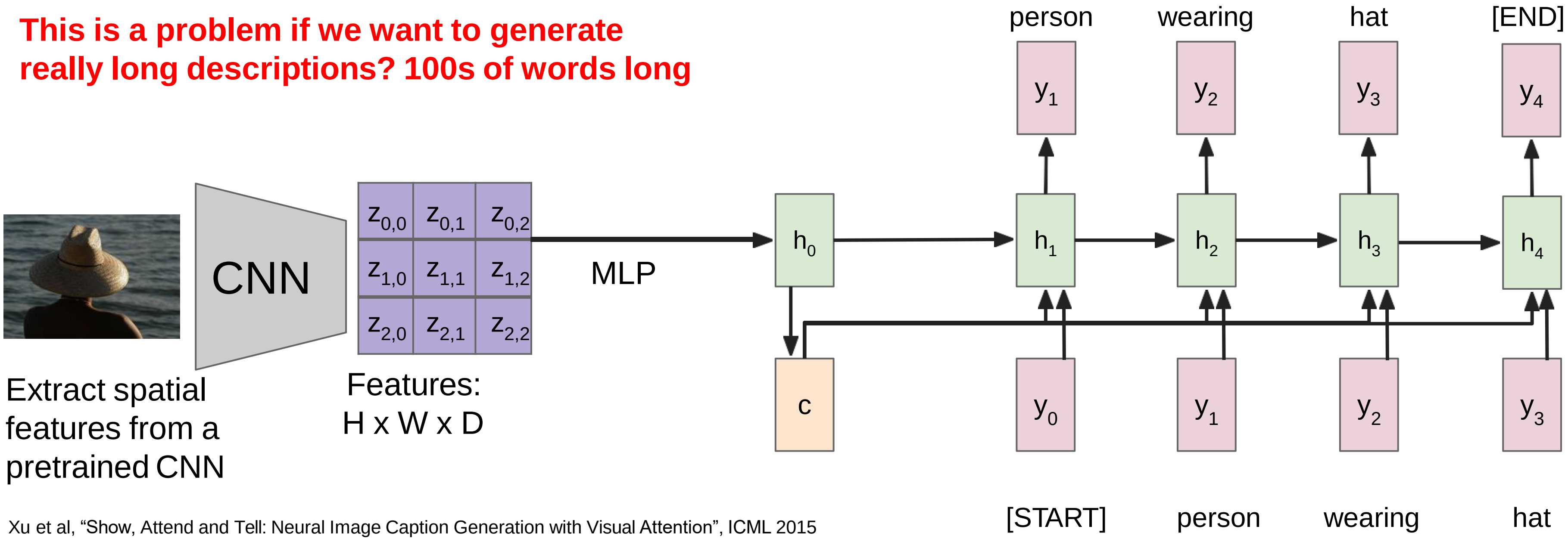
Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Image Captioning using spatial features

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



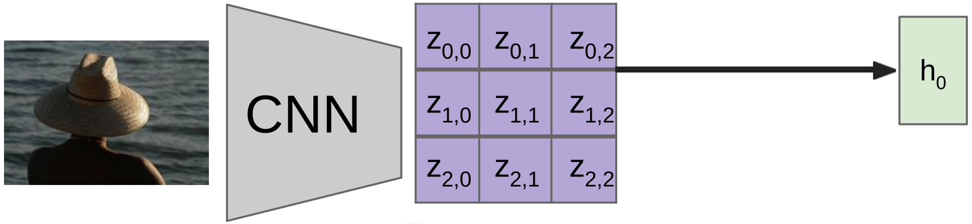
Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Image Captioning with RNNs and Attention

gif source

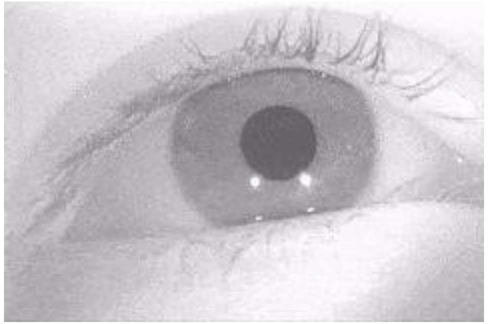
Attention idea: New context vector at every time step.

Each context vector will attend to different image regions



Extract spatial features from a pretrained CNN

Features: H x W x D



Attention Saccades in humans

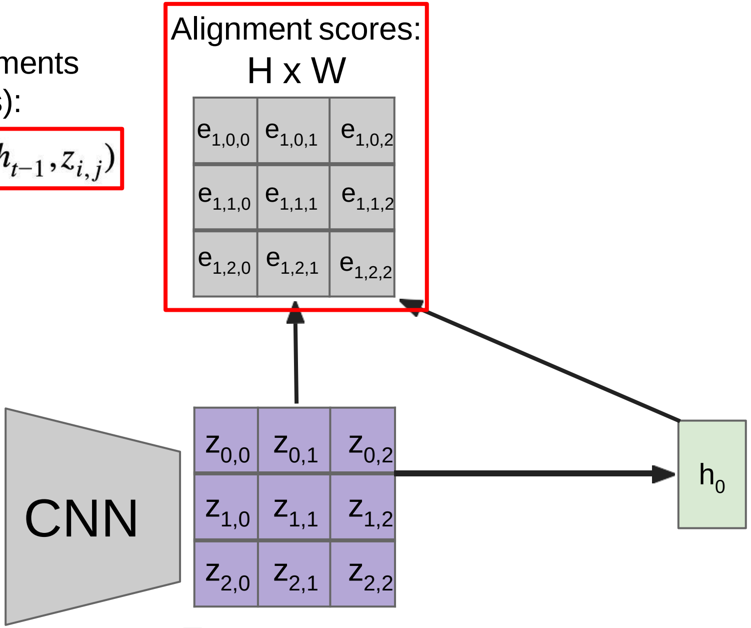
Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Image Captioning with RNNs and Attention

Compute alignments scores (scalars):

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

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



Extract spatial features from a pretrained CNN

Features:
H x W x D

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

Image Captioning with RNNs and Attention

Compute alignments scores (scalars):

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

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

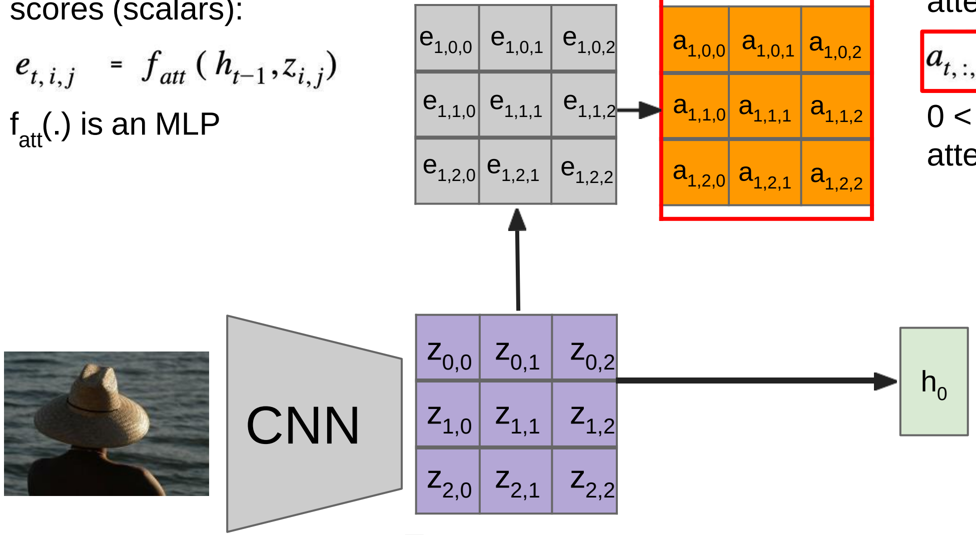
Alignment scores: H x W

Attention: H x W

Normalize to get attention weights:

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

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



Extract spatial features from a pretrained CNN

Features: H x W x D

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

Image Captioning with RNNs and Attention

Compute alignments scores (scalars):

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

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

Alignment scores:
H x W

Attention:
H x W

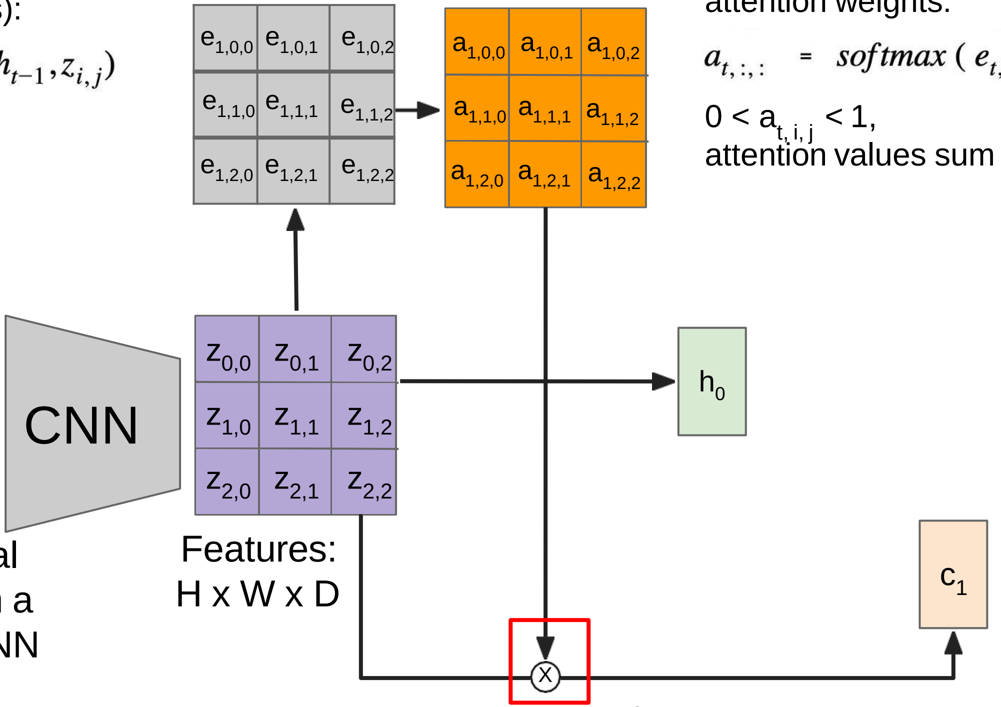
Normalize to get attention weights:

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

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

Compute context vector:

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



Extract spatial features from a pretrained CNN

Features:
H x W x D

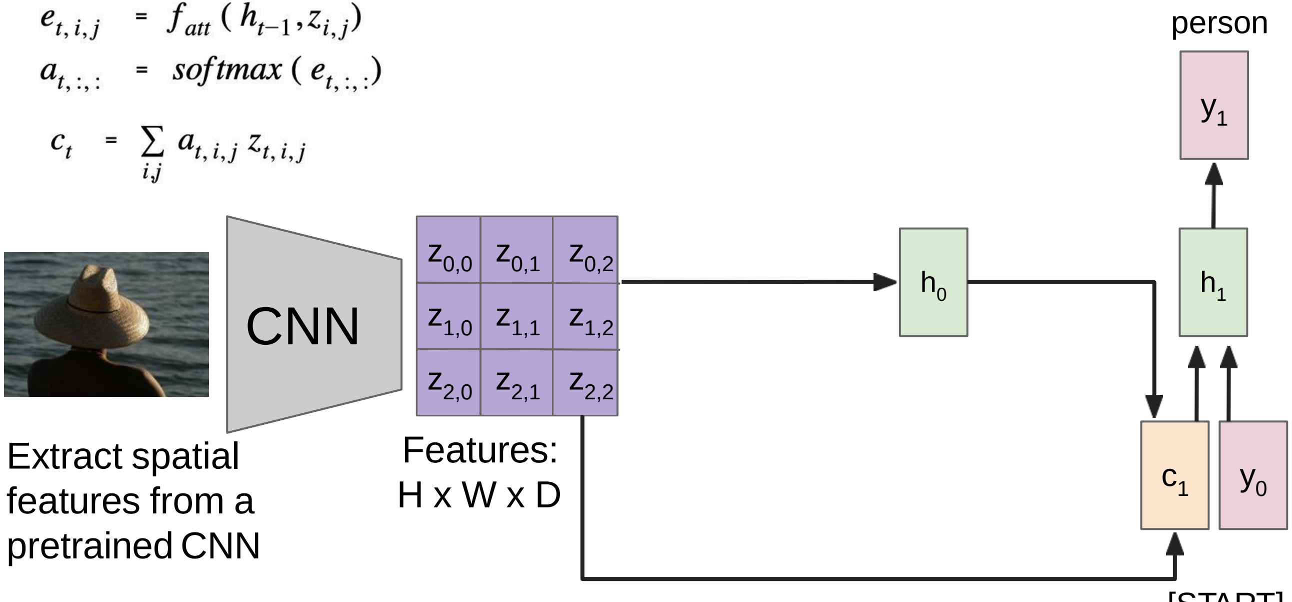
Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Image Captioning with RNNs 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

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

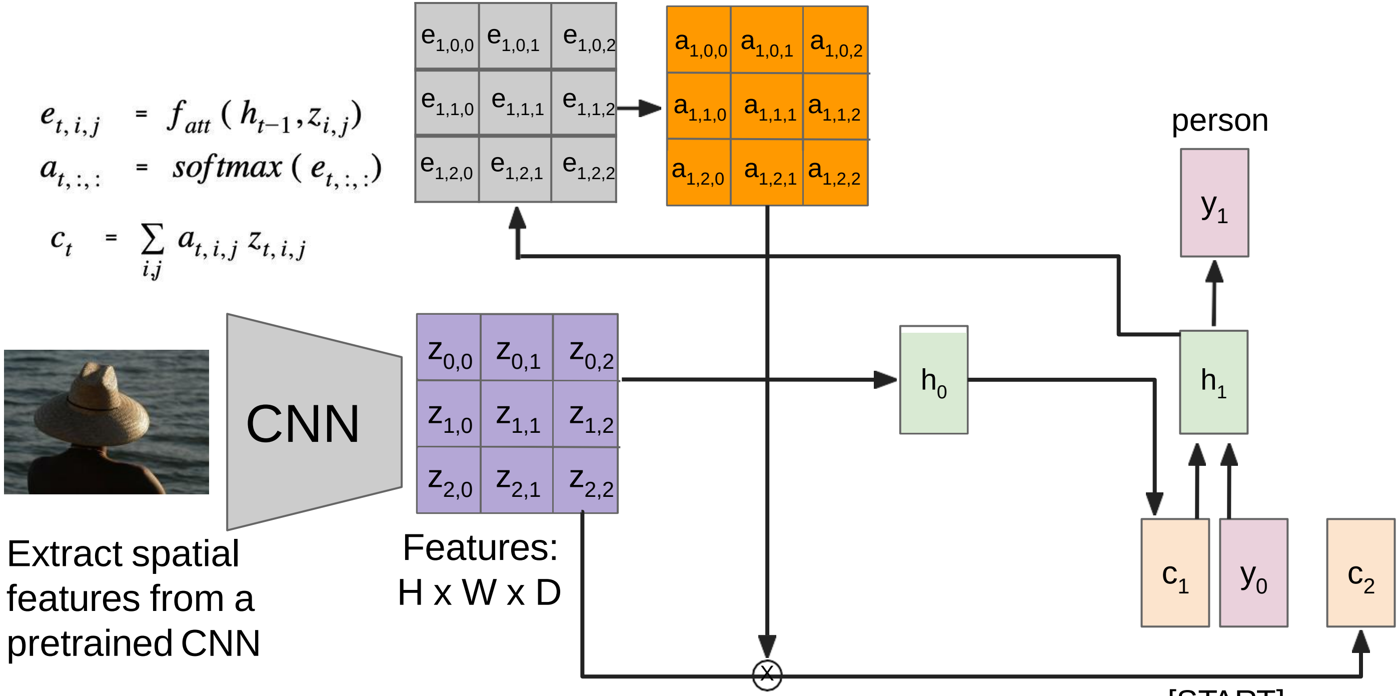


Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with RNNs and Attention

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

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



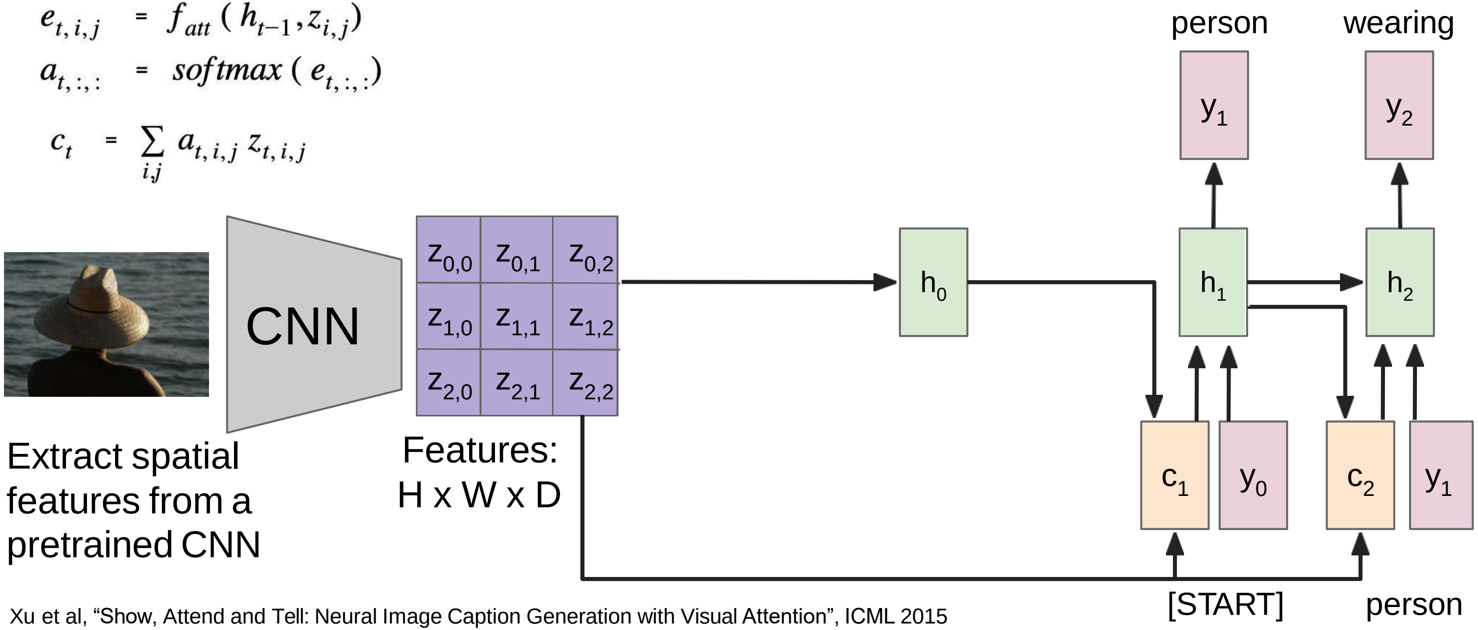
Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Image Captioning with RNNs 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

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



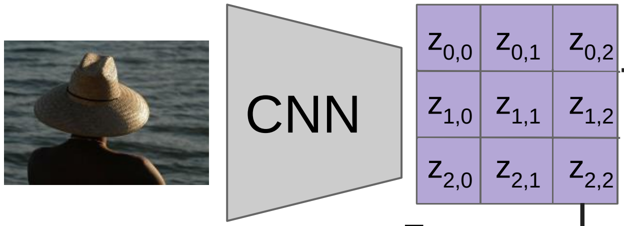
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with RNNs and Attention

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

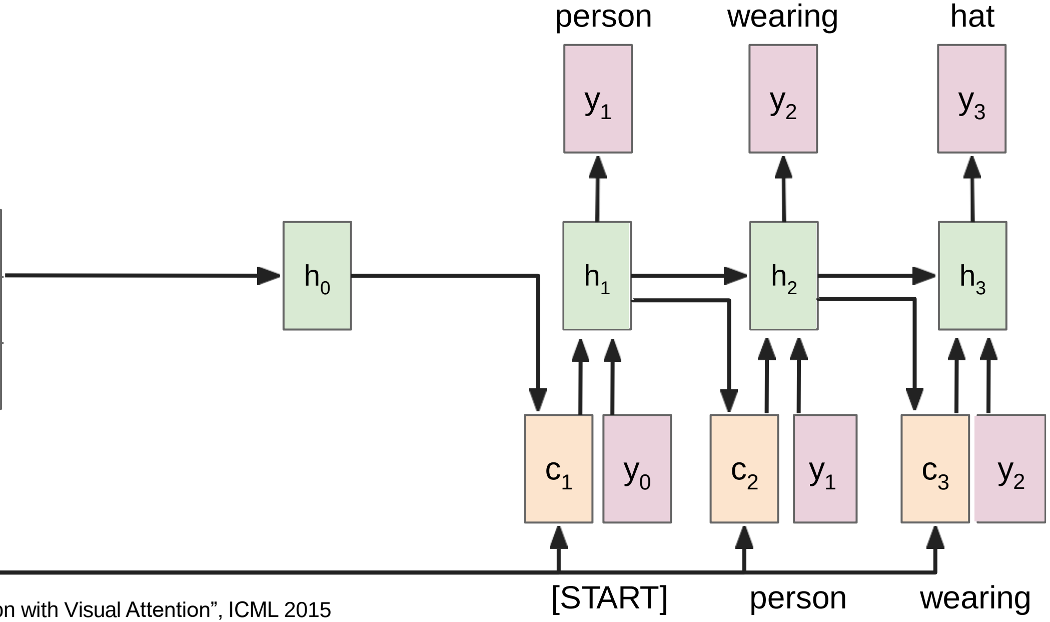
Decoder: $y_t = g_V(y_{t-1}, h_{t-1}, c_t)$
New context vector at every time step

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



Extract spatial features from a pretrained CNN

Features: H x W x D



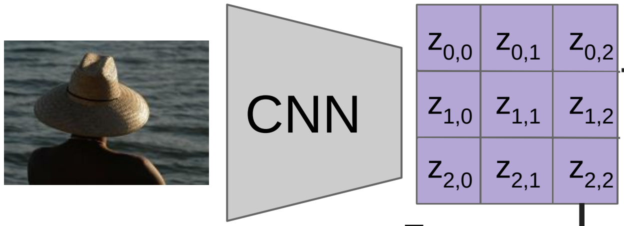
Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

Image Captioning with RNNs and Attention

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

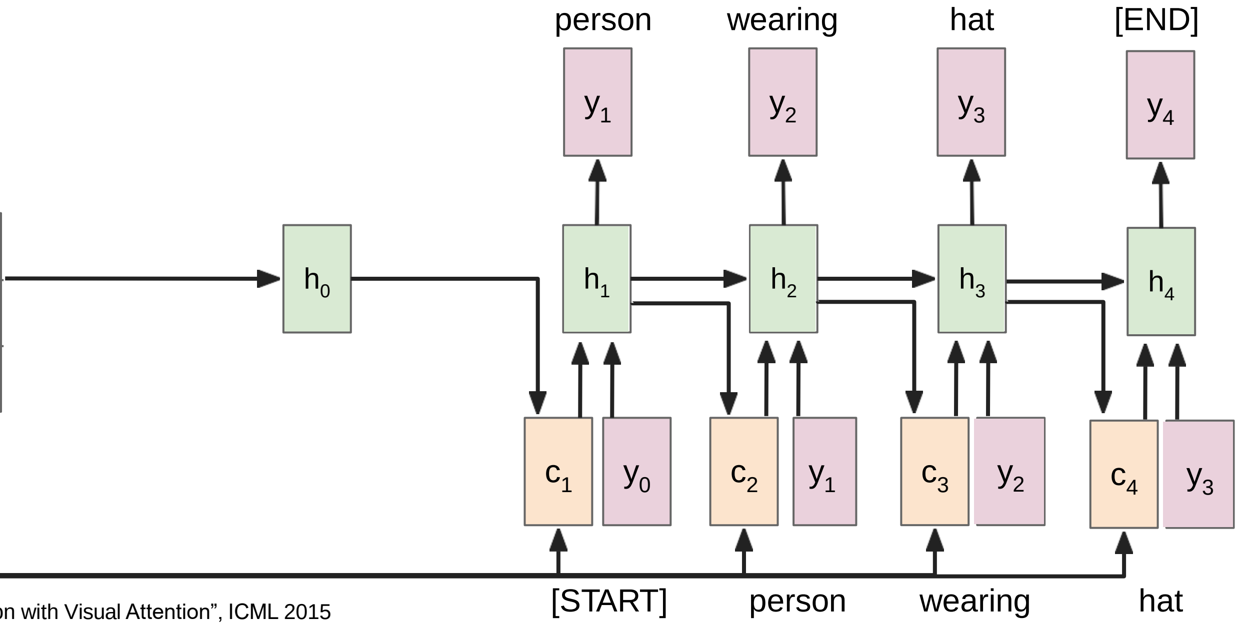
Each timestep of decoder uses a different context vector that looks at different parts of the input image

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



Extract spatial features from a pretrained CNN

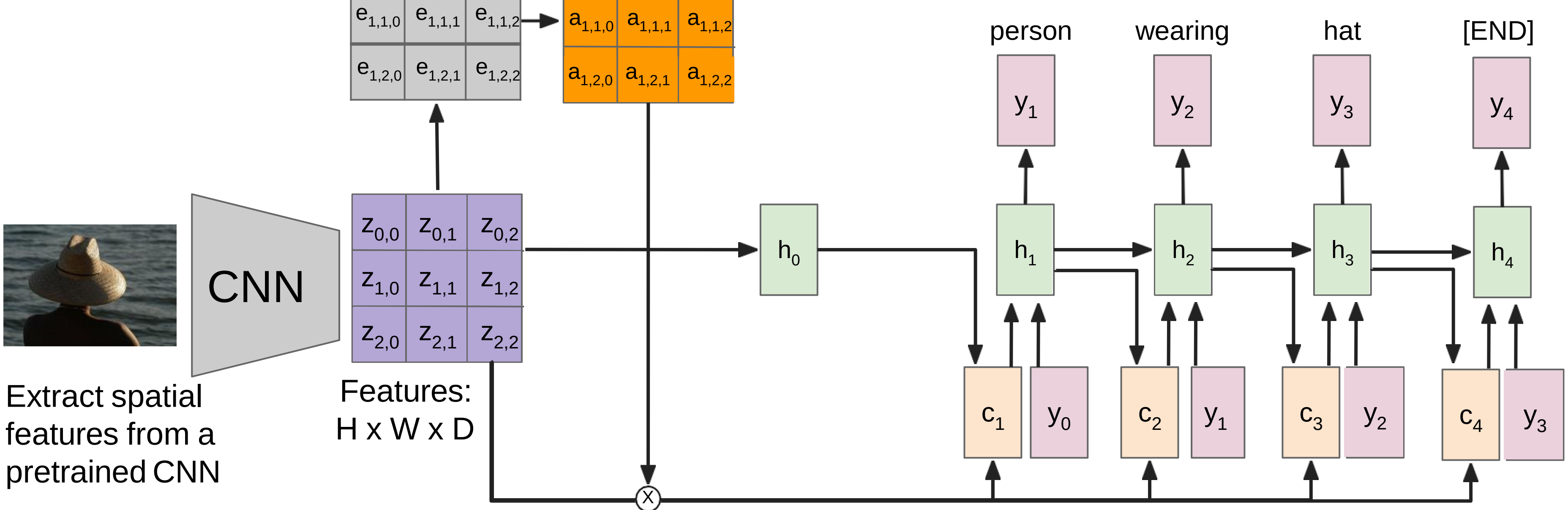
Features: H x W x D



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

Image Captioning with RNNs and Attention

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

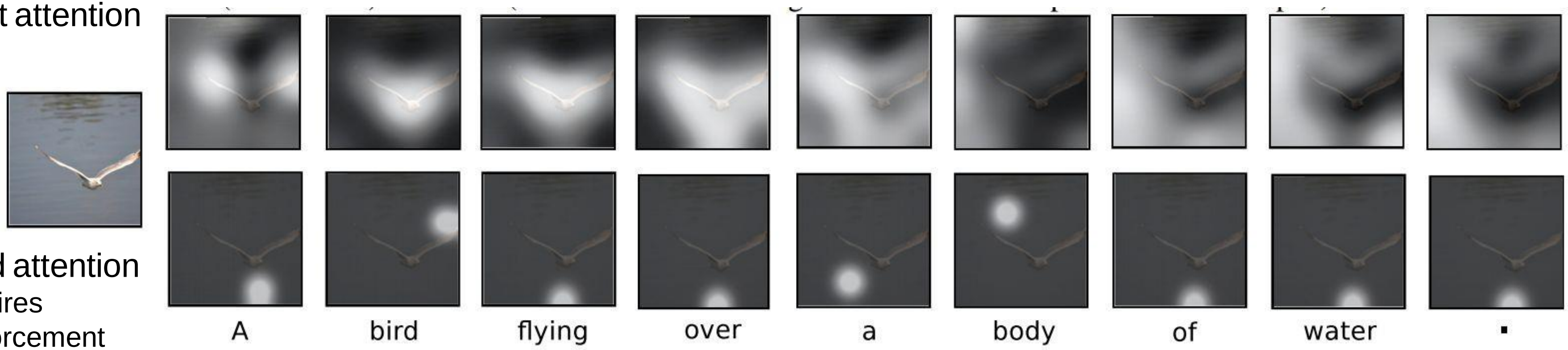


Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with Attention

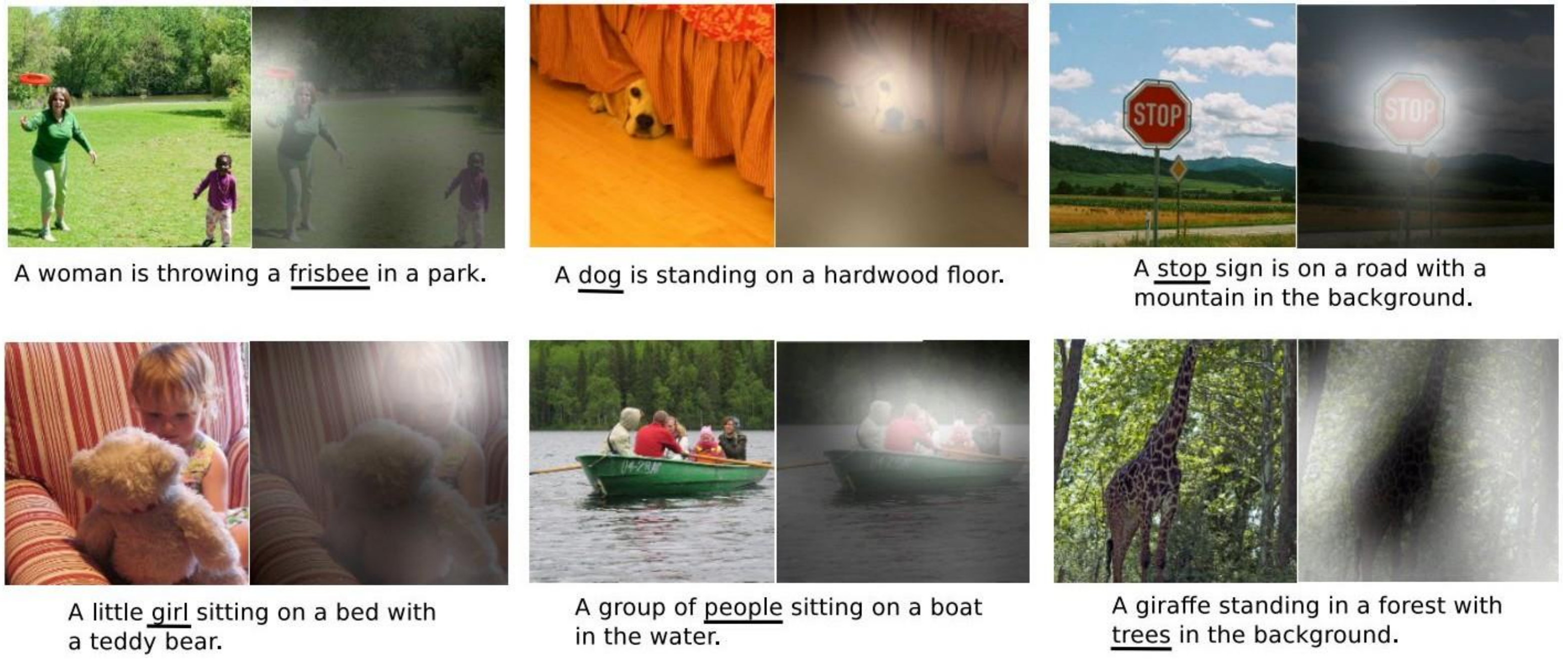
Soft attention

Hard attention
(requires
reinforcement
learning)



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

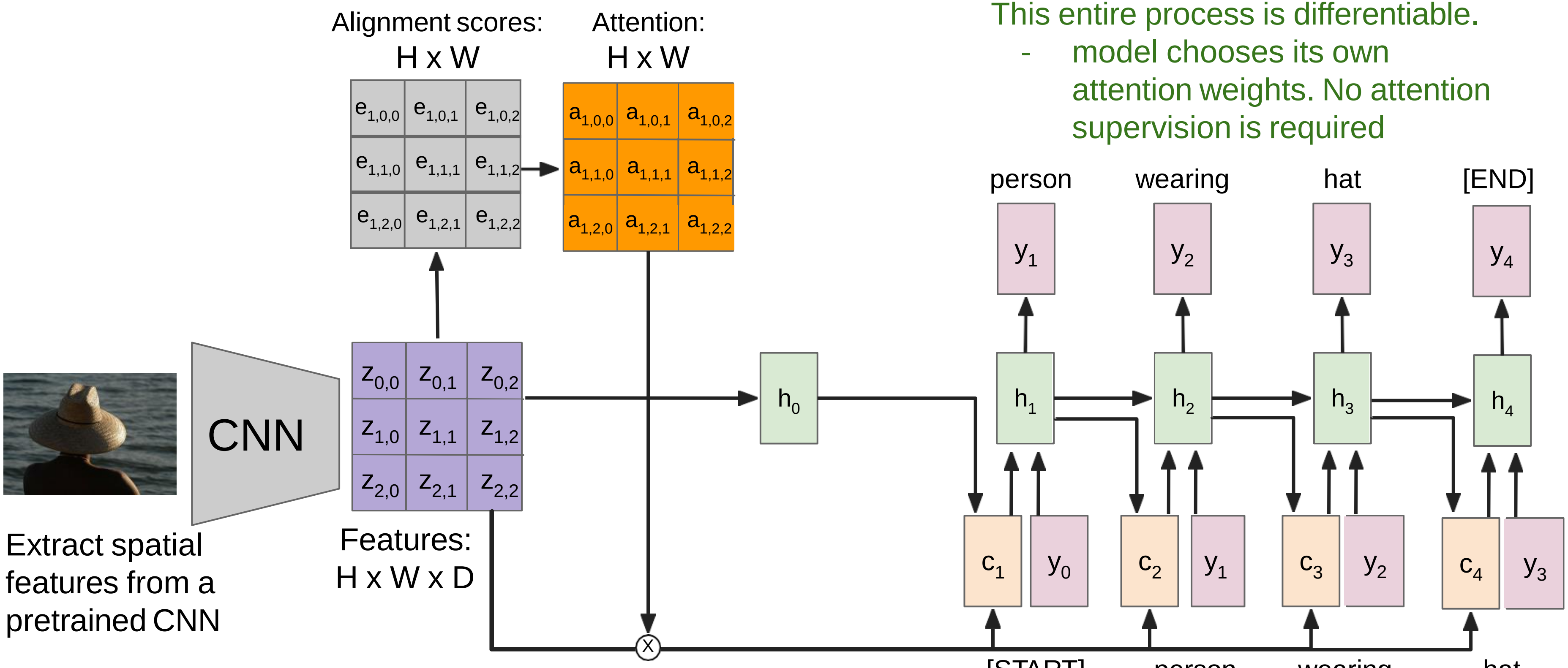
Image Captioning with Attention



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

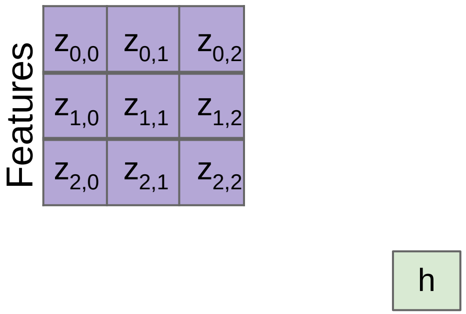
Image Captioning with RNNs and Attention

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



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

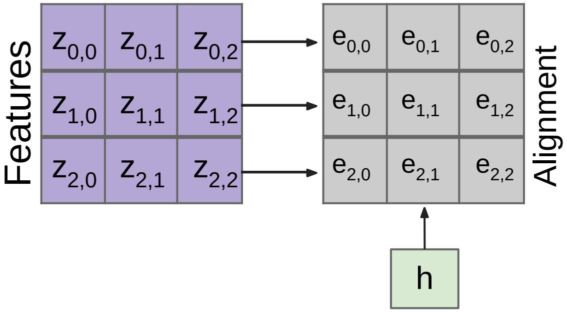
Attention we just saw in image captioning



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

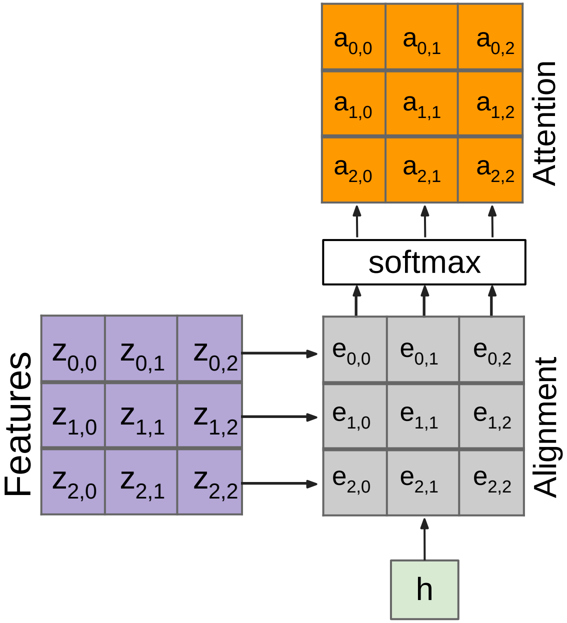
Attention we just saw in image captioning

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



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

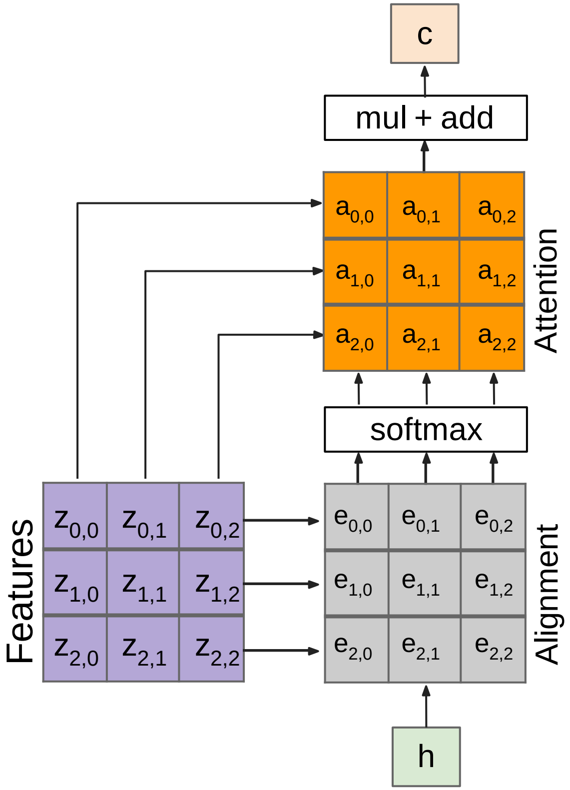
Attention we just saw in image captioning



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

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

Attention we just saw in image captioning

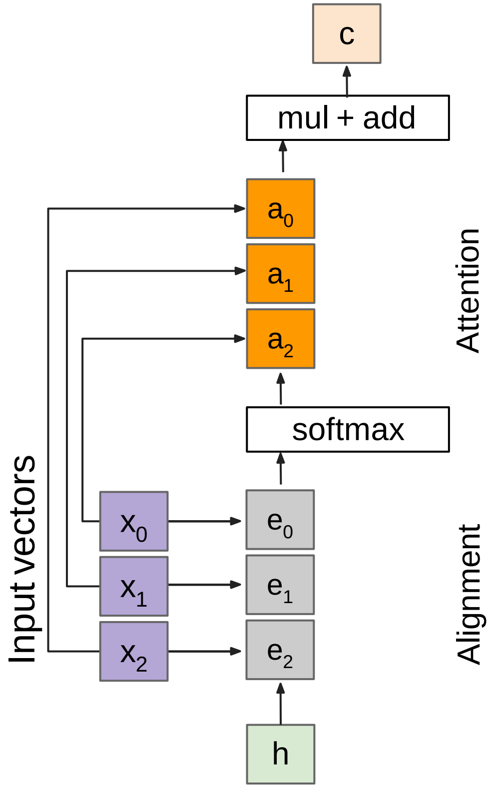


Outputs:
context vector: **c** (shape: D)

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

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

General attention layer



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

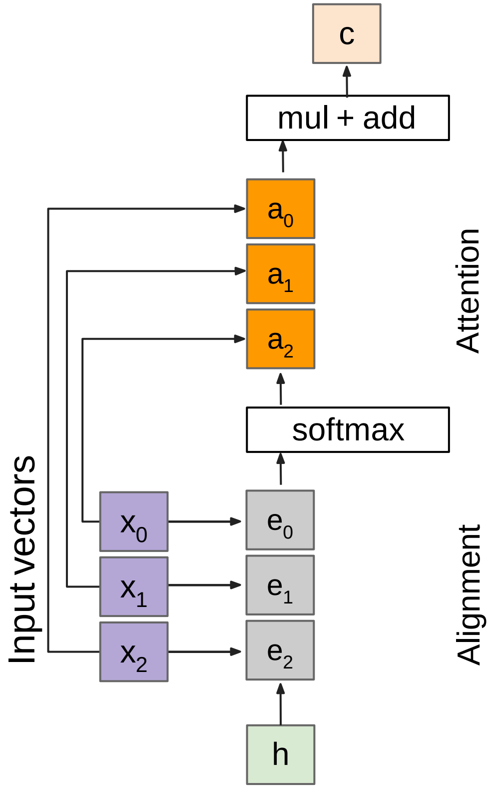
Operations:
Alignment: $e_i = f_{att}(h, x_i)$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $\mathbf{c} = \sum_i a_i x_i$

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: **c** (shape: D)

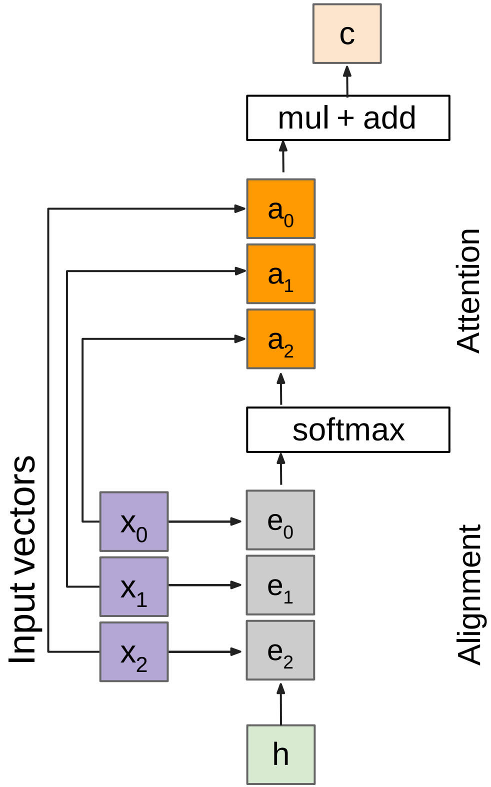
Operations:
$\boxed{\text{Alignment: } e_i = h \cdot x_i}$
Attention: **a** = softmax(**e**)
Output: **c** = $\sum_i a_i x_i$

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

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

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

General attention layer



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

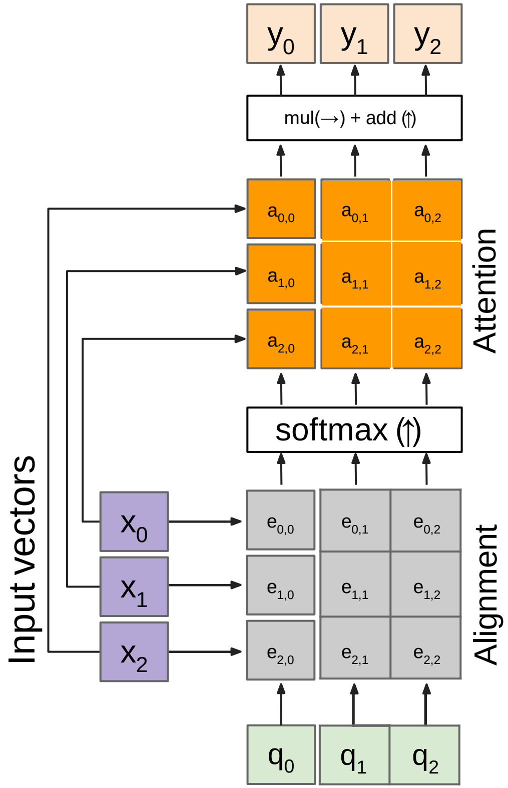
Operations:
$\boxed{\text{Alignment: } e_i = h \cdot x_i / \sqrt{D}}$
Attention: **a** = softmax(**e**)
Output: **c** = $\sum_i a_i x_i$

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

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

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

General attention layer



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

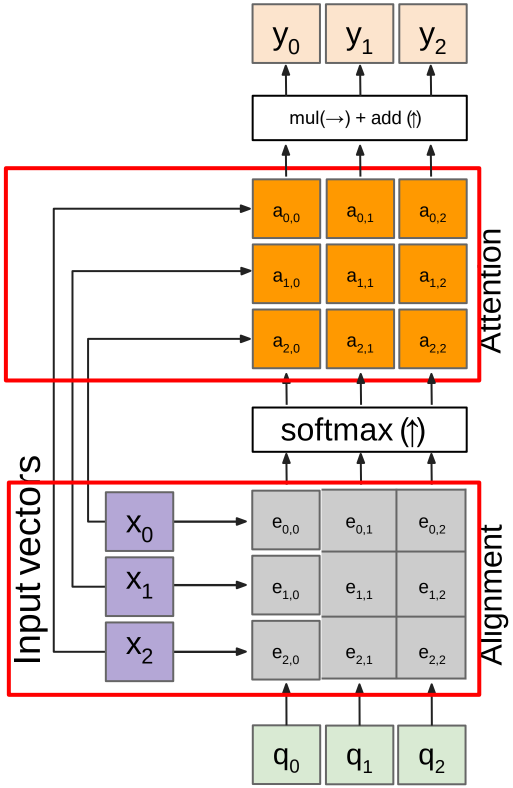
Multiple query vectors
- each query creates a new output context vector

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

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)

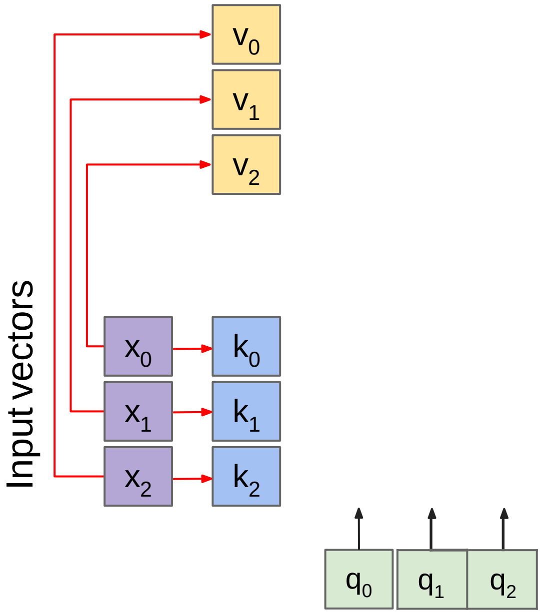
Operations:
Alignment: $e_{i,j} = q_j \cdot x_i / \sqrt{D}$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $y_j = \sum_i a_{i,j} x_i$

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

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

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

General attention layer



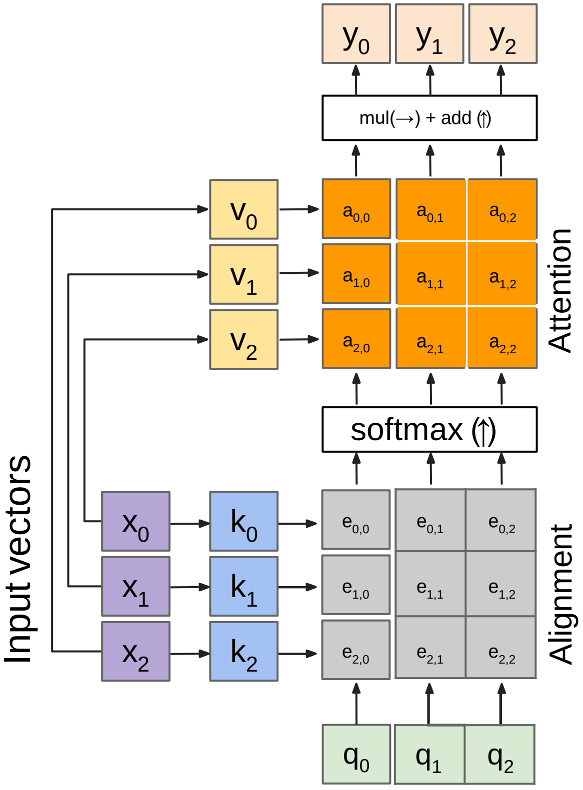
Operations:
Key vectors: $\mathbf{k} = \mathbf{x}\mathbf{W}_k$
Value vectors: $\mathbf{v} = \mathbf{x}\mathbf{W}_v$

Inputs:
Input vectors: $\mathbf{x}$ (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: $\boxed{D_v}$)

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

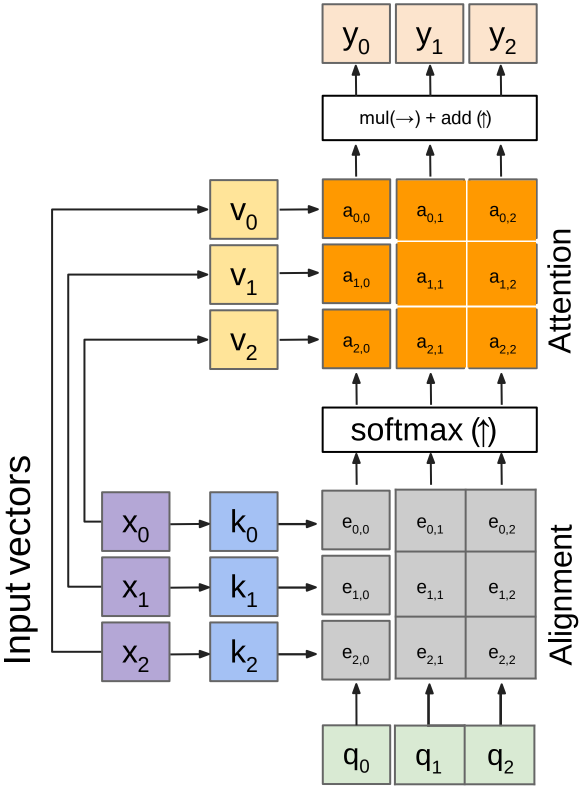
Inputs:
Input vectors: **x** (shape: N x D)
Queries: **q** (shape: M x $\boxed{D_k}$)

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



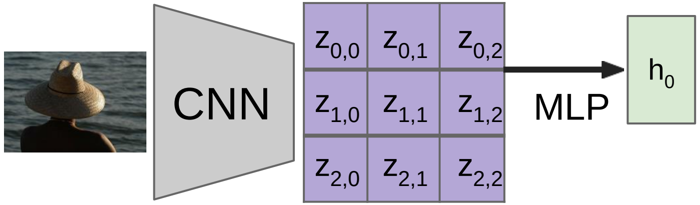
Outputs:
context vectors: **y** (shape: D_v)

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

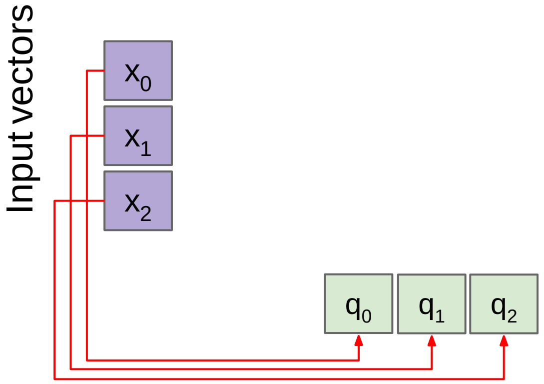
Inputs:
Input vectors: **x** (shape: N x D)
Queries: **q** (shape: M x D_k)

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



Self attention layer



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

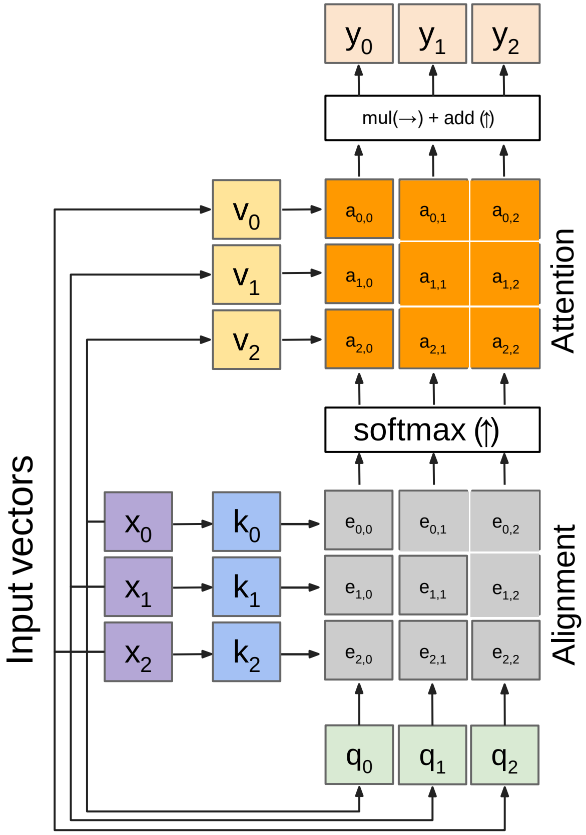
Inputs:
Input vectors: $\mathbf{x}$ (shape: N x D)
~~Queries: **q** (shape: M x D_k)~~

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

Self attention layer

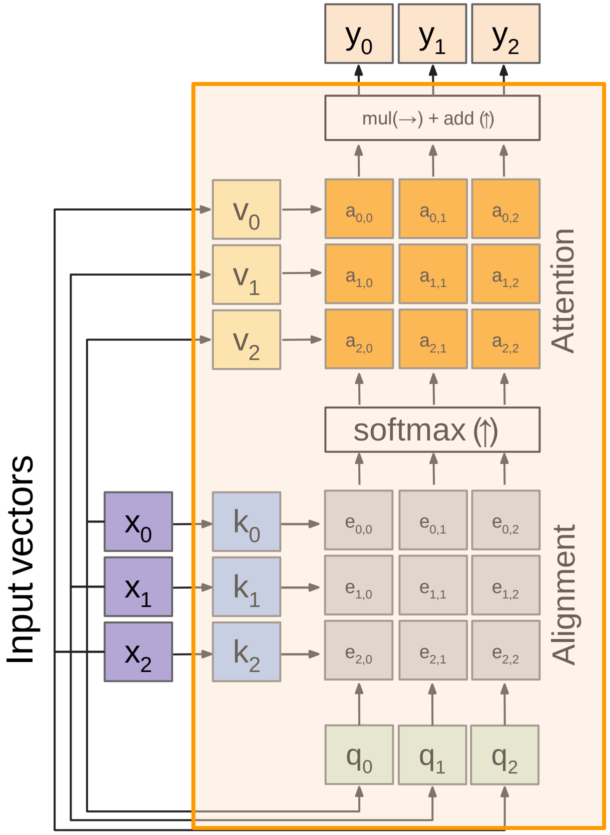


Outputs:
context vectors: **y** (shape: D_v)

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

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

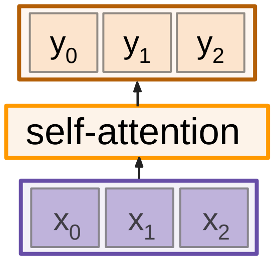
Self attention layer - attends over sets of inputs



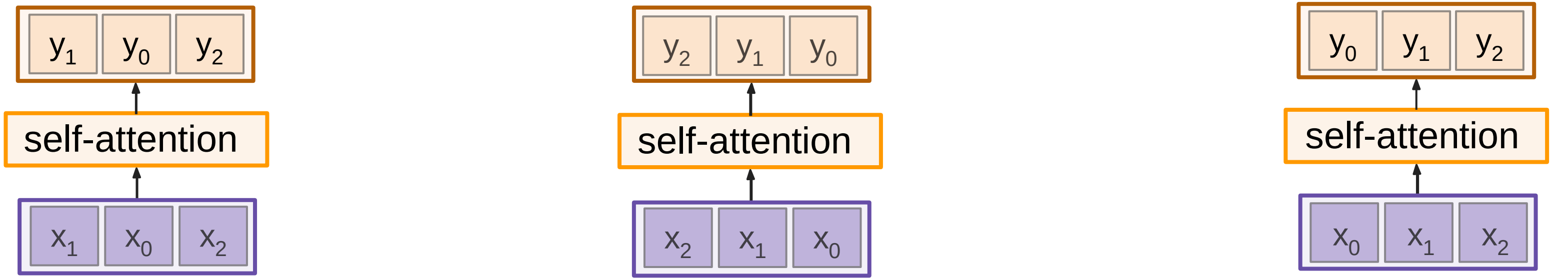
Outputs:
context vectors: **y** (shape: D_v)

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

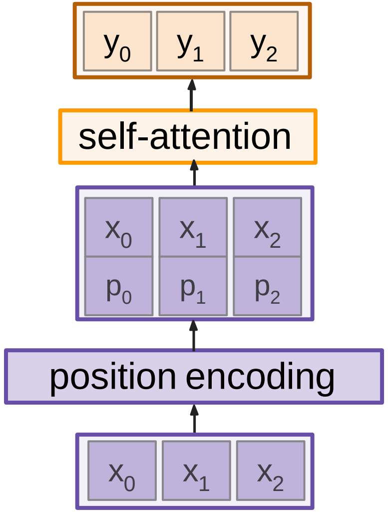


Permutation equivariant

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

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

Positional encoding



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

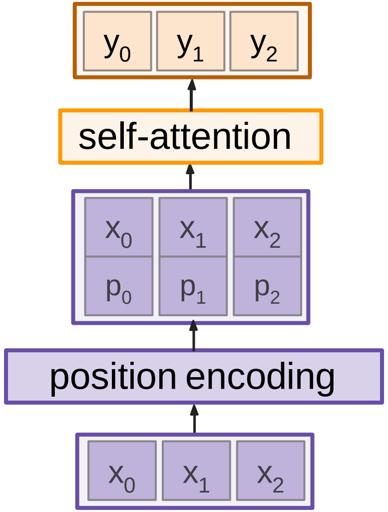
We use a function *pos*: $N \rightarrow R^d$ to process the position j of the vector into a d-dimensional vector

So, $p_j = pos(j)$

Desiderata of *pos*(.) :

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

Positional encoding



Concatenate special positional encoding p_j to each input vector x_j

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

So, $p_j = pos(j)$

Options for *pos*(.)

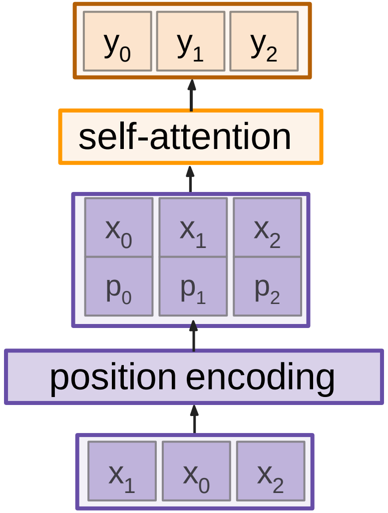
1. Learn a lookup table:
   - Learn parameters to use for *pos*(t) for t ε[0, T)
   - Lookup table contains T x d parameters.

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**.

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

Positional encoding



Concatenate special positional encoding p_j to each input vector x_j

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

So, p_j = *pos*(j)

Options for *pos*(.)

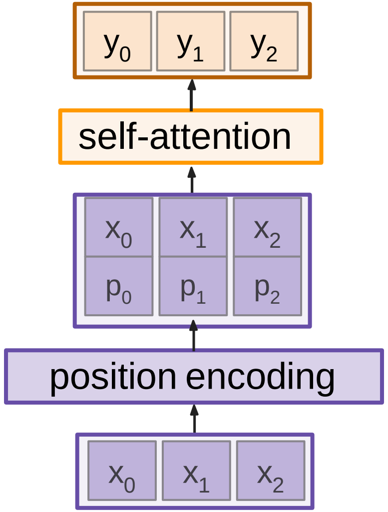
1. Learn a lookup table:
   - Learn parameters to use for *pos*(t) for $t \in [0, T)$
   - Lookup table contains T x d parameters.

2. Design a fixed function with the desiderata

$$\mathbf{p(t)} = \begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}_d \quad \text{where} \quad \omega_k = \frac{1}{10000^{2k/d}}$$

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

Positional encoding



Concatenate special positional encoding p_j to each input vector x_j

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

So, $p_j = pos(j)$

Options for *pos*(.)

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

2. Design a fixed function with the desiderata

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

Intuition:

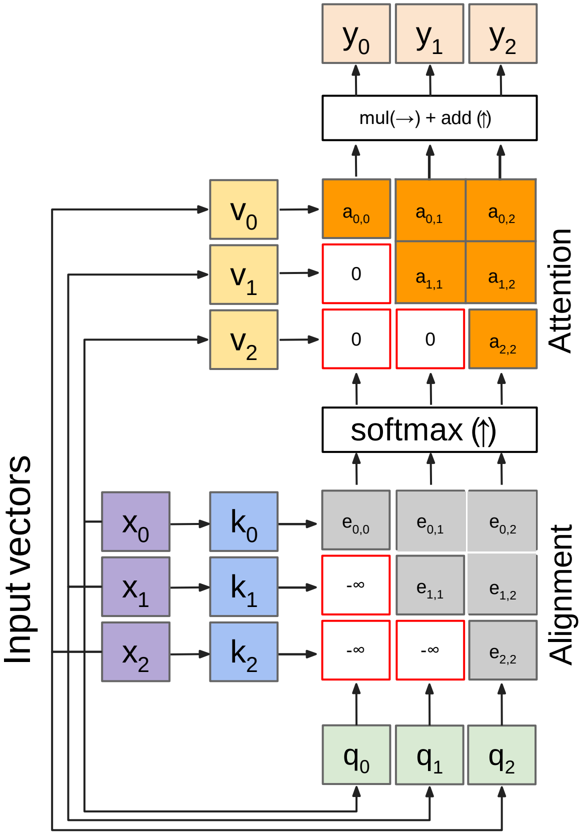
0 :	0	0	0	0	8 :	1	0	0	0
1 :	0	0	0	1	9 :	1	0	0	1
2 :	0	0	1	0	10 :	1	0	1	0
3 :	0	0	1	1	11 :	1	0	1	1
4 :	0	1	0	0	12 :	1	1	0	0
5 :	0	1	0	1	13 :	1	1	0	1
6 :	0	1	1	0	14 :	1	1	1	0
7 :	0	1	1	1	15 :	1	1	1	1

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

image source

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

Masked self-attention layer



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

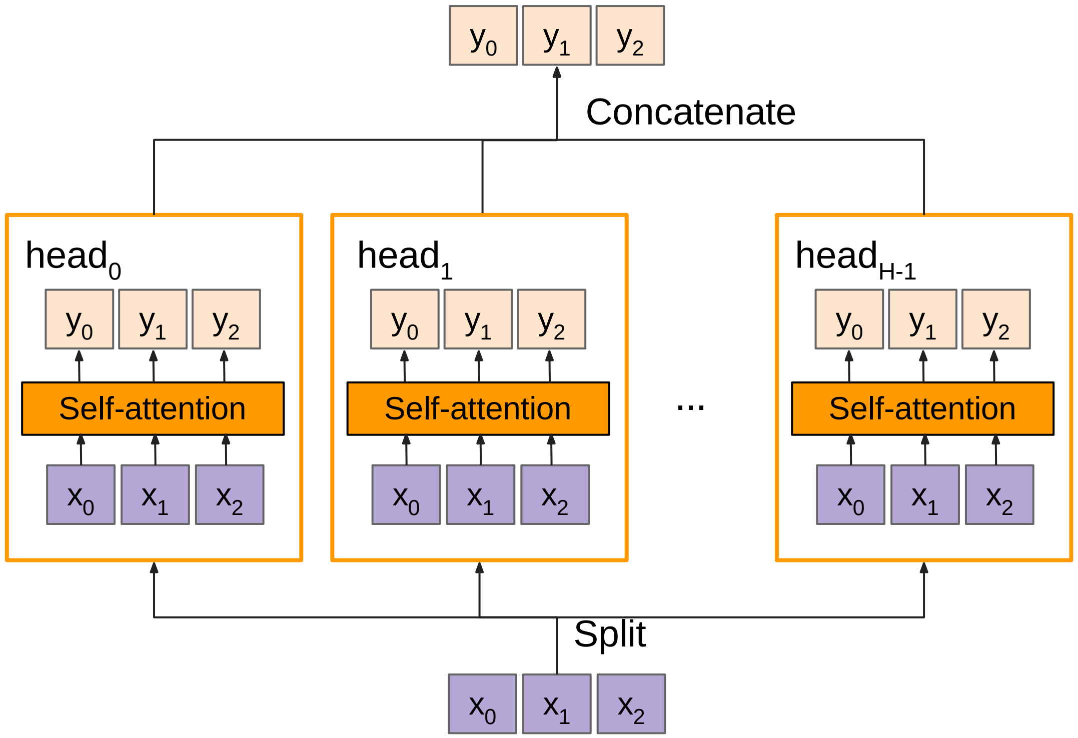
Operations:
Key vectors: $\mathbf{k} = \mathbf{x}\mathbf{W}_k$
Value vectors: $\mathbf{v} = \mathbf{x}\mathbf{W}_v$
Query vectors: $\mathbf{q} = \mathbf{x}\mathbf{W}_q$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $y_j = \sum_i a_{i,j} v_i$

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

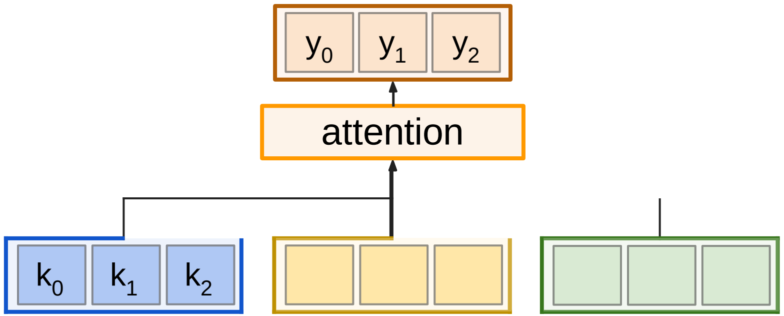
- Prevent vectors from looking at future vectors.
- Manually set alignment scores to -infinity

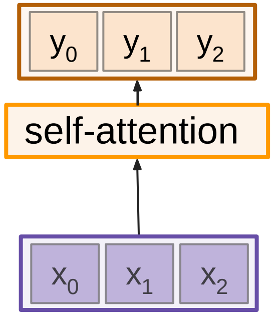
Multi-head self-attention layer

- Multiple self-attention heads in parallel



General attention versus self-attention

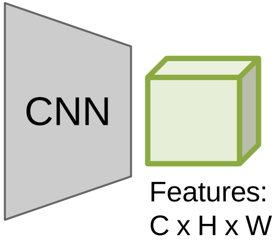




Example: CNN with Self-Attention

Input Image

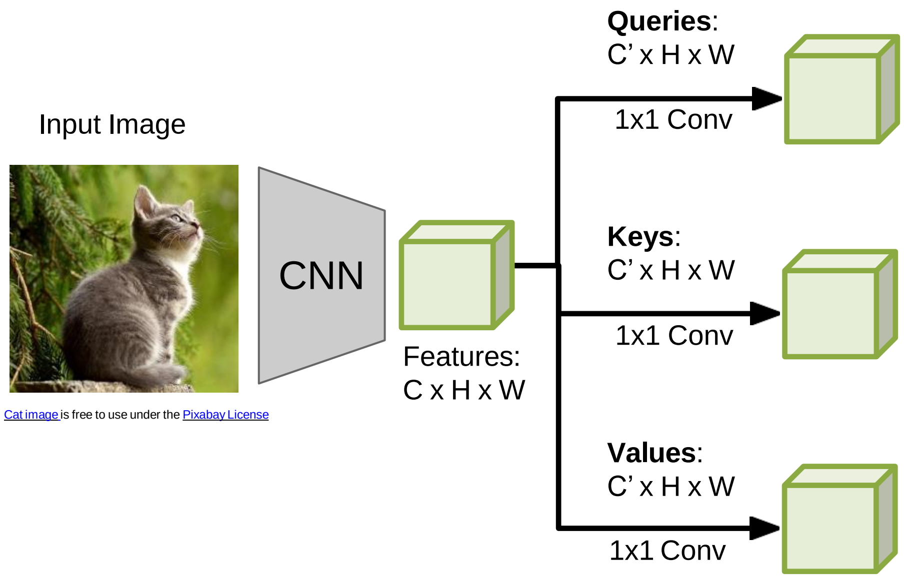
Cat image is free to use under the Pixabay License



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

Slide credit: Justin Johnson

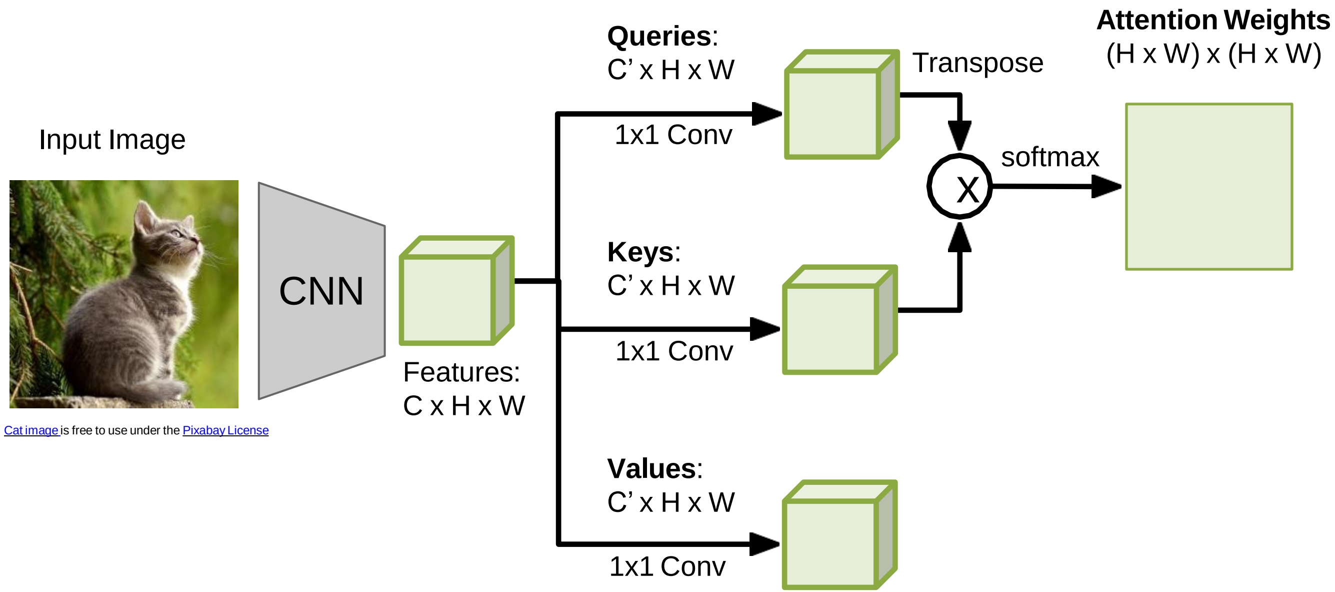
Example: CNN with Self-Attention



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

Slide credit: Justin Johnson

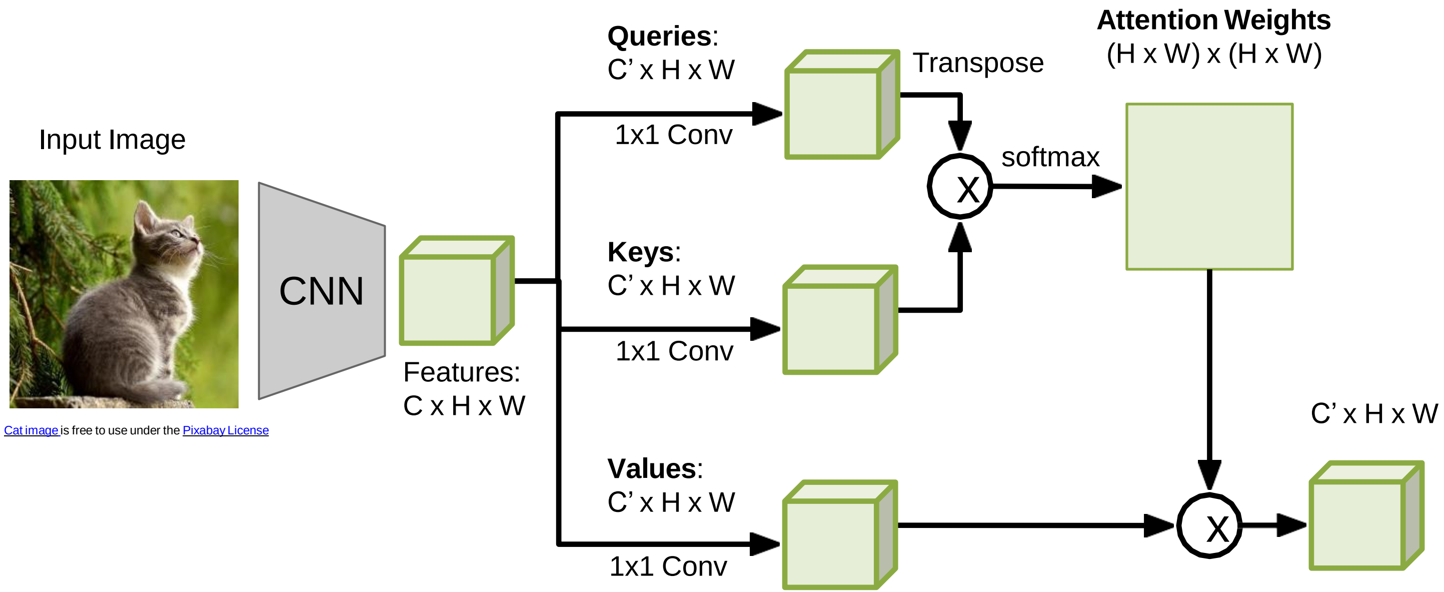
Example: CNN with Self-Attention



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

Slide credit: Justin Johnson

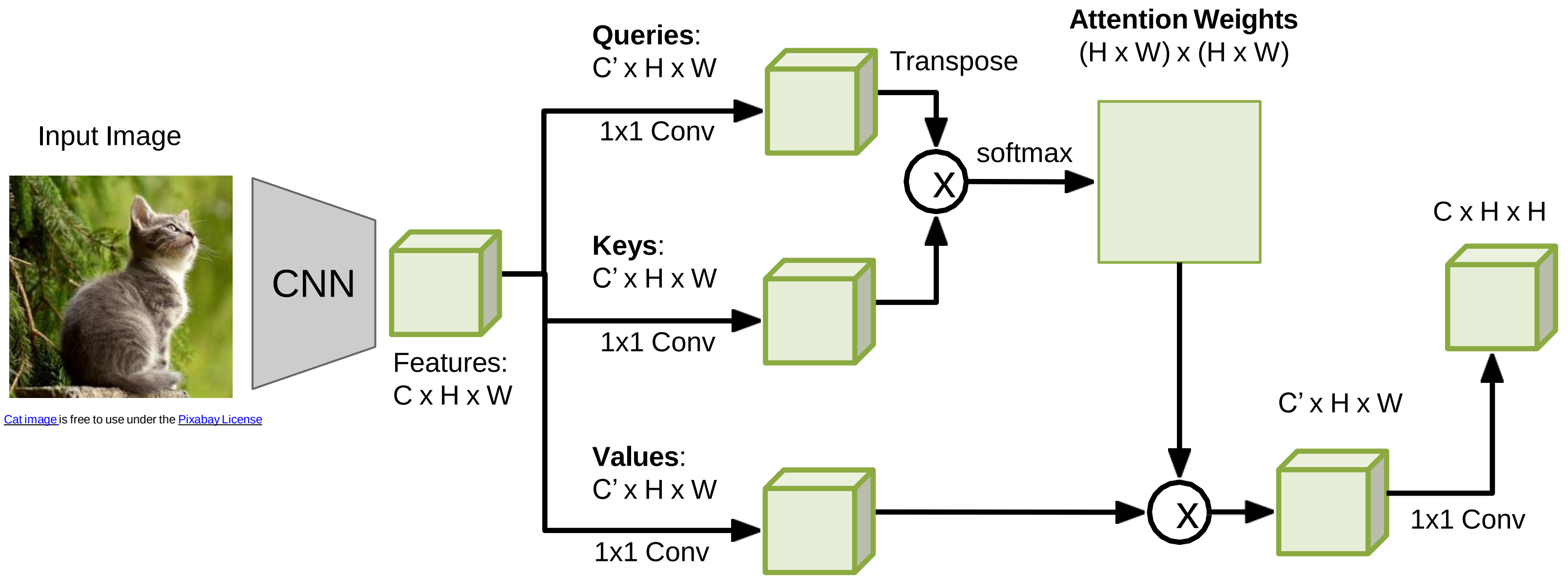
Example: CNN with Self-Attention



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

Slide credit: Justin Johnson

Example: CNN with Self-Attention



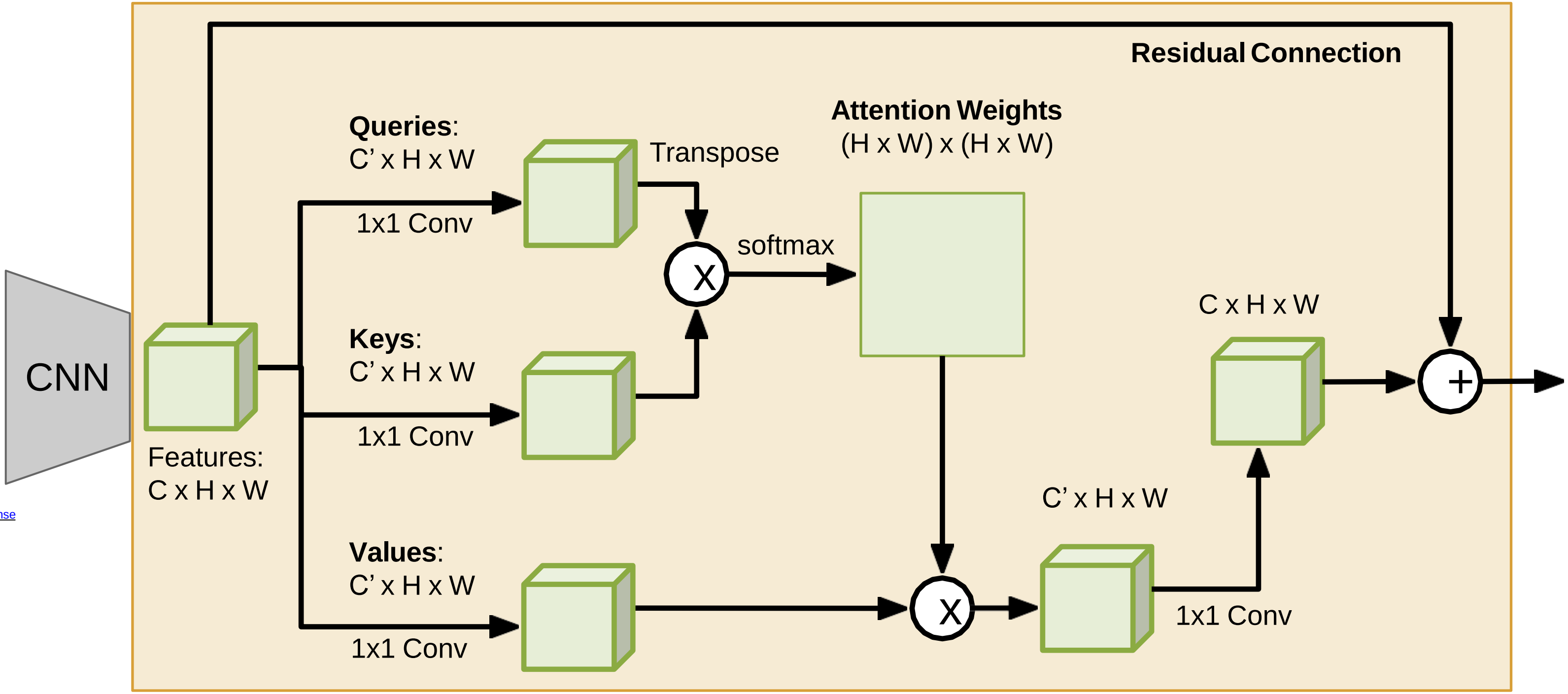
Zhang et al, “Self-Attention Generative Adversarial Networks”, ICML 2018

Slide credit: Justin Johnson

Example: CNN with Self-Attention

Input Image

Cat image is free to use under the Pixabay License



Self-Attention Module

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

Slide credit: Justin Johnson

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 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)

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 lukaszkaiser@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

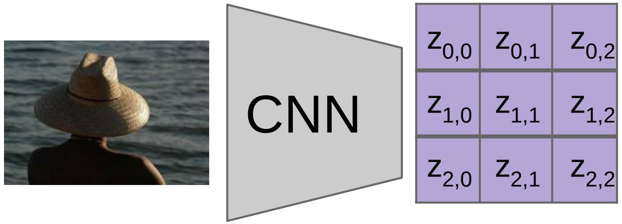
On the Opportunities and Risks of Foundation Models

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 Niebles 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 $\mathbf{y} = y_1, y_2, \ldots, y_T$



Extract spatial features from a pretrained CNN

Features:
H x W x D

Image Captioning using Transformers

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

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

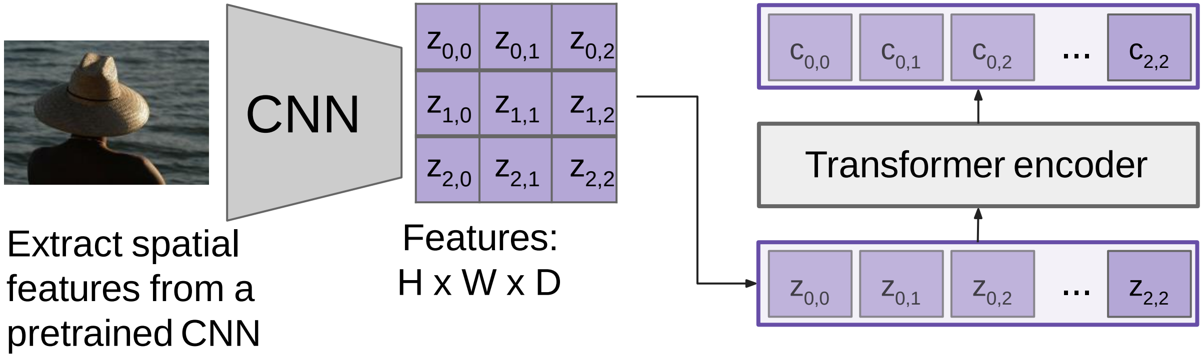
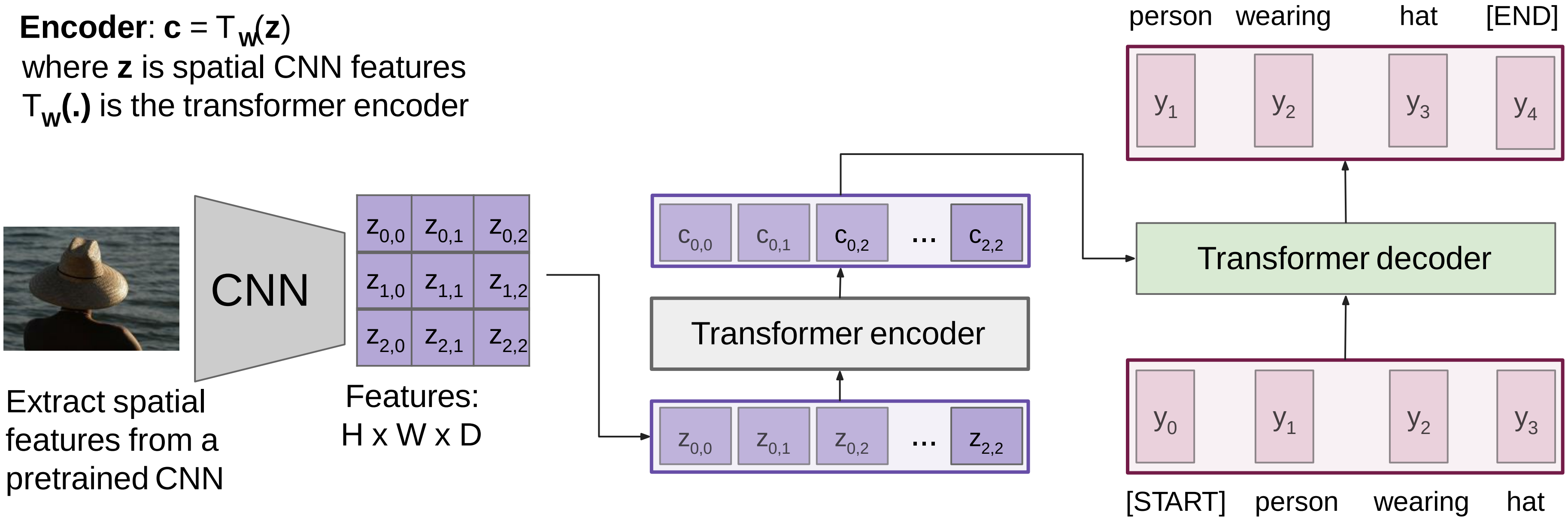


Image Captioning using Transformers

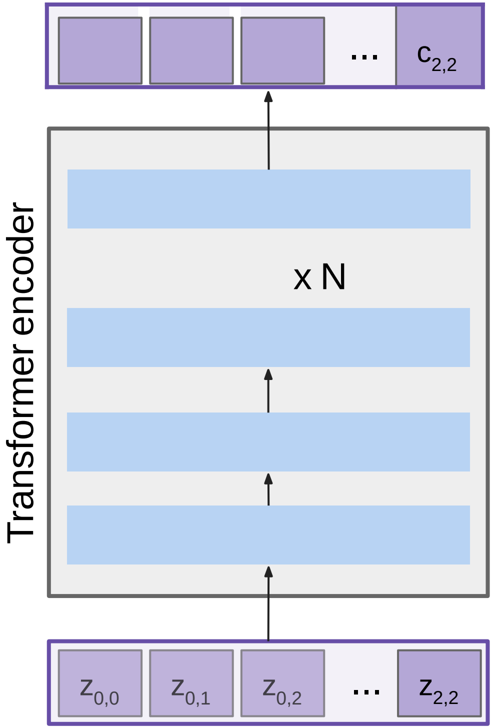
Input: Image **I**
Output: Sequence $\mathbf{y} = y_1, y_2, \ldots, y_T$

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

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



The Transformer encoder block

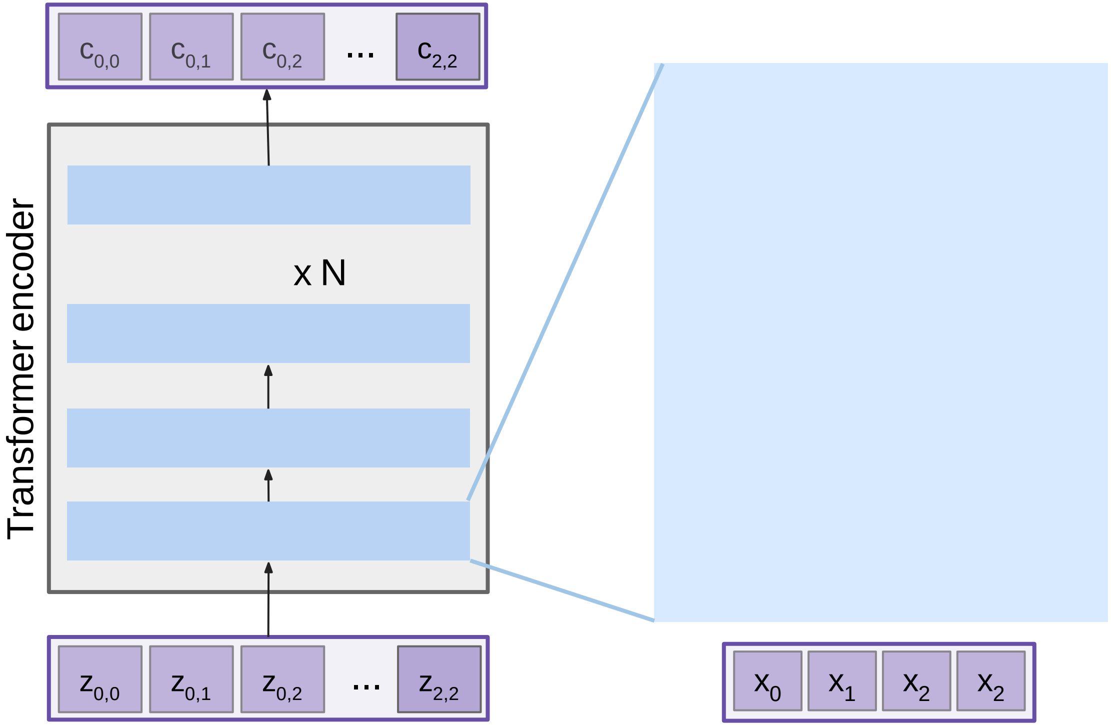


Made up of N encoder blocks.

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

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

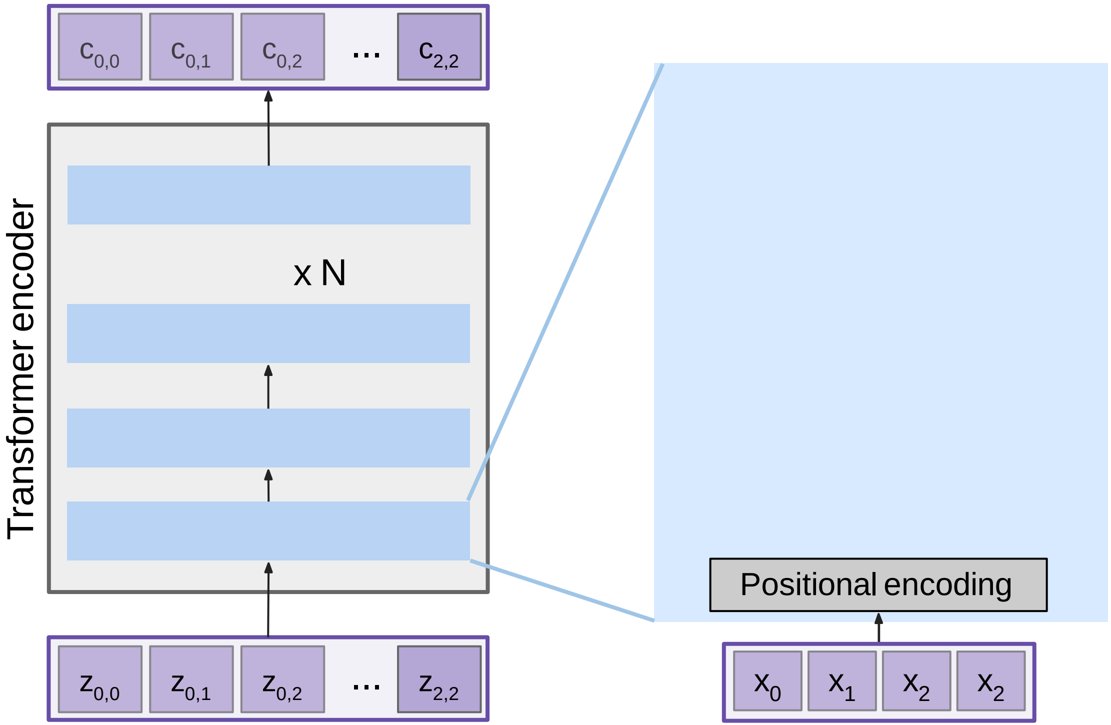
The Transformer encoder block



Let's dive into one encoder block

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

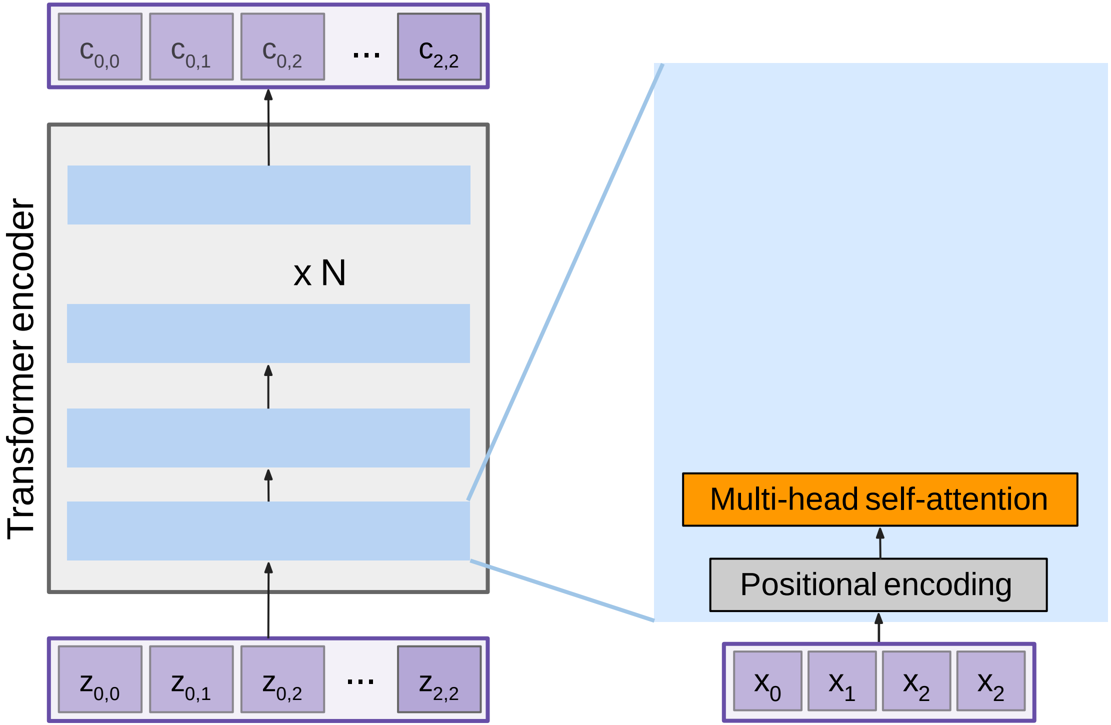
The Transformer encoder block



Add positional encoding

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

The Transformer encoder block

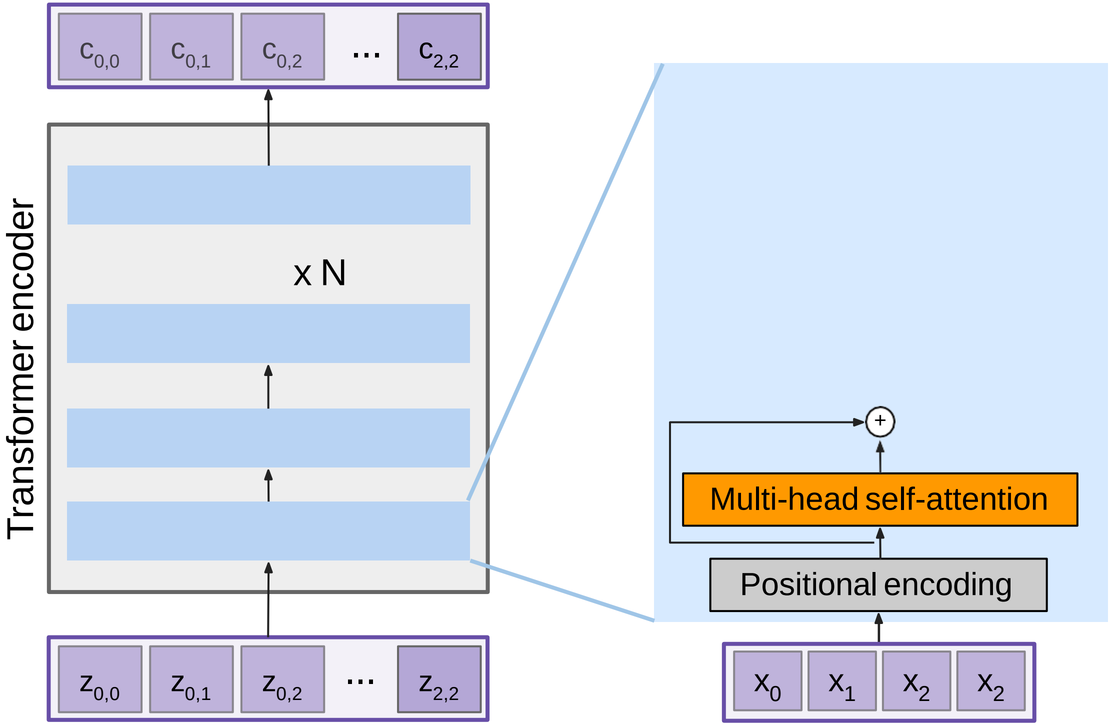


Attention attends over all the vectors

Add positional encoding

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

The Transformer encoder block



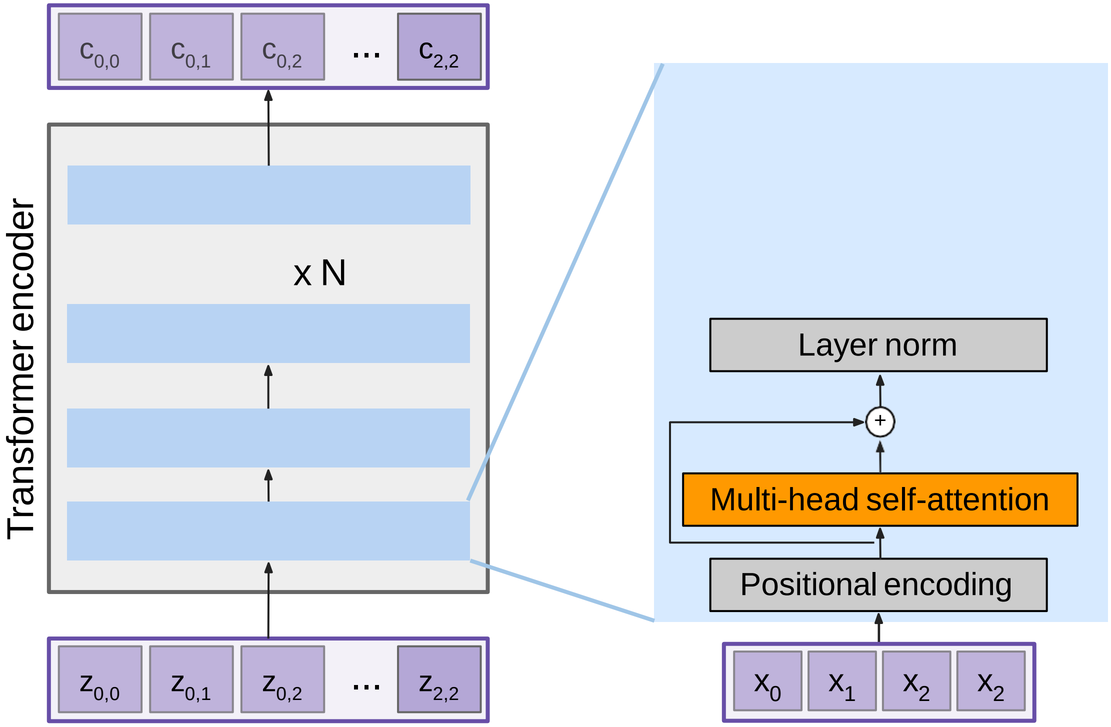
Residual connection

Attention attends over all the vectors

Add positional encoding

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

The Transformer encoder block



LayerNorm over each vector individually

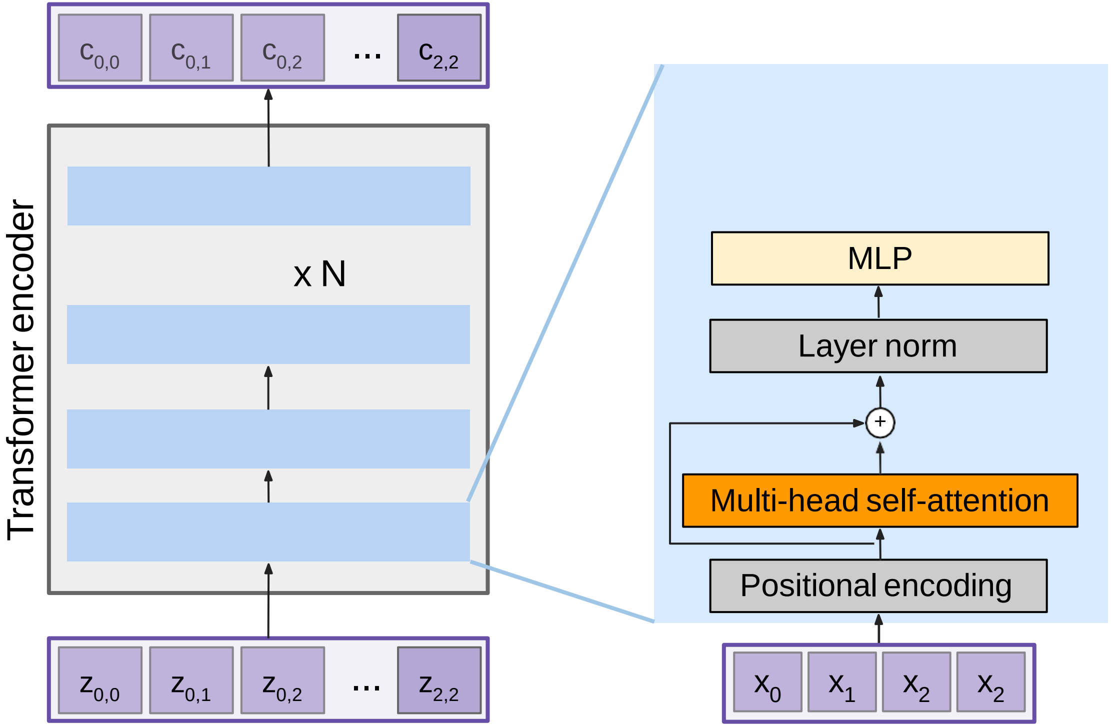
Residual connection

Attention attends over all the vectors

Add positional encoding

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

The Transformer encoder block



MLP over each vector individually

LayerNorm over each vector individually

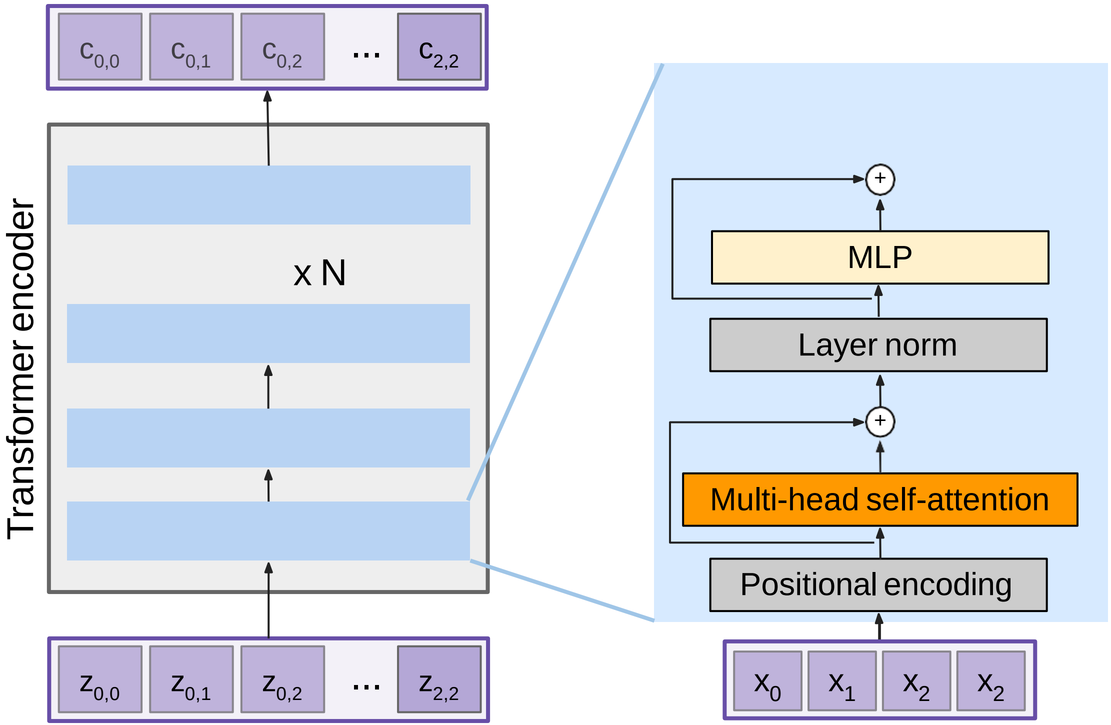
Residual connection

Attention attends over all the vectors

Add positional encoding

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

The Transformer encoder block



Residual connection

MLP over each vector individually

LayerNorm over each vector individually

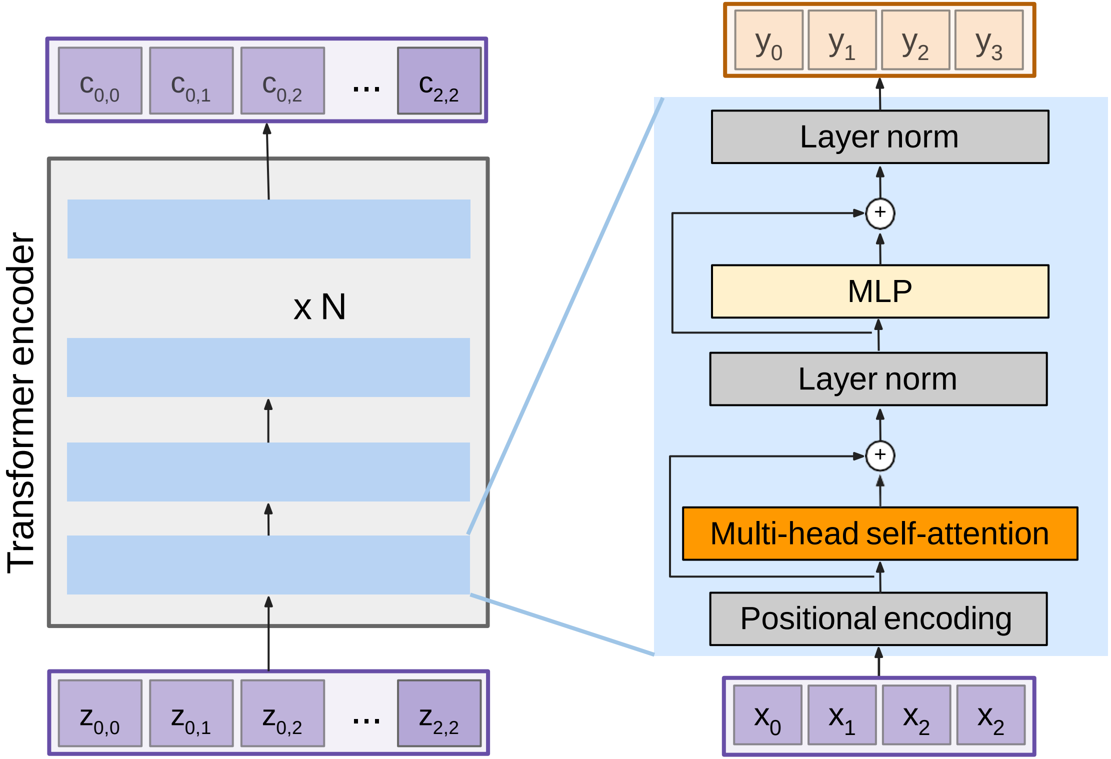
Residual connection

Attention attends over all the vectors

Add positional encoding

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

The Transformer encoder block



Transformer Encoder Block:

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

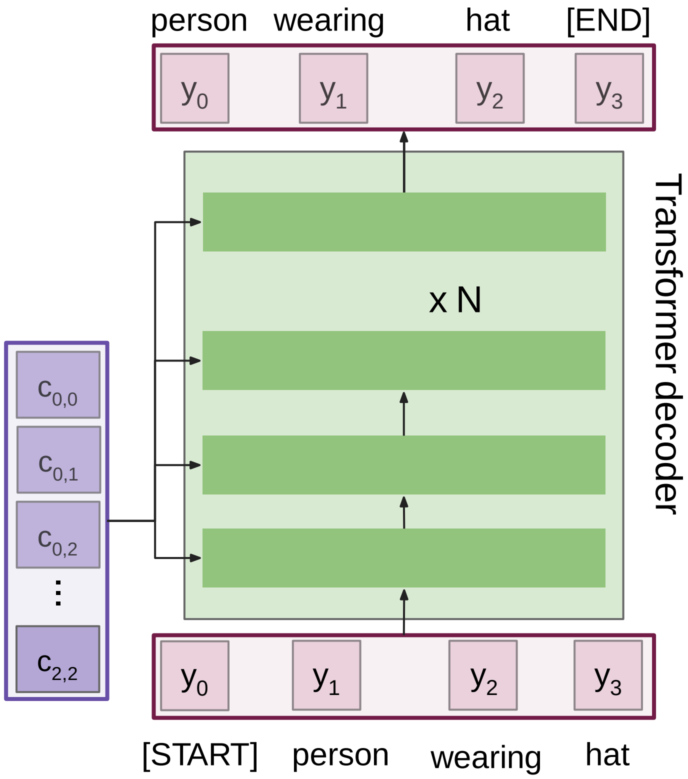
Self-attention is the only interaction between vectors.

Layer norm and MLP operate independently per vector.

Highly scalable, highly parallelizable, but high memory usage.

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

The Transformer decoder block

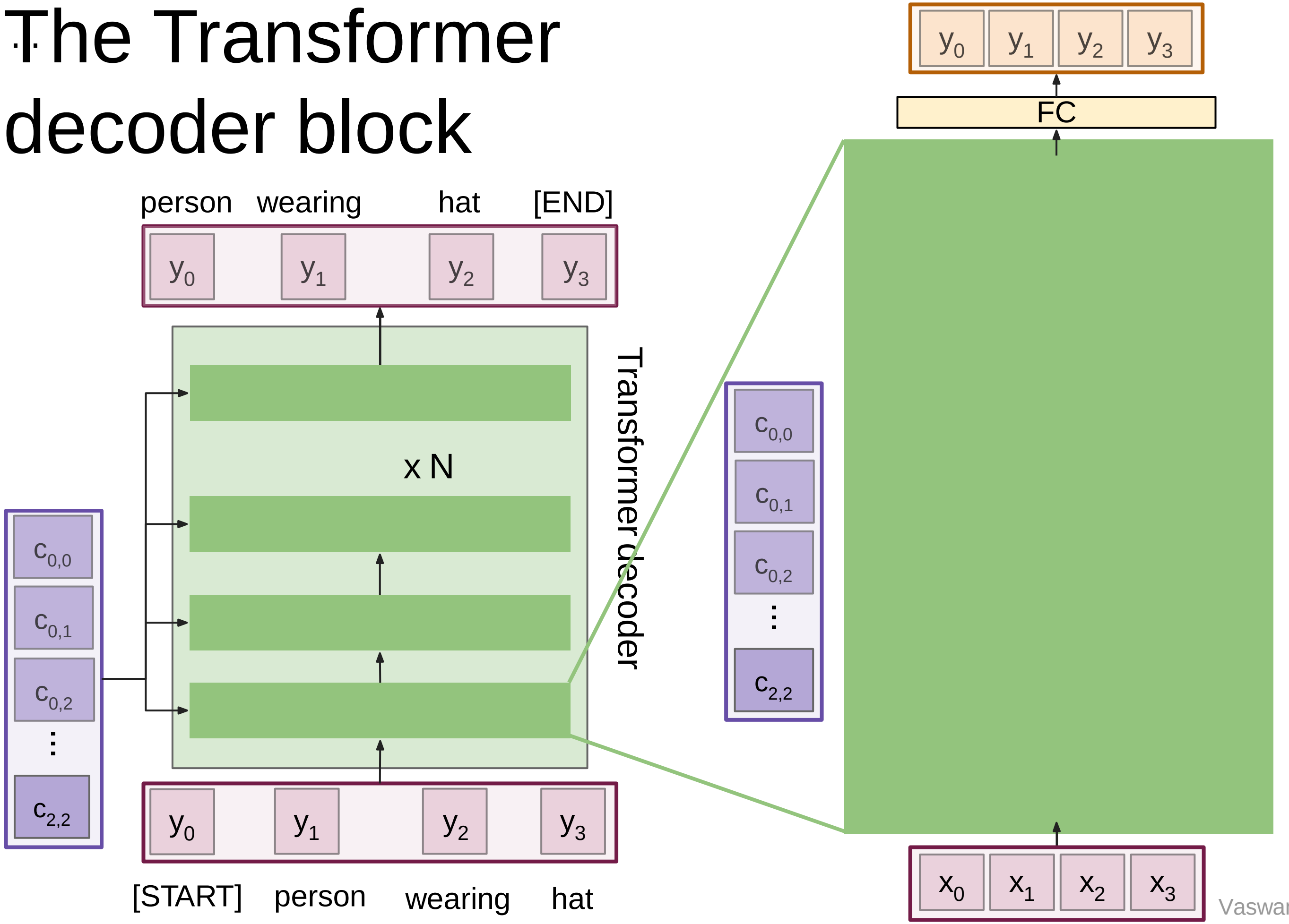


Made up of N decoder blocks.

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

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

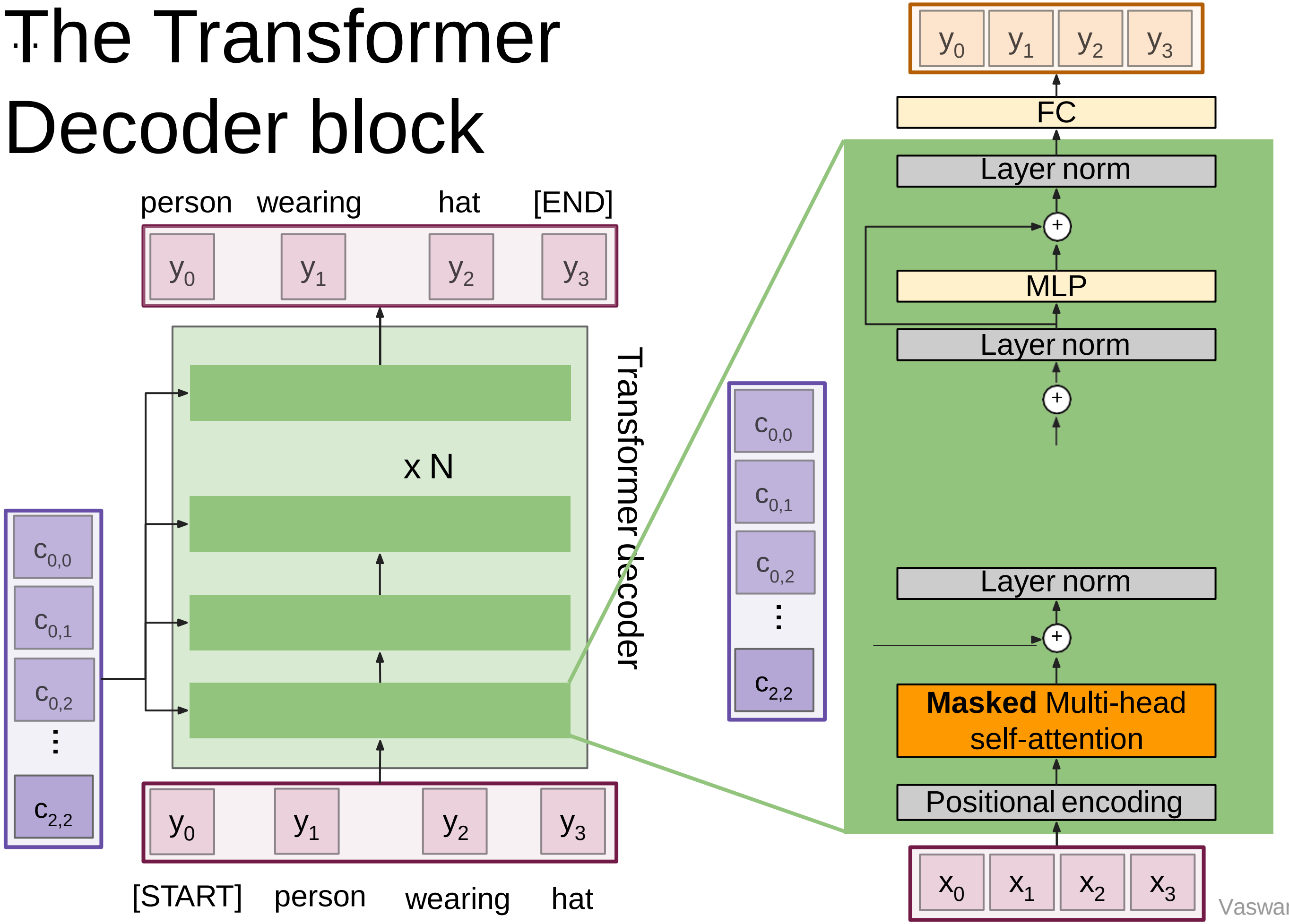
The Transformer decoder block



Let's dive into the transformer decoder block

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

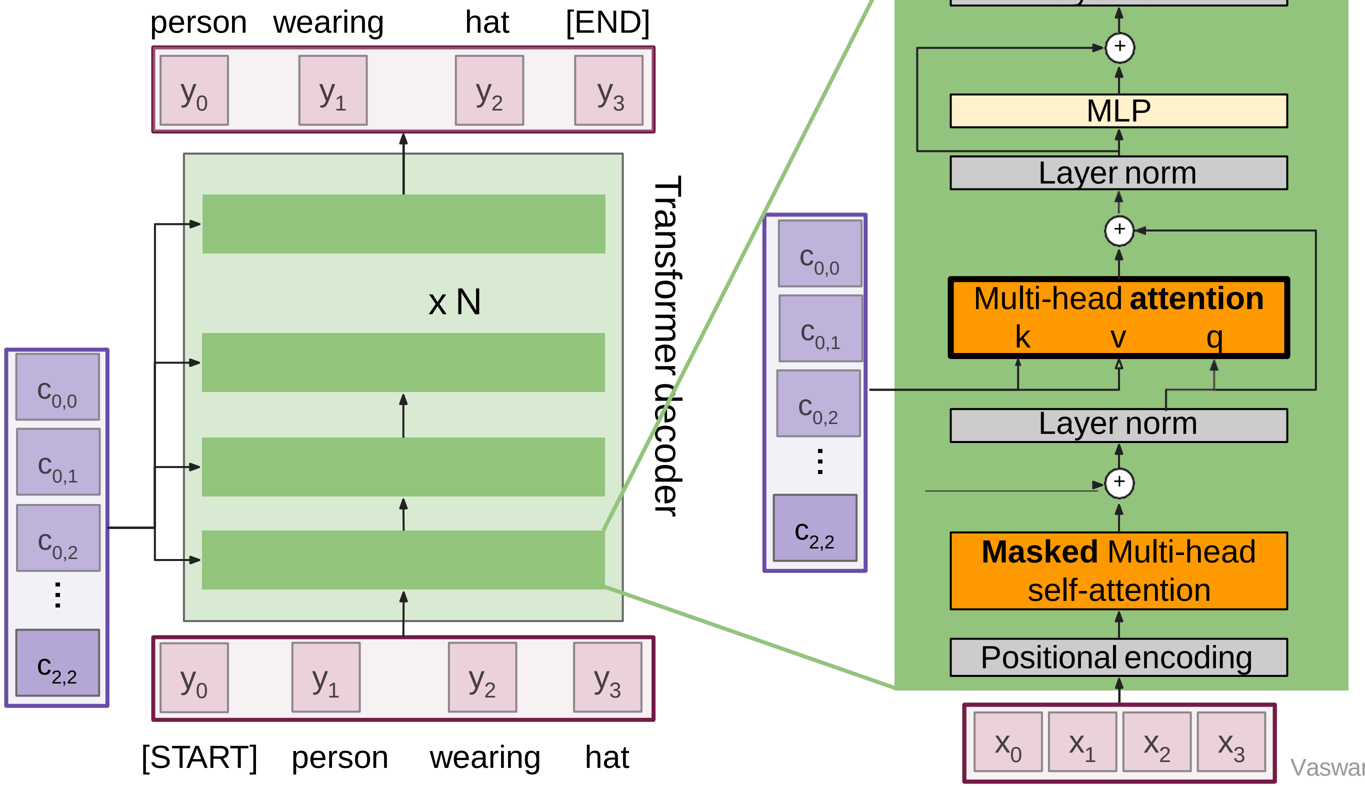
The Transformer Decoder block



Most of the network is the same the transformer encoder.

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

The Transformer Decoder block

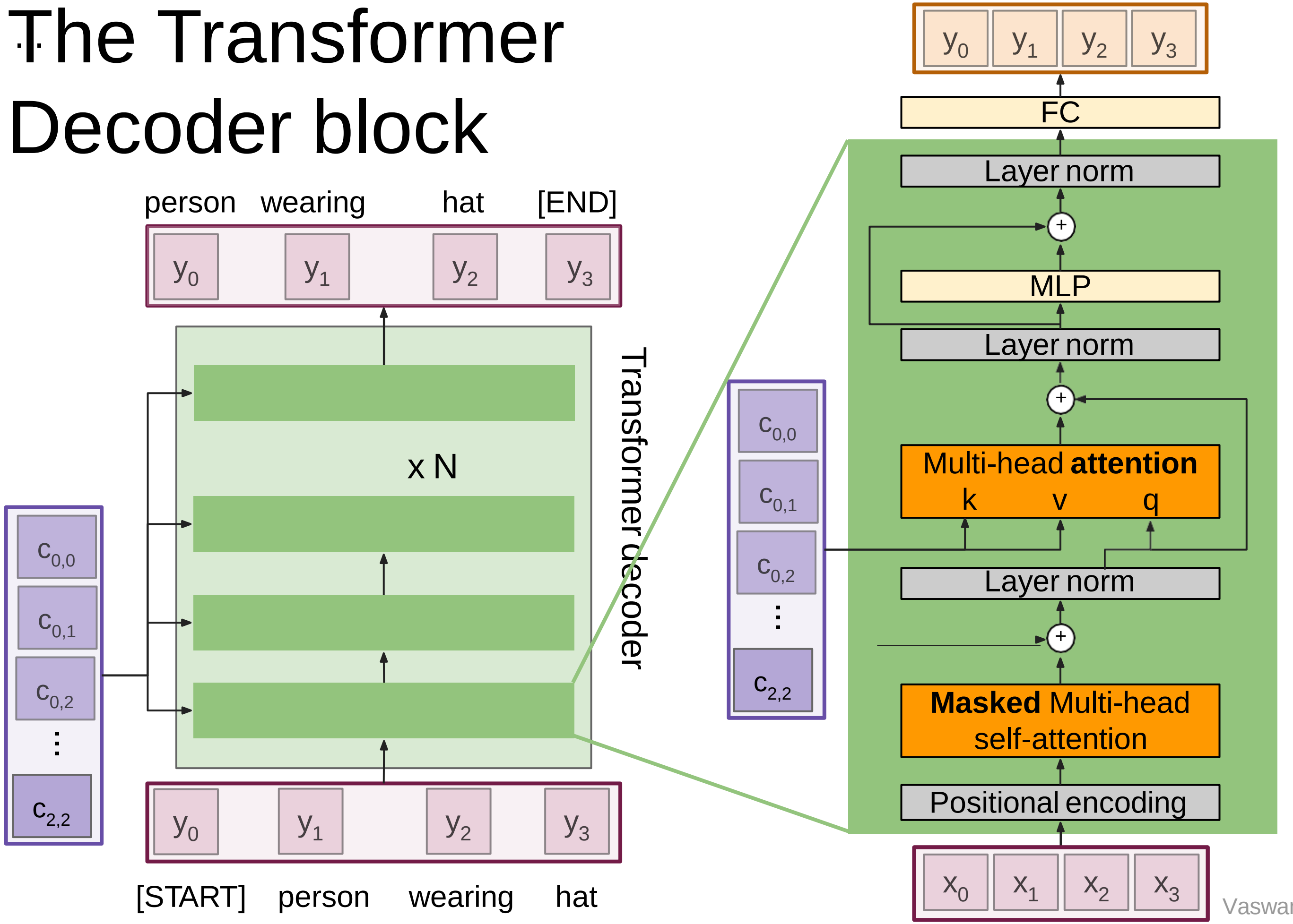


Multi-head attention block attends over the transformer encoder outputs.

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

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

The Transformer Decoder block



Transformer Decoder Block:

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

Masked Self-attention only interacts with past inputs.

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

Highly scalable, highly parallelizable, but high memory usage.

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

Image Captioning using transformers

- **No recurrence at all**

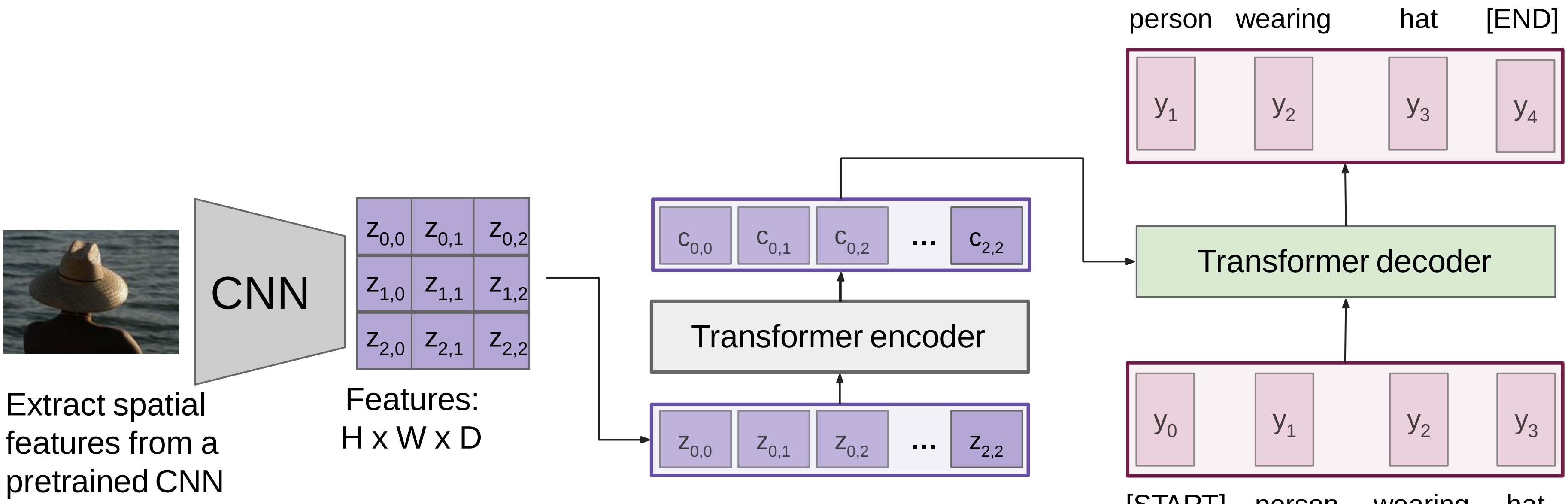


Image Captioning using transformers

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

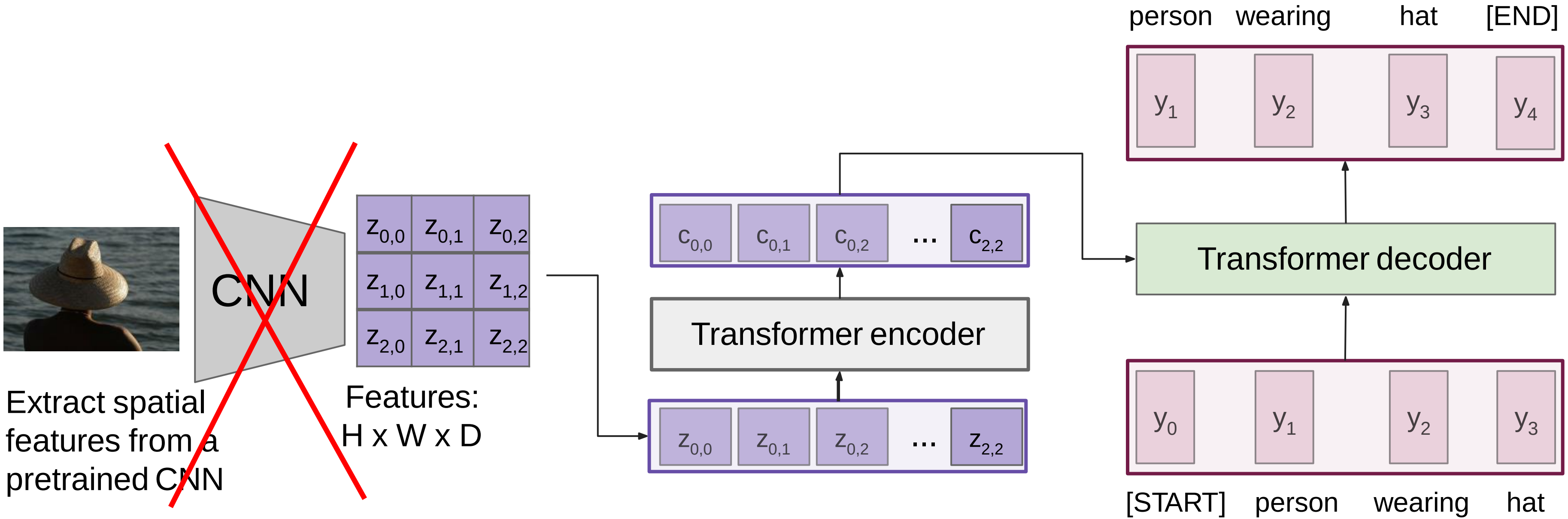
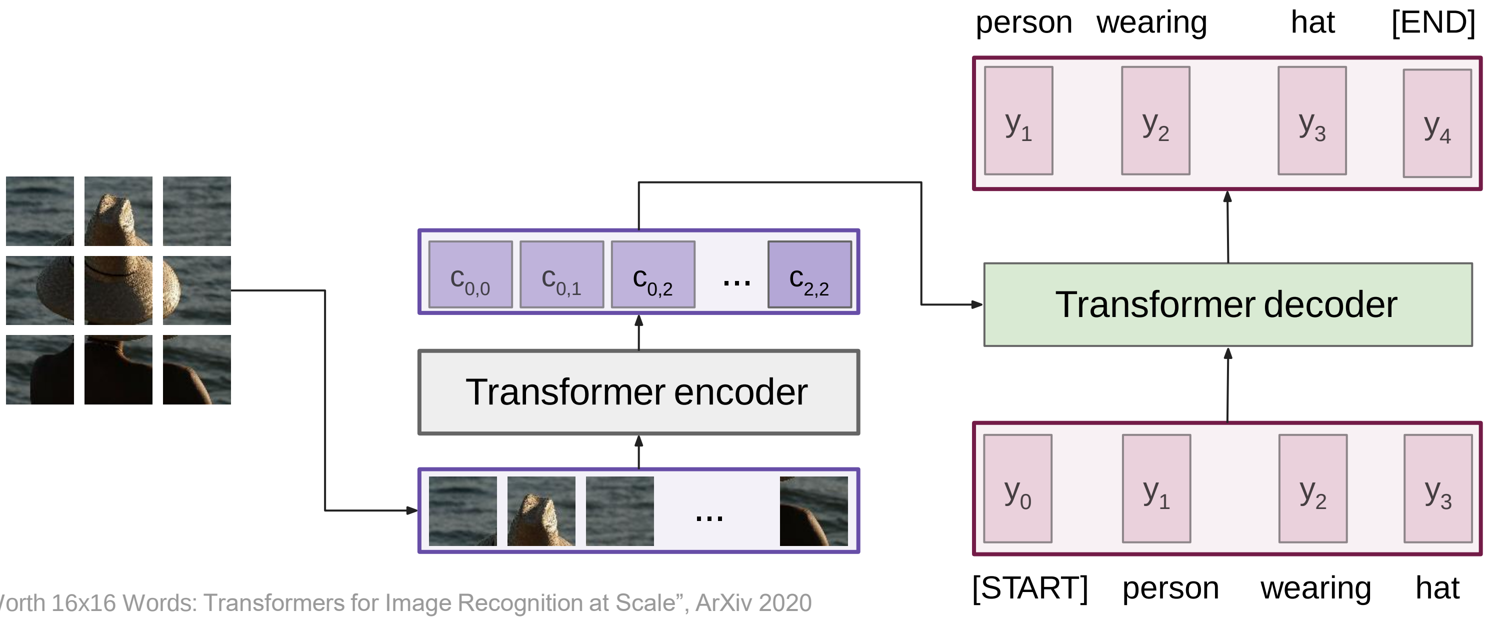


Image Captioning using ONLY transformers

- **Transformers from pixels to language**



Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ArXiv 2020

[Colab link](#) to an implementation of vision transformers

Vision Transformers vs. ResNets

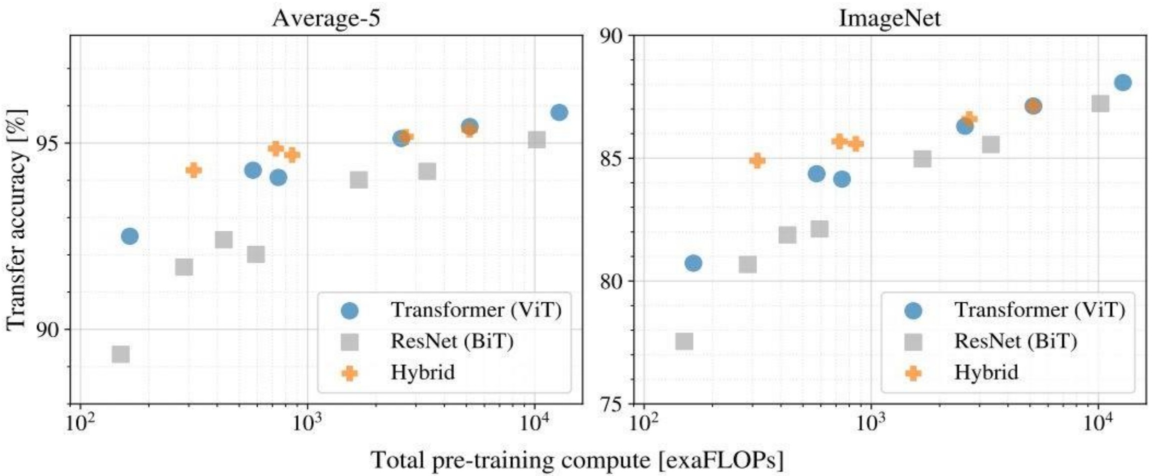
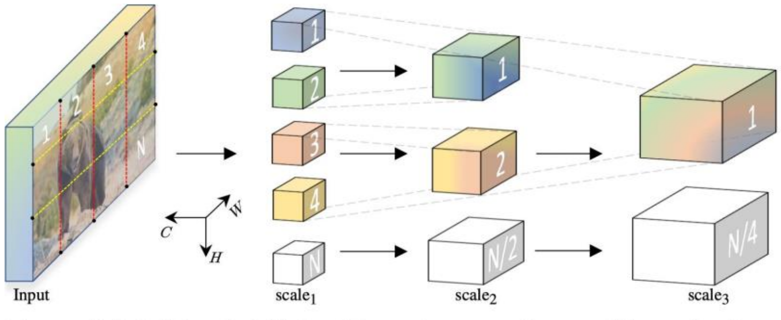


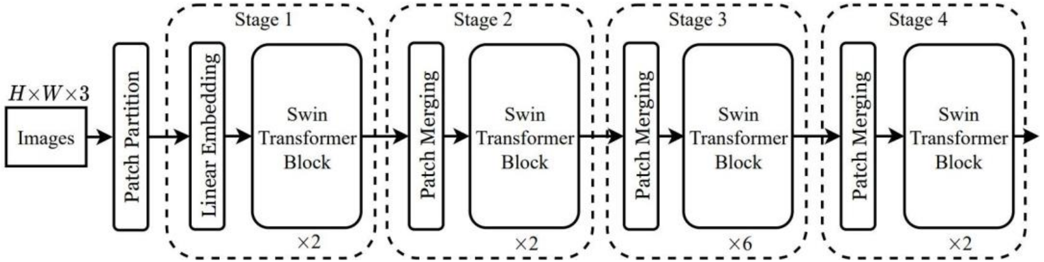
Figure 5: Performance versus cost for different architectures: Vision Transformers, ResNets, and hybrids. Vision Transformers generally outperform ResNets with the same computational budget. Hybrids improve upon pure Transformers for smaller model sizes, but the gap vanishes for larger models.

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

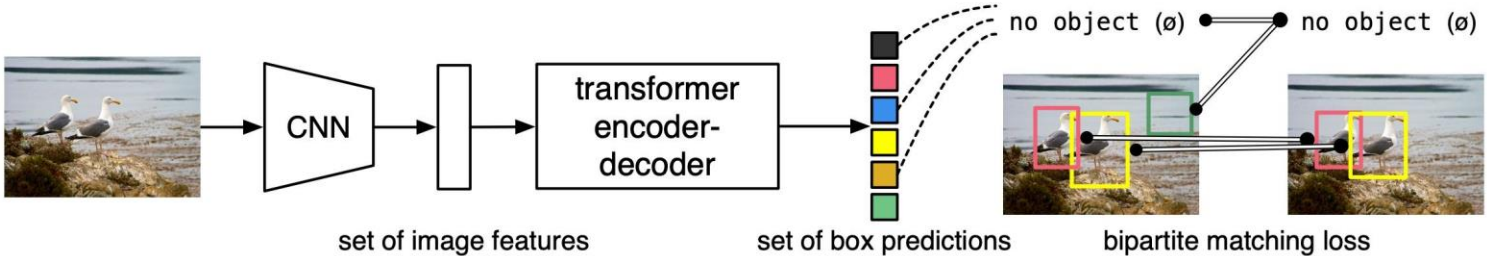
Vision Transformers



Fan et al, “Multiscale Vision Transformers”, ICCV 2021

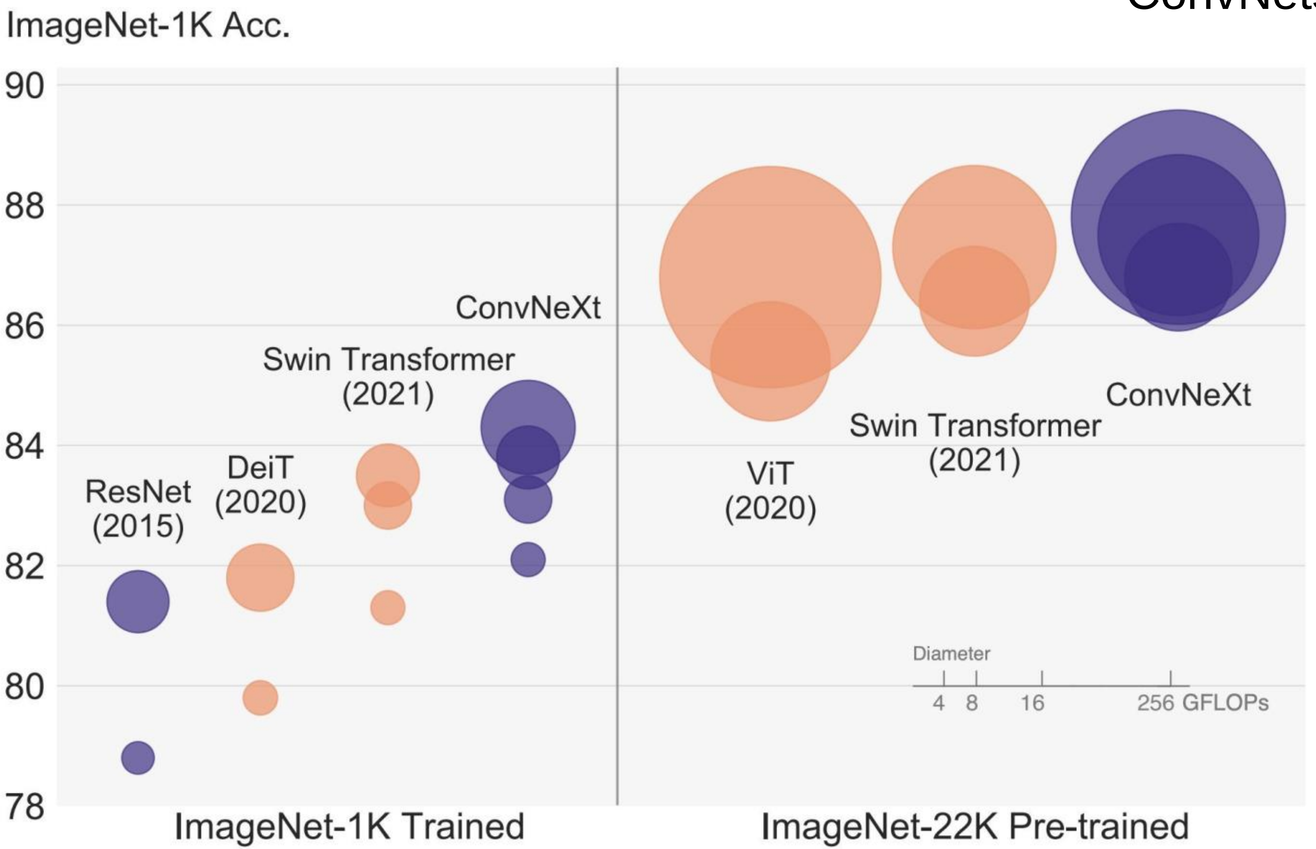


Liu et al, “Swin Transformer: Hierarchical Vision Transformer using Shifted Windows”, CVPR 2021



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

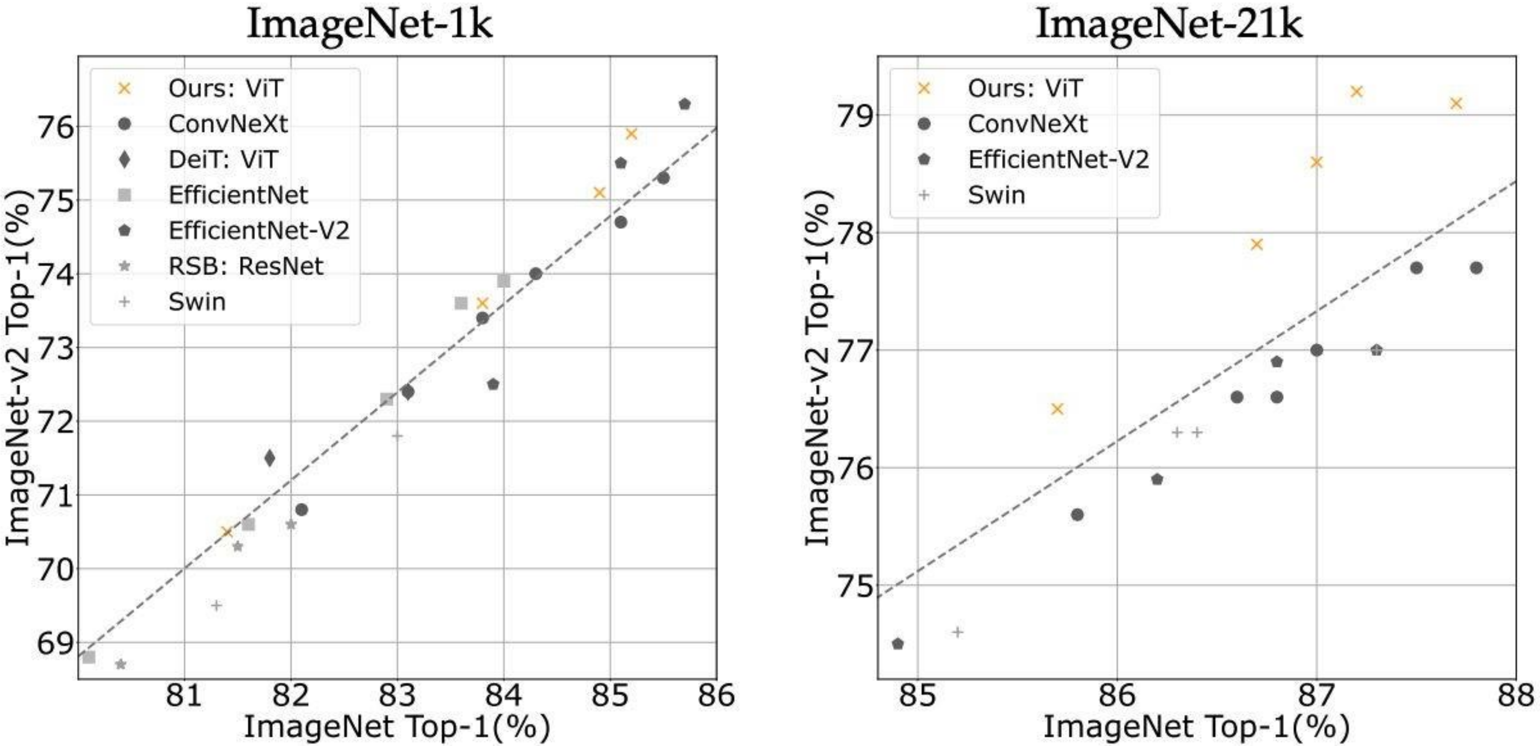
ConvNets strike back!



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.