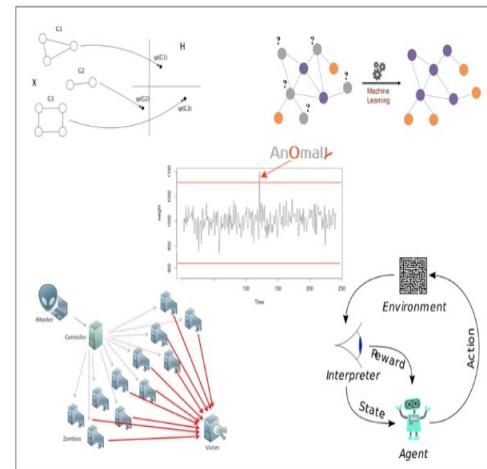
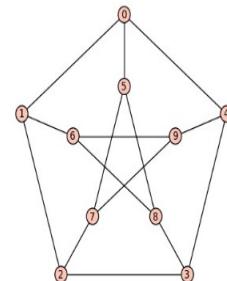


RNNs

Pierre Pereira
Université Côte d'Azur / Inria Coati
pierre.pereira@inria.fr



Sources multiples, principalement
Fei-Fei Li, Stanford

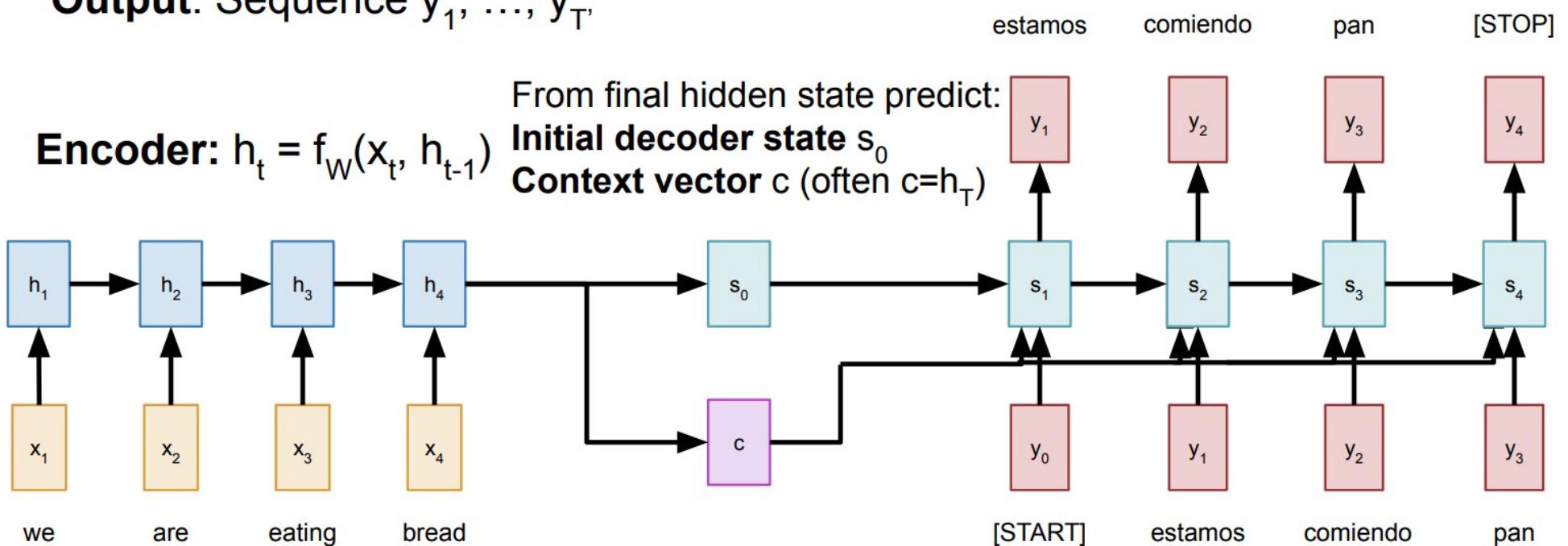
$$\begin{aligned} \min \quad & \sum_{e \in \mathcal{E}} y_e \\ \text{s.t.} \quad & \sum_{a \in A_i^+(u)} f_a^i - \sum_{a \in A_i^-(u)} f_a^i = \begin{cases} |V_i| - 1 & \text{if } u = s_i \\ -1 & \text{if } u \neq s_i \end{cases} \quad \forall u \in V_i, V_i \in C \\ & f_a^i \leq |V_i| \cdot x_a, \quad \forall V_i \in C, a \in A \\ & x_{(u,v)} \leq y_{uv}, \quad \forall uv \in \mathcal{E} \\ & x_{(v,u)} \leq y_{uv}, \quad \forall uv \in \mathcal{E} \end{aligned}$$

Sequence to Sequence with RNNs

Input: Sequence x_1, \dots, x_T

Output: Sequence y_1, \dots, y_T

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



Sequence to Sequence with RNNs

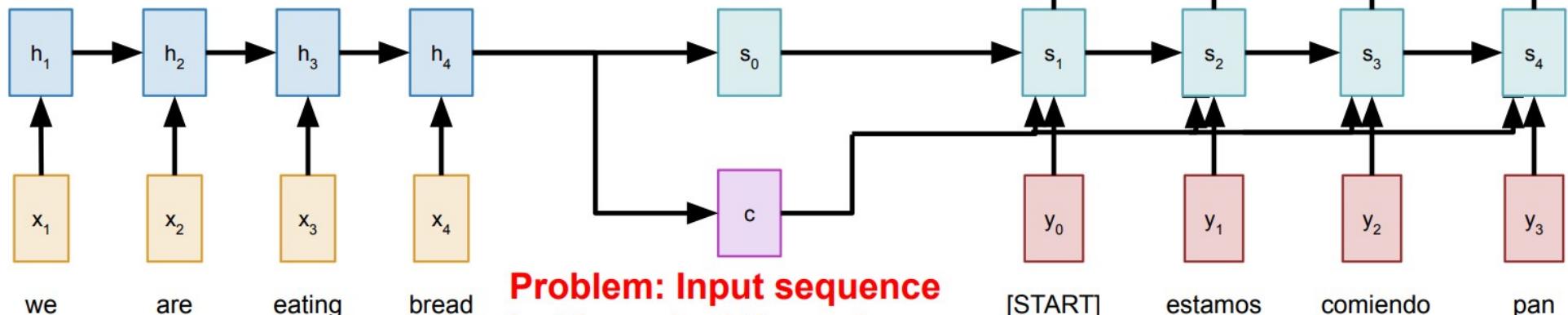
Input: Sequence x_1, \dots, x_T

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 $c=h_T$)

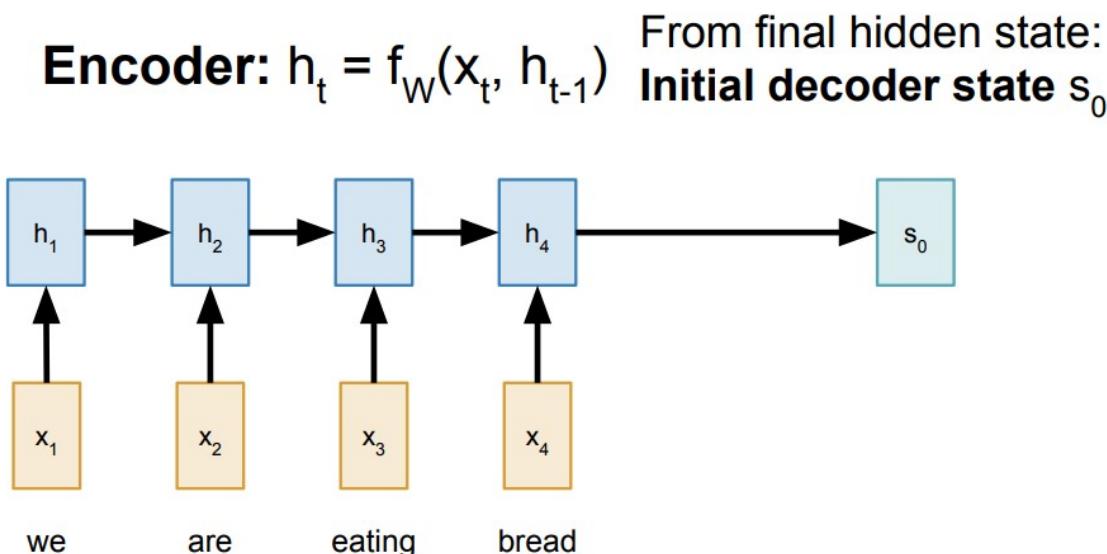


Problem: Input sequence
bottlenecked through
fixed-sized vector. What if
 $T=1000$?

Sequence to Sequence with RNNs and **Attention**

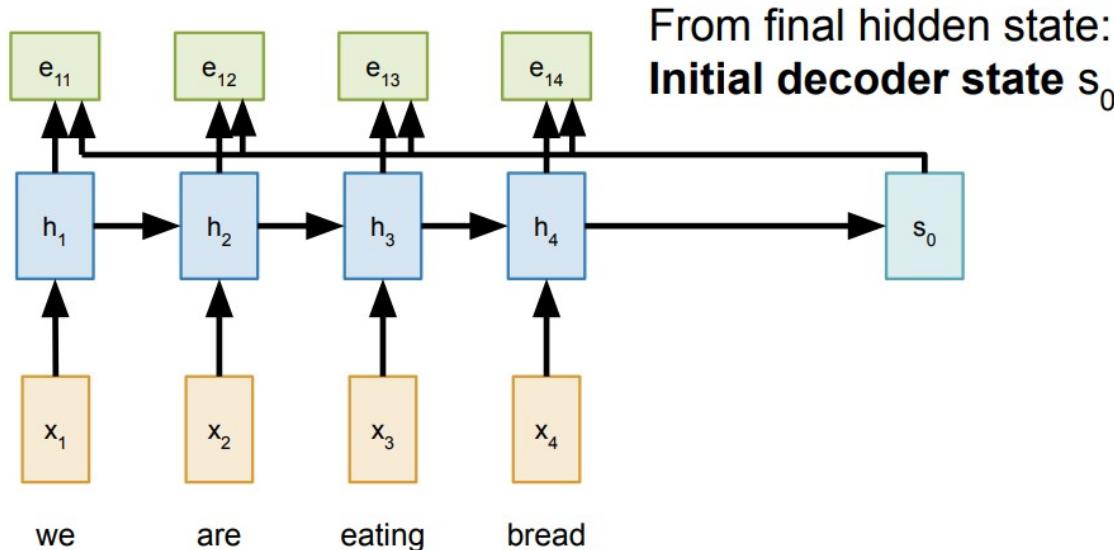
Input: Sequence x_1, \dots, x_T

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

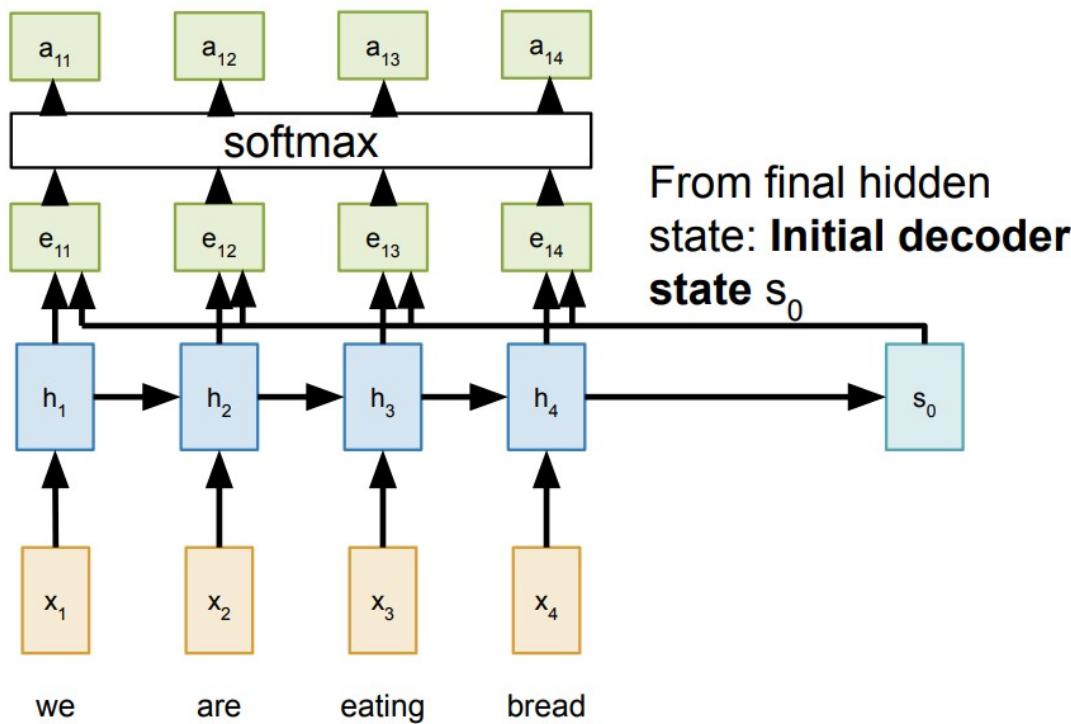


Sequence to Sequence with RNNs and Attention

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



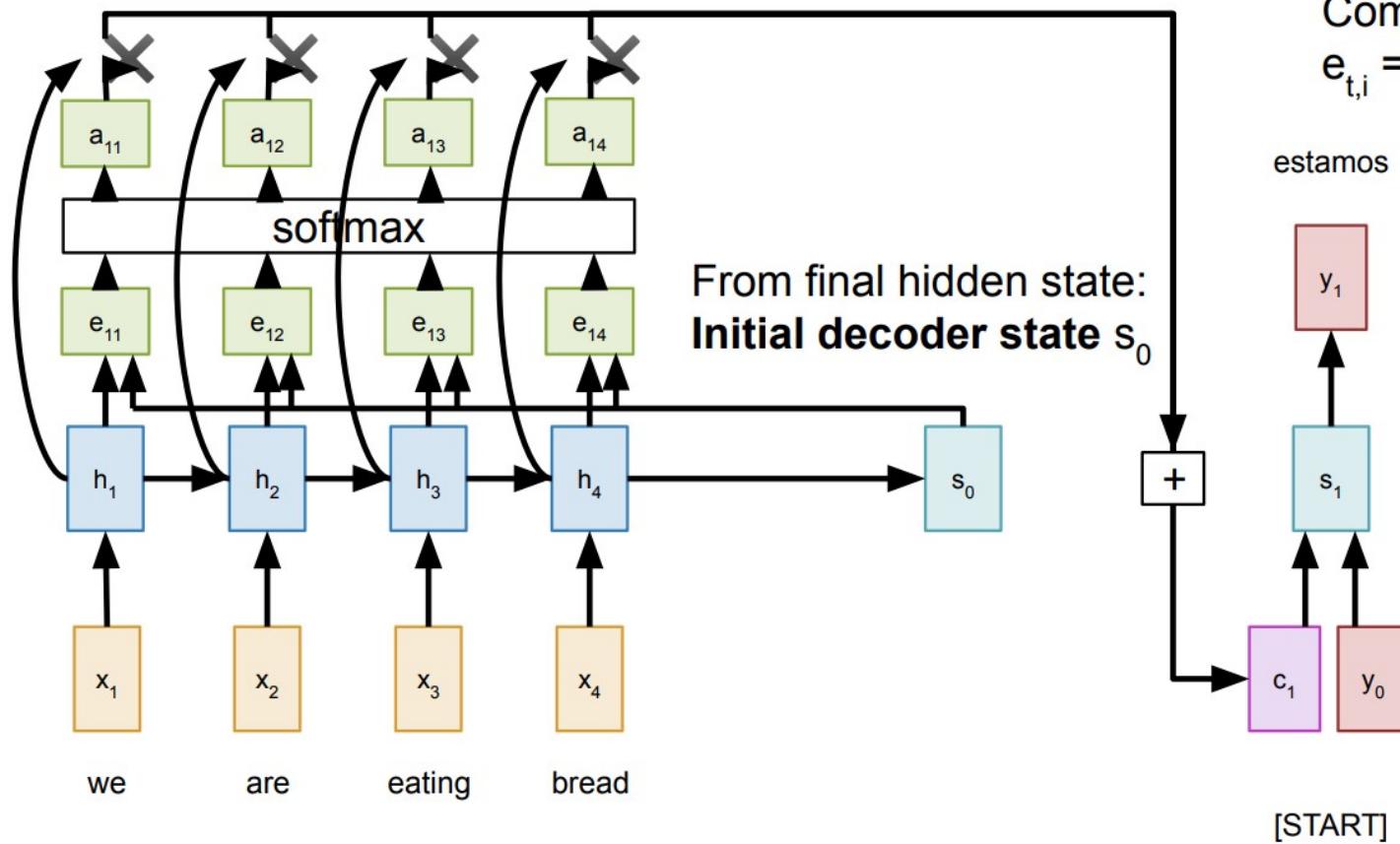
Sequence to Sequence with RNNs and Attention



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

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)

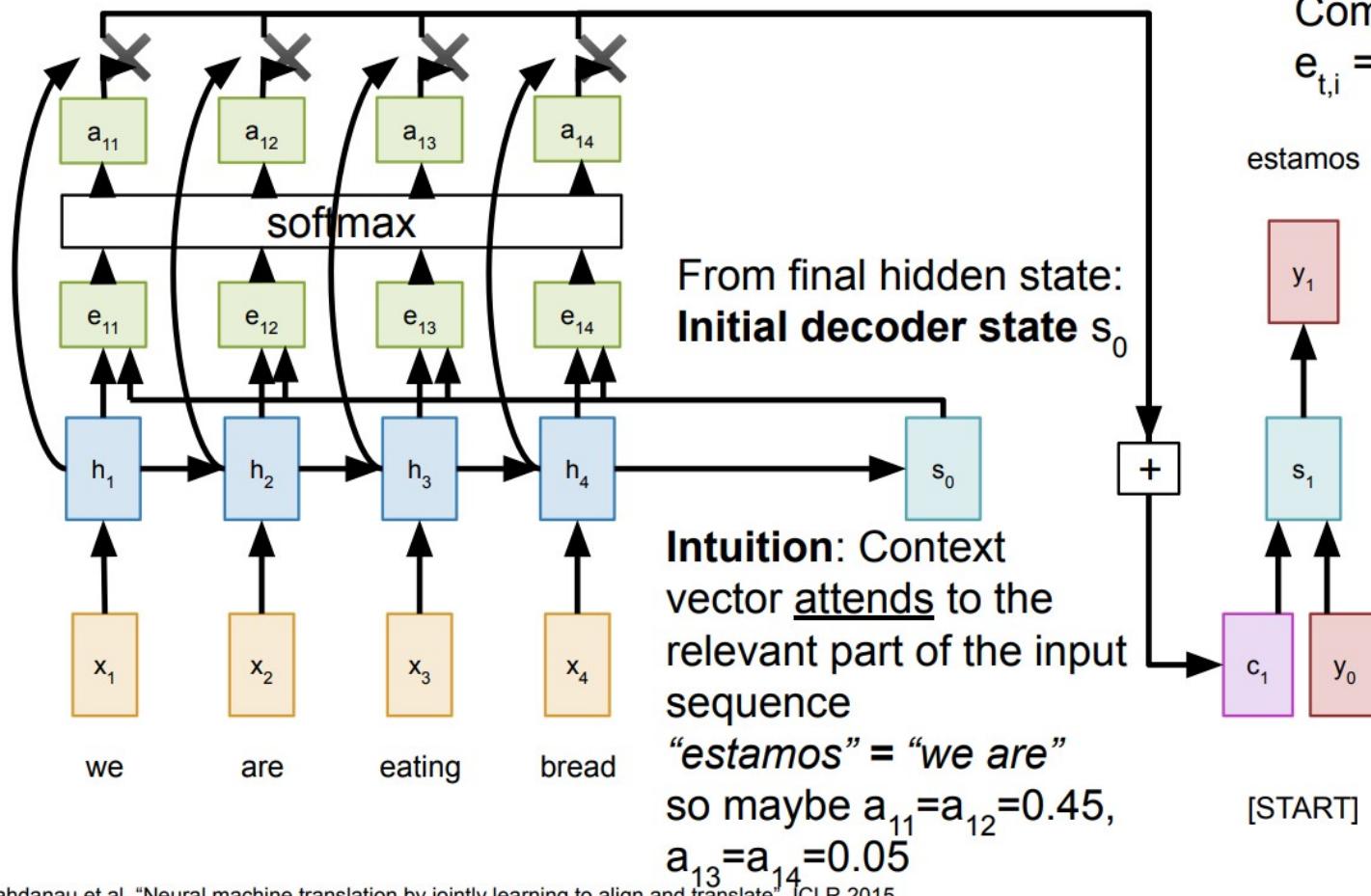
estamos

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



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

estamos

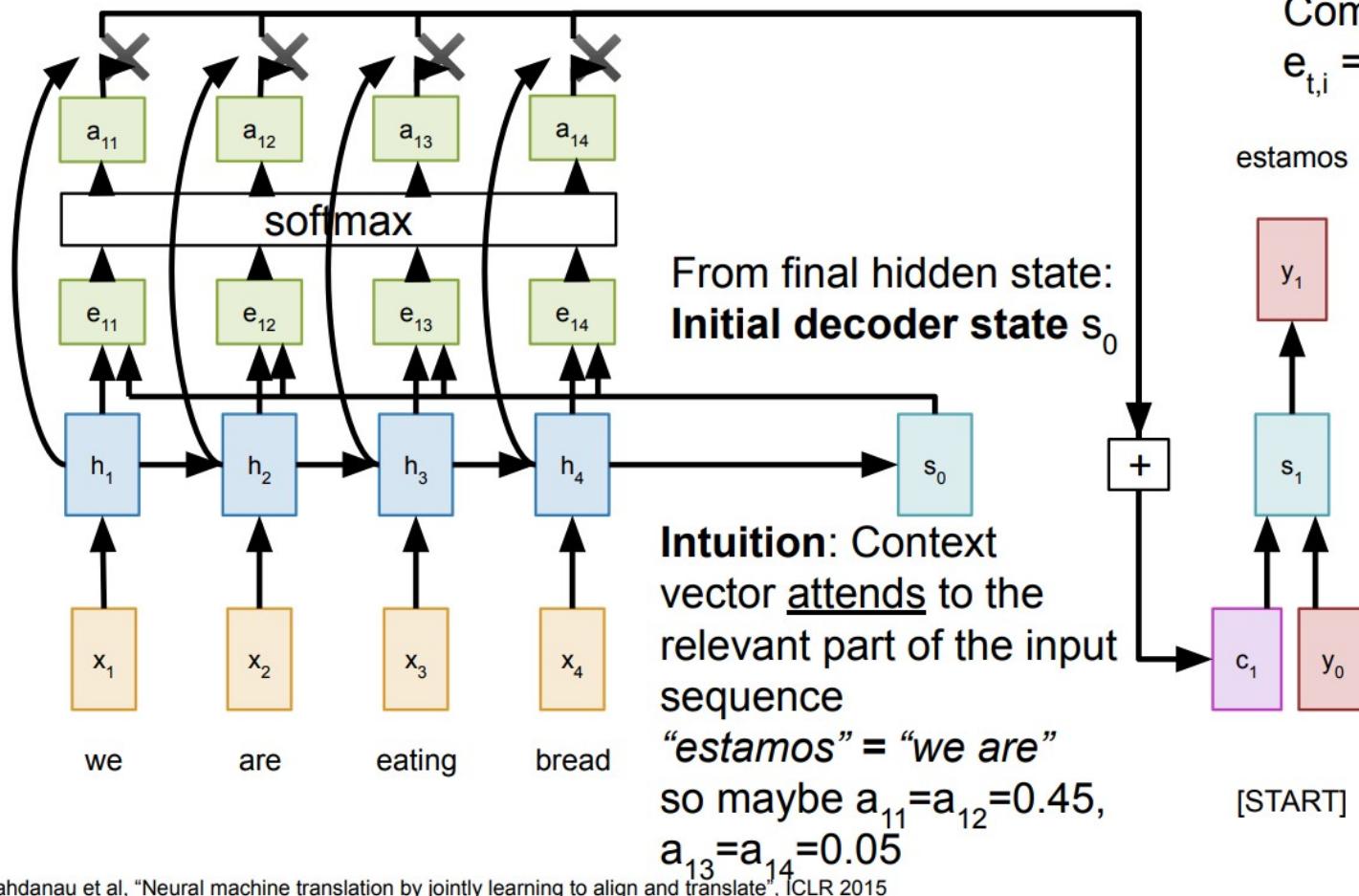
Normalize alignment scores to get **attention weights**
 $0 < a_{t,i} < 1 \quad \sum_i a_{t,i} = 1$

Compute context vector as linear combination of hidden states

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

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

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)

estamos

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

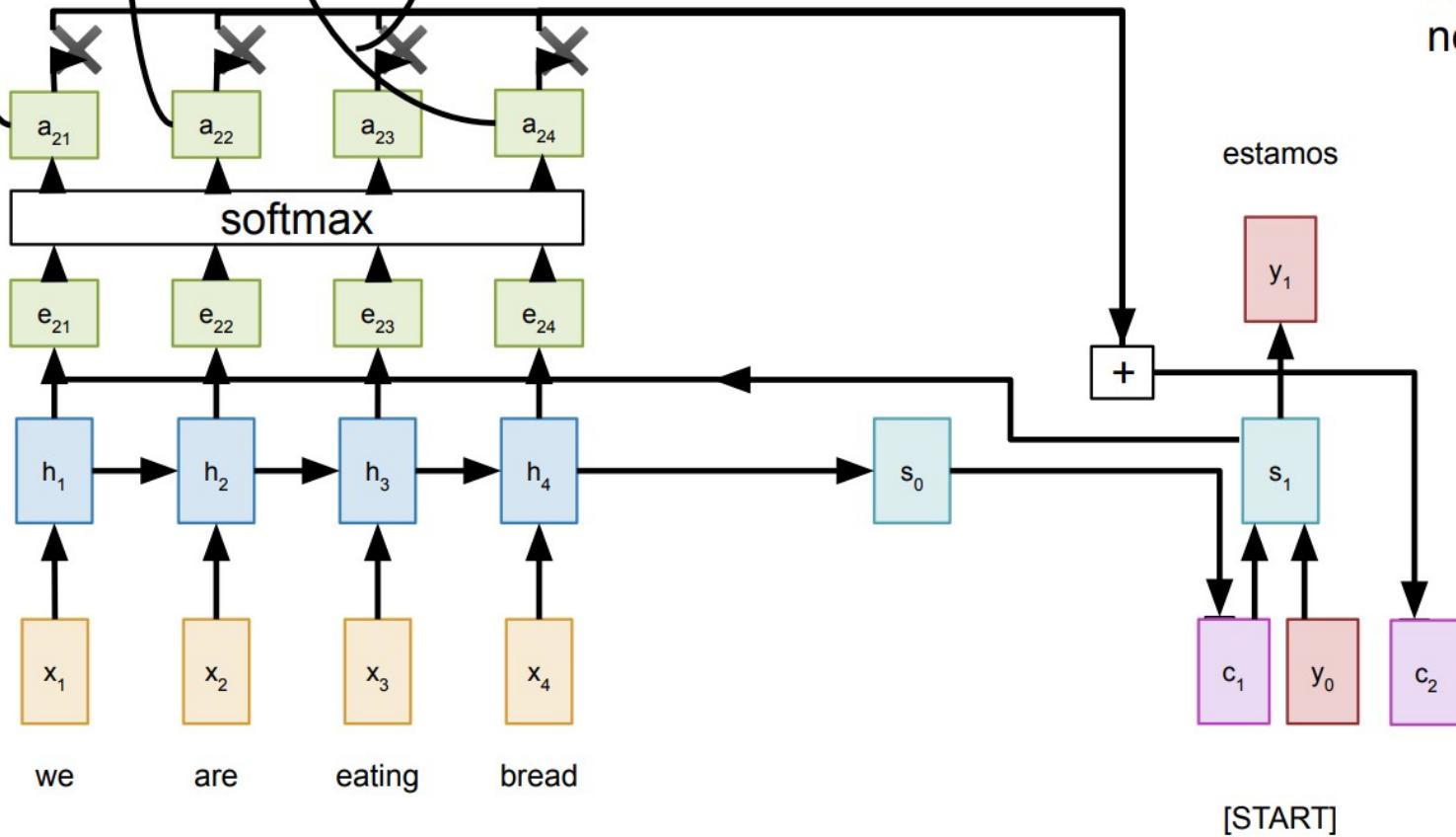
$$c_t = \sum_i a_{t,i} h_i$$

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

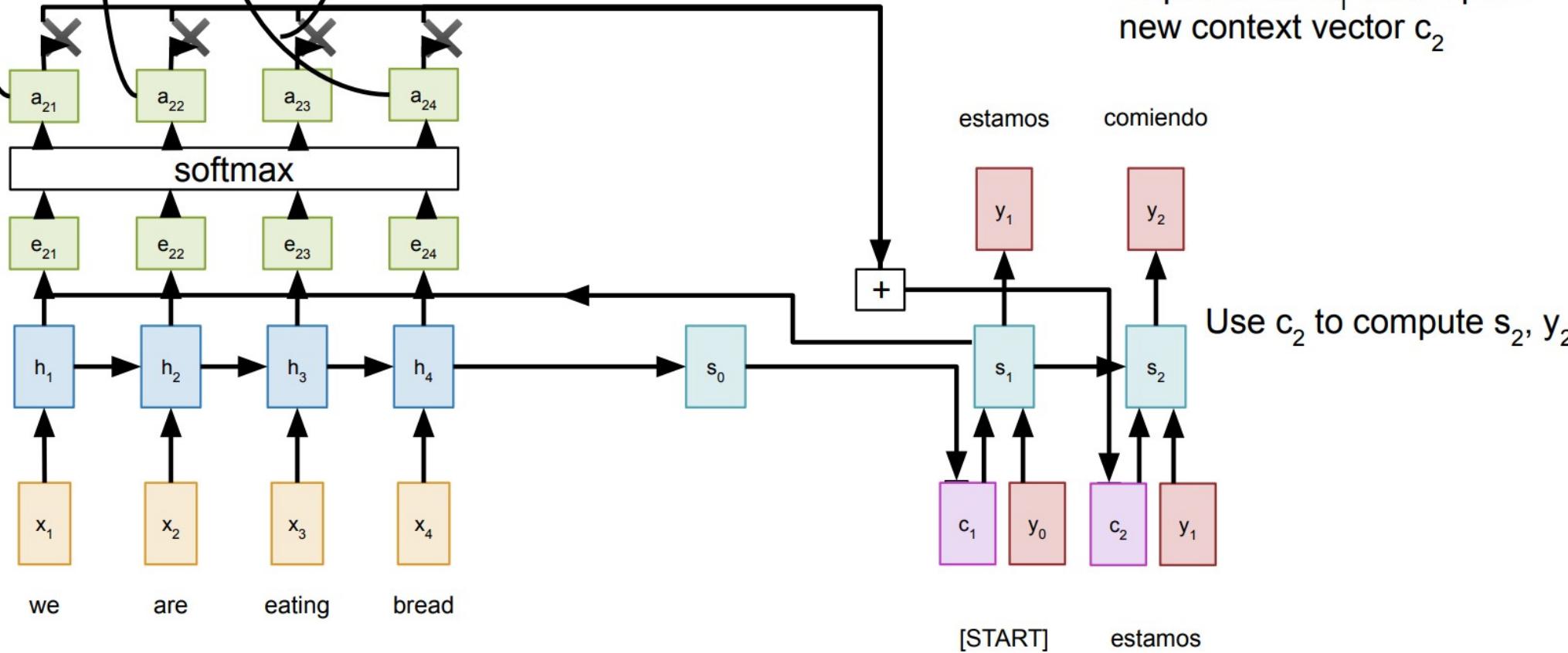
Sequence to Sequence with RNNs and Attention

Repeat: Use s_1 to compute new context vector c_2



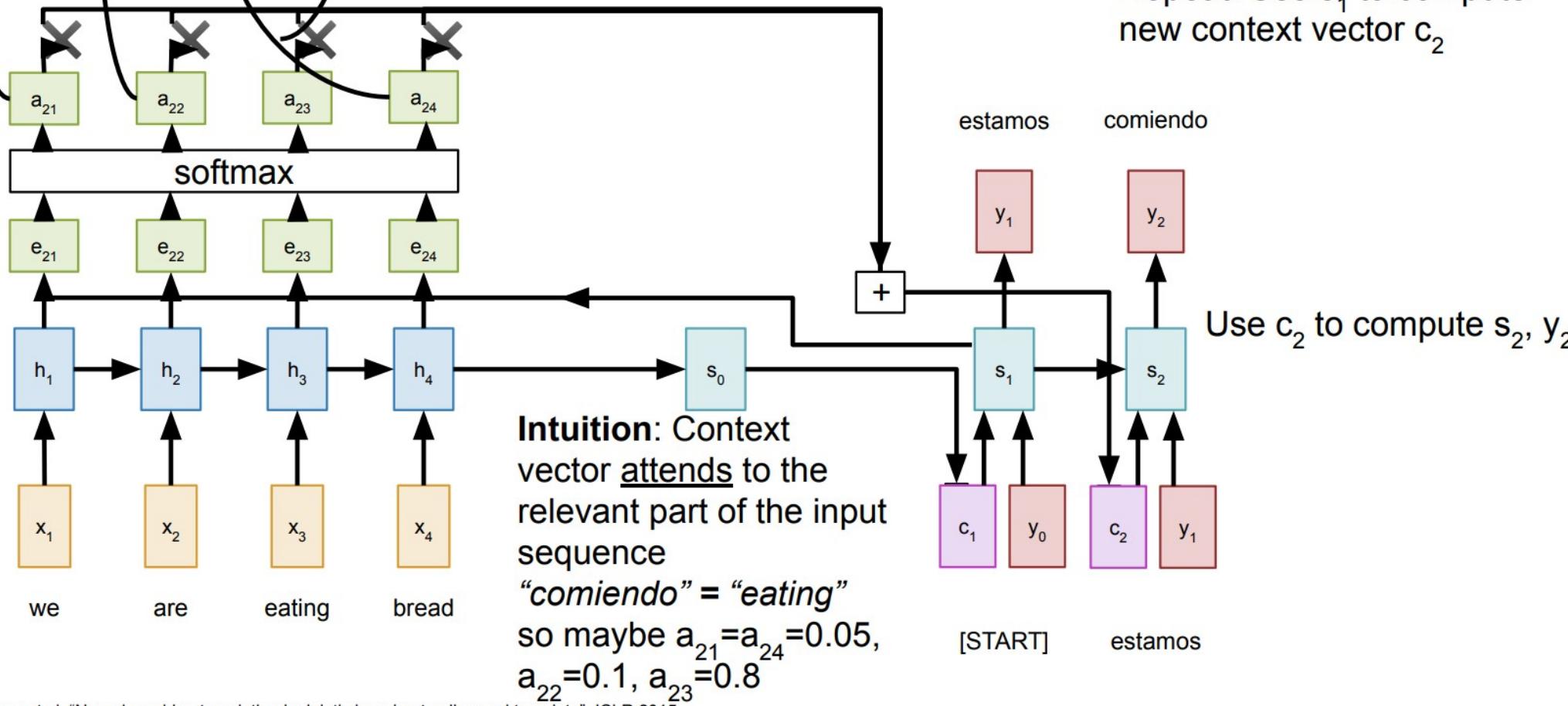
[START]

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

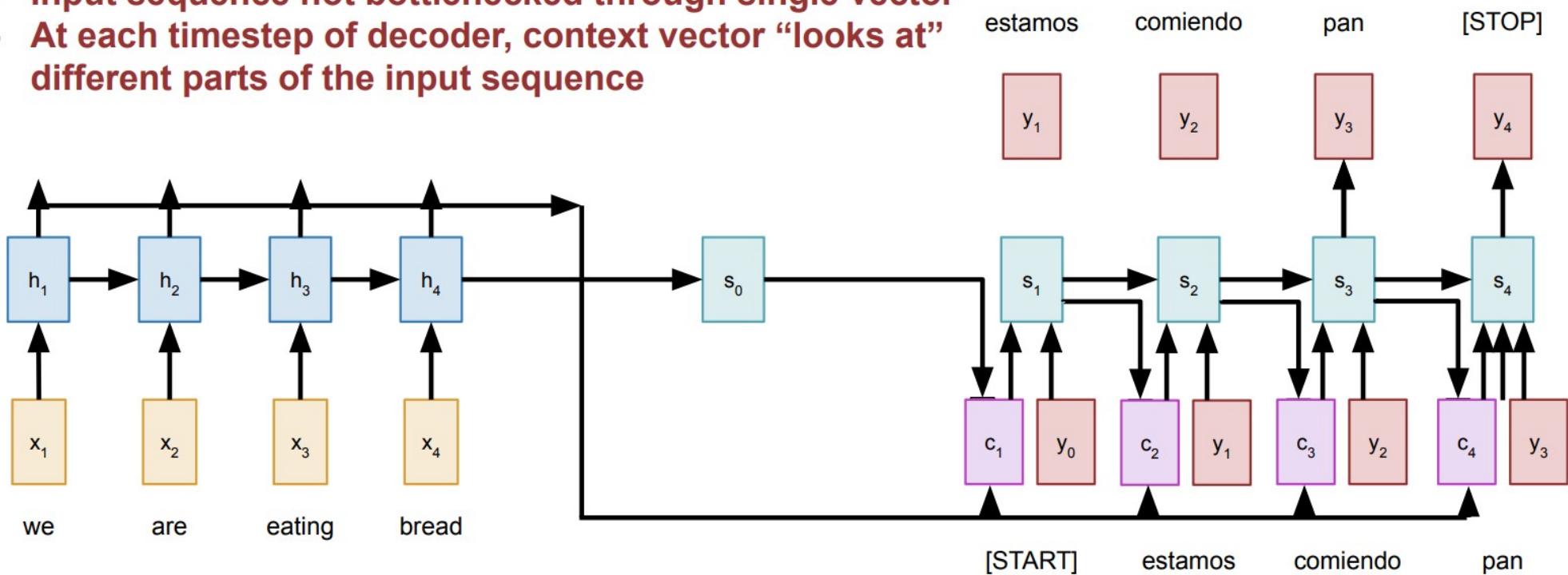


Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

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



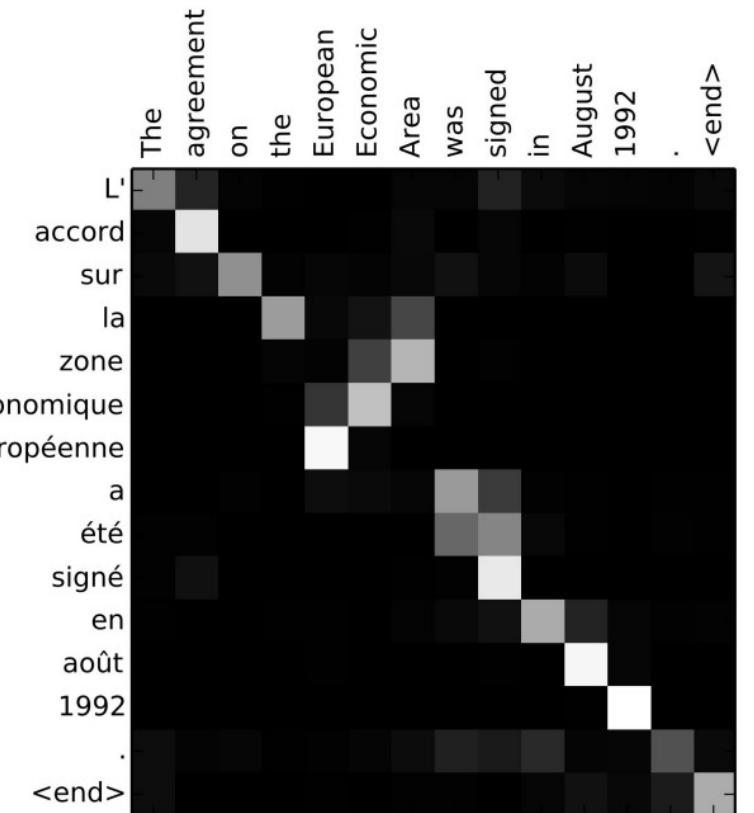
Sequence to Sequence with RNNs and Attention

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

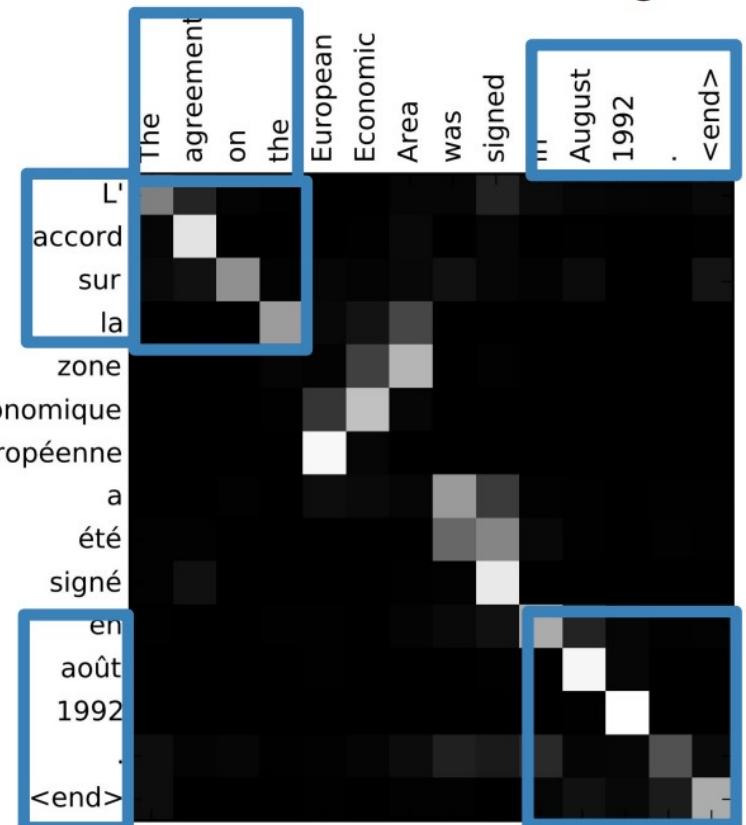
Input: “**The agreement on the European Economic Area was signed in August 1992.**”

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.”

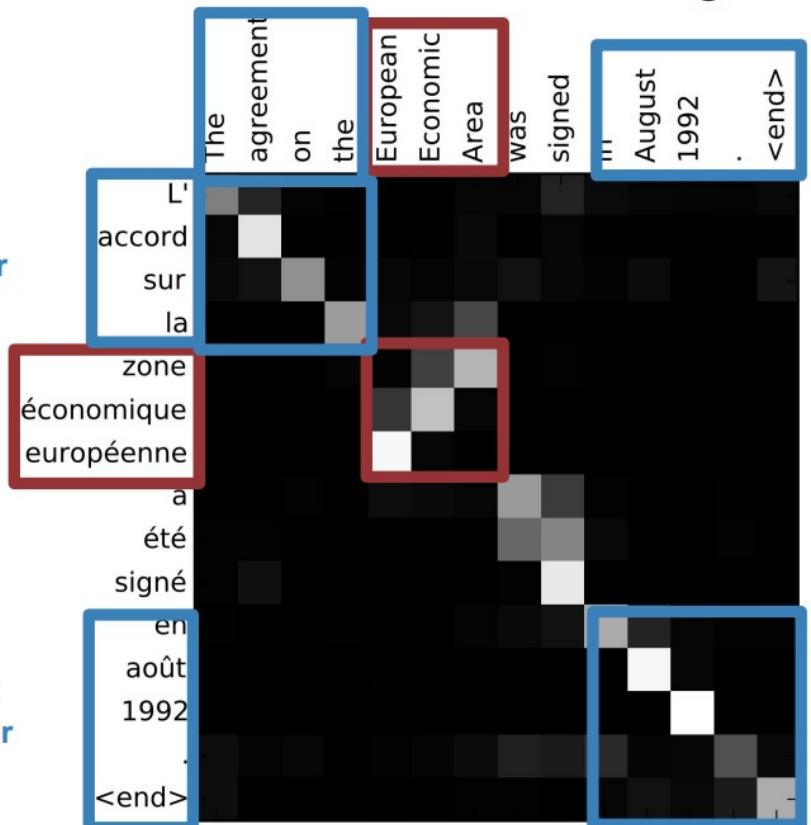
Output: “L'accord sur la zone économique européenne a été signé en août 1992.”

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 $\{h_i\}$!

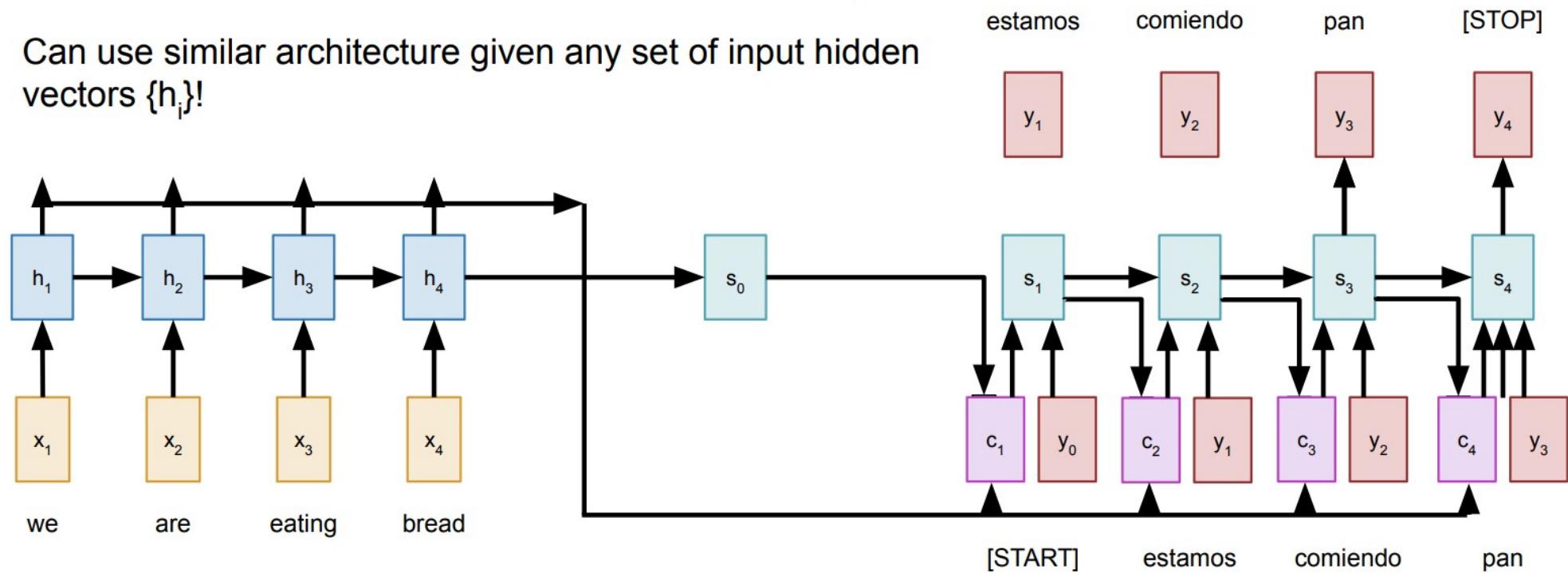
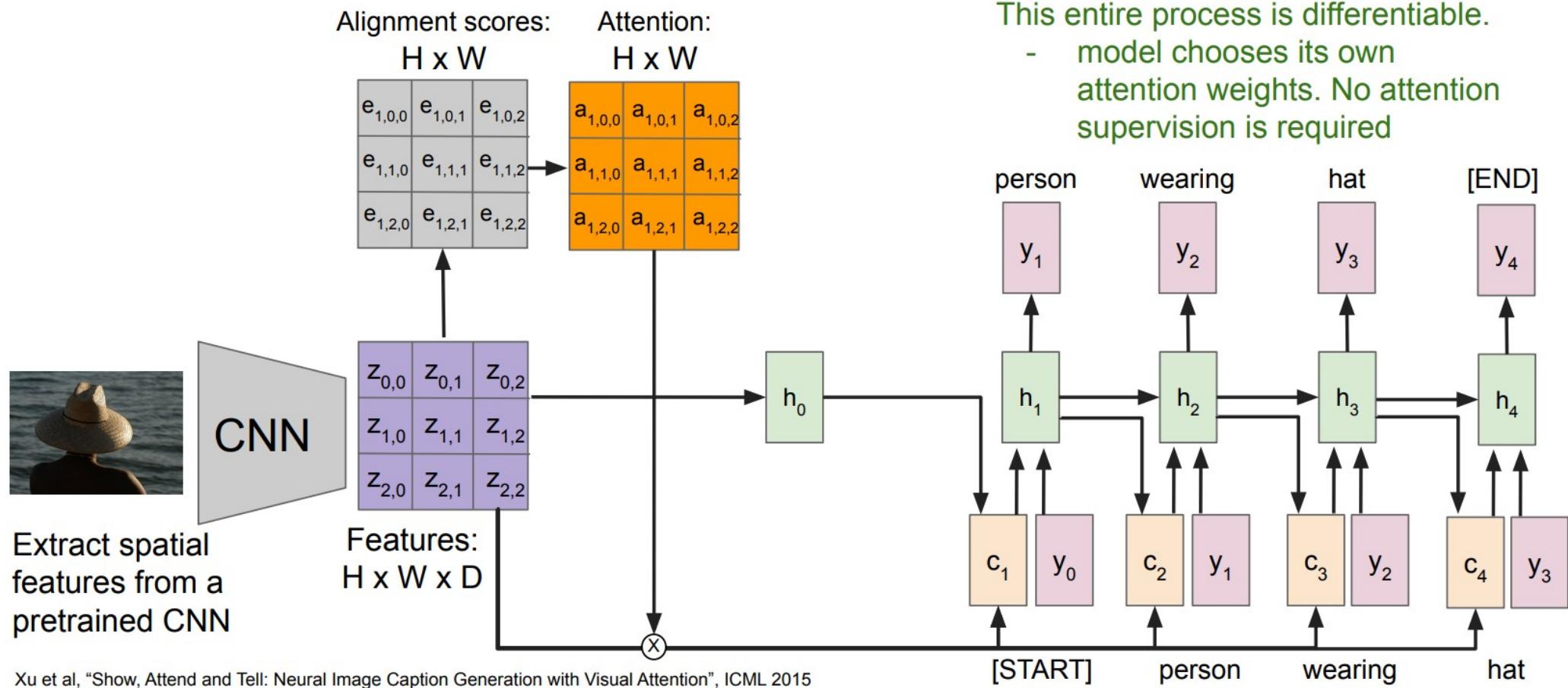


Image Captioning with RNNs and Attention



Attention we just saw in image captioning

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

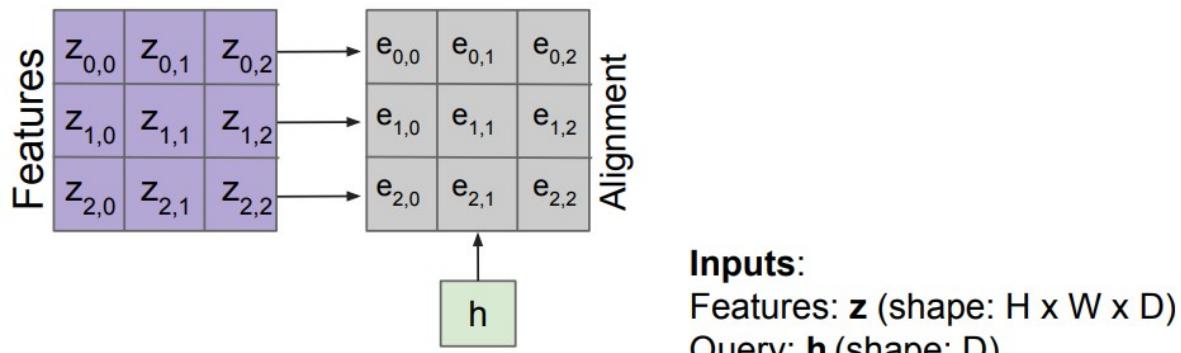
h

Inputs:

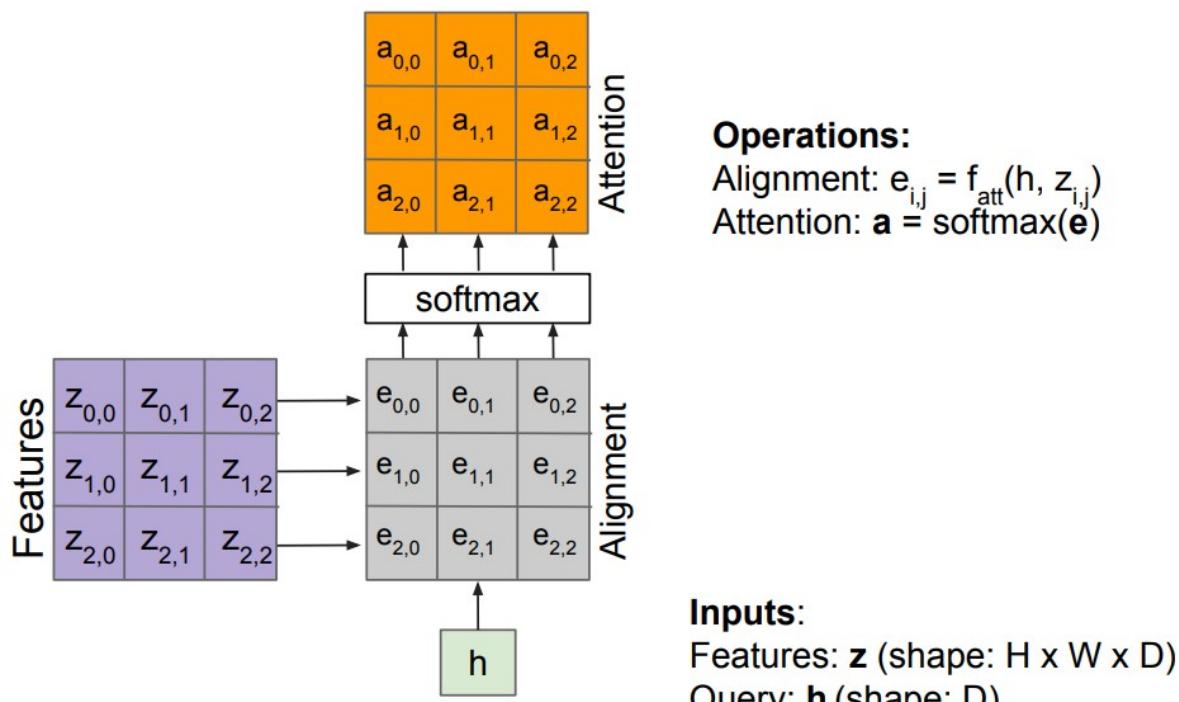
Features: \mathbf{z} (shape: $H \times W \times D$)

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

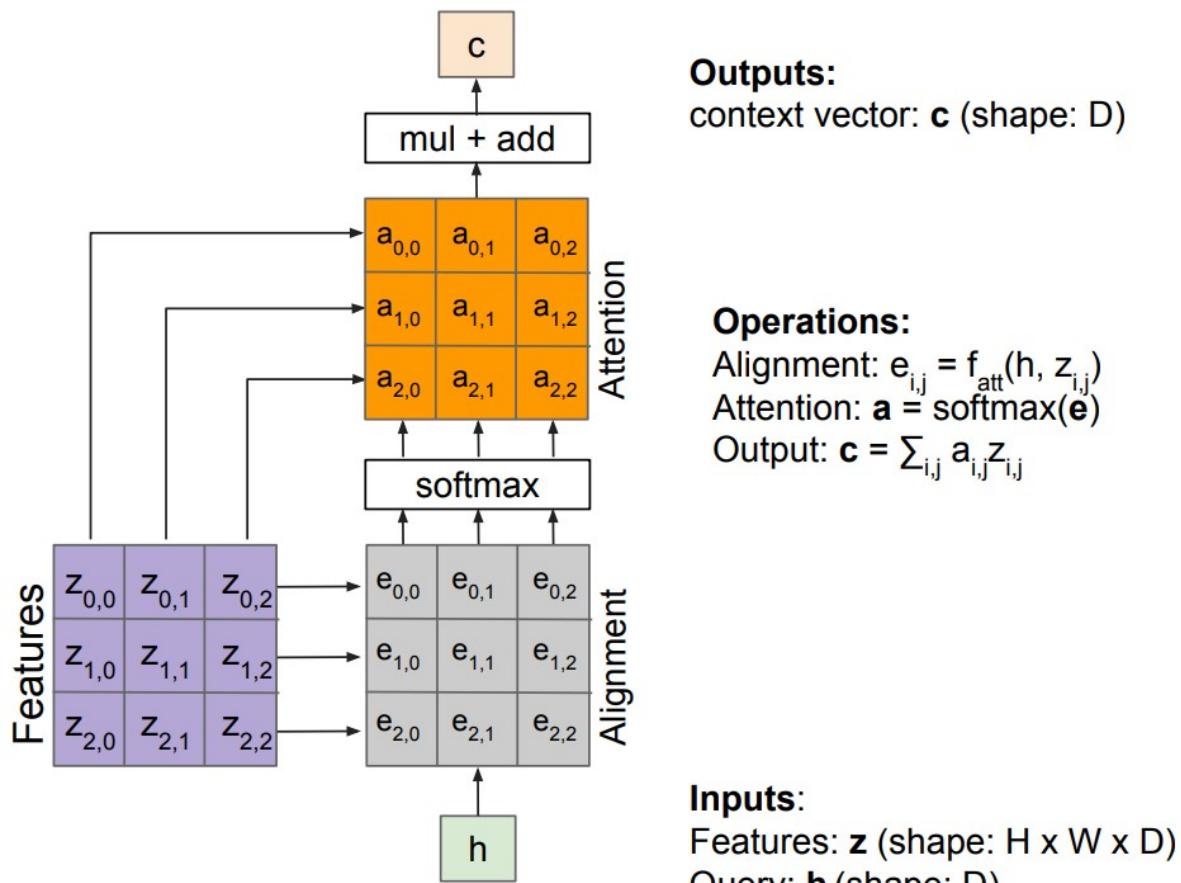
Attention we just saw in image captioning



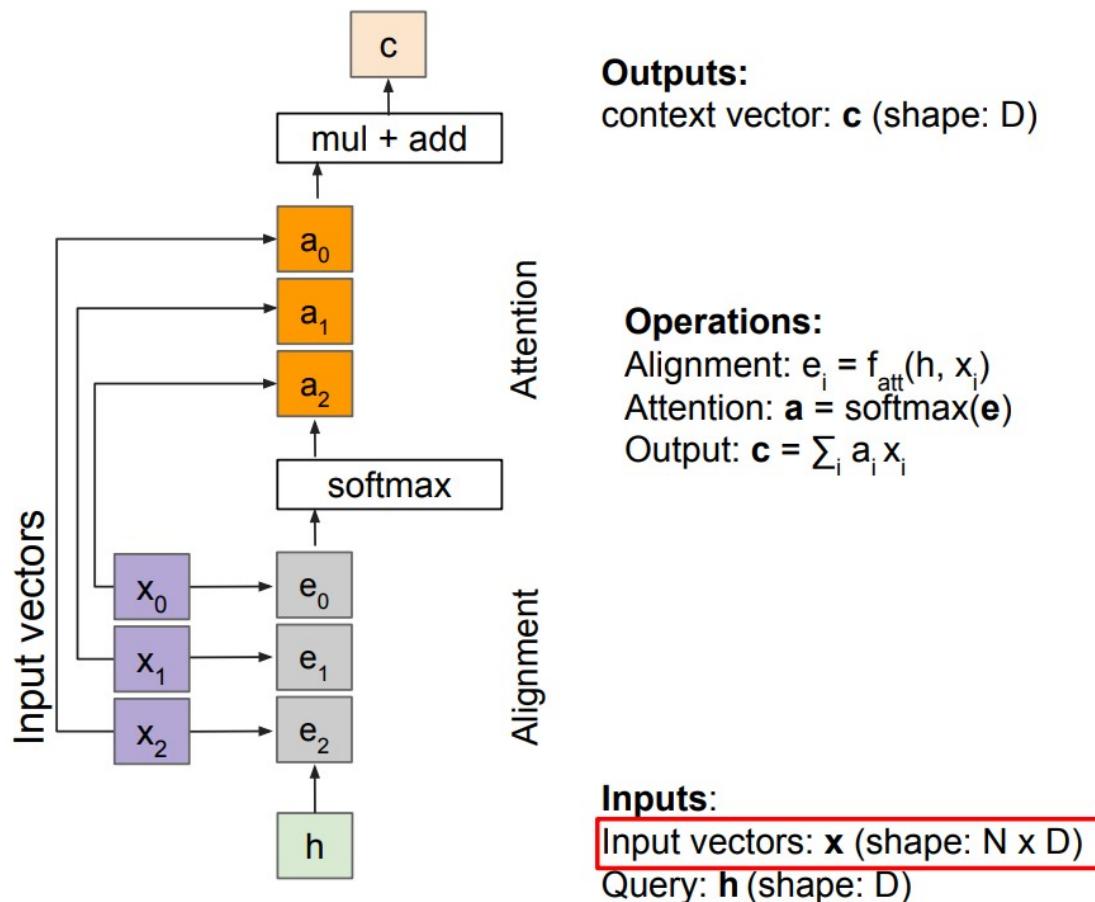
Attention we just saw in image captioning



Attention we just saw in image captioning



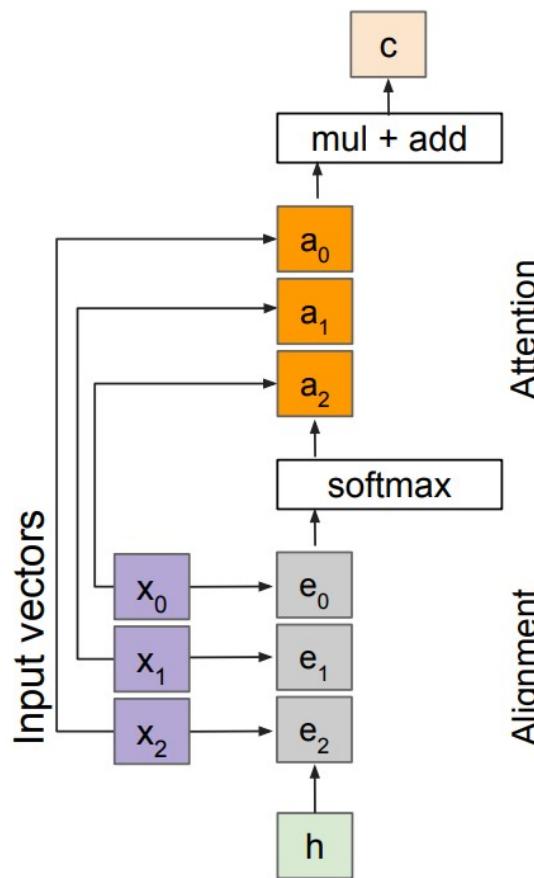
General attention layer



Attention operation is **permutation invariant**.

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

General attention layer



Outputs:

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

Operations:

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

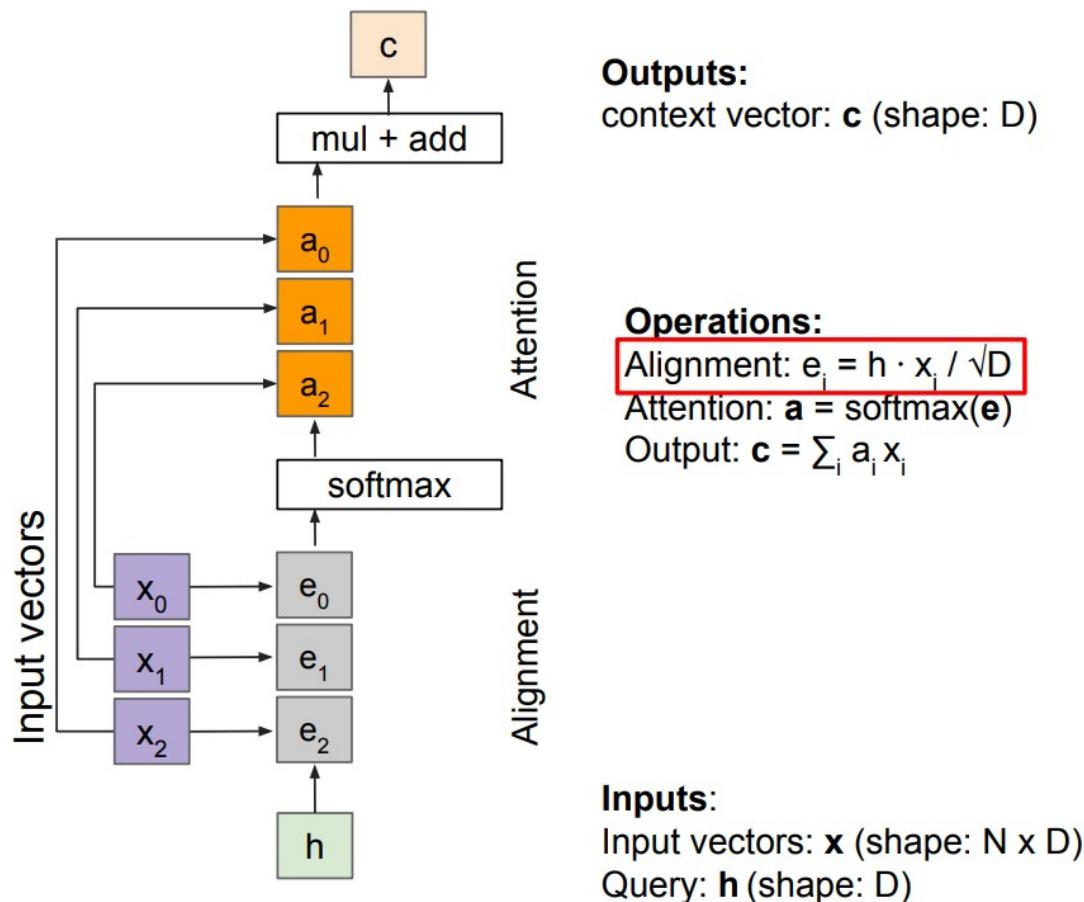
Change $f_{\text{att}}(\cdot)$ to a simple dot product

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

Inputs:

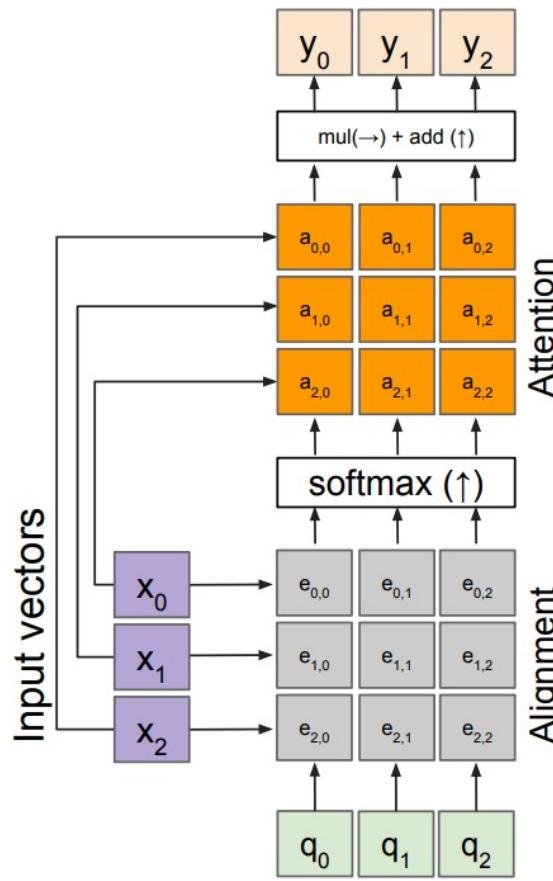
Input vectors: \mathbf{x} (shape: $N \times D$)
Query: \mathbf{h} (shape: D)

General attention layer



- Change $f_{\text{att}}(\cdot)$ 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)

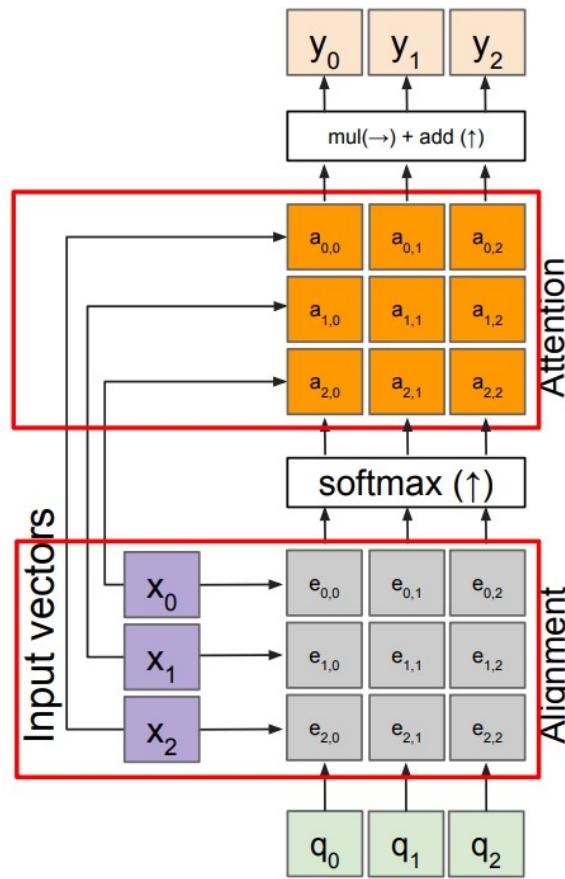
Multiple query vectors
- each query creates a new output context vector

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

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

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

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

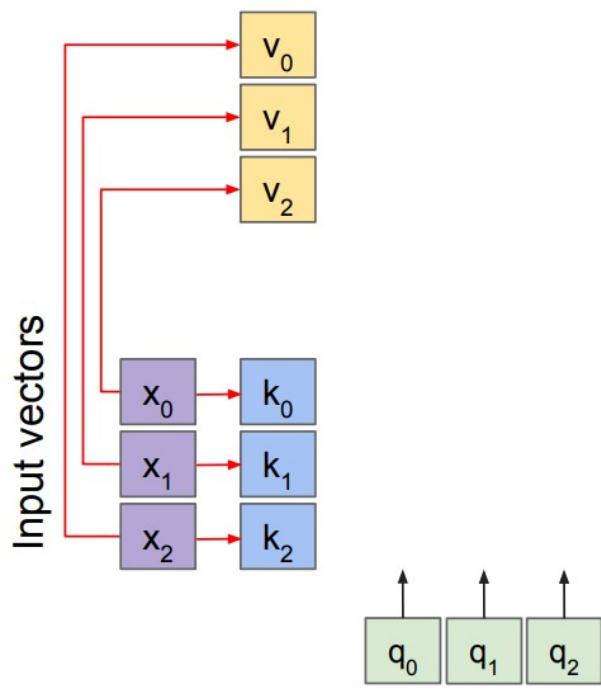
- We can add more expressivity to the layer by adding a different FC layer before each of the two steps.

Inputs:

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

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

General attention layer



Operations:

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

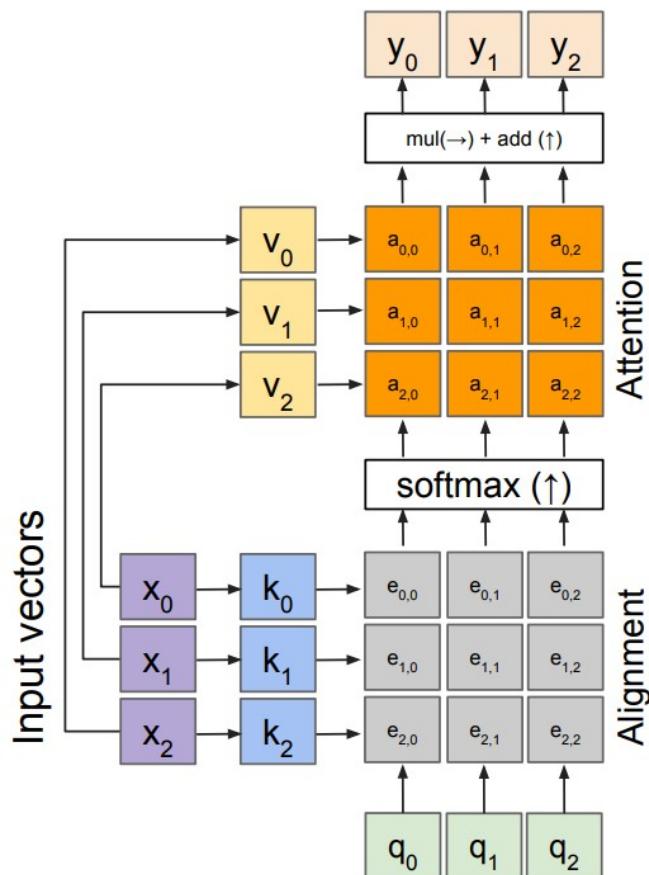
Inputs:

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

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: \mathbf{y} (shape: D_v)

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

Operations:

Key vectors: $\mathbf{k} = \mathbf{x}W_k$

Value vectors: $\mathbf{v} = \mathbf{x}W_v$

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

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

Output: $y_j = \sum_i a_{i,j} \mathbf{v}_i$

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

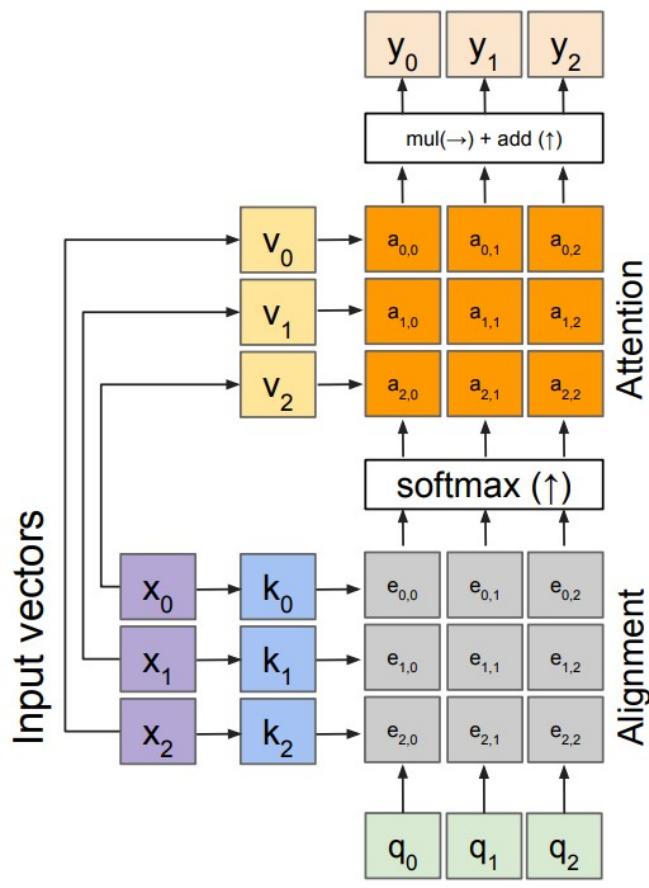
- We can add more expressivity to the layer by adding a different FC layer before each of the two steps.

Inputs:

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

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

General attention layer



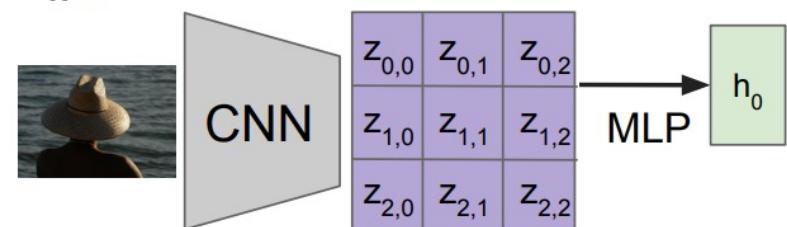
Outputs:
context vectors: \mathbf{y} (shape: D_v)

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

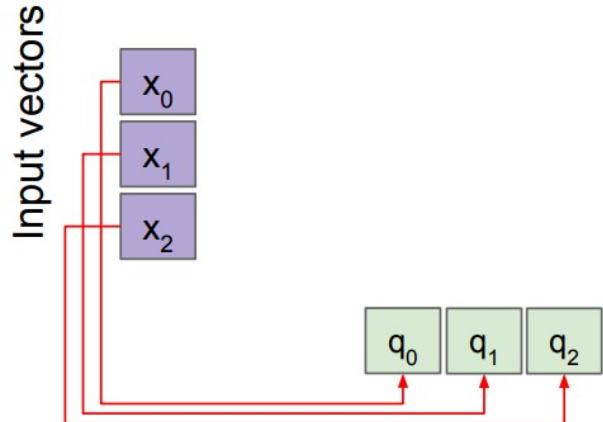
Inputs:
Input vectors: \mathbf{x} (shape: $N \times D$)
Queries: \mathbf{q} (shape: $M \times D_k$)

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

Encoder: $\mathbf{h}_0 = f_{\mathbf{w}}(\mathbf{z})$
where \mathbf{z} is spatial CNN features
 $f_{\mathbf{w}}(\cdot)$ is an MLP



Self attention layer



Operations:

Key vectors: $\mathbf{k} = \mathbf{x}W_k$

Value vectors: $\mathbf{v} = \mathbf{x}W_v$

Query vectors: $\mathbf{q} = \mathbf{x}W_q$

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

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

Output: $\mathbf{y}_j = \sum_i a_{i,j} \mathbf{v}_i$

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.

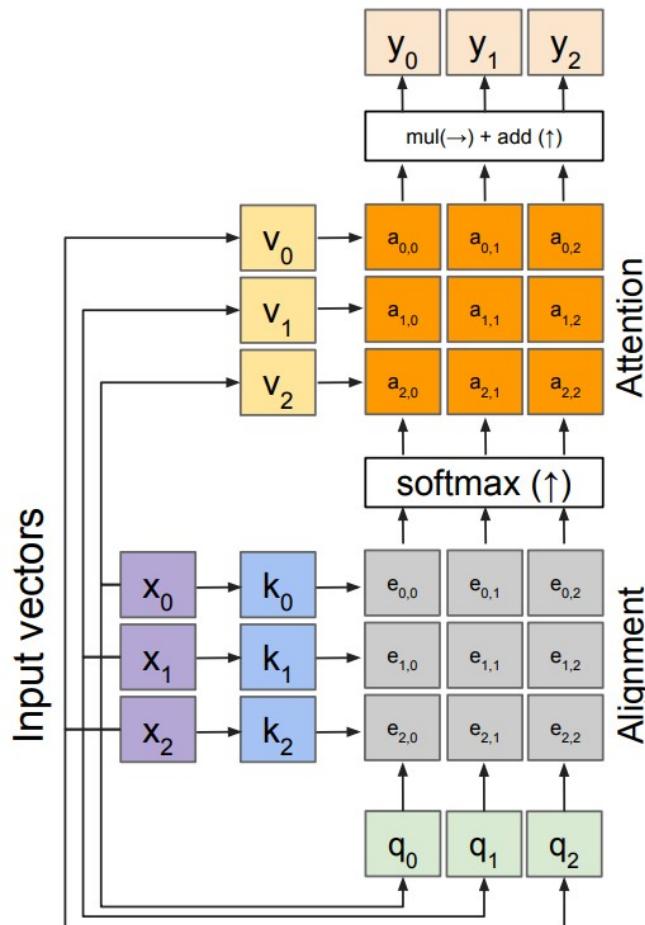
No input query vectors anymore

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: $\mathbf{k} = \mathbf{x}W_k$

Value vectors: $\mathbf{v} = \mathbf{x}W_v$

Query vectors: $\mathbf{q} = \mathbf{x}W_q$

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

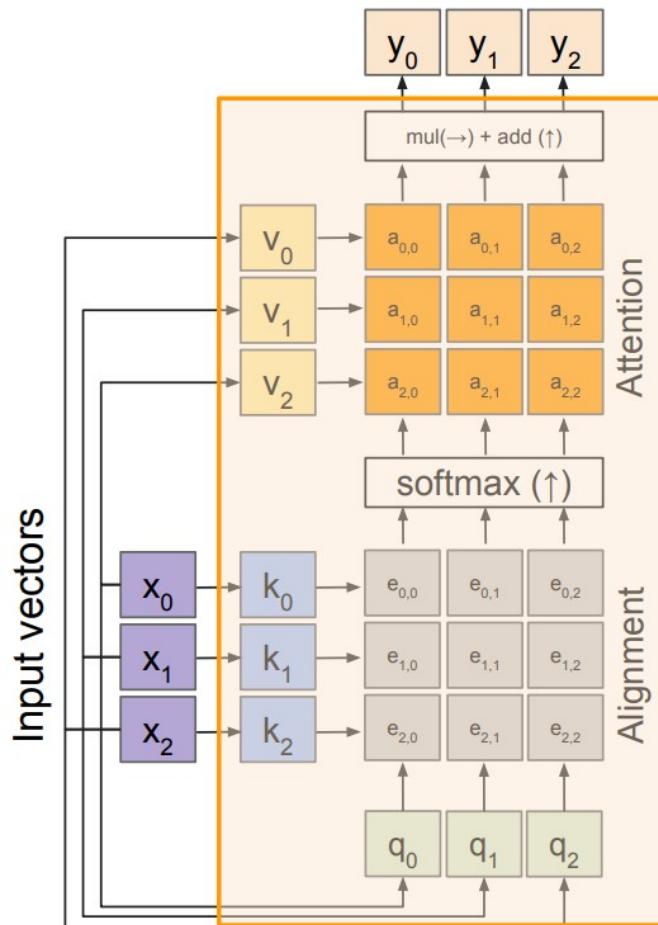
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$

Output: $\mathbf{y}_j = \sum_i \mathbf{a}_{i,j} \mathbf{v}_i$

Inputs:

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

Self attention layer - attends over sets of inputs



Outputs:

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

Operations:

Key vectors: $\mathbf{k} = \mathbf{x}W_k$

Value vectors: $\mathbf{v} = \mathbf{x}W_v$

Query vectors: $\mathbf{q} = \mathbf{x}W_q$

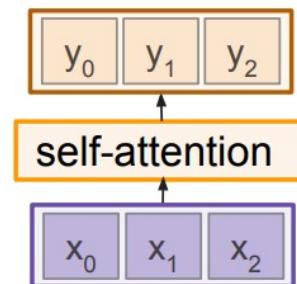
Alignment: $\mathbf{e}_{i,j} = \mathbf{q}_j \cdot \mathbf{k}_i / \sqrt{D}$

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

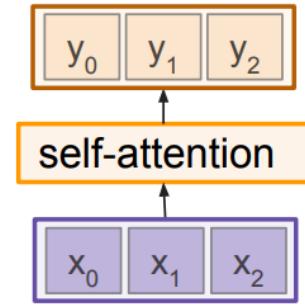
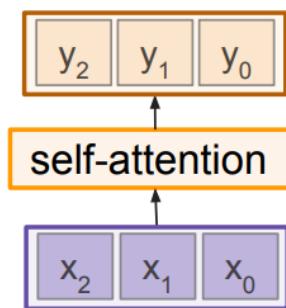
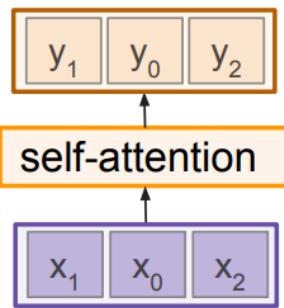
Output: $\mathbf{y}_j = \sum_i \mathbf{a}_{i,j} \mathbf{v}_i$

Inputs:

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



Self attention layer - attends over sets of inputs

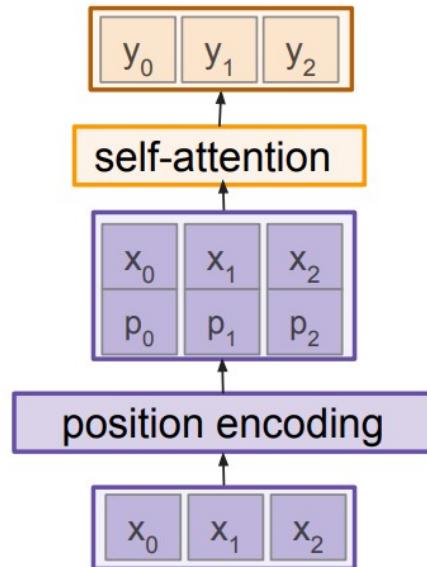


Permutation equivariant

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

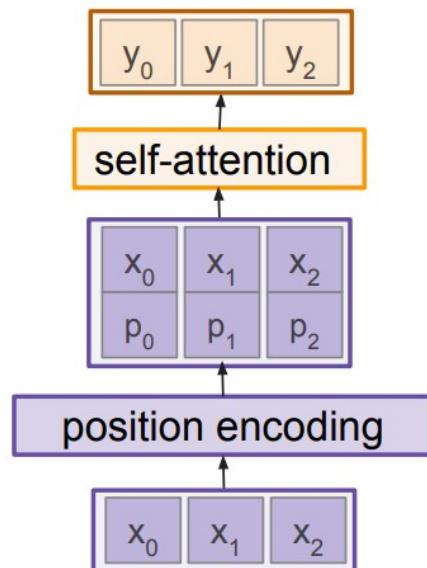
We use a function $pos: N \rightarrow \mathbb{R}^d$ to process the position j of the vector into a d -dimensional vector

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

So, $p_j = pos(j)$

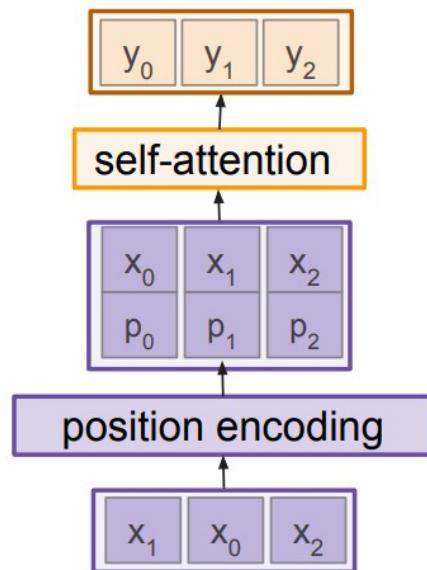
Options for $pos(\cdot)$

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

Desiderata of $pos(\cdot)$:

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

Positional encoding



Concatenate special positional encoding p_j to each input vector x_j

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

So, $p_j = pos(j)$

Options for $pos(\cdot)$

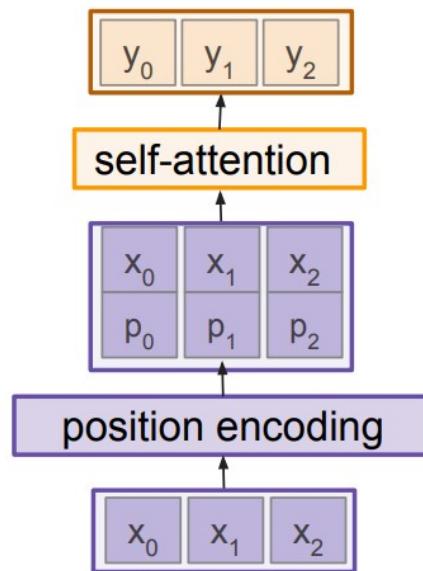
1. Learn a lookup table:
 - o Learn parameters to use for $pos(t)$ for $t \in [0, T]$
 - o Lookup table contains $T \times d$ parameters.
2. Design a fixed function with the desiderata

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

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

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

Positional encoding



Concatenate special positional encoding p_j to each input vector x_j

We use a function $pos: N \rightarrow \mathbb{R}^d$ 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:
 - o Learn parameters to use for $pos(t)$ for $t \in [0, T]$
 - o Lookup table contains $T \times d$ parameters.
2. Design a fixed function with the desiderata

Intuition:

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

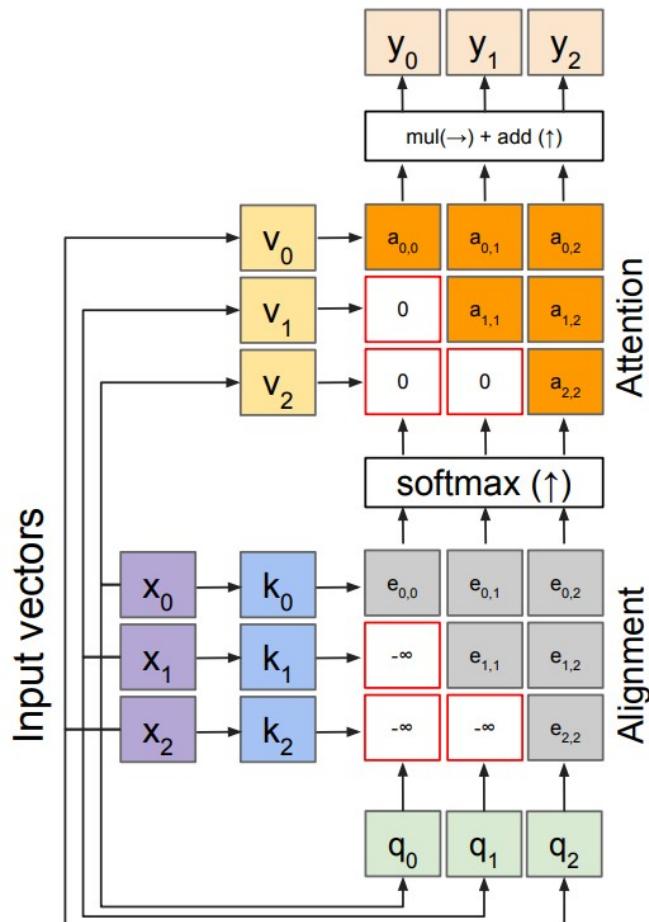
0 :	0 0 0 0	8 :	1 0 0 0
1 :	0 0 0 1	9 :	1 0 0 1
2 :	0 0 1 0	10 :	1 0 1 0
3 :	0 0 1 1	11 :	1 0 1 1
4 :	0 1 0 0	12 :	1 1 0 0
5 :	0 1 0 1	13 :	1 1 0 1
6 :	0 1 1 0	14 :	1 1 1 0
7 :	0 1 1 1	15 :	1 1 1 1

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

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

[image source](#)

Masked self-attention layer



Outputs:

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

Operations:

Key vectors: $\mathbf{k} = \mathbf{x}W_k$

Value vectors: $\mathbf{v} = \mathbf{x}W_v$

Query vectors: $\mathbf{q} = \mathbf{x}W_q$

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

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

Output: $y_j = \sum_i a_{i,j} \mathbf{v}_i$

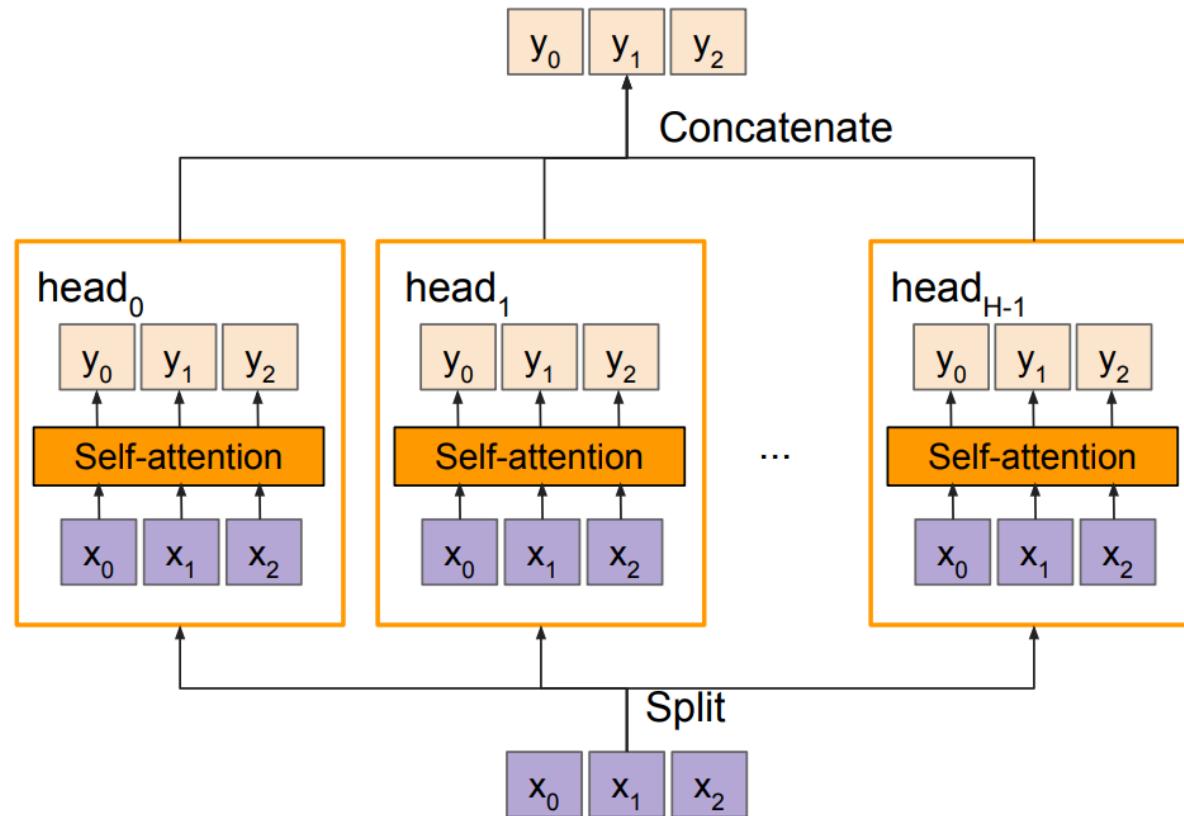
- Prevent vectors from looking at future vectors.
- Manually set alignment scores to -infinity

Inputs:

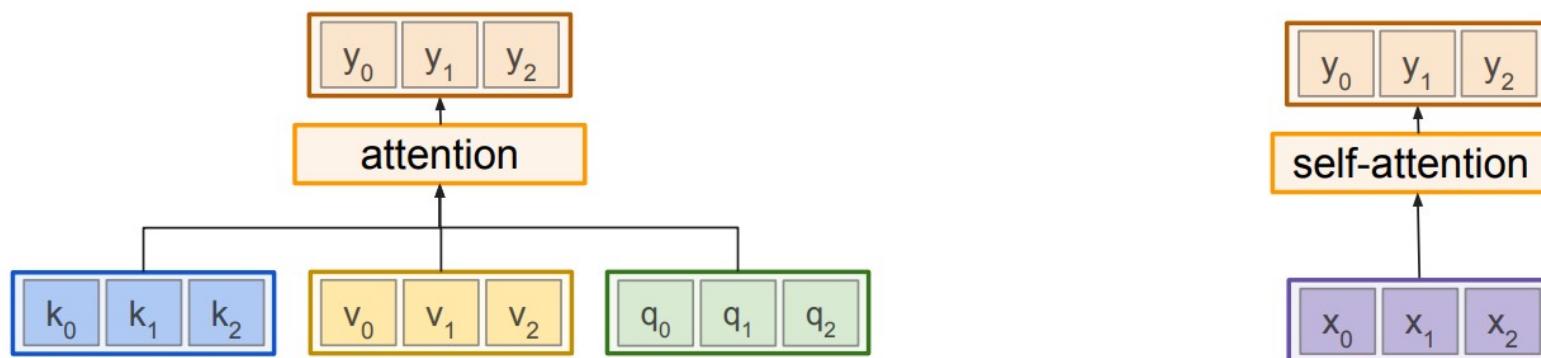
Input vectors: \mathbf{x} (shape: $N \times D$)

Multi-head self attention layer

- Multiple self-attention heads in parallel



General attention versus 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 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

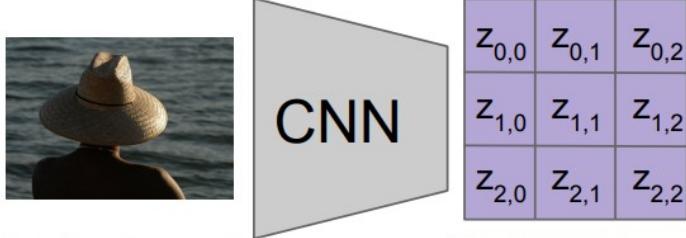
Finetuning:

Fine-tune the Transformer on your own NLP task

Image Captioning using Transformers

Input: Image \mathbf{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

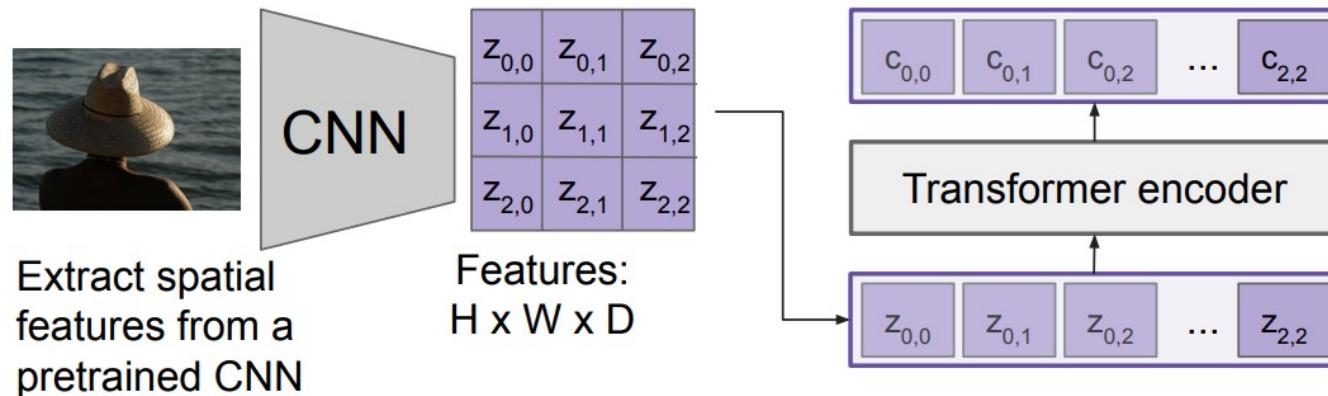


Image Captioning using Transformers

Input: Image \mathbf{I}

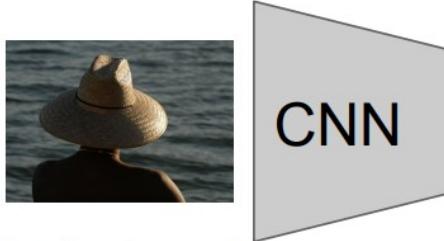
Output: Sequence $\mathbf{y} = y_1, y_2, \dots, y_T$

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

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

where \mathbf{z} is spatial CNN features

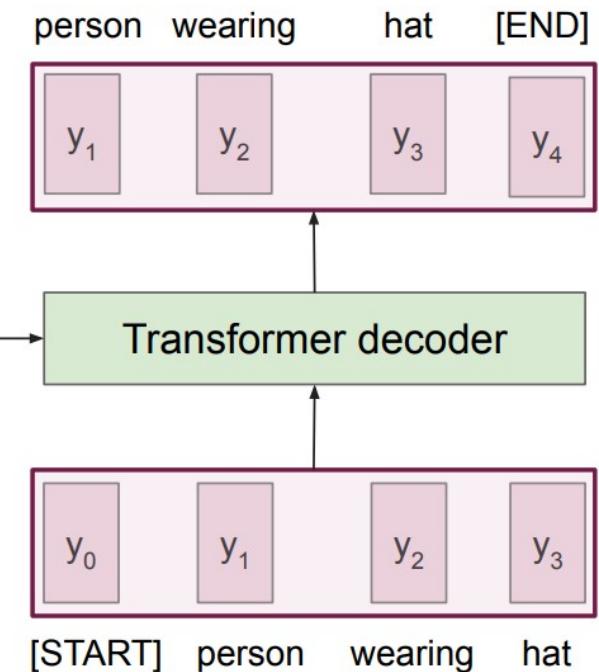
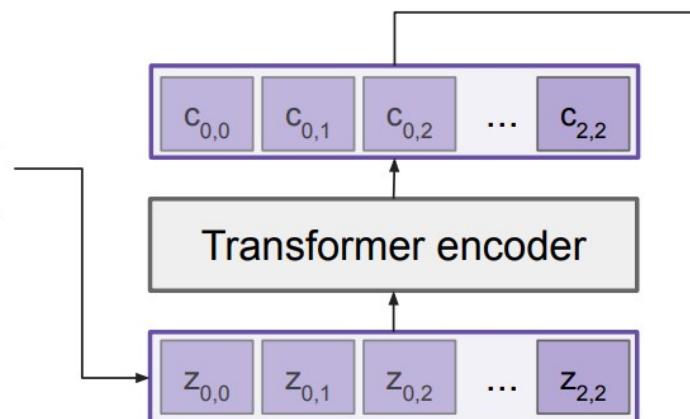
$T_W(\cdot)$ is the transformer encoder



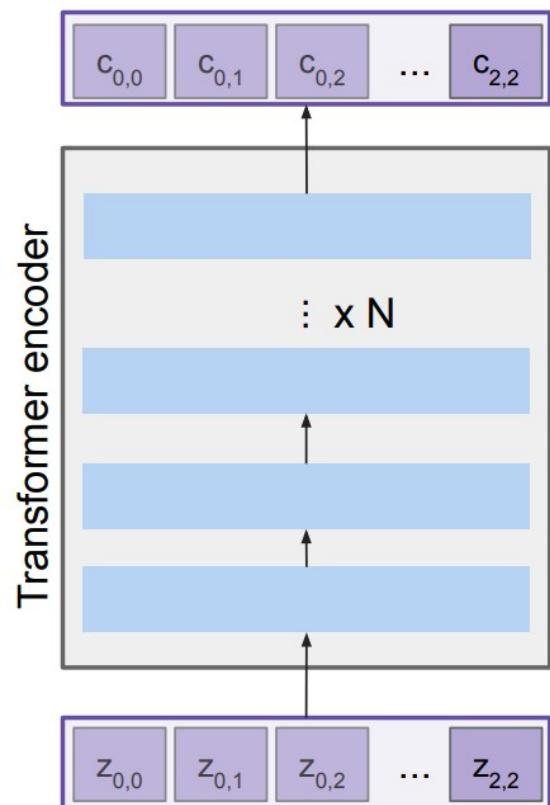
Extract spatial
features from a
pretrained CNN

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

Features:
 $H \times W \times D$



The Transformer encoder block

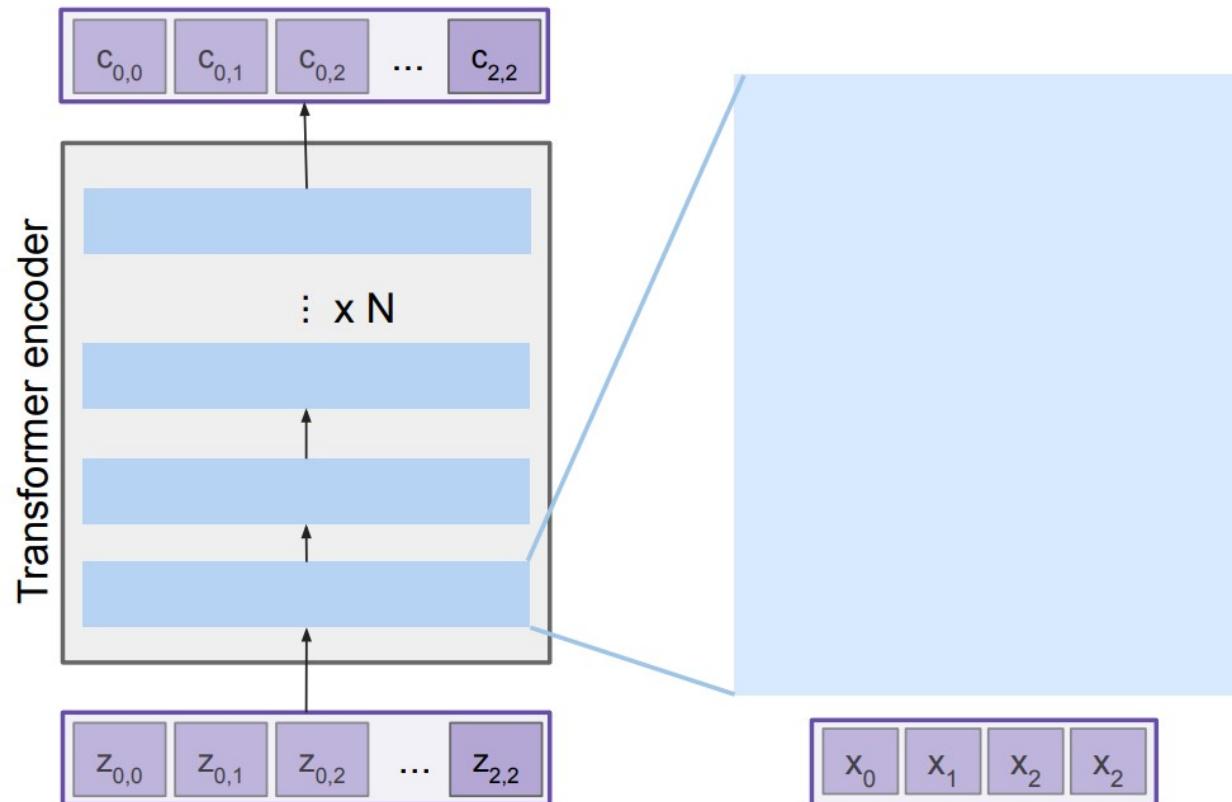


Made up of N encoder blocks.

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

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

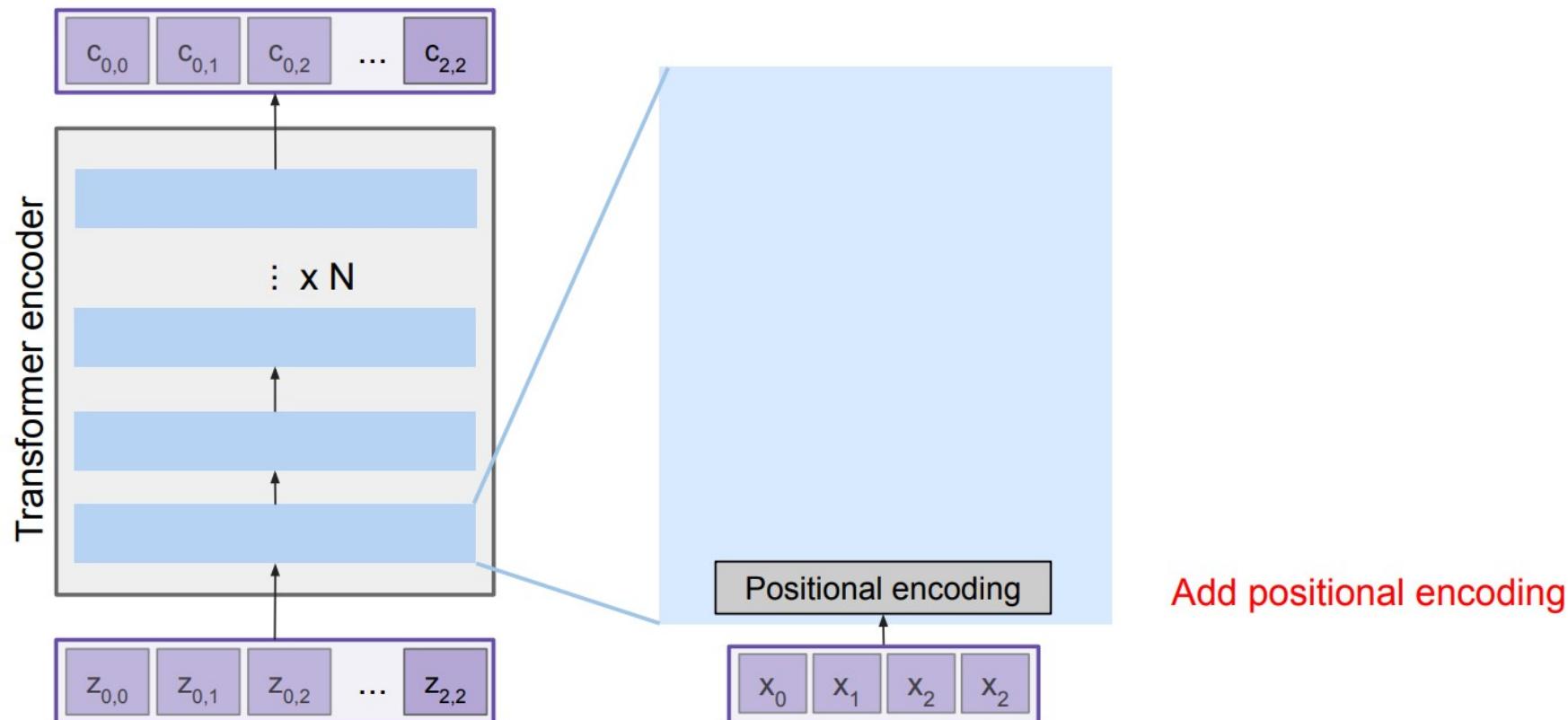
The Transformer encoder block



Let's dive into one encoder block

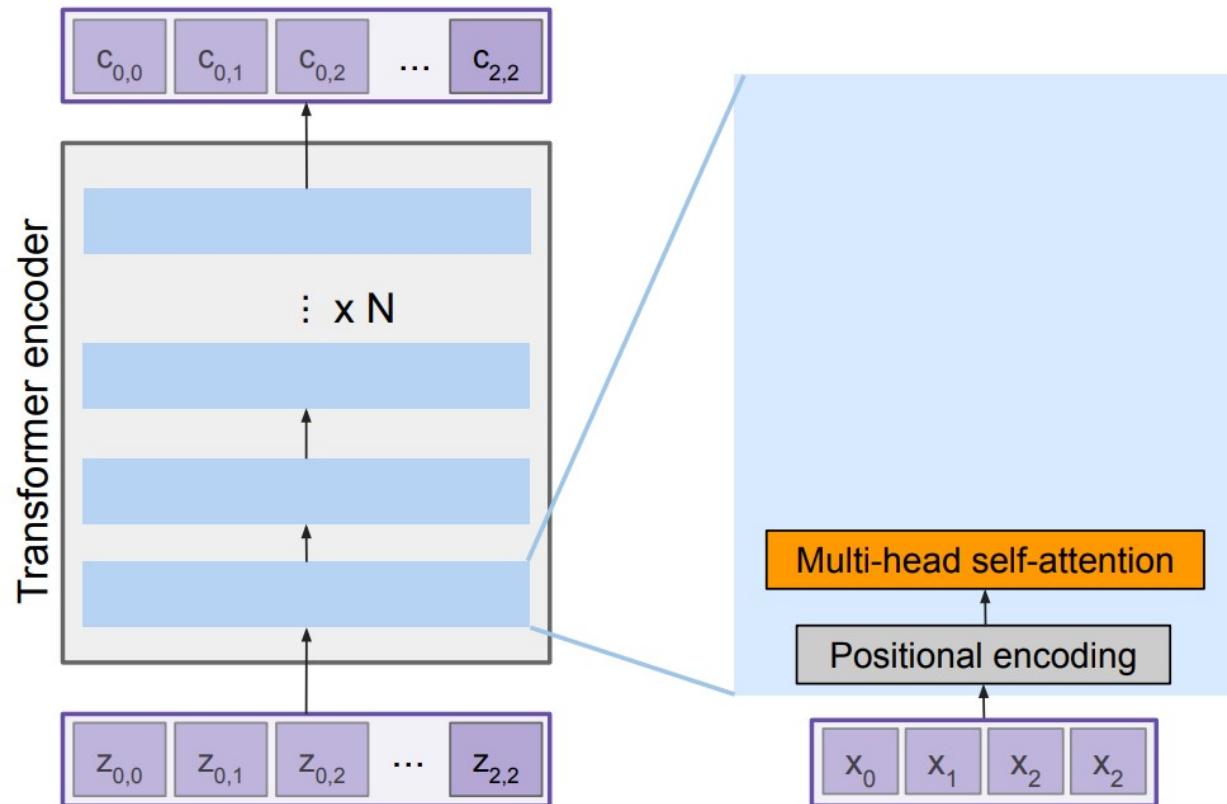
Vaswani et al, "Attention is all you need", NeurIPS 2017

The Transformer encoder block



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

The Transformer encoder block

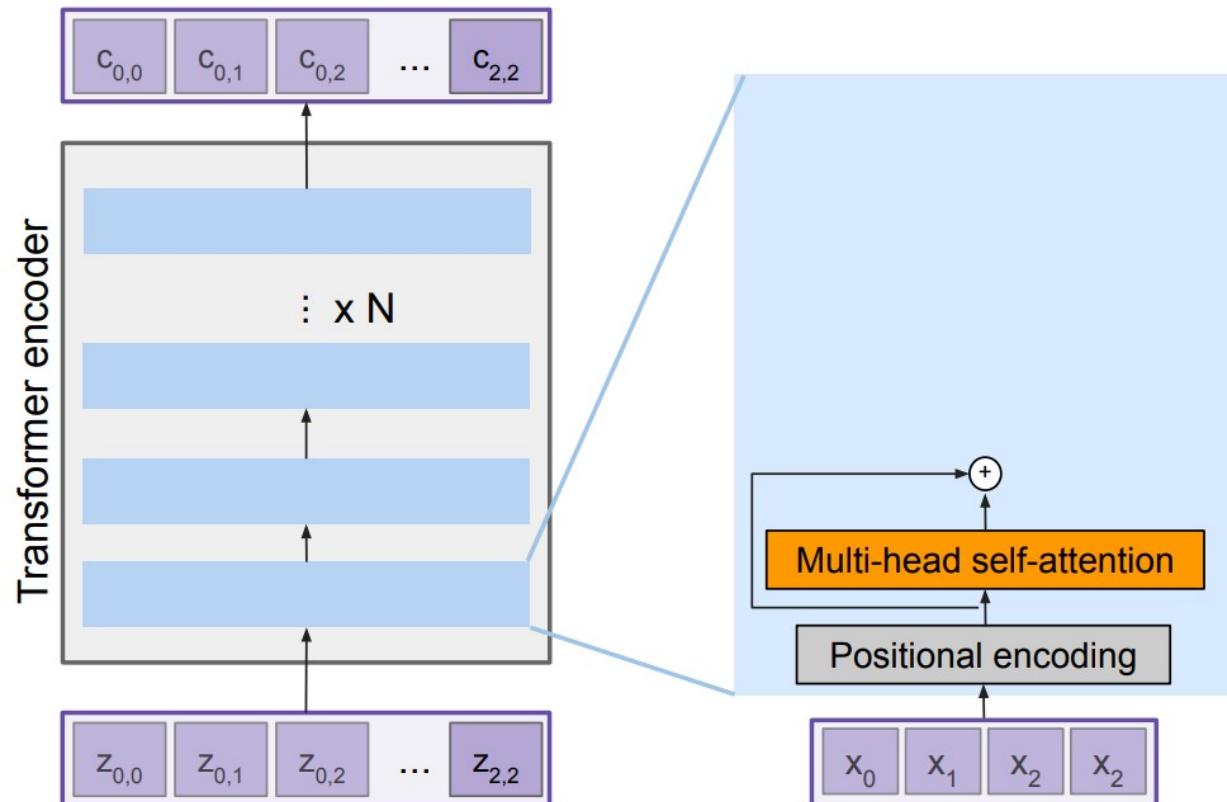


Attention attends over all the vectors

Add positional encoding

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

The Transformer encoder block



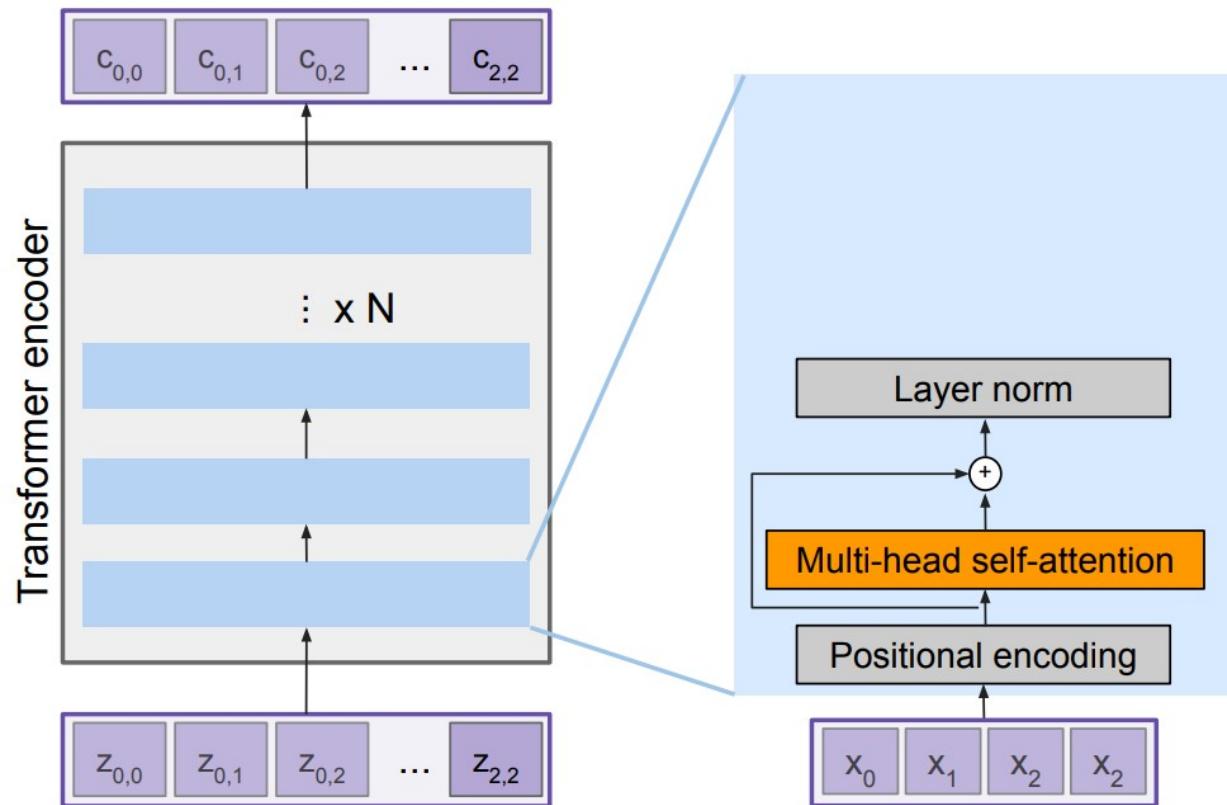
Residual connection

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

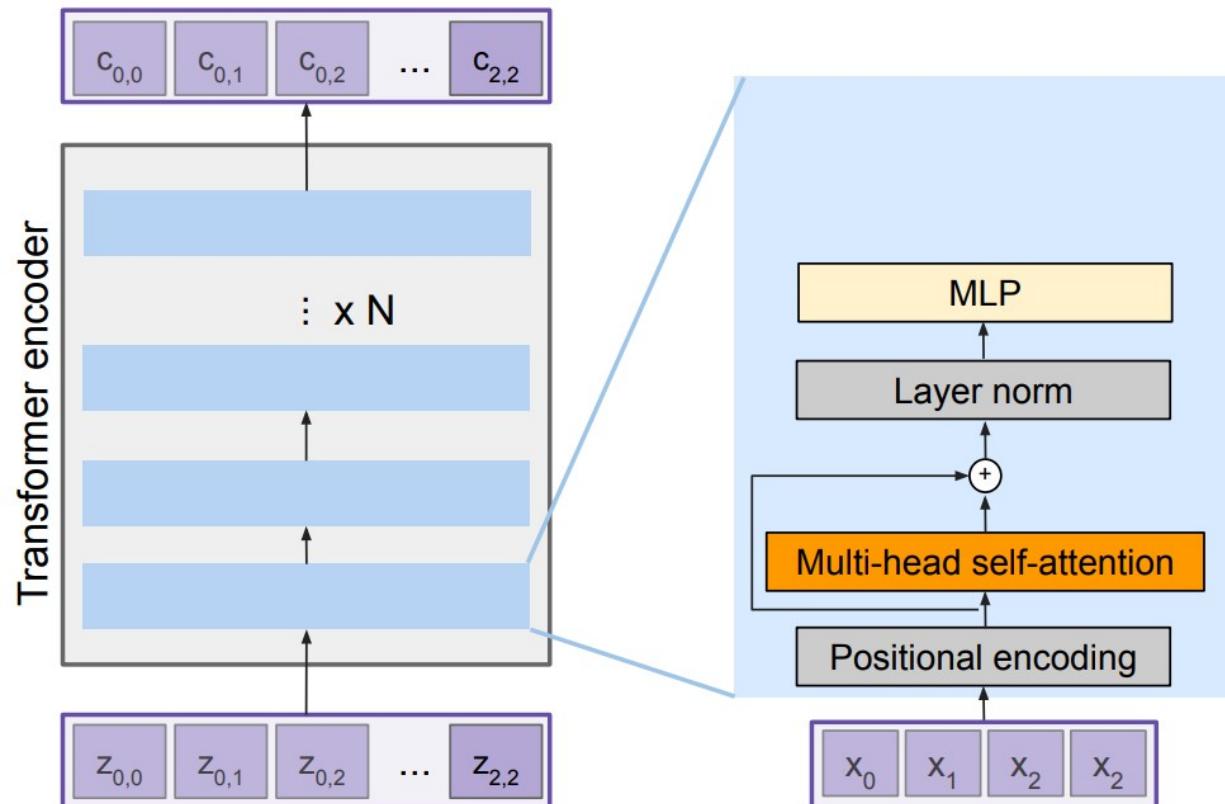
Residual connection

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

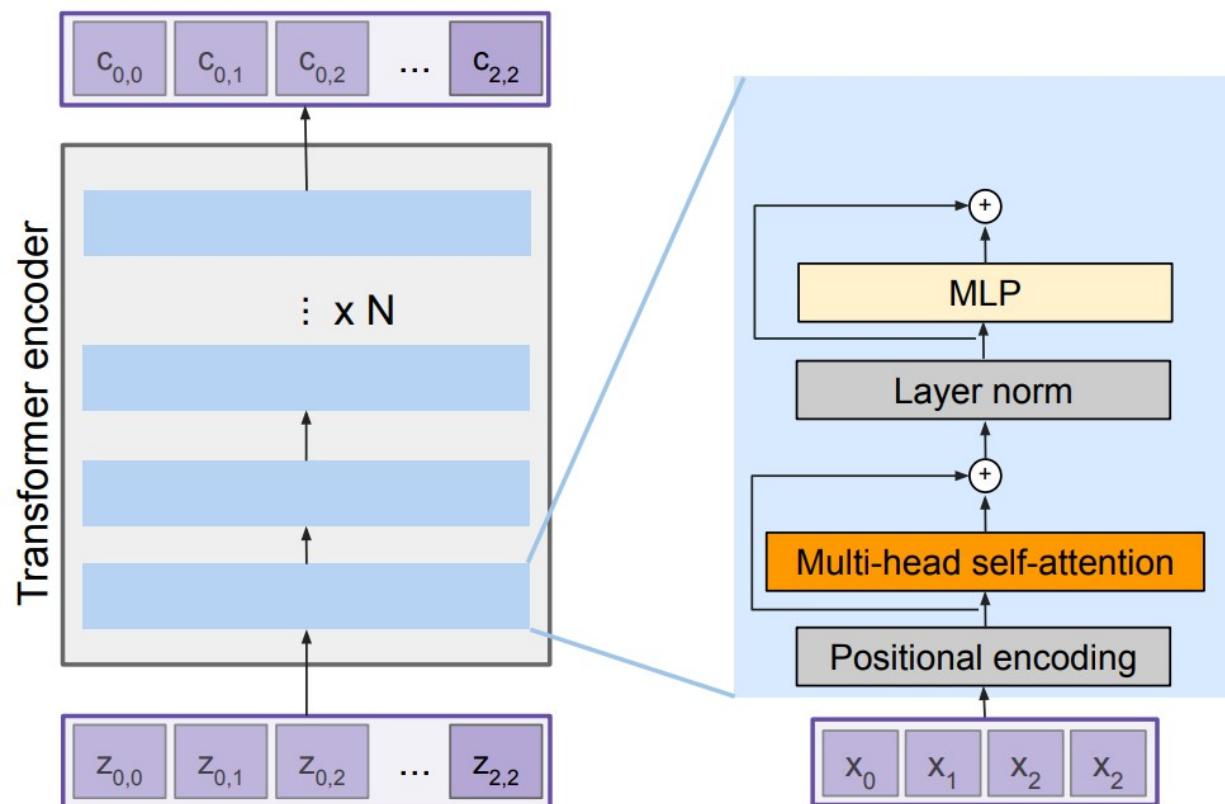
Residual connection

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

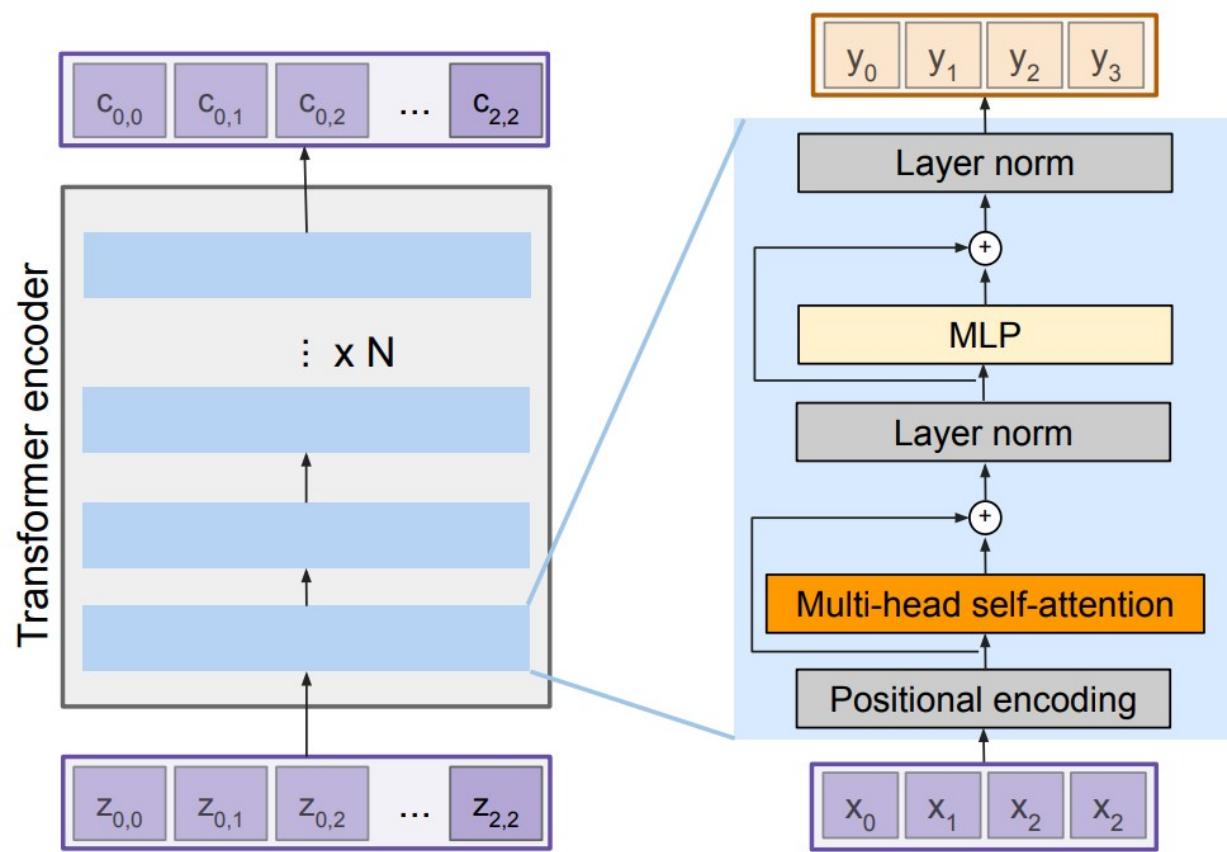
Residual connection

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 \mathbf{x}

Outputs: Set of vectors \mathbf{y}

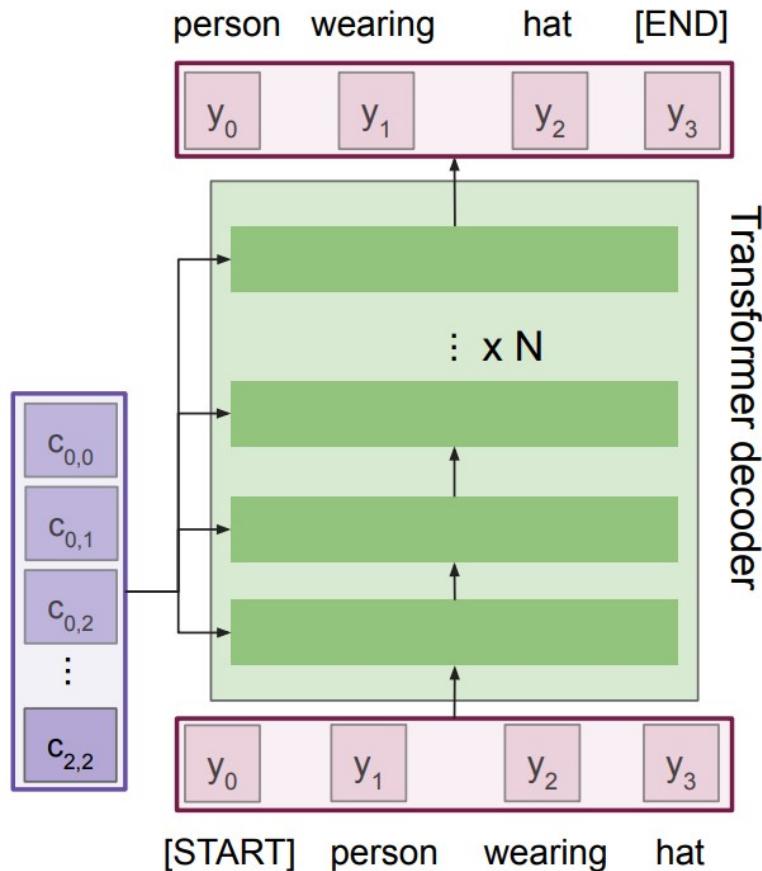
Self-attention is the only interaction between vectors.

Layer norm and MLP operate independently per vector.

Highly scalable, highly parallelizable, but high memory usage.

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

The Transformer decoder block

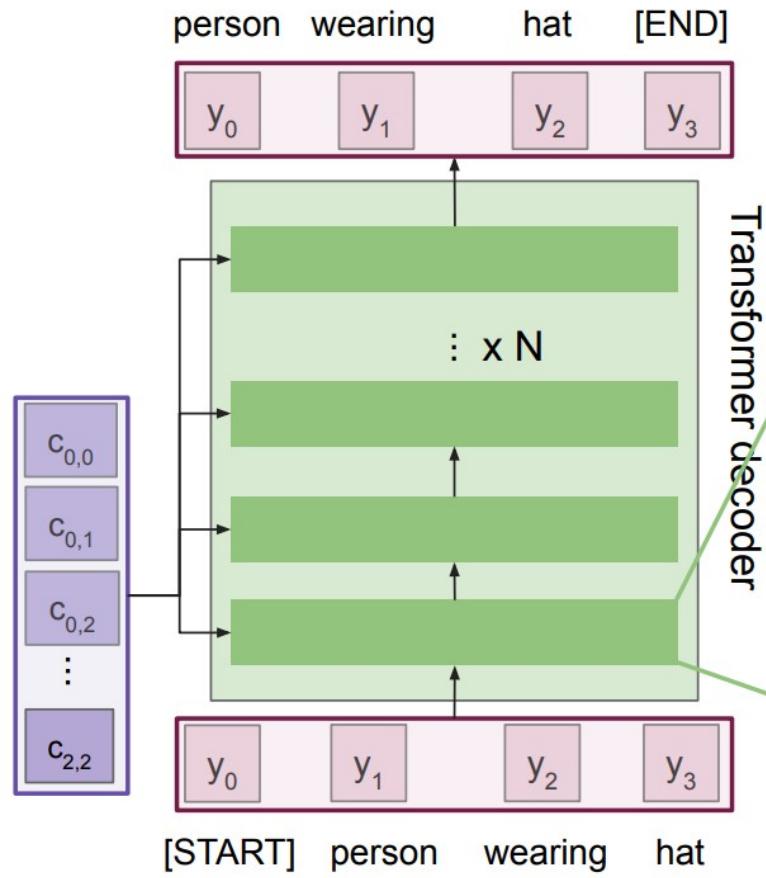


Made up of N decoder blocks.

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

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

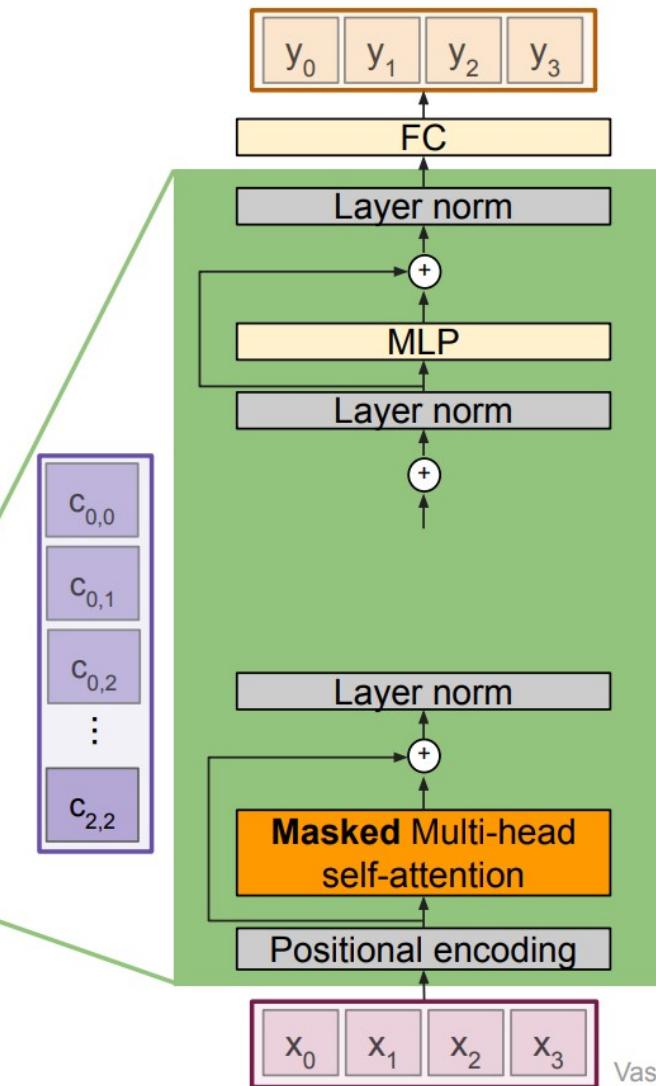
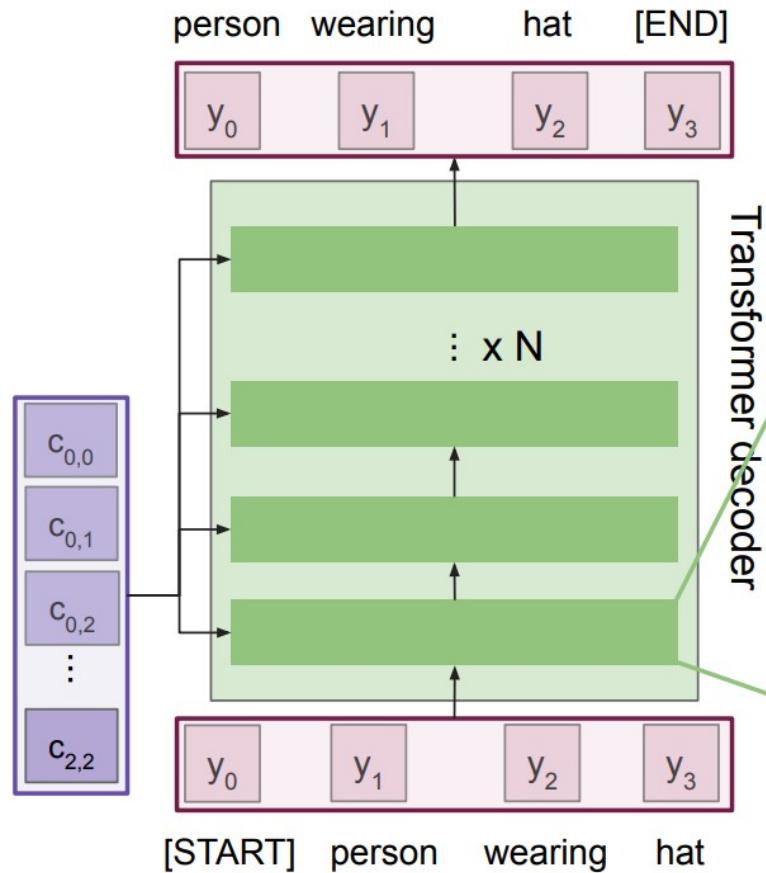
The Transformer decoder block



Let's dive into the
transformer decoder block

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

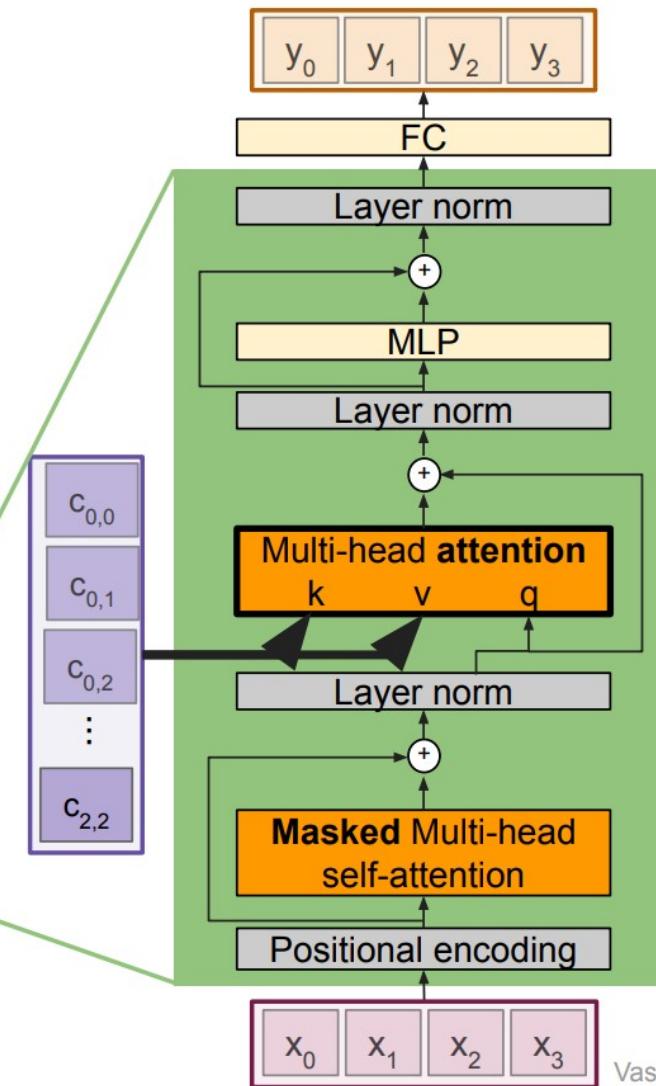
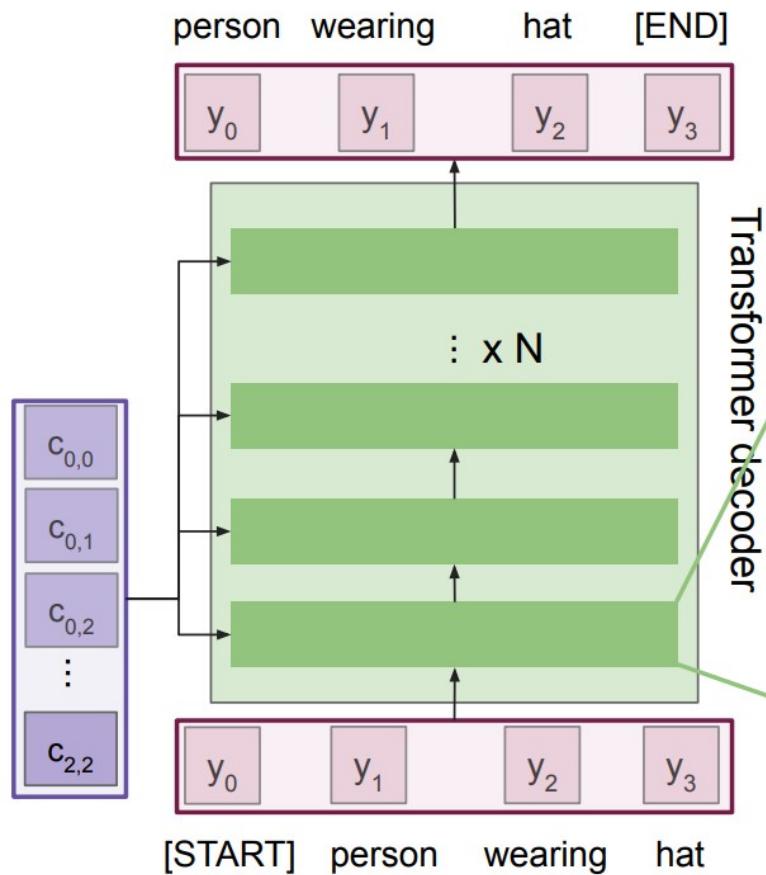
The Transformer Decoder block



Most of the network is the same the transformer encoder.

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

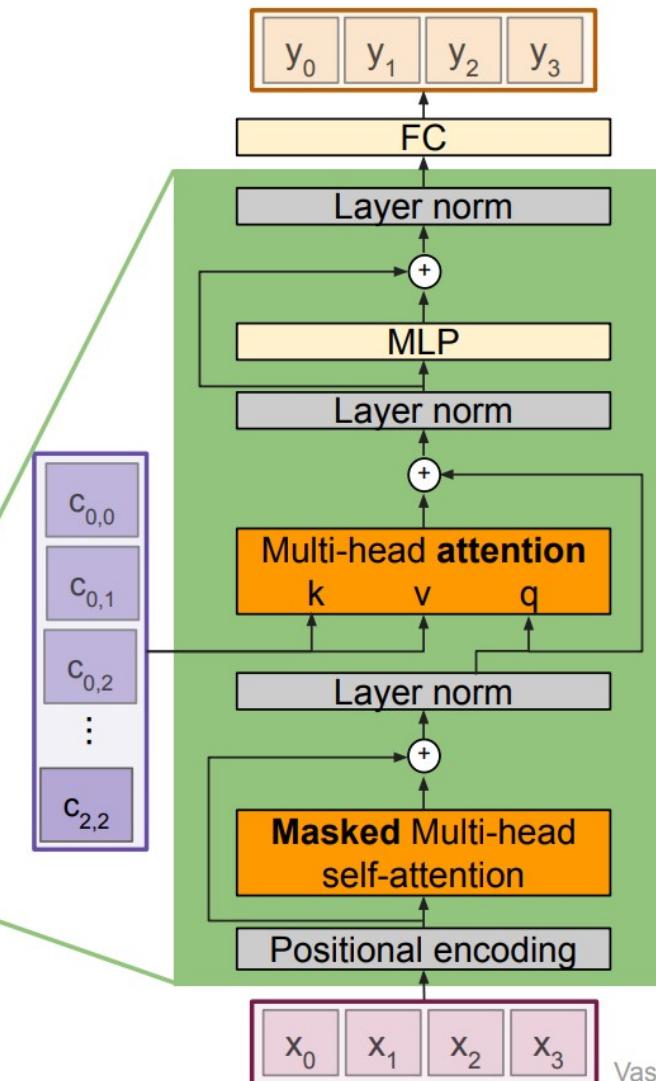
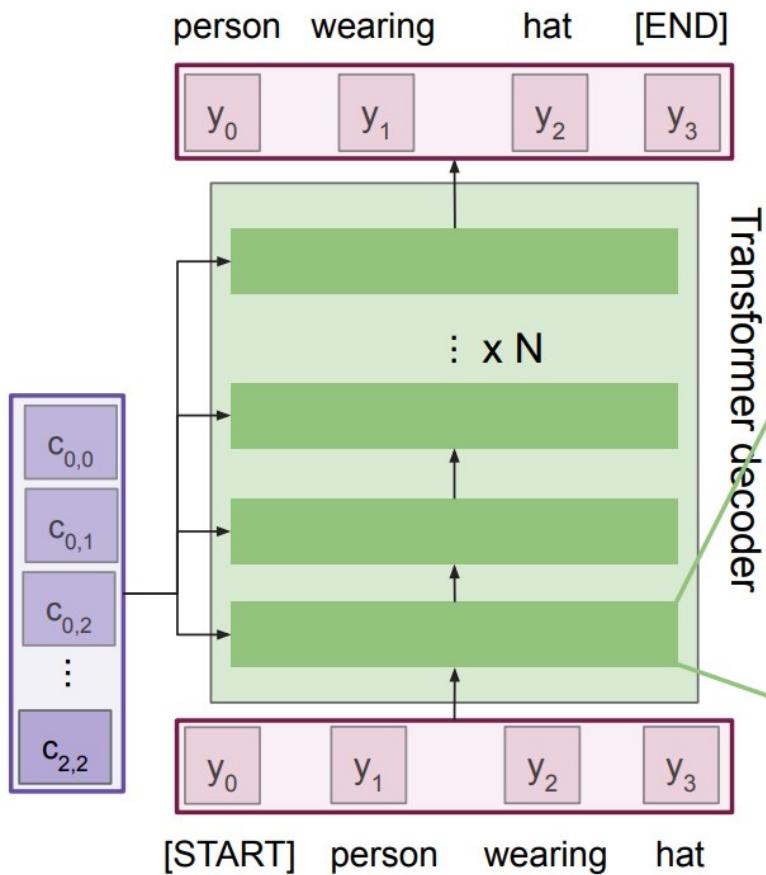
The Transformer Decoder block



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

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

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

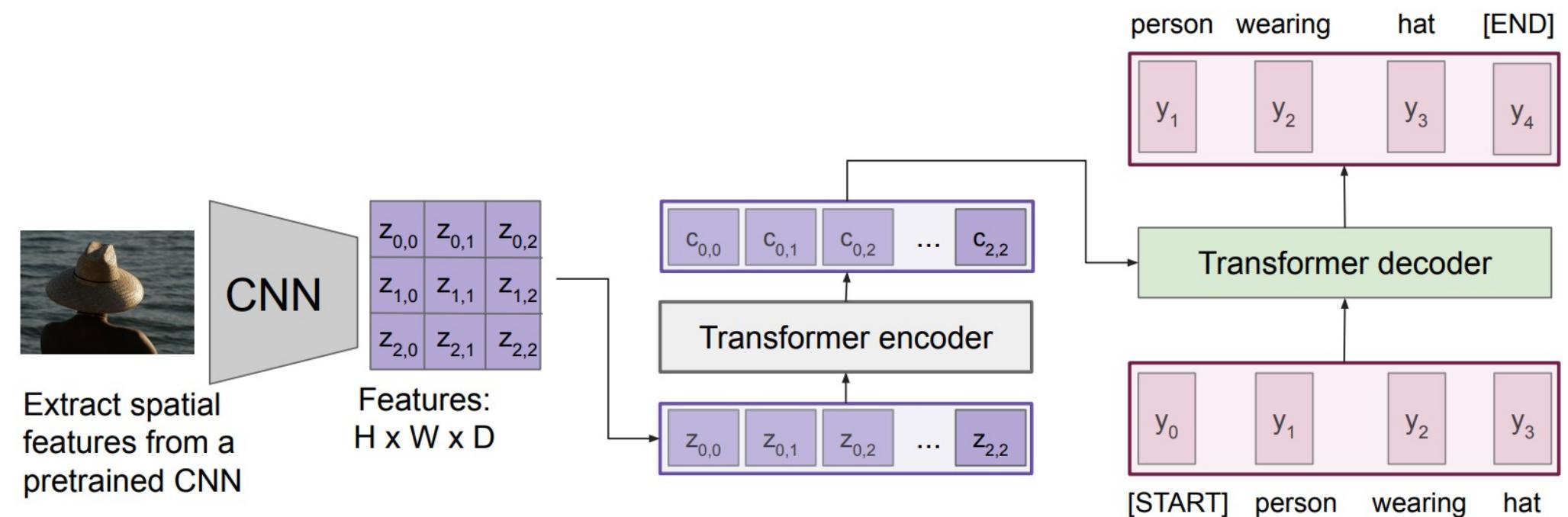


Image Captioning using transformers

- Perhaps we don't need convolutions at all?

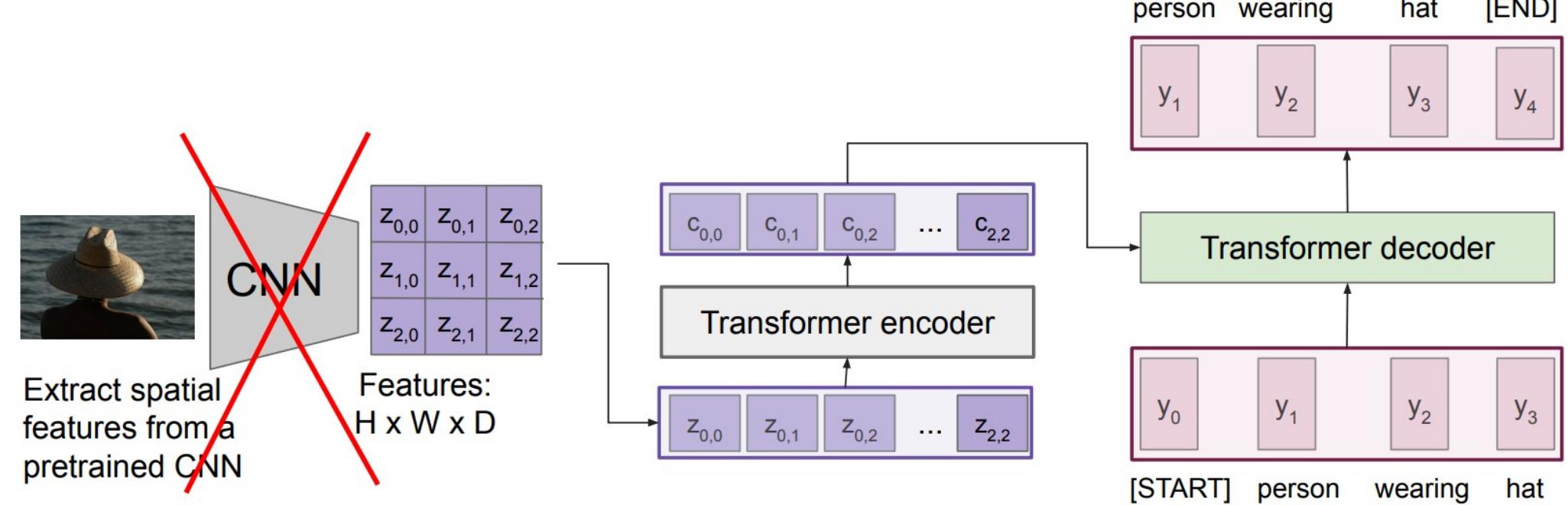
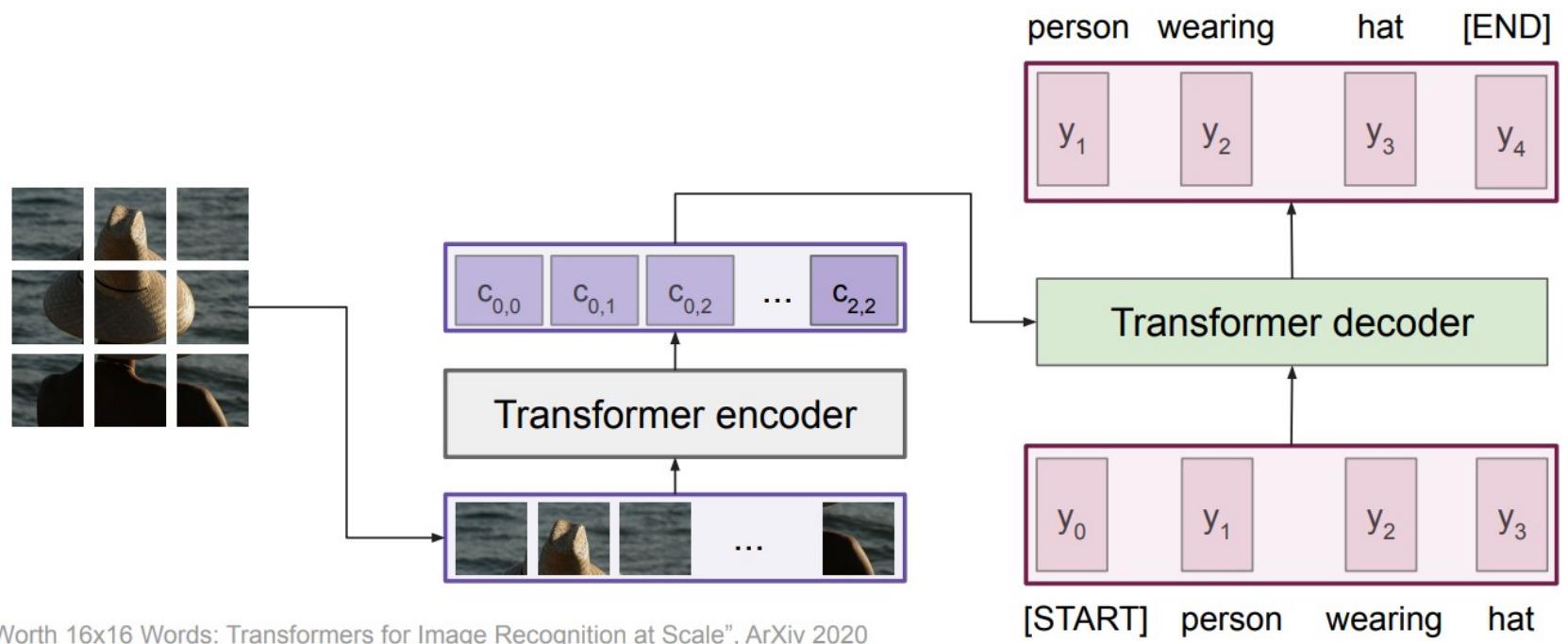


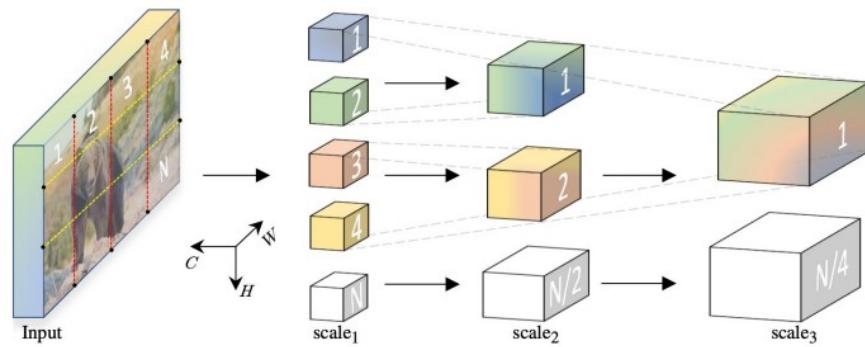
Image Captioning using **ONLY** transformers

- Transformers from pixels to language

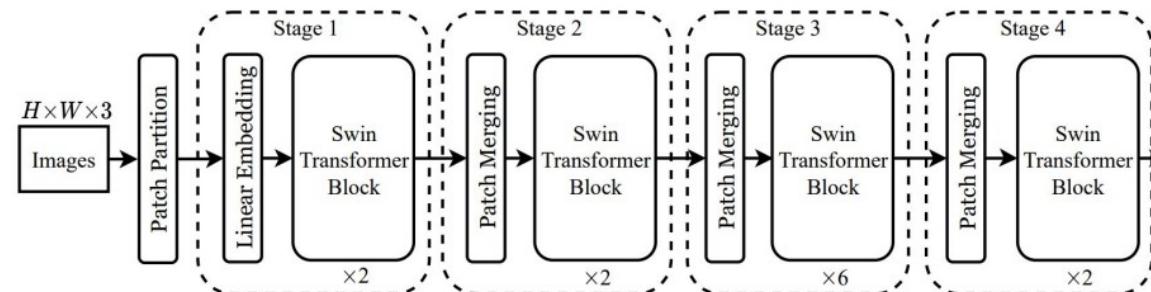


Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ArXiv 2020
[Colab link](#) to an implementation of vision transformers

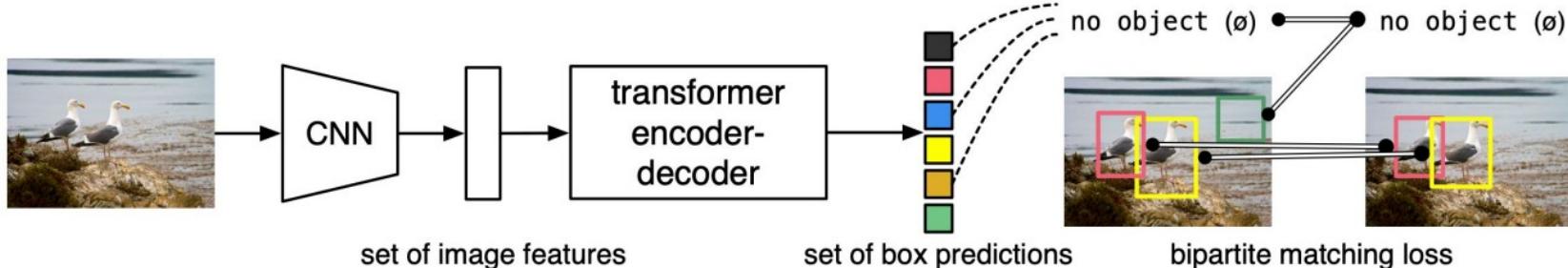
Vision Transformers



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

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