Neural Network and Deep Learning

Lecture7:

Attention and Transformers

lecture inspiration:

Fei-Fei Li, Ehsan Adeli, Zane Durante, 2024

Some papers

OUTLINE

- Sequence to Sequence with RNNs: Encoder –Decoder
- Sequence to Sequence with RNNs and Attention
- Image Captioning using Spatial Features
- Image Captioning with RNNs and Attention
- General Attention Layer
- Self Attention Layer
- Self Attention Layer - Attends Over Sets of Inputs

OUTLINE

- Positional Encoding
- Masked Self-Attention Layer
- Multi-Head Self-Attention Layer
- General Attention versus Self-Attention
- Transformer
- Comparing RNNs to Transformer
- Image Captioning using Transformers
- The Transformer Encoder Block
- The Transformer Decoder Block

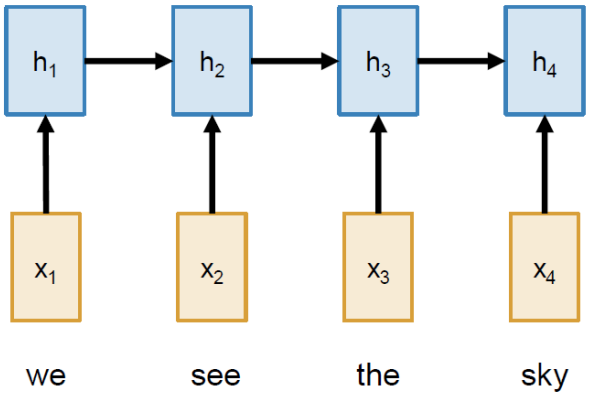
SEQUENCE TO SEQUENCE WITH RNNS: ENCODER -DECODER

A motivating example for today's discussion– machine translation! English→ Italian

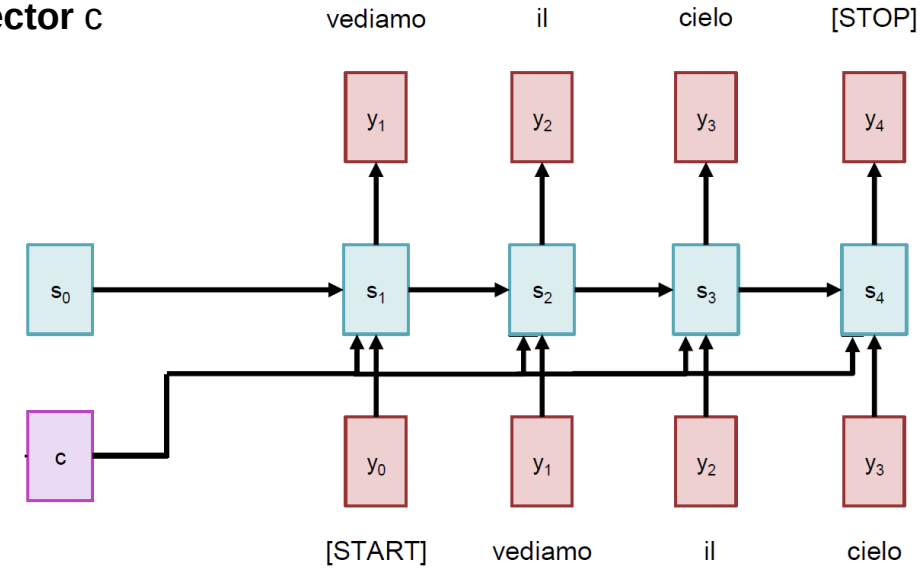
Input: Sequence $x_1, \ldots x_T$
Output: Sequence $y_1, \ldots, y_{T'}$

From final hidden state predict:
Initial decoder state s0 **Context vector** c
(often c=h_T)

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



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



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

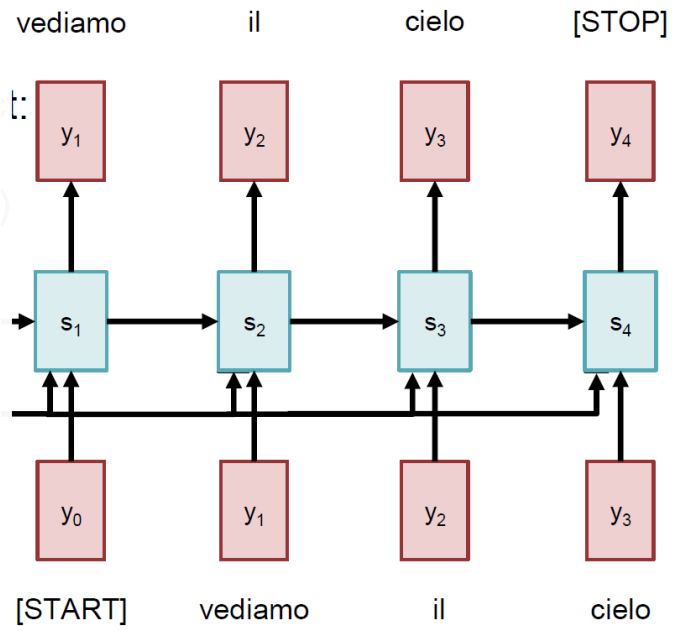
SEQUENCE TO SEQUENCE WITH RNNS: ENCODER -DECODER

Remember:

During Training:
Often, we use the "correct" token even if the model is wrong. Called **teacher forcing**

During Test-time:
We sample from the model's outputs until we sample [STOP]

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



SEQUENCE TO SEQUENCE WITH RNNS: ENCODER -DECODER

Problem statement

In the Encoder–Decoder framework, an encoder reads the input sentence, a sequence of vectors into a vector c. The most common approach is to use an RNN such that

$$h_t = f(x_t, h_{t-1}) \qquad c = q(\{h_1, \dots, h_T\})$$

f and q are some nonlinear functions

Sutskever et al. (2014) used an LSTM as f and $q(\{h_1, \dots, h_T\}) = h_T$

c is a vector generated from the sequence of the hidden states.

$h_t \in R^n$ is a hidden state at time t

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

SEQUENCE TO SEQUENCE WITH RNNS: ENCODER -DECODER

Problem statement

- The **context vector** c_i depends on a sequence of annotations $\{h_1, \dots, h_T\}$ to which an encoder maps the input sentence.
- Each annotation h_j contains information about the whole input sequence with a strong focus on the parts surrounding the j-th word of the input sequence.
- a is a **feedforward** neural network which is jointly trained with the other components of the system.
- s_i is an RNN hidden state for time *i*.

From final hidden state predict:
Initial decoder state s0 **Context vector** c0

$$h_t = f(x_t, h_{t-1})$$

$$e_{ij} = a(s_{i-1}, h_j) \quad \alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{T} exp(e_{ik})}$$

Scalar alignment score

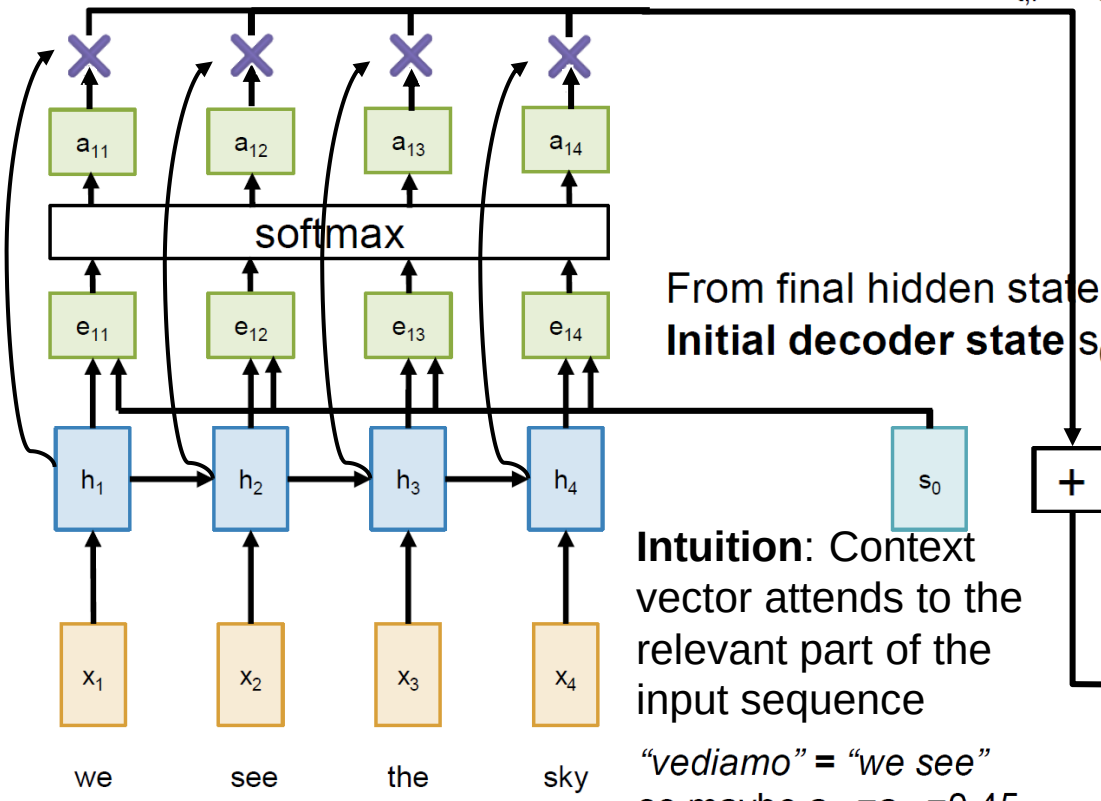
$$c_i = \sum_{j=1}^{T} \alpha_{ij} h_j$$

Context vector

$$s_i = g(s_{i-1}, y_{i-1}, c_i)$$

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 a Linear Layer)

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

From final hidden state:
Initial decoder state s_0

Compute context vector as **weighted sum of hidden states**
$c_t = \sum_i a_{t,i} h_i$

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

"vediamo" = *"we see"*
so maybe a_{11}=a_{12}=0.45, a_{13}=a_{14}=0.05

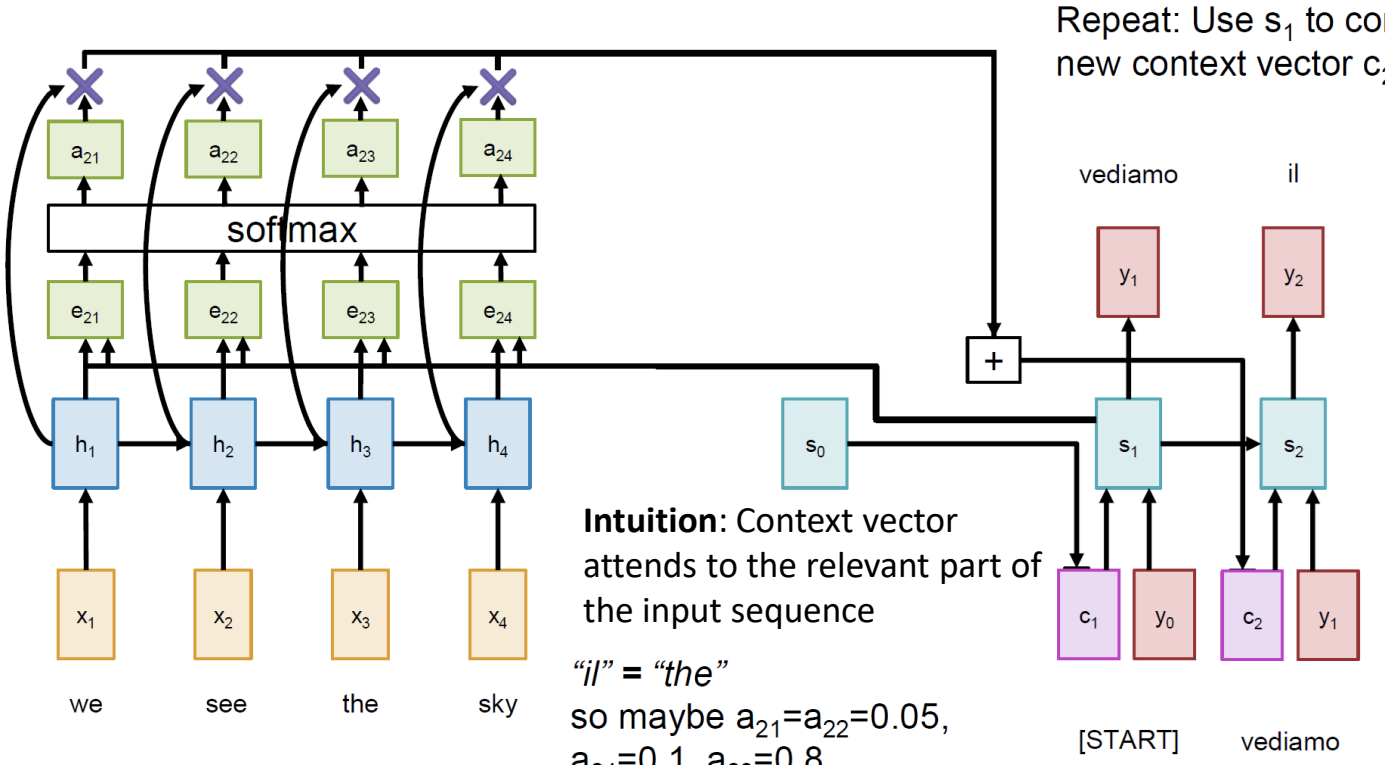
Use context vector in decoder:

$$s_t = g_U(y_{t-1}, s_{t-1}, c_t)$$

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

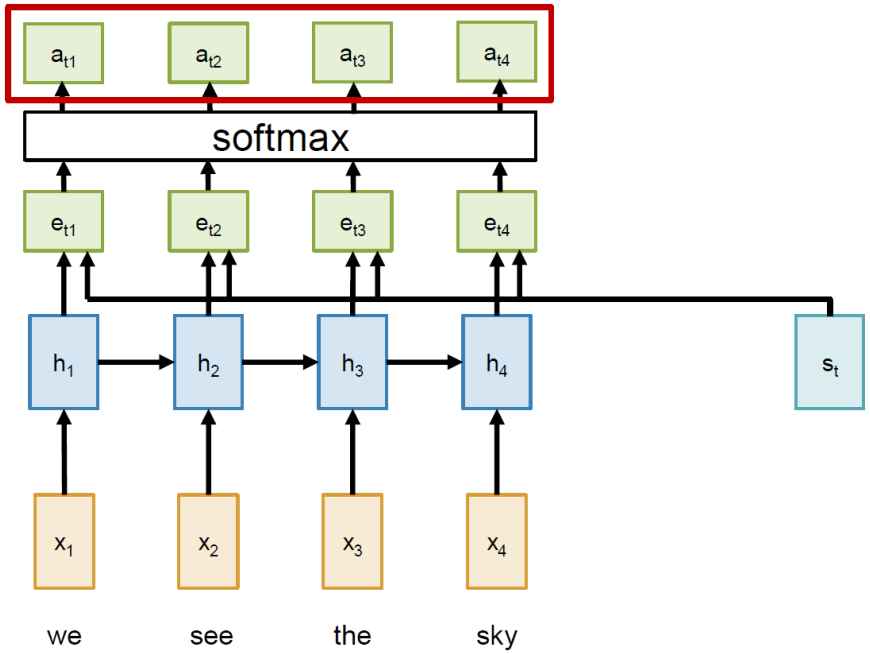
SEQUENCE TO SEQUENCE WITH RNNS AND ATTENTION

Context vectors don't use the fact that h_i form an ordered sequence – it just treats them as an unordered set $\{h_i\}$

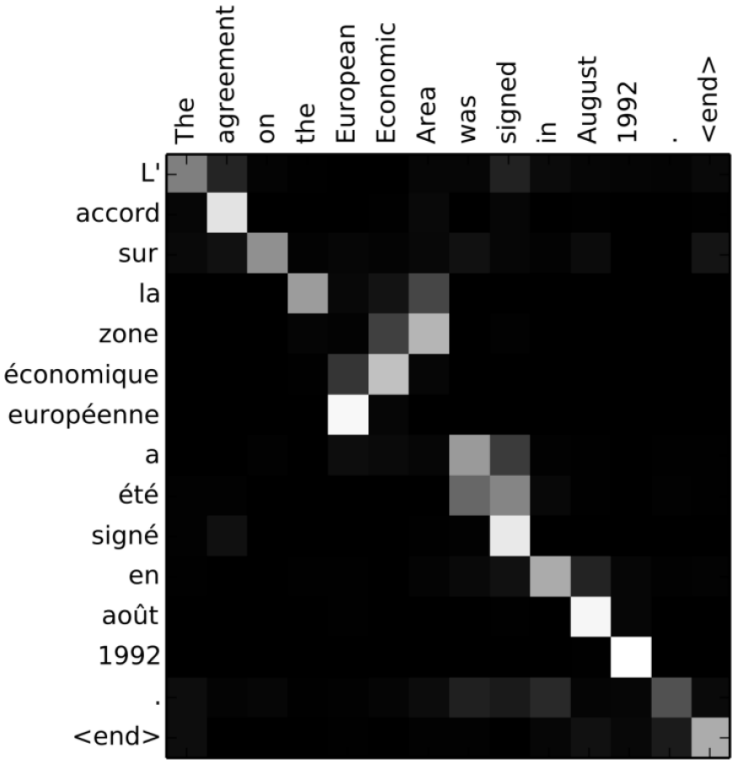


SEQUENCE TO SEQUENCE WITH RNNS AND ATTENTION

Example: English to French translation



Visualize attention weights $a_{t,i}$



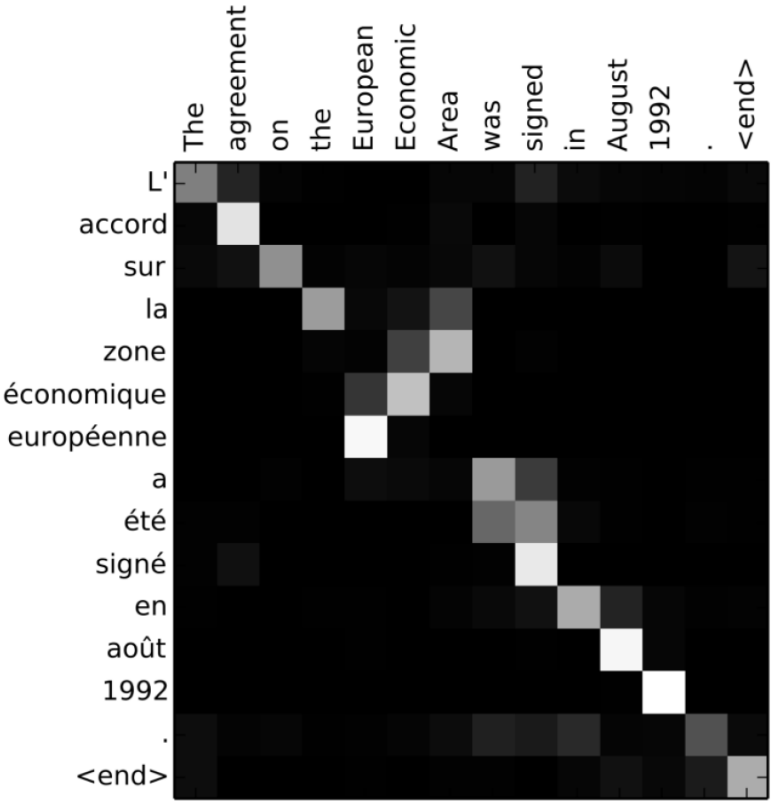
SEQUENCE TO SEQUENCE WITH RNNS AND ATTENTION

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.”

Visualize attention weights $a_{t,i}$



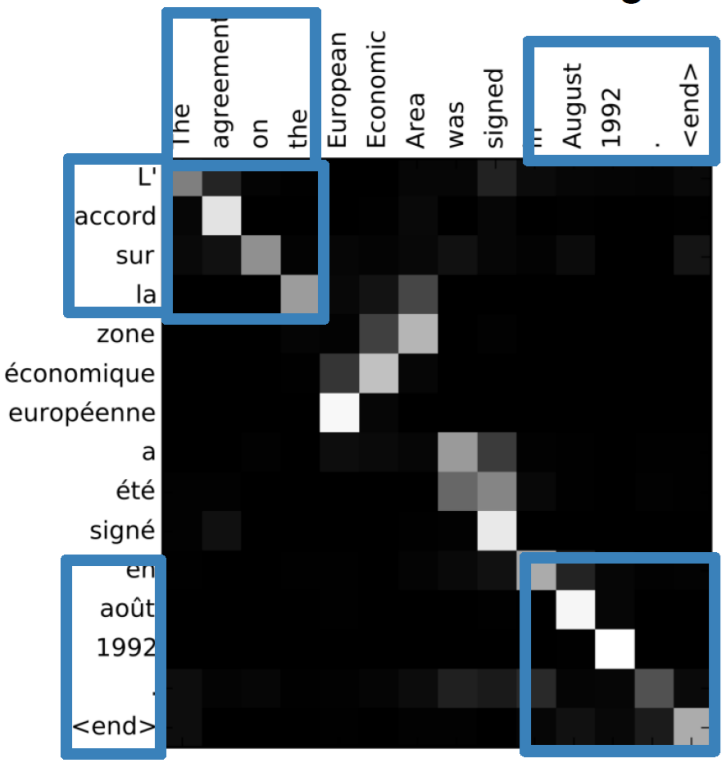
SEQUENCE TO SEQUENCE WITH RNNS AND ATTENTION

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**."

Visualize attention weights $a_{t,i}$



Diagonal attention means words correspond in order

Diagonal attention means words correspond in order

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 $a_{t,i}$

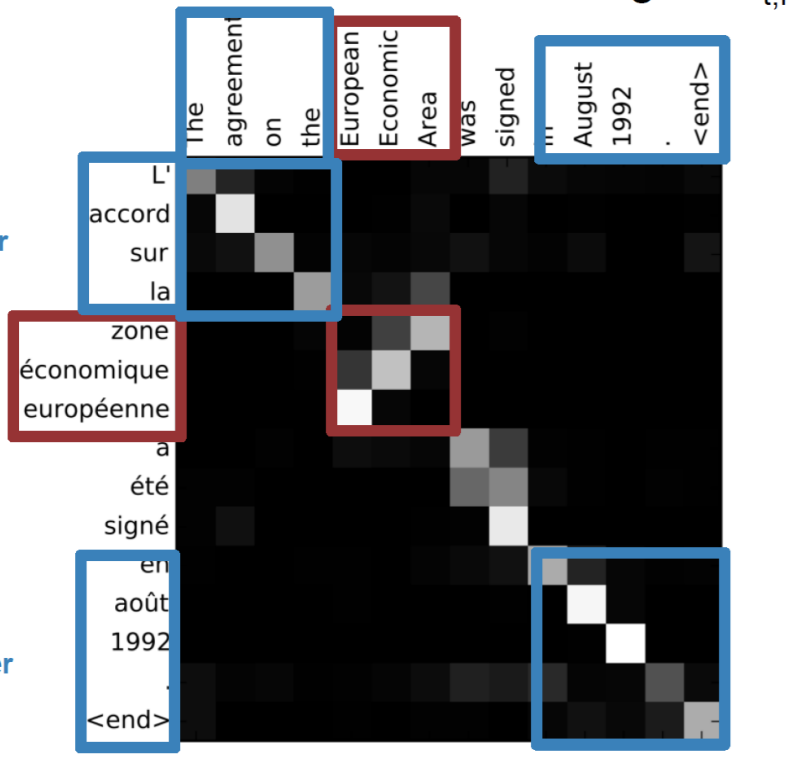
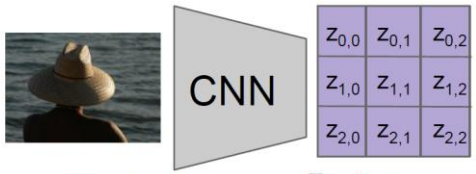


IMAGE CAPTIONING USING SPATIAL FEATURES

- An example network for image captioning **without attention**

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

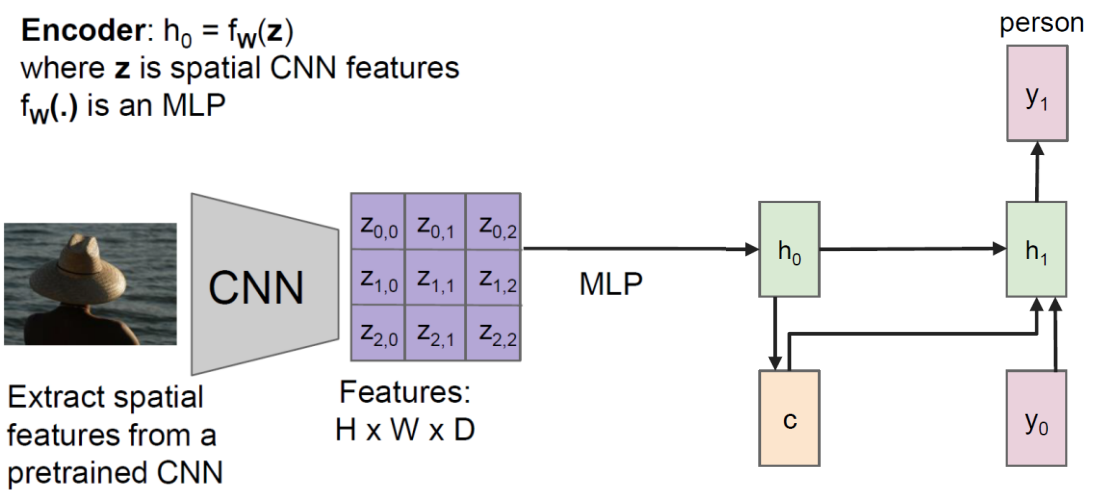
Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

IMAGE CAPTIONING USING SPATIAL FEATURES

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

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

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



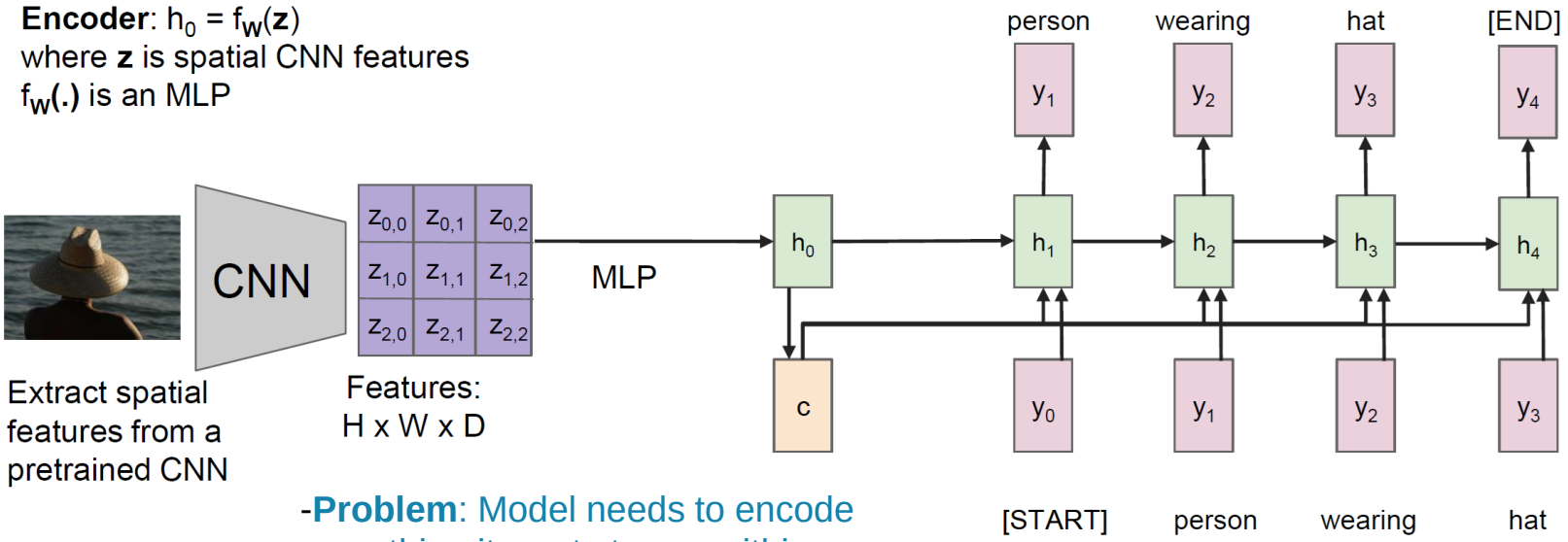
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

IMAGE CAPTIONING USING SPATIAL FEATURES

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

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

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



Extract spatial features from a pretrained CNN

Features: H x W x D

-**Problem**: Model needs to encode everything it wants to say within c

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

IMAGE CAPTIONING USING SPATIAL FEATURES

Problem statement

- ENCODER: CONVOLUTIONAL FEATURES:
  - Our model takes a single raw image and generates a caption y encoded as a sequence of 1-of-K encoded words.

$$y = \{\mathbf{y}_1, \ldots, \mathbf{y}_C\}, \ \mathbf{y}_i \in \mathbb{R}^K$$

  - where K is the size of the vocabulary and C is the length of the caption.
  - We use a convolutional neural network in order to extract a set of feature vectors which we refer to as annotation vectors.
  - The extractor produces L vectors, each of which is a D-dimensional representation corresponding to a part of the image.

$$a = \{\mathbf{a}_1, \ldots, \mathbf{a}_L\}, \ \mathbf{a}_i \in \mathbb{R}^D$$

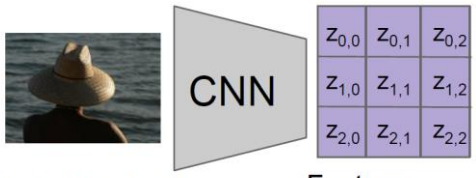
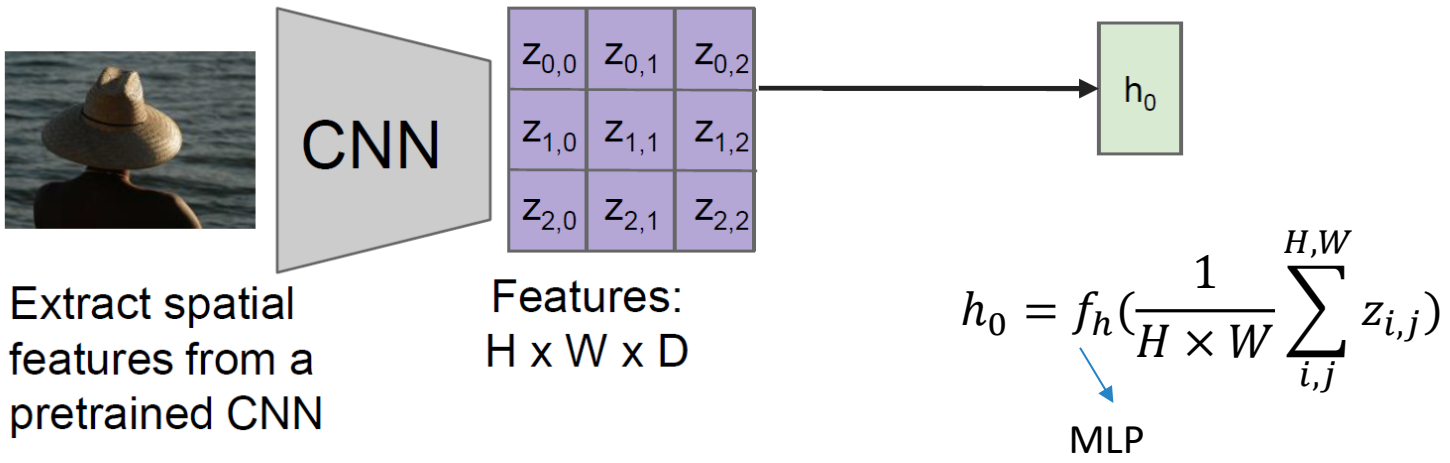


IMAGE CAPTIONING USING SPATIAL FEATURES

- **Problem statement**
  - ENCODER: CONVOLUTIONAL FEATURES:
    - In order to obtain a correspondence between the feature vectors and portions of the 2-D image, we extract features from a lower convolutional layer unlike previous work which instead used a fully connected layer. This allows the decoder to selectively focus on certain parts of an image by selecting a subset of all the feature vectors.

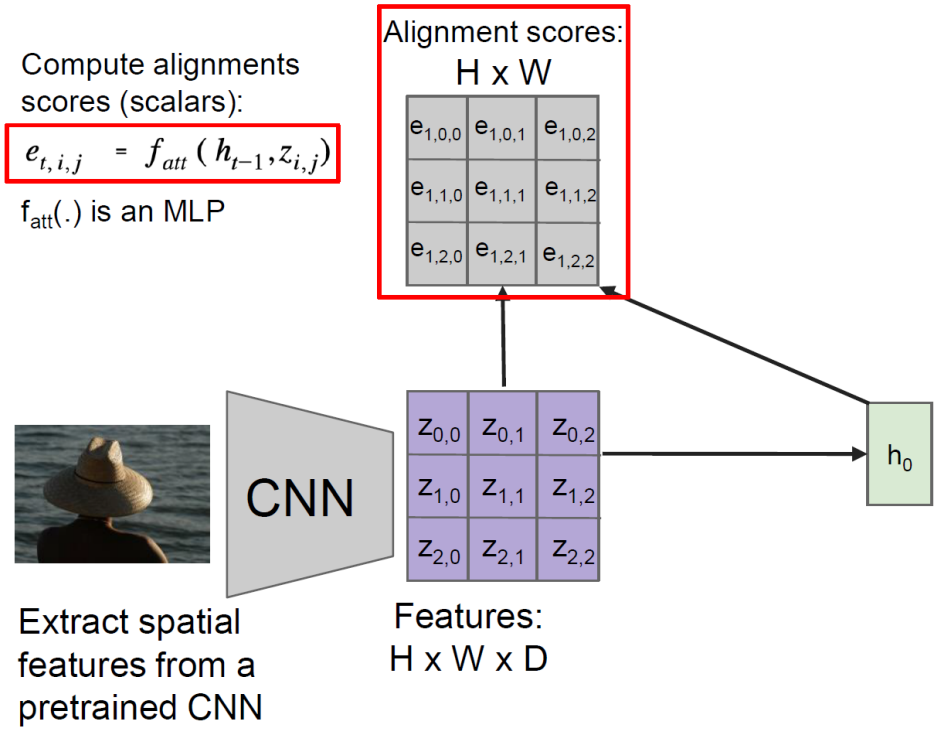
IMAGE CAPTIONING WITH RNNS AND ATTENTION

- **Attention idea: New context vector at every time step.**
- **Each context vector will attend to different image regions**



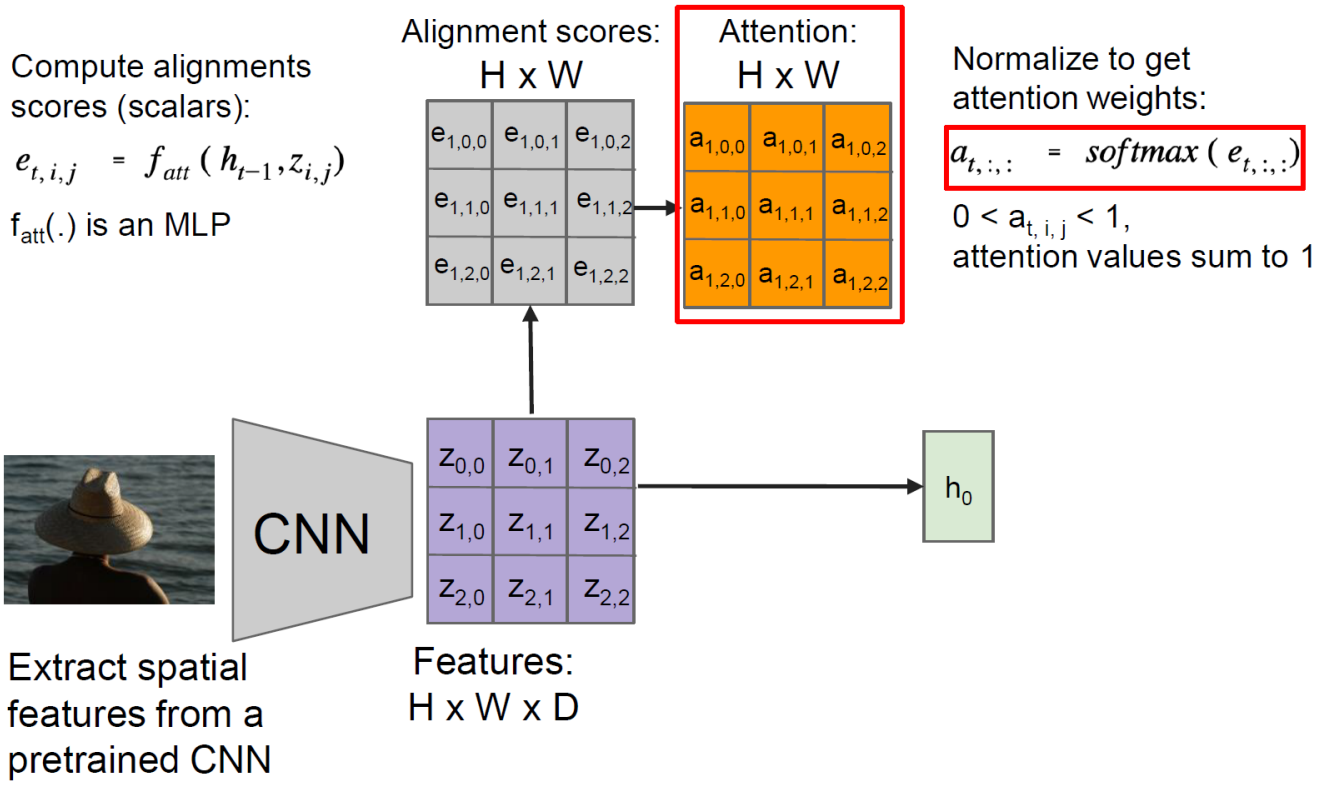
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

IMAGE CAPTIONING WITH RNNS AND ATTENTION



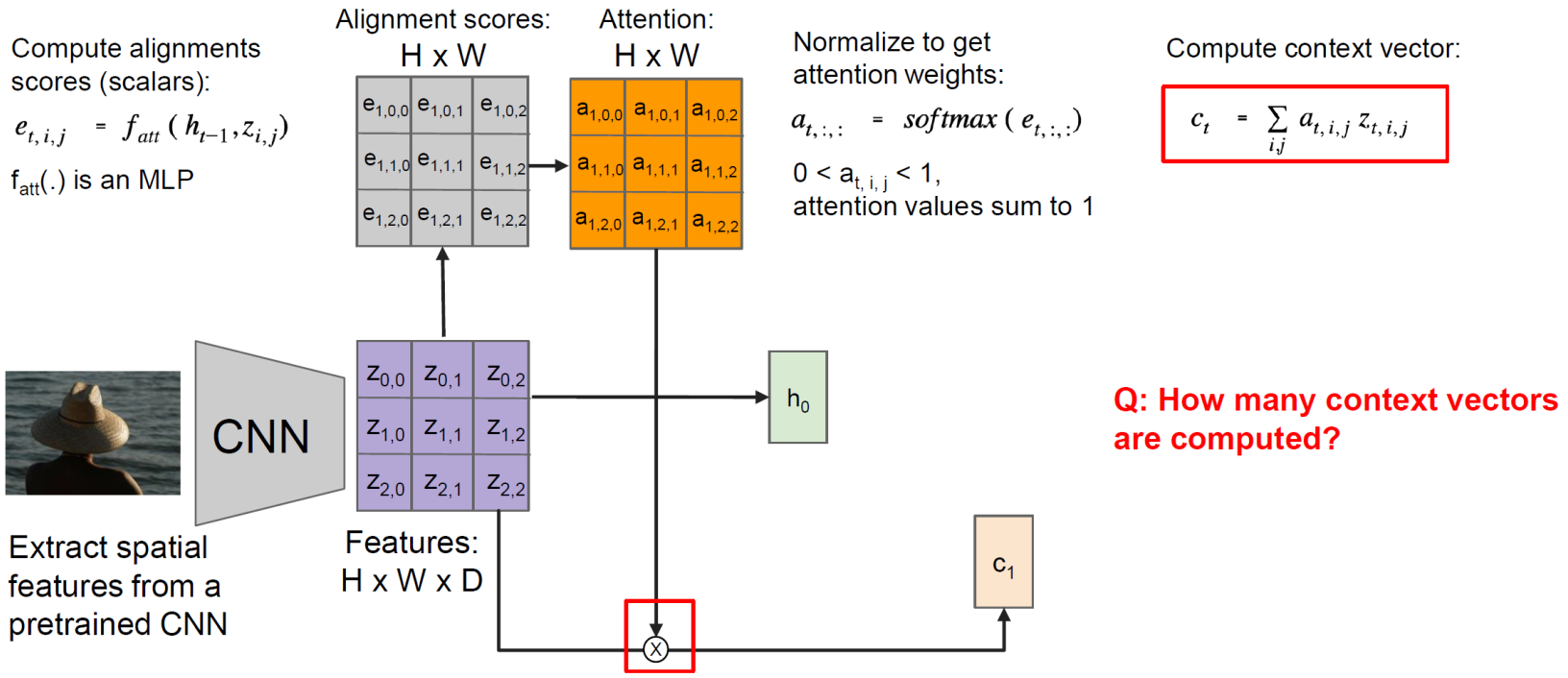
Xu et al, “Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015

IMAGE CAPTIONING WITH RNNS AND ATTENTION



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

IMAGE CAPTIONING WITH RNNS AND ATTENTION



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

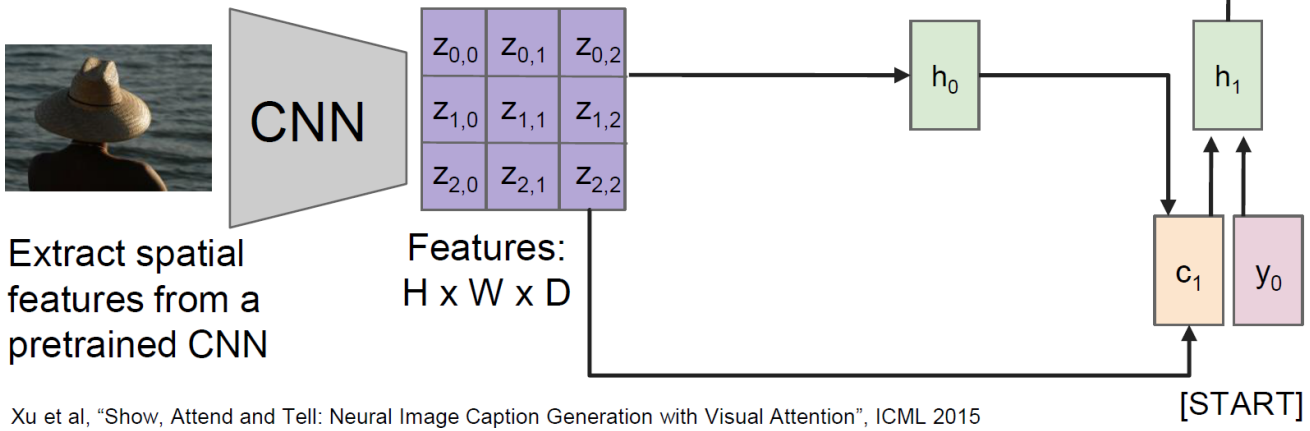
IMAGE CAPTIONING WITH RNNS AND ATTENTION

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

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

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

$$\begin{pmatrix} i_t \\ f_t \\ o_t \\ g_t \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ tanh \end{pmatrix} T_{4n \times (m+n+D)} \begin{pmatrix} Ey_{t-1} \\ h_{t-1} \\ c_t \end{pmatrix}$$



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

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

CNN

Extract spatial features from a pretrained CNN

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

Features: H x W x D

h_0 h_1 h_2 h_3 h_4

y_1 y_2 y_3 y_4

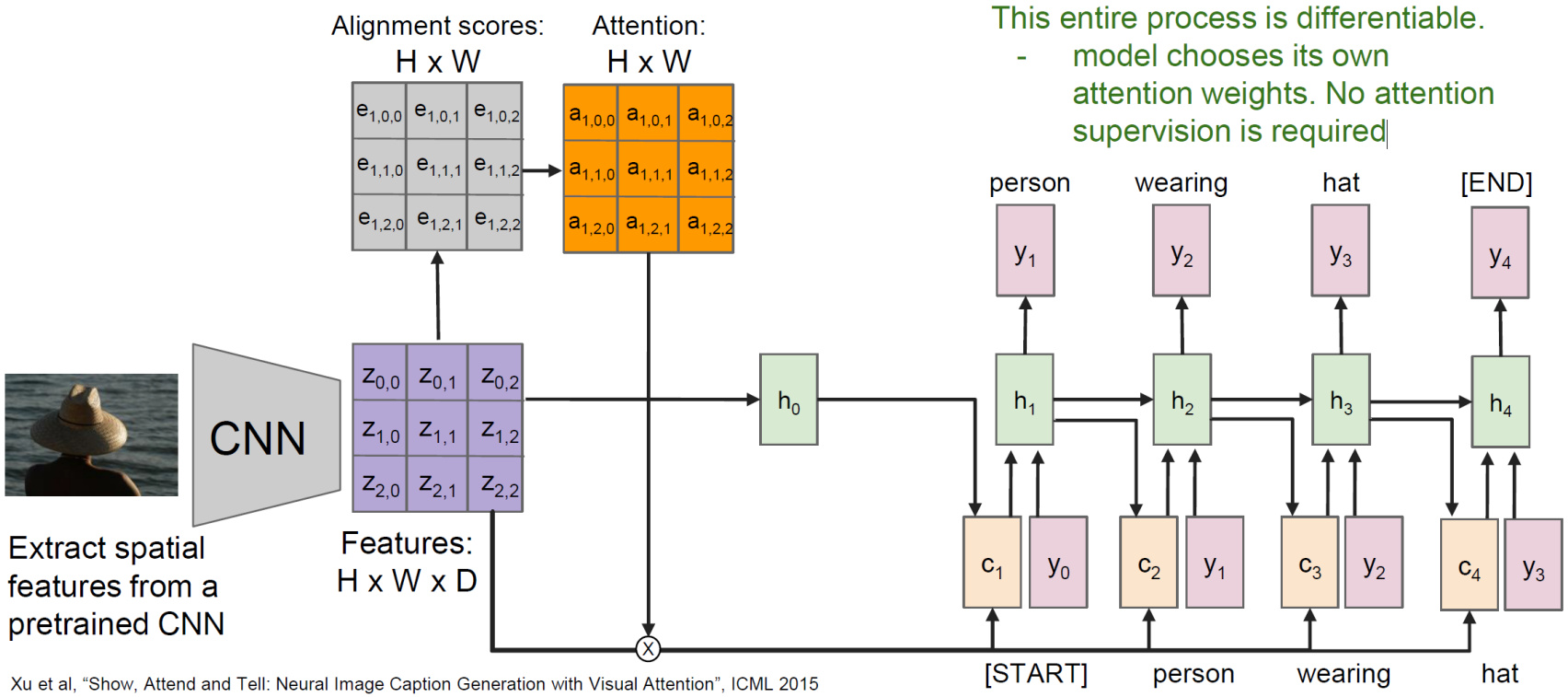
person wearing hat [END]

c_1 y_0 c_2 y_1 c_3 y_2 c_4 y_3

[START] person wearing hat

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

IMAGE CAPTIONING WITH RNNS AND ATTENTION



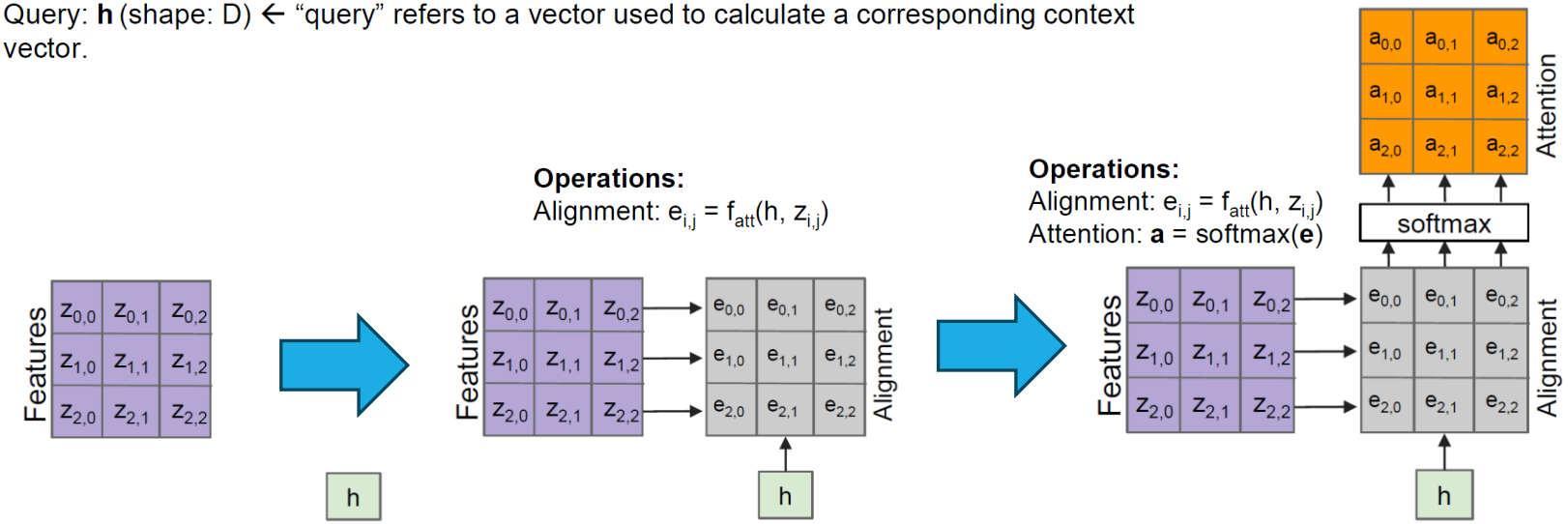
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

IMAGE CAPTIONING WITH ATTENTION

Xu et al, “Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention”, ICML 2015
Figure copyright Kelvin Xu, Jimmy Lei Ba, Jamie Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Benchio, 2015.

ATTENTION WE JUST SAW IN IMAGE CAPTIONING

Inputs:
Features: **z** (shape: H x W x D)
Query: **h** (shape: D) ← "query" refers to a vector used to calculate a corresponding context vector.

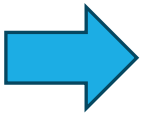


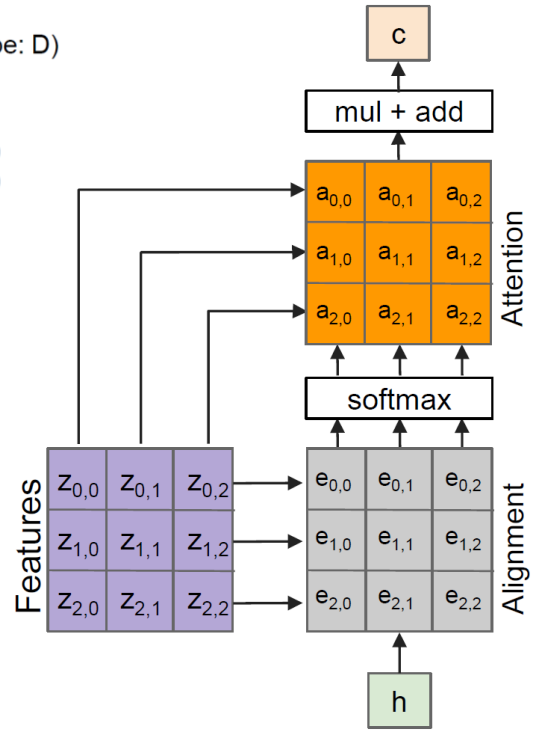
ATTENTION WE JUST SAW IN IMAGE CAPTIONING

Inputs:
Features: **z** (shape: H x W x D)
Query: **h** (shape: D) ← "query" refers to a vector used to calculate a corresponding context vector.

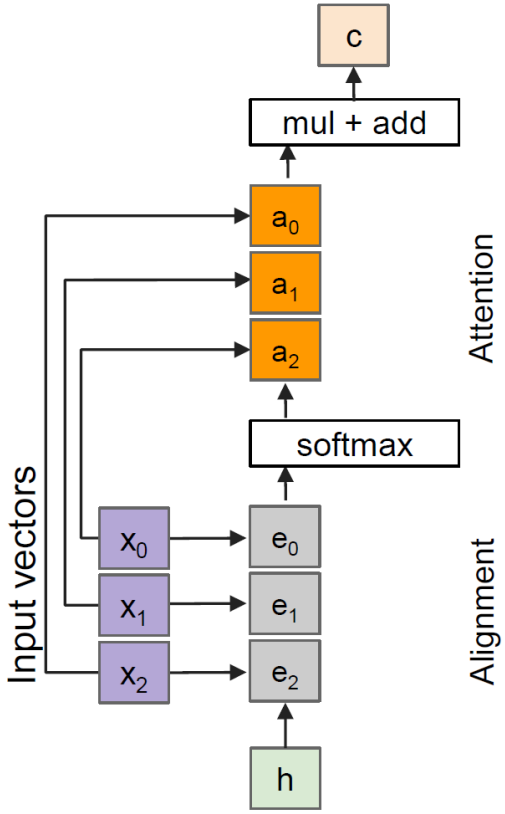
Outputs:
context vector: **c** (shape: D)

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





GENERAL ATTENTION LAYER



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

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

Change fatt(.) to a dot product, this actually can work well in practice, but a simple dot product can have some issues…

Alignment: $e_i = h \cdot x_i$

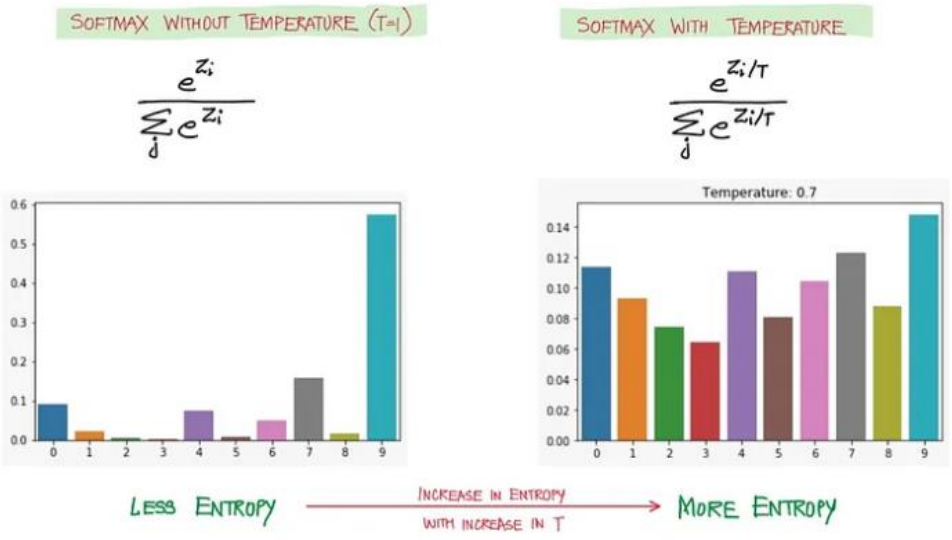
Attention operation is **permutation invariant.**

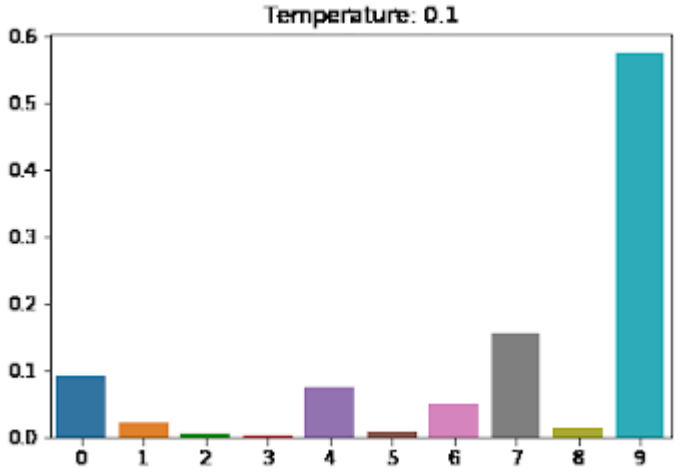
- Doesn't care about ordering of the features
- Stretch into **N = H x W** vectors

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

GENERAL ATTENTION LAYER

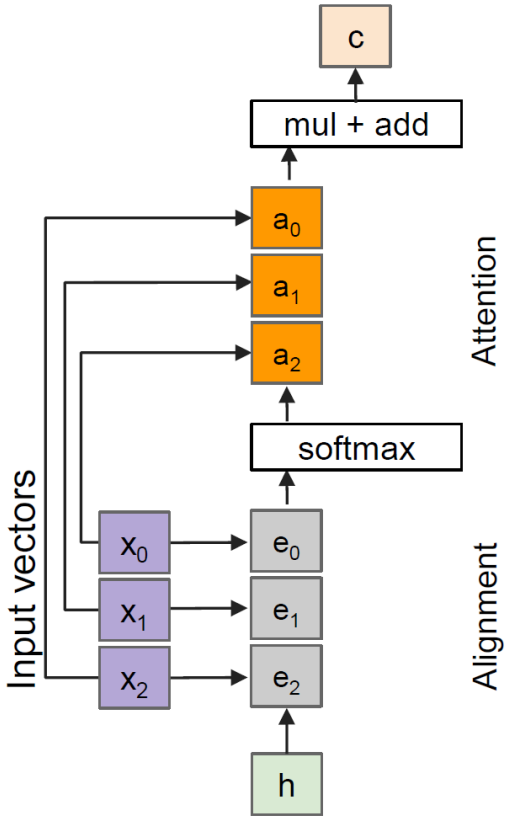
- **Softmax Temperature**





Visualizing the Effects of Temperature Scaling. Each word gets equal probability as the Temperature increases

GENERAL ATTENTION LAYER



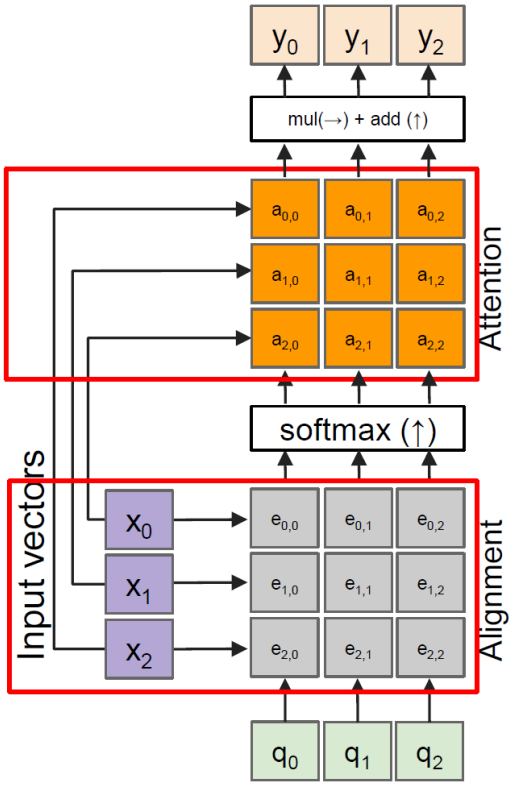
Outputs:
context vector: **c** (shape: D)

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

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

- Change fatt(.) 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.
- 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
- Similar to Xavier and Kaiming Initialization!

GENERAL ATTENTION LAYER



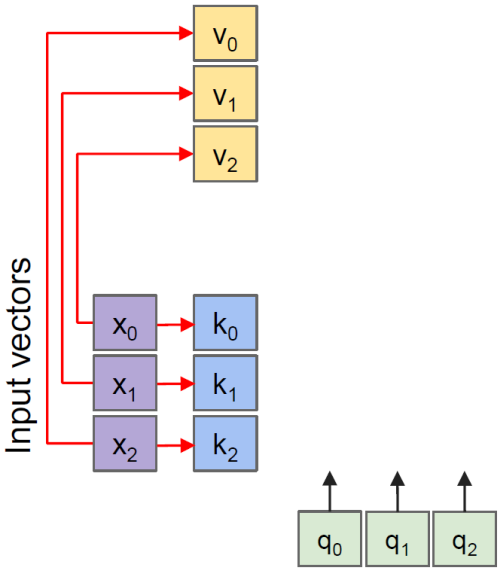
Outputs:
context vectors: **y** (shape: D)

Operations:
Alignment: $e_{i,j} = q_j \cdot x_i / \sqrt{D}$
Attention: **a** = softmax(**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

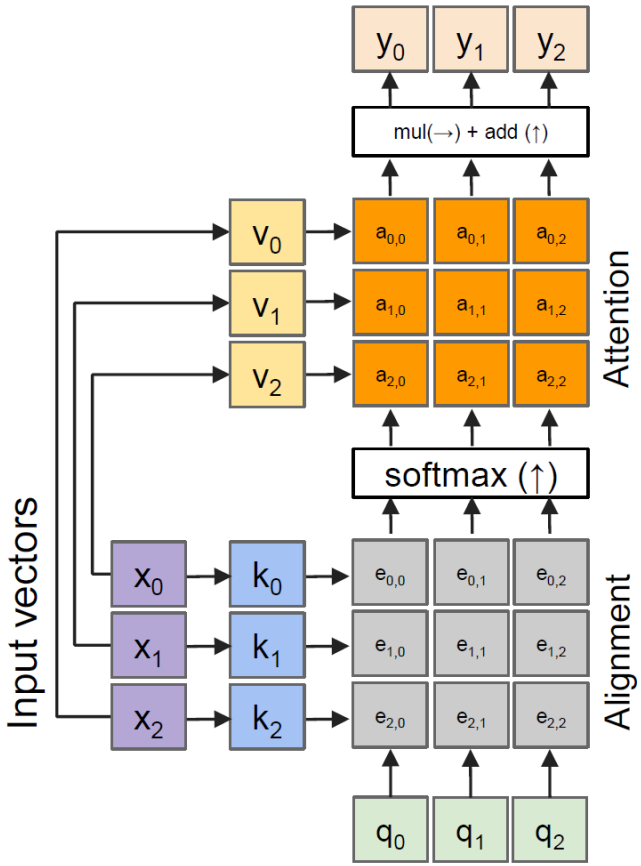


Operations:
Key vectors: $\mathbf{k} = \mathbf{x}W_k$
Value vectors: $\mathbf{v} = \mathbf{x}W_v$

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

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

GENERAL ATTENTION LAYER



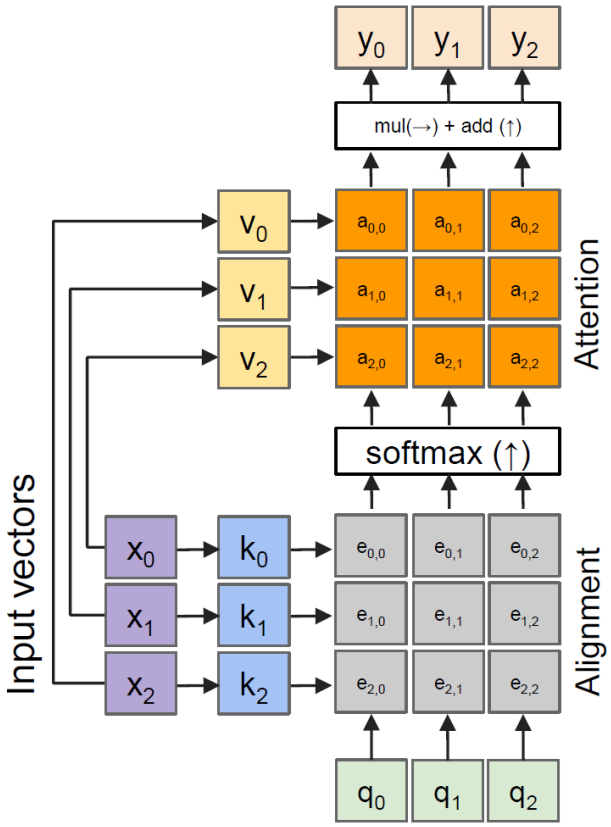
Outputs:
context vectors: **y** (shape: D_v)

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

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

- ✓ The input and output dimensions can now change depending on the key and value FC layers
- ✓ Since the alignment scores are just scalars, the value vectors can be any dimension we want

GENERAL ATTENTION LAYER



This is a working example of how we could use an attention layer + CNN encoder for image captioning

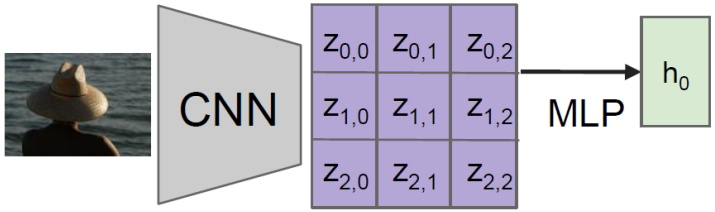
Outputs:
context vectors: **y** (shape: D_v)

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

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

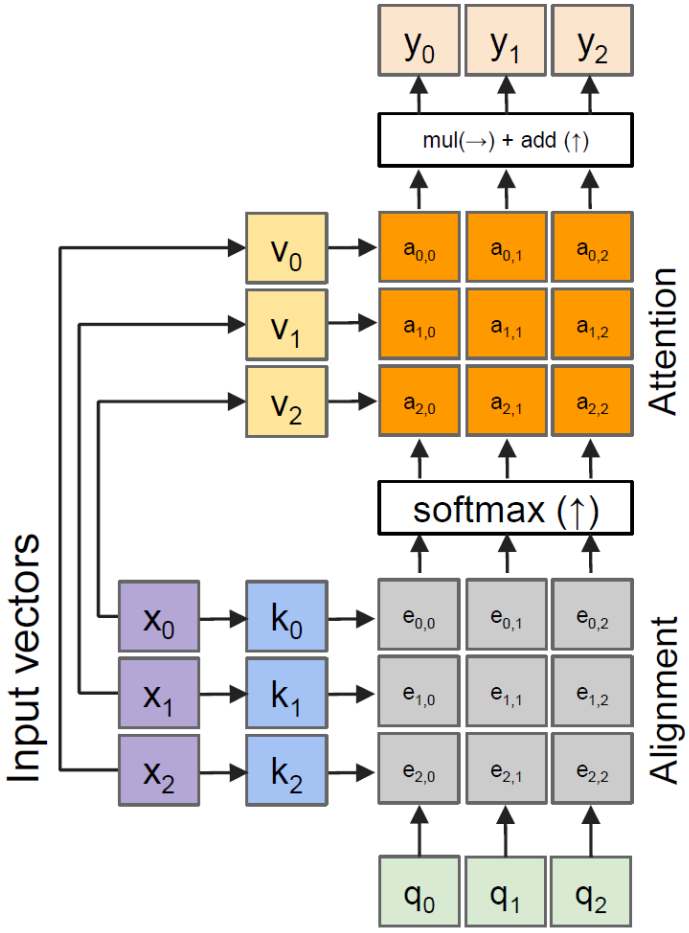
Recall that the query vector was a function of the input vectors

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



We used h_0 as q_0 previously

THE SELF-ATTENTION LAYER



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

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

Idea: leverages the strengths of attention layers without the need for separate query vectors.

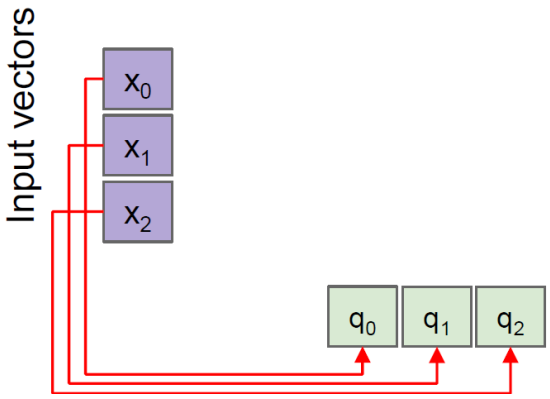
Inputs:
Input vectors: **x** (shape: N x D)
Queries: **q** (shape: M x D_k)

SELF ATTENTION LAYER

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

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

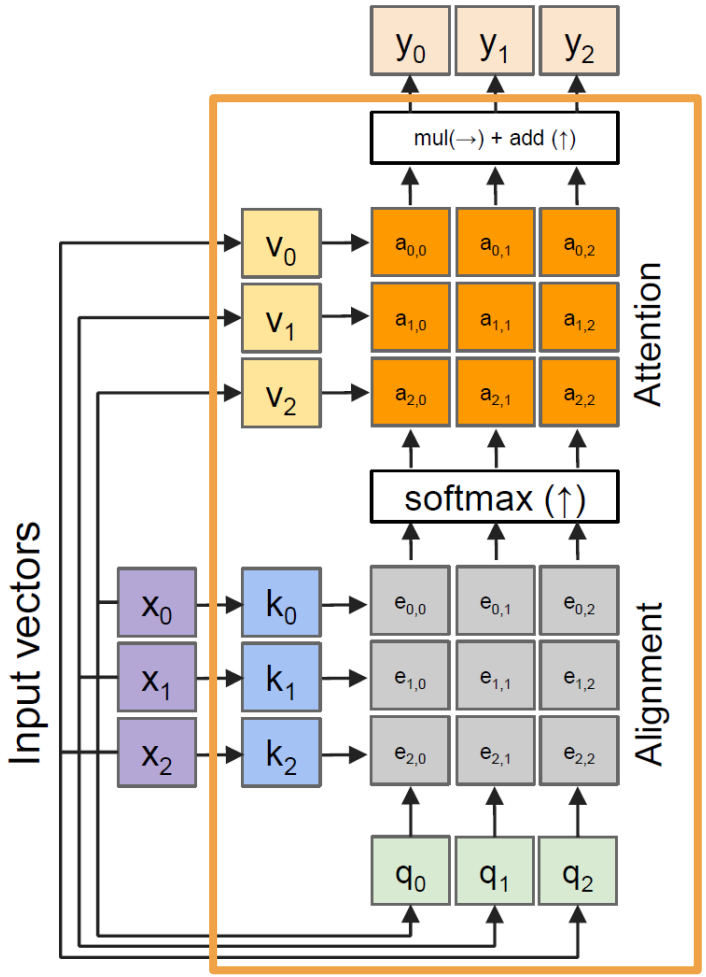
Instead, query vectors are calculated using a FC layer.



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

No input query vectors anymore

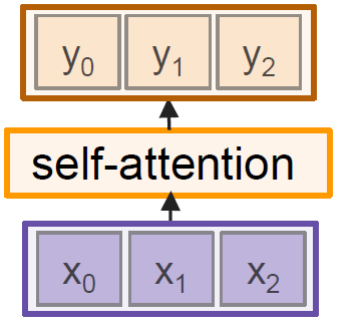
SELF ATTENTION LAYER - ATTENDS OVER SETS OF INPUTS



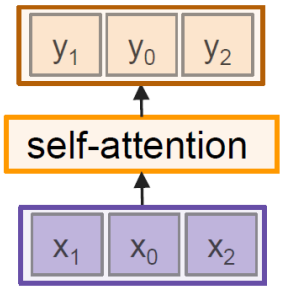
Outputs:
context vectors: **y** (shape: D_v)

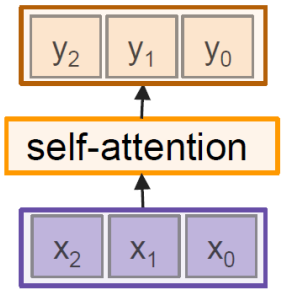
Operations:
Key vectors: $\mathbf{k} = \mathbf{x}W_k$
Value vectors: $\mathbf{v} = \mathbf{x}W_v$
Query vectors: $\mathbf{q} = \mathbf{x}W_q$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $y_j = \sum_i a_{i,j} v_i$

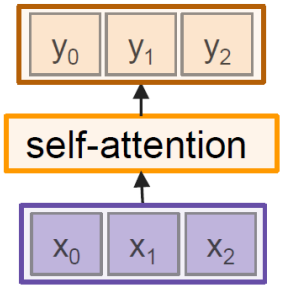
Inputs:
Input vectors: **x** (shape: N x D)



SELF ATTENTION LAYER - ATTENDS OVER SETS OF INPUTS







Permutation 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?

TRANSFORMER

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

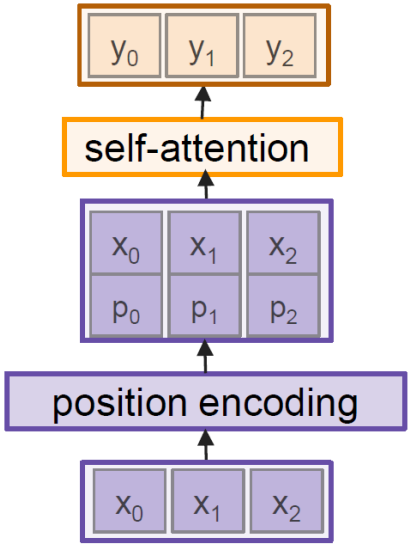
Train a giant Transformer model for language modeling

Finetuning:

Fine-tune the Transformer on your own NLP task

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

POSITIONAL ENCODING



Concatenate or **add** special positional encoding p_j to each input vector x_j.

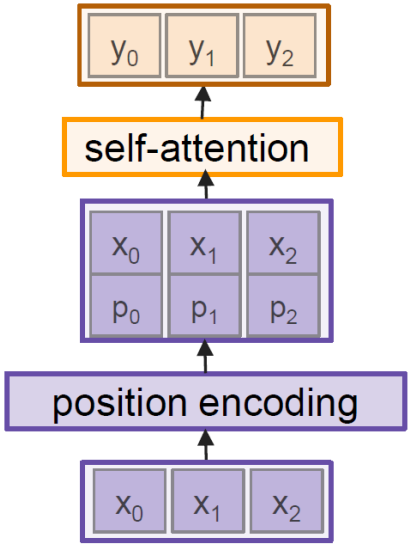
We use a function *pos*: $N \rightarrow R^d$ to process the position j of the vector into a d-dimensional vector

So, p_j = *pos*(j)

Possible desirable properties of pos(.) :

1. It should output a unique encoding for each timestep (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 or **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)

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 desired properties

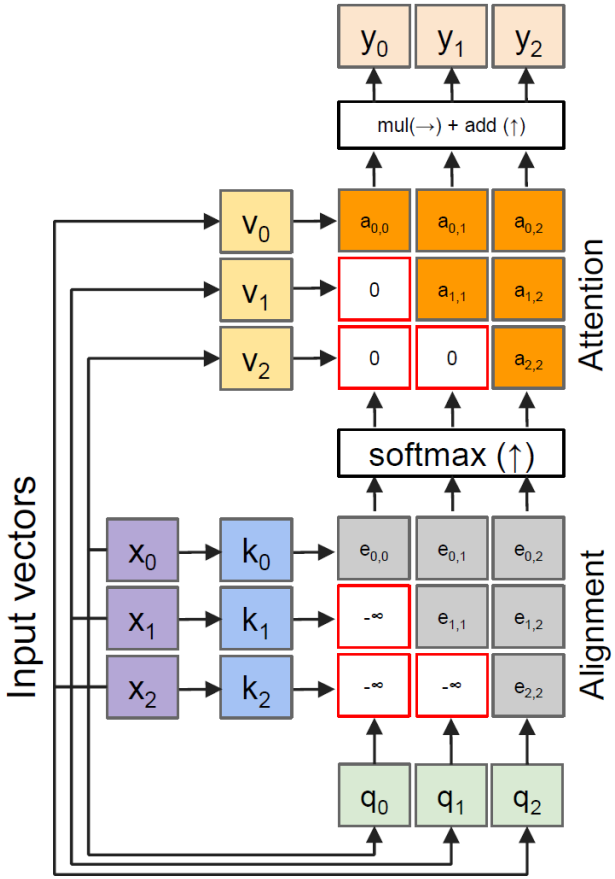
$$pos(t) = \begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \vdots \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}_d , where \ \omega_k = \frac{1}{10000^{2k/d}}$$

The positional encodings have the same dimension d as the embeddings, so that the two can be summed. There are many choices of positional encodings, learned and fixed [9].

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

[9] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. arXiv preprint arXiv:1705.03122v2, 2017

MASKED SELF-ATTENTION LAYER



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

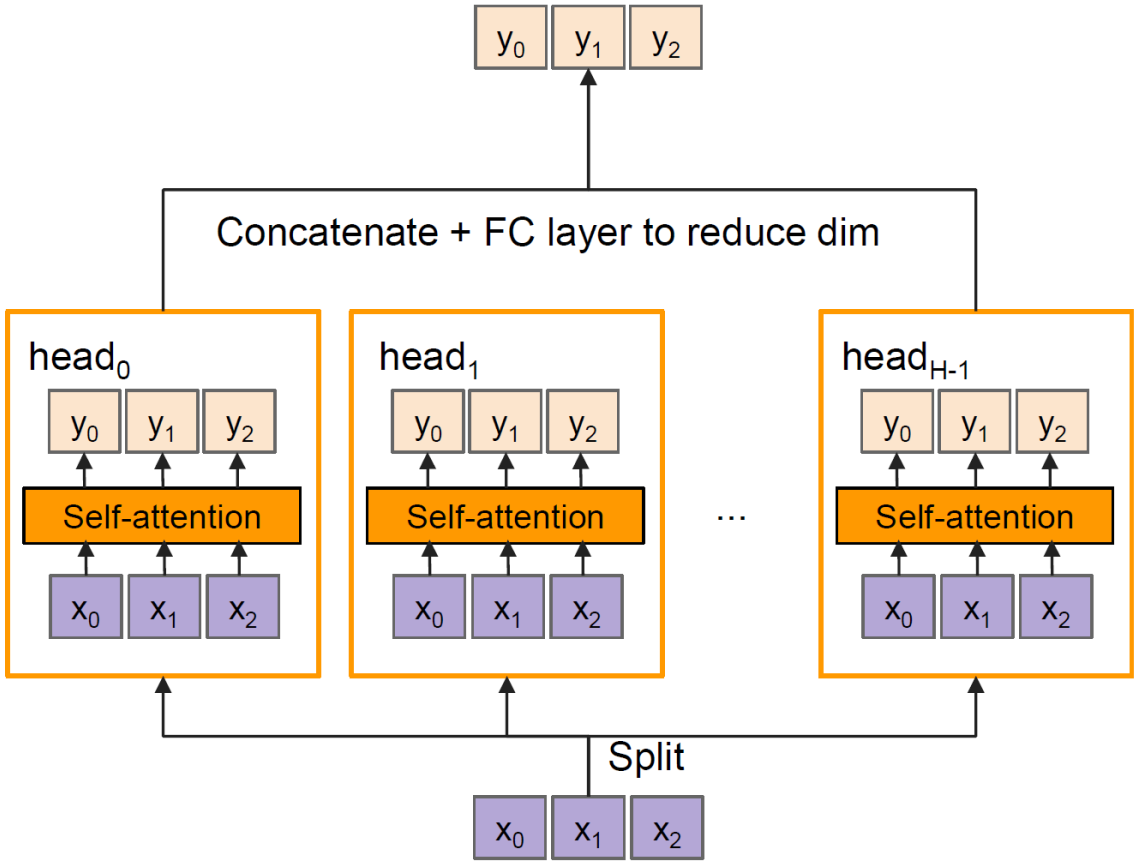
Operations:
Key vectors: $\mathbf{k} = \mathbf{x}W_k$
Value vectors: $\mathbf{v} = \mathbf{x}W_v$
Query vectors: $\mathbf{q} = \mathbf{x}W_q$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $y_j = \sum_i a_{i,j} v_i$

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

- Allows us to parallelize attention across time
- Don't need to calculate the context vectors from the previous timestep first!
- Prevent vectors from looking at future vectors.
- Manually set alignment scores to –infinity (-nan)

MULTI-HEAD SELF-ATTENTION LAYER

- Multiple self-attention “heads” in parallel



Q: Why do this?

A: We may want to have multiple sets of queries/keys/values calculated in the layer.
This is a similar idea to having multiple conv filters learned in a layer

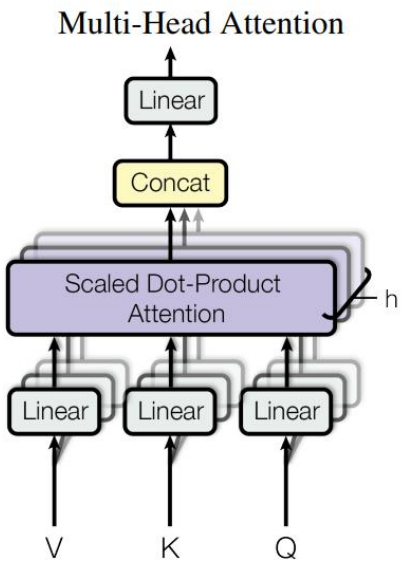
MULTI-HEAD SELF-ATTENTION LAYER

- Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

$$W_i^Q \in \mathbb{R}^{d_\text{model} \times d_k}, W_i^K \in \mathbb{R}^{d_\text{model} \times d_k}, W_i^V \in \mathbb{R}^{d_\text{model} \times d_v}$$

$$W^O \in \mathbb{R}^{hd_v \times d_\text{model}}$$



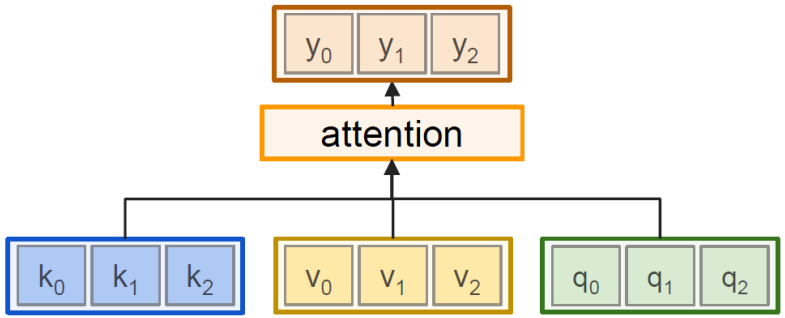
h = 8 parallel attention layers, or heads

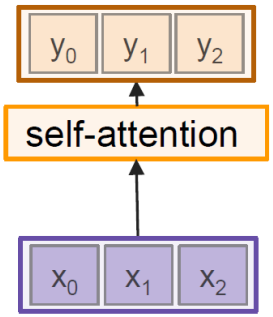
$$d_k = d_v = \frac{d_{model}}{h} = 64$$

machine translation task: Vaswani et al, “Attention is all you need”, NeurIPS 2017

GENERAL ATTENTION VERSUS SELF-ATTENTION

Transformer models rely on many, stacked self-attention layers





Self Study

TRANSFORMER

- **Definition**:
  - A transformer is a deep learning architecture developed by researchers at Google and based on the multi-head attention mechanism, proposed in the 2017 paper "Attention Is All You Need".
  - Text is converted to numerical representations called tokens, and each token is converted into a vector via lookup from a word embedding table.
  - At each layer, each token is then contextualized within the scope of the context window with other (unmasked) tokens via a parallel multi-head attention mechanism, allowing the signal for key tokens to be amplified and less important tokens to be diminished.

- **Advantages**:
  - Transformers have the advantage of having no recurrent units, therefore requiring less training time than earlier recurrent neural architectures (RNNs) such as long short-term memory (LSTM).
  - Later variations have been widely adopted for training large language models (LLM) on large (language) datasets, such as the Wikipedia corpus and Common Crawl.

- **Applications**:
  - They are used in large-scale natural language processing, computer vision (vision transformers), reinforcement learning, audio, multimodal learning, robotics, and even playing chess.
  - It has also led to the development of pre-trained systems, such as generative pre-trained transformers (GPTs) and BERT (bidirectional encoder representations from transformers).

https://en.wikipedia.org/wiki/Transformer_(deep_learning_architecture)

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.

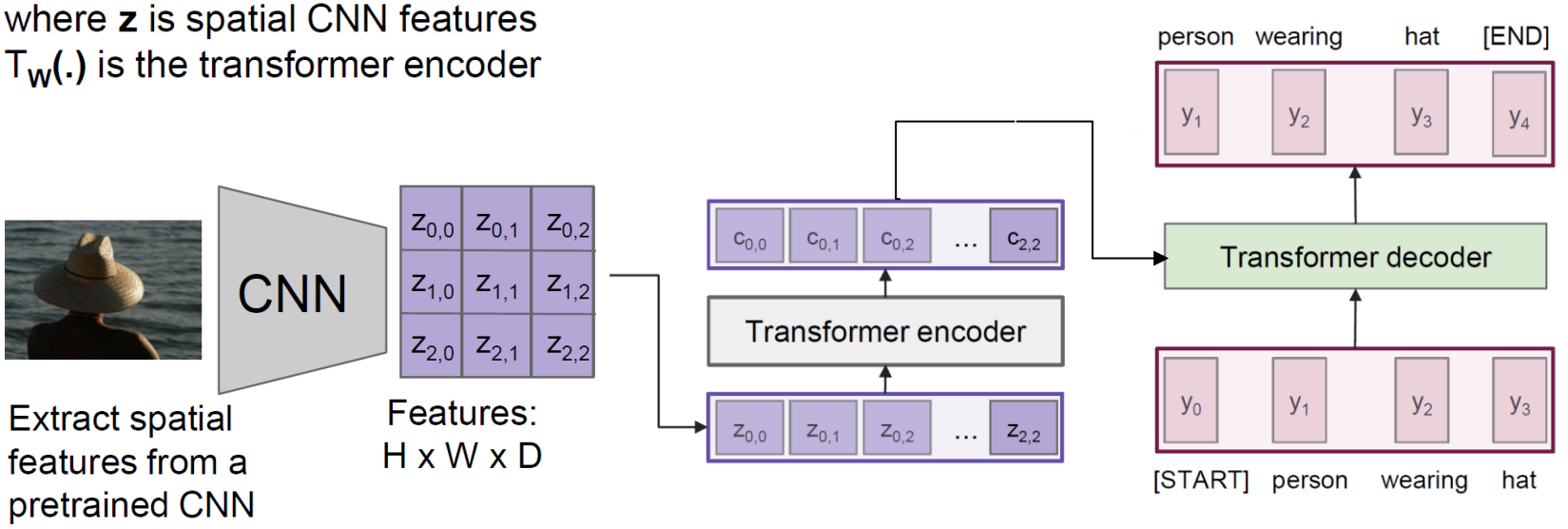
  (-) Requires a lot of memory: N x M alignment and attention scalers need to be calculated and stored for a single self-attention head. (but GPUs are getting bigger and better)

IMAGE CAPTIONING USING TRANSFORMERS

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

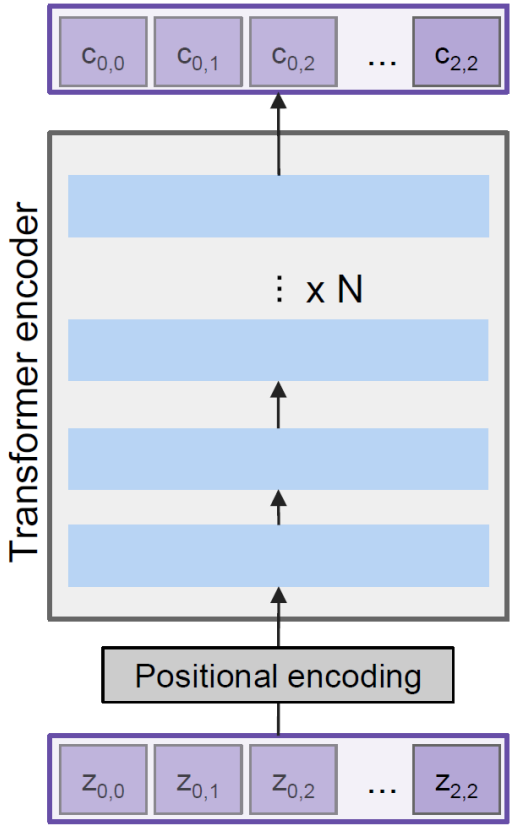
Decoder: $y_t = T_D(\mathbf{y}_{0:t-1}, \mathbf{c})$
where $T_D(.)$ is the transformer decoder

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



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

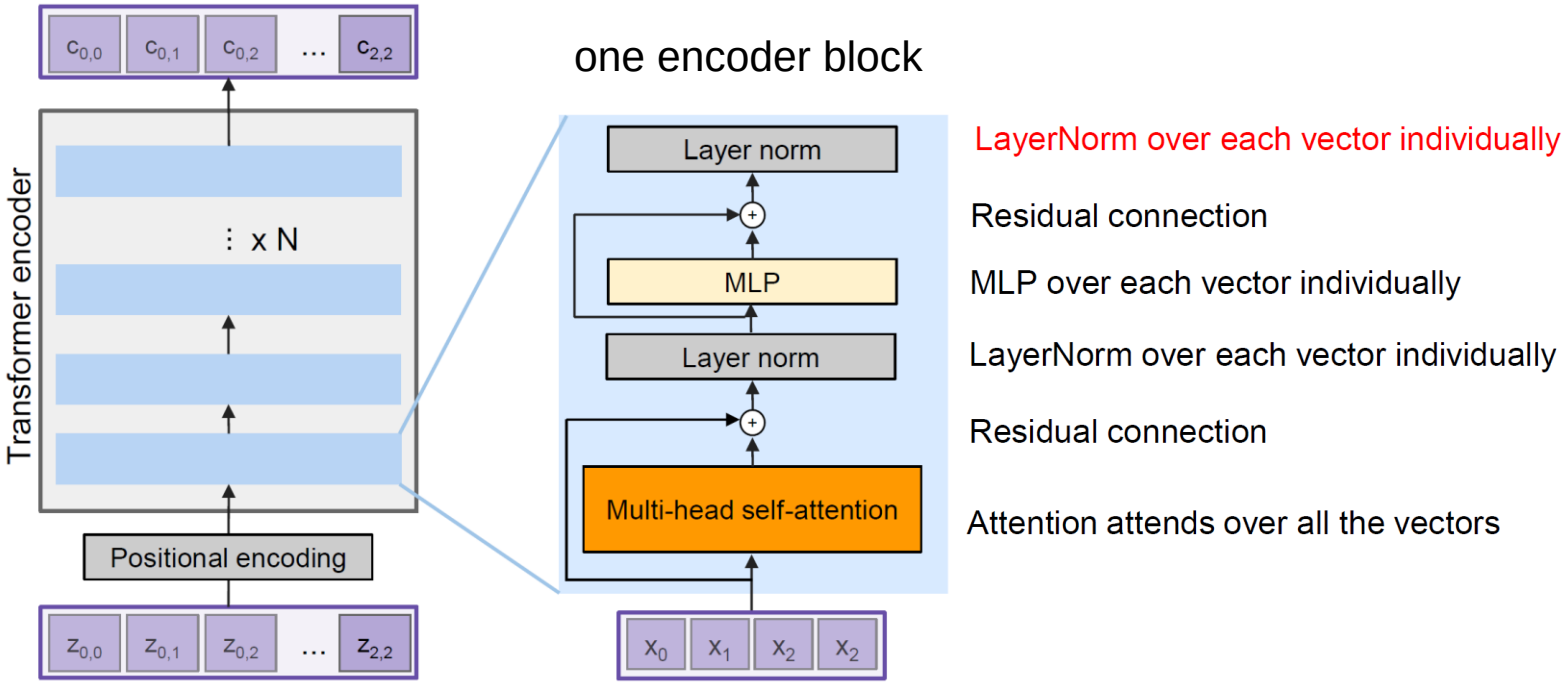
THE TRANSFORMER ENCODER BLOCK



Made up of N encoder blocks.
In vaswani et al. N = 6, d_q= 512

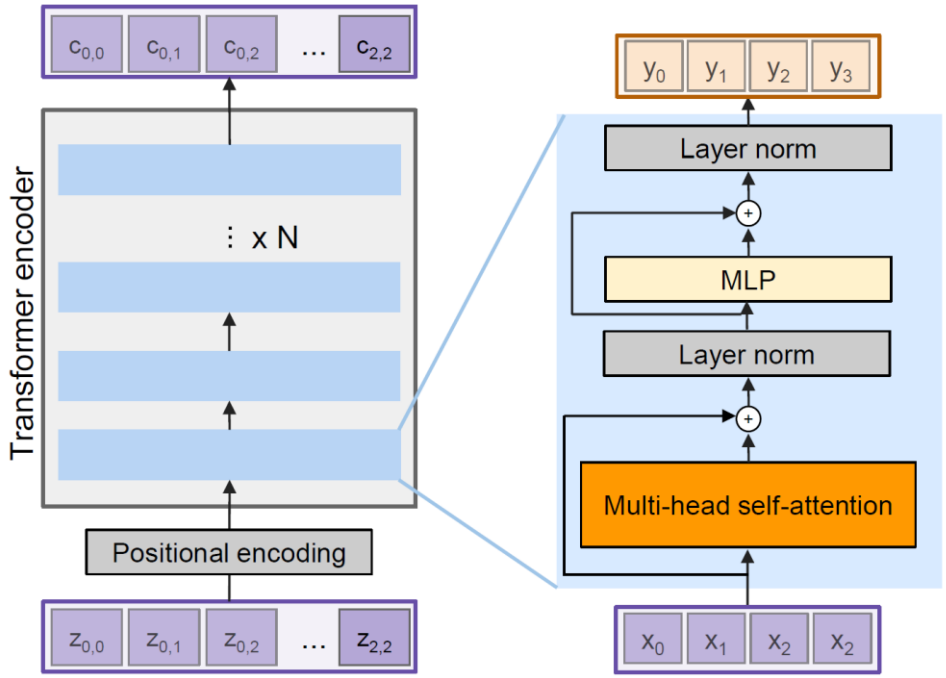
Vaswani et al, “Attention is all you need”, NeurIPS 2017

THE TRANSFORMER ENCODER BLOCK



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**

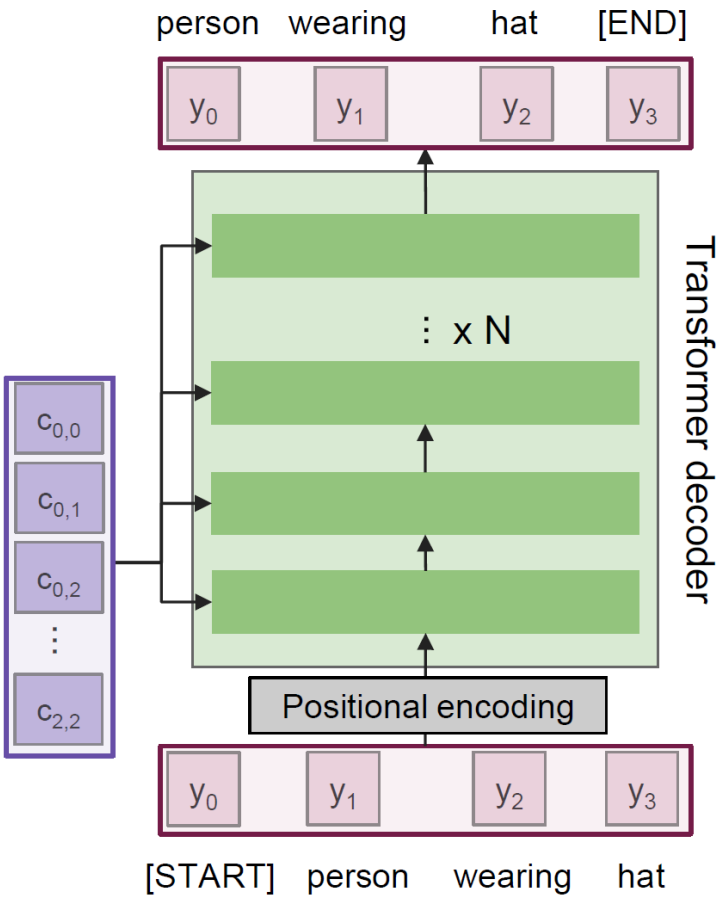
Self-attention is the only interaction between vectors.

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

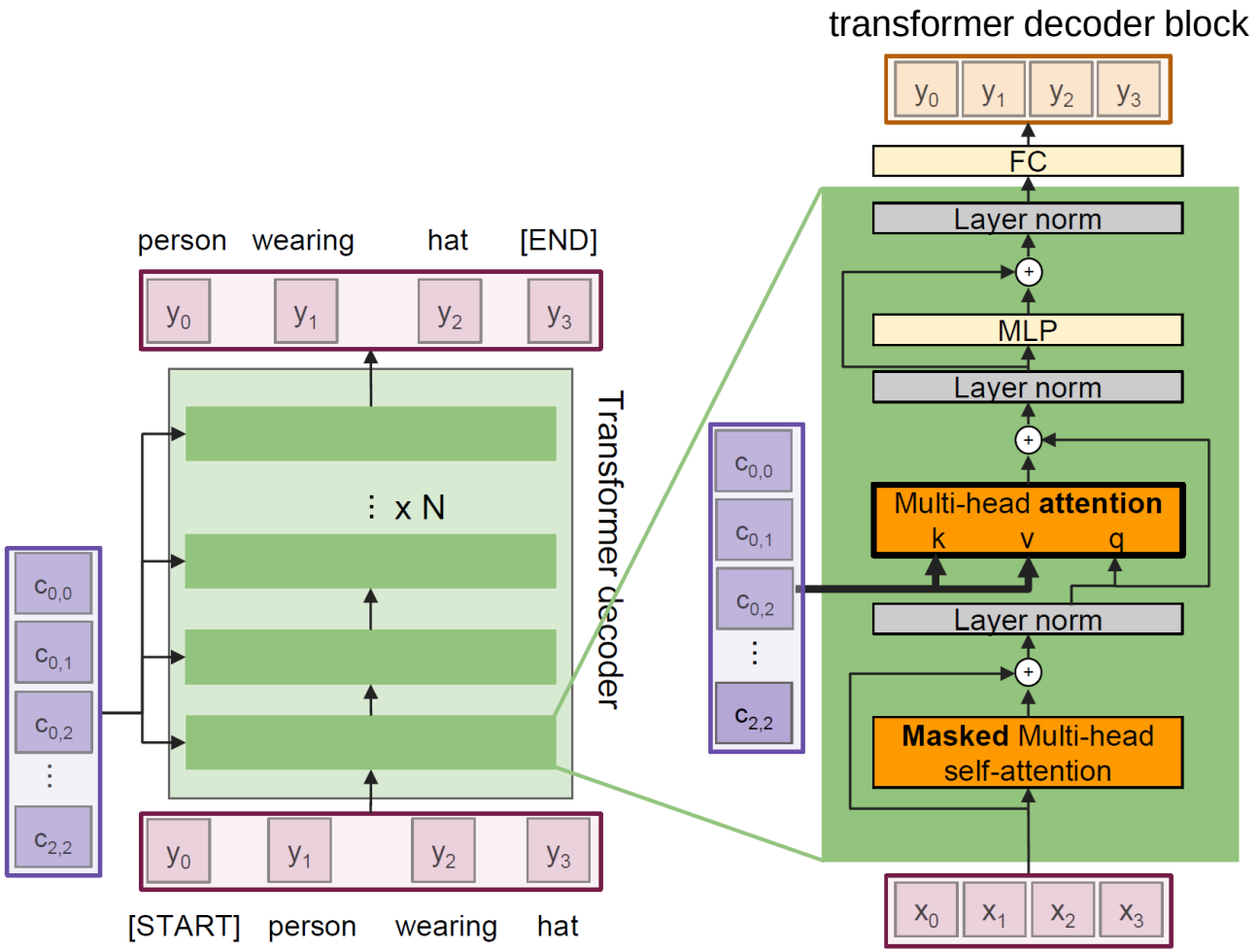
THE TRANSFORMER DECODER



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

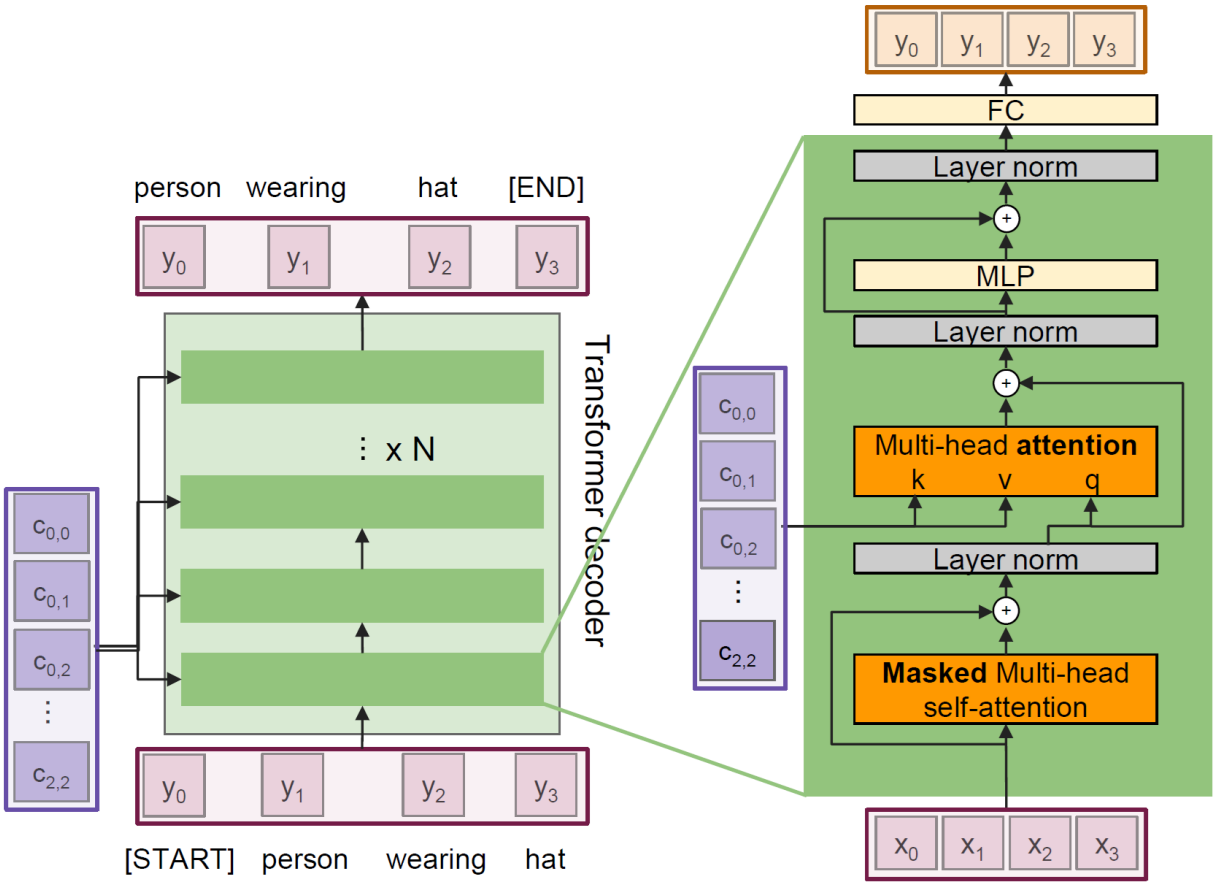


Multi-head attention block attends over the transformer encoder outputs.

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.**

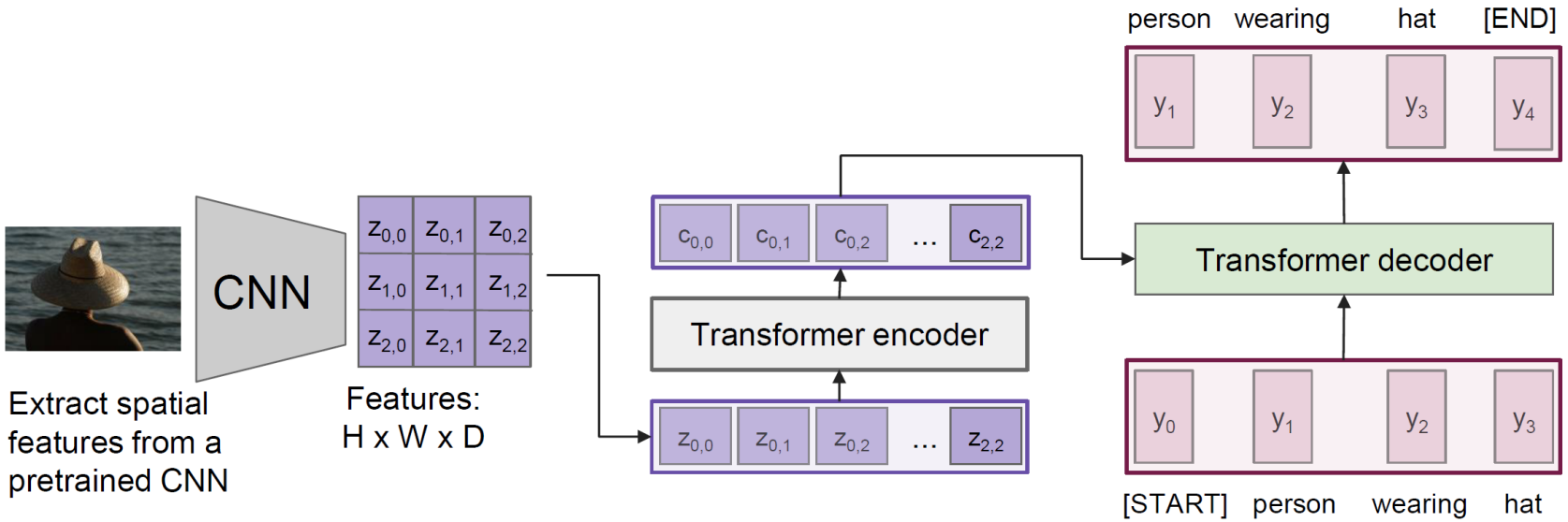
Masked Self-attention only interacts with past inputs.

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

Highly scalable, highly parallelizable, but high memory usage.

IMAGE CAPTIONING USING TRANSFORMERS

- **No recurrence at all**



Colab link:
vit_jax.ipynb - Colab

IMAGE CAPTIONING USING TRANSFORMERS

- Perhaps we don't need convolutions at all?

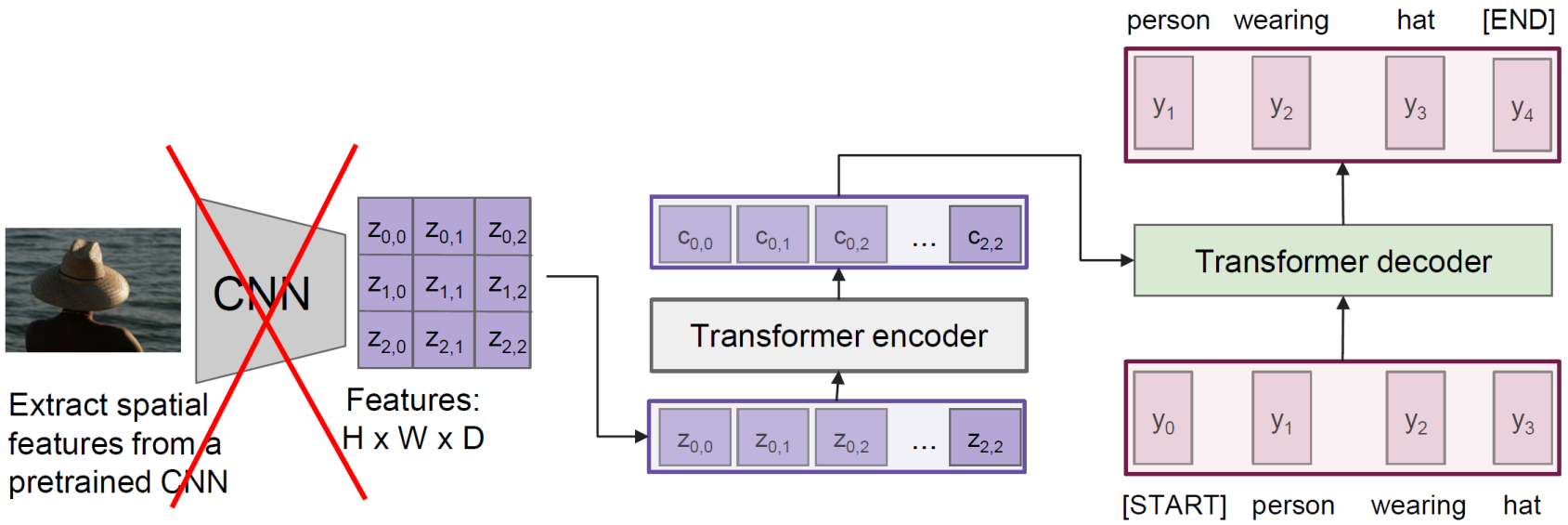
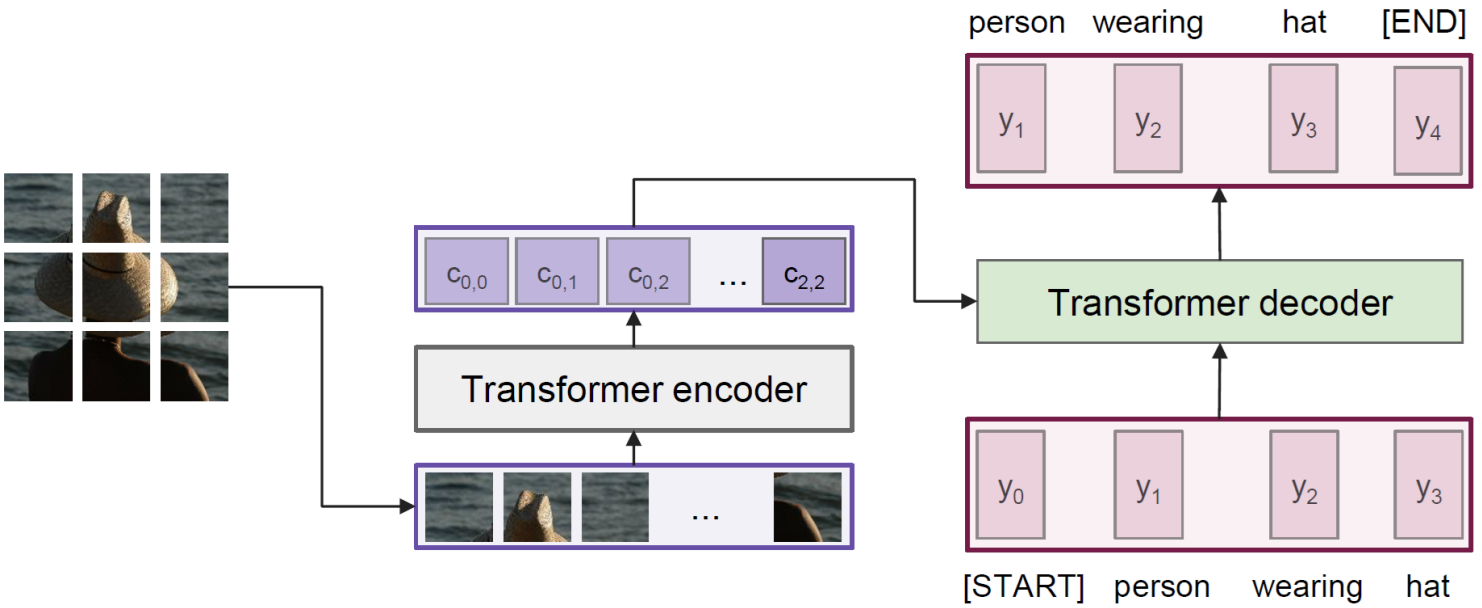


IMAGE CAPTIONING USING ONLY TRANSFORMERS

- **Transformers from pixels to language**



Dosovitskiy et al,"An Image is Worth16x16 Words: Transformers for Image Recognition at Scale", ArXiv 2020

VITS—VISION TRANSFORMERS

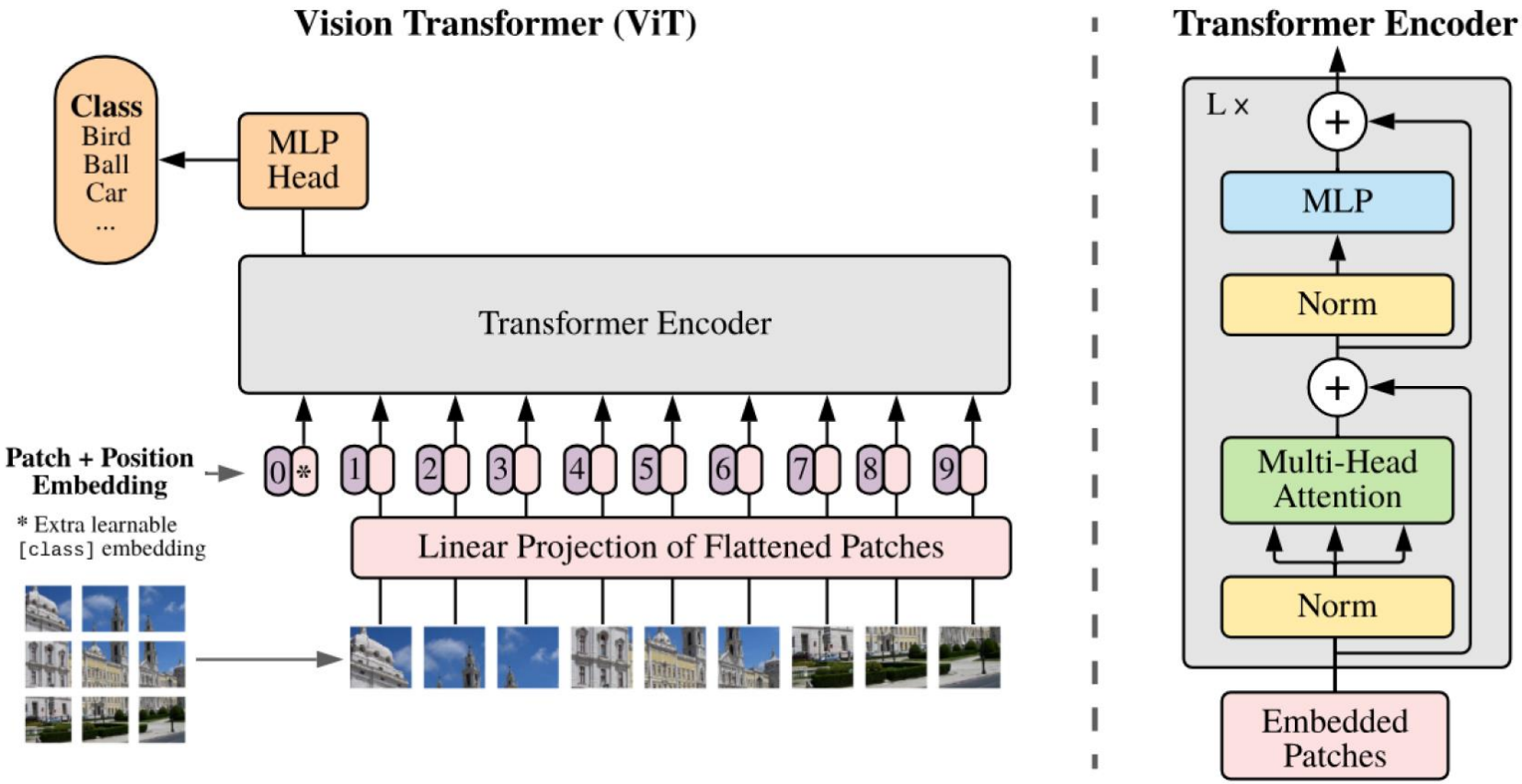


Figure from:
Dosovitskiy et al, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, ArXiv 2020

VISION TRANSFORMERS VS. RESNETS

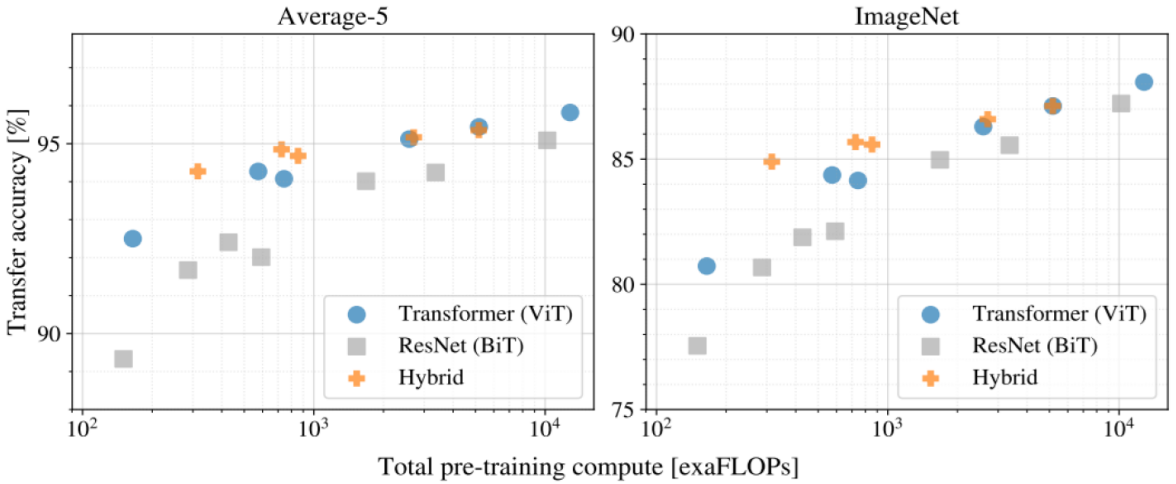


Figure 5: Performance versus cost for different architectures: Vision Transformers, ResNets, and hybrids. Vision Transformers generally outperform ResNets with the same computational budget. Hybrids improve upon pure Transformers for smaller model sizes, but the gap vanishes for larger models.

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

SUMMARY OF TRANSFORMER

➢ **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.