Deep Learning

Lecture 4: Transformer

National University of Singapore
CEG5304/EE6934
Instructor: Joey Tianyi Zhou

Outline

NLP

- Contextualized Word Embeddings
- Subword Tokenization
- Byte-Pair Encoding
- GPT, BERT, RoBERTa

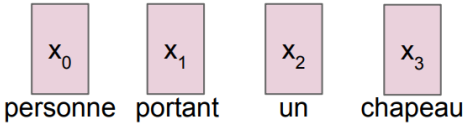
ML

- Transformer
- Self Attention

Recall: RNN Encoder-Decoder; Attention; Self Attention

Language translation example (Attention with RNNs)

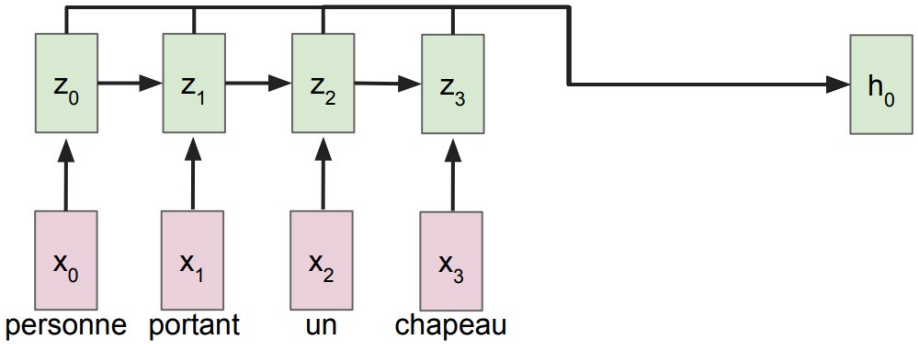
Input: Sequence **x** = $x_1, x_2, \ldots, x_T$
Output: Sequence **y** = $y_1, y_2, \ldots, y_T$



Recall: RNN Encoder-Decoder; Attention; Self Attention

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

Encoder: $h_0 = f_{\mathbf{W}}(\mathbf{z})$
where $z_t = \mathrm{RNN}(x_t, u_{t-1})$
$f_{\mathbf{W}}(.)$ is MLP
u is the hidden RNN state

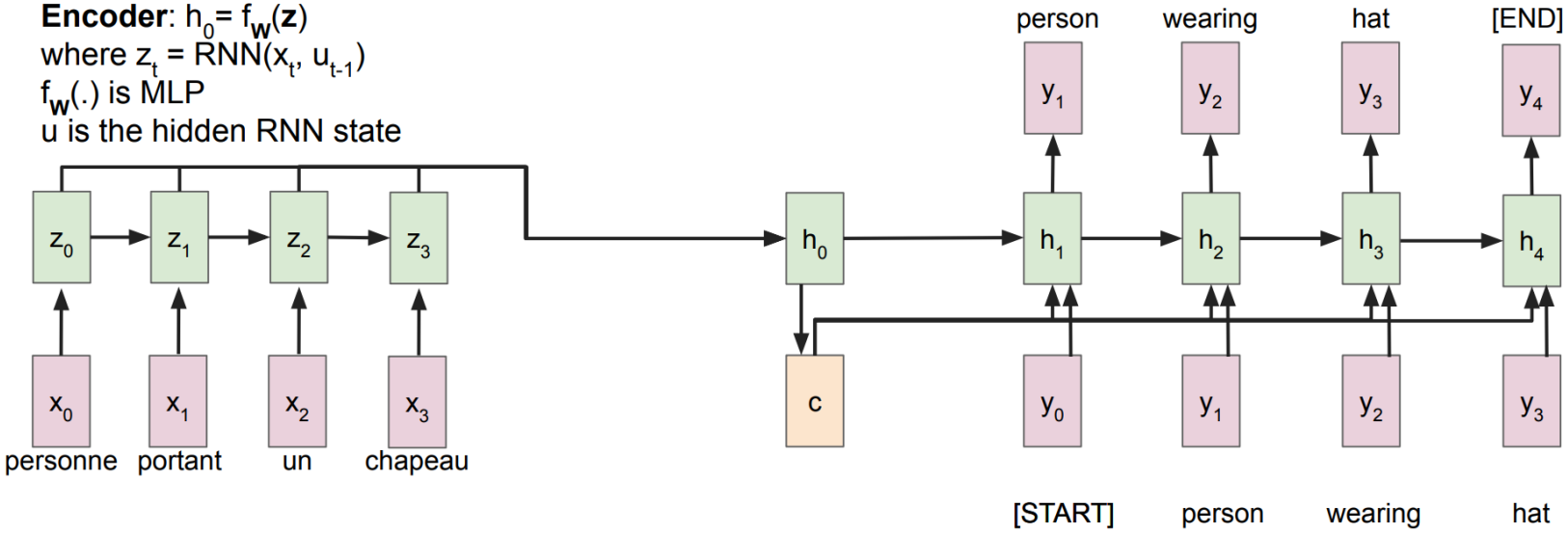


Recall: RNN Encoder-Decoder; Attention; Self Attention

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

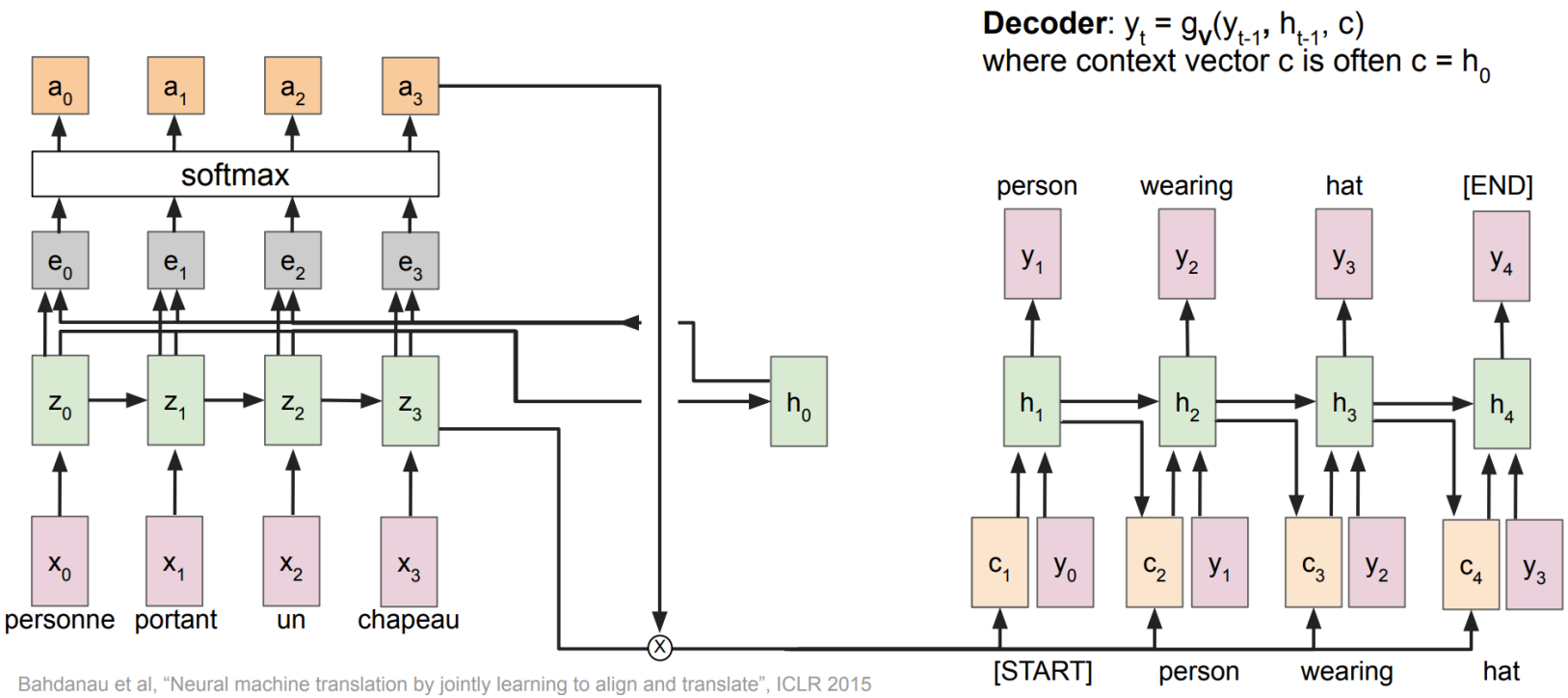
Decoder: $y_t = g_{\mathbf{V}}(y_{t-1}, h_{t-1}, c)$
where context vector c is often $c = h_0$

Encoder: $h_0 = f_{\mathbf{W}}(\mathbf{z})$
where $z_t = \mathrm{RNN}(x_t, u_{t-1})$
$f_{\mathbf{W}}(.)$ is MLP
u is the hidden RNN state



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

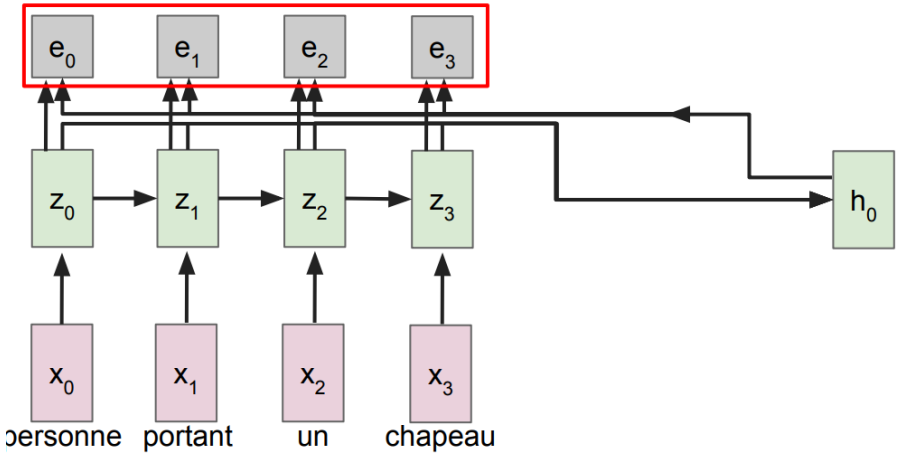
Attention with RNNs



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

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

Recall: RNN Encoder-Decoder; Attention;



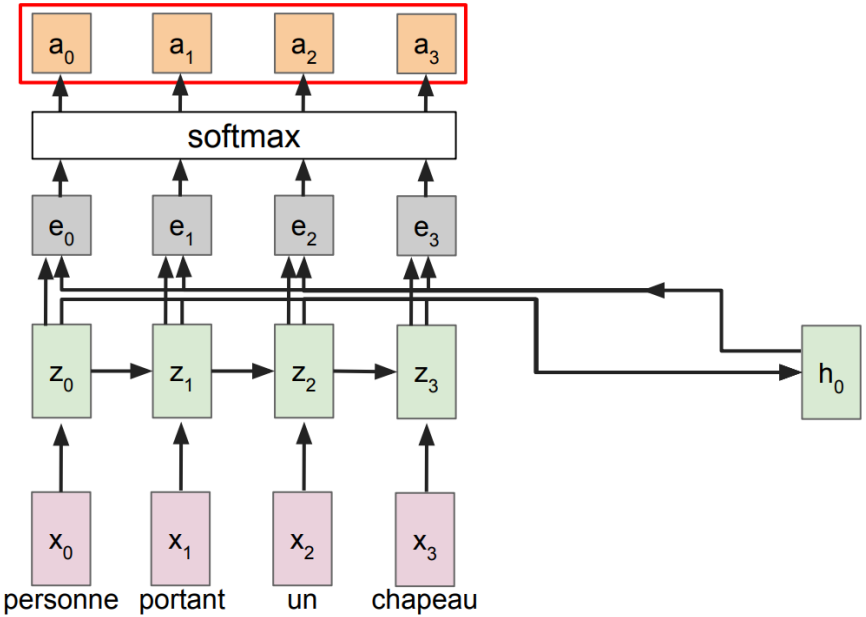
Compute alignments scores (scalars):

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

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

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

Attention with RNNs



Compute alignments scores (scalars):

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

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

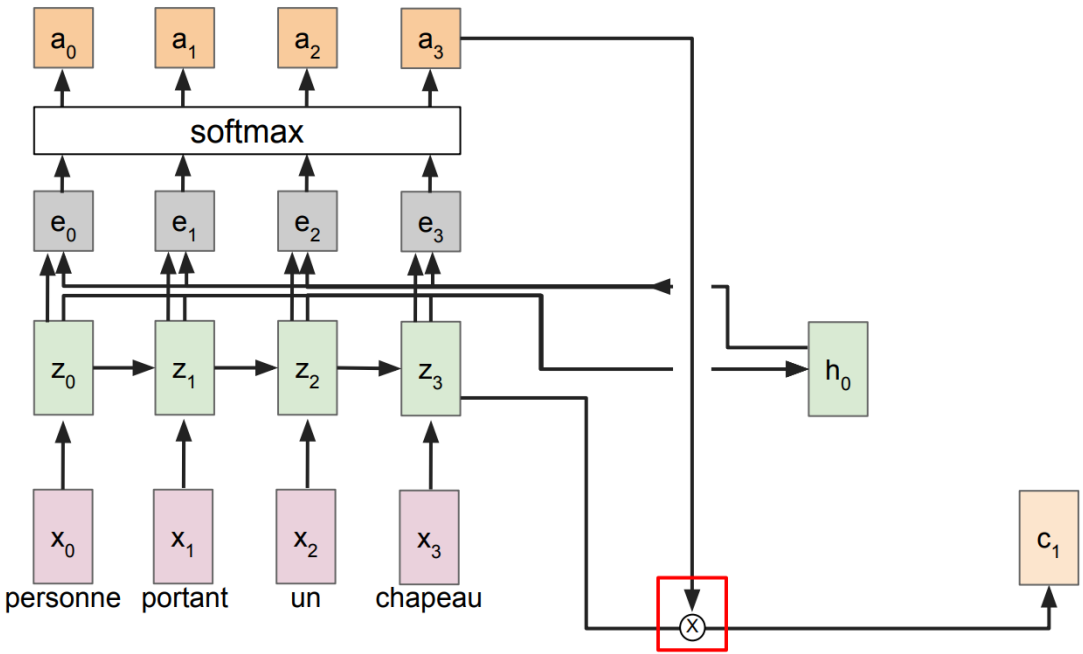
Normalize to get attention weights:

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

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

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

Attention with RNNs



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

Compute alignments scores (scalars):

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

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

Normalize to get attention weights:

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

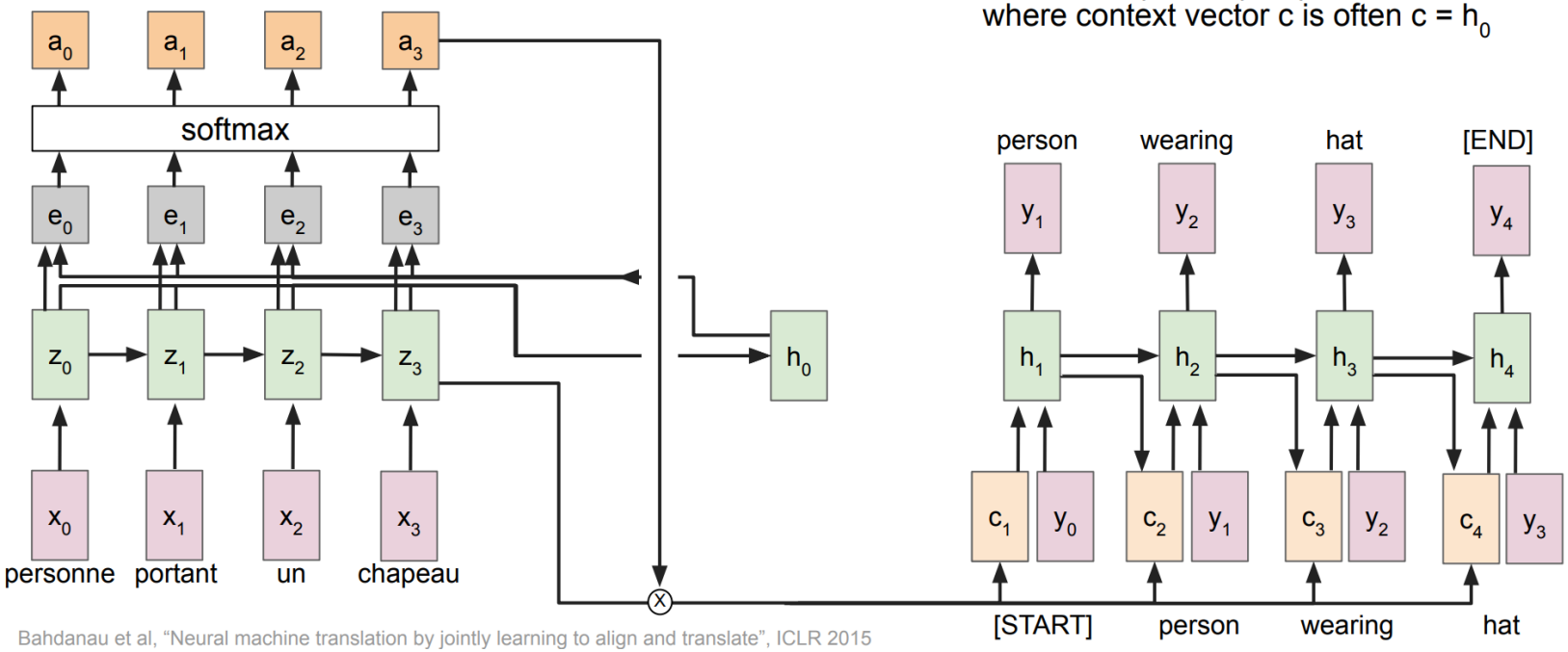
$0 < a_{t,i,j} < 1$,
attention values sum to 1

Compute context vector:

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

Attention with RNNs

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



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

Intuition: Context vector attends to the relevant part of the input sequence. At each timestep of decoder, context vector "looks at" different parts of the input sequence. Repeat: Use h_1 to compute new context vector c_2.

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

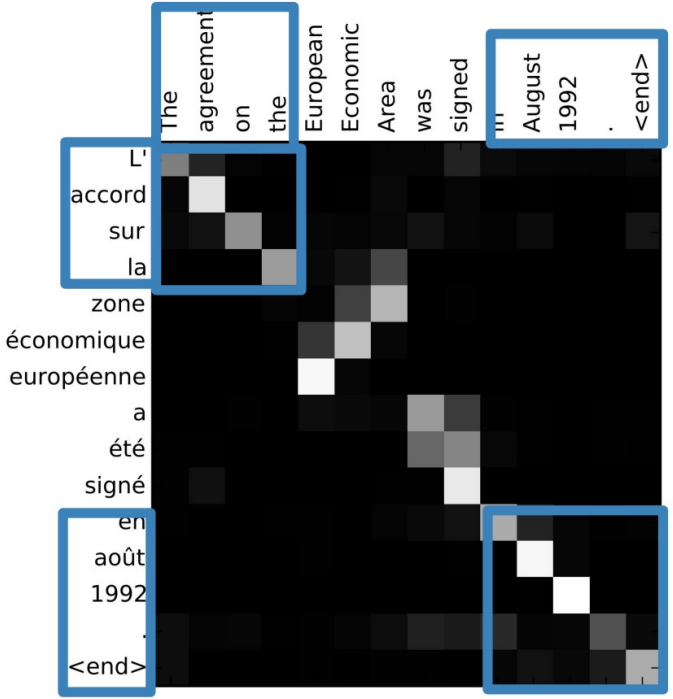
Sequence to Sequence with RNNs and Attention

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



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

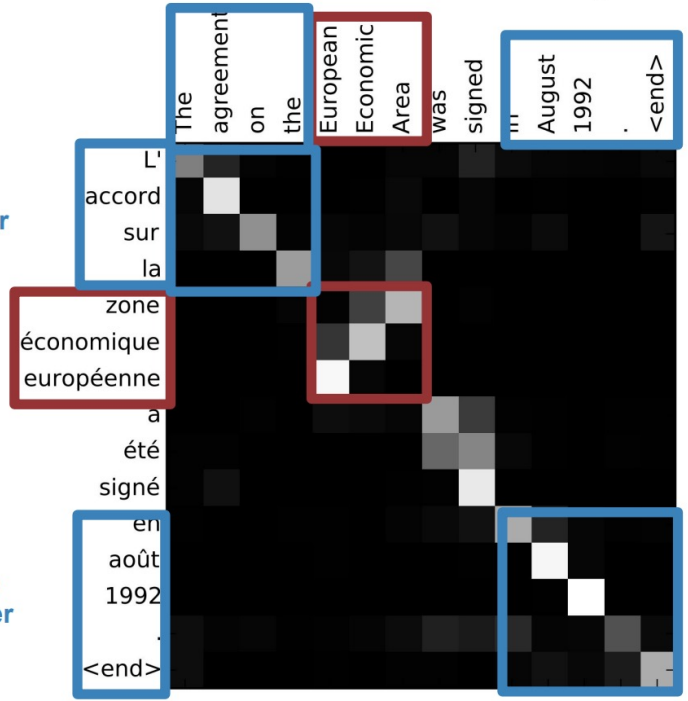
Sequence to Sequence with RNNs and Attention

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

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

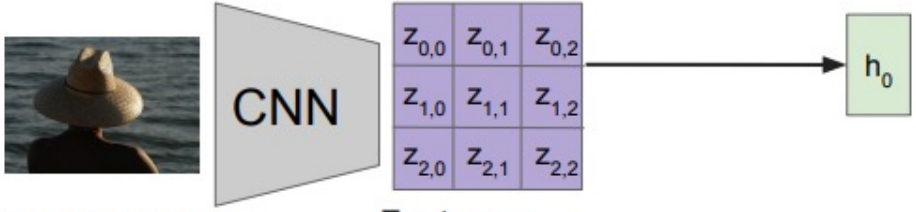
Features: H x W x D

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

Attention can be used in computer vision tasks– Image Captioning

Attention idea: New context vector at every time step.

Each context vector will attend to different image regions

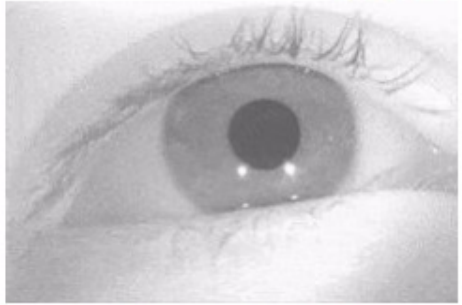


Extract spatial features from a pretrained CNN

Features: H x W x D

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

gif source



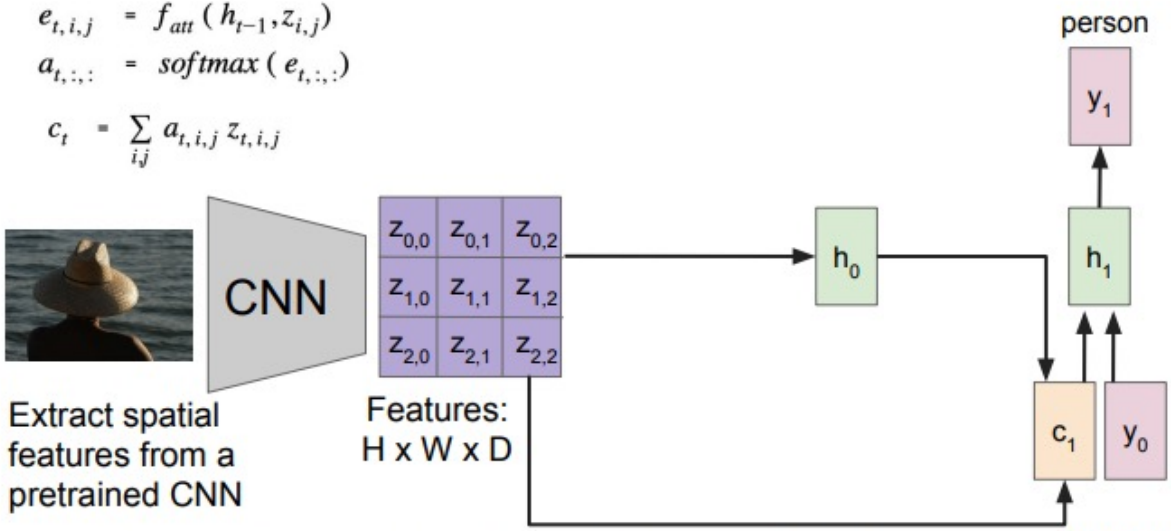
Attention Saccades in humans

Image Captioning with RNNs & Attention

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

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

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



Extract spatial features from a pretrained CNN

Features: H x W x D

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

Image Captioning with RNNs & Attention

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

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

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

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

y_1 y_2

person wearing

c_1 y_0 c_2 y_1

[START] person

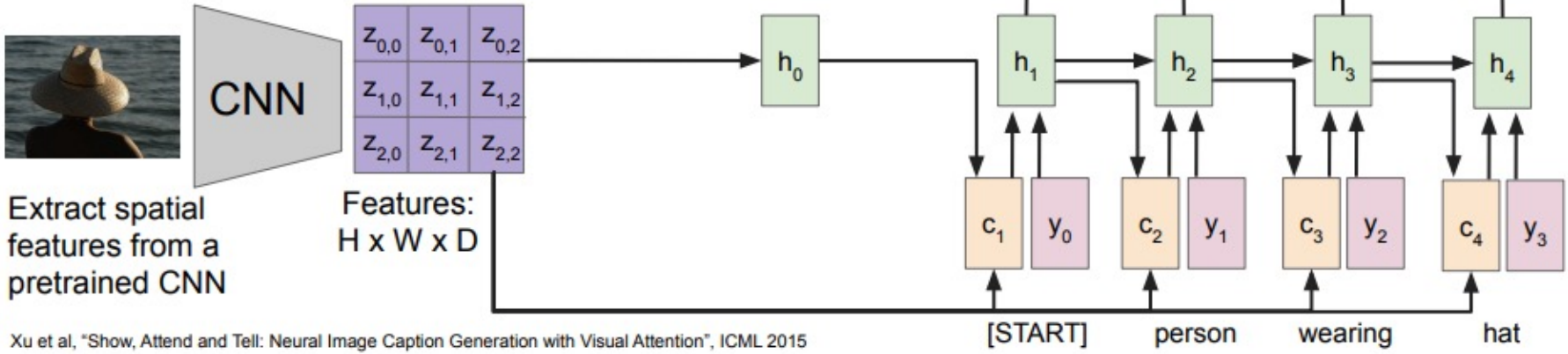
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with RNNs & 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

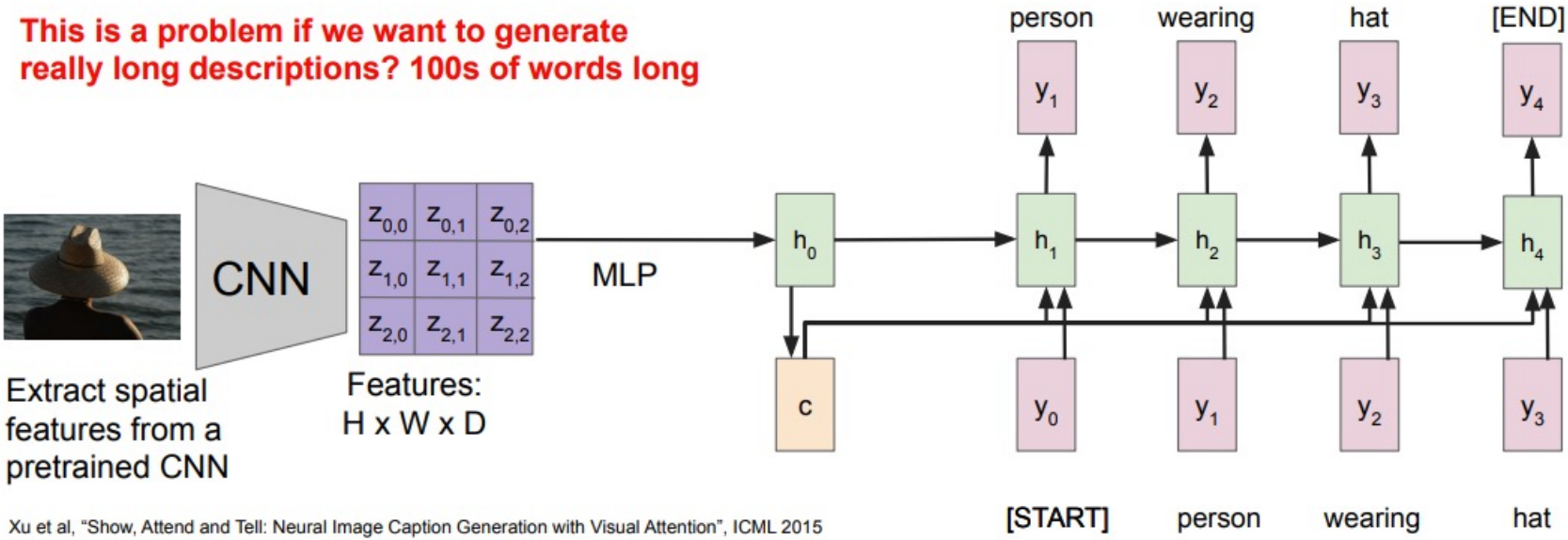
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning 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



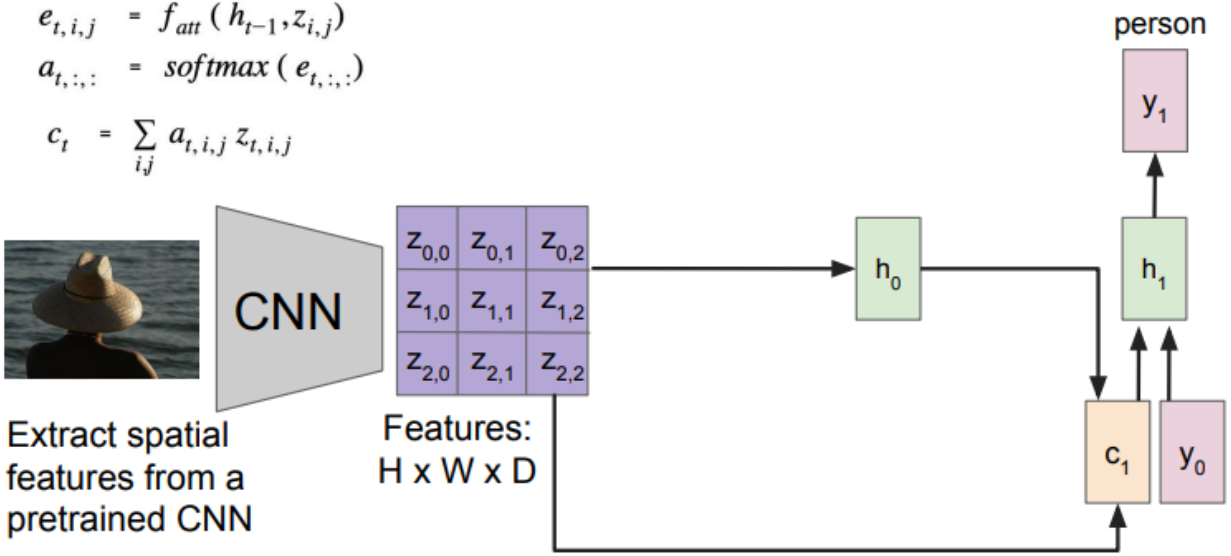
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with RNNs and Attention

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

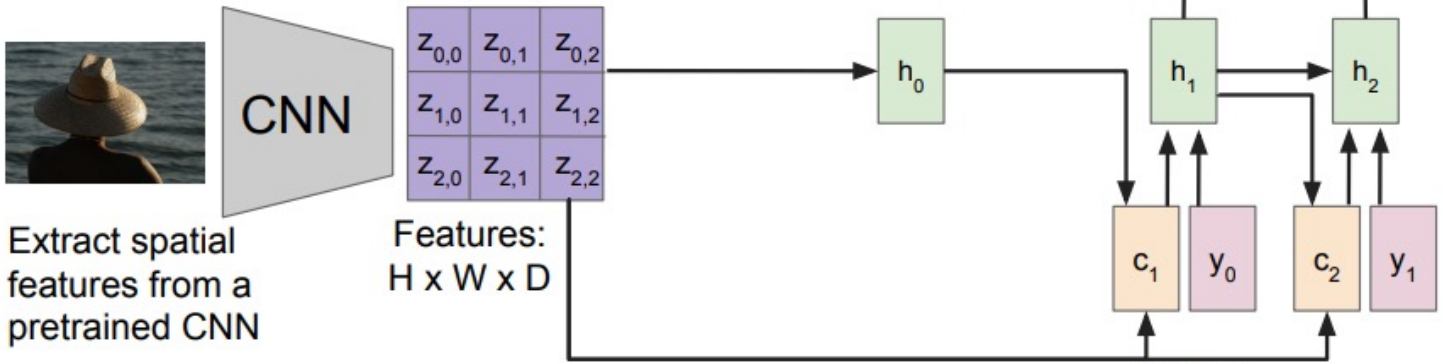
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with RNNs and Attention

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

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

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

CNN

Extract spatial features from a pretrained CNN

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

Features: H x W x D

h_0

c_1 y_0 → h_1 → y_1 (person)
c_2 y_1 → h_2 → y_2 (wearing)
c_3 y_2 → h_3 → y_3 (hat)
c_4 y_3 → h_4 → y_4 ([END])

[START] person wearing hat

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

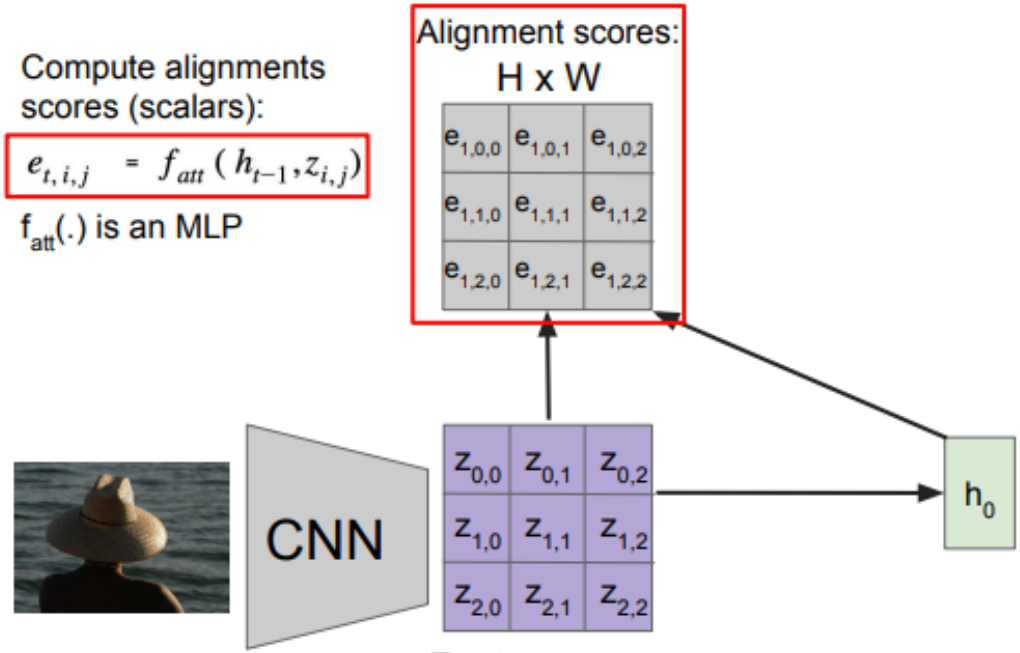
How to generate different C_t?

Image Captioning with RNNs & Attention

Compute alignments scores (scalars):

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

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



Extract spatial features from a pretrained CNN

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

Image Captioning with RNNs and Attention

Compute alignments scores (scalars):

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

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

Alignment scores:
H x W

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

Attention:
H x W

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

Normalize to get attention weights:

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

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

CNN

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

h_0

Extract spatial features from a pretrained CNN

Features:
H x W x D

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

Image Captioning with RNNs & Attention

Compute alignments scores (scalars):

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

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

Alignment scores: H x W

Attention: H x W

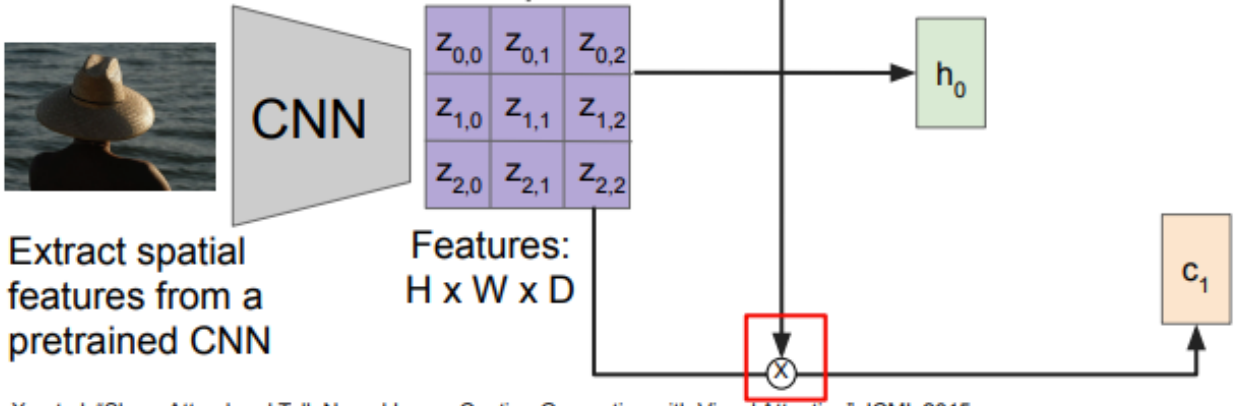
Normalize to get attention weights:

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

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

Compute context vector:

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



Extract spatial features from a pretrained CNN

Features: H x W x D

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

Image Captioning with RNNs & Attention

Compute alignments scores (scalars):

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

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

CNN

Extract spatial features from a pretrained CNN

Features: H x W x D

$$\begin{matrix} z_{0,0} & z_{0,1} & z_{0,2} \\ z_{1,0} & z_{1,1} & z_{1,2} \\ z_{2,0} & z_{2,1} & z_{2,2} \end{matrix}$$

h_0

Alignment scores: H x W

$$\begin{matrix} e_{1,0,0} & e_{1,0,1} & e_{1,0,2} \\ e_{1,1,0} & e_{1,1,1} & e_{1,1,2} \\ e_{1,2,0} & e_{1,2,1} & e_{1,2,2} \end{matrix}$$

Attention: H x W

$$\begin{matrix} a_{1,0,0} & a_{1,0,1} & a_{1,0,2} \\ a_{1,1,0} & a_{1,1,1} & a_{1,1,2} \\ a_{1,2,0} & a_{1,2,1} & a_{1,2,2} \end{matrix}$$

Normalize to get attention weights:

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

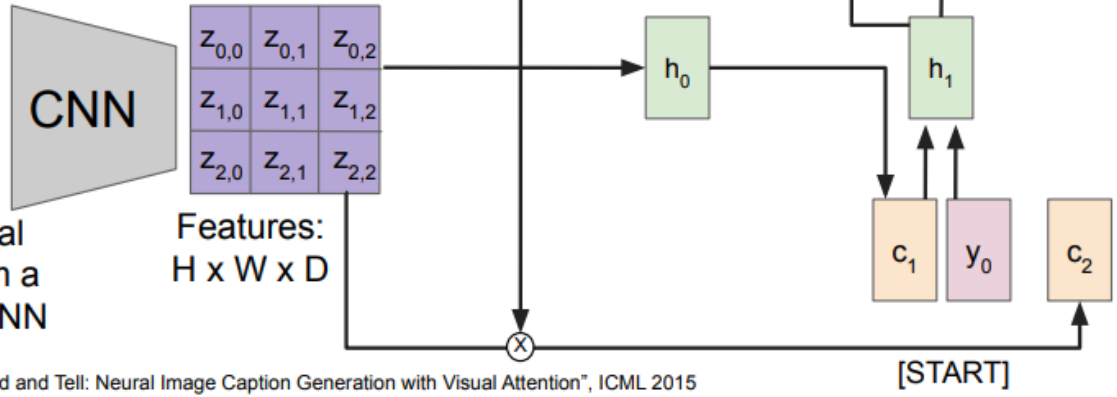
$0 < a_{t,i,j} < 1$, attention values sum to 1

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

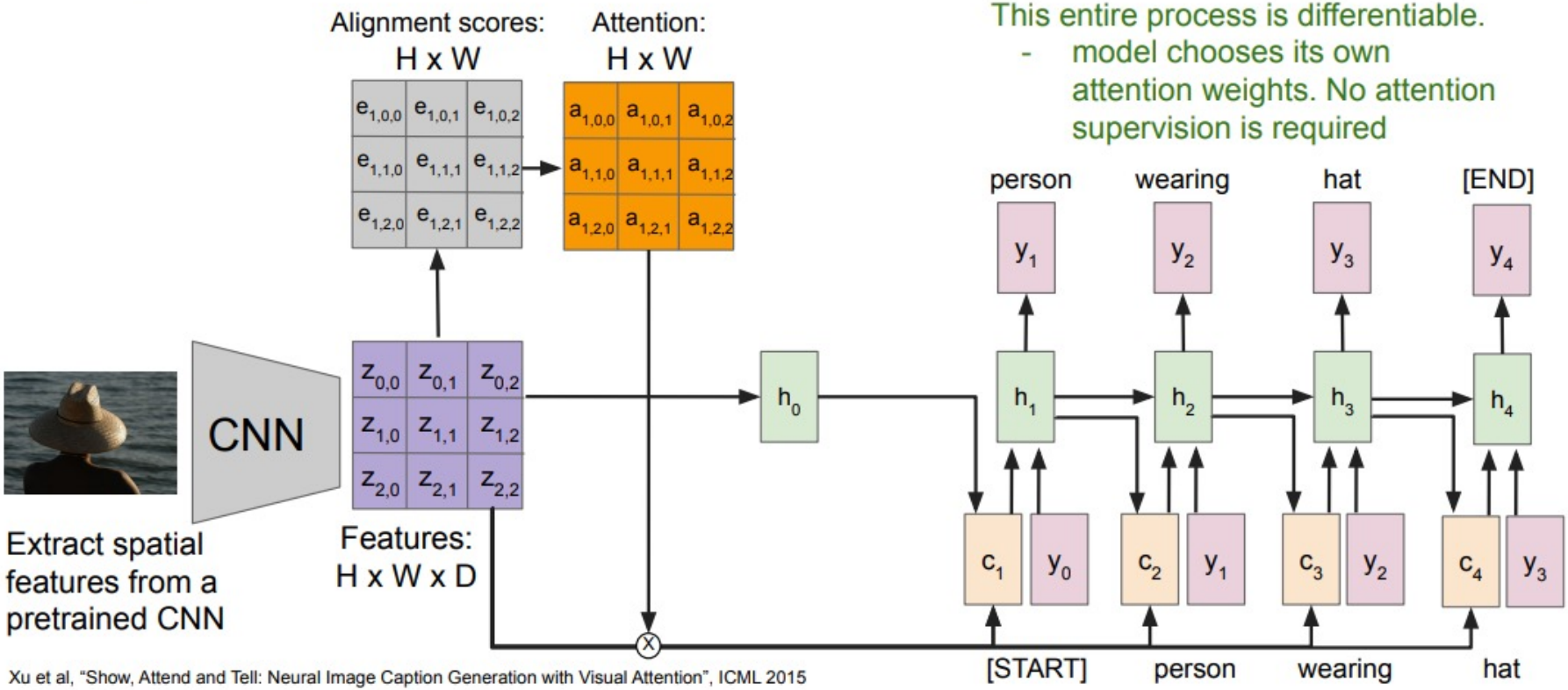
$$e_{t,i,j} = f_{att}(h_{t-1}, z_{i,j})$$
$$a_{t,:,:} = softmax(e_{t,:,:})$$
$$c_t = \sum_{i,j} a_{t,i,j} z_{t,i,j}$$



Extract spatial features from a pretrained CNN

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

Image Captioning with RNNs and Attention



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

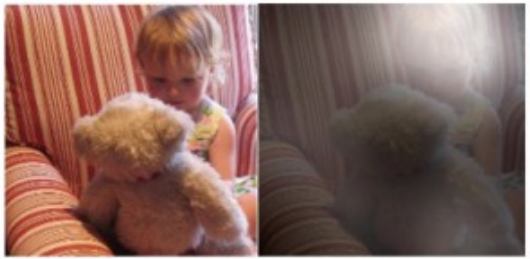
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with Attention

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

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



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

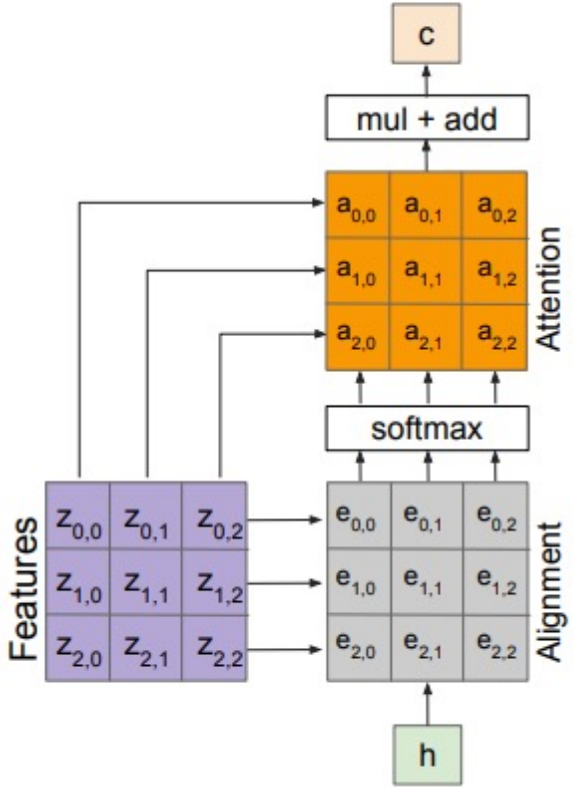
A group of people sitting on a boat in the water.

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

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

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

Attention we just saw in image captioning

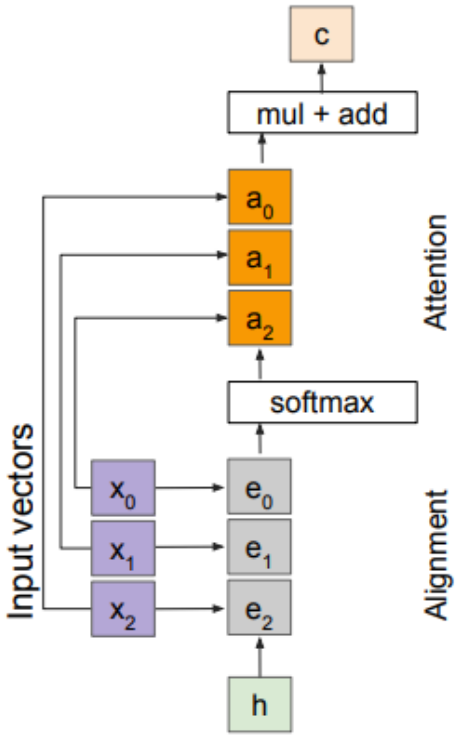


Outputs:
context vector: **c** (shape: D)

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)

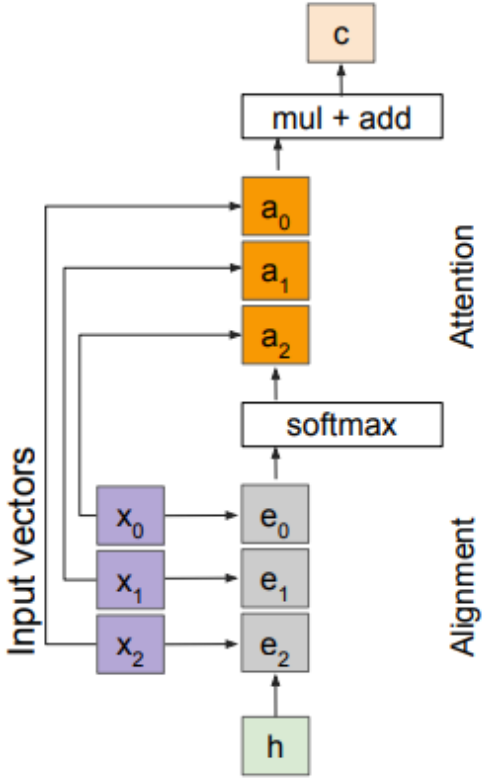
Operations:
Alignment: $e_i = f_{att}(h, x_i)$
Attention: **a** = softmax(**e**)
Output: $\mathbf{c} = \sum_i a_i x_i$

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

Attention operation is **permutation invariant.**

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

General attention layer



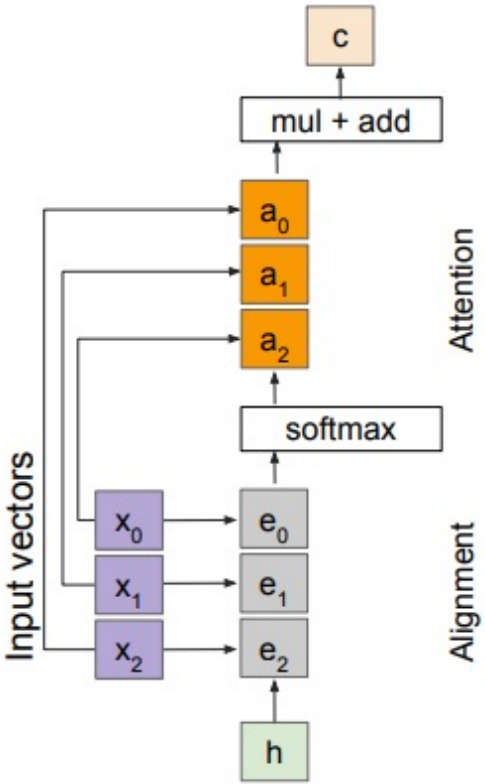
Outputs:
context vector: **c** (shape: D)

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

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

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

General attention layer



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

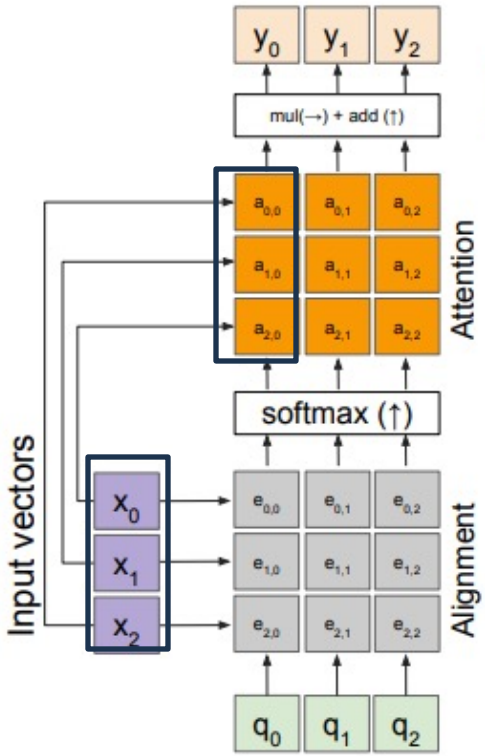
Operations:
Alignment: $e_i = h \cdot x_i / \sqrt{D}$
Attention: **a** = softmax(**e**)
Output: $\mathbf{c} = \sum_i a_i x_i$

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

General attention layer



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

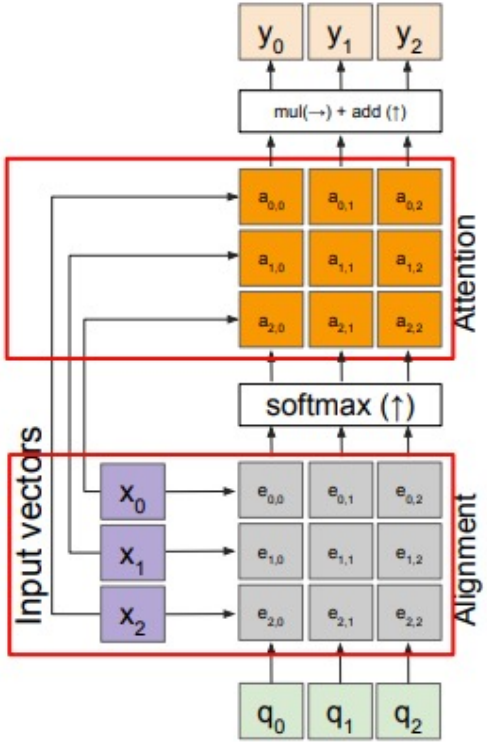
Multiple query vectors
- each query creates a new output context vector

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

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

Multiple query vectors

General attention layer



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

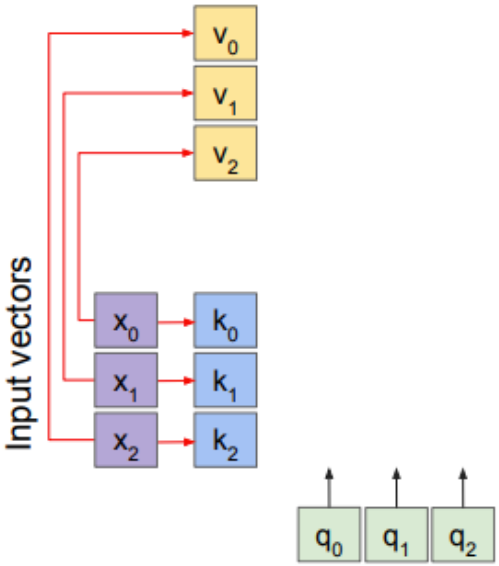
Operations:
Alignment: $e_{i,j} = q_j \cdot x_i / \sqrt{D}$
Attention: **a** = softmax(**e**)
Output: $y_j = \sum_i a_{i,j} x_i$

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



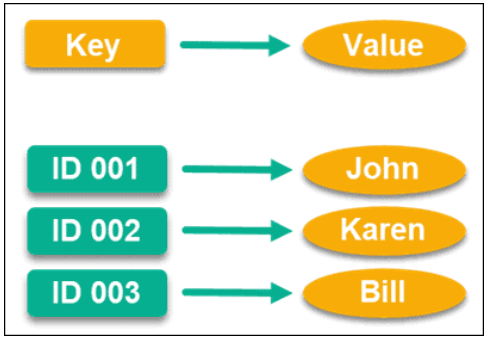
Operations:
Key vectors: $\mathbf{k} = \mathbf{x}\mathbf{W}_k$
Value vectors: $\mathbf{v} = \mathbf{x}\mathbf{W}_v$

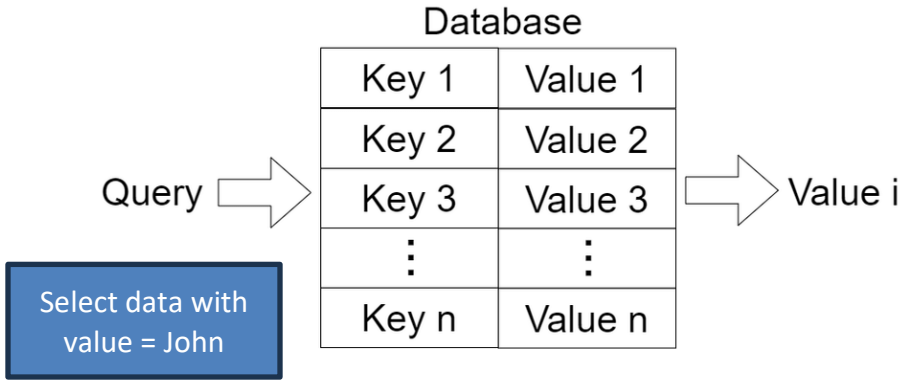
Inputs:
Input vectors: **x** (shape: N x D)
Queries: **q** (shape: M x D_k)

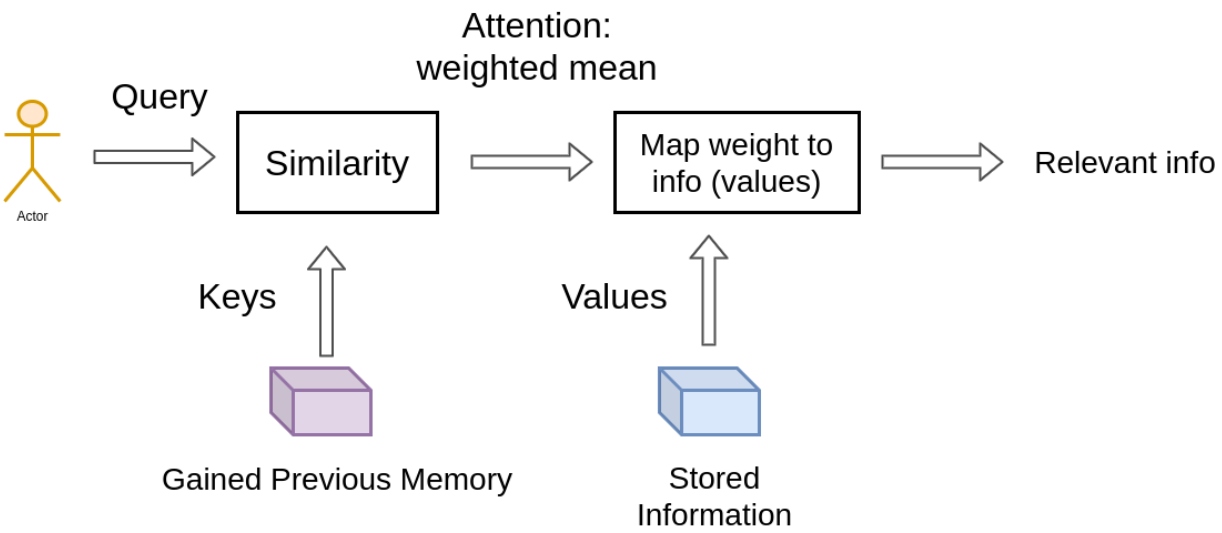
Notice that the input vectors are used for both the alignment as well as the attention calculations.

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

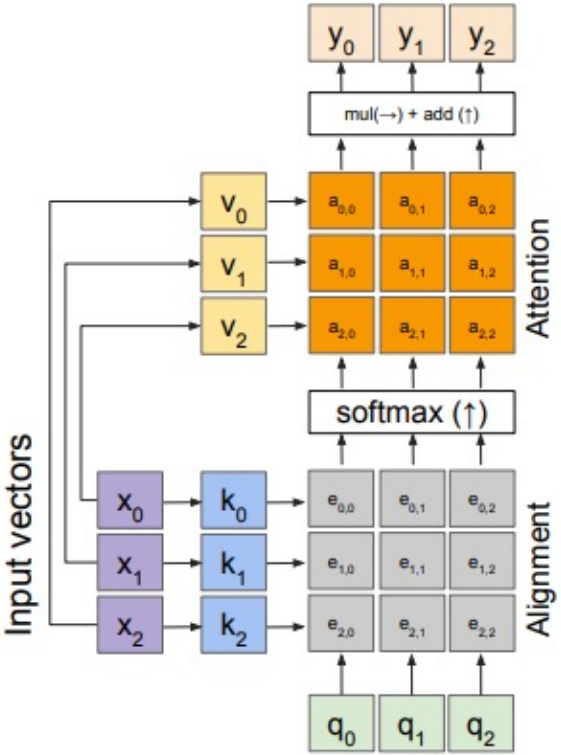
Attention Connecting to Database







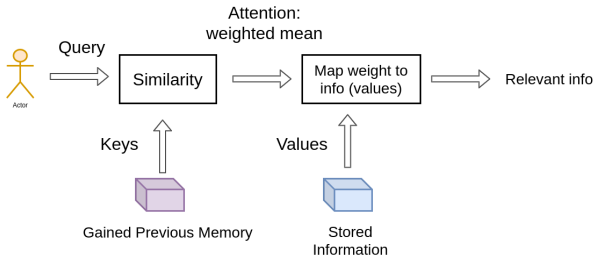
General attention layer



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

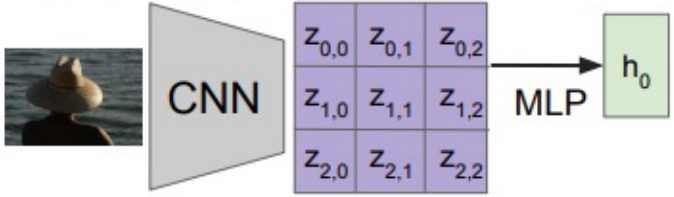
Operations:
Key vectors: $\mathbf{k} = \mathbf{x}\mathbf{W}_k$
Value vectors: $\mathbf{v} = \mathbf{x}\mathbf{W}_v$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $\mathbf{a} = \mathrm{softmax}(\mathbf{e})$
Output: $y_j = \sum_i a_{i,j} v_i$

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

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

Operations:

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

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

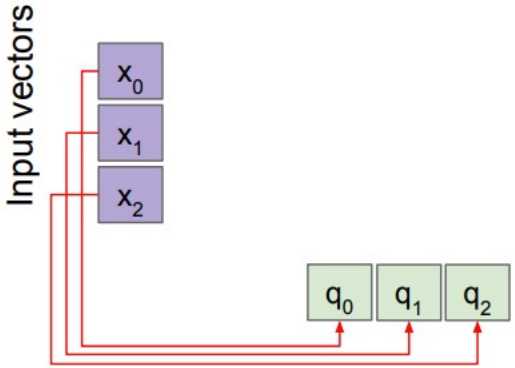
Query vectors: $\mathbf{q} = \mathbf{x}\mathbf{W}_q$

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

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

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

Instead, query vectors are calculated using a FC layer.



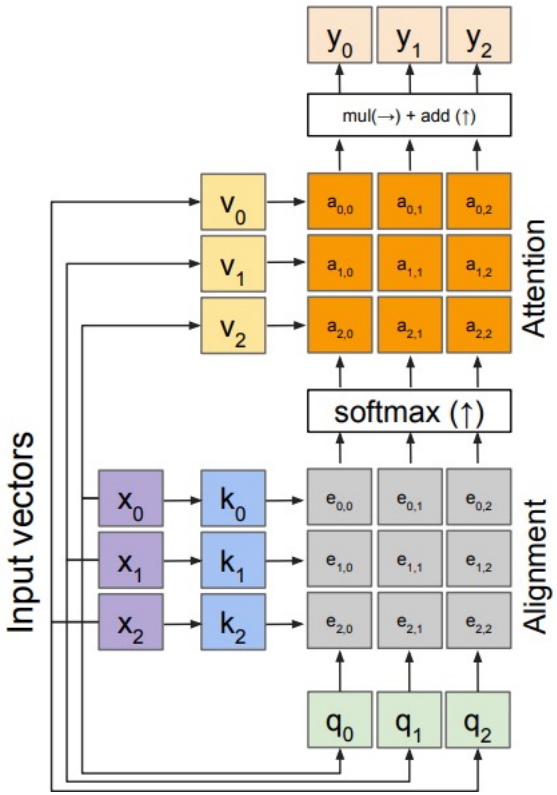
No input query vectors anymore

Inputs:

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

~~Queries: $\mathbf{q}$ (shape: M x D_K)~~

Self attention layer

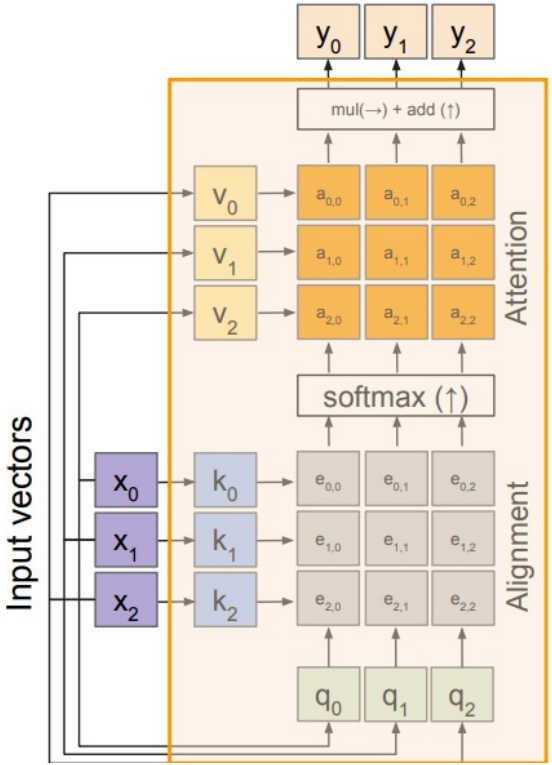


Outputs:
context vectors: **y** (shape: D_v)

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)

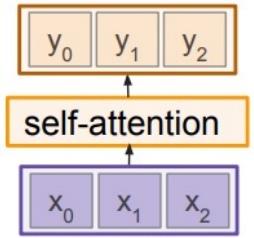
Self attention layer - attends over sets of inputs



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: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $y_j = \sum_i a_{i,j} v_i$

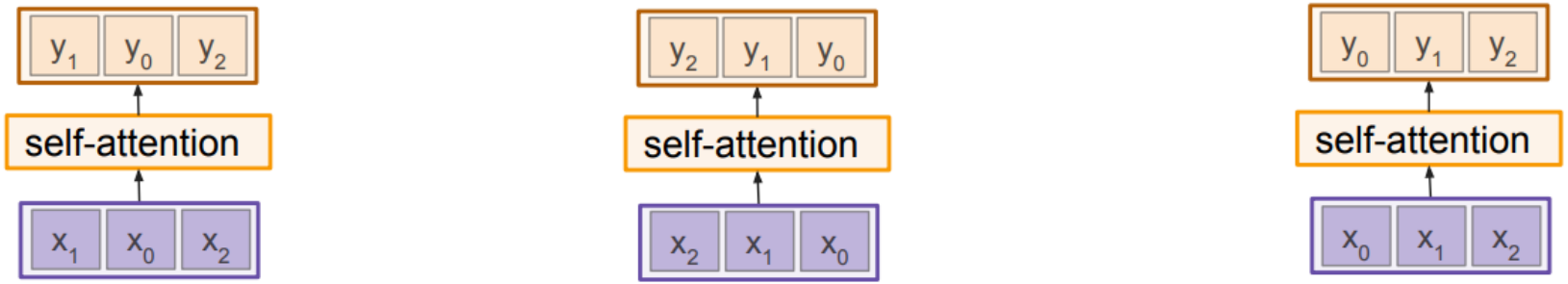
Inputs:
Input vectors: **x** (shape: N x D)



Self attention layer - attends over sets of inputs

Permutation invariant

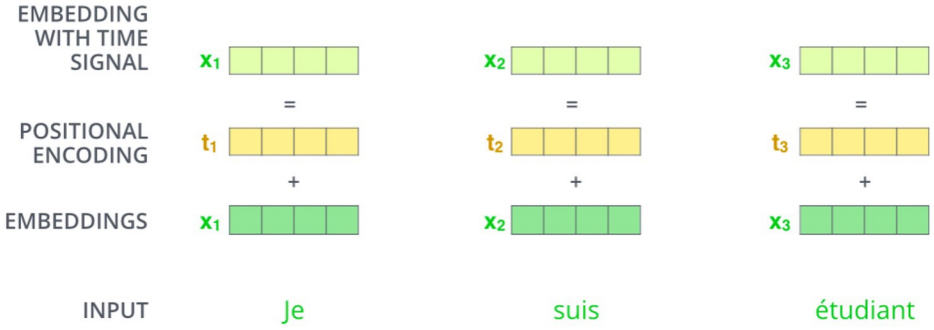
Problem: how can we encode ordered sequences like language or spatially ordered image features?

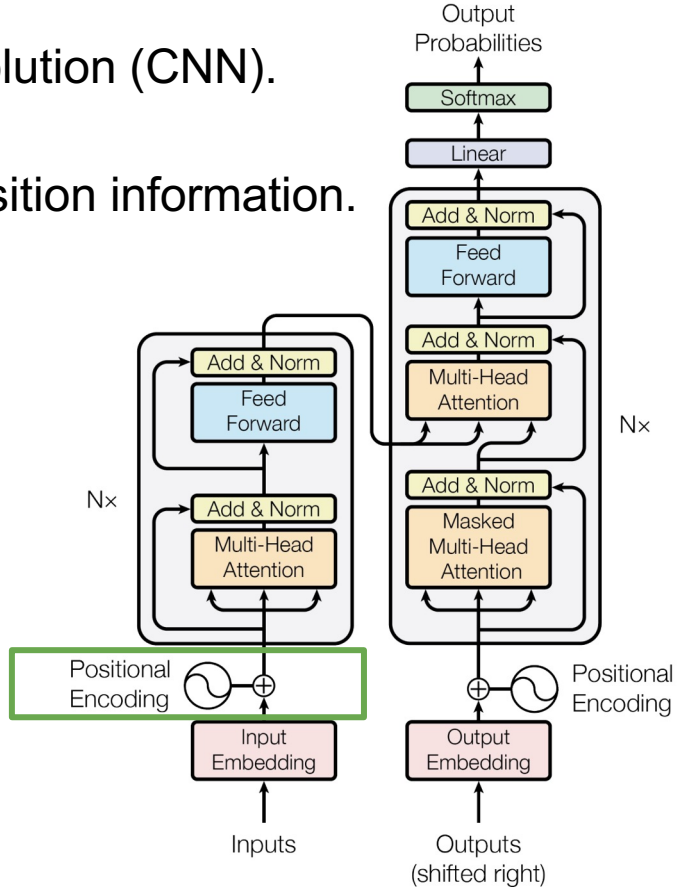


- There is no recurrence (RNN) and no convolution (CNN).
- Self-attention do not have order information.
- So we need positional encoding to bring position information.

Positional Encoding

- There is no recurrence (RNN) and no convolution (CNN).
- Self-attention do not have order information.
- So we need positional encoding to bring position information.





Why we need position embedding?

Aspect-based Sentiment Analysis

I like this movie because it doesn't have an overhead history. I don't like this movie because it has an overhead history.

Positive

Negative

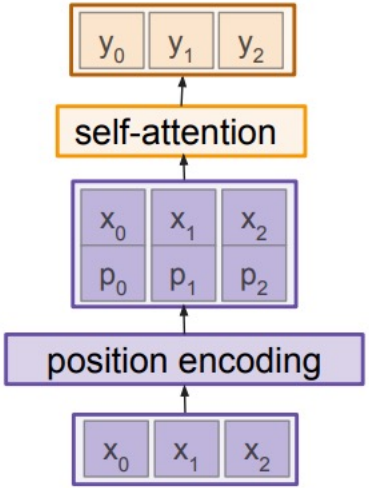
Named Entity Analysis

I believe I can be the best.

The subject who is making this statement

The object of this statement.

Positional encoding



Concatenate special positional encoding p_j to each input vector x_j

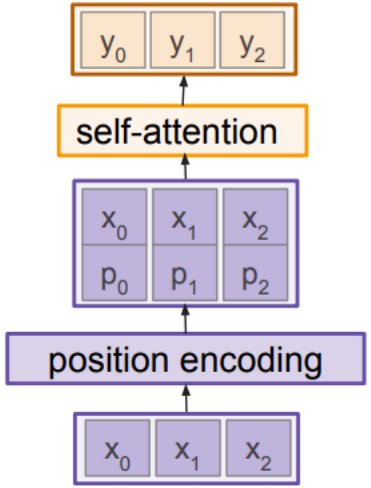
We use a function *pos*: $N \rightarrow R^d$ to process the position j of the vector into a d-dimensional vector

So, $p_j = pos(j)$

Desiderata of *pos*(.) :

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

Positional encoding



Concatenate special positional encoding p_j to each input vector x_j

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

So, $p_j = pos(j)$

Options for *pos*(.)

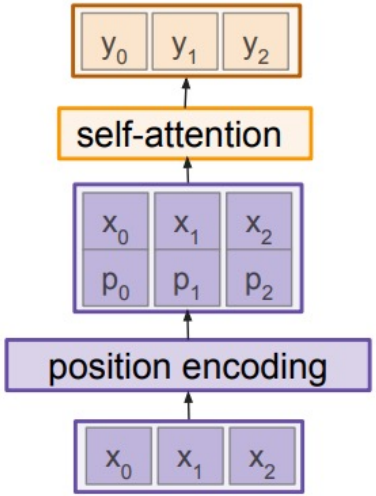
1. Learn a lookup table:
   - Learn parameters to use for *pos*(t) for $t \in [0, T)$
   - Lookup table contains T x d parameters.

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



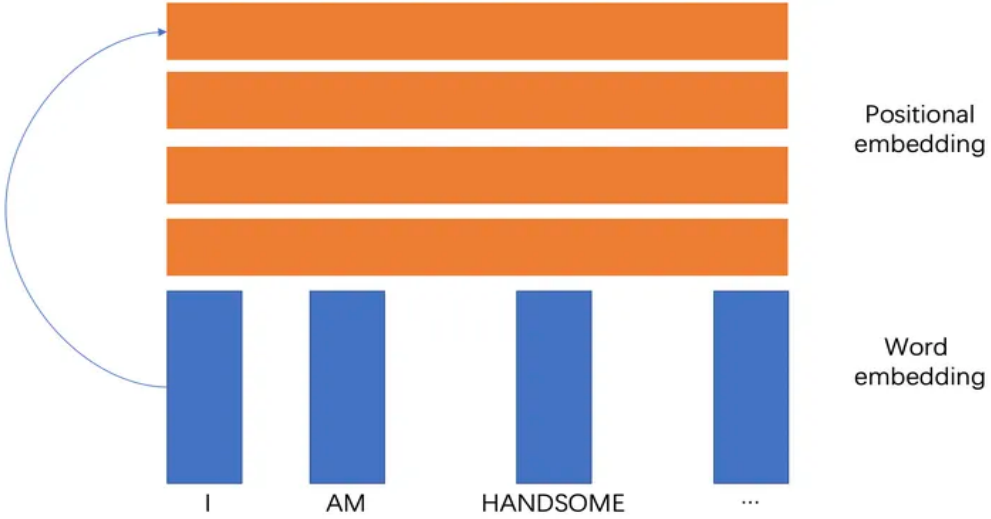
Concatenate special positional encoding p_j to each input vector x_j

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

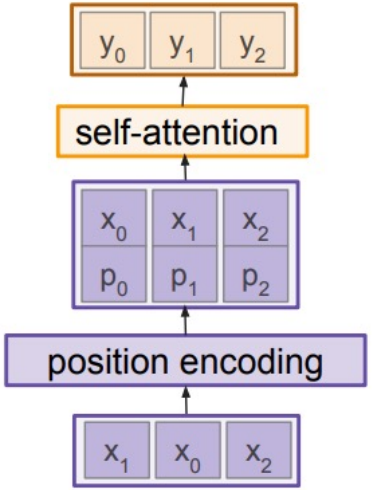
So, $p_j = pos(j)$

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.



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
   -

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

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

- In Transformer, we use **Sinusoidal position encodings**: fixed embeddings and can generalize to any sequence length.

Sinusoidal Positional Encoding

- In Transformer, we use **Sinusoidal position encodings**: fixed embeddings and can generalize to any sequence length.
- Intuition: (4-dimensional position encodings for 16 positions)

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

https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

Sinusoidal Positional Encoding

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

4-dimensional position encodings for 16 positions

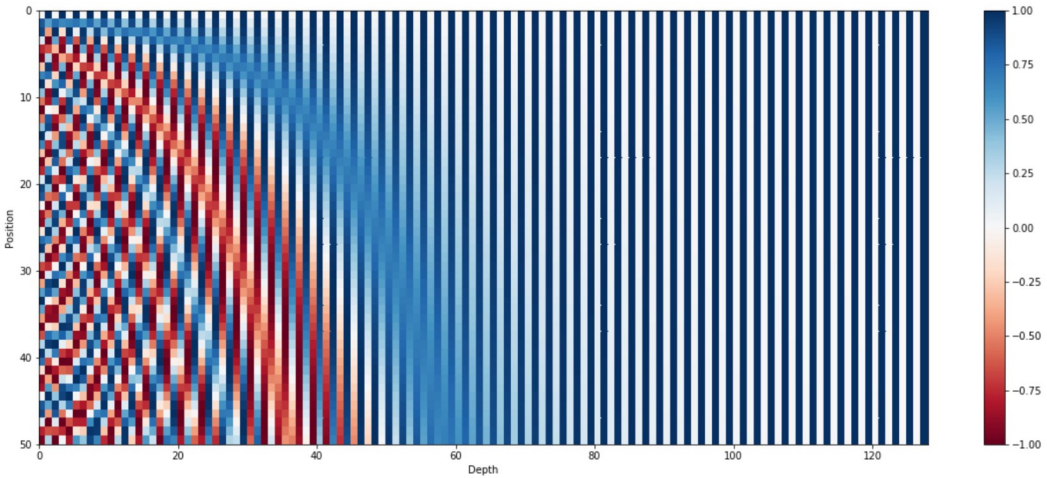
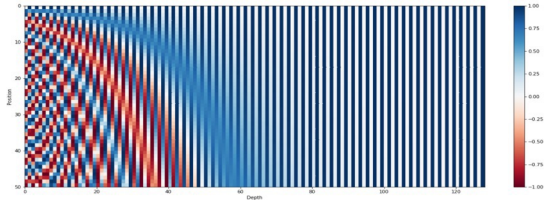


Figure 2 - The 128-dimensional positonal encoding for a sentence with the maximum lenght of 50. Each row represents the embedding vector $\overrightarrow{p_t}$

https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

Sinusoidal Positional Encoding



$$\overrightarrow{p_t}^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k . t), & \text{if } i = 2k \\ \cos(\omega_k . t), & \text{if } i = 2k+1 \end{cases} \quad \text{with} \quad \omega_k = \frac{1}{10000^{2k/d}}$$

t: position index

i: dimension index

d: total dimension

$$\overrightarrow{p_t} = \begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}_{d \times 1}$$

https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

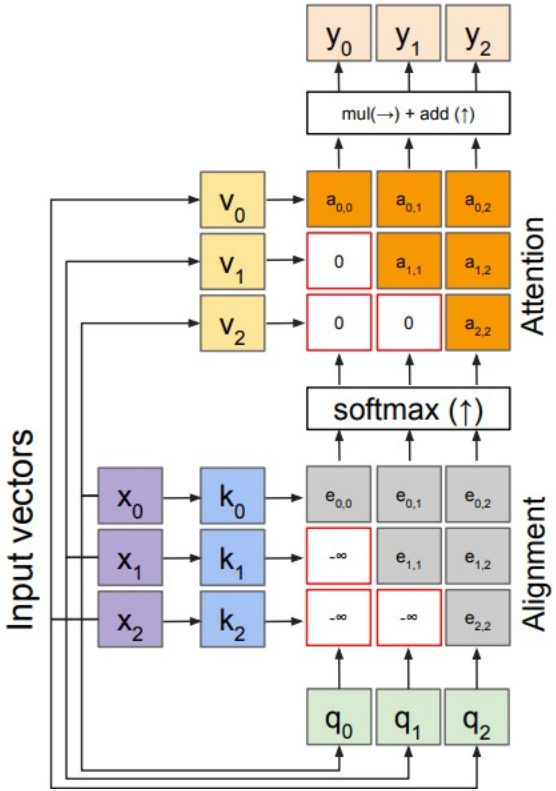
Sinusoidal Positional Encoding - Properties

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

$$\sin(\alpha + \beta) = \sin\alpha\cos\beta + \cos\alpha\sin\beta$$
$$\cos(\alpha + \beta) = \cos\alpha\cos\beta - \sin\alpha\sin\beta$$

https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

Masked self-attention layer



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

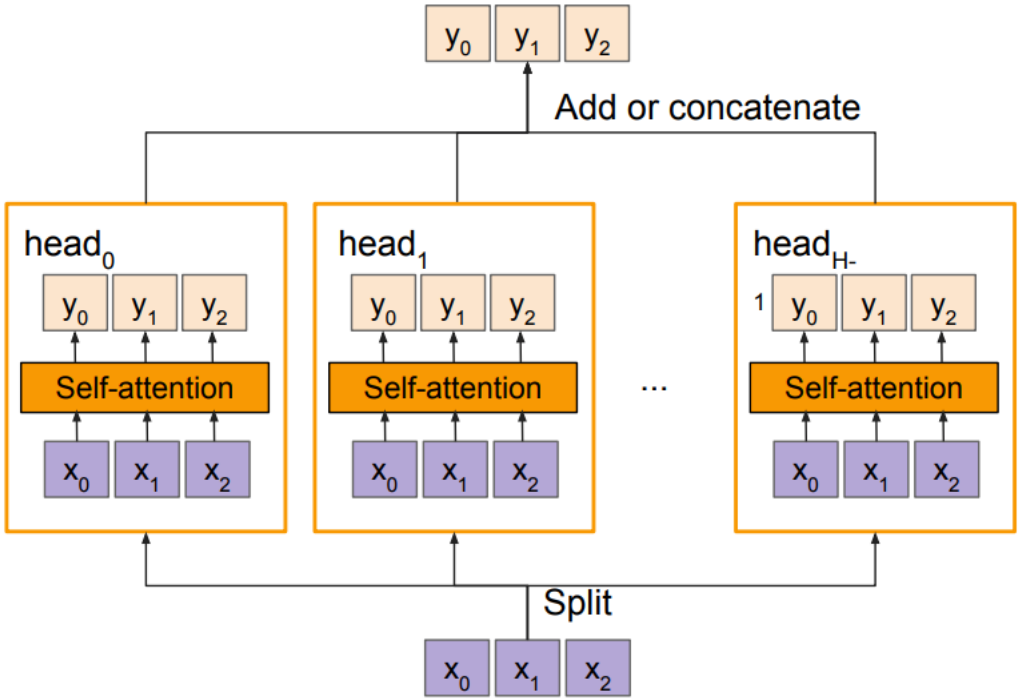
Operations:
Key vectors: $\mathbf{k} = \mathbf{x}\mathbf{W}_k$
Value vectors: $\mathbf{v} = \mathbf{x}\mathbf{W}_v$
Query vectors: $\mathbf{q} = \mathbf{x}\mathbf{W}_q$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $y_j = \sum_i a_{i,j} v_i$

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



Comparing RNNs to Transformers

RNNs
(+) LSTMs work reasonably well for long sequences.
(-) Expects an ordered sequences of inputs
(-) Sequential computation: subsequent hidden states can only be computed after the previous ones are done.

Transformers:
(+) Good at long sequences. Each attention calculation looks at all inputs.
(+) Can operate over unordered sets or ordered sequences with positional encodings.
(+) Parallel computation: All alignment and attention scores for all inputs can be done in parallel.
(-) Requires a lot of memory: N x M alignment and attention scalers need to be calculated and stored for a single self-attention head. (but GPUs are getting bigger and better)

Today's Agenda:

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

NLP using Transformer

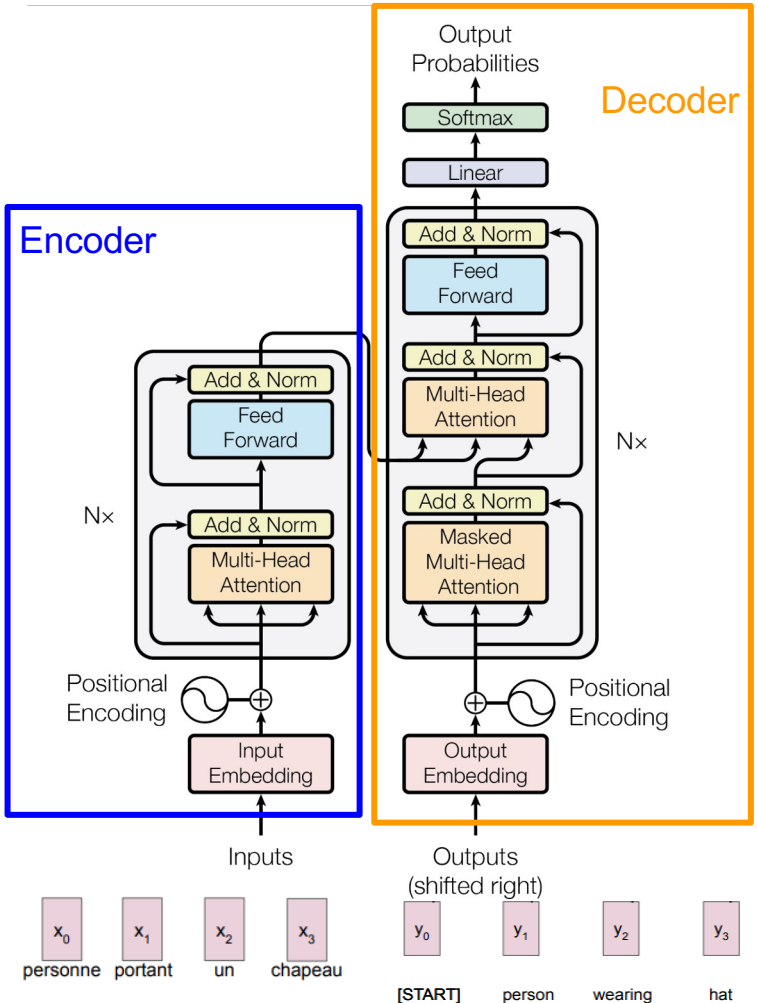
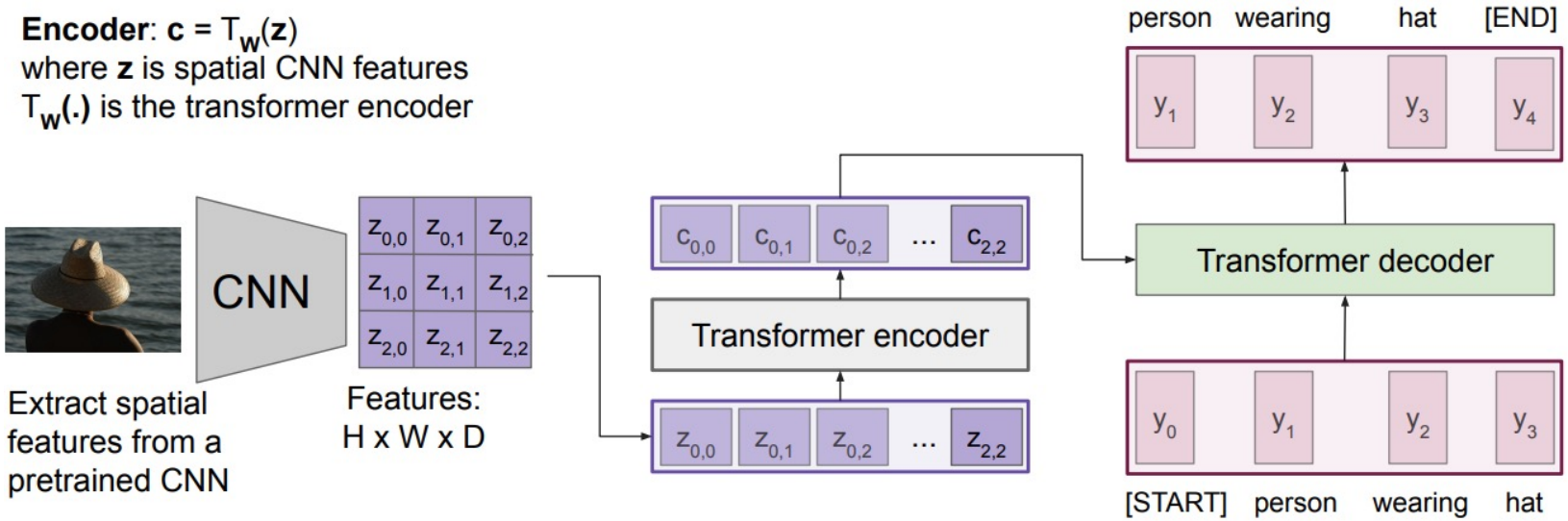


Image Captioning using transformers

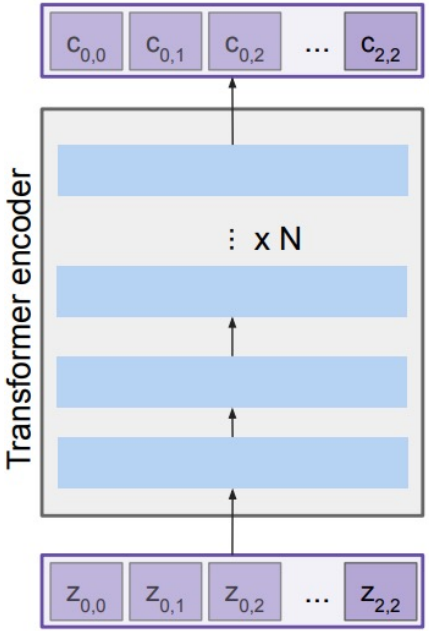
Input: Image **I**
Output: Sequence $\mathbf{y} = y_1, y_2, \ldots, y_T$

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 **z** is spatial CNN features
$T_W(.)$ is the transformer encoder



The Transformer encoder block

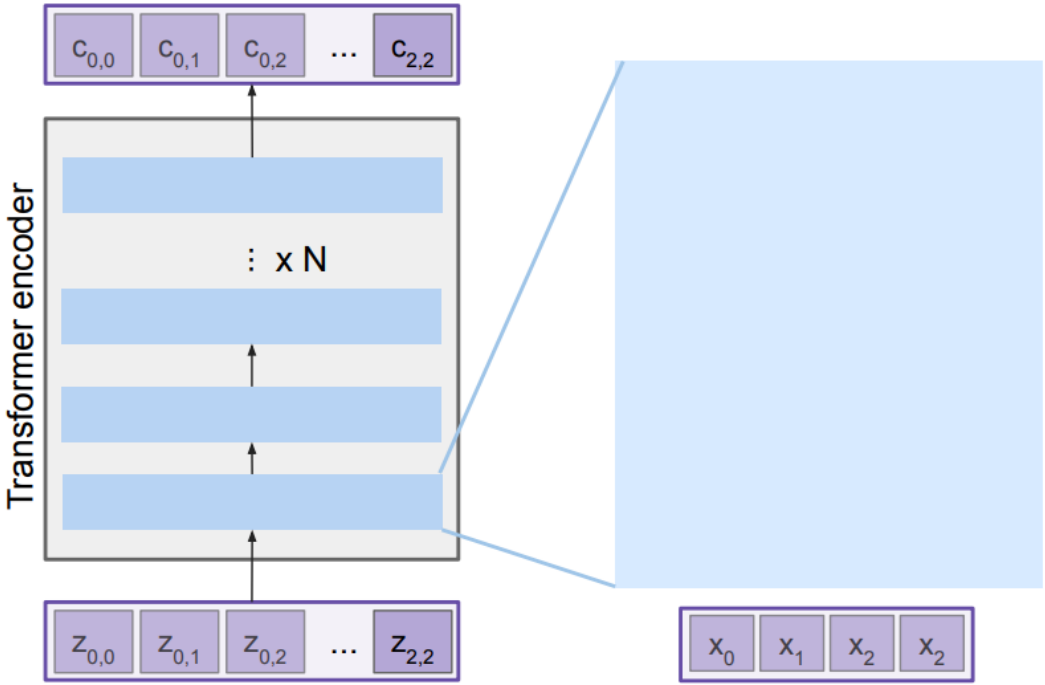


Made up of N encoder blocks.

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

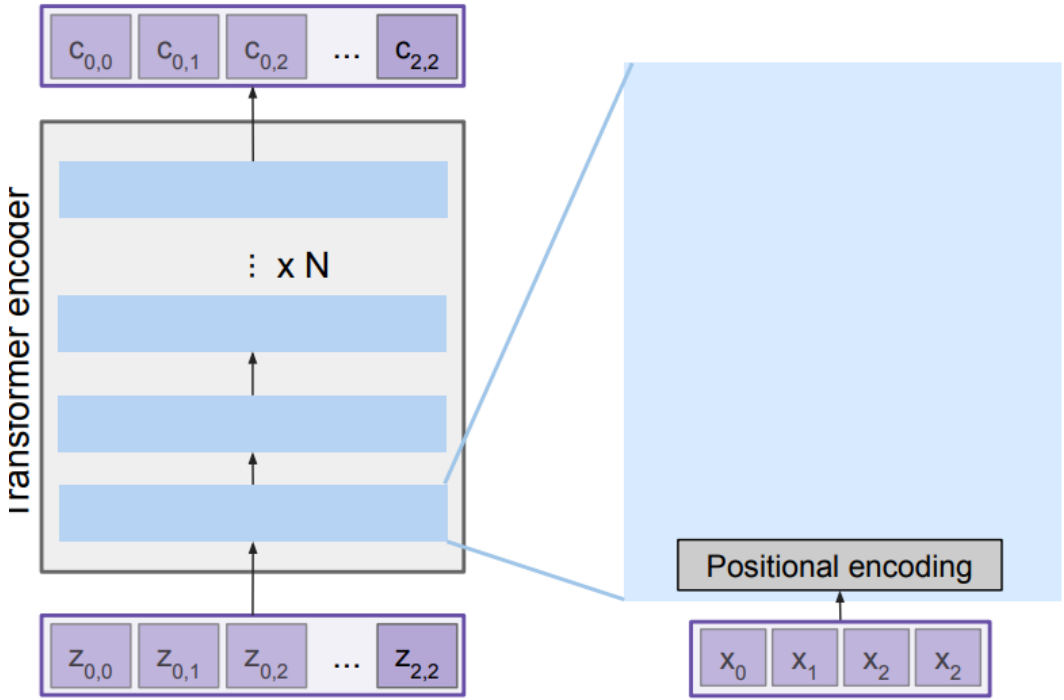
Vaswani et al, “Attention is all you need”, NeurIPS 2017

The Transformer encoder block



Let's dive into one encoder block

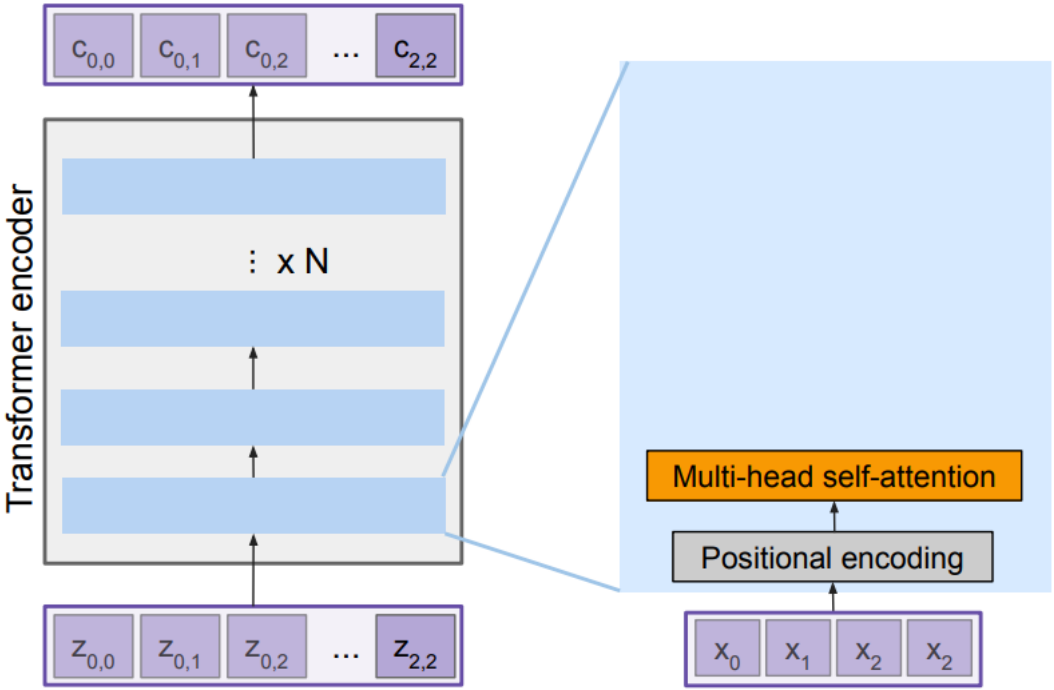
The Transformer encoder block



Add positional encoding

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

The Transformer encoder block

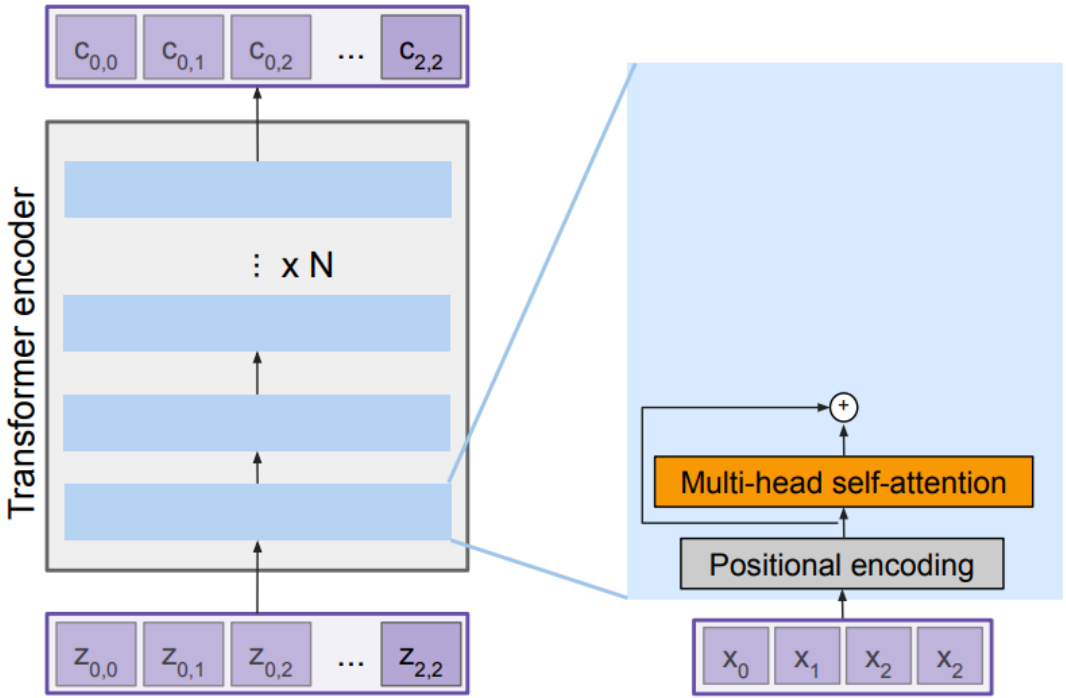


Attention attends over all the vectors

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

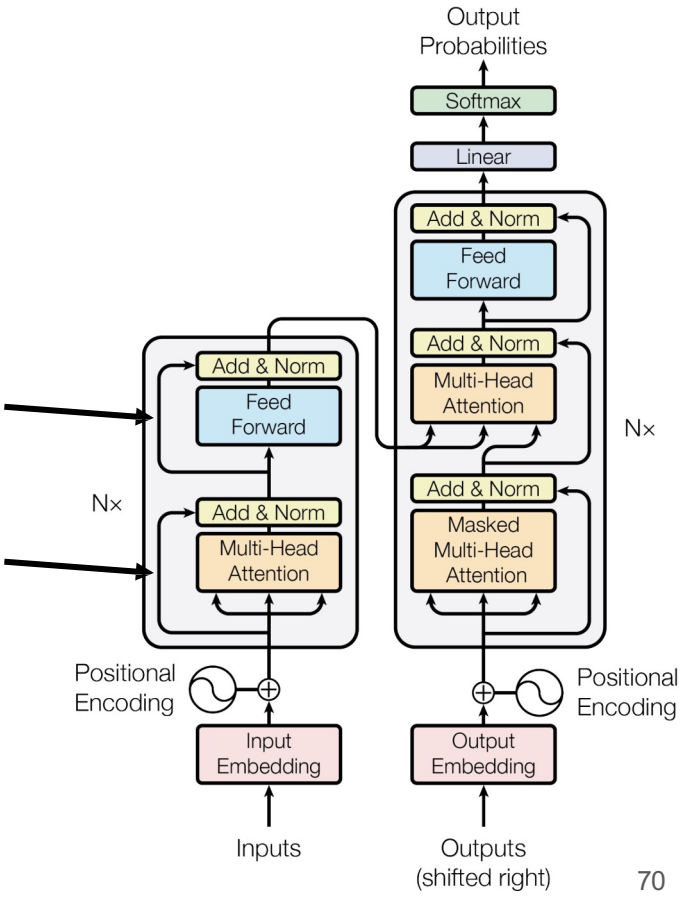
Residual Connection

Residual Connection (He et al., 2016)

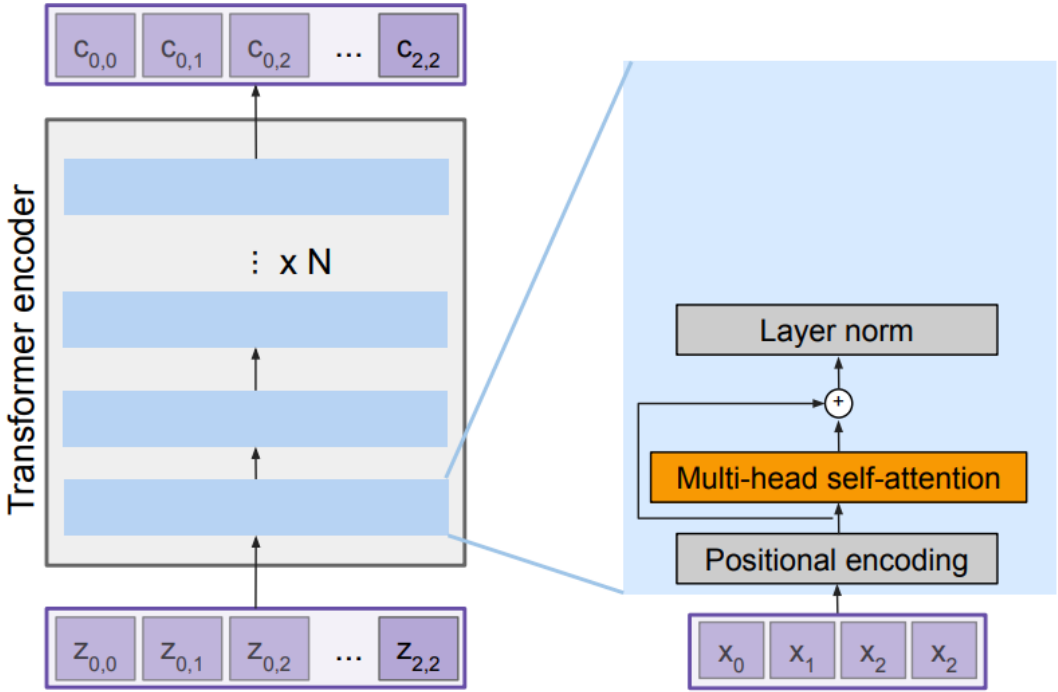
- Add input to its output

1. Residual connections mainly help mitigate the vanishing gradient problem. Without the residual connections, a large part of the training signal would get lost during back-propagation. Residual connections reduce effect because summation is linear with respect to derivative, so each residual block also gets a signal that is not affected by the vanishing gradient. The summation operations of residual connections form a path in the computation graphs where the gradient does not get lost.

2. Another effect of residual connections is that the information stays local in the Transformer layer stack. The self-attention mechanism allows an arbitrary information flow in the network and thus arbitrary permuting the input tokens. The residual connections, however, always "remind" the representation of what the original state was. To some extent, the residual connections give a guarantee that contextual representations of the input tokens really represent the tokens.



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

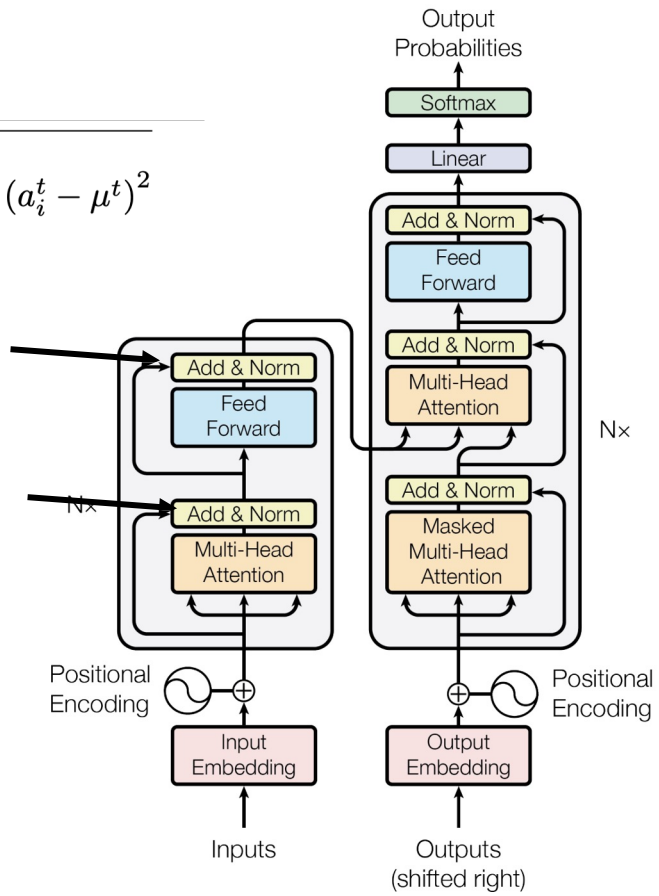
Layer Normalization

Layer Normalization (Ba et al., 2016)

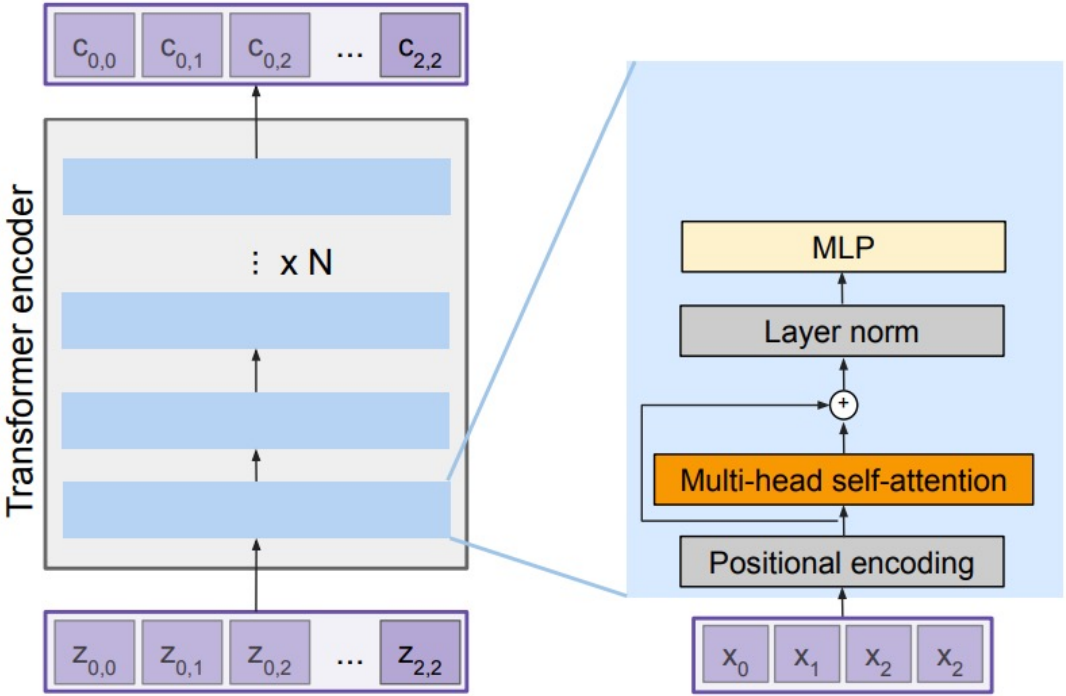
$$\mathbf{h}^t = f\left[\frac{\mathbf{g}}{\sigma^t} \odot \left(\mathbf{a}^t - \mu^t\right) + \mathbf{b}\right] \qquad \mu^t = \frac{1}{H}\sum_{i=1}^{H} a_i^t \qquad \sigma^t = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^t - \mu^t\right)^2}$$

```python
class LayerNorm(nn.Module):
    "Construct a layernorm module (See citation for details)."
    def __init__(self, features, eps=1e-6):
        super(LayerNorm, self).__init__()
        self.a_2 = nn.Parameter(torch.ones(features))
        self.b_2 = nn.Parameter(torch.zeros(features))
        self.eps = eps

    def forward(self, x):
        mean = x.mean(-1, keepdim=True)
        std = x.std(-1, keepdim=True)
        return self.a_2 * (x - mean) / (std + self.eps) + self.b_2
```



The Transformer encoder block



MLP over each vector individually

LayerNorm over each vector individually

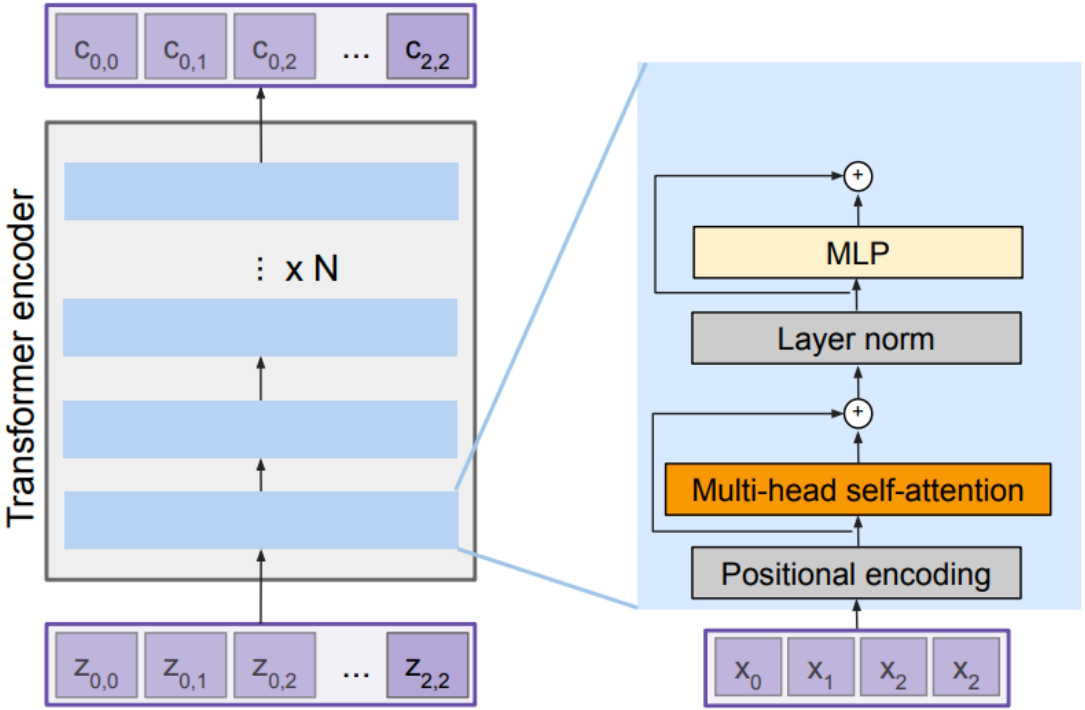
Residual connection

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

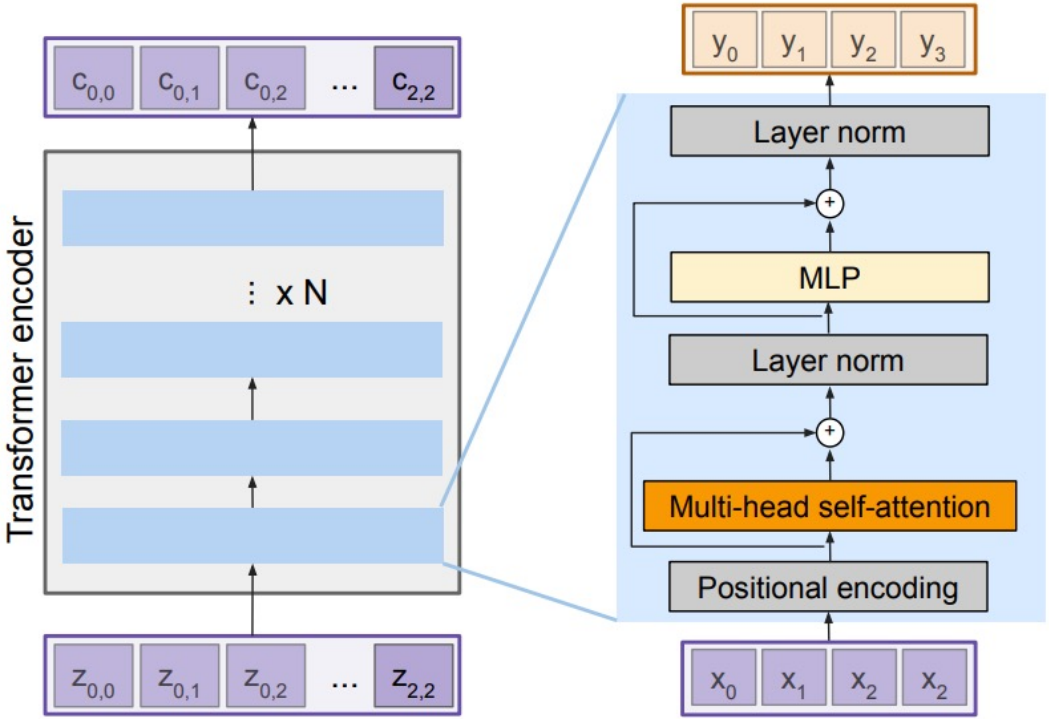
Residual connection

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**

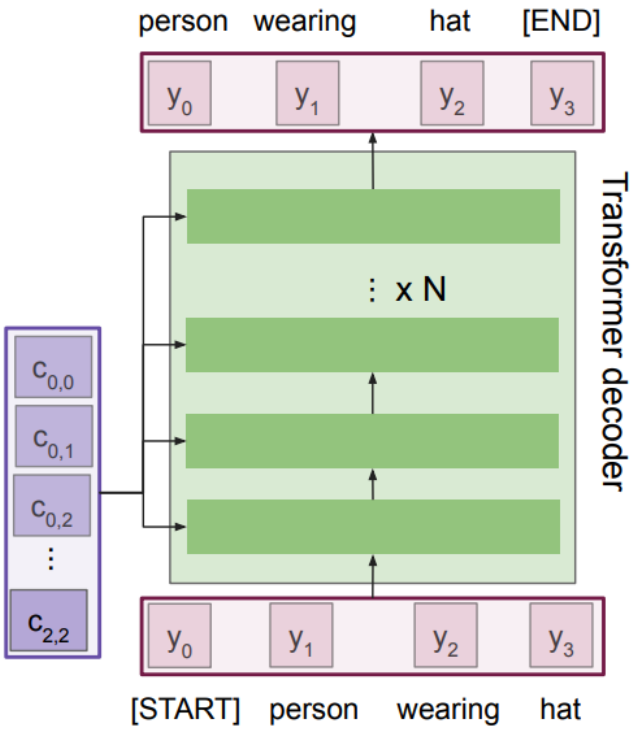
Self-attention is the only interaction between vectors.

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

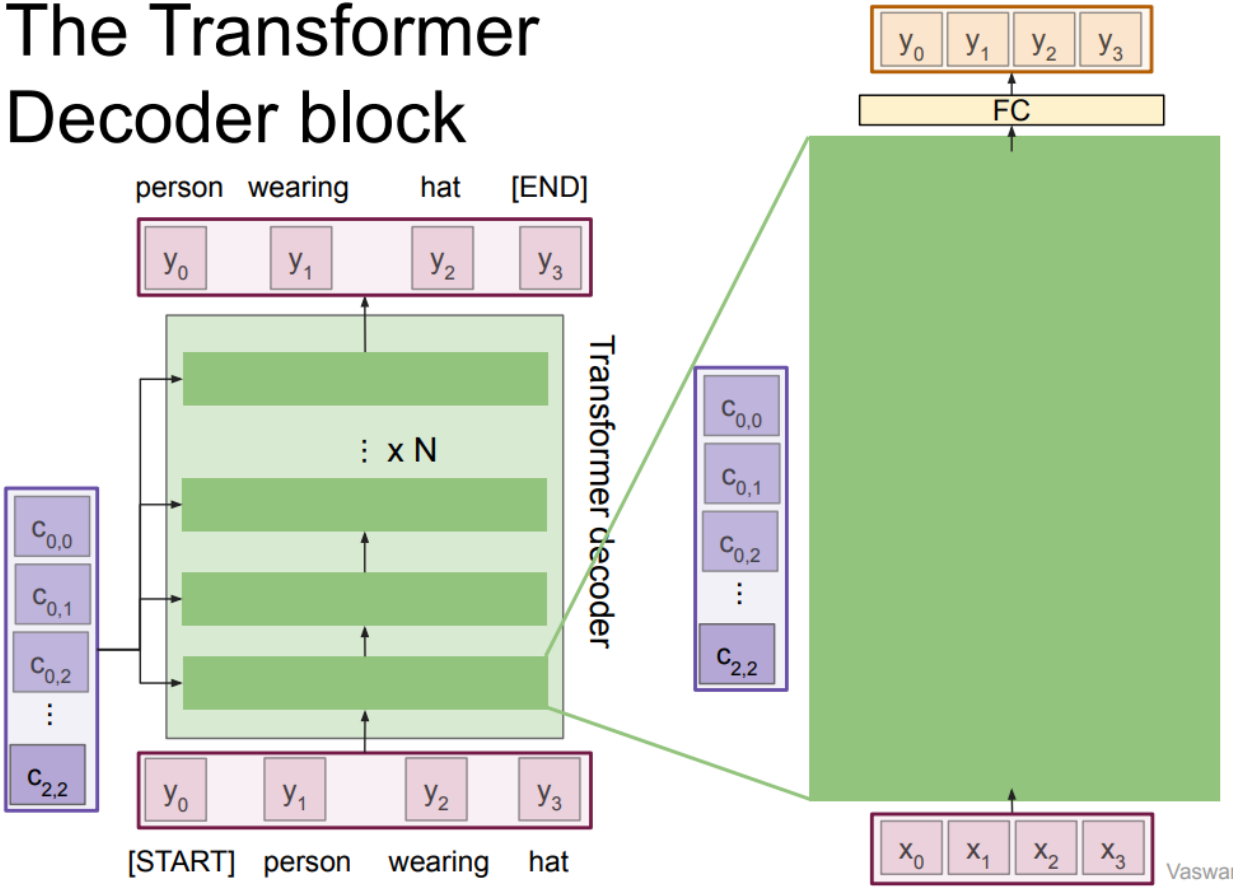


Made up of N decoder blocks.

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

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

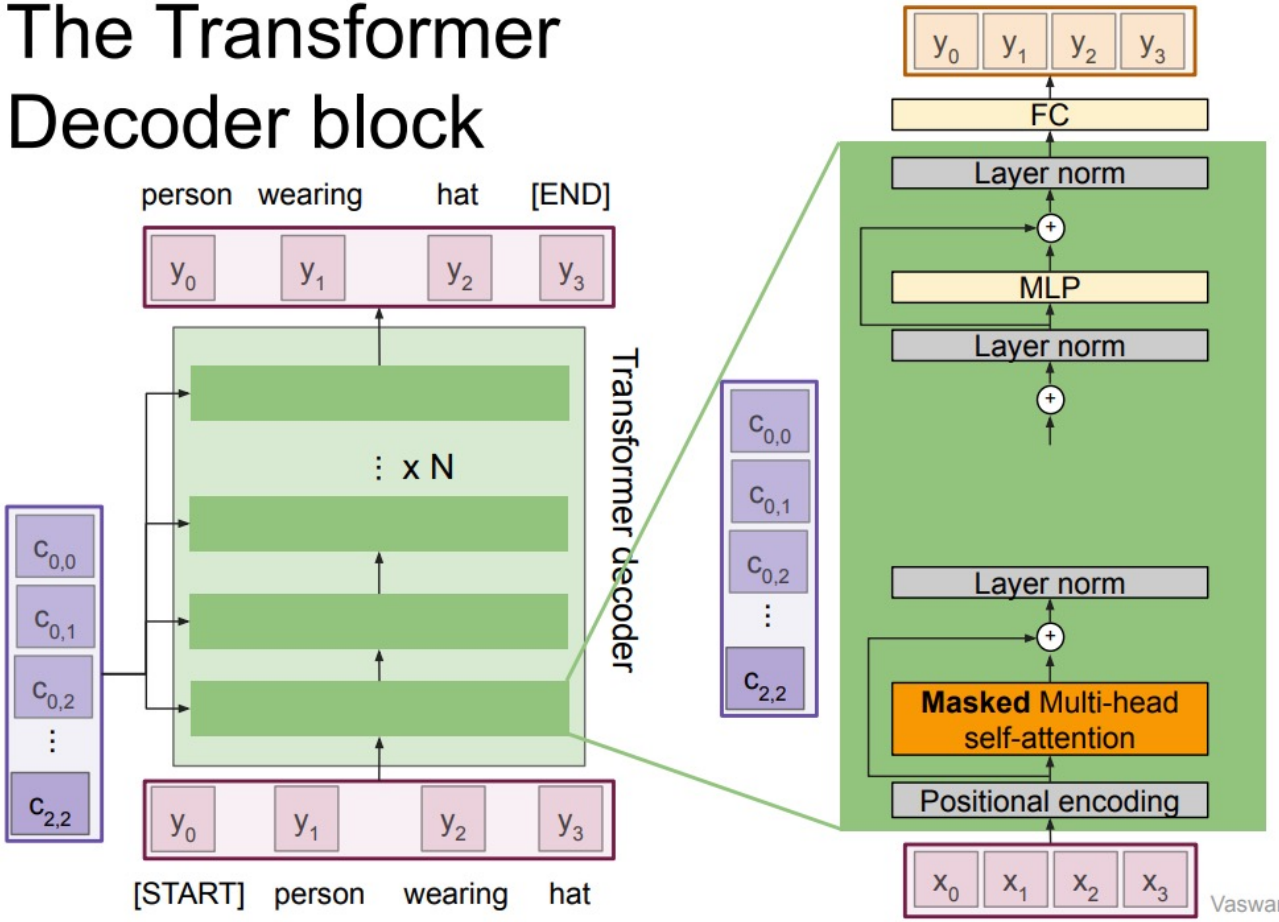
The Transformer Decoder block



Let's dive into the transformer decoder block

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

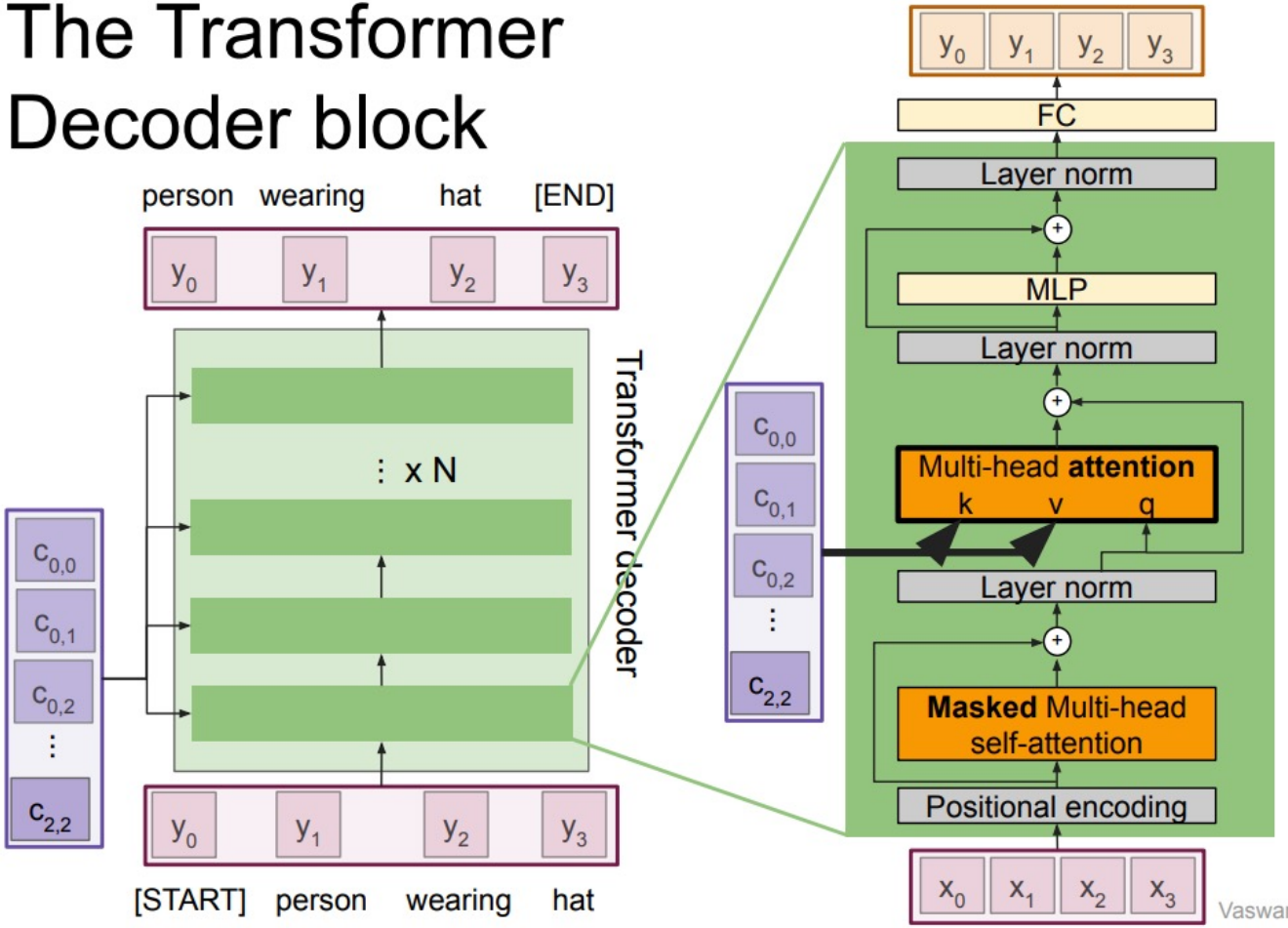
The Transformer Decoder block



Most of the network is the same the transformer encoder.

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

The Transformer Decoder block



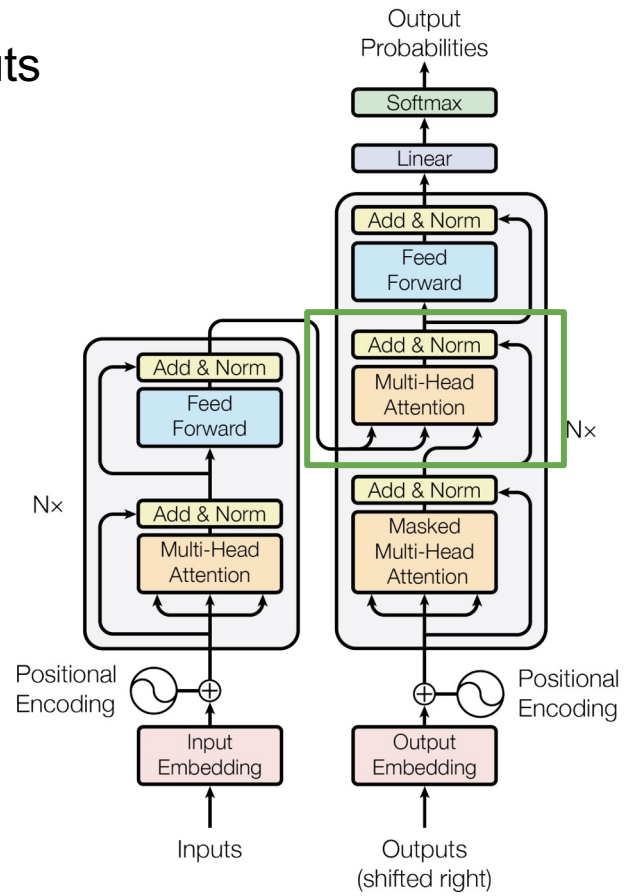
Multi-head attention block attends over the transformer encoder outputs.

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

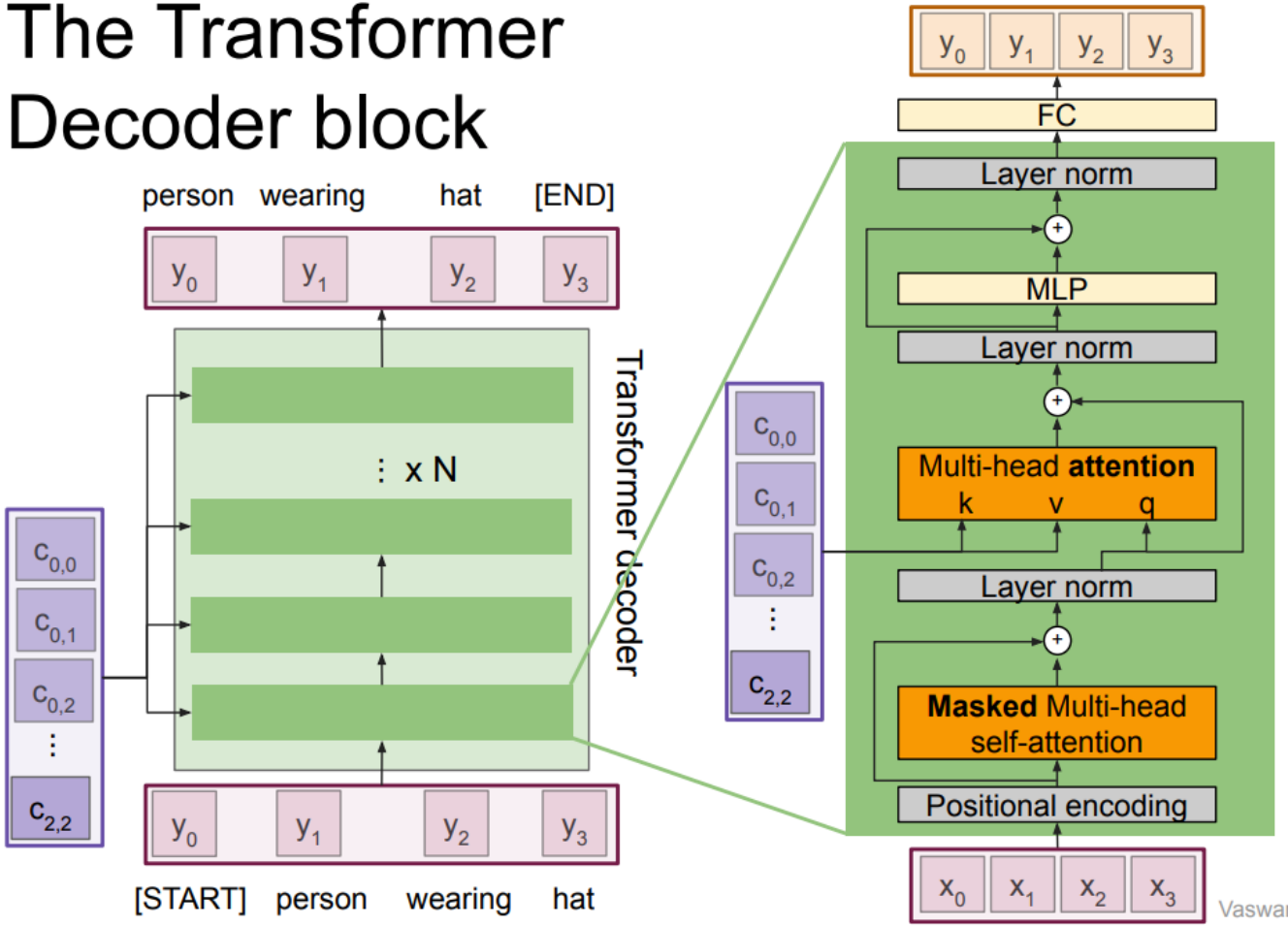
Vaswani et al, "Attention is all you need", NeurIPS 2017

Encoder-Decoder Cross Attention

- Attention between encoder self-attention outputs and decoder masked self attention outputs.



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.

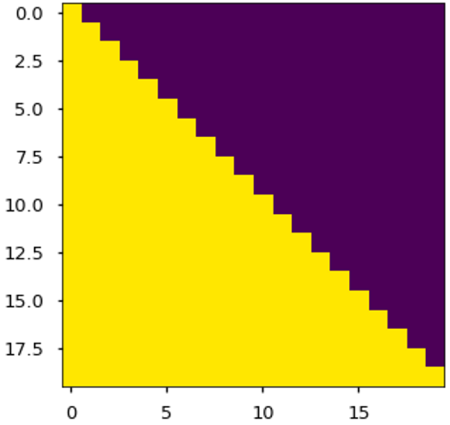
Multi-head attention block is NOT self-attention. It attends over encoder outputs.

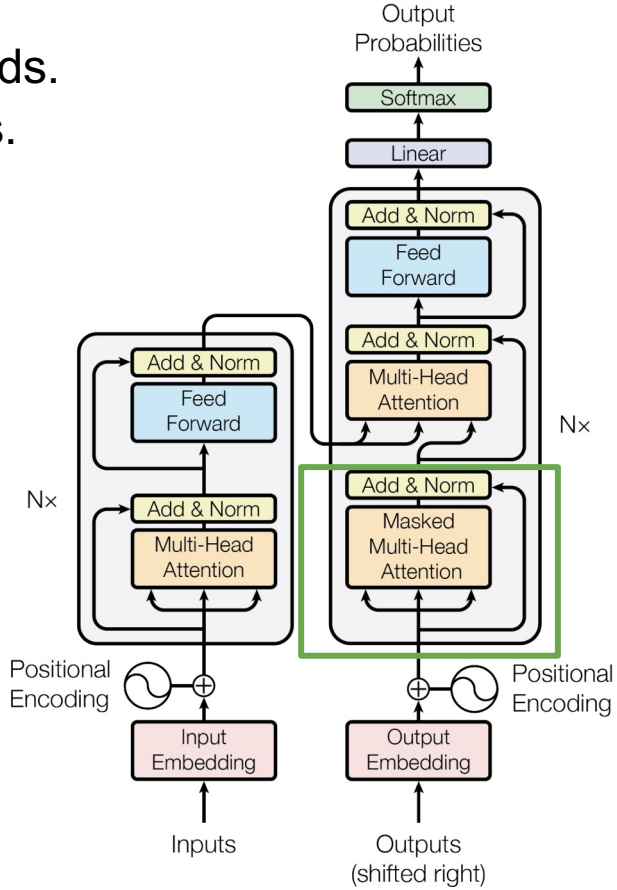
Highly scalable, highly parallelizable, but high memory usage.

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

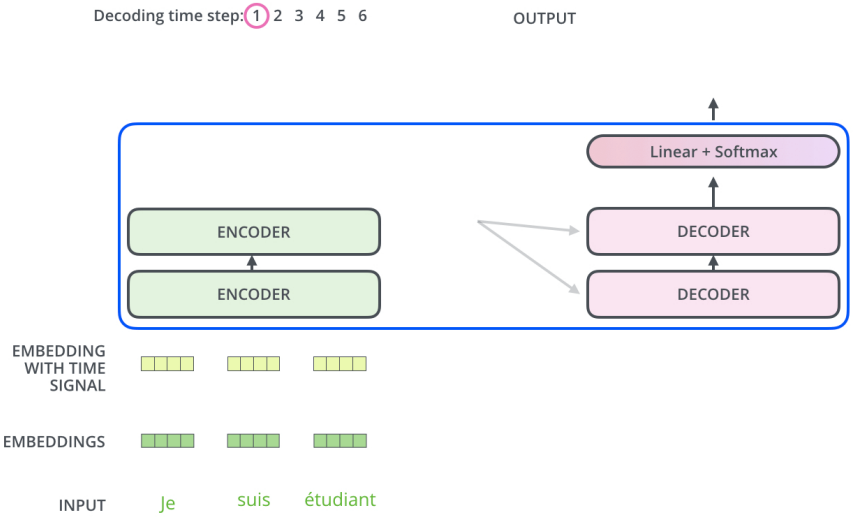
Masked Self Attention in Decoder

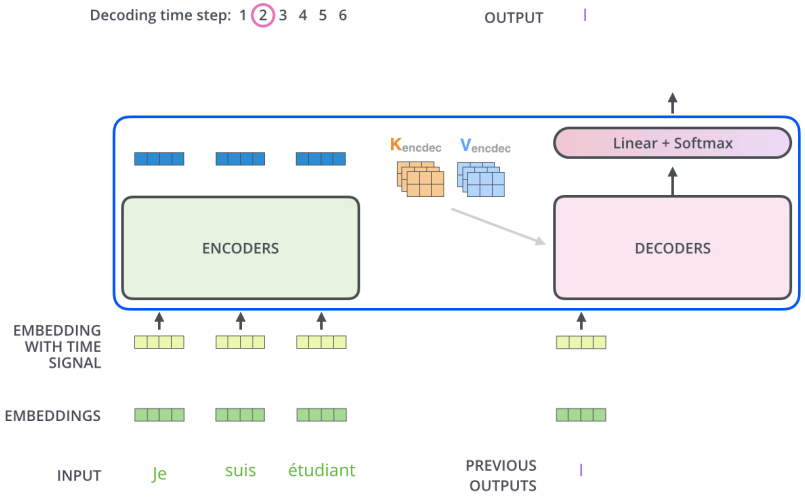
- In encoder, we can always look at all input words.
- In decoder, we can only look at previous words.
  - That means, we need to "**mask**" future words when performing attention.
  - Words are blocked for attending to future words during training.





Transformer – decoder side





https://jalammar.github.io/illustrated-transformer/

Image Captioning using transformers

- **No recurrence at all**

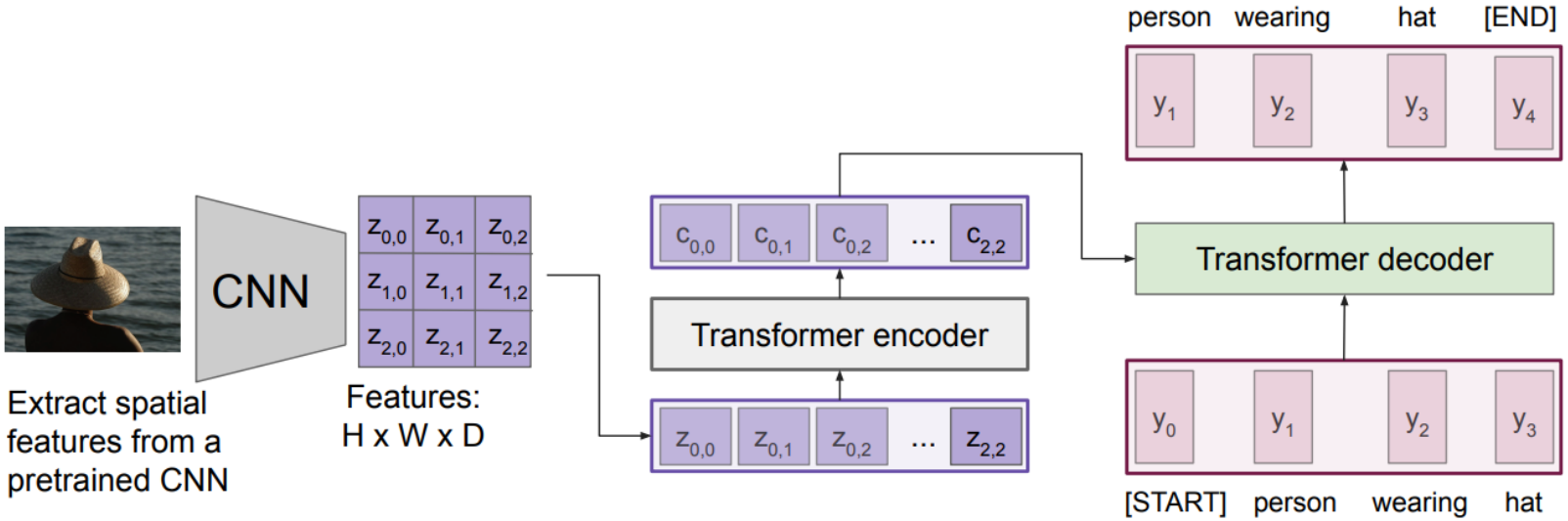


Image Captioning using transformers

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

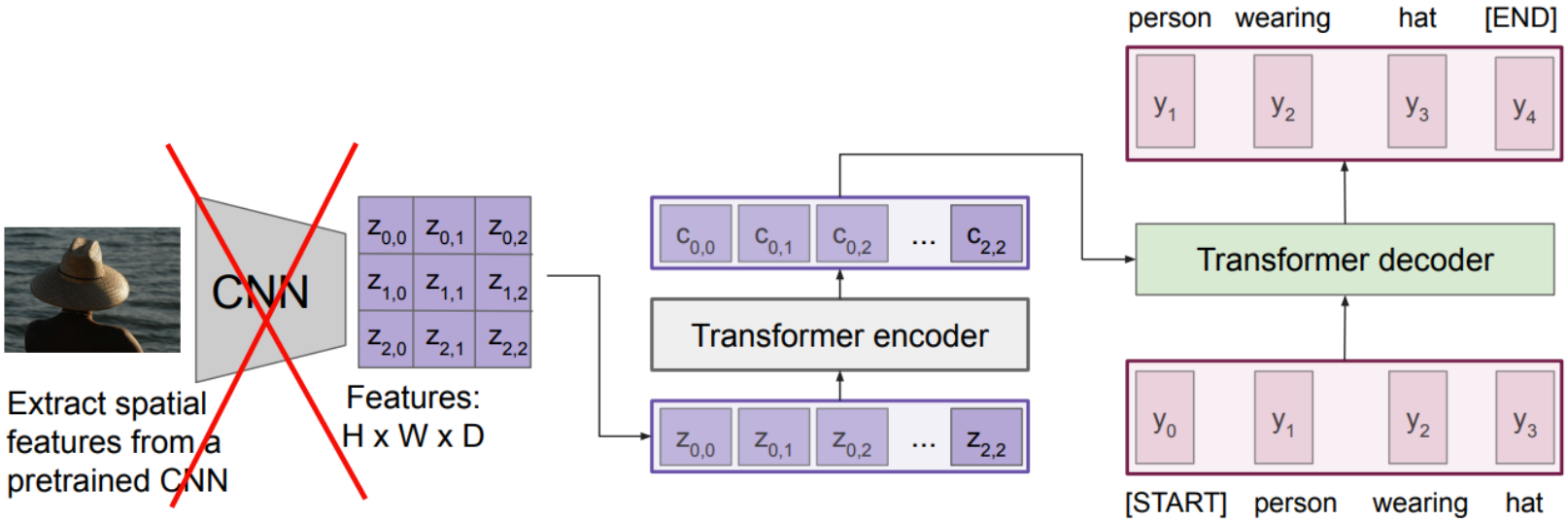
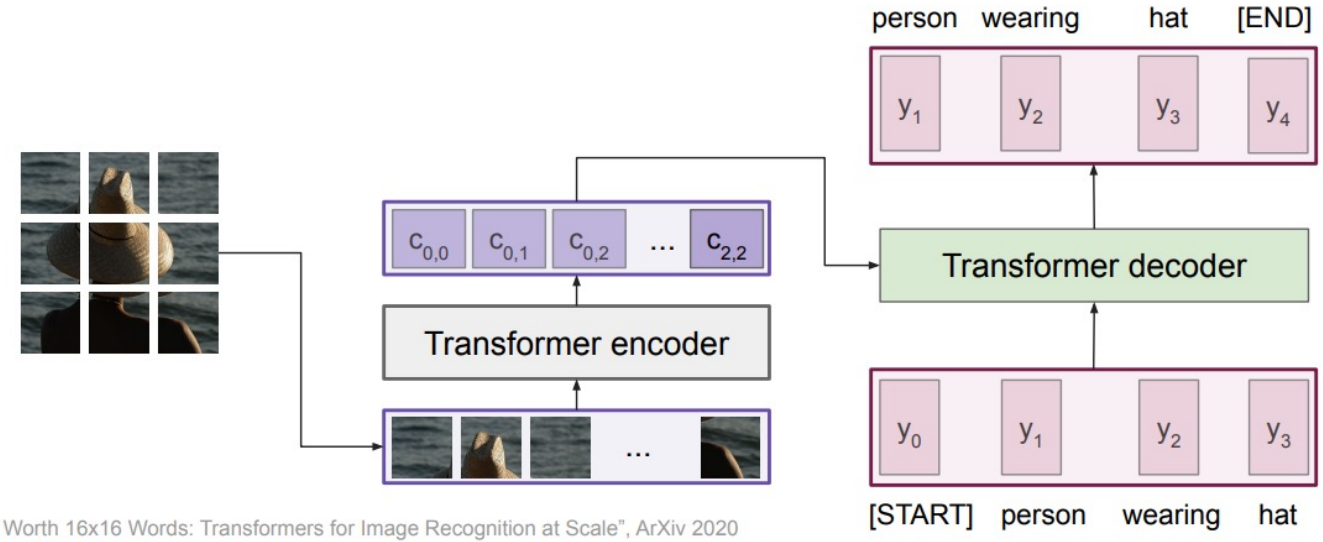


Image Captioning using ONLY transformers

- **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

Image Captioning using ONLY transformers

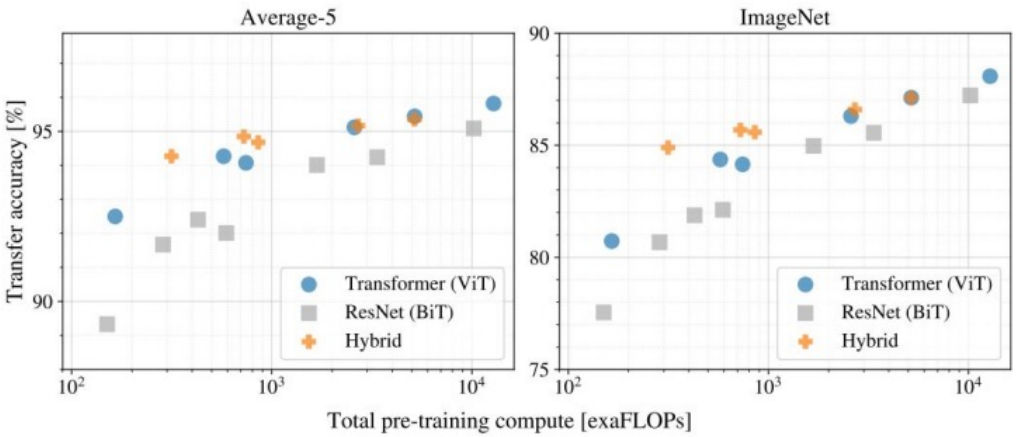


Figure 5: Performance versus cost for different architectures: Vision Transformers, ResNets, and hybrids. Vision Transformers generally outperform ResNets with the same computational budget. Hybrids improve upon pure Transformers for smaller model sizes, but the gap vanishes for larger models.

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

Other Tricks in Transformers

- Subword Tokenization
- Optimizer
- Label Smoothing

OOV Issue

How to handle unknown words, i.e., a word at test time that we've never seen in our training data. It is also called OOV (out-of-vocabulary) words.

If we use word-level tokenization, we cannot have an index for unknown words, and we cannot have a word embedding for unknown words.

OOV Issue

Old solution: replace low-frequency words in training data with a special token <UNK>; then after training, we have a word embedding for <UNK>; if we have an unknown word during testing, we replace it with <UNK>.

New solution: **Subword tokenization**.

Subword Tokenization

Use Subword Units for tokenization.

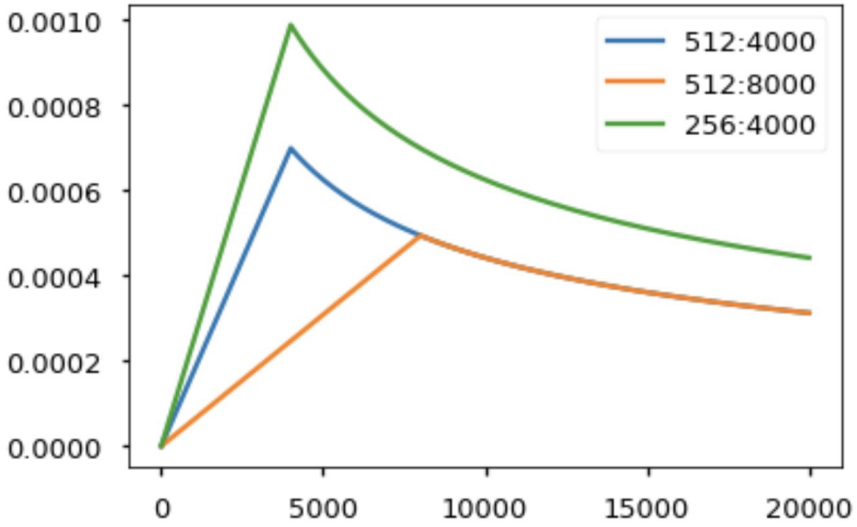
Based on a simple algorithm called byte pair encoding (Gage, 1994).

First used successfully for NLP for machine translation by (Sennrich et al., 2016).

Optimizer

Adam Optimizer with Linear Warmup Learning Rate

We used the Adam optimizer (cite) with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\epsilon = 10^{-9}$. We varied the learning rate over the course of training, according to the formula: $lrate = d_{\text{model}}^{-0.5} \cdot \min(step_num^{-0.5}, step_num \cdot warmup_steps^{-1.5})$ This corresponds to increasing the learning rate linearly for the first $warmup_steps$ training steps, and decreasing it thereafter proportionally to the inverse square root of the step number. We used $warmup_steps = 4000$.



- Adam

$$m_0 = 0, v_0 = 0$$

$$m_{t+1} \leftarrow \beta_1 m_t + (1 - \beta_1)\nabla_\theta \mathcal{L}(\theta)$$

Momentum

$$v_{t+1} \leftarrow \beta_2 v_t + (1 - \beta_2)\nabla_\theta \mathcal{L}(\theta)^2$$

RMS Prop

$$\theta_j \leftarrow \theta_j - \frac{\epsilon}{\sqrt{v_{t+1}} + 1e^{-5}} m_{t+1}$$

RMS Prop + Momentum

Algo of Adam.

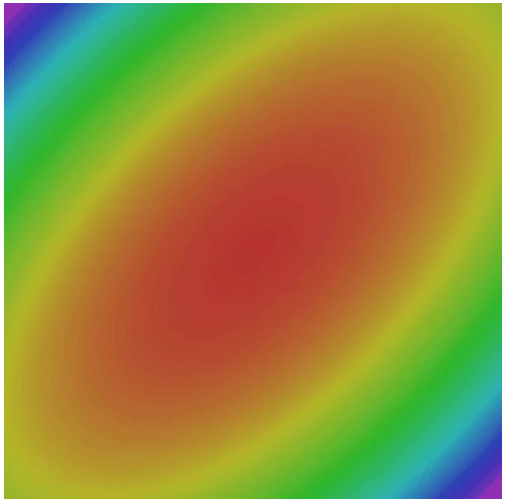


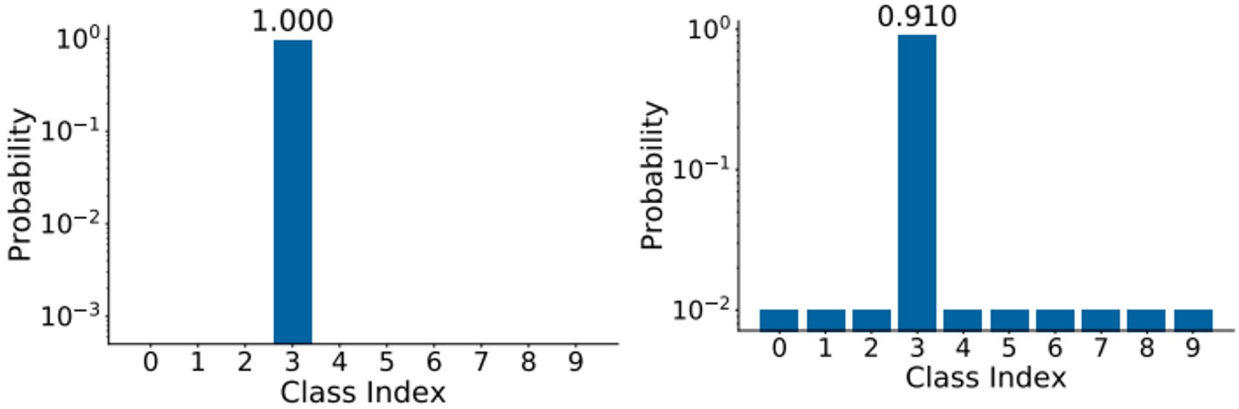
Fig. 6 Comparison of all optimization algorithms. (Visualization)

https://morioh.com/p/3f20600908f3

Label Smoothing

During training, we employed label smoothing of value $\epsilon_{ls} = 0.1$ (cite). This hurts perplexity, as the model learns to be more unsure, but improves accuracy and BLEU score.

> *We implement label smoothing using the KL div loss. Instead of using a one-hot target distribution, we create a distribution that has* `confidence` *of the correct word and the rest of the* `smoothing` *mass distributed throughout the vocabulary.*



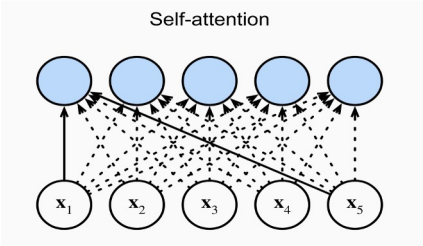
(a) Hard Label (b) LS

https://towardsdatascience.com/label-smoothing-make-your-model-less-over-confident-b12ea6f81a9a

Computational Complexity: Quadratic in Length!

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$\boxed{O(n^2 \cdot d)}$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$



Results

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	

Model Variations

Table 3: Variations on the Transformer architecture. Unlisted values are identical to those of the base model. All metrics are on the English-to-German translation development set, newstest2013. Listed perplexities are per-wordpiece, according to our byte-pair encoding, and should not be compared to per-word perplexities.

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)	2									6.11	23.7	36
	4									5.19	25.3	50
	8									4.88	25.5	80
		256			32	32				5.75	24.5	28
		1024			128	128				4.66	26.0	168
			1024							5.12	25.4	53
			4096							4.75	26.2	90
(D)							0.0			5.77	24.6	
							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)	positional embedding instead of sinusoids									4.92	25.7	
big	6	1024	4096	16			0.3		300K	**4.33**	**26.4**	213

Transformer-based Language Models

It is becoming the underlying architecture of most popular language models. Two most famous are:

- GPT: OpenAI Transformer-based Language Models
- BERT: Google Transformer-based Language Models
- RoBERTa: Facebook Transformer-based Language Models

Reading

Annotated Transformers by Alexander M. Rush

The Illustrated Transformer by Jay Alammar

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.