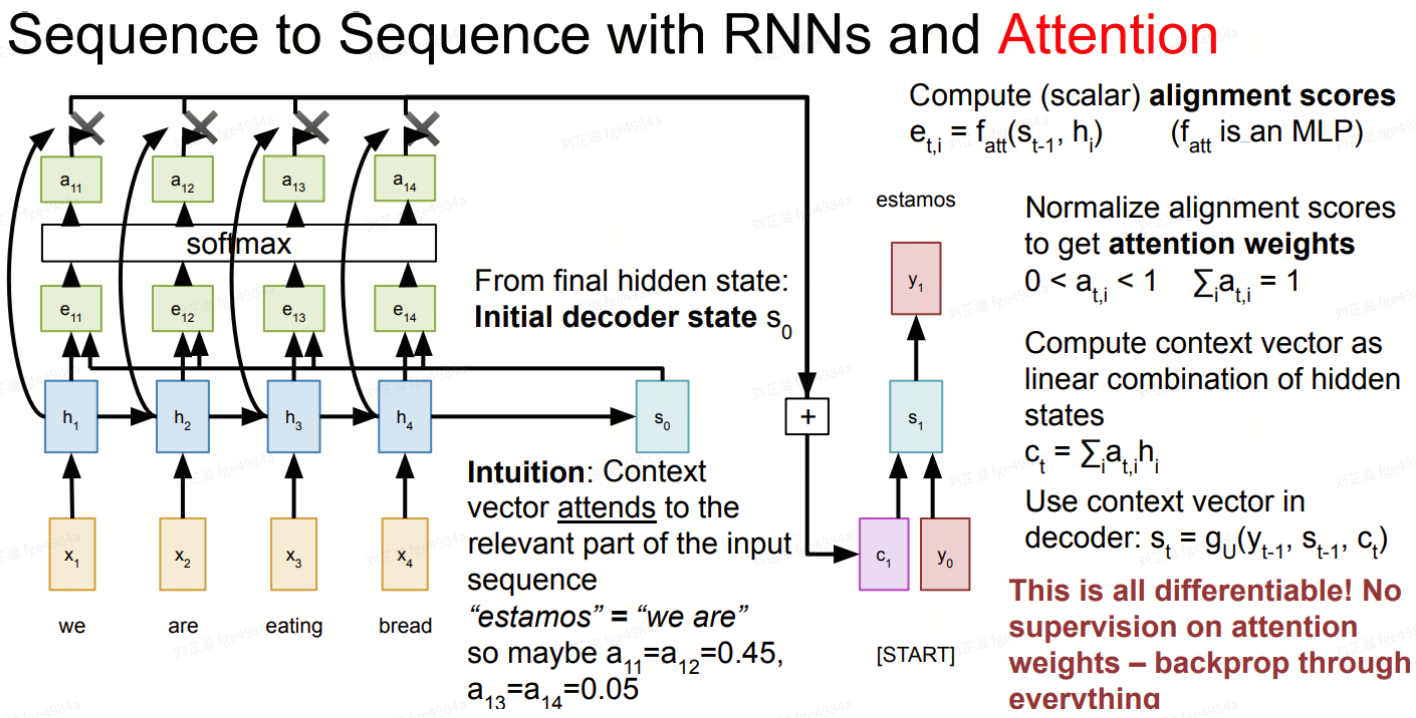
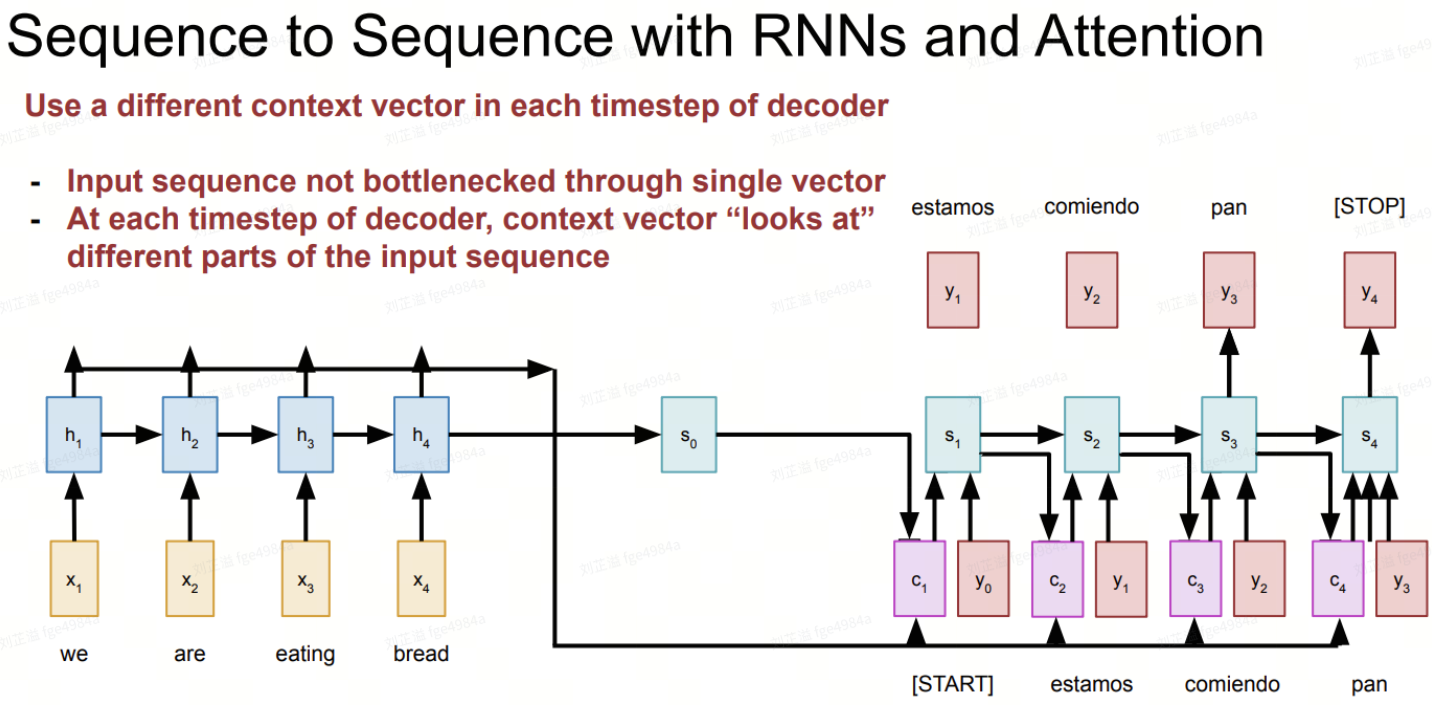
Attention and Transformers

Attention

- encoder中设置上下文向量
- decoder中每个时间步的上下文向量都不同

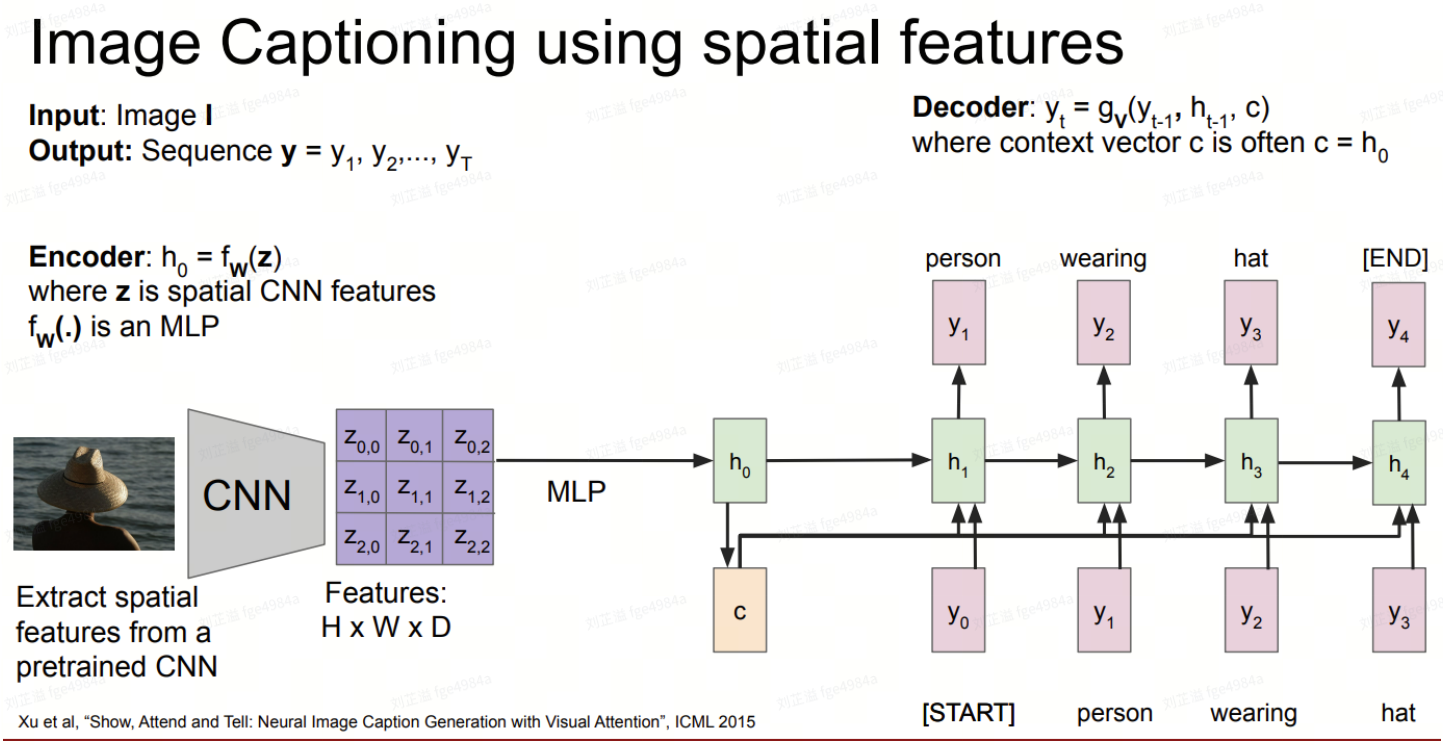




CNN + Attention

CNN + RNN

- Input is "bottlenecked" through c; **because model needs to encode everything with c**



CNN + Attention

- encoder每个步的上下文变量都不同，会重新根据注意力机制计算
- 模型自己选择注意力权重，不需要注意力监督

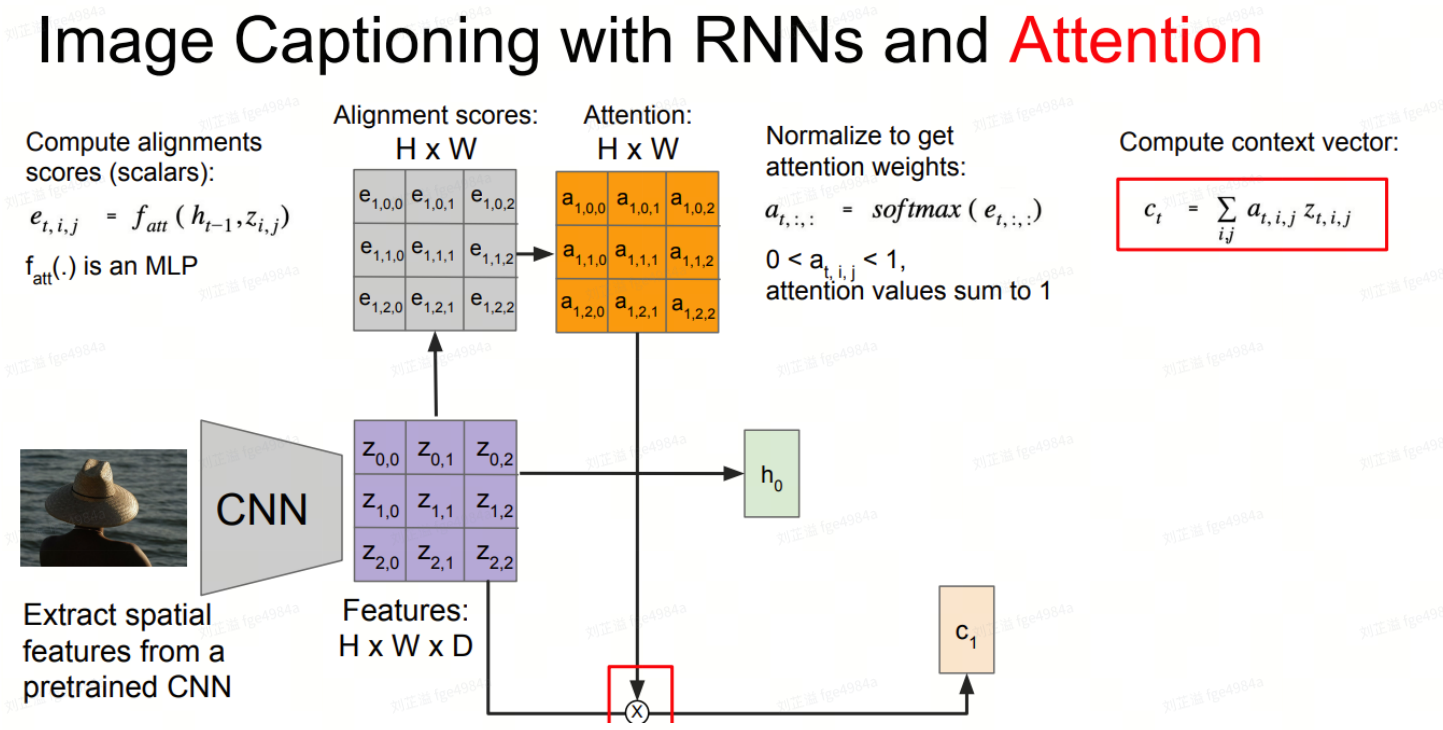
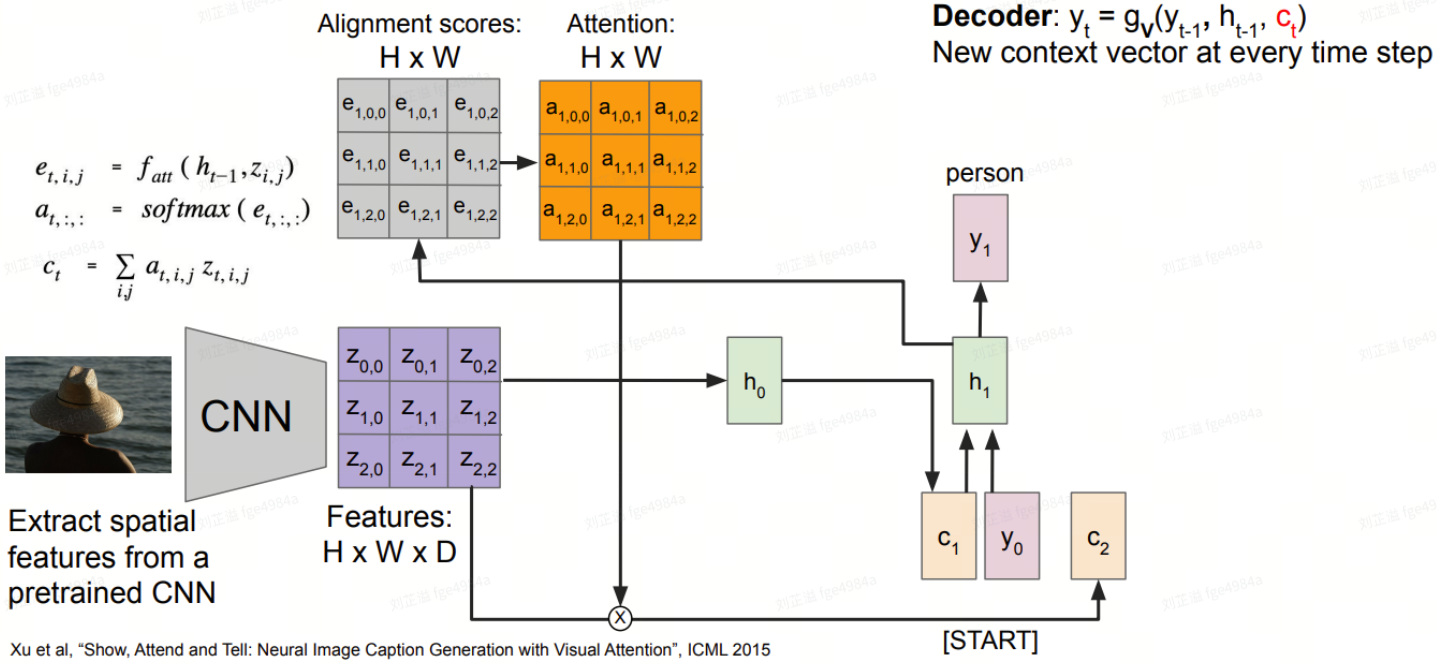


Image Captioning with RNNs and Attention

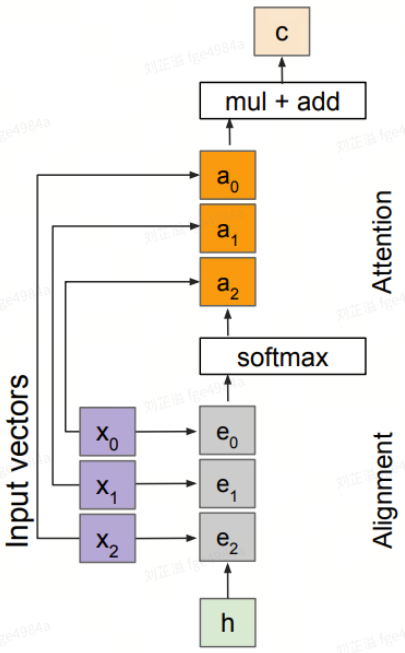


Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Attention Layer

- 通过除以D**0.5，消除大幅度向量对整体向量的影响

General attention layer



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

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

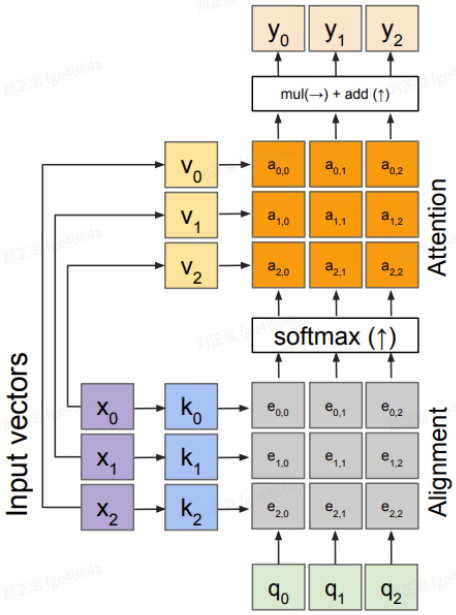
Inputs:
Input vectors: **x** (shape: N x D)
Query: **h** (shape: D)

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 √D to reduce effect of large magnitude vectors

- 增加不同的全连接层增加该层的表现力

General attention layer



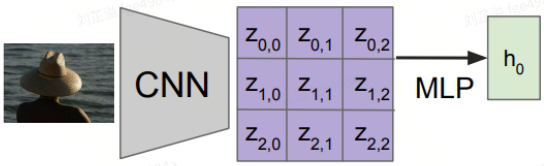
Outputs:
context vectors: **y** (shape: D_v)

Operations:
Key vectors: $k = xW_k$
Value vectors: $v = xW_v$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $a = softmax(e)$
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

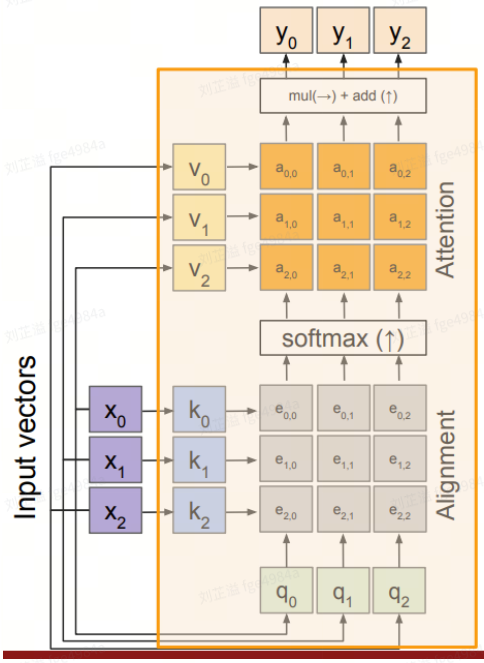
Encoder: $h_0 = f_W(z)$
where **z** is spatial CNN features
$f_W(.)$ is an MLP



Self Attention Layer

- 不关心输入的排列顺序，需要增加**positional encoding**

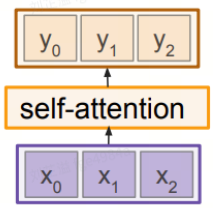
Self attention layer - attends over sets of inputs



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

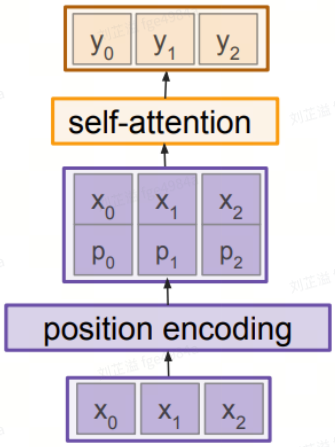
Operations:
Key vectors: $k = xW_k$
Value vectors: $v = xW_v$
Query vectors: $q = xW_q$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: $a = softmax(e)$
Output: $y_j = \sum_i a_{i,j} v_i$

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



Positional Encoding

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, $p_i = pos(j)$

Options for *pos*(.)

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

Desiderata of *pos*(.) :

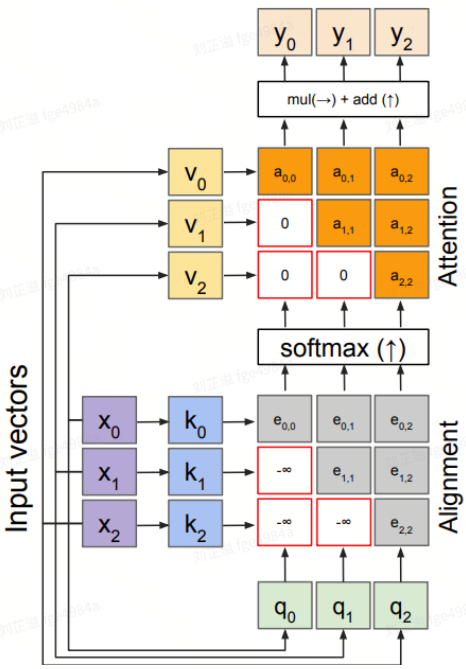
1. It should output a **unique** encoding for each time-step (word's position in a sentence)
2. **Distance** between any two time-steps should be consistent across sentences with different lengths.
3. Our model should generalize to **longer** sentences without any efforts. Its values should be bounded.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

Vaswani et al. "Attention is all you nee

Masked self-attention layer

- 防止向量查看未来向量

Masked self-attention layer



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

Operations:
Key vectors: $k = xW_k$
Value vectors: $v = xW_v$
Query vectors: $q = xW_q$
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$
Attention: **a** = softmax(**e**)
Output: $y_j = \sum_i a_{i,j} v_i$

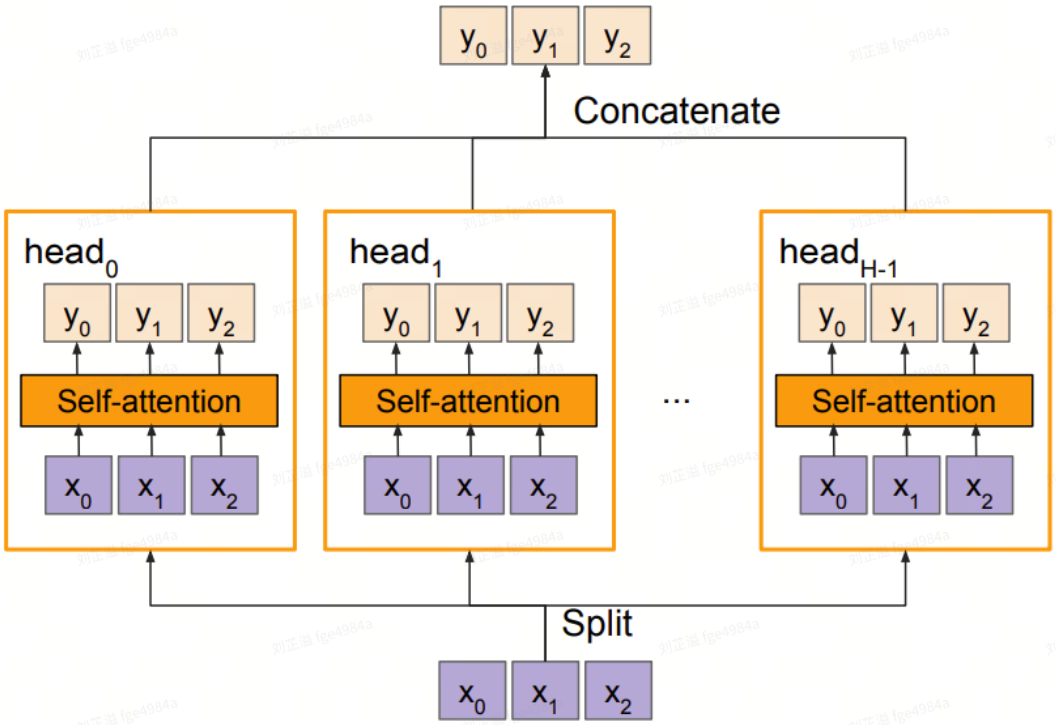
Inputs:
Input vectors: **x** (shape: N x D)

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

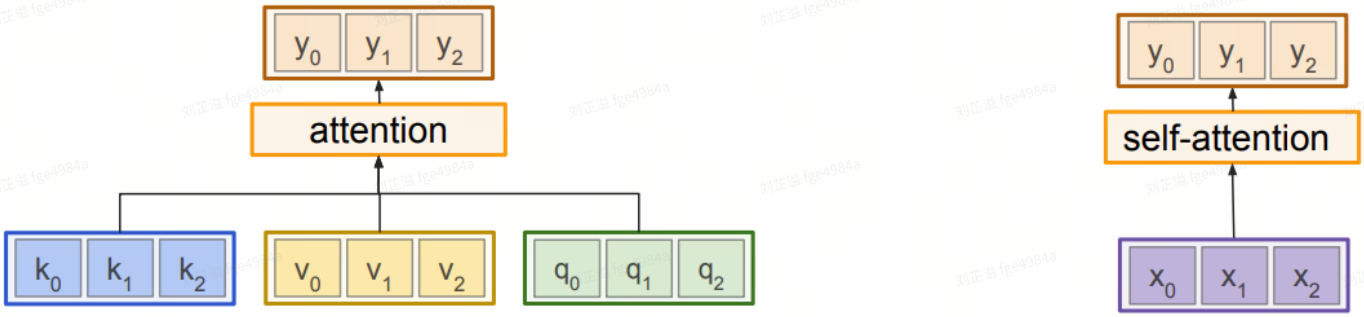
Multi-head self-attention layer

Multi-head self-attention layer

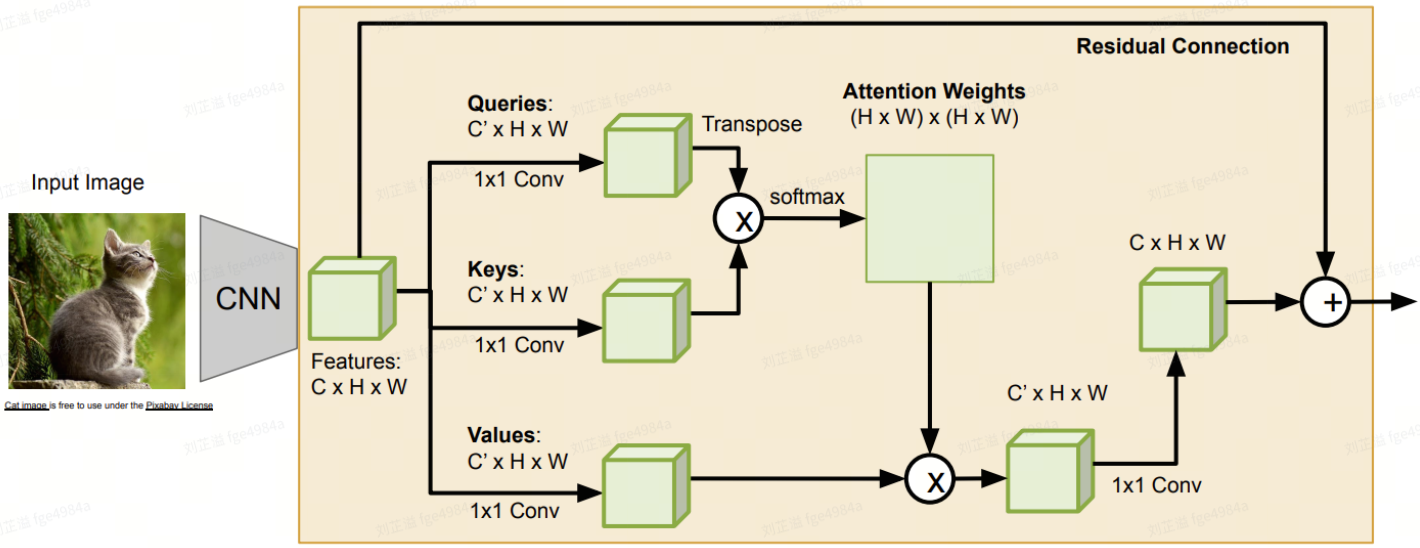
- Multiple self-attention heads in parallel



General attention versus self-attention



CNN with Self-Attention



Self-Attention Module

RNNs VS. Transformer

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)

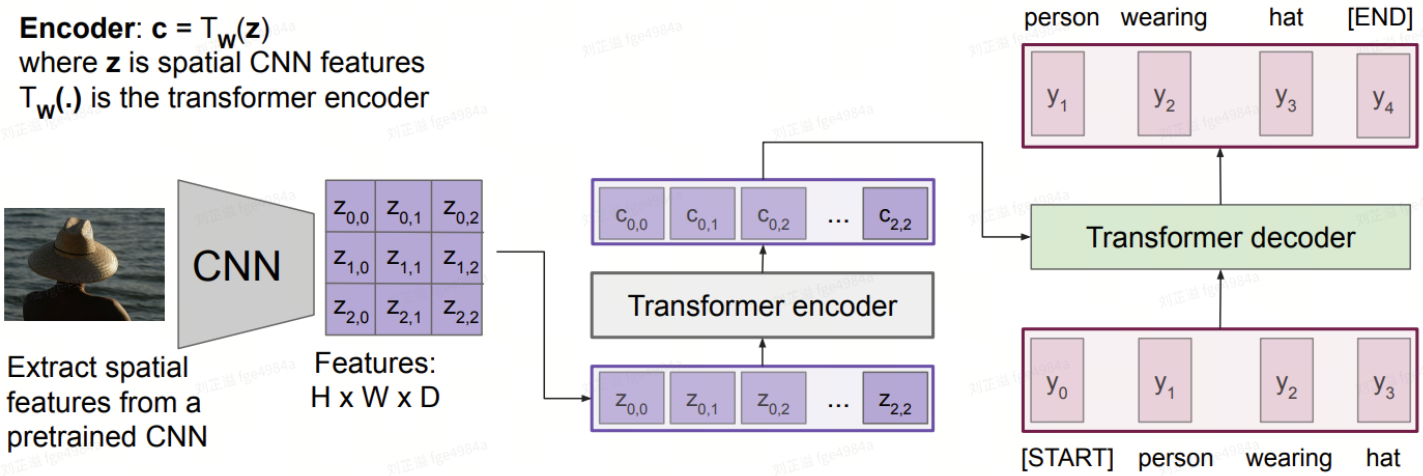
Transformer

Image Captioning using Transformers

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

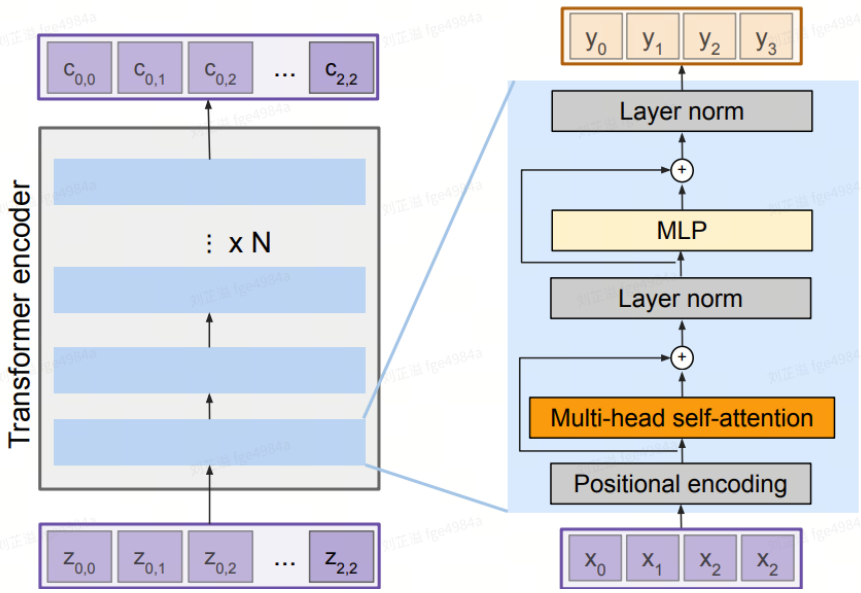
Decoder: $y_t = T_D(y_{0:t-1}, c)$
where $T_D(.)$ is the transformer decoder

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



Transformer encoder block

The Transformer encoder block



Transformer Encoder Block:

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

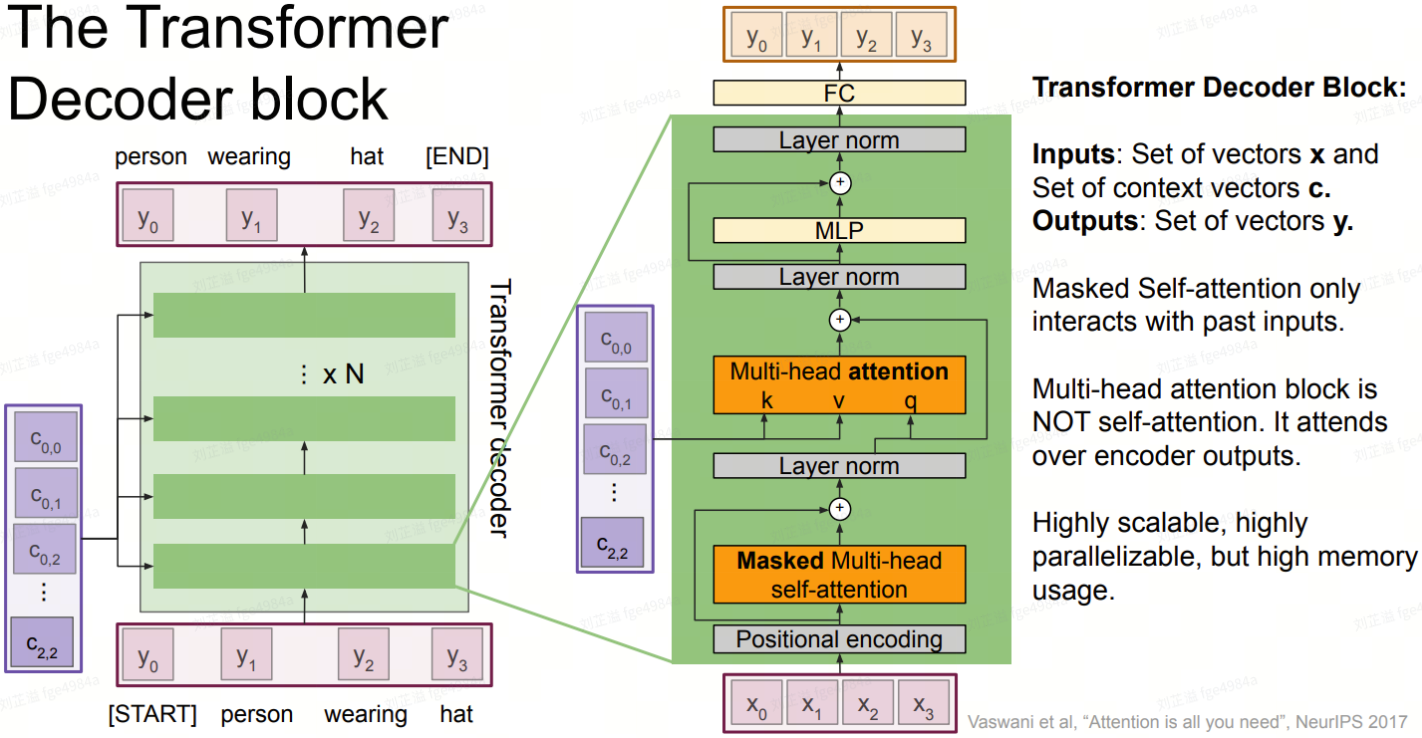
Self-attention is the only interaction between vectors.

Layer norm and MLP operate independently per vector.

Highly scalable, highly parallelizable, but high memory usage.

Transformer decoder block

The Transformer Decoder block



Transformer Decoder Block:

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

Masked Self-attention only interacts with past inputs.

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

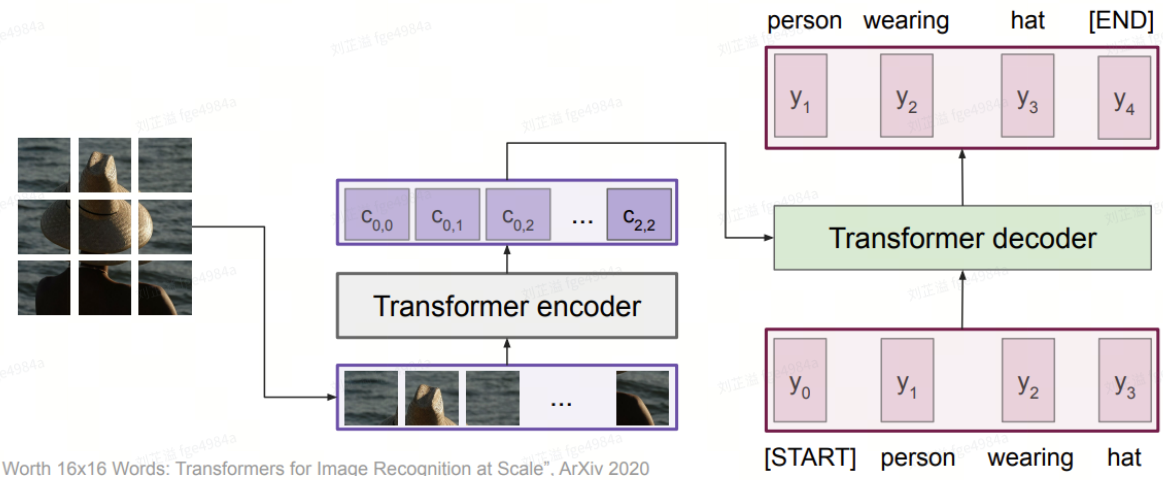
Highly scalable, highly parallelizable, but high memory usage.

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

Image Captioning using ONLY Tranformers

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