

计算机视觉

Computer Vision

Lecture 12: 注意力机制



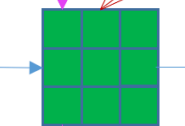
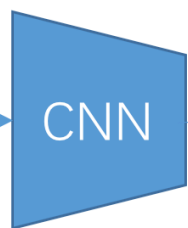
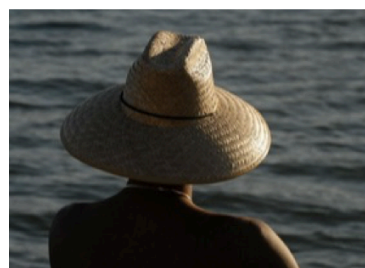
L10: 循环神经网络

图片描述：注意力机制的使用

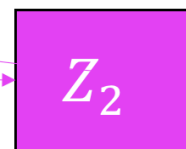
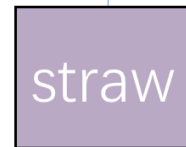
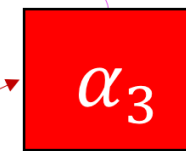
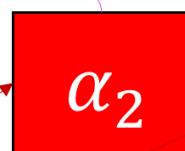
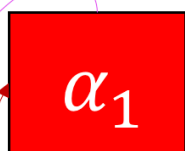
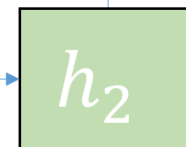
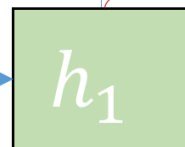
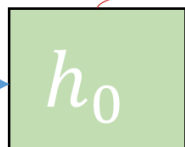
使用attention

$$e_{ti} = f_{\text{att}}(\mathbf{a}_i, \mathbf{h}_{t-1})$$

$$\alpha_{ti} = \frac{\exp(e_{ti})}{\sum_{k=1}^L \exp(e_{tk})}$$



$\mathbf{a} = L \times D$



$$\mathbf{Z}_t = \sum_{i=1}^L \mathbf{a}_i * \alpha_{ti}$$

加权特征向量

注意力向量

注意力机制

- RNNs中的注意力机制
 - ✓ 计算机视觉中的应用： 图片描述
 - ✓ 自然语言处理中的应用： 机器翻译
- 一般的注意力层结构（general attention layer）
 - ✓ 自注意力（self-attention）
 - ✓ 位置编码（positional encoding）
 - ✓ 遮挡的注意力（masked attention）
 - ✓ 多头注意力（multi-head attention）
- Transformers
 - ✓ 完全基于注意力机制的全新的神经网络结构（相对RNNs和CNNs）

注意力机制

- RNNs中的注意力机制
 - ✓ 计算机视觉中的应用： 图片描述
 - ✓ 自然语言处理中的应用： 机器翻译
- 一般的注意力层结构（general attention layer）
 - ✓ 自注意力（self-attention）
 - ✓ 位置编码（positional encoding）
 - ✓ 遮挡的注意力（masked attention）
 - ✓ 多头注意力（multi-head attention）
- Transformers
 - ✓ 完全基于注意力机制的全新的神经网络结构（相对RNNs和CNNs）

图片描述：不使用attention机制

Input: 图片 I

Output: 文本 $y = y_1, y_2, \dots, y_T$



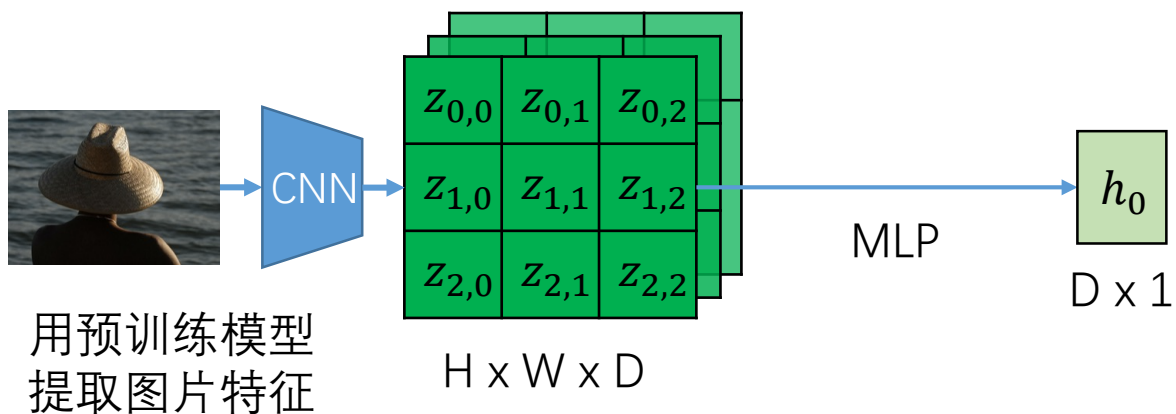
图片描述：不使用attention机制

Input: 图片 I

Output: 文本 $y = y_1, y_2, \dots, y_T$

Encoder: $h_0 = f_w(\mathbf{Z})$

- ✓ \mathbf{Z} : CNN提取的图片特征
- ✓ $f_w()$: Multi-Layer Perceptron (若干层FC)



图片描述：不使用attention机制

Input: 图片 I

Output: 文本 $y = y_1, y_2, \dots, y_T$

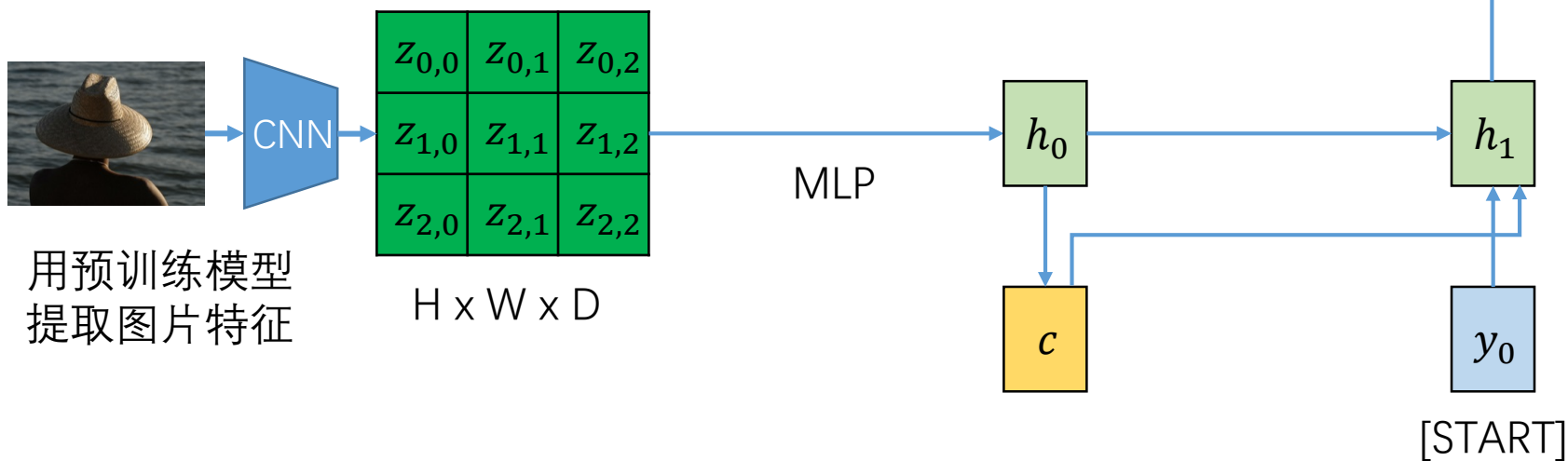
Decoder: $y_t = g(h_t) = g_w(y_{t-1}, h_{t-1}, c)$

✓ c : contex vector, 辅助文本预测, 通常设为 h_0

Encoder: $h_0 = f_w(Z)$

✓ Z : CNN提取的图片特征

✓ $f_w()$: Multi-Layer Perceptron (若干层FC)



图片描述：不使用attention机制

Input: 图片 I

Output: 文本 $y = y_1, y_2, \dots, y_T$

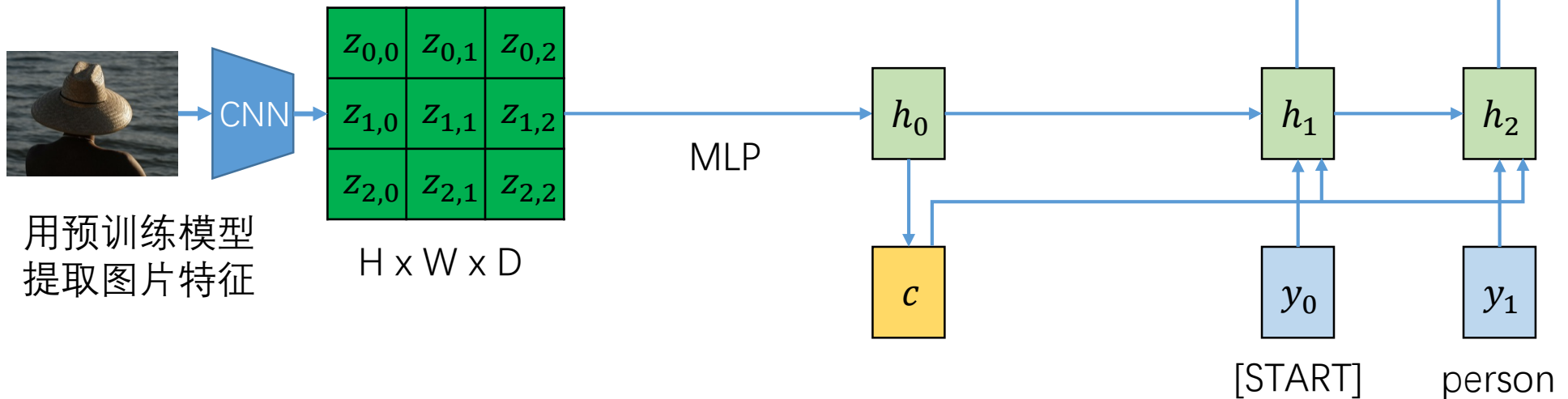
Decoder: $y_t = g(h_t) = g_w(y_{t-1}, h_{t-1}, c)$

✓ c : contex vector, 辅助文本预测, 通常设为 h_0

Encoder: $h_0 = f_w(Z)$

✓ Z : CNN提取的图片特征

✓ $f_w()$: Multi-Layer Perceptron (若干层FC)



图片描述：不使用attention机制

Input: 图片 I

Output: 文本 $y = y_1, y_2, \dots, y_T$

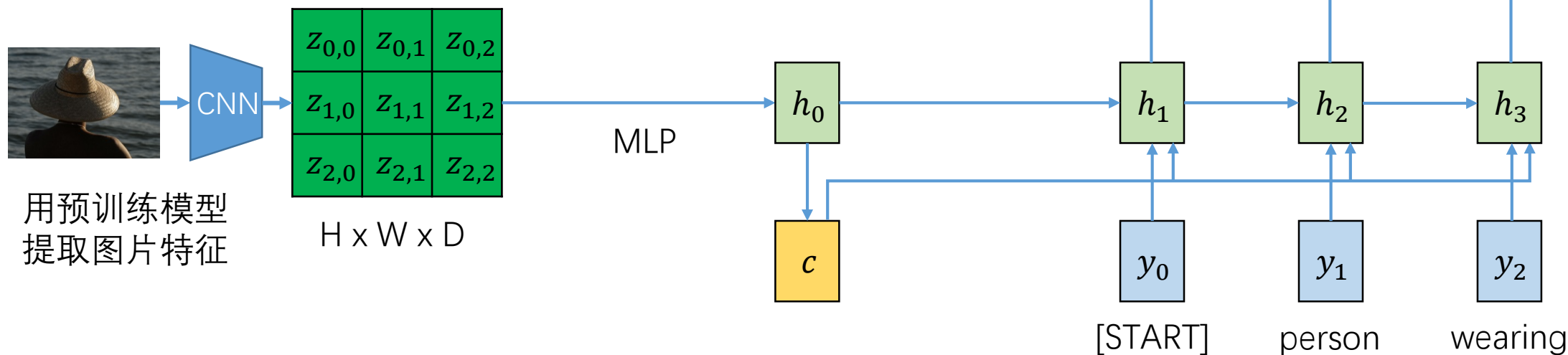
Decoder: $y_t = g(h_t) = g_w(y_{t-1}, h_{t-1}, c)$

✓ c : contex vector, 辅助文本预测, 通常设为 h_0

Encoder: $h_0 = f_w(Z)$

✓ Z : CNN提取的图片特征

✓ $f_w()$: Multi-Layer Perceptron (若干层FC)



图片描述：不使用attention机制

Input: 图片 I

Output: 文本 $y = y_1, y_2, \dots, y_T$

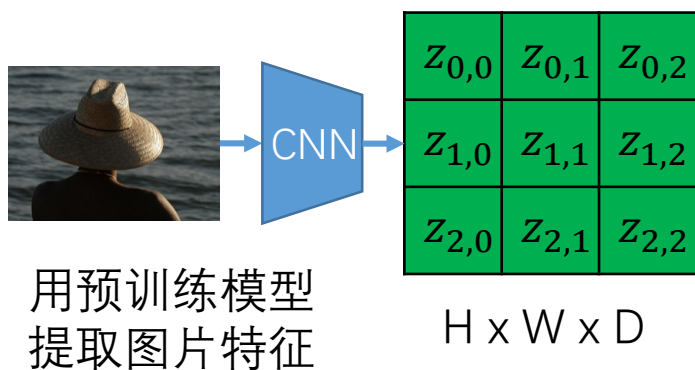
Decoder: $y_t = g(h_t) = g_w(y_{t-1}, h_{t-1}, c)$

✓ c : contex vector, 辅助文本预测, 通常设为 h_0

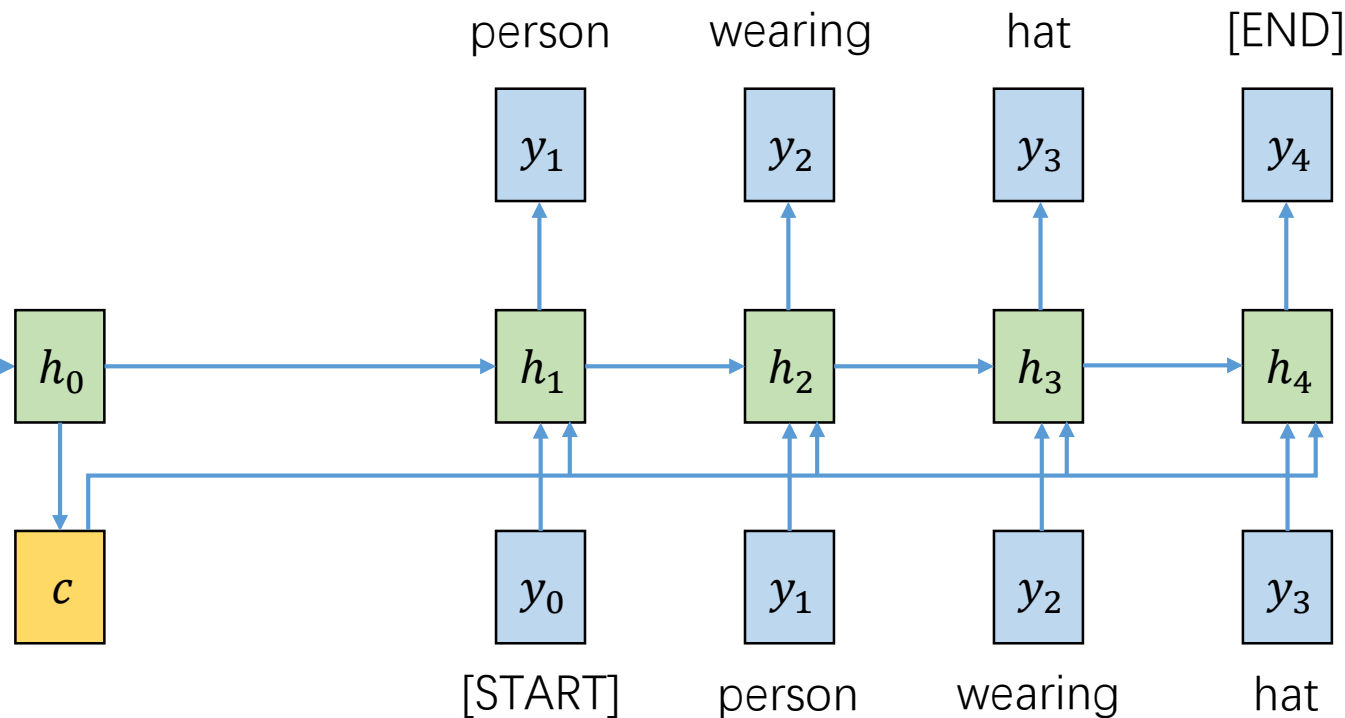
Encoder: $h_0 = f_w(Z)$

✓ Z : CNN提取的图片特征

✓ $f_w()$: Multi-Layer Perceptron (若干层FC)



MLP



图片描述：不使用attention机制

Input: 图片 I

Output: 文本 $y = y_1, y_2, \dots, y_T$

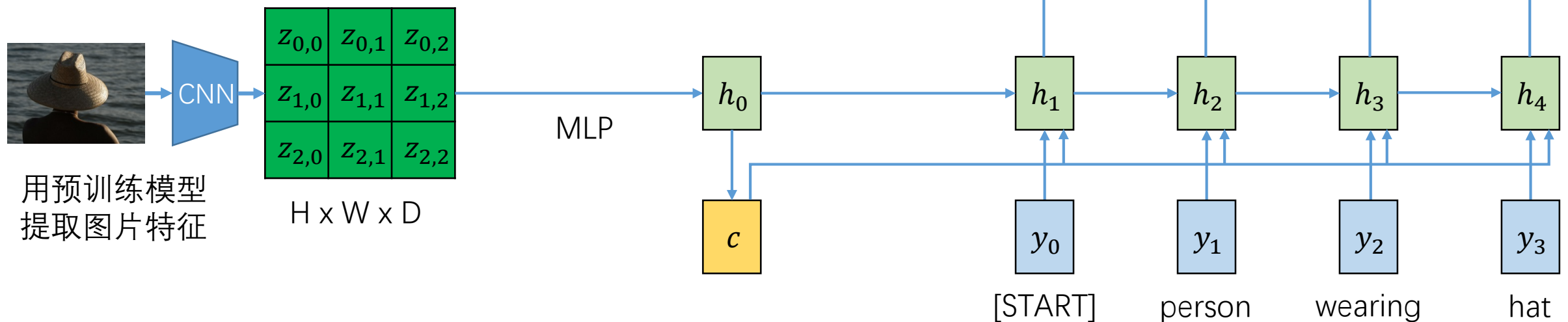
Encoder: $h_0 = f_w(\mathbf{Z})$

- ✓ \mathbf{Z} : CNN提取的图片特征
- ✓ $f_w()$: Multi-Layer Perceptron (若干层FC)

问题：所有时刻都使用相同的、代表整个图片信息的context vector c ，不利于不同时刻捕捉差异化的图片信息 (bottleneck)

Decoder: $y_t = g(h_t) = g_w(y_{t-1}, h_{t-1}, c)$

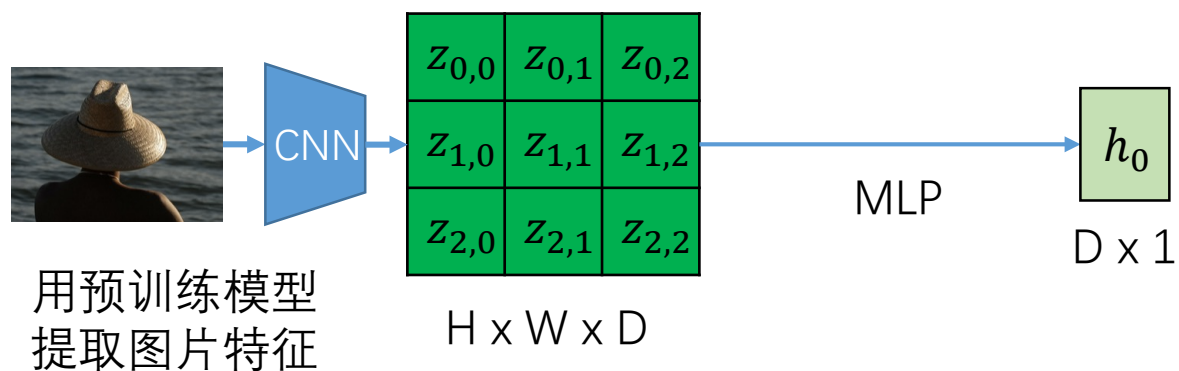
- ✓ c : context vector, 辅助文本预测，通常设为 h_0



图片描述：使用attention机制

attention想法：

- ✓ 模拟人类视觉，描述图片时会注意图片不同的区域
- ✓ 每个时刻使用一个新的context vector，表示图片不同区域的特征

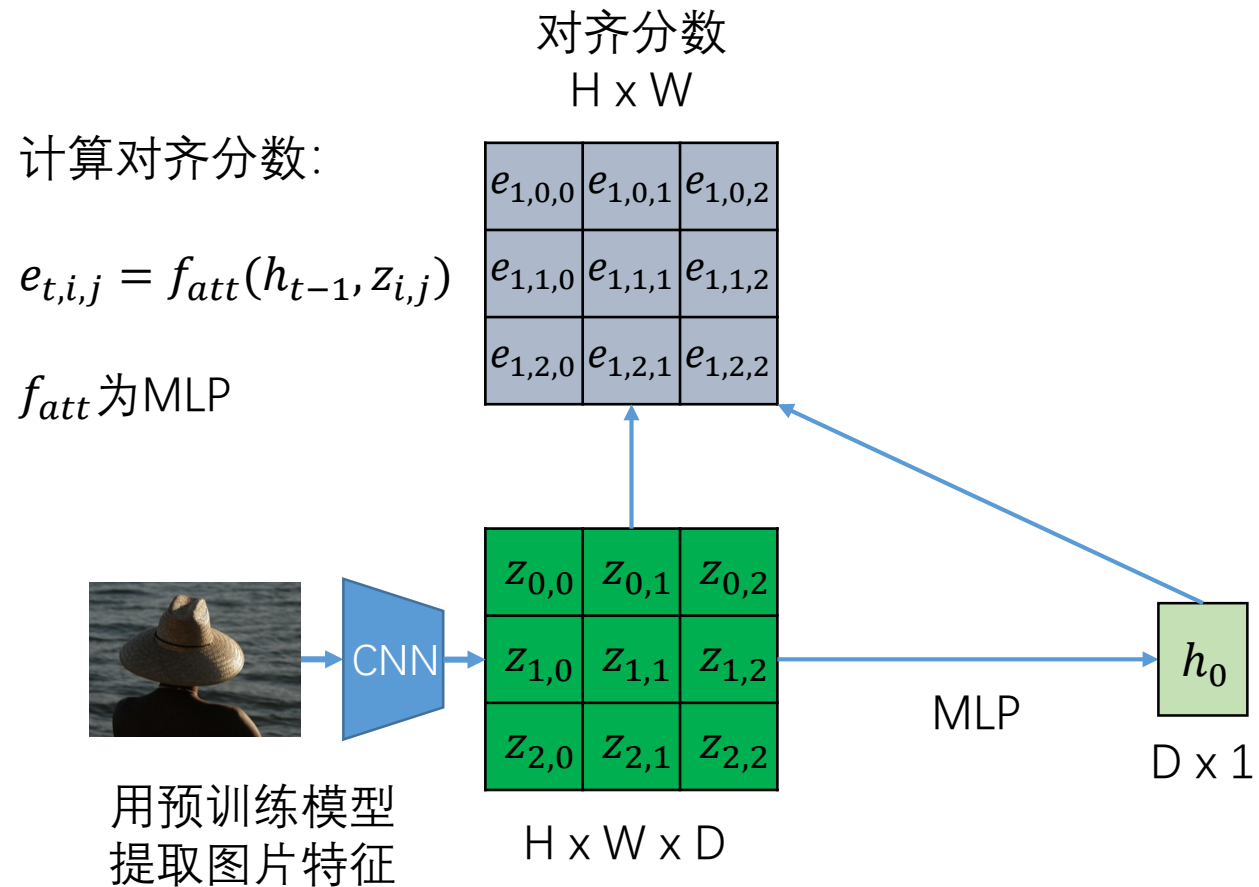


用预训练模型
提取图片特征

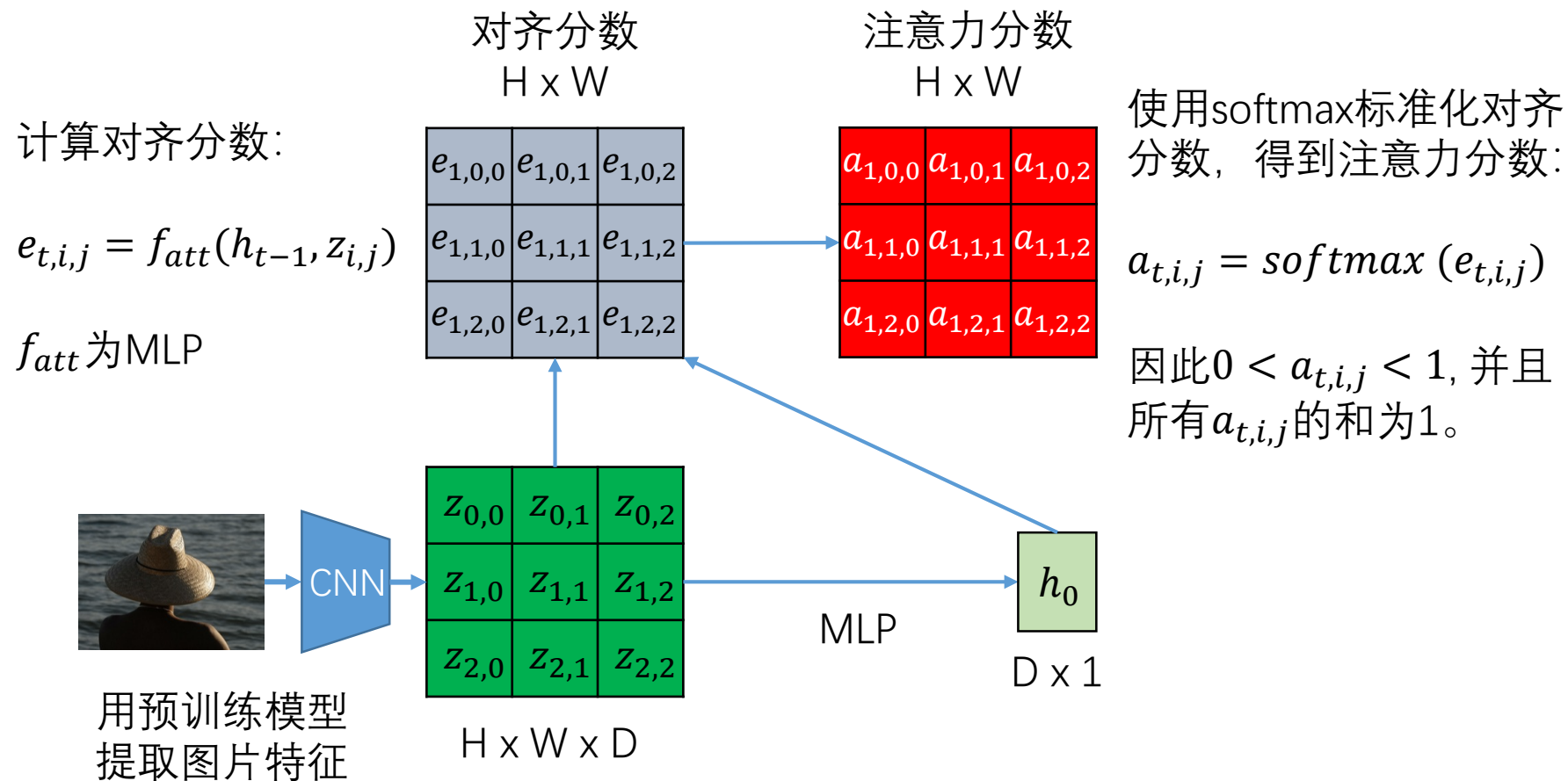
MLP

$D \times 1$

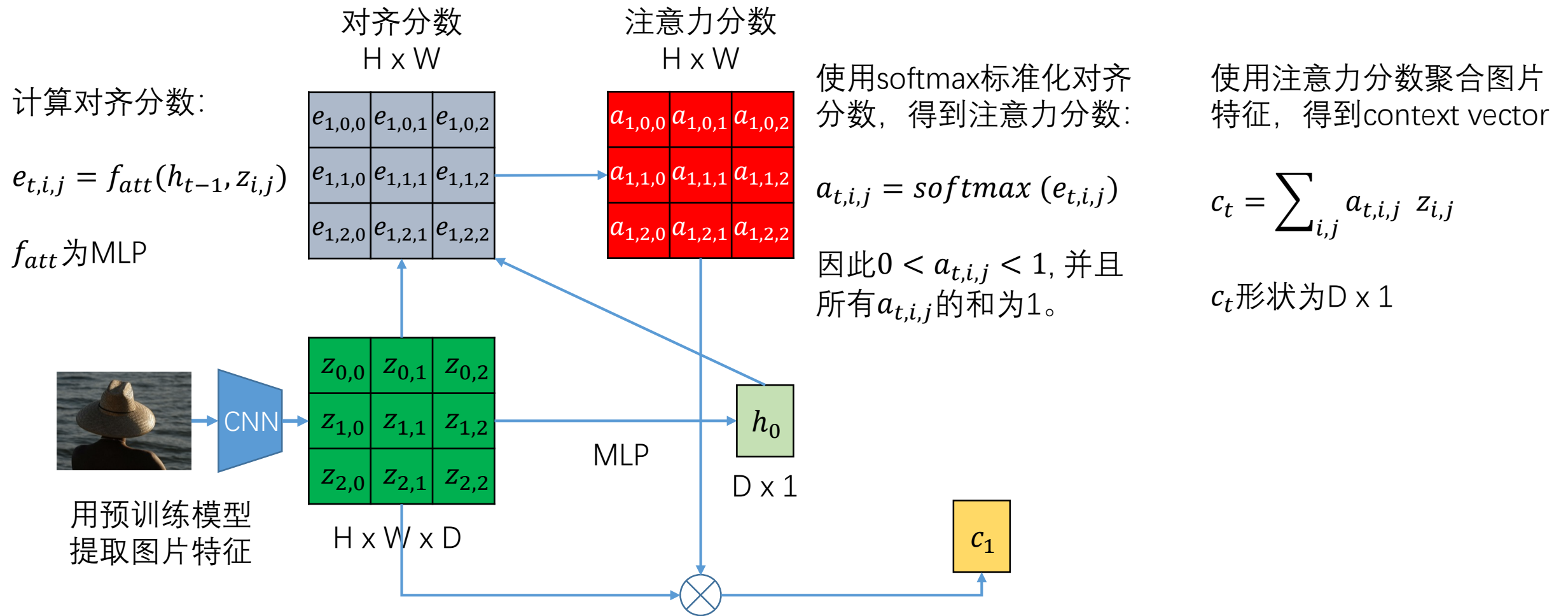
图片描述：使用attention机制



图片描述：使用attention机制



图片描述：使用attention机制



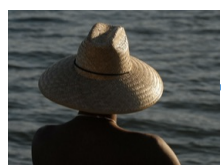
图片描述：使用attention机制

注意力计算：

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

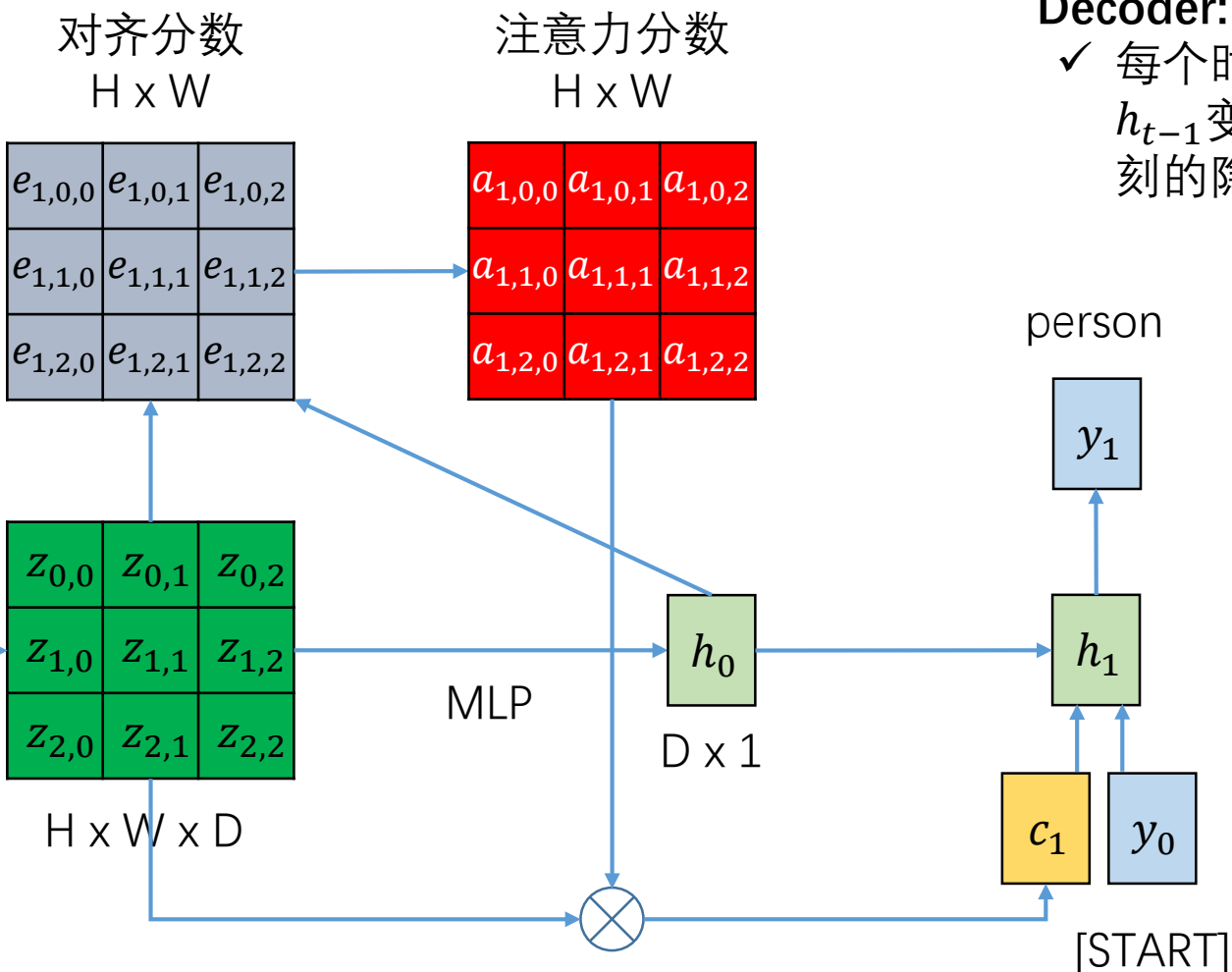
$$a_{t,i,j} = \text{softmax}(e_{t,i,j})$$

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



用预训练模型
提取图片特征

CNN



Decoder: $y_t = g(h_t) = g_w(y_{t-1}, h_{t-1}, c_t)$

- ✓ 每个时刻 t 的context vector,由固定的 c 改为随 h_{t-1} 变化的 c_t , 即每个时刻可以根据前一时刻的隐层状态关注图片的不同区域

图片描述：使用attention机制

注意力计算：

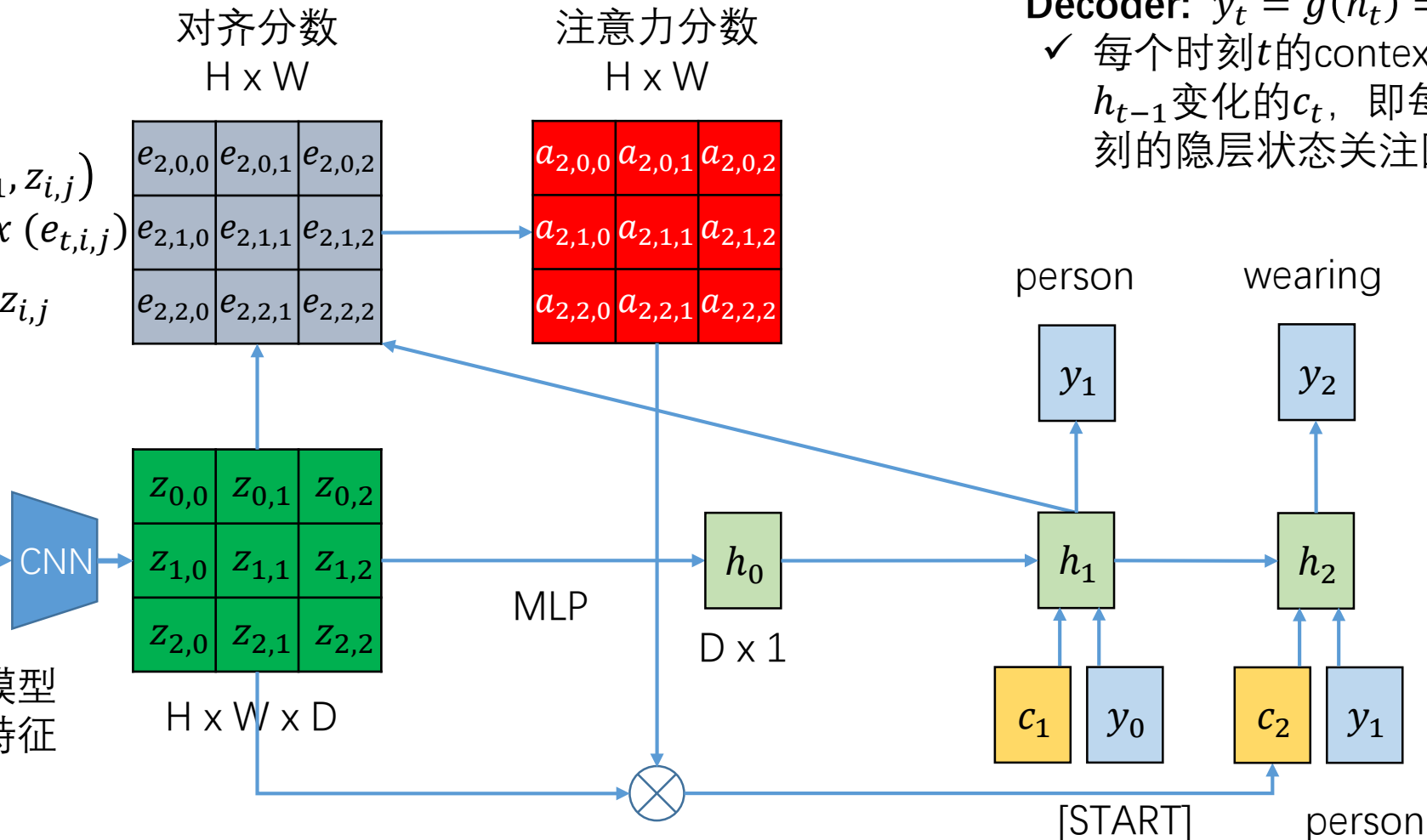
$$e_{t,i,j} = f_{att}(h_{t-1}, z_{i,j})$$

$$a_{t,i,j} = \text{softmax}(e_{t,i,j})$$

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



用预训练模型
提取图片特征



Decoder: $y_t = g(h_t) = g_w(y_{t-1}, h_{t-1}, c_t)$

✓ 每个时刻 t 的 context vector, 由固定的 c 改为随 h_{t-1} 变化的 c_t , 即每个时刻可以根据前一时刻的隐层状态关注图片的不同区域

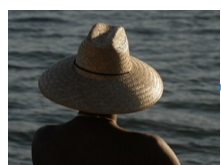
图片描述：使用attention机制

注意力计算：

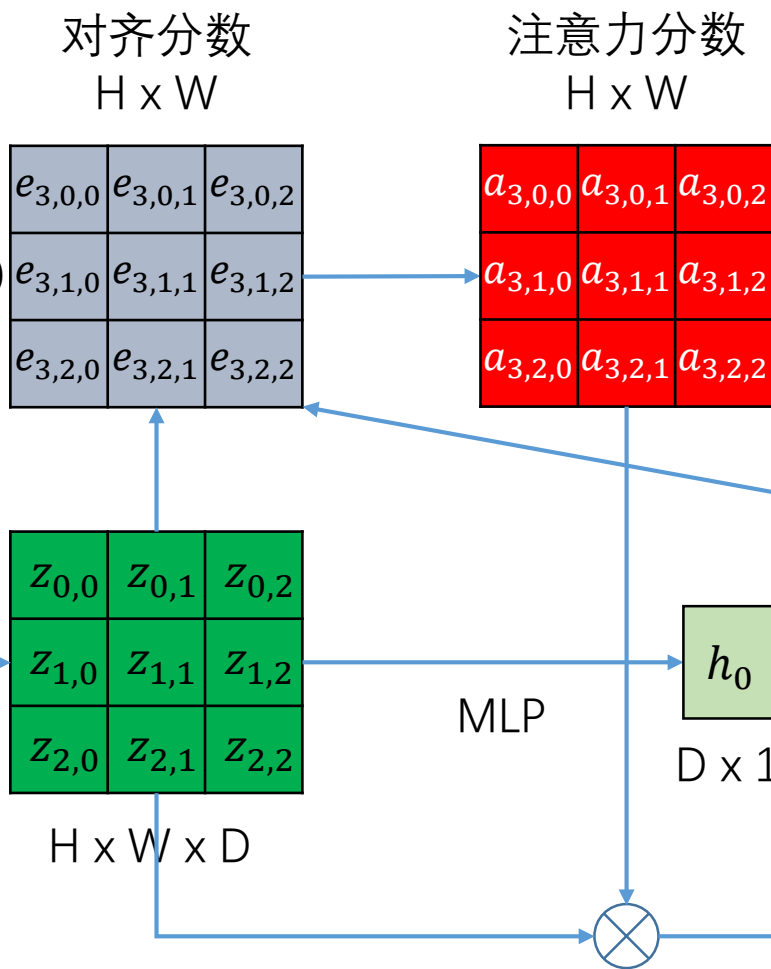
$$e_{t,i,j} = f_{att}(h_{t-1}, z_{i,j})$$

$$a_{t,i,j} = \text{softmax}(e_{t,i,j})$$

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

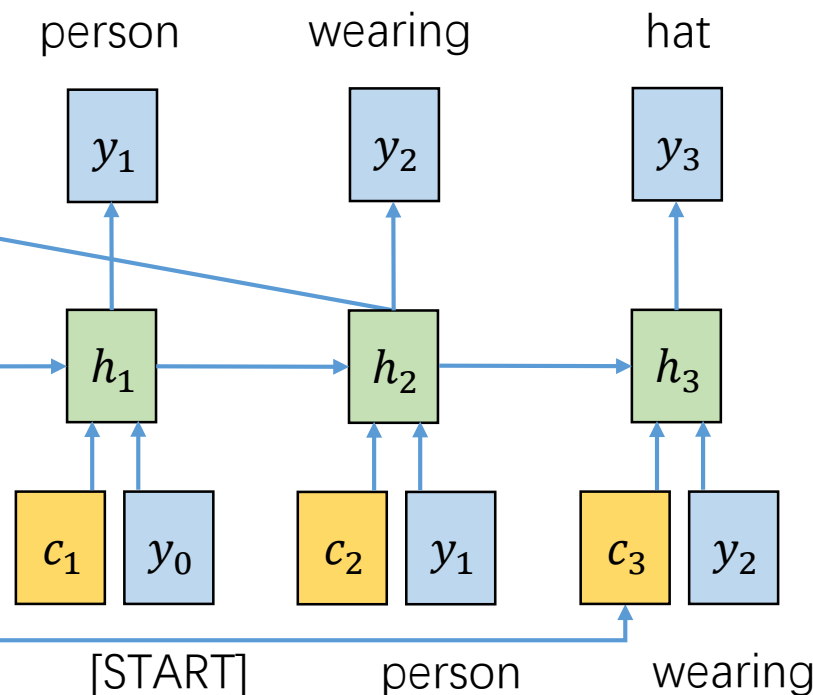


用预训练模型
提取图片特征



Decoder: $y_t = g(h_t) = g_w(y_{t-1}, h_{t-1}, c_t)$

✓ 每个时刻 t 的context vector,由固定的 c 改为随 h_{t-1} 变化的 c_t , 即每个时刻可以根据前一时刻的隐层状态关注图片的不同区域



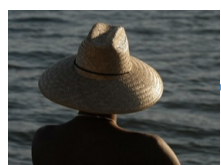
图片描述：使用attention机制

注意力计算：

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

$$a_{t,i,j} = \text{softmax}(e_{t,i,j})$$

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



用预训练模型
提取图片特征

CNN

对齐分数
H x W

$e_{4,0,0}$	$e_{4,0,1}$	$e_{4,0,2}$
$e_{4,1,0}$	$e_{4,1,1}$	$e_{4,1,2}$
$e_{4,2,0}$	$e_{4,2,1}$	$e_{4,2,2}$

注意力分数
H x W

$a_{4,0,0}$	$a_{4,0,1}$	$a_{4,0,2}$
$a_{4,1,0}$	$a_{4,1,1}$	$a_{4,1,2}$
$a_{4,2,0}$	$a_{4,2,1}$	$a_{4,2,2}$

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

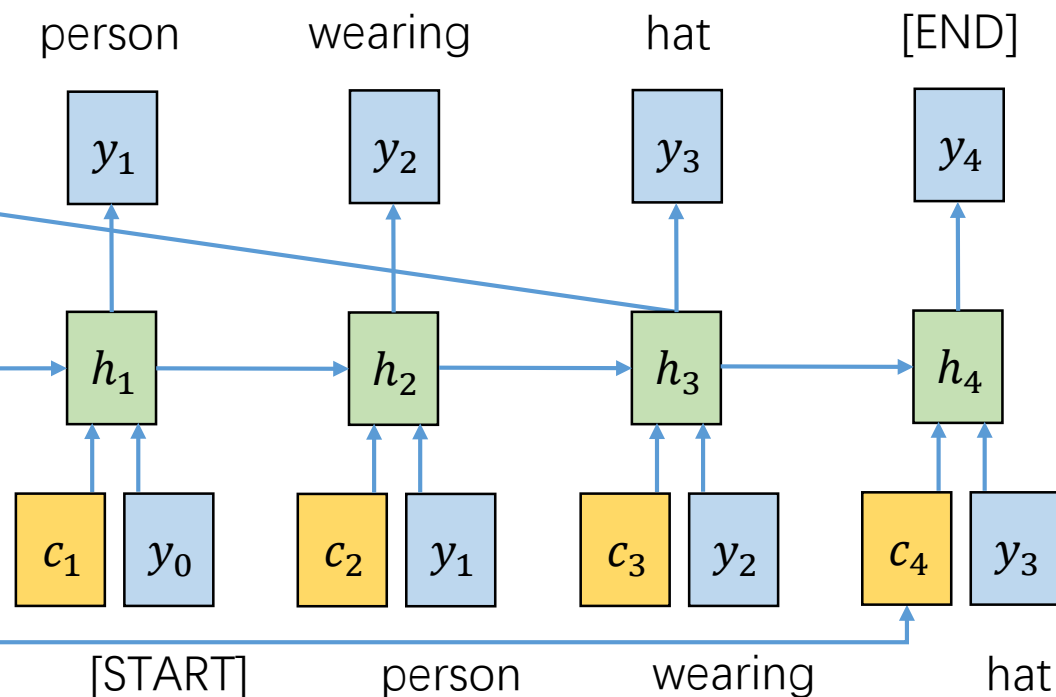
H x W x D

MLP

h_0
D x 1

Decoder: $y_t = g(h_t) = g_w(y_{t-1}, h_{t-1}, c_t)$

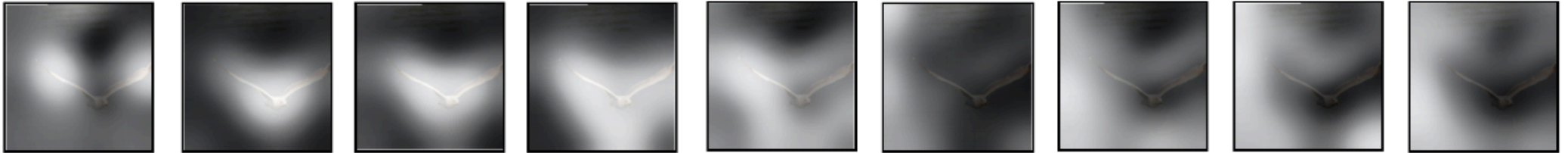
✓ 每个时刻 t 的context vector,由固定的 c 改为随 h_{t-1} 变化的 c_t , 即每个时刻可以根据前一时刻的隐层状态关注图片的不同区域



图片描述：使用attention机制

Figure 3. Visualization of the attention for each generated word. The rough visualizations obtained by upsampling the attention weights and smoothing. (top) “soft” and (bottom) “hard” attention (note that both models generated the same captions in this example).

soft attention



hard attention
(Monte Carlo
based sampling
or REINFORCE)



A

bird

flying

over

a

body

of

water

.

图片描述：使用attention机制

Figure 4. Examples of attending to the correct object (*white* indicates the attended regions, *underlines* indicated the corresponding word)



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.

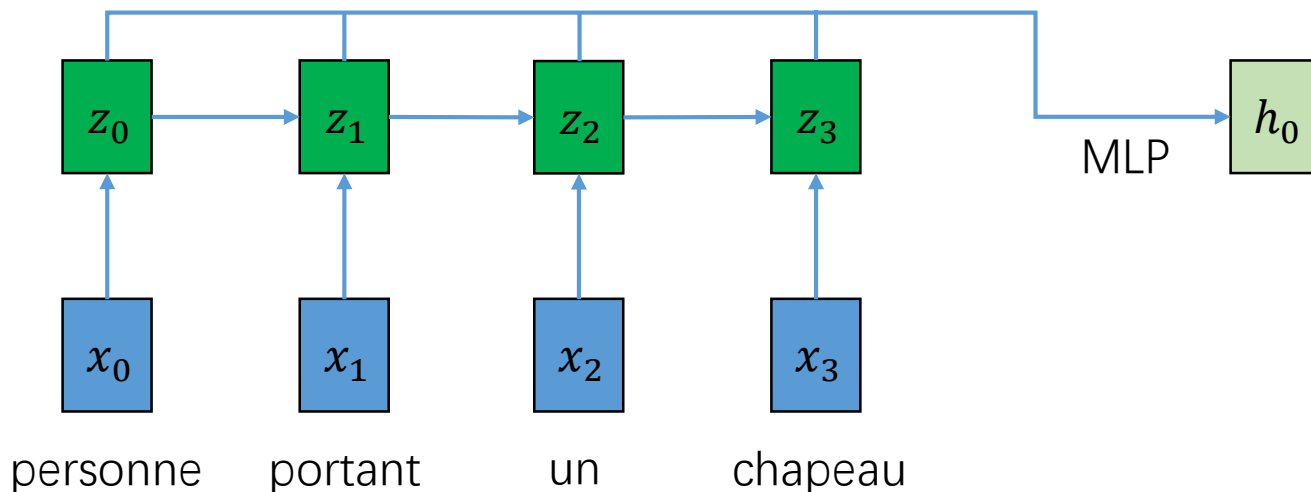
NLP应用： 机器翻译（不使用attention）

Input: 文本 $\mathbf{x} = x_1, x_2, \dots, x_T$

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

Encoder: $h_0 = f_w(\mathbf{Z})$

- ✓ z_t : hidden states
- ✓ h_0 : 通过 f_w 聚合的 hidden state, 输入到 decoder



NLP应用： 机器翻译（不使用attention）

Input: 文本 $x = x_1, x_2, \dots, x_T$

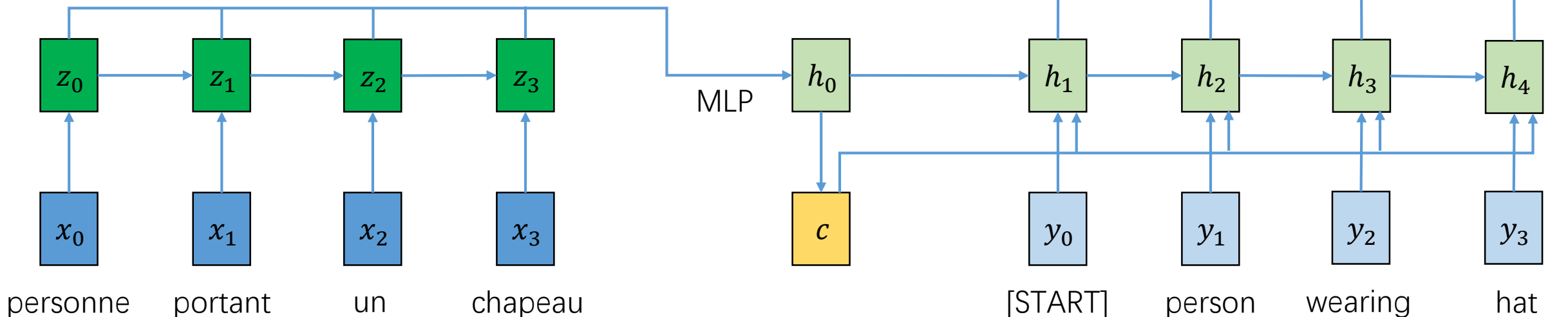
Output: 文本 $y = y_1, y_2, \dots, y_T$

Decoder: $y_t = g(h_t) = g_w(y_{t-1}, h_{t-1}, c)$
 ✓ c : contex vector, 辅助文本预测, 通常设为 h_0

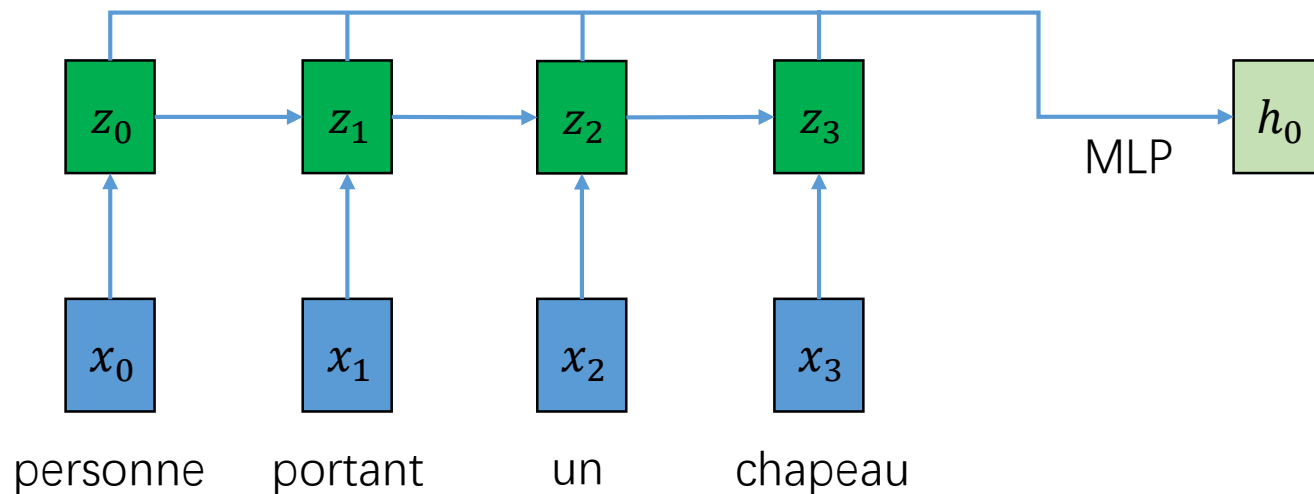
Encoder: $h_0 = f_w(\mathbf{Z})$

✓ z_t : hidden states

✓ h_0 : 通过 f_w 聚合的hidden state, 输入到decoder



NLP应用： 机器翻译（使用attention）



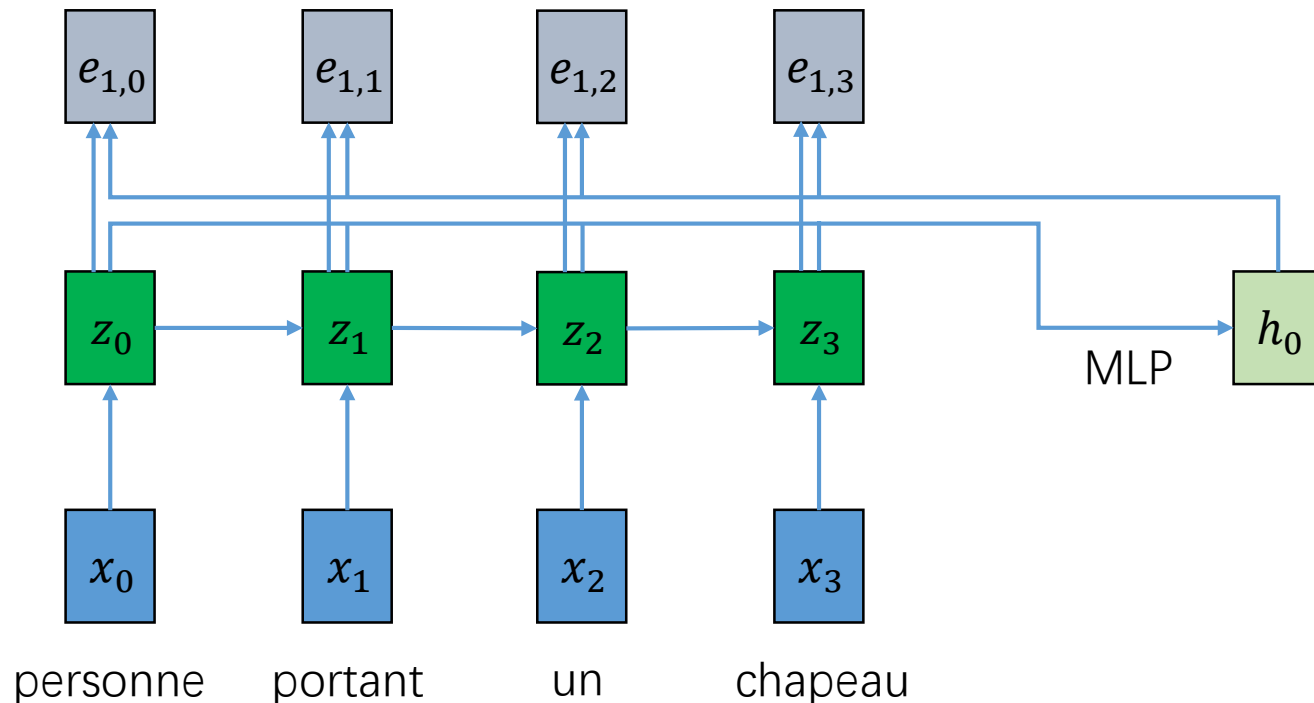
Bahdanau, D., Cho, K. and Bengio, Y. Neural machine translation by jointly learning to align and translate. ICLR 2015.

NLP应用： 机器翻译（使用attention）

计算对齐分数：

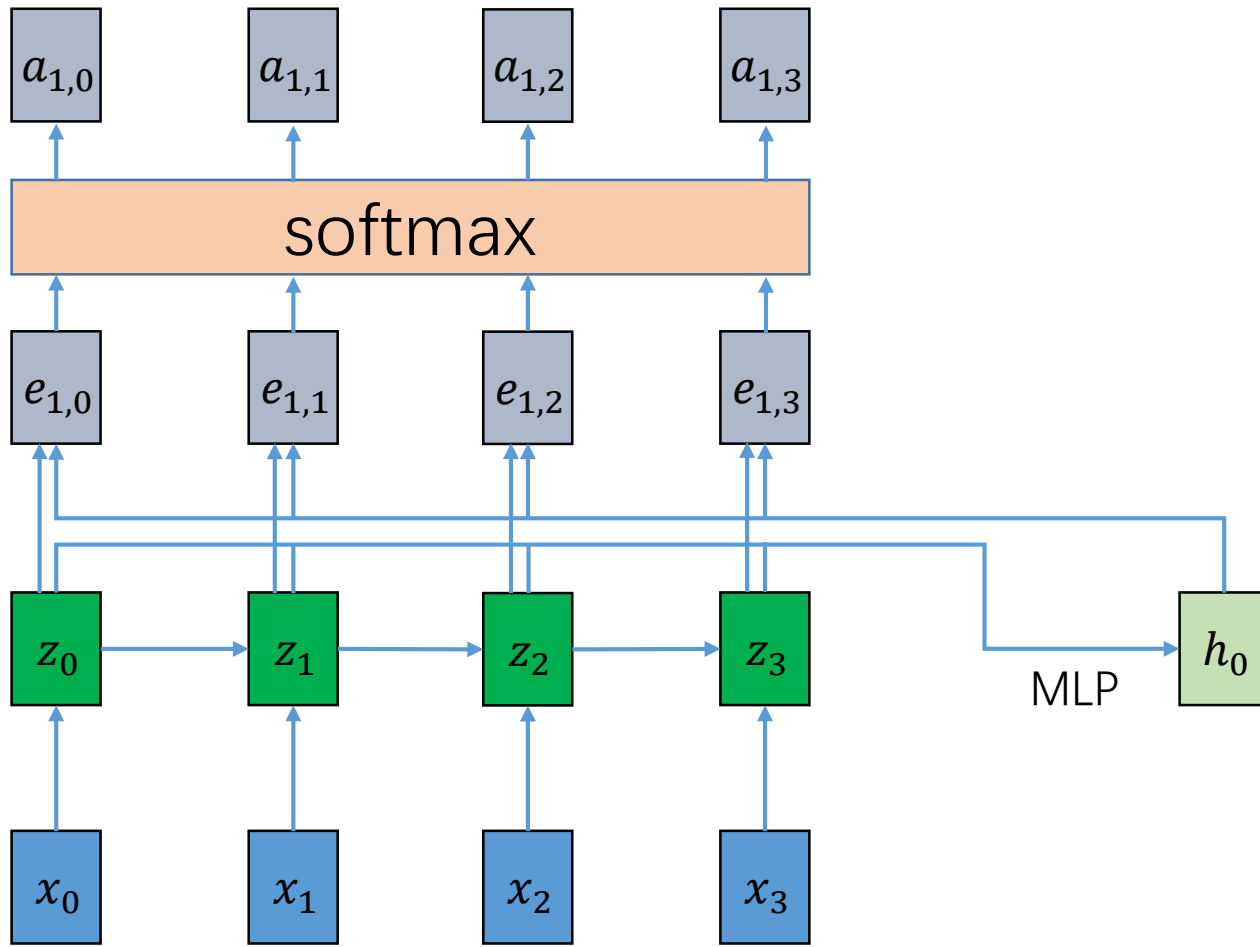
$$e_{t,i} = f_{att}(h_{t-1}, z_i)$$

f_{att} 为MLP



Bahdanau, D., Cho, K. and Bengio, Y. Neural machine translation by jointly learning to align and translate. ICLR 2015.

NLP应用：机器翻译（使用attention）



计算对齐分数:

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

f_{att} 为MLP

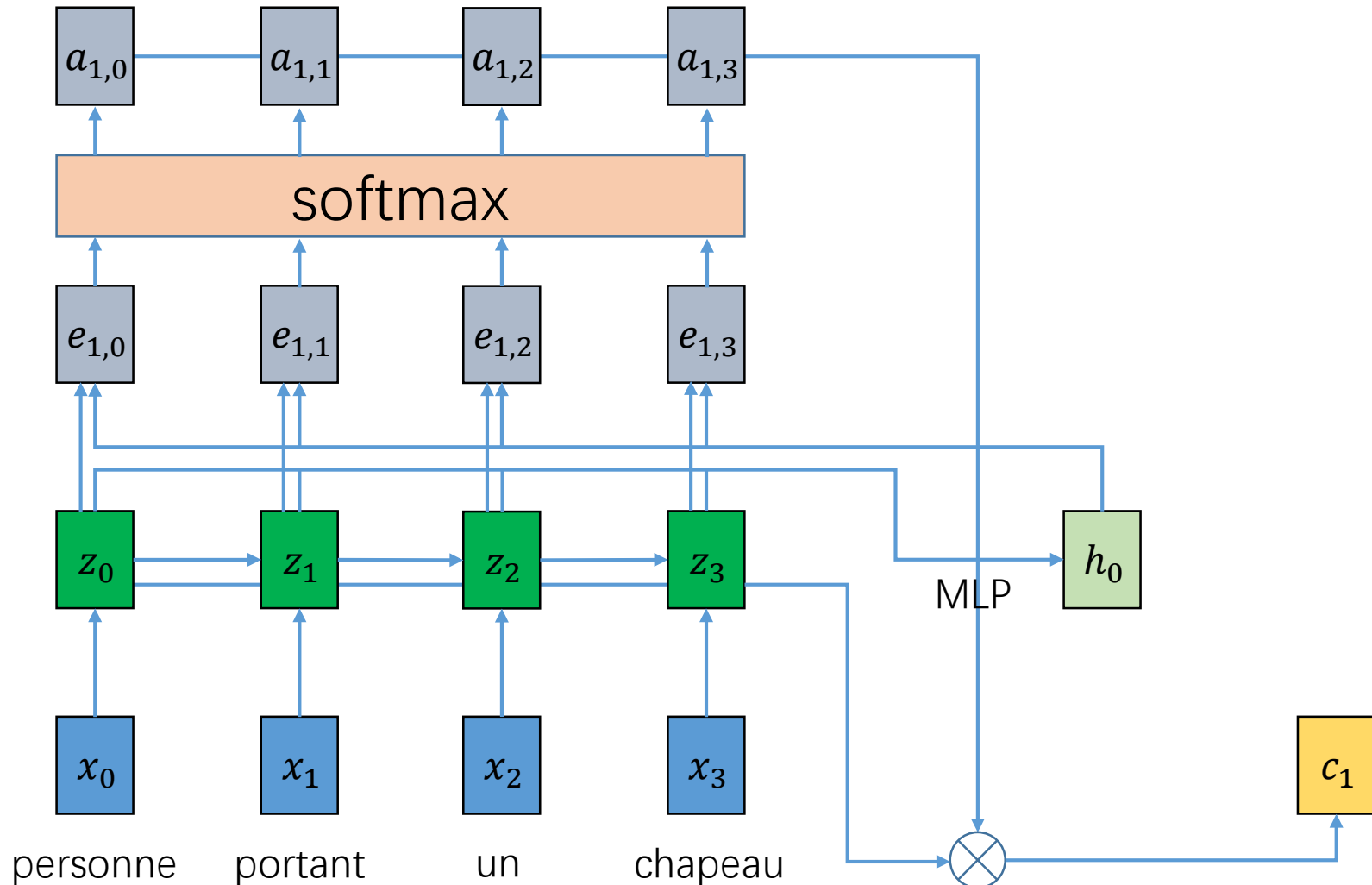
计算注意力分数:

$$a_{t,i} = \text{softmax}(e_{t,i})$$

personne portant un chapeau

Bahdanau, D., Cho, K. and Bengio, Y. Neural machine translation by jointly learning to align and translate. ICLR 2015.

NLP应用：机器翻译（使用attention）



计算对齐分数:

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

f_{att} 为MLP

计算注意力分数:

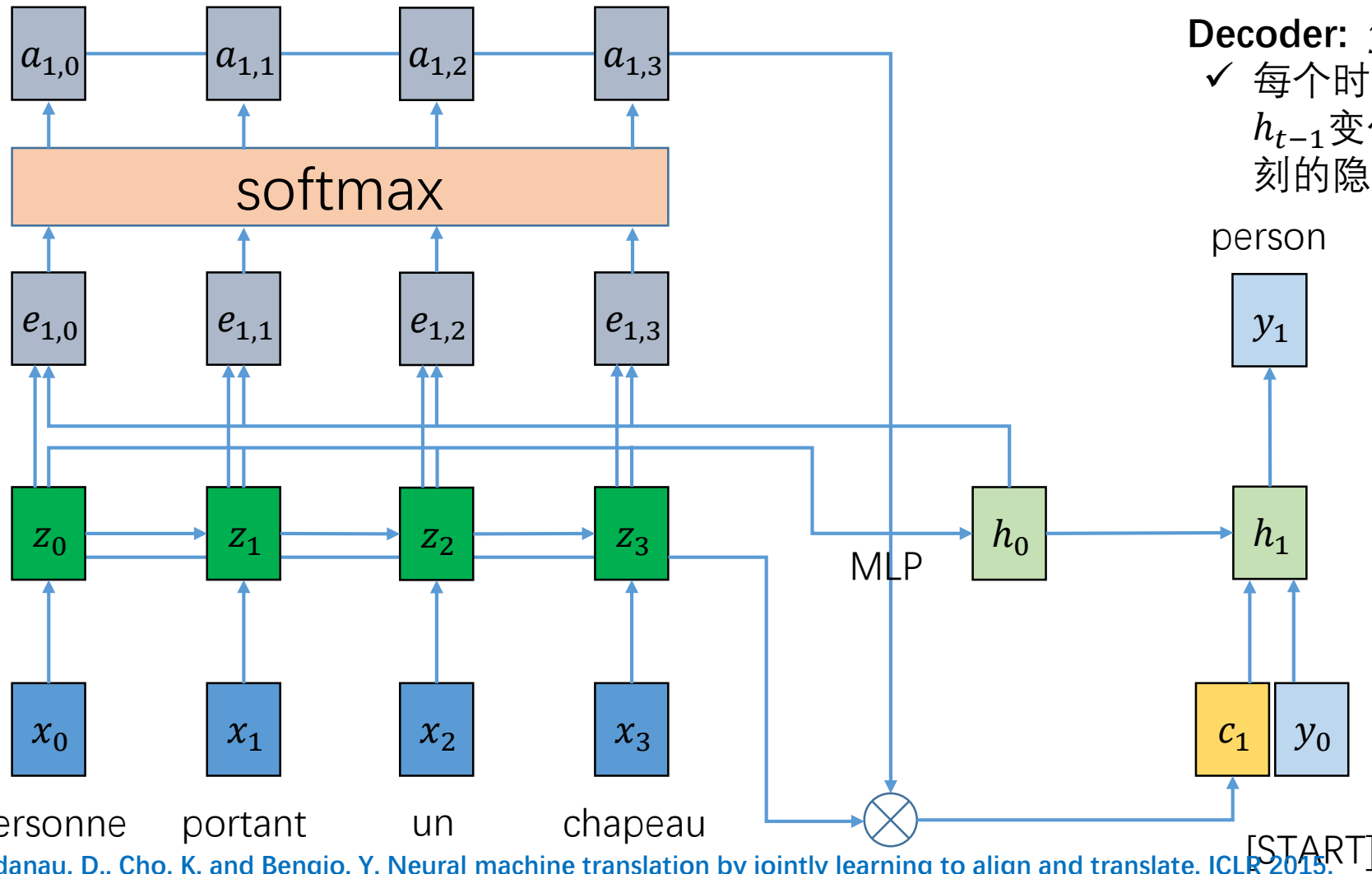
$$a_{t,i} = \text{softmax}(e_{t,i})$$

计算context vector:

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

Bahdanau, D., Cho, K. and Bengio, Y. Neural machine translation by jointly learning to align and translate. ICLR 2015.

NLP应用：机器翻译（使用attention）



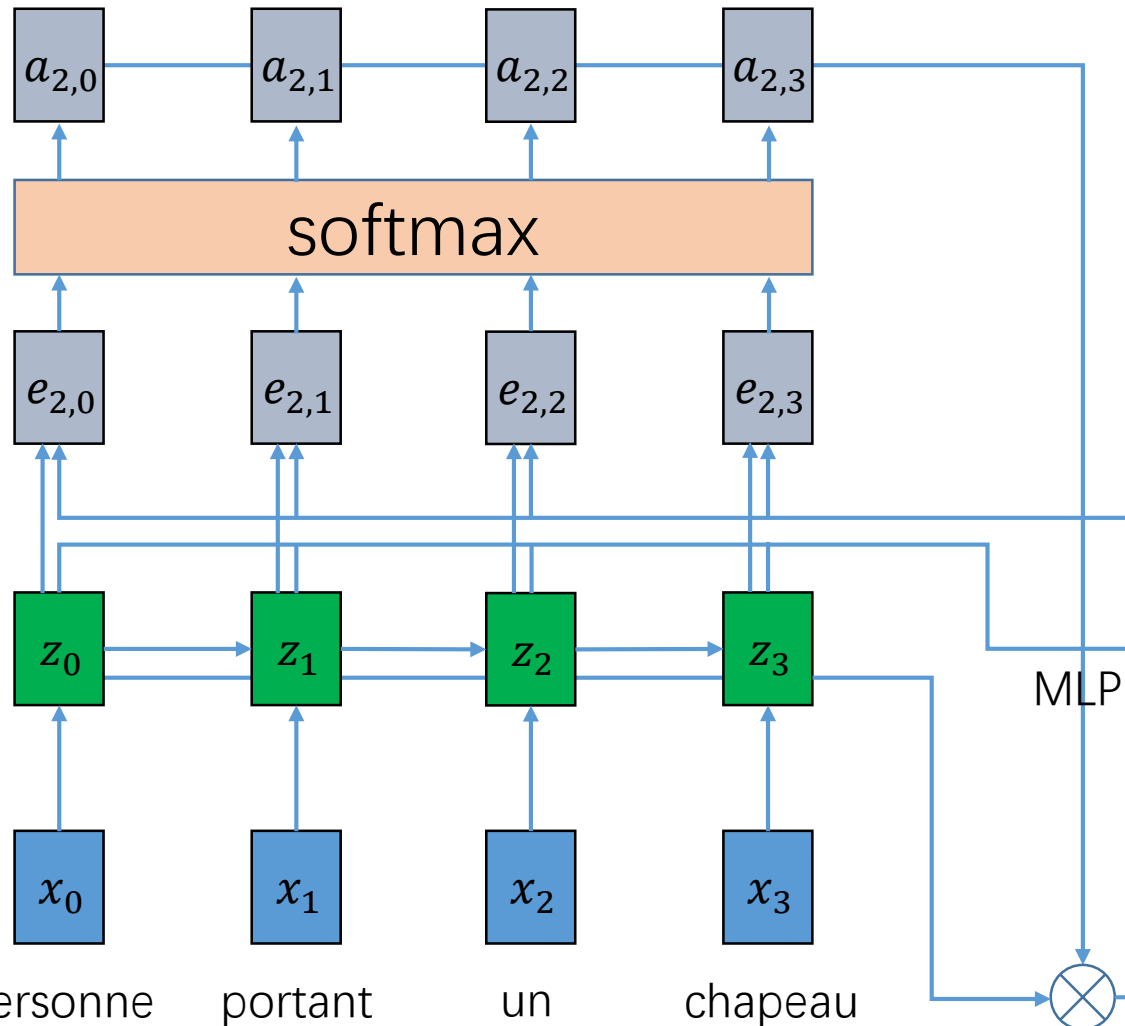
Decoder: $y_t = g(h_t) = g_w(y_{t-1}, h_{t-1}, c_t)$

- ✓ 每个时刻 t 的context vector,由固定的 c 改为随 h_{t-1} 变化的 c_t , 即每个时刻可以根据前一时刻的隐层状态关注输入文本的不同部分

[START]

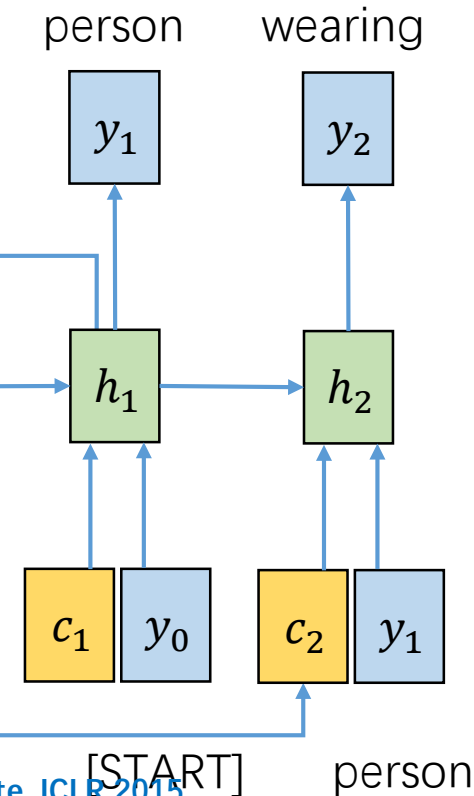
Bahdanau, D., Cho, K. and Bengio, Y. Neural machine translation by jointly learning to align and translate. ICLR 2015.

NLP应用：机器翻译（使用attention）



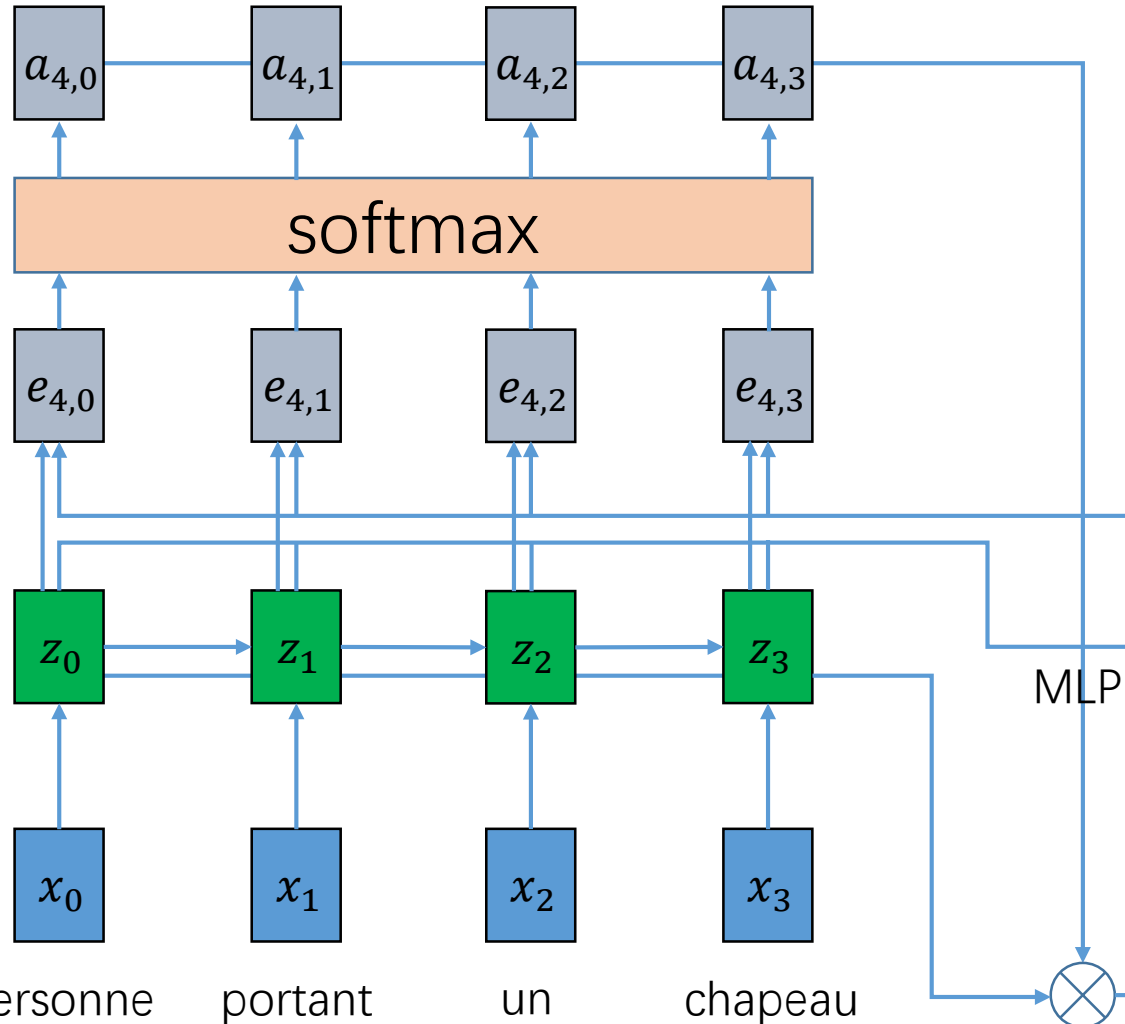
Decoder: $y_t = g(h_t) = g_w(y_{t-1}, h_{t-1}, c_t)$

- ✓ 每个时刻 t 的context vector,由固定的 c 改为随 h_{t-1} 变化的 c_t , 即每个时刻可以根据前一时刻的隐层状态关注输入文本的不同部分



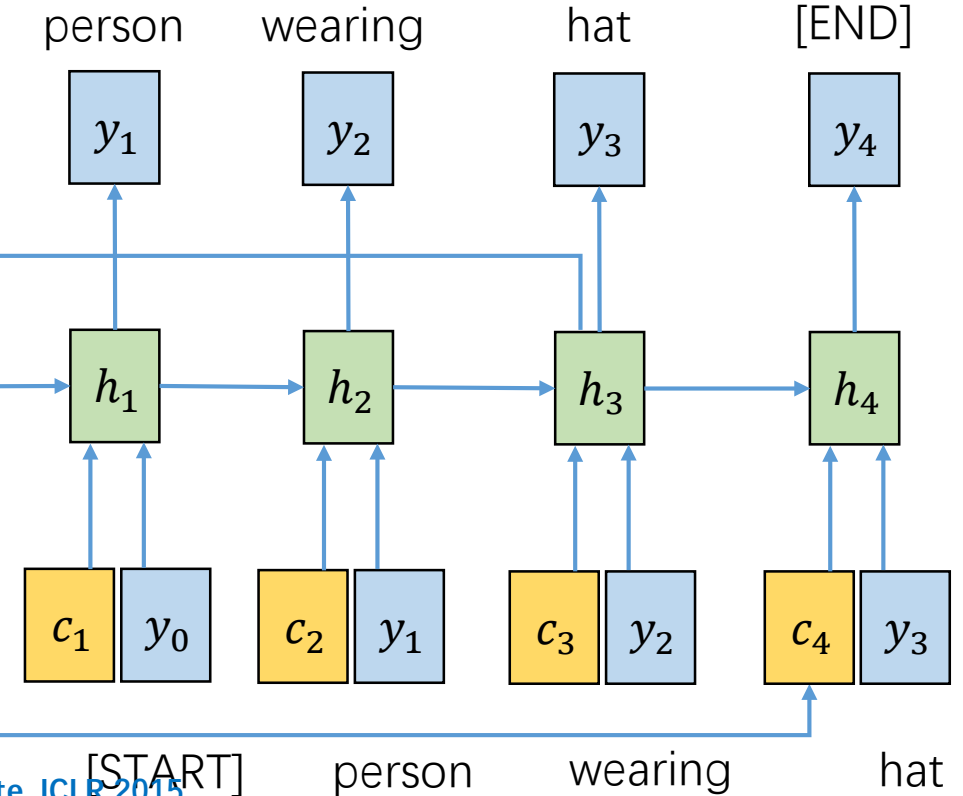
Bahdanau, D., Cho, K. and Bengio, Y. Neural machine translation by jointly learning to align and translate. ICLR 2015.

NLP应用：机器翻译（使用attention）



Decoder: $y_t = g(h_t) = g_w(y_{t-1}, h_{t-1}, c_t)$

- ✓ 每个时刻 t 的context vector,由固定的 c 改为随 h_{t-1} 变化的 c_t , 即每个时刻可以根据前一时刻的隐层状态关注输入文本的不同部分

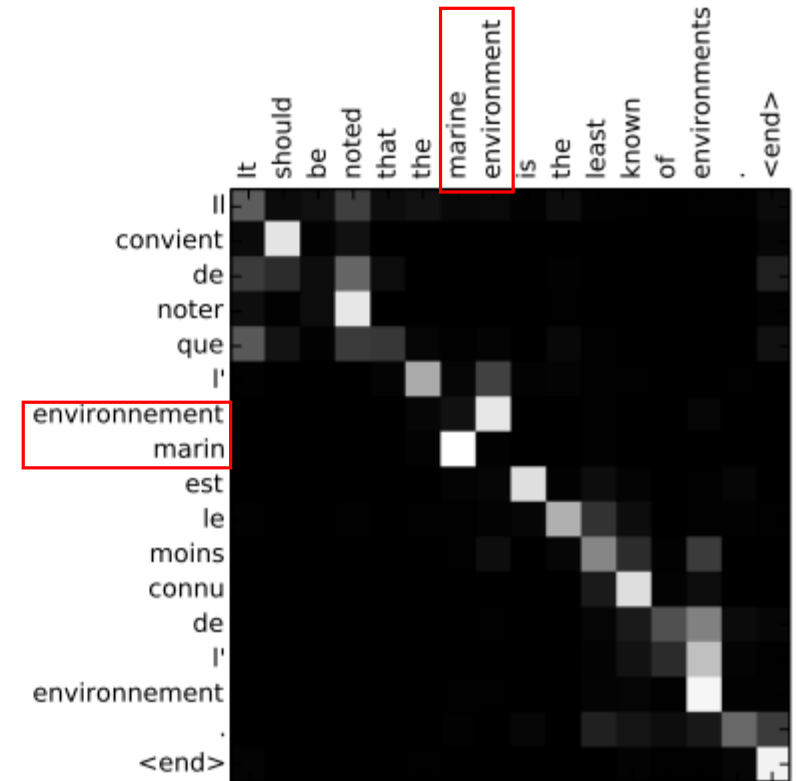
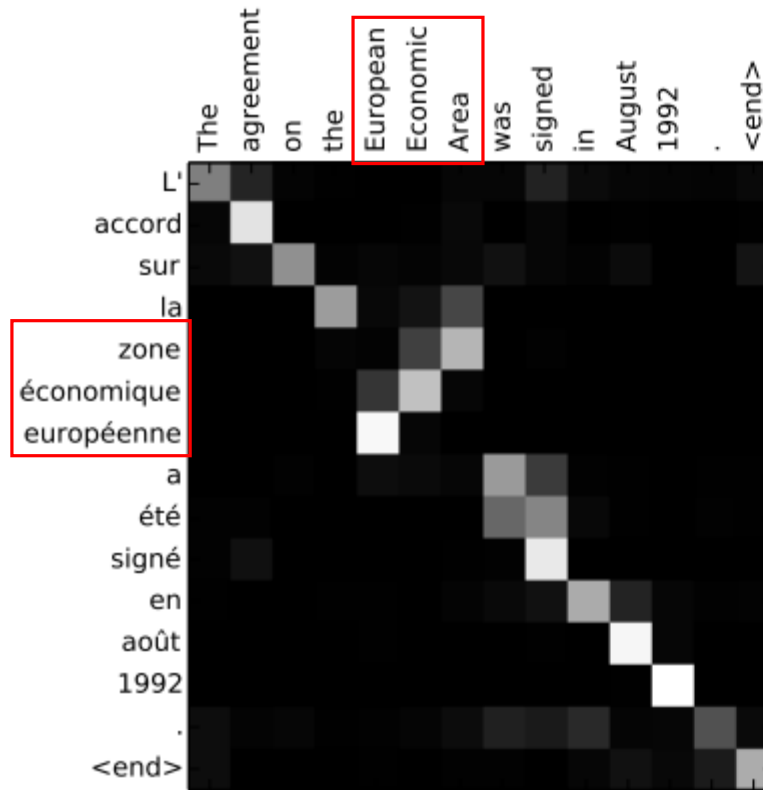


Bahdanau, D., Cho, K. and Bengio, Y. Neural machine translation by jointly learning to align and translate. ICLR 2015.

机器翻译注意力分数的可视化

English-to-French Translation

注意力机制可以将不同位置的、
但有对应关系的单词对齐。



注意力机制

- RNNs中的注意力机制
 - ✓ 计算机视觉中的应用： 图片描述
 - ✓ 自然语言处理中的应用： 机器翻译
- 一般的注意力层结构 (general attention layer)
 - ✓ 自注意力 (self-attention)
 - ✓ 位置编码 (positional encoding)
 - ✓ 遮挡的注意力 (masked attention)
 - ✓ 多头注意力 (multi-head attention)
- Transformers
 - ✓ 完全基于注意力机制的全新的神经网络结构 (相对RNNs和CNNs)

图像描述中的注意力计算

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

图像特征

h

Input:

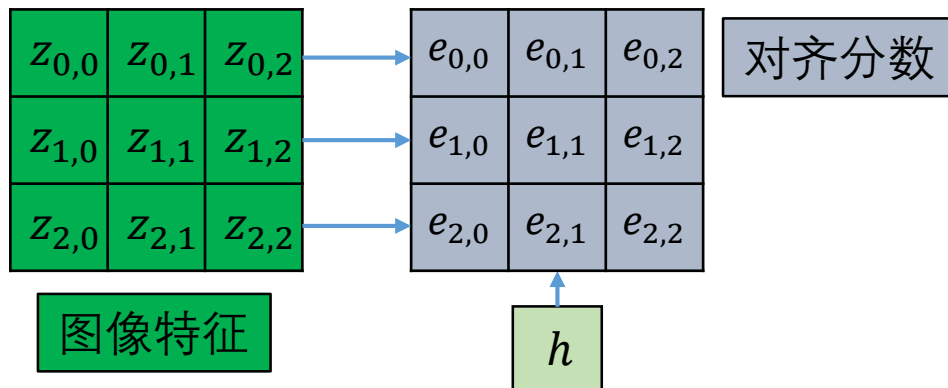
图像特征: $\mathbf{z} \in \mathbb{R}^{H \times W \times D}$

隐向量 (Query) : $\mathbf{h} \in \mathbb{R}^D$

图像描述中的注意力计算

Computations:

对齐: $e_{i,j} = f_{att}(h, z_{i,j})$

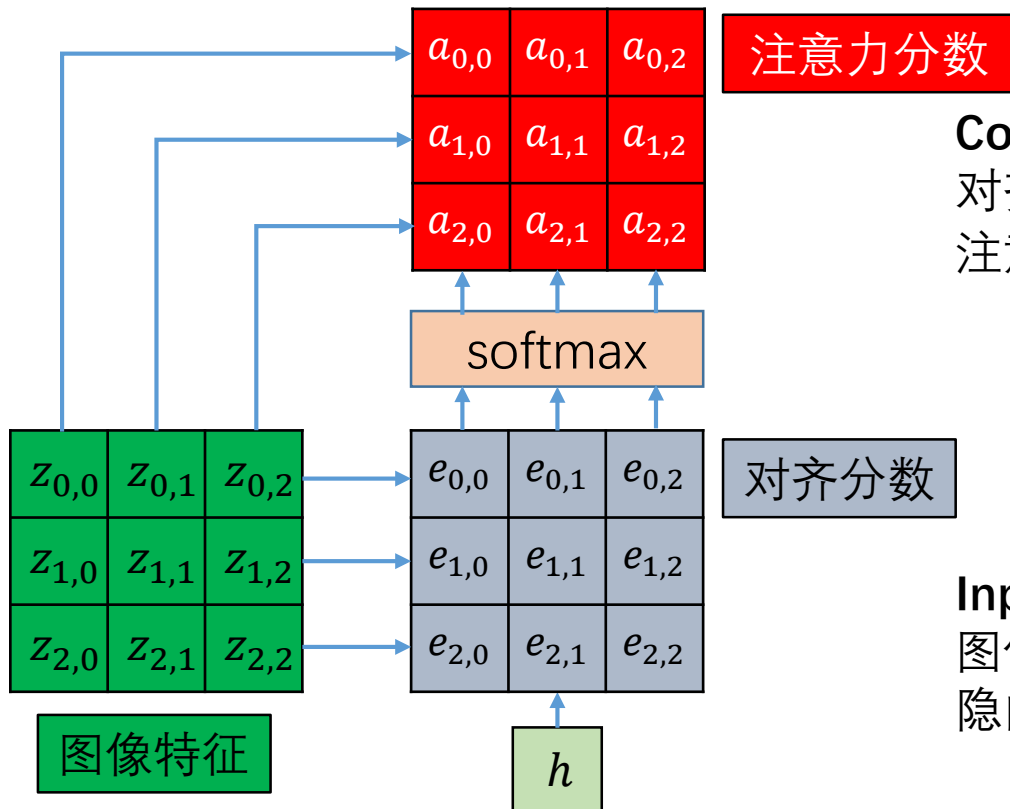


Input:

图像特征: $\mathbf{z} \in \mathbb{R}^{H \times W \times D}$

隐向量 (Query) : $\mathbf{h} \in \mathbb{R}^D$

图像描述中的注意力计算



Computations:

对齐: $e_{i,j} = f_{att}(h, z_{i,j})$

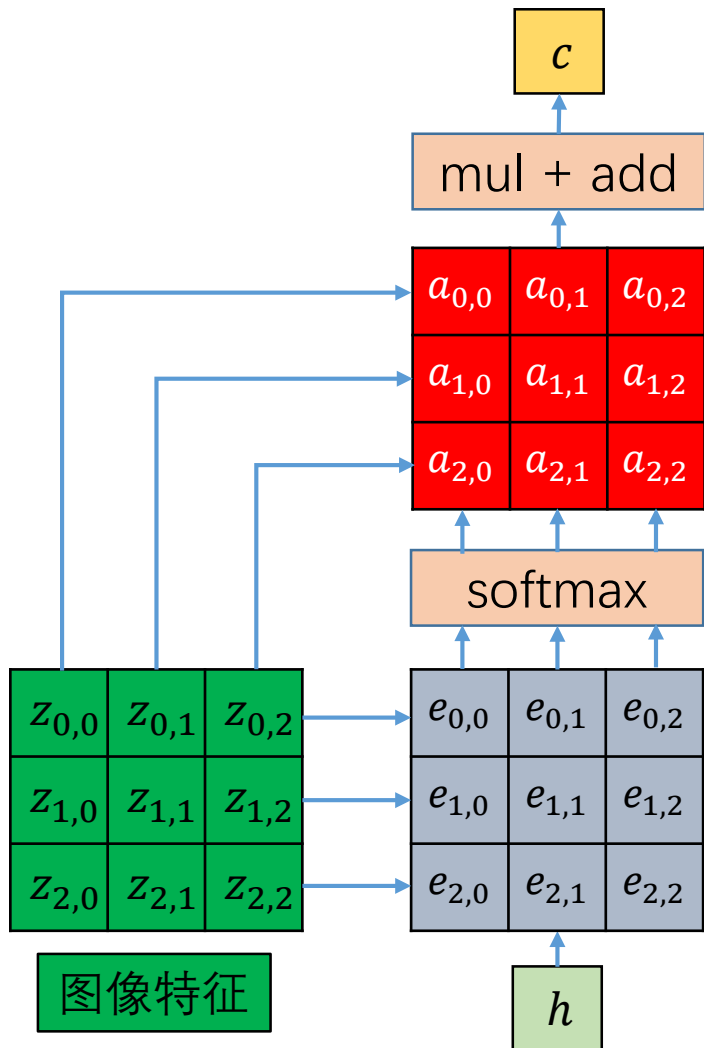
注意力: $a_{i,j} = softmax(e_{i,j})$

Input:

图像特征: $\mathbf{z} \in \mathbb{R}^{H \times W \times D}$

隐向量 (Query): $\mathbf{h} \in \mathbb{R}^D$

图像描述中的注意力计算



Output:

context vector: $\mathbf{c} \in \mathbb{R}^D$

Computations:

对齐: $e_{i,j} = f_{att}(\mathbf{h}, \mathbf{z}_{i,j})$

注意力: $a_{i,j} = \text{softmax}(e_{i,j})$

context: $\mathbf{c} = \sum_{i,j} a_{i,j} \mathbf{z}_{i,j}$

Input:

图像特征: $\mathbf{z} \in \mathbb{R}^{H \times W \times D}$

隐向量 (Query) : $\mathbf{h} \in \mathbb{R}^D$

一般注意力层

输入向量

x_0
 x_1
 x_2

h

Input:

N 个输入向量: $\mathbf{x} \in \mathbb{R}^{N \times D}$

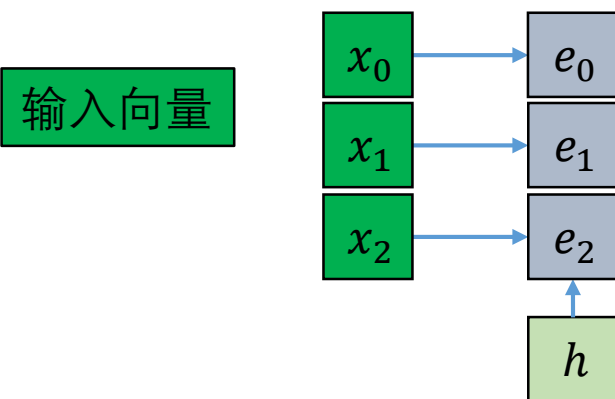
Query: $\mathbf{h} \in \mathbb{R}^D$

- ✓ 将 $H \times W$ 个长度为 D 的区域，拉伸成 N 个长度为 D 的向量 ($H \times W = N$)
- ✓ 向量的排列顺序不影响最终结果 (排列不变性)

一般注意力层

Computations:

对齐: $e_i = f_{att}(h, x_i)$



Input:

N 个输入向量: $\mathbf{x} \in \mathbb{R}^{N \times D}$

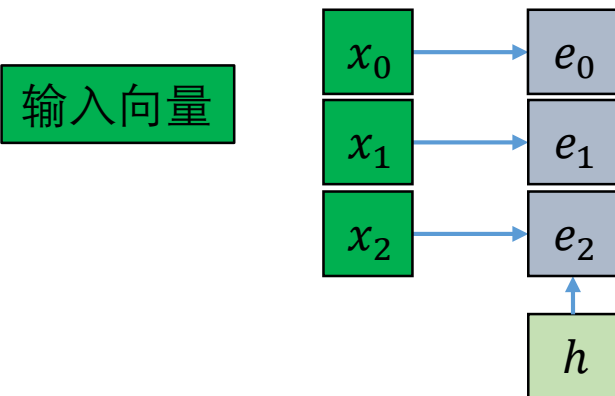
Query: $\mathbf{h} \in \mathbb{R}^D$

一般注意力层

Computations:

对齐: $e_i = h \cdot x_i$

- ✓ 将MLP ($f_{att}()$) 转化为两个向量点乘, 简化运算
- ✓ 在Transformer中取得很好的效果



Input:

N 个输入向量: $\mathbf{x} \in \mathbb{R}^{N \times D}$

Query: $\mathbf{h} \in \mathbb{R}^D$

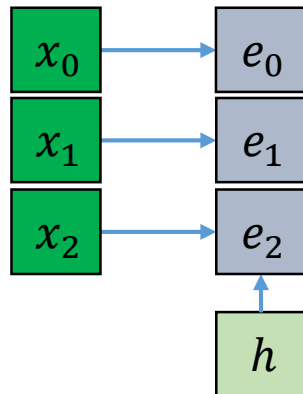
一般注意力层

Computations:

对齐: $e_i = h \cdot x_i / \sqrt{D}$

- ✓ $h \cdot x_i$ 的点乘运算，在向量长度很长的情况下，会造成对齐分数的方差很大（大的分数很大，小的分数很小）
- ✓ 在softmax运算之后，使得注意力分数只集中在少数输入向量上
- ✓ 因此，除以 \sqrt{D} 减轻这种效应， D 为向量长度。

输入向量



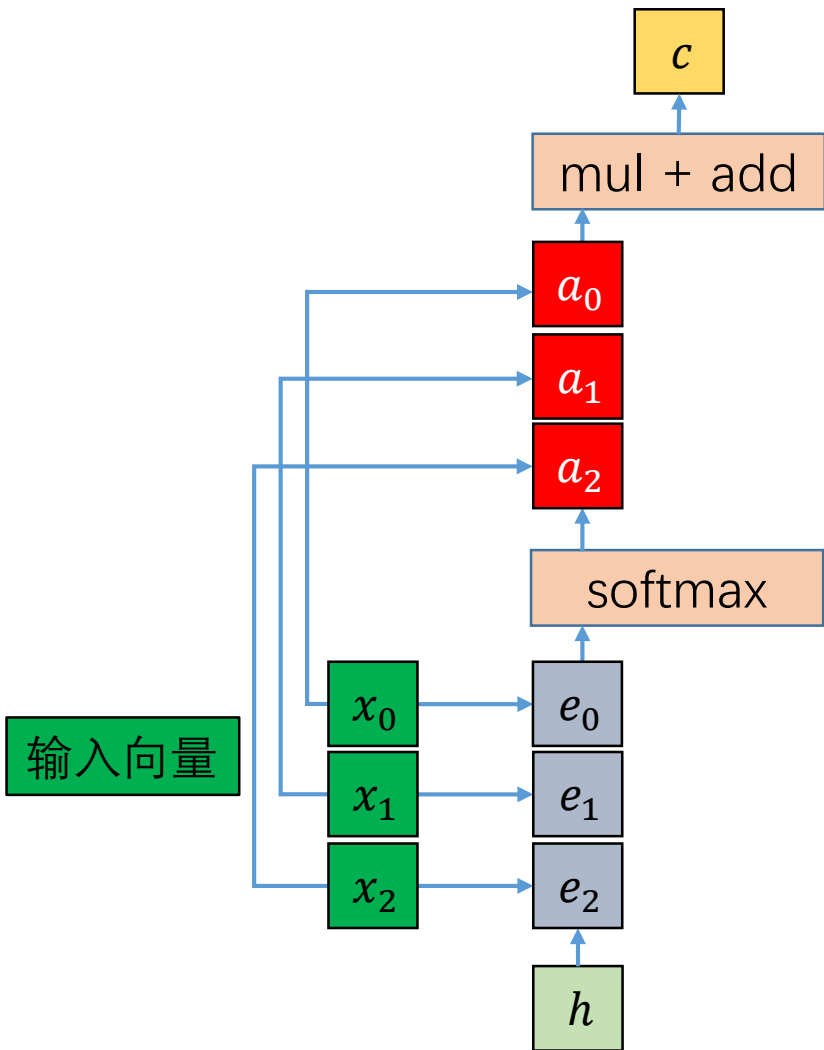
对齐分数

Input:

N 个输入向量: $\mathbf{x} \in \mathbb{R}^{N \times D}$

Query: $\mathbf{h} \in \mathbb{R}^D$

一般注意力层



Output:

context vector: $\mathbf{c} \in \mathbb{R}^D$

注意力分数

Computations:

对齐: $e_i = \mathbf{h} \cdot \mathbf{x}_i / \sqrt{D}$

注意力: $a_i = \text{softmax}(e_i)$

context: $\mathbf{c} = \sum_i a_i \mathbf{x}_i$

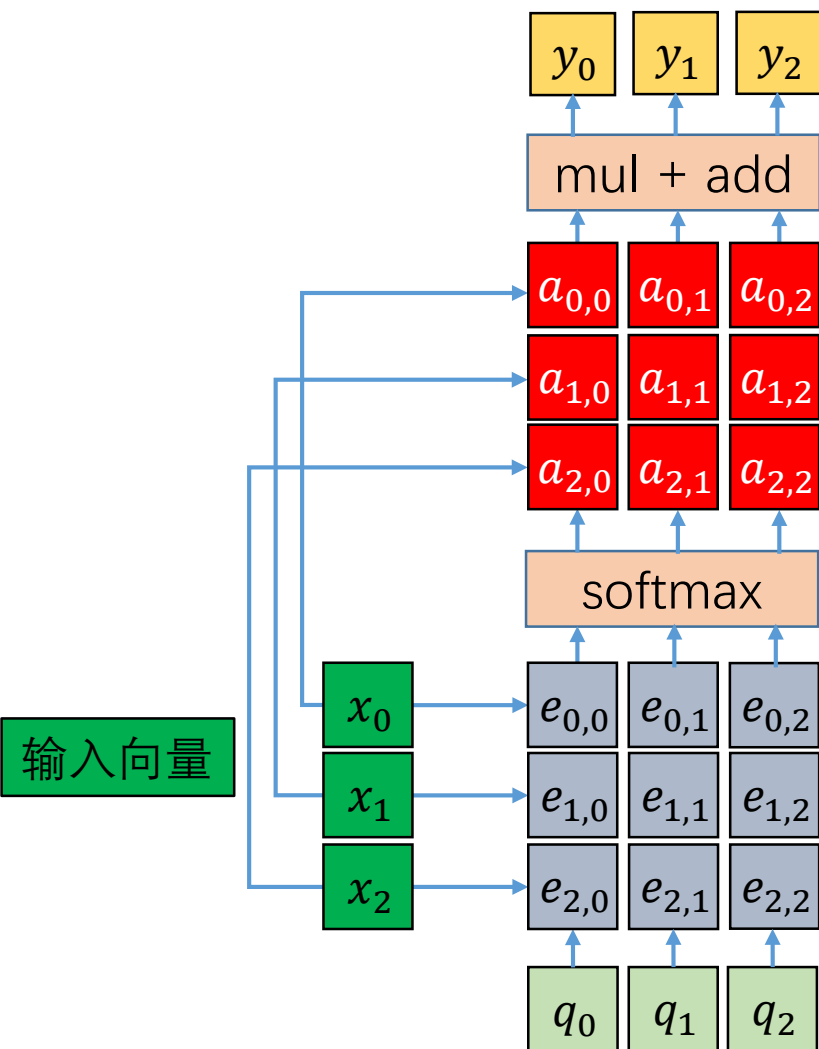
对齐分数

Input:

N 个输入向量: $\mathbf{x} \in \mathbb{R}^{N \times D}$

Query: $\mathbf{h} \in \mathbb{R}^D$

一般注意力层



Output:

context vectors: $\mathbf{y} \in \mathbb{R}^{M \times D}$ (每个query生成一个context vector)

Computations:

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

注意力: $a_{i,j} = \text{softmax}(e_{i,j})$

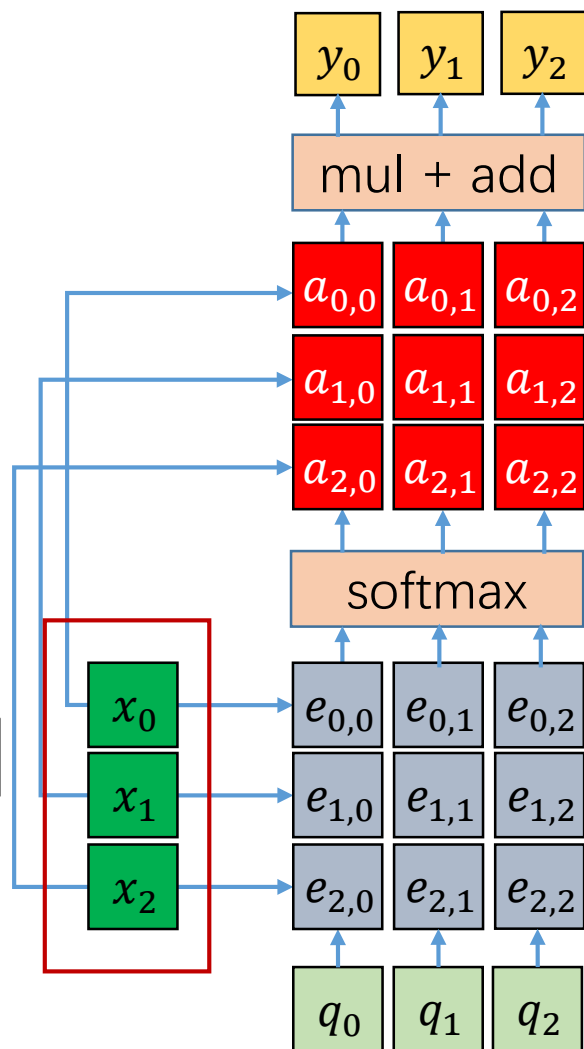
context: $y_j = \sum_i a_{i,j} x_i$

Input:

N 个输入向量: $\mathbf{x} \in \mathbb{R}^{N \times D}$

Queries: $\mathbf{q} \in \mathbb{R}^{M \times D}$ (M 个长度为 D 的query)

一般注意力层



Output:

context vectors: $\mathbf{y} \in \mathbb{R}^{M \times D}$ (每个query生成一个context vector)

注意力分数

Computations:

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

注意力: $a_{i,j} = \text{softmax}(e_{i,j})$

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

对齐分数

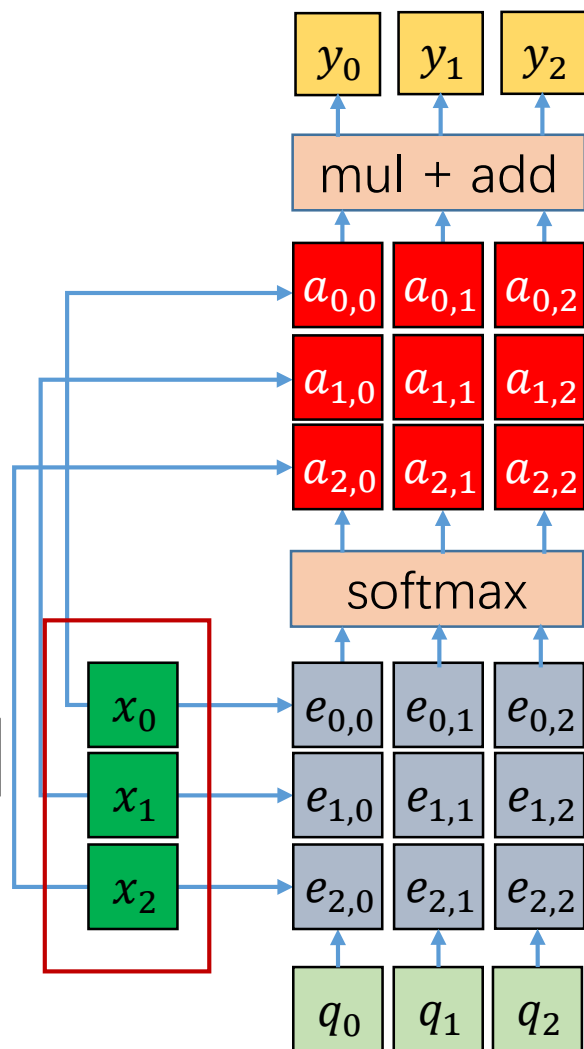
Input:

N 个输入向量: $\mathbf{x} \in \mathbb{R}^{N \times D}$

Queries: $\mathbf{q} \in \mathbb{R}^{M \times D}$ (M 个长度为 D 的query)

✓ 缺点: 输入向量既作为对齐运算的输入, 又作为注意力运算的输入, 缺乏输入特征上的变化

一般注意力层



Output:

context vectors: $\mathbf{y} \in \mathbb{R}^{M \times D}$ (每个query生成一个context vector)

注意力分数

Computations:

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

注意力: $a_{i,j} = \text{softmax}(e_{i,j})$

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

对齐分数

Input:

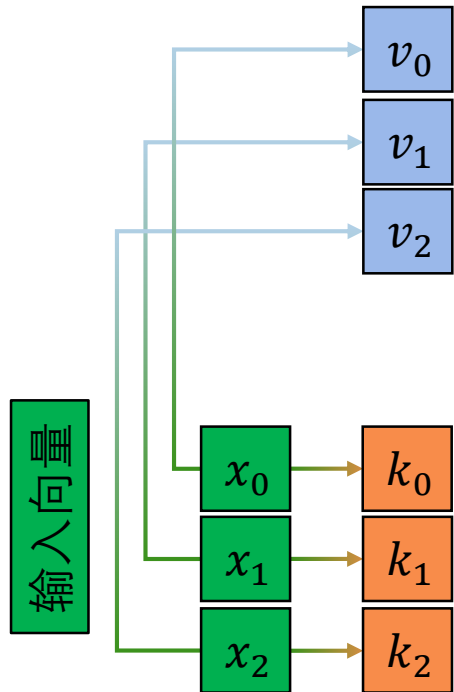
N 个输入向量: $\mathbf{x} \in \mathbb{R}^{N \times D}$

Queries: $\mathbf{q} \in \mathbb{R}^{M \times D}$ (M 个长度为 D 的query)

- ✓ 缺点: 输入向量既作为对齐运算的输入, 又作为注意力运算的输入, 缺乏输入特征上的变化
- ✓ 解决方法: 在计算对齐分数和注意力分数时, 分别用一个线性层对输入向量进行转化

输入向量

一般注意力层



Computations:

Key向量: $\mathbf{k} = \mathbf{x}W_k$

Value向量: $\mathbf{v} = \mathbf{x}W_v$

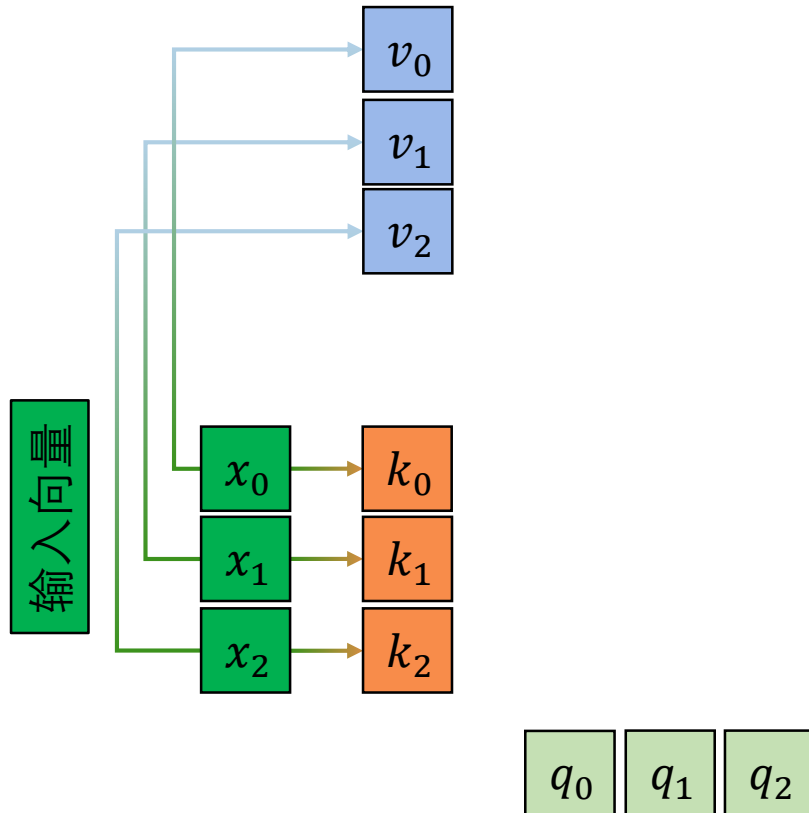
- ✓ 缺点: 输入向量即作为对齐运算的输入, 又作为注意力运算的输入, 缺乏输入特征上的变化
- ✓ 解决方法: 在计算对齐分数和注意力分数时, 分别用一个线性层对输入向量进行转化

Input:

N 个输入向量: $\mathbf{x} \in \mathbb{R}^{N \times D}$

Queries: $\mathbf{q} \in \mathbb{R}^{M \times D}$ (M 个长度为 D 的query)

一般注意力层



Computations:

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

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

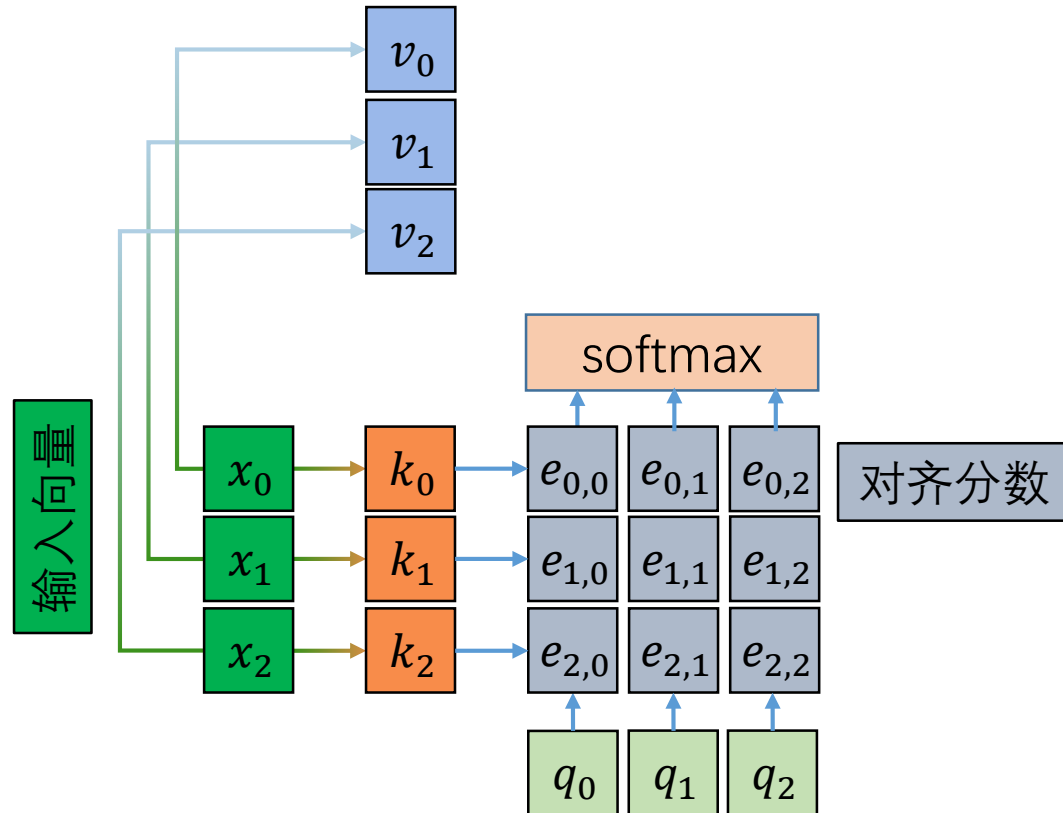
- ✓ 缺点: 输入向量即作为对齐运算的输入, 又作为注意力运算的输入, 缺乏输入特征上的变化
- ✓ 解决方法: 在计算对齐分数和注意力分数时, 分别用一个线性层对输入向量进行转化

Input:

N 个输入向量: $\mathbf{x} \in \mathbb{R}^{N \times D}$

Queries: $\mathbf{q} \in \mathbb{R}^{M \times D_k}$ (M 个长度为 D_k 的query)

一般注意力层



Computations:

Key向量: $k = xW_k$

Value向量: $v = xW_v$

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

Input:

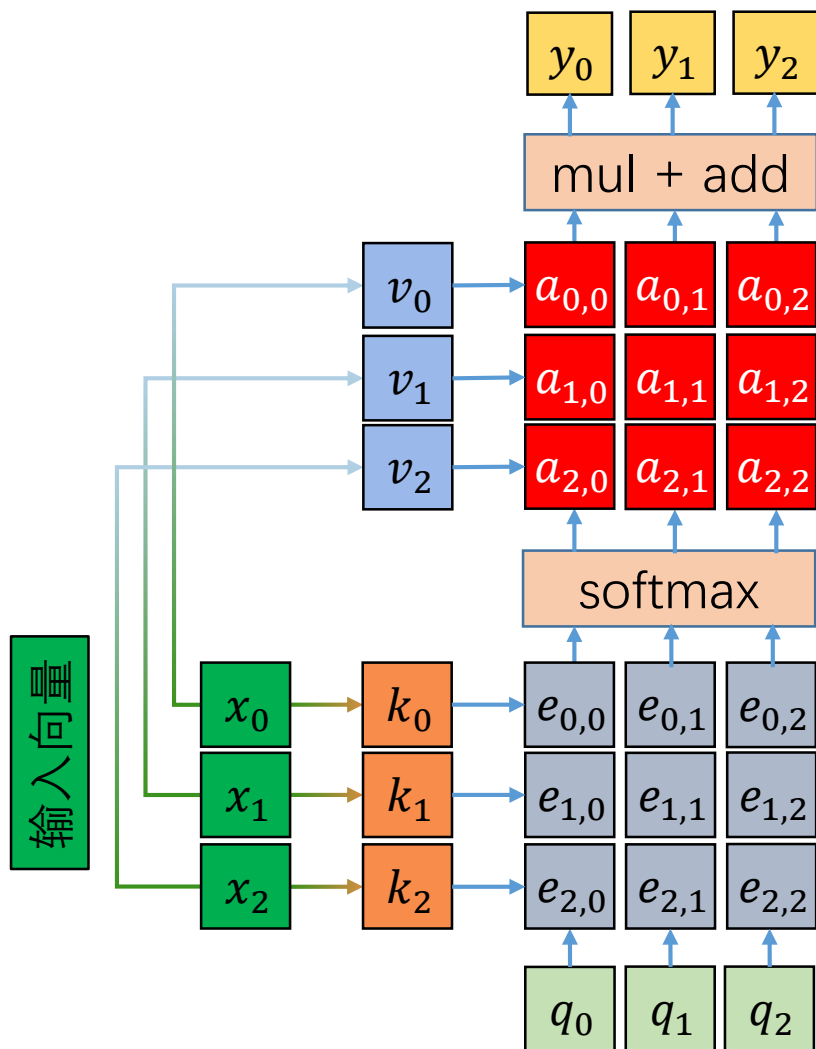
N 个输入向量: $x \in \mathbb{R}^{N \times D}$

Queries: $q \in \mathbb{R}^{M \times D_k}$ (M 个长度为 D_k 的query)

- ✓ 缺点: 输入向量即作为对齐运算的输入, 又作为注意力运算的输入, 缺乏输入特征上的变化
- ✓ 解决方法: 在计算对齐分数和注意力分数时, 分别用一个线性层对输入向量进行转化

一般注意力层

✓ 输出向量和输入向量的长度可以不同（通过value线性层发生改变）



Output:

context vectors: $\mathbf{y} \in \mathbb{R}^{M \times D_v}$ (每个query生成一个context vector)

Computations:

Key向量: $\mathbf{k} = \mathbf{x}W_k$

Value向量: $\mathbf{v} = \mathbf{x}W_v$

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

注意力: $a_{i,j} = \text{softmax}(e_{i,j})$

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

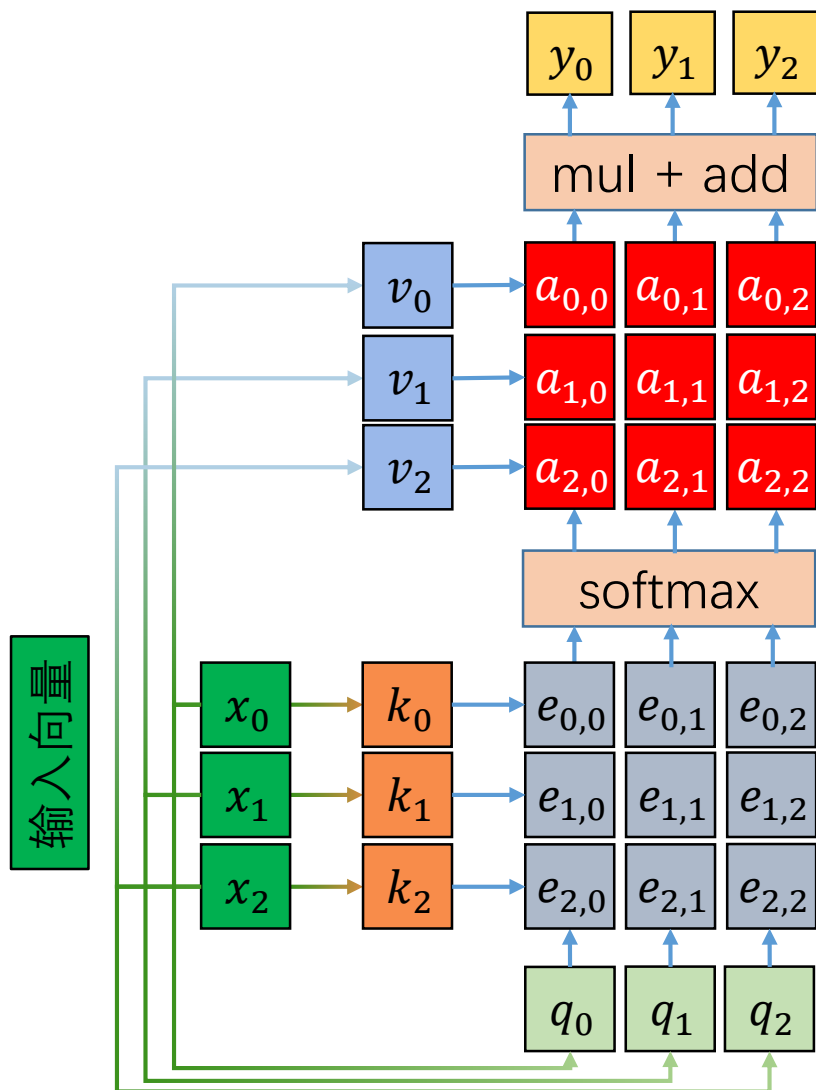
Input:

N 个输入向量: $\mathbf{x} \in \mathbb{R}^{N \times D}$

Queries: $\mathbf{q} \in \mathbb{R}^{M \times D_k}$ (M 个长度为 D_k 的query)

- ✓ 缺点: 输入向量即作为对齐运算的输入, 又作为注意力运算的输入, 缺乏输入特征上的变化
- ✓ 解决方法: 在计算对齐分数和注意力分数时, 分别用一个线性层对输入向量进行转化

一般注意力层



注意力分数

对齐分数

Output:

context vectors: $\mathbf{y} \in \mathbb{R}^{M \times D_v}$ (每个query生成一个context vector)

Computations:

Key向量: $\mathbf{k} = \mathbf{x}W_k$

Value向量: $\mathbf{v} = \mathbf{x}W_v$

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

注意力: $a_{i,j} = \text{softmax}(e_{i,j})$

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

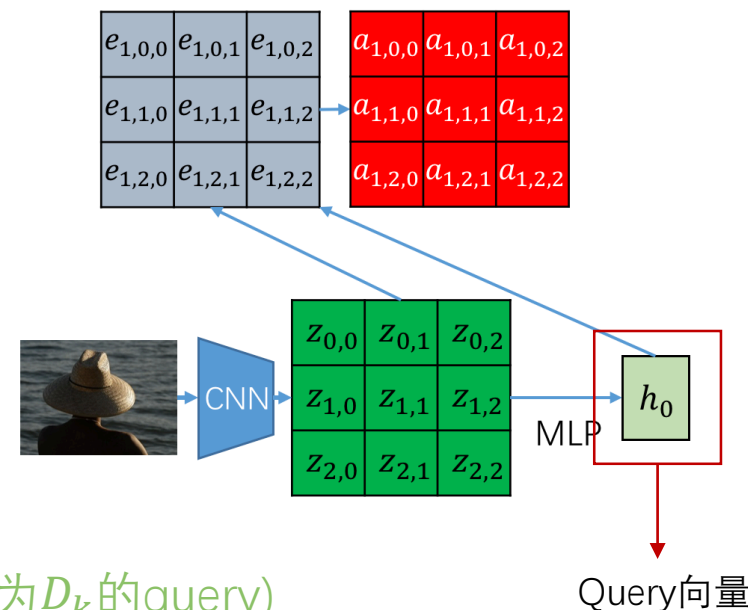
Input:

N 个输入向量: $\mathbf{x} \in \mathbb{R}^{N \times D}$

Queries: $\mathbf{q} \in \mathbb{R}^{M \times D_k}$ (M 个长度为 D_k 的query)

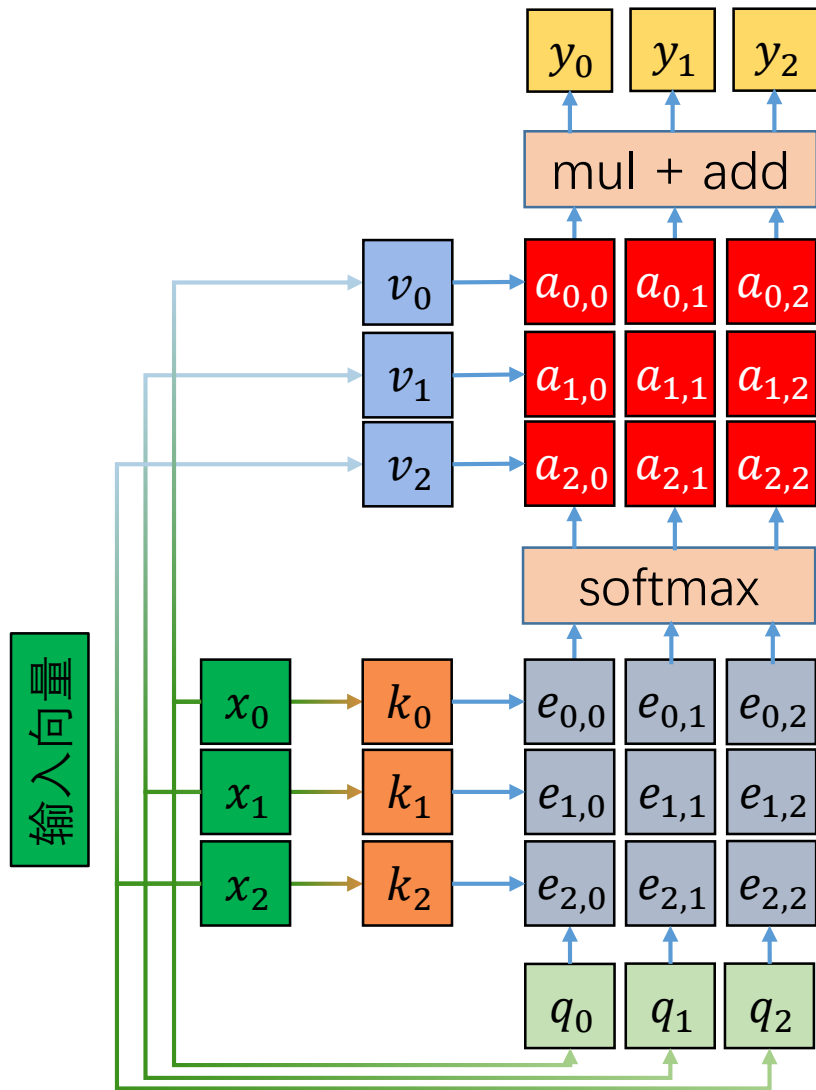
✓ 自注意力 (self attention)

✓ Query向量也可以由输入向量转化而来



Vaswani, A., Shazeer, N., et al. Attention is all you need. NIPS 2017.

自注意力层



Output:

context vectors: $\mathbf{y} \in \mathbb{R}^{M \times D_v}$ (每个query生成一个context vector)

注意力分数

Computations:

Key向量: $\mathbf{k} = \mathbf{x}W_k$

Value向量: $\mathbf{v} = \mathbf{x}W_v$

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

注意力: $a_{i,j} = \text{softmax}(e_{i,j})$

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

对齐分数

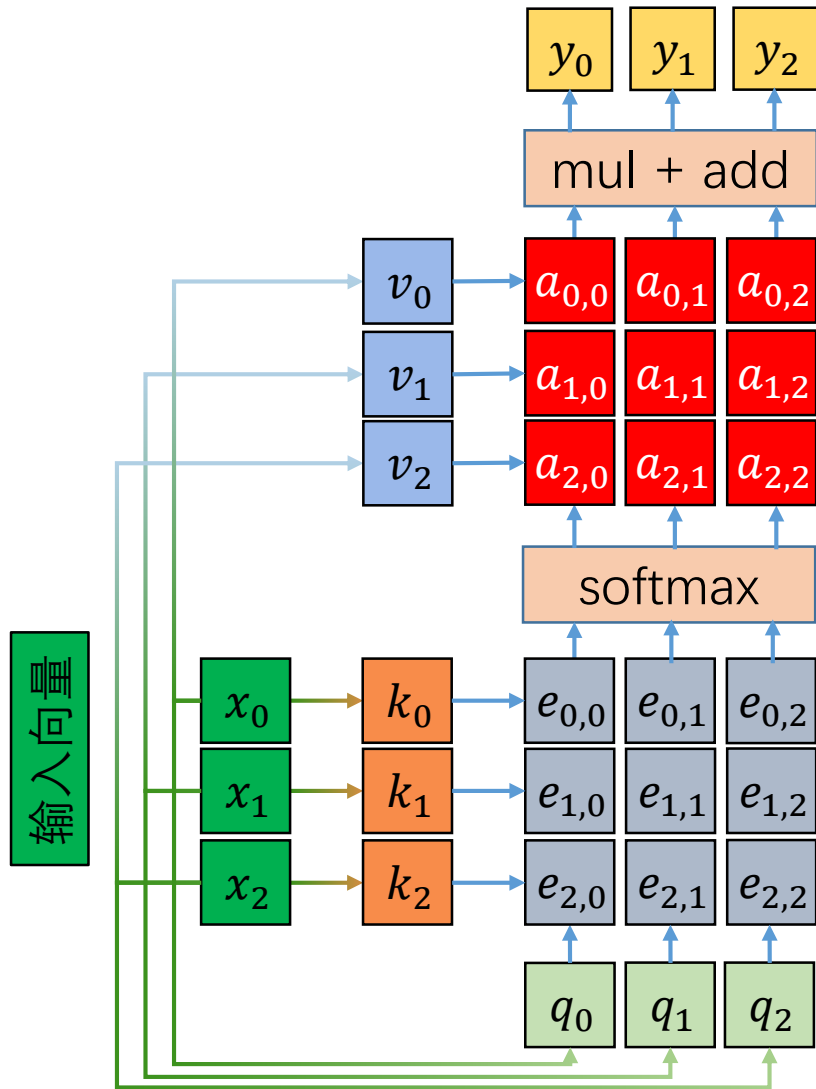
由输入向量经线性层得到

Input:

N 个输入向量: $\mathbf{x} \in \mathbb{R}^{N \times D}$

Queries: $\mathbf{q} \in \mathbb{R}^{M \times D_k}$ (M 个长度为 D_k 的query)

自注意力层



Output:

context vectors: $\mathbf{y} \in \mathbb{R}^{M \times D_v}$ (每个query生成一个context vector)

注意力分数

Computations:

Key向量: $\mathbf{k} = \mathbf{x}W_k$

Value向量: $\mathbf{v} = \mathbf{x}W_v$

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

注意力: $a_{i,j} = \text{softmax}(e_{i,j})$

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

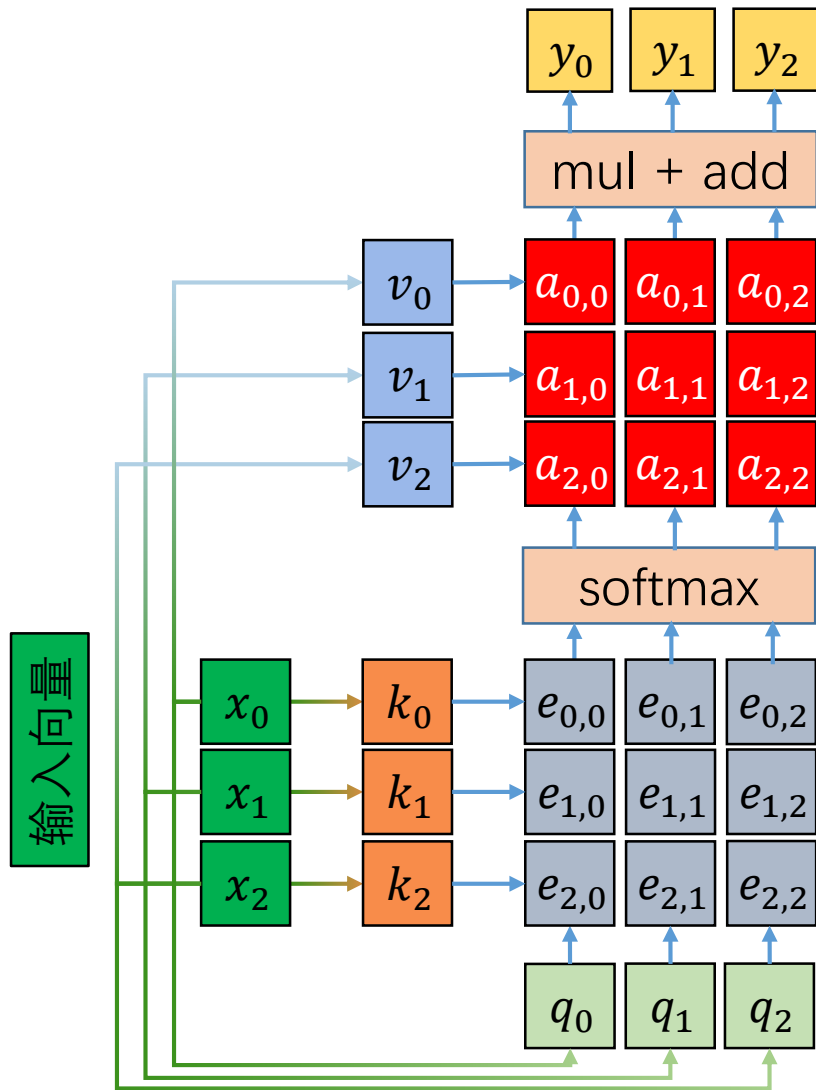
对齐分数

Input:

N 个输入向量: $\mathbf{x} \in \mathbb{R}^{N \times D}$

Queries: $\mathbf{q} \in \mathbb{R}^{M \times D_k}$ (M 个长度为 D_k 的query)

自注意力层



注意力分数

对齐分数

Output:

context vectors: $\mathbf{y} \in \mathbb{R}^{M \times D_v}$

Computations:

Key向量: $\mathbf{k} = \mathbf{x}W_k$

Value向量: $\mathbf{v} = \mathbf{x}W_v$

Query向量: $\mathbf{q} = \mathbf{x}W_q$

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

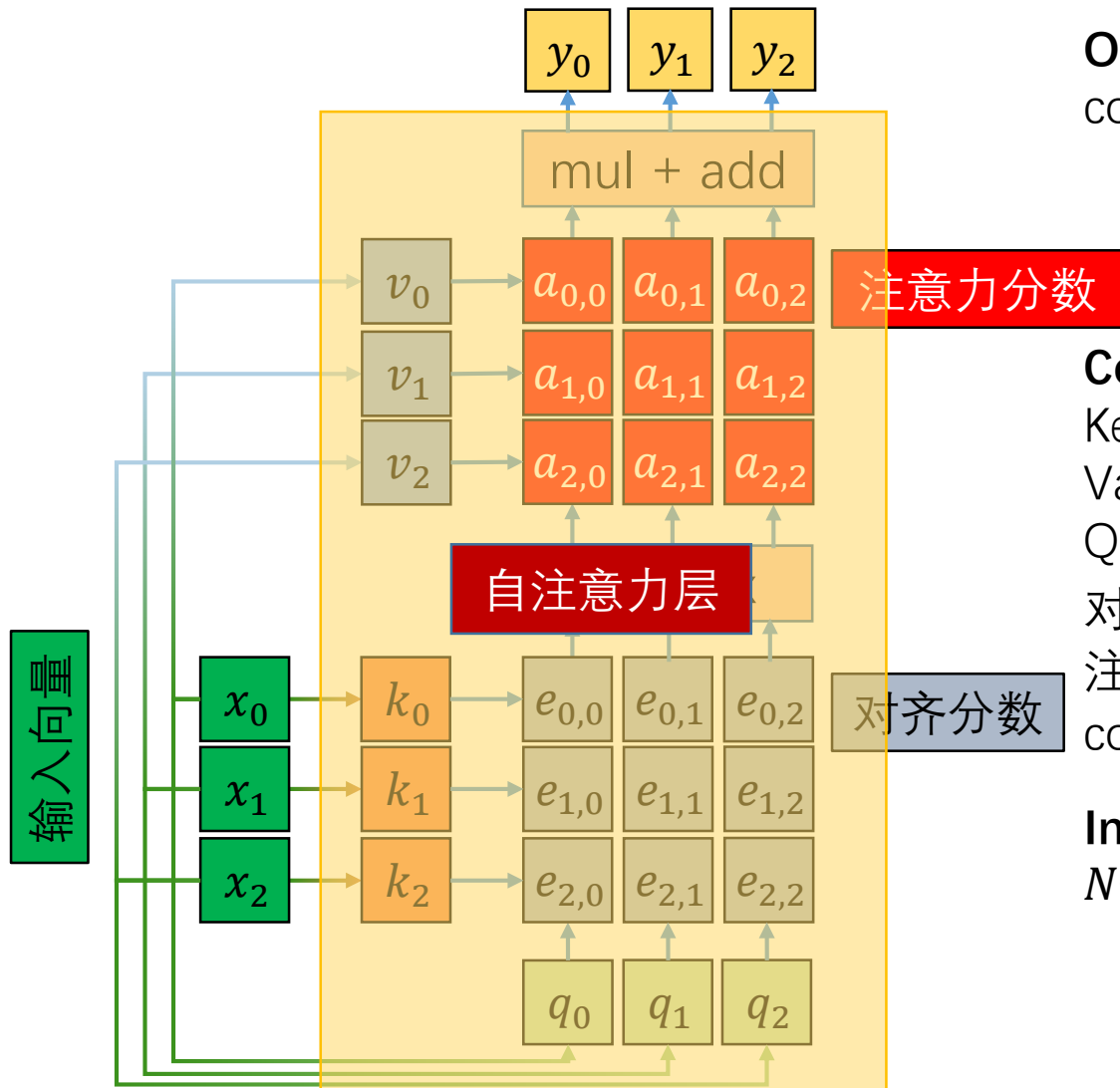
注意力: $a_{i,j} = \text{softmax}(e_{i,j})$

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

Input:

N 个输入向量: $\mathbf{x} \in \mathbb{R}^{N \times D}$

自注意力层：一组输入向量的注意力运算



Output:

context vectors: $\mathbf{y} \in \mathbb{R}^{M \times D_v}$

Computations:

Key向量: $\mathbf{k} = \mathbf{x}W_k$

Value向量: $\mathbf{v} = \mathbf{x}W_v$

Query向量: $\mathbf{q} = \mathbf{x}W_q$

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

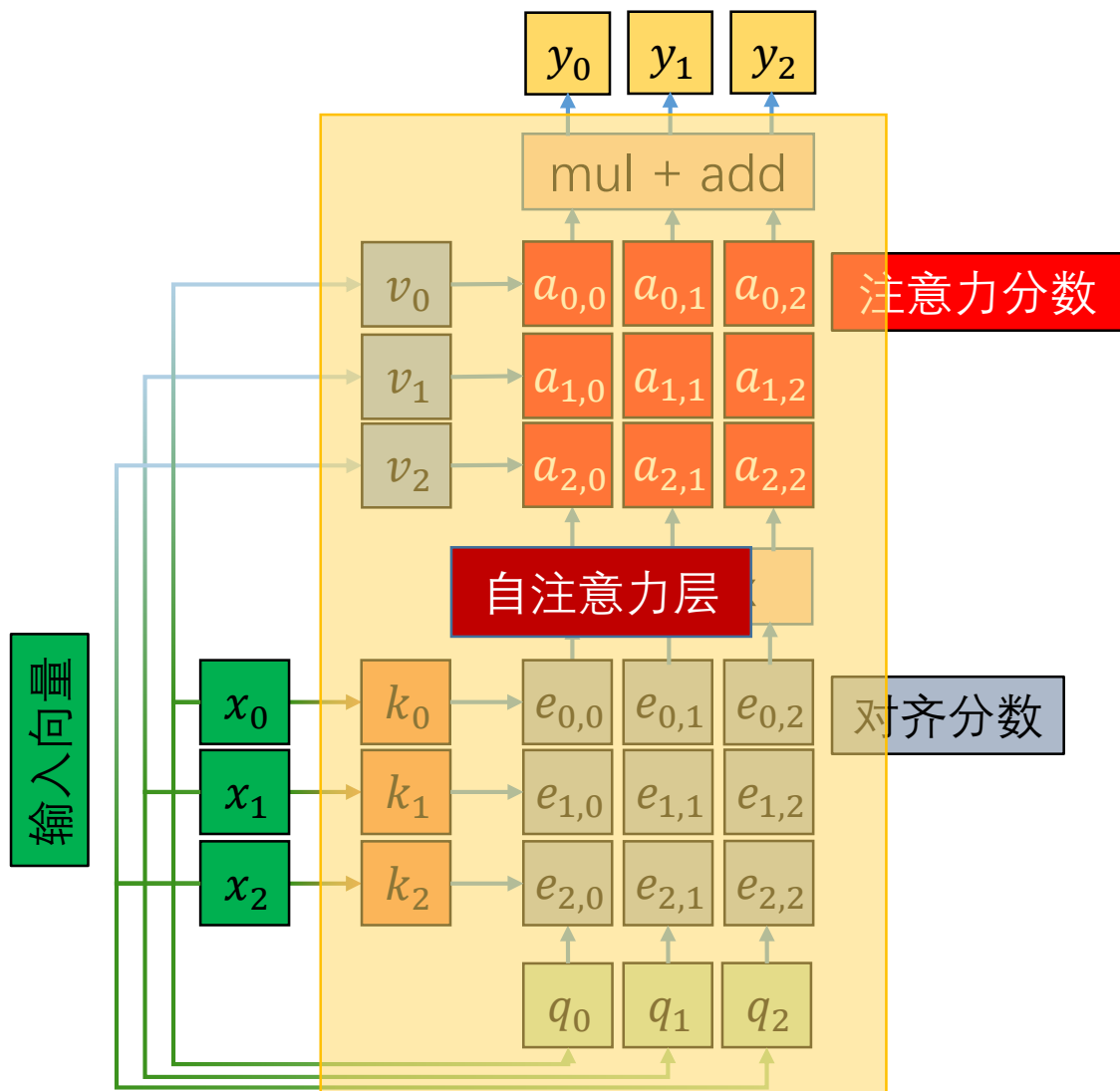
注意力: $a_{i,j} = \text{softmax}(e_{i,j})$

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

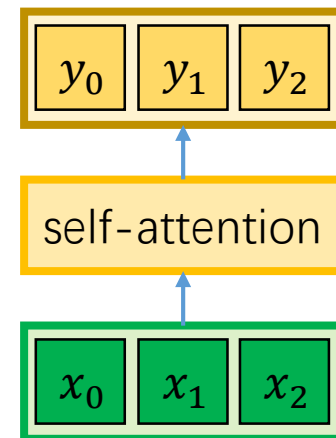
Input:

N 个输入向量: $\mathbf{x} \in \mathbb{R}^{N \times D}$

自注意力层：一组输入向量的注意力运算



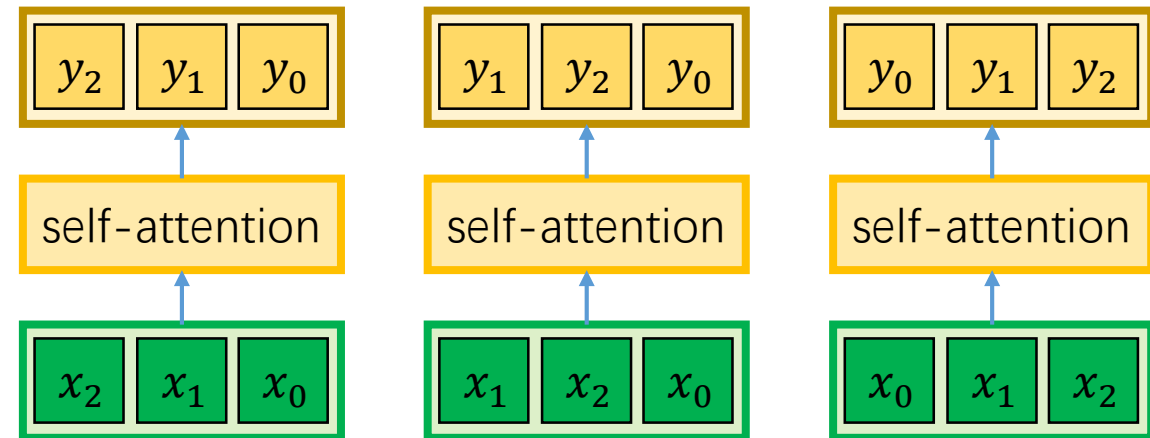
简化



Vaswani, A., Shazeer, N., et al. Attention is all you need. NIPS 2017.

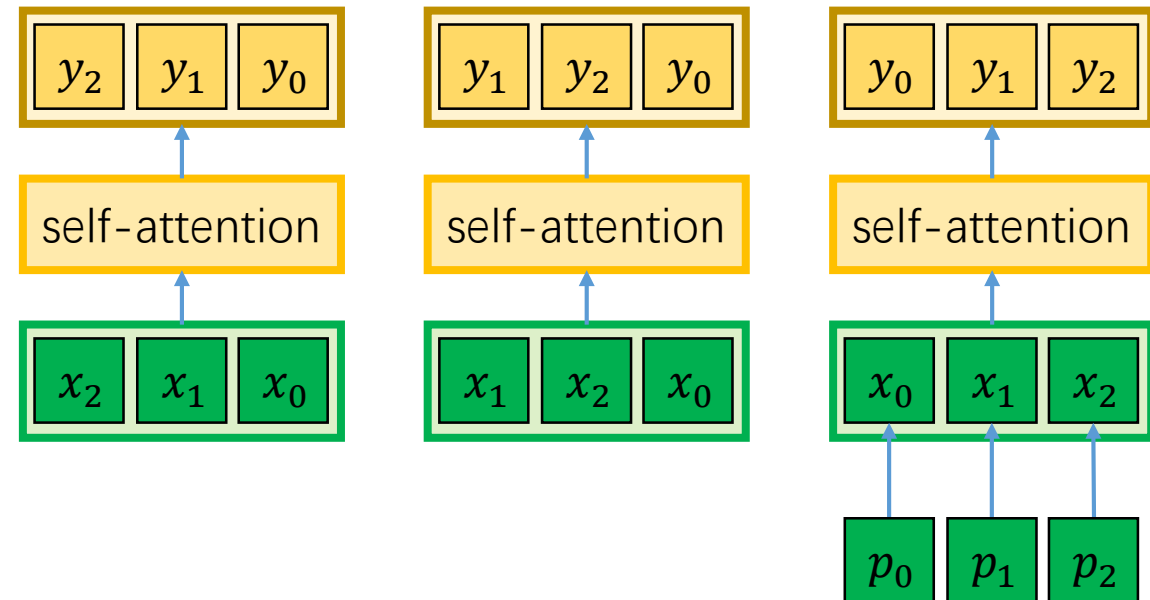
排列不变性 (Permutation invariant)

✓ 生成的编码 y_i 无法学到输入向量的排列顺序信息

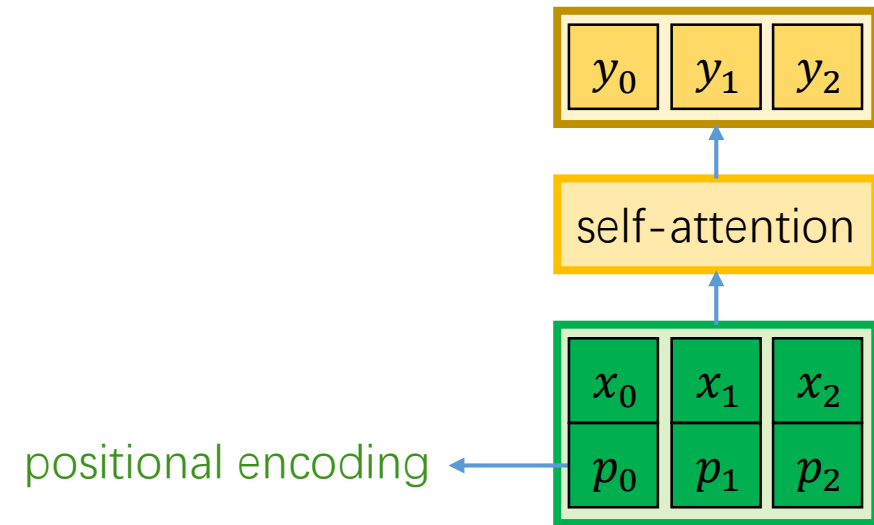


排列不变性 (Permutation invariant)

- ✓ 生成的编码 y_i 无法学到输入向量的排列顺序信息
- ✓ 解决方案：给输入向量加入在序列中的位置信息（能够表示每个向量的绝对位置或相对位置）



位置编码 (positional encoding)



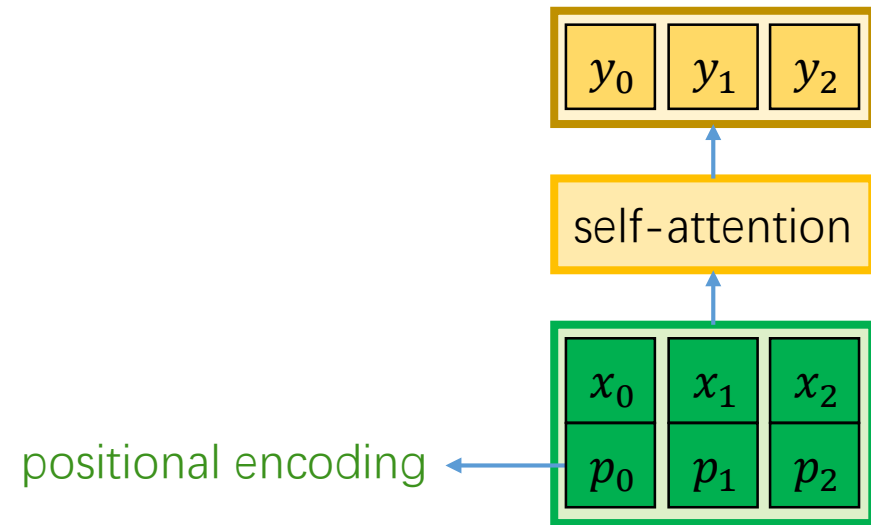
将每个位置编码 p_t 拼接或者
加到相应的输入向量 x_t

位置编码 (positional encoding)

位置编码 p_t

需要满足以下特性：

- ✓ 唯一性：每个时刻的编码唯一
- ✓ 一致性：对于所有输入长度，两个时刻之间的距离是一致的
- ✓ 泛化性：对于所有输入长度都可以使用这套编码



将每个位置编码 p_t 拼接或者
加到相应的输入向量 x_t

位置编码 (positional encoding)

位置编码 p_t

需要满足以下特性:

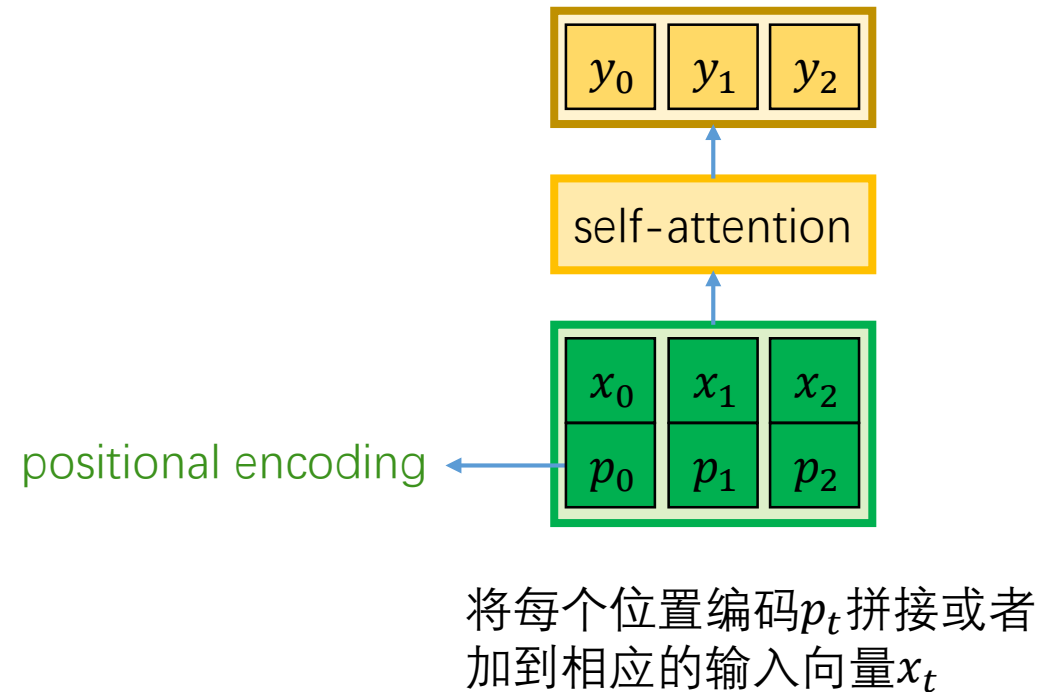
- ✓ 唯一性: 每个时刻的编码唯一
- ✓ 一致性: 对于所有输入长度, 两个时刻之间的距离是一致的
- ✓ 泛化性: 对于所有输入长度都可以使用这套编码

$$p_t^i = \begin{cases} \sin(\omega_k \cdot t), & \text{if } i = 2k \\ \cos(\omega_k \cdot t), & \text{if } i = 2k + 1 \end{cases}$$

$t \geq 0$ 代表时刻

$0 \leq i < D$ 代表编码的第 i 个位置

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



位置编码 (positional encoding)

位置编码 p_t

需要满足以下特性:

- ✓ 唯一性: 每个时刻的编码唯一
- ✓ 一致性: 对于所有输入长度, 两个时刻之间的距离是一致的
- ✓ 泛化性: 对于所有输入长度都可以使用这套编码

$$p_t^i = \begin{cases} \sin(\omega_k \cdot t), & \text{if } i = 2k \\ \cos(\omega_k \cdot t), & \text{if } i = 2k + 1 \end{cases}$$

$t \geq 0$ 代表时刻

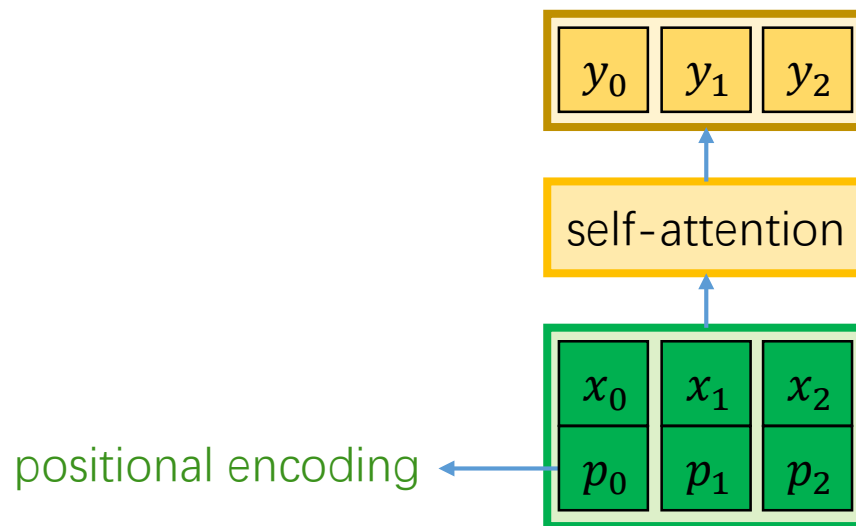
$0 \leq i < D$ 代表编码的第 i 个位置

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



$$p_t = \begin{bmatrix} \sin(\omega_1 \cdot t) \\ \cos(\omega_1 \cdot t) \\ \sin(\omega_2 \cdot t) \\ \cos(\omega_2 \cdot t) \\ \dots \\ \sin(\omega_{D/2} \cdot t) \\ \cos(\omega_{D/2} \cdot t) \end{bmatrix}$$

$D \times 1$



将每个位置编码 p_t 拼接或者加到相应的输入向量 x_t

位置编码 (positional encoding)

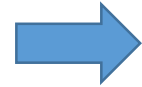
位置编码 p_t 的一致性

$$p_t^i = \begin{cases} \sin(\omega_k \cdot t), & \text{if } i = 2k \\ \cos(\omega_k \cdot t), & \text{if } i = 2k + 1 \end{cases}$$

$t \geq 0$ 代表时刻

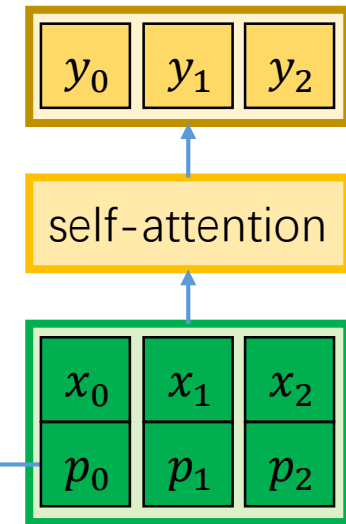
$0 \leq i < D$ 代表编码的第 i 个位置

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



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

positional encoding



将每个位置编码 p_t 拼接或者
加到相应的输入向量 x_t

位置编码 (positional encoding)

位置编码 p_t 的一致性

$$p_t^i = \begin{cases} \sin(\omega_k \cdot t), & \text{if } i = 2k \\ \cos(\omega_k \cdot t), & \text{if } i = 2k + 1 \end{cases}$$

$t \geq 0$ 代表时刻

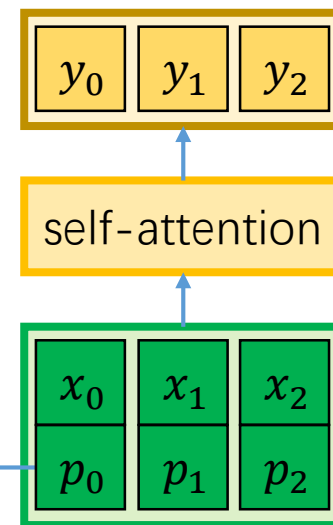
$0 \leq i < D$ 代表编码的第 i 个位置

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



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

positional encoding



将每个位置编码 p_t 拼接或者加到相应的输入向量 x_t

考虑任意一对sin和cos, 以下公式成立

$$\begin{bmatrix} \cos(\omega_k \cdot \Delta t) & \sin(\omega_k \cdot \Delta t) \\ -\sin(\omega_k \cdot \Delta t) & \cos(\omega_k \cdot \Delta t) \end{bmatrix} \cdot \begin{bmatrix} \sin(\omega_k \cdot t) \\ \cos(\omega_k \cdot t) \end{bmatrix} = \begin{bmatrix} \sin(\omega_k \cdot (t + \Delta t)) \\ \cos(\omega_k \cdot (t + \Delta t)) \end{bmatrix}$$

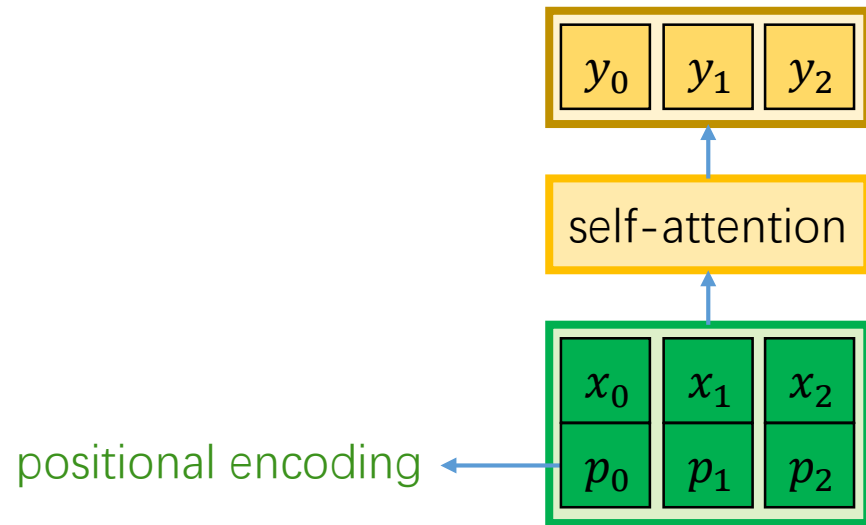
位置编码 (positional encoding)

位置编码 p_t 的一致性

$$p_t^i = \begin{cases} \sin(\omega_k \cdot t), & \text{if } i = 2k \\ \cos(\omega_k \cdot t), & \text{if } i = 2k + 1 \end{cases}$$

$t \geq 0$ 代表时刻
 $0 \leq i < D$ 代表编码的第 i 个位置
 $\omega_k = \frac{1}{10000^{2k/D}}$

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



将每个位置编码 p_t 拼接或者加到相应的输入向量 x_t

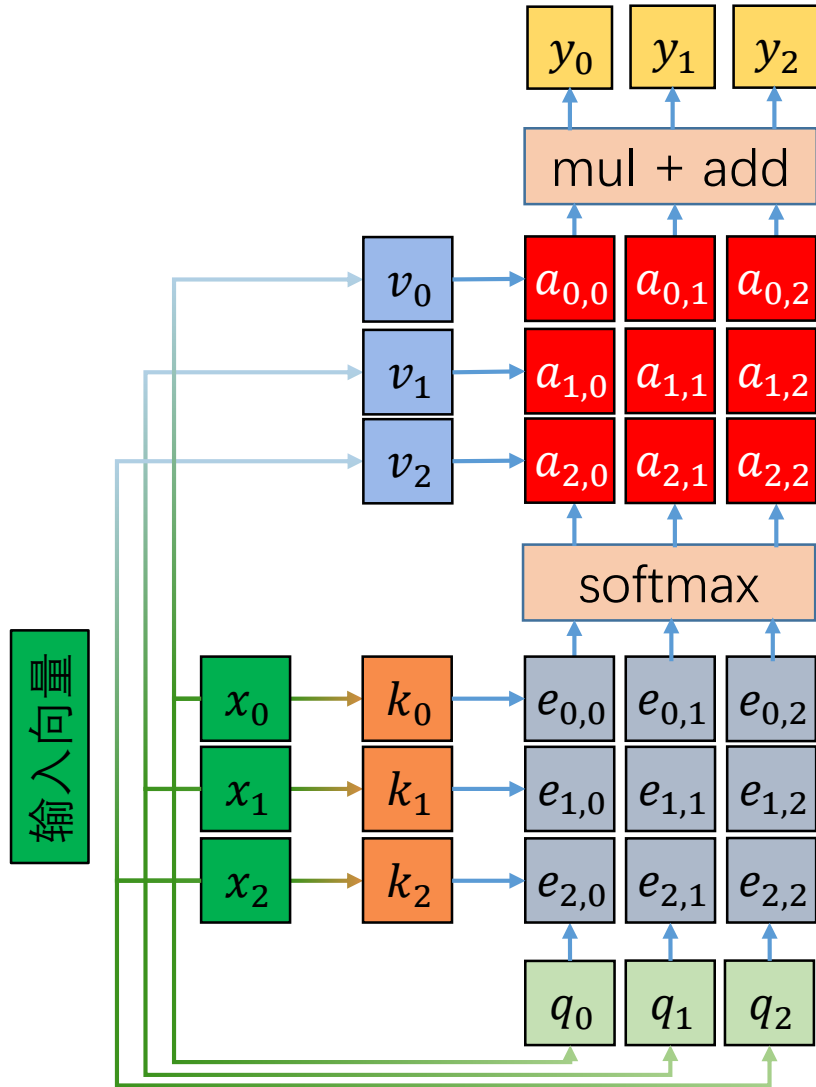
考虑任意一对sin和cos, 以下公式成立

$$\begin{bmatrix} \cos(\omega_k \cdot \Delta t) & \sin(\omega_k \cdot \Delta t) \\ -\sin(\omega_k \cdot \Delta t) & \cos(\omega_k \cdot \Delta t) \end{bmatrix} \cdot \begin{bmatrix} \sin(\omega_k \cdot t) \\ \cos(\omega_k \cdot t) \end{bmatrix} = \begin{bmatrix} \sin(\omega_k \cdot (t + \Delta t)) \\ \cos(\omega_k \cdot (t + \Delta t)) \end{bmatrix}$$

对于任意一组对sin和cos, p_t 和 $p_{t+\Delta t}$ 的编码表示转换是两个时刻距离 Δt 的线性方程 (即和输入长度无关!)

Vaswani, A., Shazeer, N., et al. Attention is all you need. NIPS 2017.

遮挡的自注意力层 (masked self-attention layer)



Output:

context vectors: $\mathbf{y} \in \mathbb{R}^{M \times D_v}$

注意力分数

Computations:

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

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

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

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

注意力: $a_{i,j} = \text{softmax}(e_{i,j})$

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

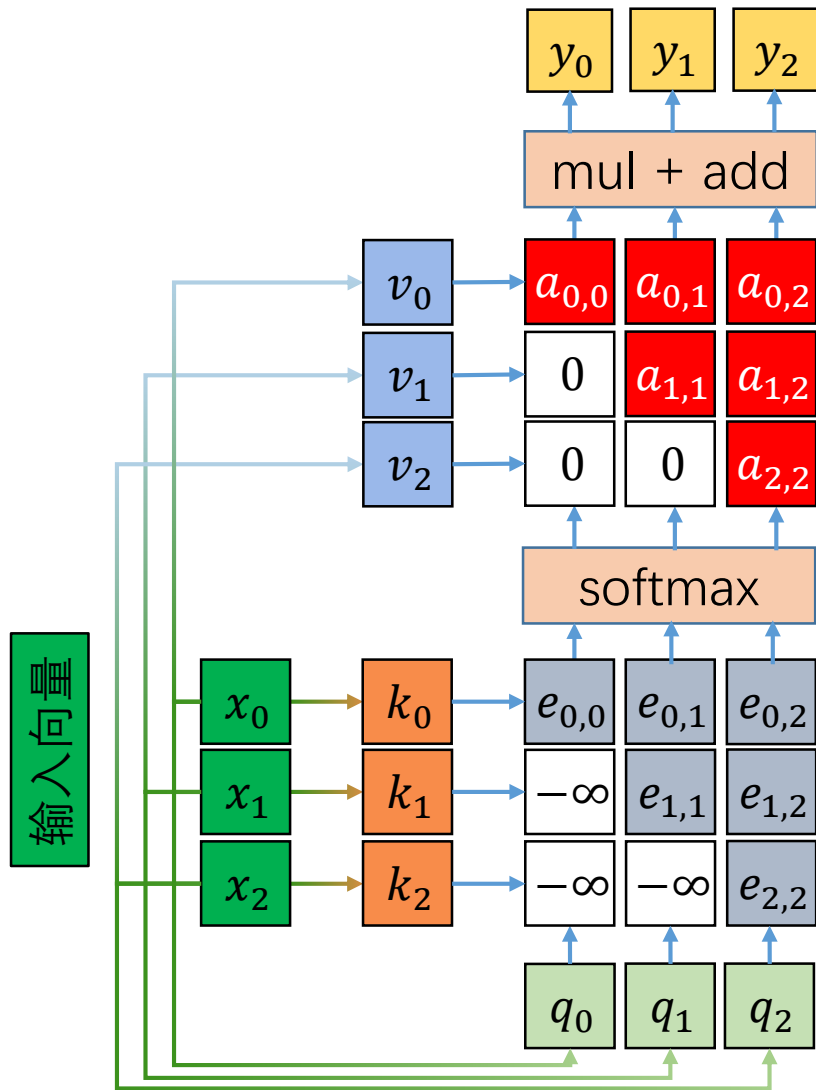
Input:

N 个输入向量: $\mathbf{x} \in \mathbb{R}^{N \times D}$

对齐分数

- ✓ 输入向量是有序性的，每一个位置的输出编码不应该提前看到后面的输入
- ✓ 输出 y_0 时不应该看到 x_1 和 x_2 ，输出 y_1 时不应该看到 x_2

遮挡的自注意力层 (masked self-attention layer)



Output:

context vectors: $\mathbf{y} \in \mathbb{R}^{M \times D_v}$

注意力分数

Computations:

Key向量: $\mathbf{k} = \mathbf{x}W_k$

Value向量: $\mathbf{v} = \mathbf{x}W_v$

Query向量: $\mathbf{q} = \mathbf{x}W_q$

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

注意力: $a_{i,j} = \text{softmax}(e_{i,j})$

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

Input:

N 个输入向量: $\mathbf{x} \in \mathbb{R}^{N \times D}$

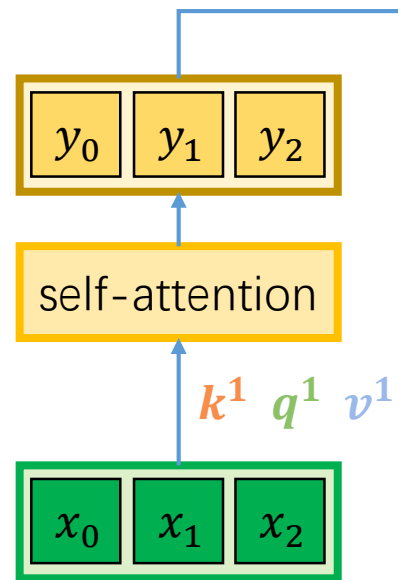
对齐分数

- ✓ 输入向量是有序性的，每一个位置的输出编码不应该提前看到后面的输入
- ✓ 输出 y_0 时不应该看到 x_1 和 x_2 ，输出 y_1 时不应该看到 x_2
- ✓ 方案：将当前向量后面的对齐分数置为负无穷，相应的注意力分数即被置0

针对decoder

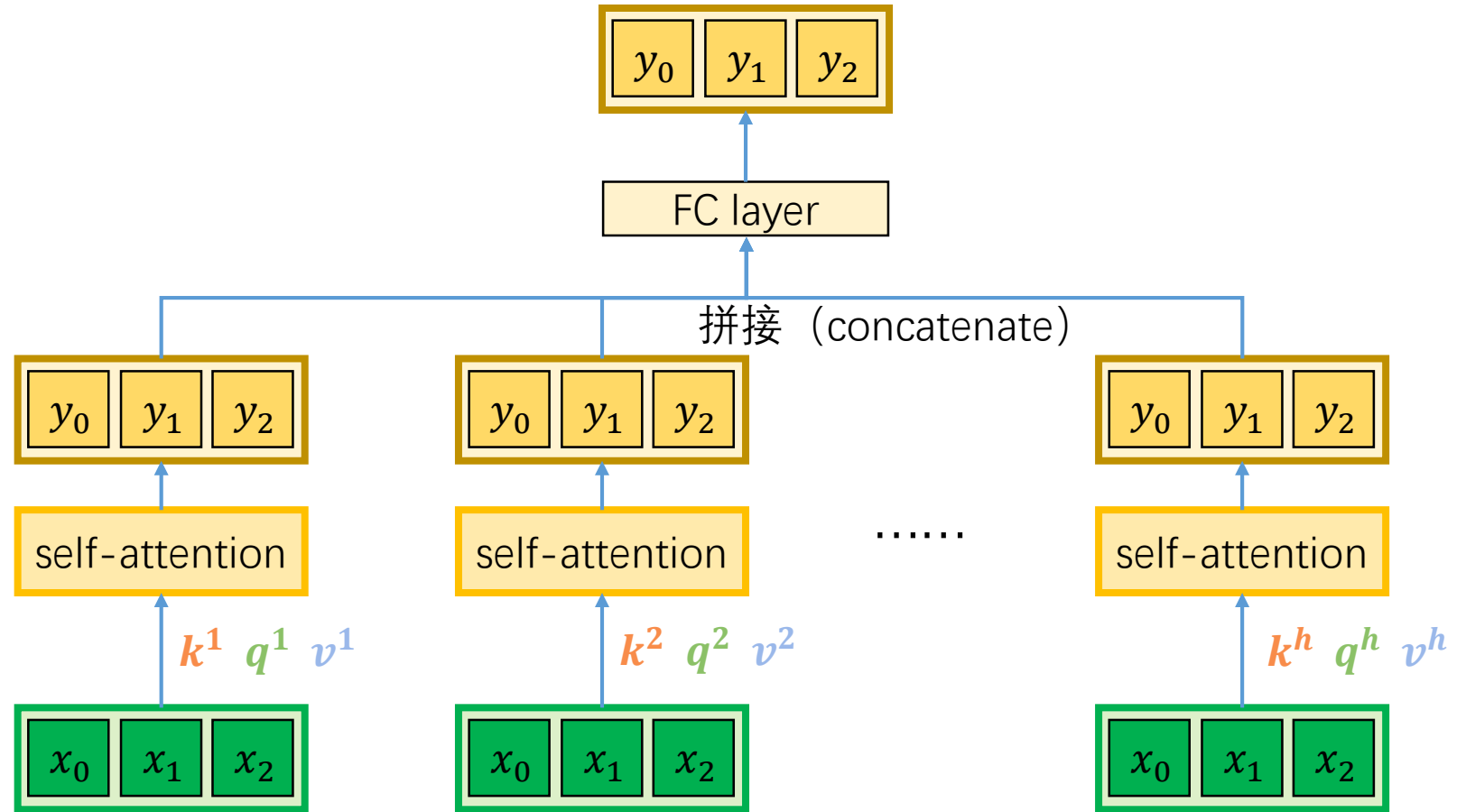
多头自注意力层 (multi-head self-attention layer)

- ✓ 用多组不同的 $k/q/v$ (所谓的head)计算多组不同的注意力分数，每组注意力分数会关注输入中的不同部分，提高对于子空间的特征捕获能力
- ✓ 比如设置 $h = 8$ ，即8个head，每个head生成一组输出，最后将8组输出拼接 (+转换)



多头自注意力层 (multi-head self-attention layer)

- ✓ 用多组不同的 $k/q/v$ (所谓的head)计算多组不同的注意力分数，每组注意力分数会关注输入中的不同部分，提高对于子空间的特征捕获能力
- ✓ 比如设置 $h = 8$ ，即8个head，每个head生成一组输出，最后将8组输出拼接 (+转换)

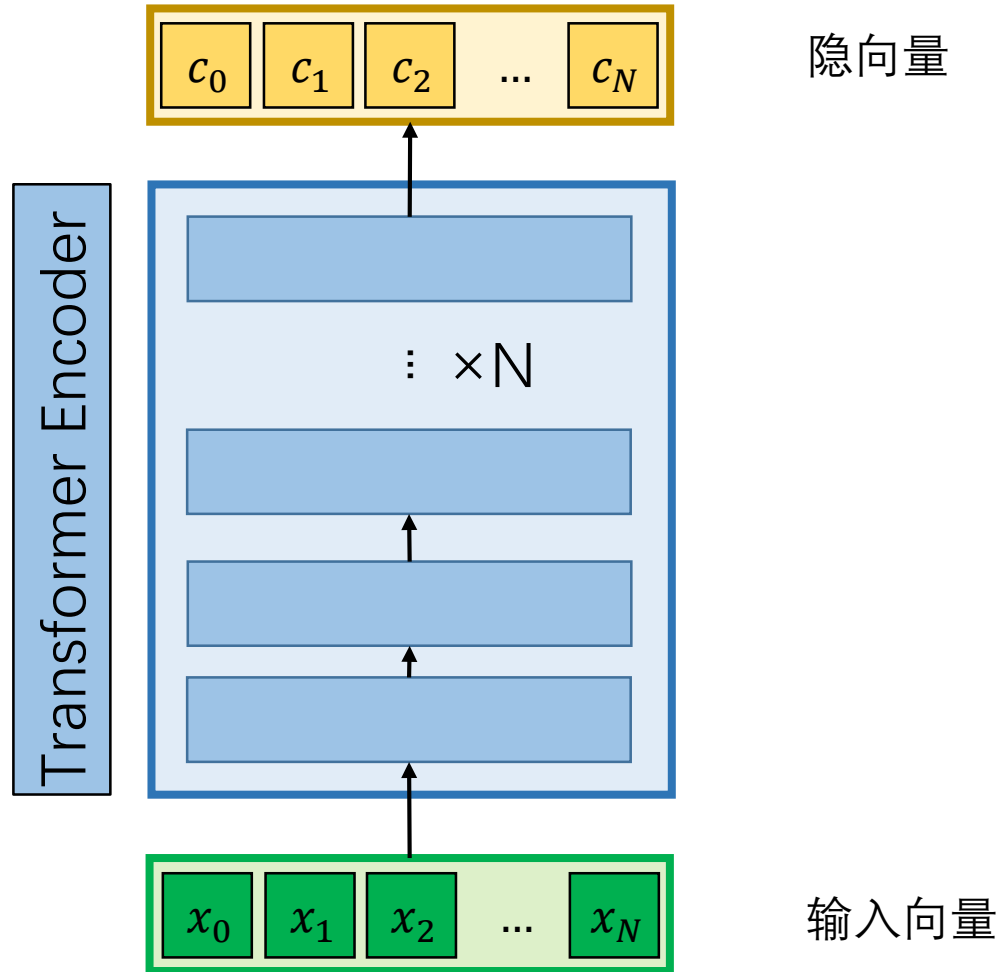


Vaswani, A., Shazeer, N., et al. Attention is all you need. NIPS 2017.

注意力机制

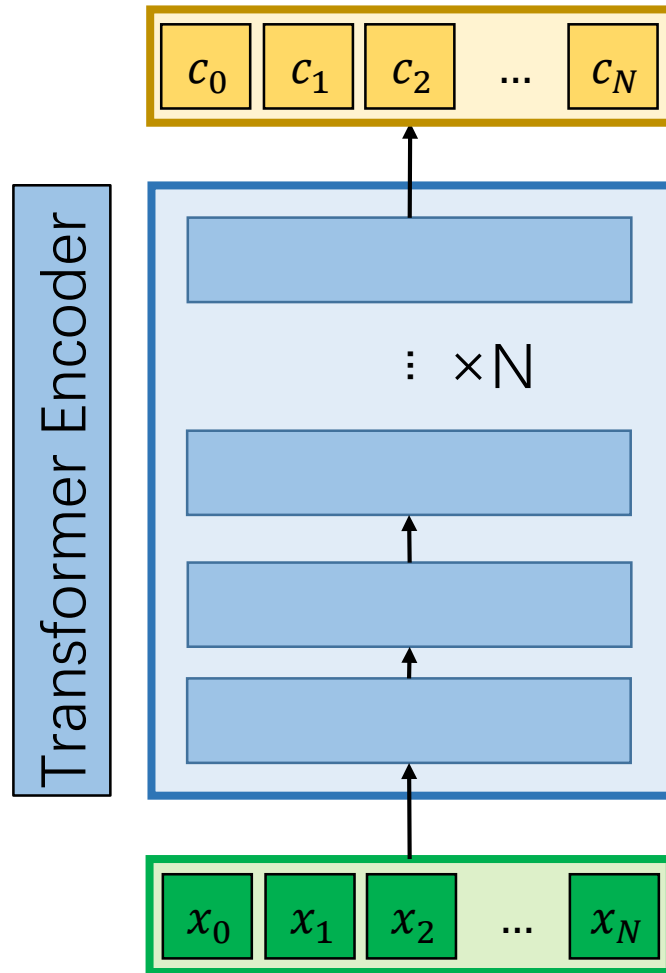
- RNNs中的注意力机制
 - ✓ 计算机视觉中的应用： 图片描述
 - ✓ 自然语言处理中的应用： 机器翻译
- 一般的注意力层结构（general attention layer）
 - ✓ 自注意力（self-attention）
 - ✓ 位置编码（positional encoding）
 - ✓ 遮挡的注意力（masked attention）
 - ✓ 多头注意力（multi-head attention）
- Transformers
 - ✓ 完全基于注意力机制的全新的神经网络结构（相对RNNs和CNNs）

Transformer: Encoder (编码器)



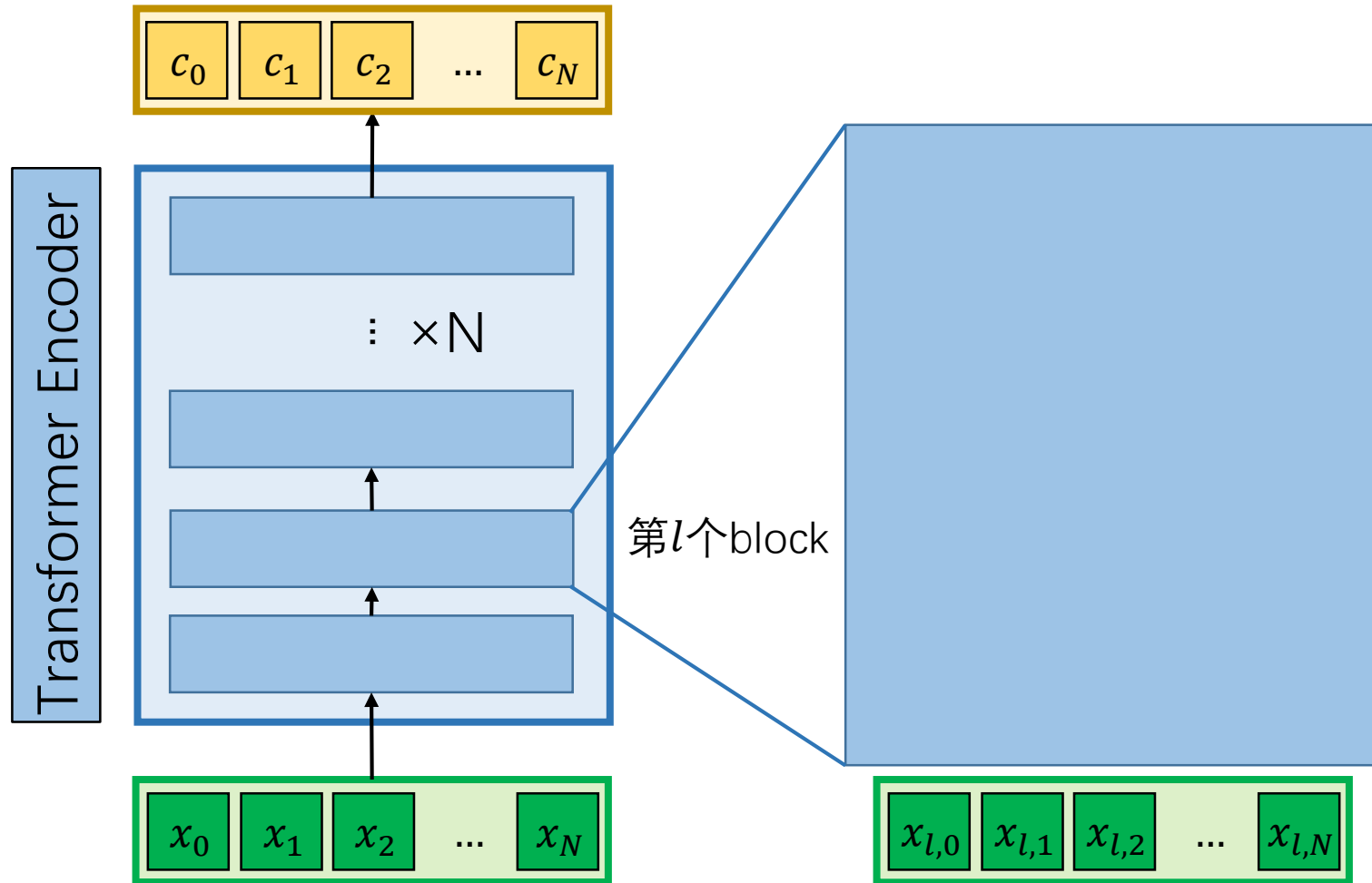
- 编码器+解码器结构
 - ✓ 编码器：将输入编码成隐向量
 - ✓ 解码器：将隐向量解码成输出

Transformer: Encoder



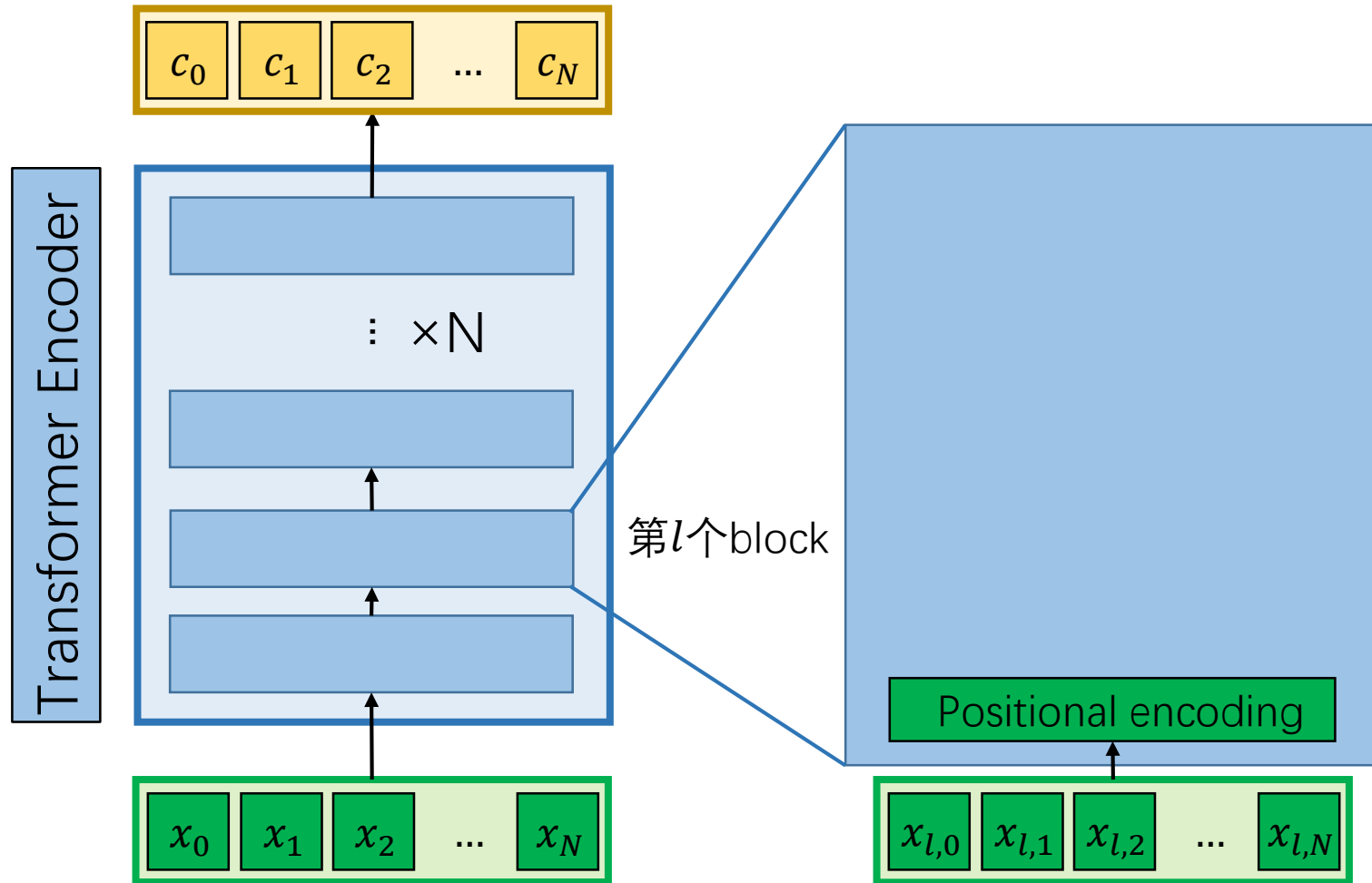
N 个相同的block组成, e.g. $N = 6, D_q = 512$

Transformer: Encoder



N 个相同的block组成, e.g. $N = 6, D_q = 512$

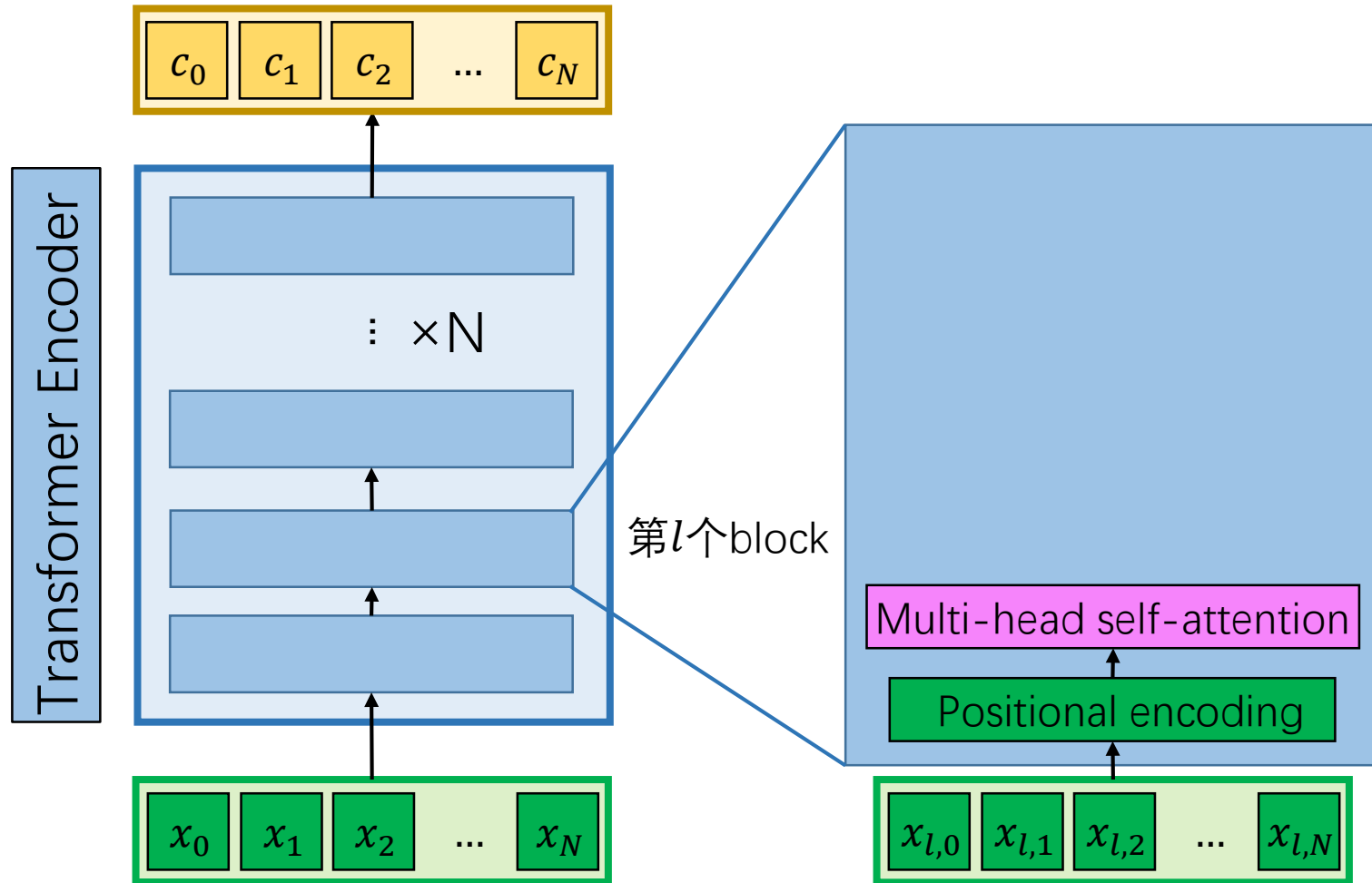
Transformer: Encoder



加入位置编码

N 个相同的block组成, e.g. $N = 6, D_q = 512$

Transformer: Encoder

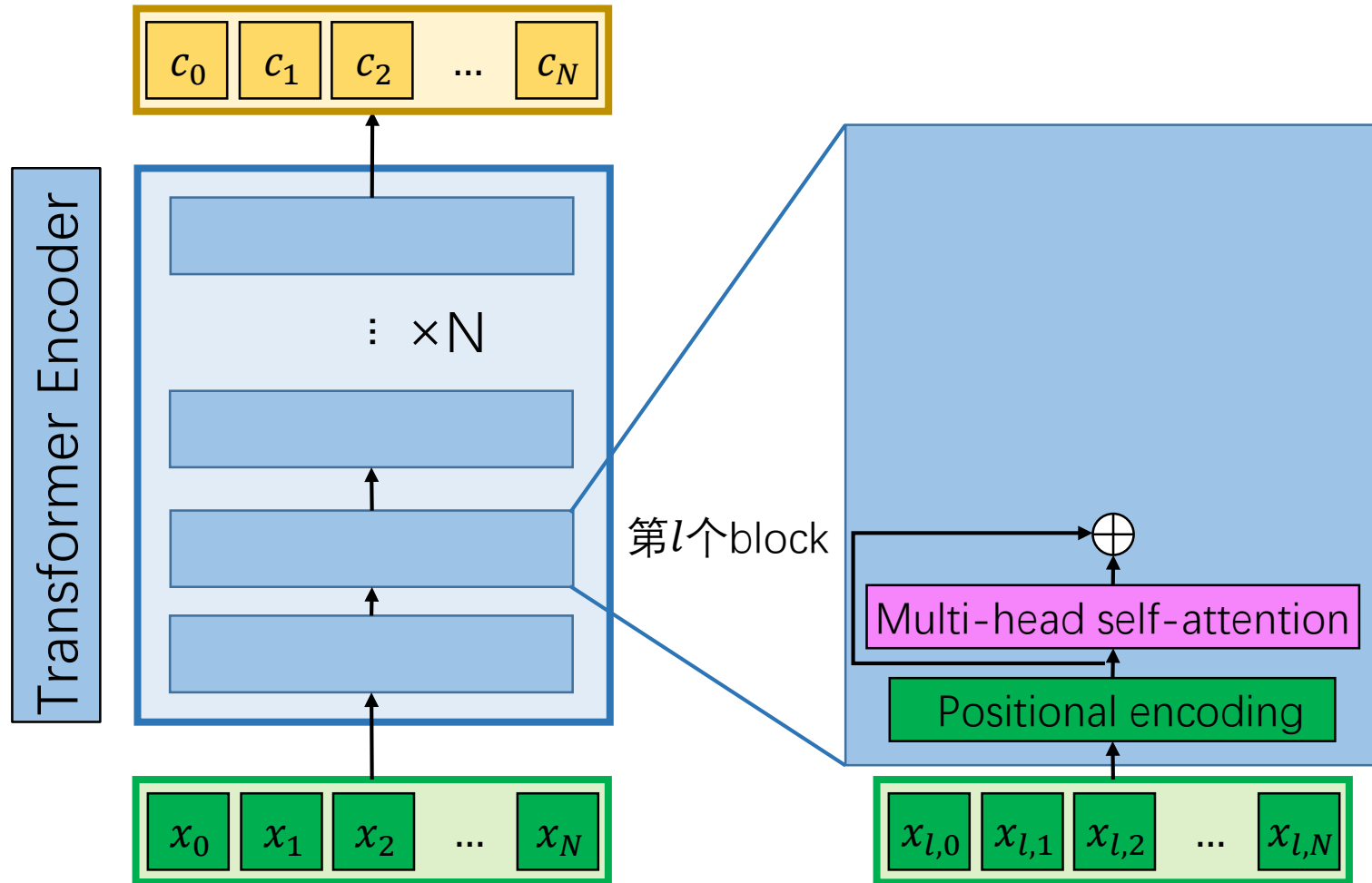


叠加一层多头自注意力层

加入位置编码

N 个相同的block组成, e.g. $N = 6, D_q = 512$

Transformer: Encoder



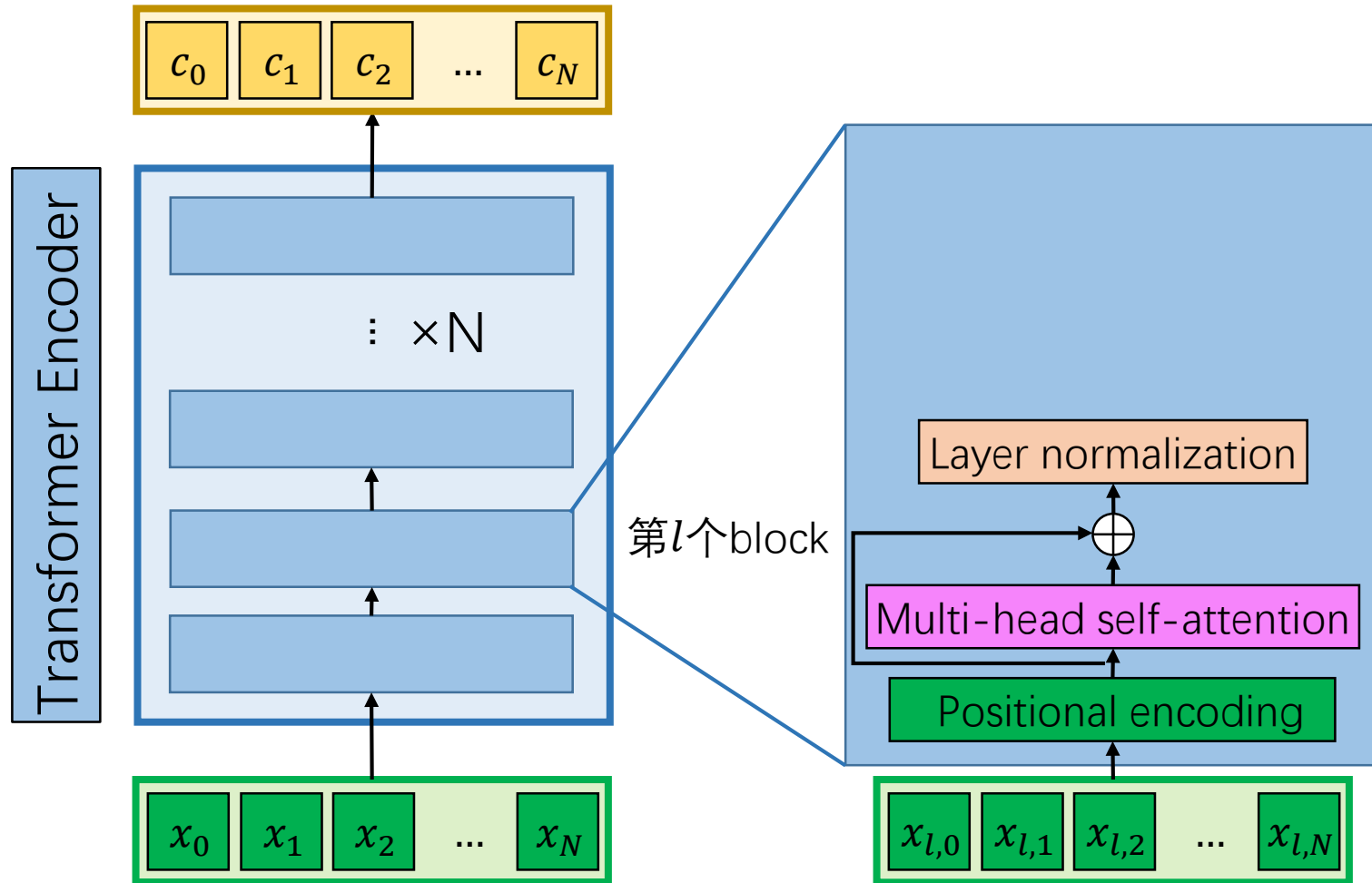
残差连接

叠加一层多头自注意力层

加入位置编码

N 个相同的block组成, e.g. $N = 6, D_q = 512$

Transformer: Encoder



对每个向量分别做layer normalization

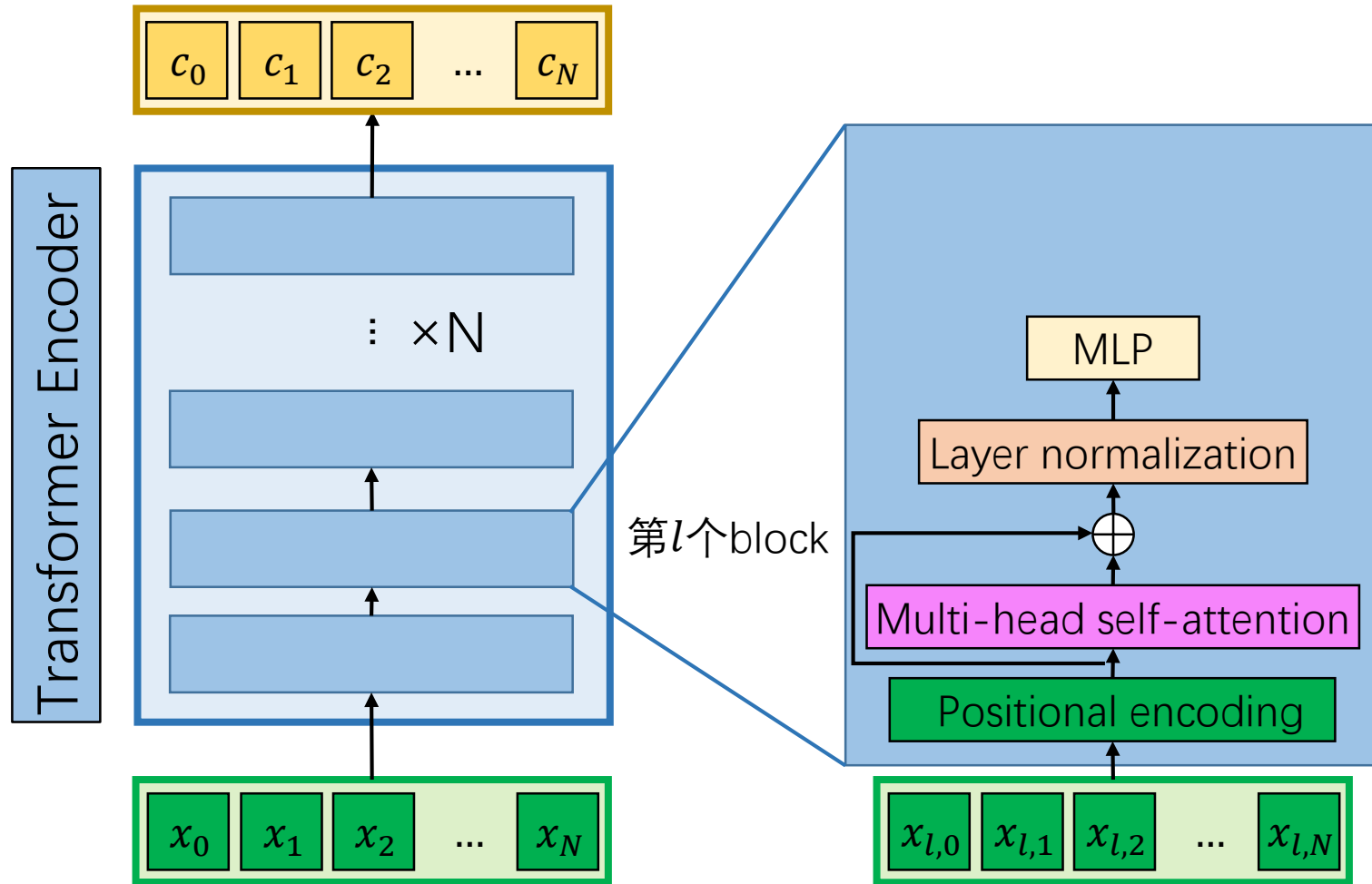
残差连接

叠加一层多头自注意力层

加入位置编码

N 个相同的block组成, e.g. $N = 6, D_q = 512$

Transformer: Encoder



对每个向量分别增加一个MLP层

对每个向量分别做layer normalization

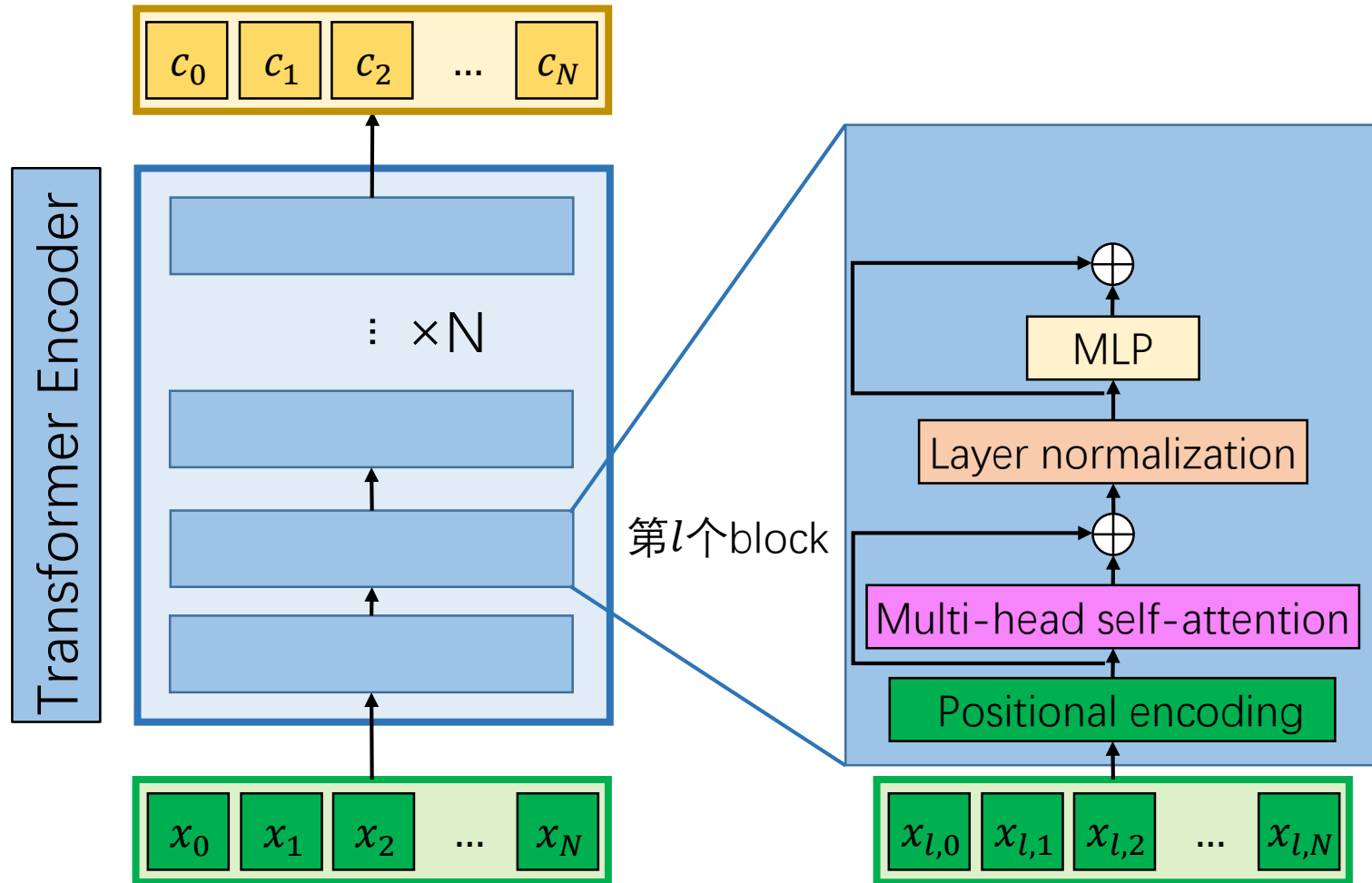
残差连接

叠加一层多头自注意力层

加入位置编码

N 个相同的block组成, e.g. $N = 6, D_q = 512$

Transformer: Encoder



残差连接

对每个向量分别增加一个MLP层

对每个向量分别做layer normalization

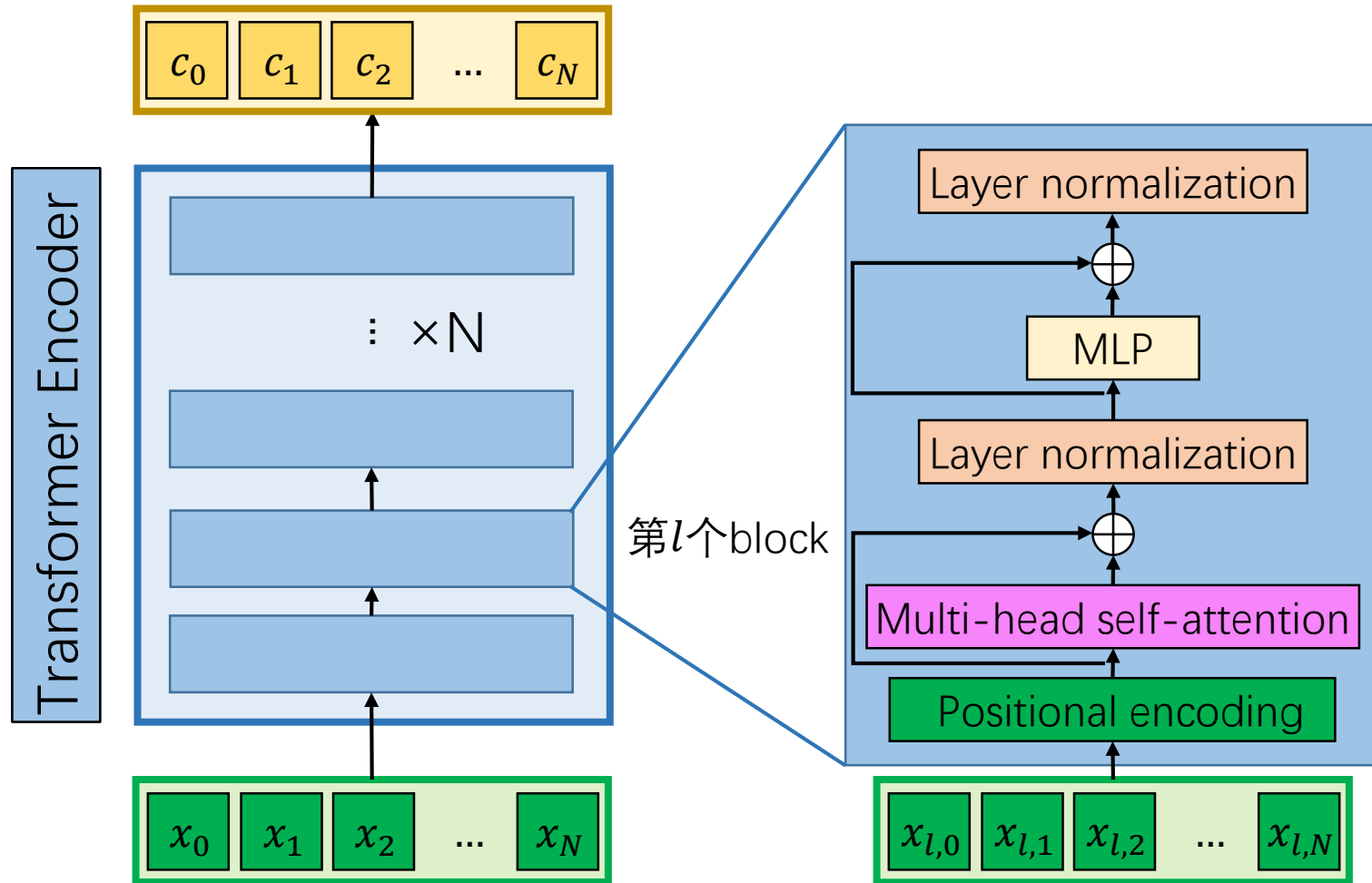
残差连接

叠加一层多头自注意力层

加入位置编码

N 个相同的block组成, e.g. $N = 6, D_q = 512$

Transformer: Encoder



对每个向量分别做layer normalization

残差连接

对每个向量分别增加一个MLP层

对每个向量分别做layer normalization

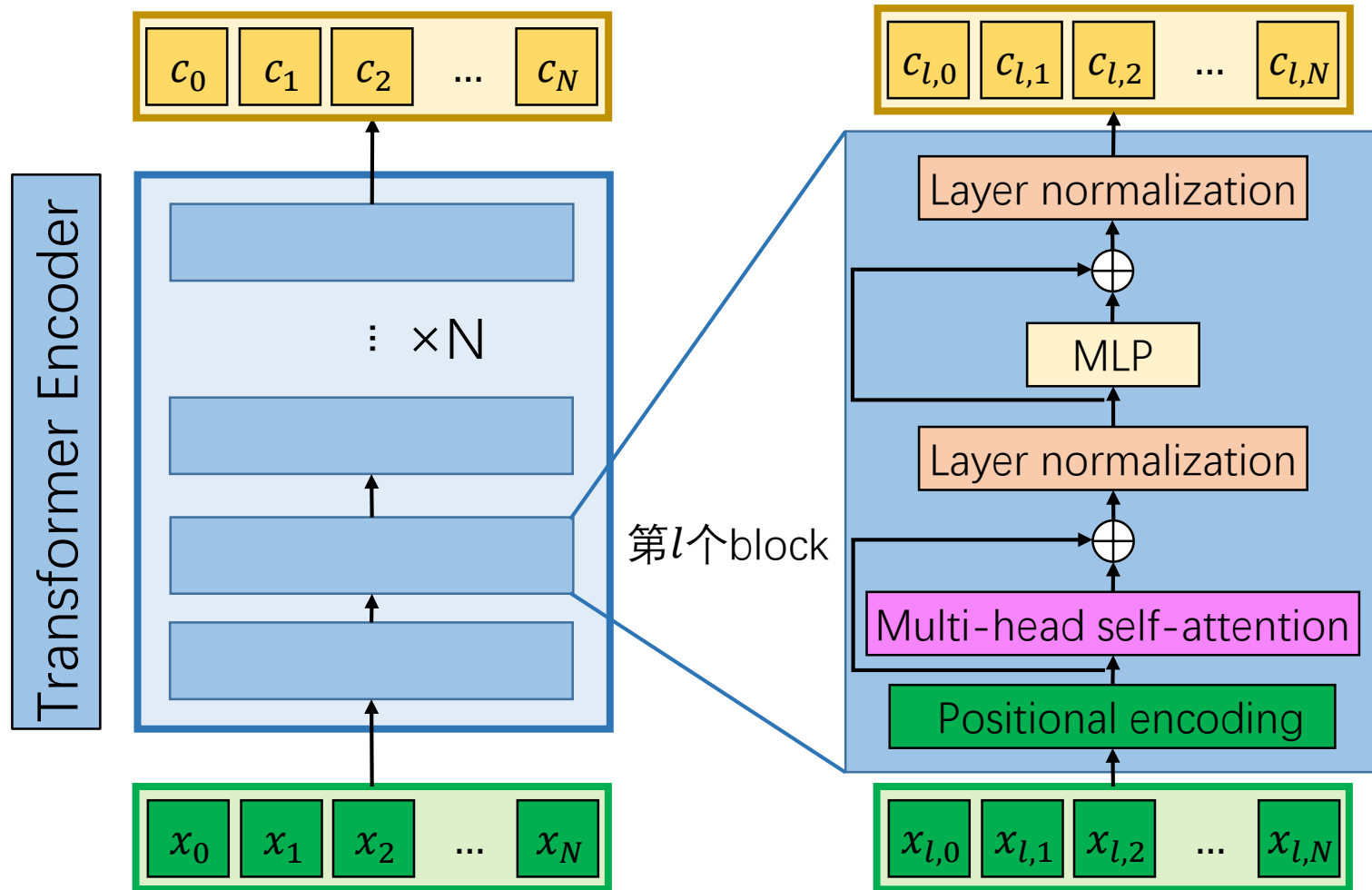
残差连接

叠加一层多头自注意力层

加入位置编码

N 个相同的block组成, e.g. $N = 6, D_q = 512$

Transformer: Encoder



输出作为第 $l + 1$ 个block的输入

对每个向量分别做layer normalization

残差连接

对每个向量分别增加一个MLP层

对每个向量分别做layer normalization

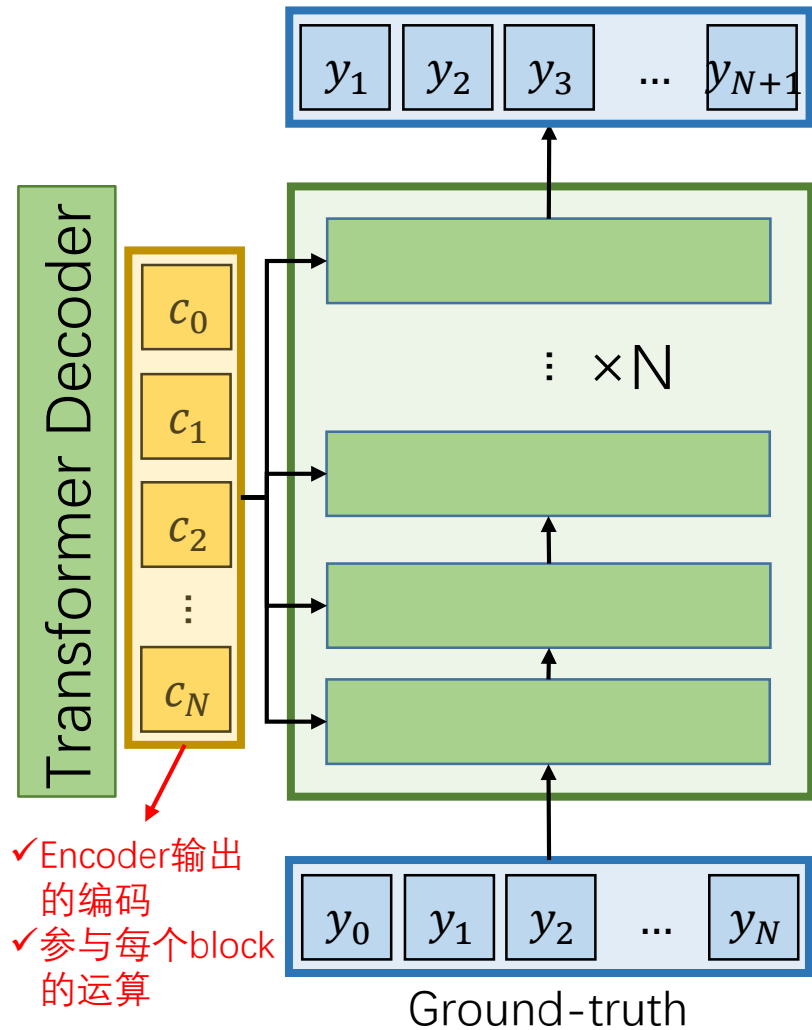
残差连接

叠加一层多头自注意力层

加入位置编码

N 个相同的block组成, e.g. $N = 6, D_q = 512$

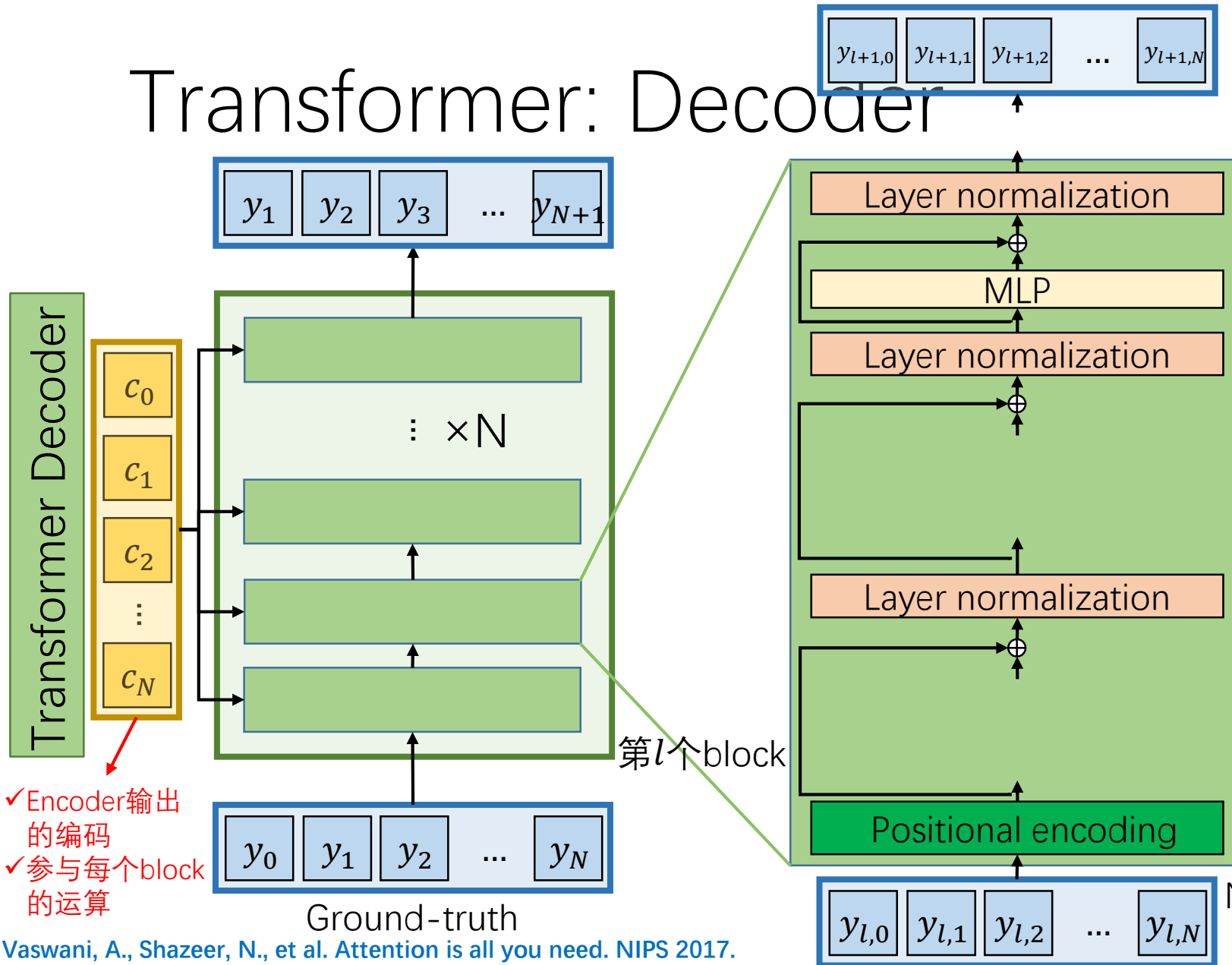
Transformer: Decoder



N 个相同的block组成, e.g. $N = 6, D_q = 512$

Transformer: Decoder

大部分结构和Encoder类似

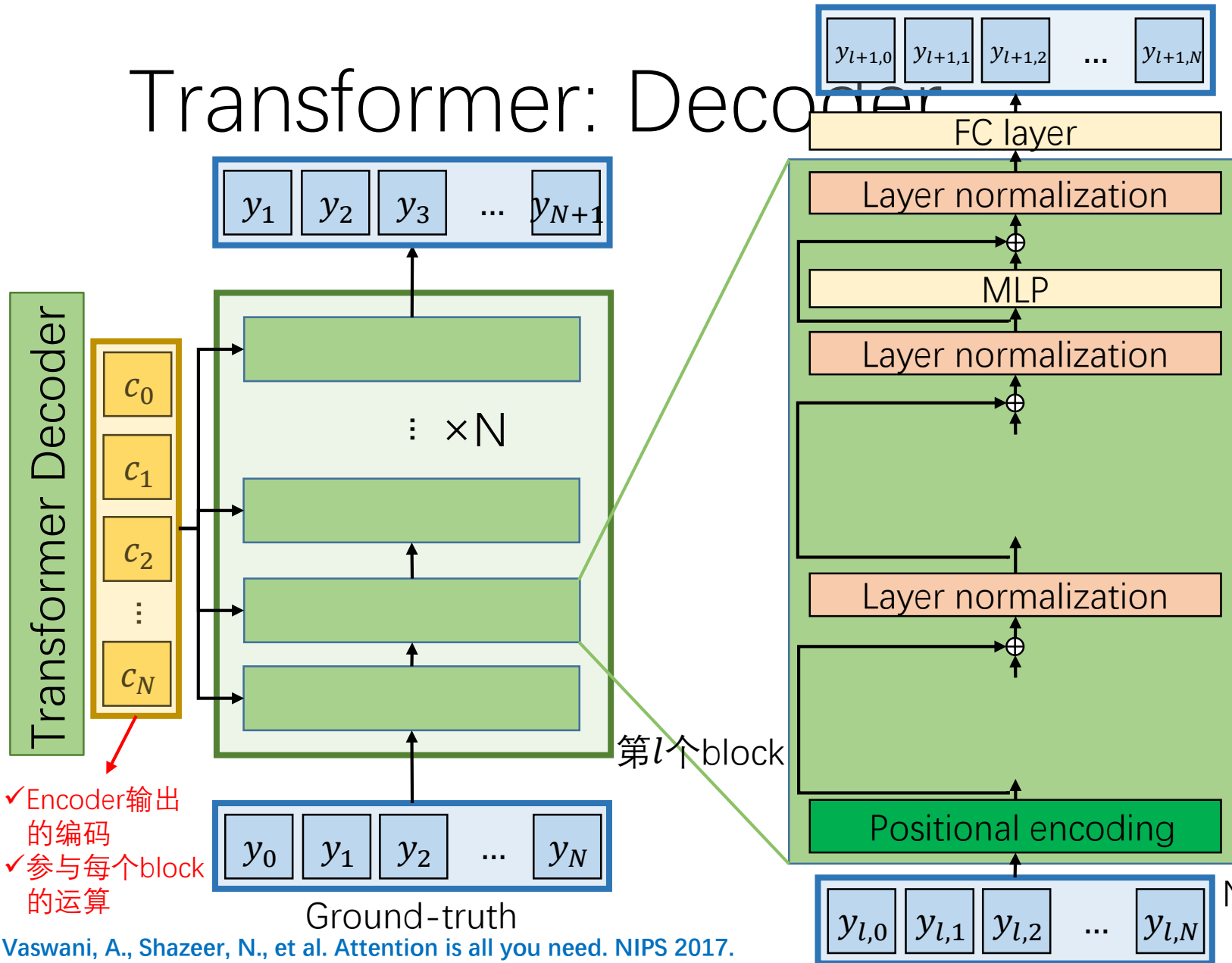


N 个相同的block组成, e.g. $N = 6, D_q = 512$

- ✓ Encoder输出的编码
- ✓ 参与每个block的运算

Vaswani, A., Shazeer, N., et al. Attention is all you need. NIPS 2017.

Transformer: Decoder



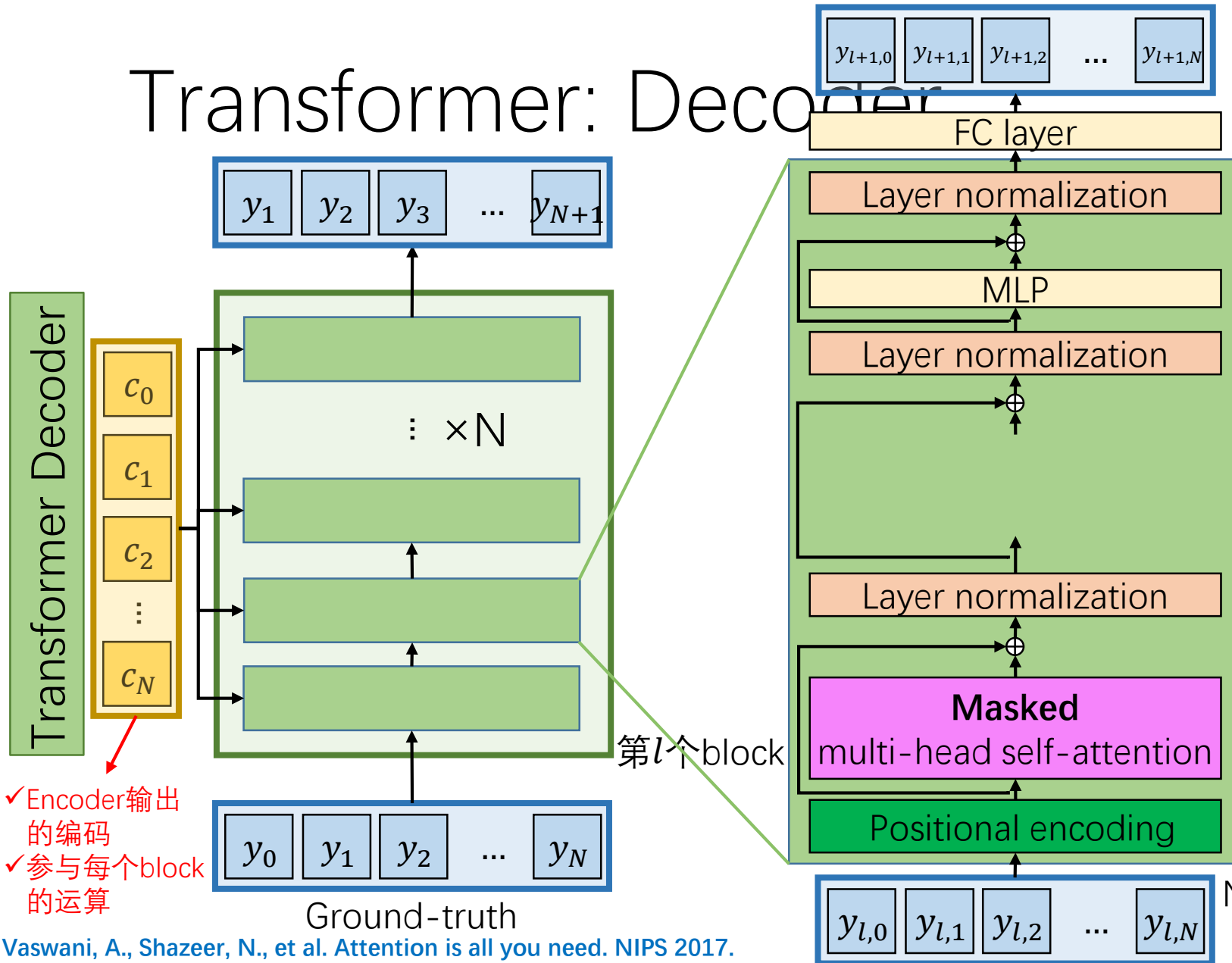
大部分结构和Encoder类似
每层输出之前增加一个全连接层

N 个相同的block组成, e.g. $N = 6, D_q = 512$

- ✓ Encoder输出的编码
- ✓ 参与每个block的运算

Vaswani, A., Shazeer, N., et al. Attention is all you need. NIPS 2017.

Transformer: Decoder



大部分结构和Encoder类似
每层输出之前增加一个全连接层

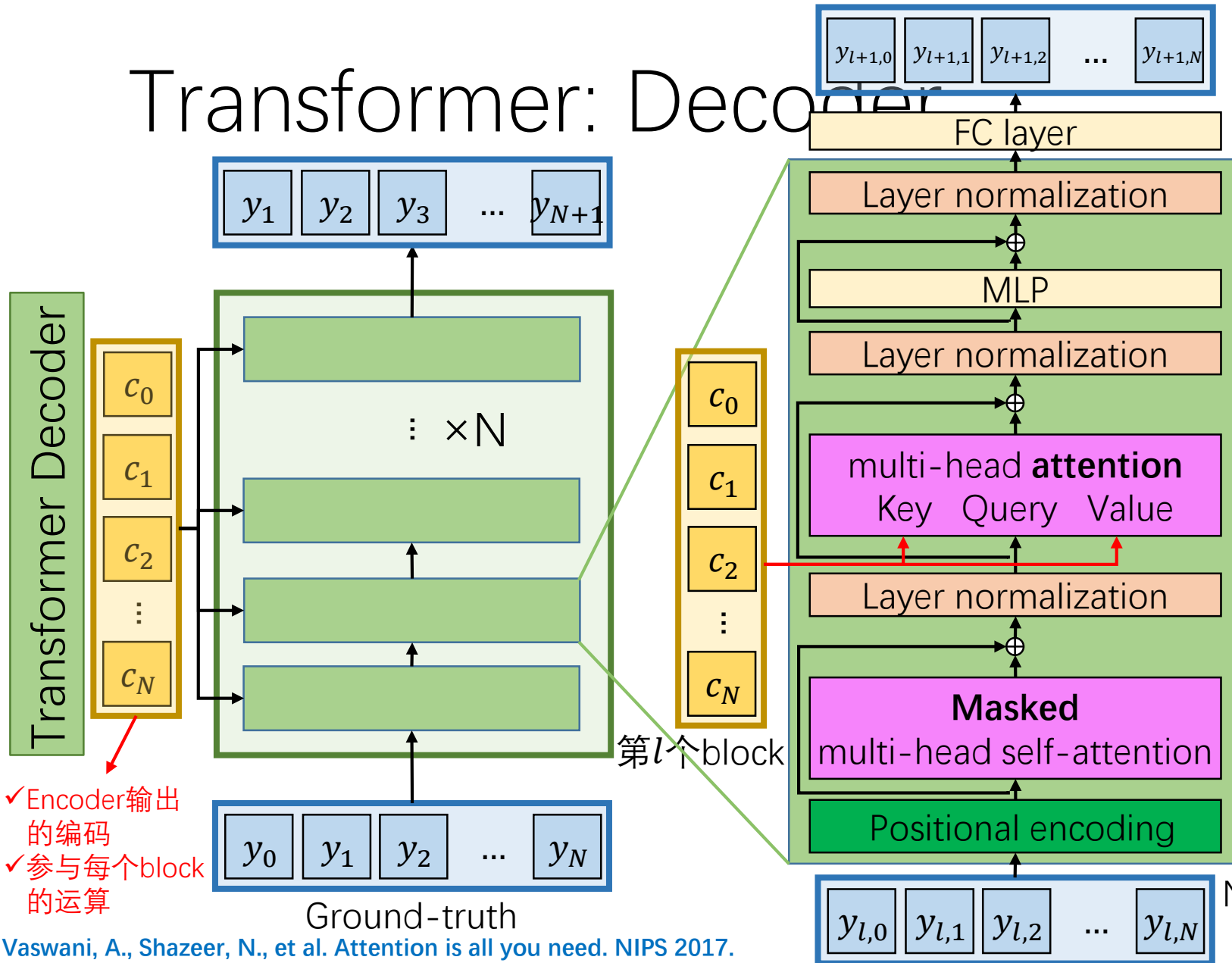
遮挡的注意力层，每个输出不能看到其后面的向量

N 个相同的block组成， e.g. $N = 6, D_q = 512$

- ✓ Encoder输出的编码
- ✓ 参与每个block的运算

Vaswani, A., Shazeer, N., et al. Attention is all you need. NIPS 2017.

Transformer: Decoder



大部分结构和Encoder类似
每层输出之前增加一个全连接层

Decoder的输入和Encoder的输出做
multi-head attention计算:
Encoder输出生成keys和values
Decoder输入生成queries

遮挡的注意力层, 每个输出不能看到
其后面的向量

N 个相同的block组成, e.g. $N = 6, D_q = 512$

✓ Encoder输出的
编码
✓ 参与每个block
的运算

Vaswani, A., Shazeer, N., et al. Attention is all you need. NIPS 2017.

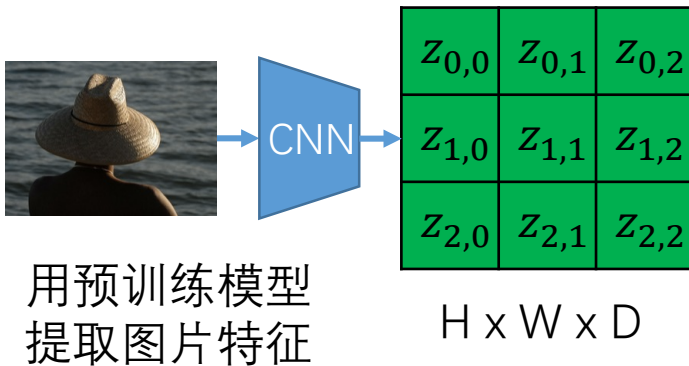
RNNs vs. Transformers

- RNNs
 - + RNN对长序列容易梯度爆炸/消失，LSTM对长序列性能较好
 - 顺序计算：要计算当前时刻的隐向量，必须等前一时刻隐向量计算完毕
 - 输入必须是有序的
- Transformers
 - + 对长序列性能很好，每个时刻可以对所有时刻输入做attention
 - + 可以处理无序的输入，也可以处理加入位置编码的有序输入
 - + 所有时刻的注意力计算可以高度并行
 - 对GPU内存要求高：需要存储对齐分数和注意力分数两个矩阵（每个head分别存储这两个矩阵）

使用Transformer训练图片描述任务

Input: 图片 I

Output: 文本 $y = y_1, y_2, \dots, y_T$



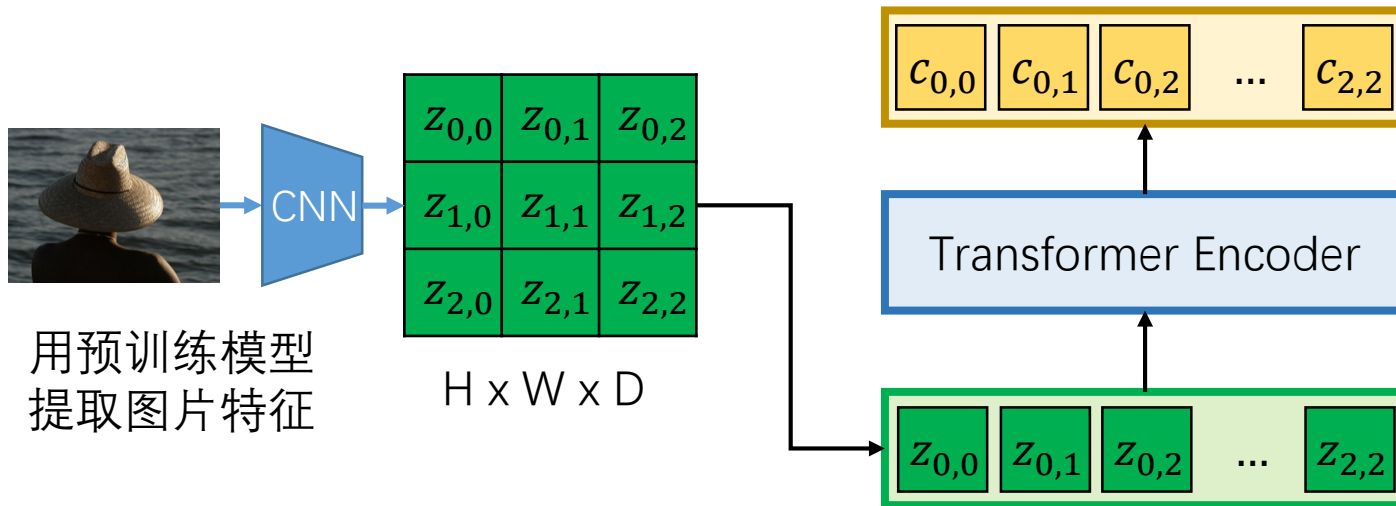
使用Transformer训练图片描述任务

Input: 图片 I

Output: 文本 $y = y_1, y_2, \dots, y_T$

Encoder: $c = T_e(Z)$

- ✓ Z : CNN提取的图片特征
- ✓ $T_e()$: Transformer encoder



用预训练模型
提取图片特征

使用Transformer训练图片描述任务

Input: 图片 I

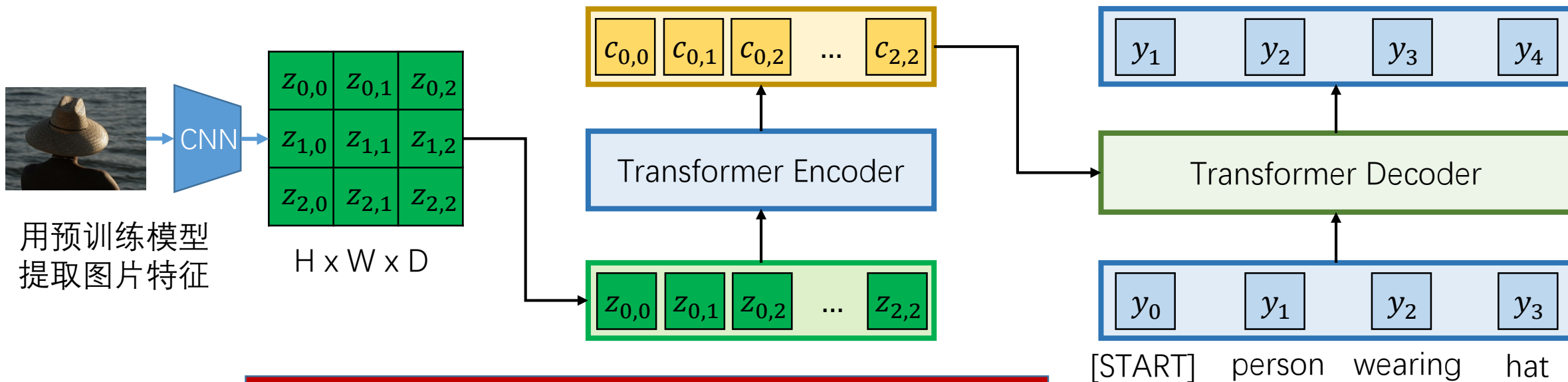
Output: 文本 $y = y_1, y_2, \dots, y_T$

Encoder: $c = T_e(Z)$

- ✓ Z : CNN提取的图片特征
- ✓ $T_e()$: Transformer encoder

Decoder: $y_t = T_d(y_{0:t-1}, c)$

- ✓ c : Transformer encoder的输出
- ✓ $T_d()$: Transformer decoder



相比RNN/LSTM, 没有循环迭代, 训练时可以高度并行

使用Transformer训练图片描述任务

Input: 图片 I

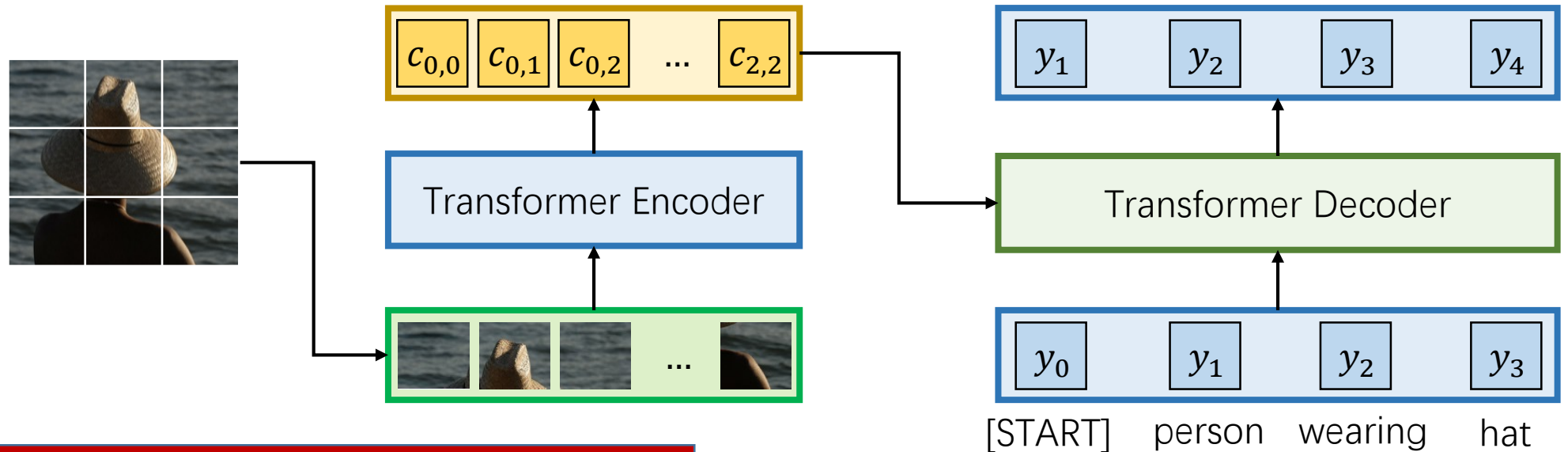
Output: 文本 $y = y_1, y_2, \dots, y_T$

Encoder: $c = T_e(Z)$

- ✓ Z : CNN提取的图片特征
- ✓ $T_e()$: Transformer encoder

Decoder: $y_t = T_d(y_{0:t-1}, c)$

- ✓ c : Transformer encoder的输出
- ✓ $T_d()$: Transformer decoder



甚至可以舍弃CNN，直接使用图片像素

小结

- RNNs+attention应用于图片描述和机器翻译
 - ✓模型可以在每个时刻注意输入的不同部分，提高输出准确度
- 一般注意力层
 - ✓有别于RNNs和CNNs的神经网络结构
- Transformers
 - ✓一种基于自注意力机制的encoder-decoder模型
 - ✓高可扩展，高度并行
 - ✓收敛更快，模型更大，CV和NLP任务上性能更好
 - ✓逐渐代替RNNs和CNNs

L13: 目标检测和图像分割