# INF721

2024/2

UFV

# Deep Learning

## L17: Transformers

# Logistics

**Last Lecture**

- Machine Translation
- Decoding
  - Greedy Search
  - Beam Search
- Attention in RNNs

UFV

# Lecture Outline

- Machine Translation
- Problems with RNNs
- Transformers
  - Self-Attention
  - Multi-head Attention
  - Encoder & Decoder
  - Positional Encoding
  - Masked Multi-head Attention

# Machine Translation

Given a dataset of sentence pairs:

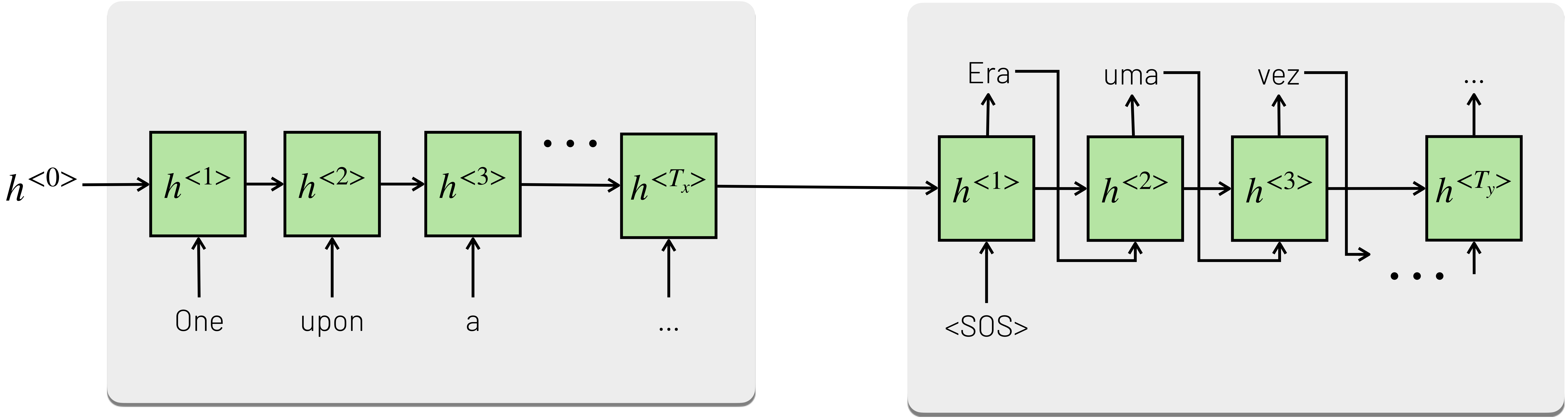
$(x = \{x^{<1>}, x^{<2>}, \ldots, x^{<T_x>}\}, y = \{y^{<1>}, y^{<2>}, \ldots, y^{<T_y>}\})$,

we want to learn a model that maps $x$ into $y$.

| Portuguese | English |
|---|---|
| Olá, como vai você? | Hello, how are you? |
| O livro está em cima da mesa. | The book is on the table. |
| Lucas irá viajar ao Rio em Dezembro. | Lucas is travelling to Rio in December. |
| Em Dezembro, Lucas irá viajar ao Rio. | Lucas is travelling to Rio in December. |
| .... | .... |

# Problems with RNNs

- Struggle to capture long dependencies in sequences
- Hard to parallelize



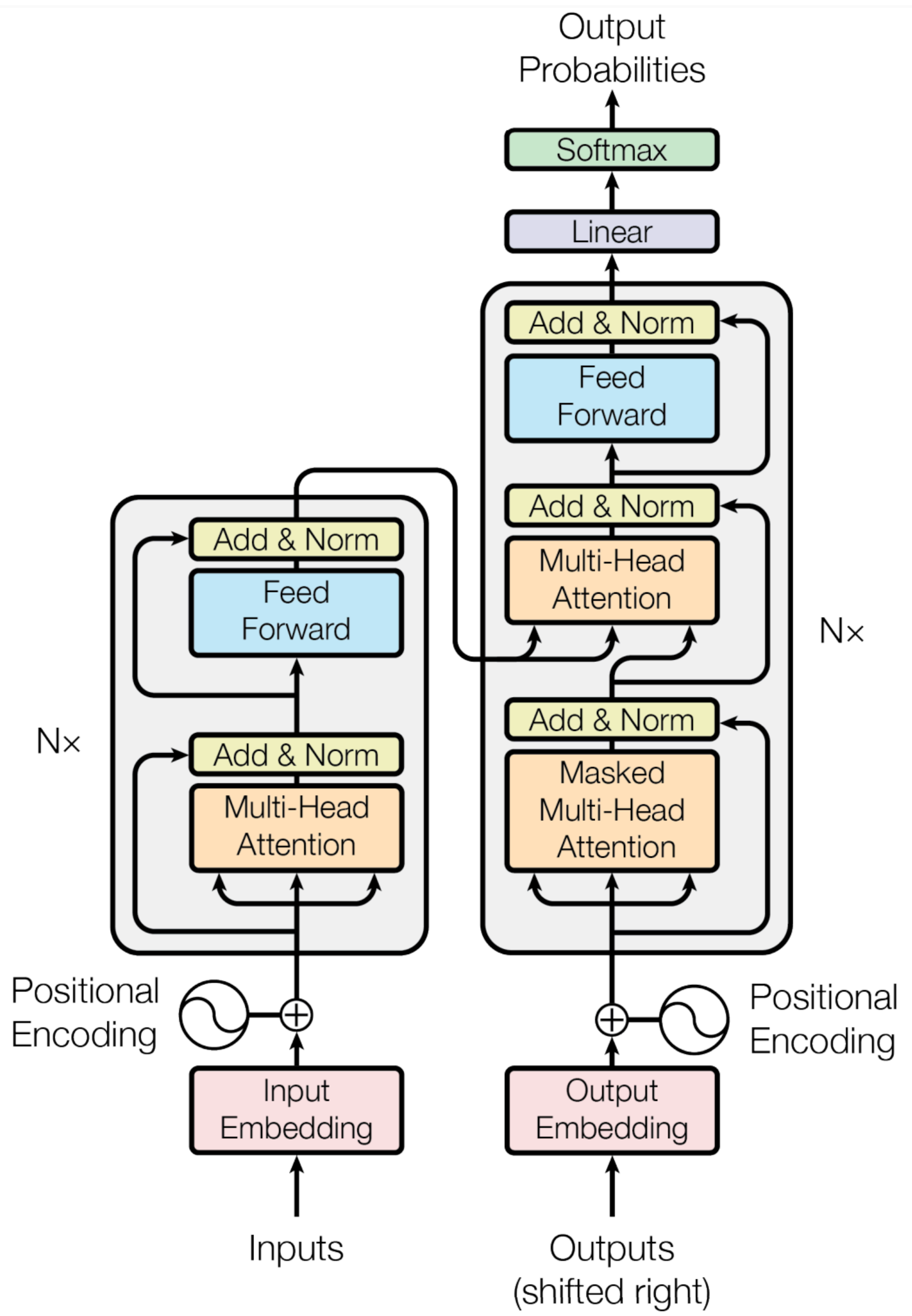

**Encoder** **Decoder**

# Transformers

**Transformers** are an encoder-decoder architecture to process sequences using only attention (eliminating recurrence).

Initially proposed for machine translation, but proved to be very effective in many other problems in:

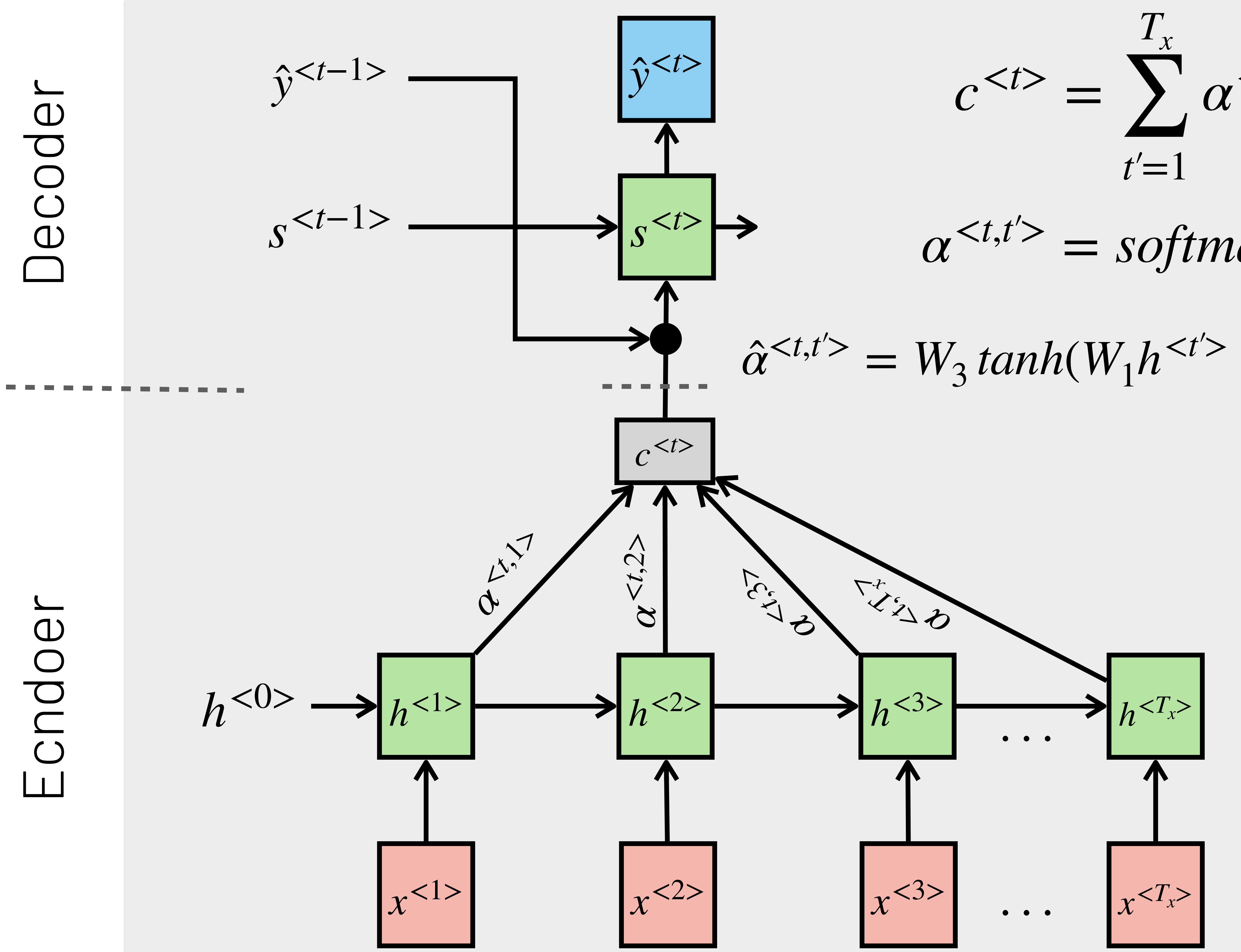
- Natural Language Processing
- Computer Vision
- Reinforcement Learning
- ...




Vaswani et al. 2017, Attention Is All You Need

# Attention in RNNs vs. Transformers

## **RNNs** Badahnau (Additive) Attention

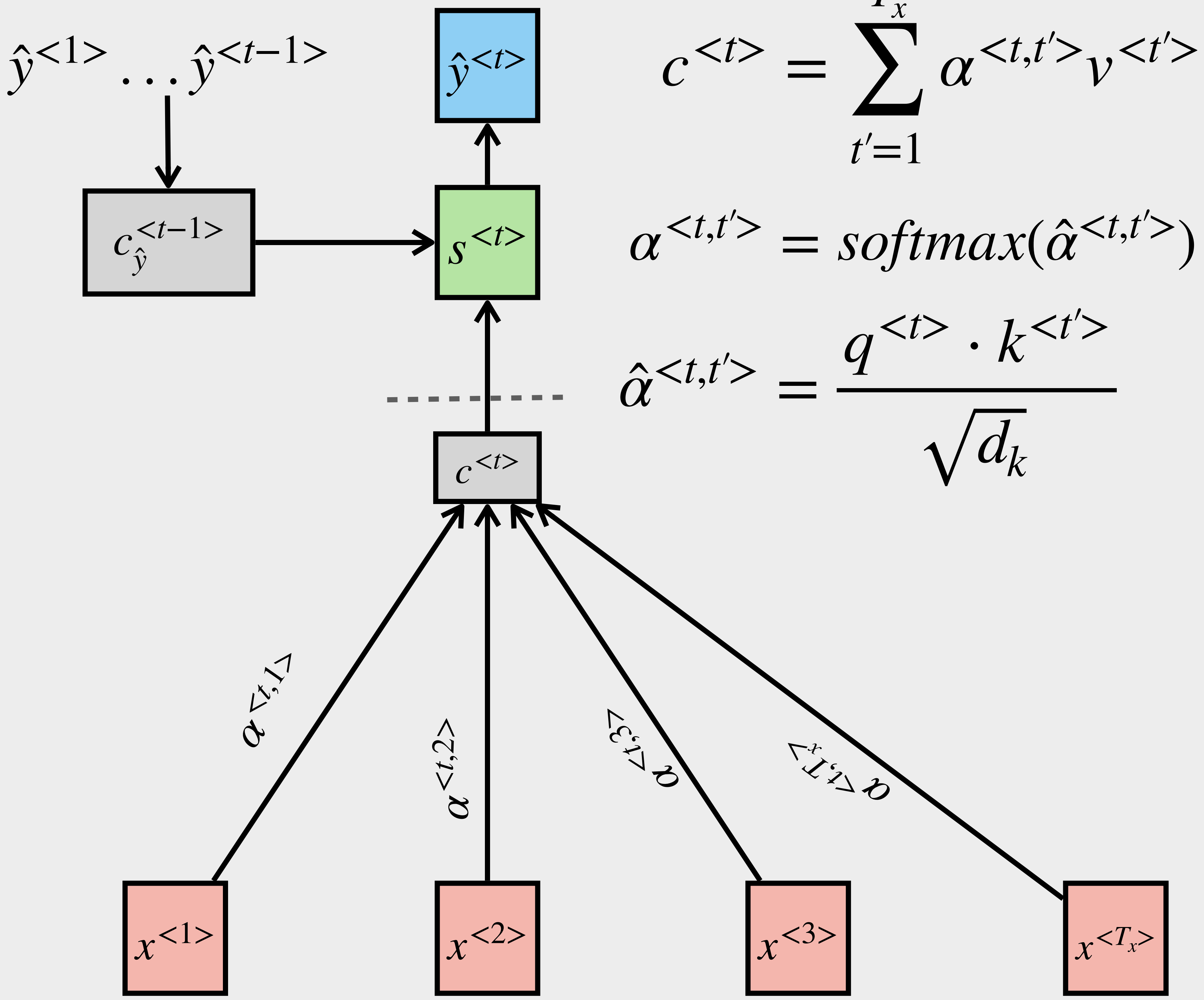



$$c^{<t>} = \sum_{t'=1}^{T_x} \alpha^{<t,t'>} h^{<t'>}$$

$$\alpha^{<t,t'>} = softmax(\hat{\alpha}^{<t,t'>})$$

$$\hat{\alpha}^{<t,t'>} = W_3\, tanh(W_1 h^{<t'>} + W_2 s^{<t-1>})$$

## **Transformers** Scaled Dot-Product Attention



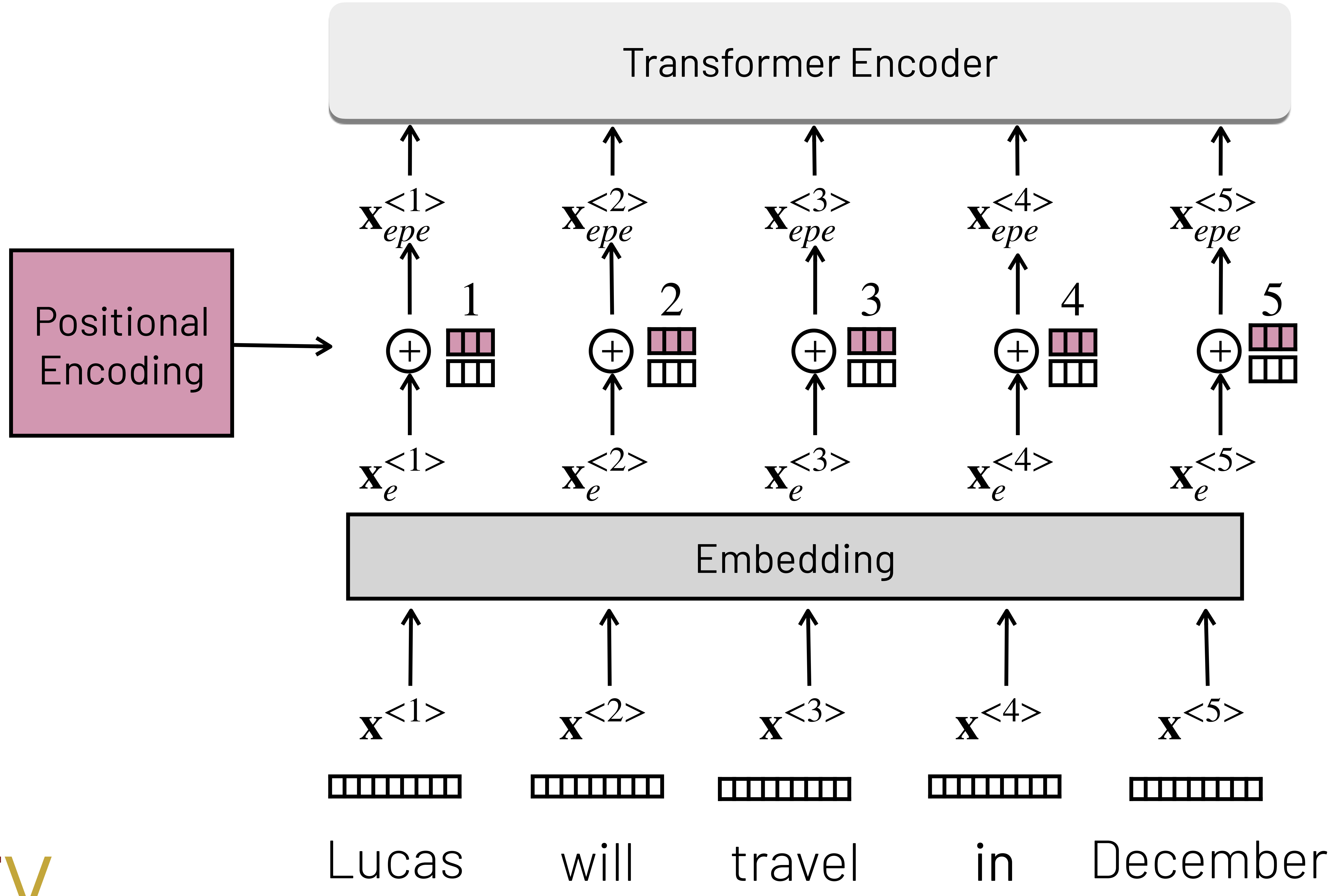

$$c^{<t>} = \sum_{t'=1}^{T_x} \alpha^{<t,t'>} v^{<t'>}$$

$$\alpha^{<t,t'>} = softmax(\hat{\alpha}^{<t,t'>})$$

$$\hat{\alpha}^{<t,t'>} = \frac{q^{<t>} \cdot k^{<t'>}}{\sqrt{d_k}}$$

# Encoder Input

The transformer encoder takes as input a sequence of word embeddings summed with positional encodings. This sequence has the constant size $(T_x, d_{model})$ throughout the entire model




The contextual embeddings size is typically called $d_{model}$

$X_{epe} = X_e + PE$

$X_{epe} \in \mathbb{R}^{T_x \times d_{model}} \rightarrow$ word + pos. embed.

$X_e = EX$

$X_e \in \mathbb{R}^{T_x \times d_{model}} \rightarrow$ word embedding

$E \in \mathbb{R}^{|V| \times d_{model}} \rightarrow$ embedding matrix

$X \in \mathbb{R}^{T_x \times |V|} \rightarrow$ one-hot

# Self-Attention

The key idea behing the Transformer is the **self-attention mechanism**, which learns a context vector $c^{<t>}$ for each input element $x^{<t>}$ based on the input sequence $x$ itself.

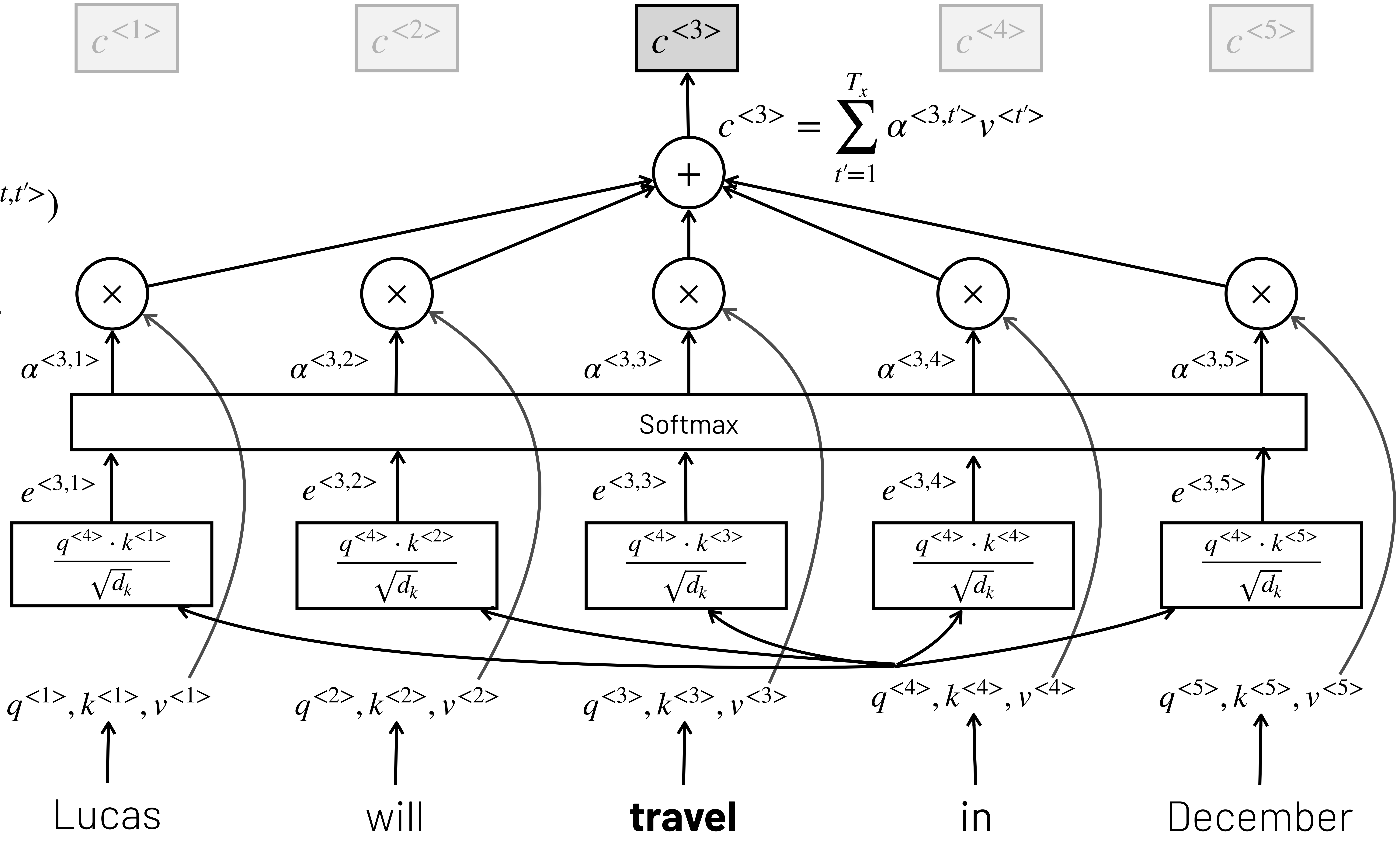
$$\alpha^{<t,t'>} = softmax(\hat{\alpha}^{<t,t'>})$$

$$\hat{\alpha}^{<t,t'>} = \frac{q^{<t>} \cdot k^{<t'>}}{\sqrt{d_k}}$$

$$q^{<t>} = \mathbf{x}_{epe}^{<t>} W^q$$

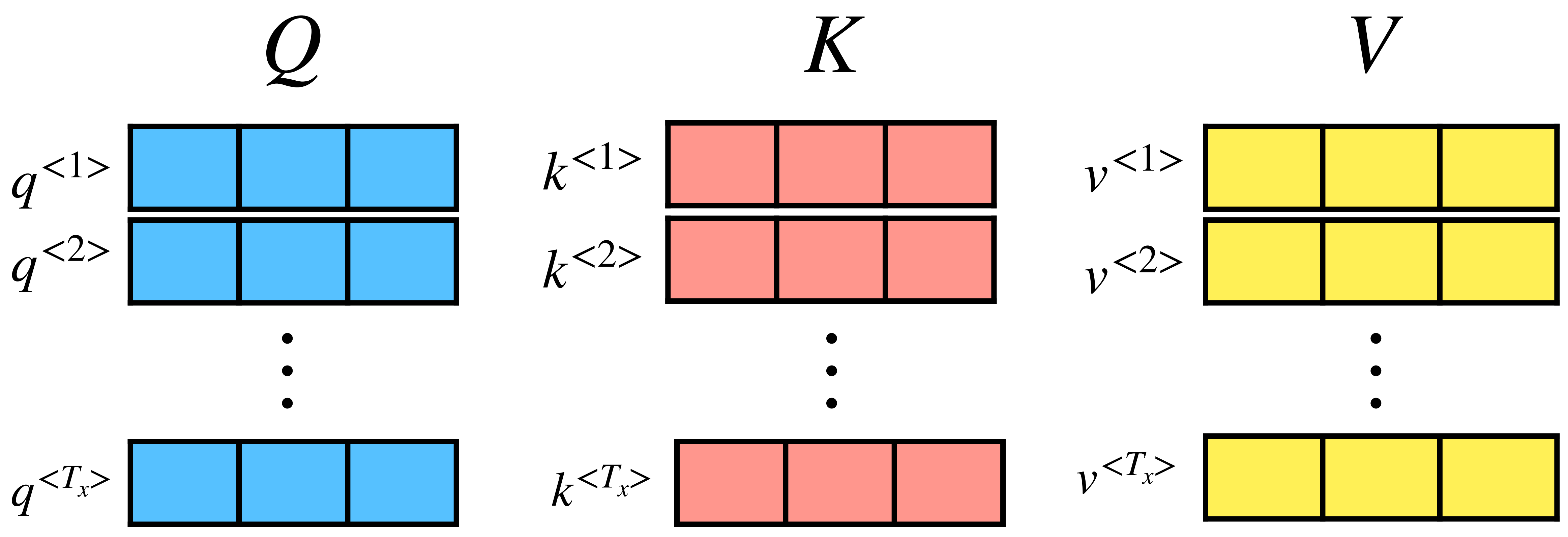
$$k^{<t>} = \mathbf{x}_{epe}^{<t>} W^k$$

$$v^{<t>} = \mathbf{x}_{epe}^{<t>} W^v$$

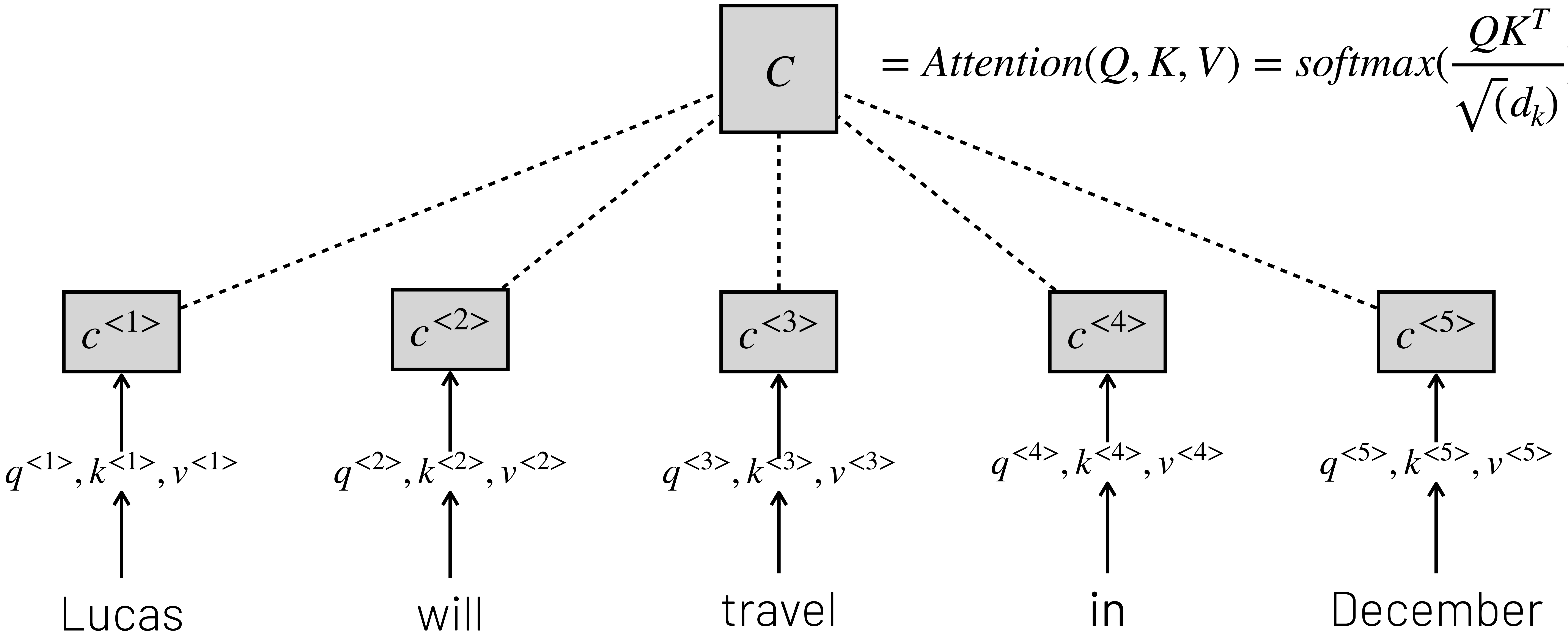


# Self-Attention

The contextal represention $C = \{c^{<1>}, \ldots, c^{<T_x>}\}$ of the entire input sequence $x = \{x^{<1>}, \ldots, x^{<T_x>}\}$ can be computed in a vectorized way combining vectors $q^{<t>}$, $k^{<t>}$, $v^{<t>}$ in matrices $Q$, $K$ e $V$



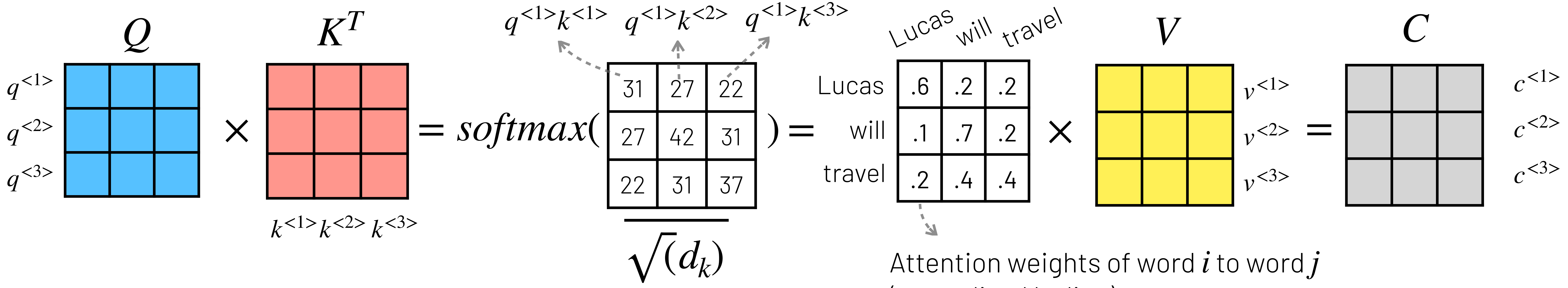

$$C = Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{(d_k)}})V$$



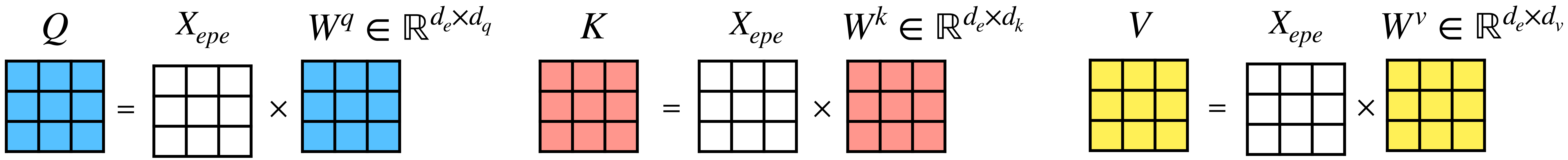

UFV

# Self-Attention

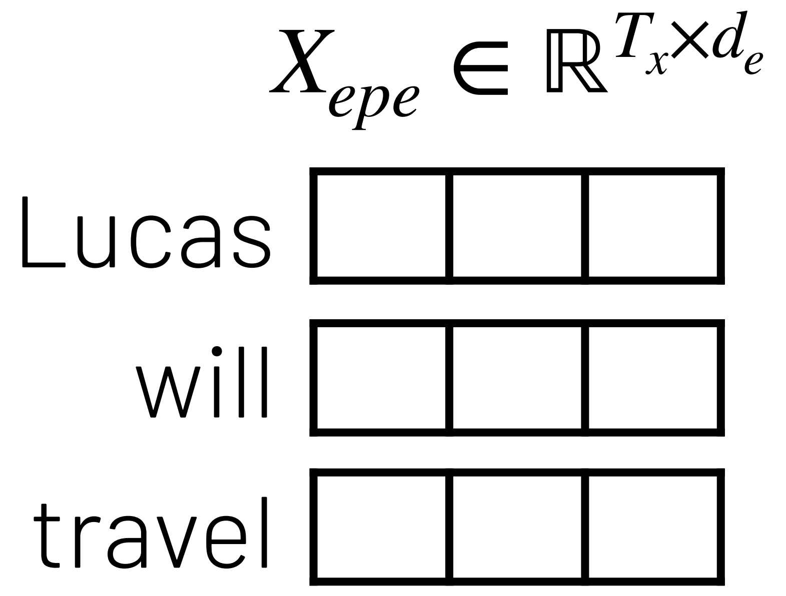
$$C = Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{(d_k)}}\right)V$$




Attention weights of word $i$ to word $j$
(normalized by line)




The attention layer receives the embedded sequence $X_{epe}$ as input




$d_e = d_q = d_k = d_v = 3$

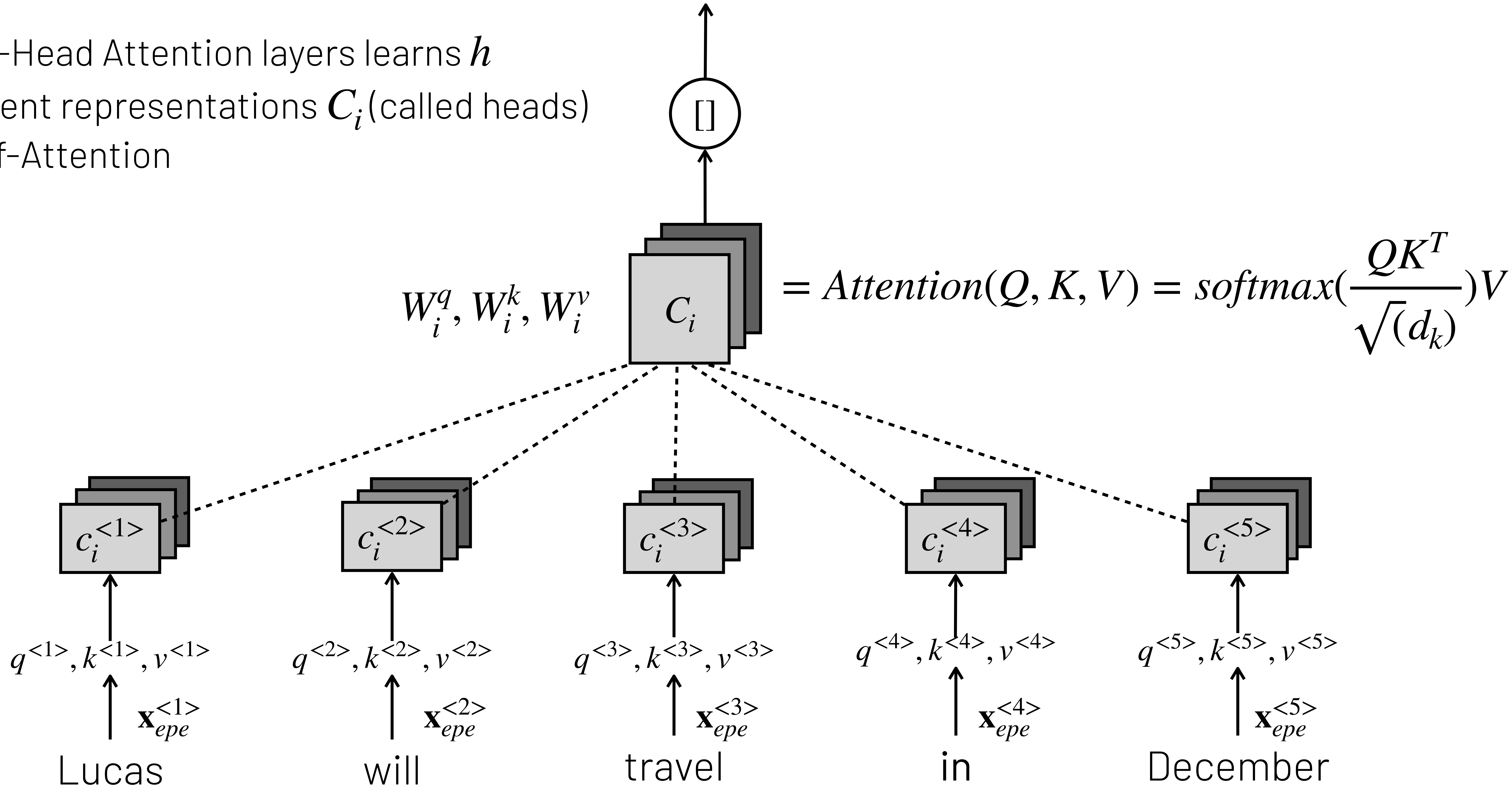
The sizes of key and query have to be the same. But embeddings and value typically also have same sizes.

UFV

# Multi-Head Attention

$$Multihead(Q, K, V) = Concat(C_1, C_2, \ldots, C_h)W^o$$

The Multi-Head Attention layers learns $h$ independent representations $C_i$ (called heads) using Self-Attention



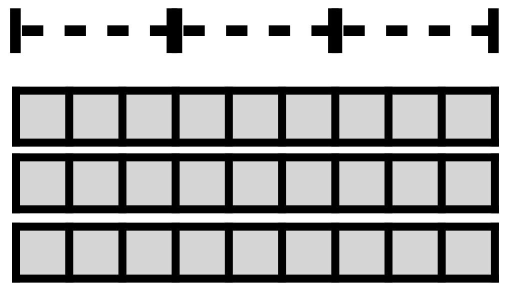

UFV

# Multi-Head Attention

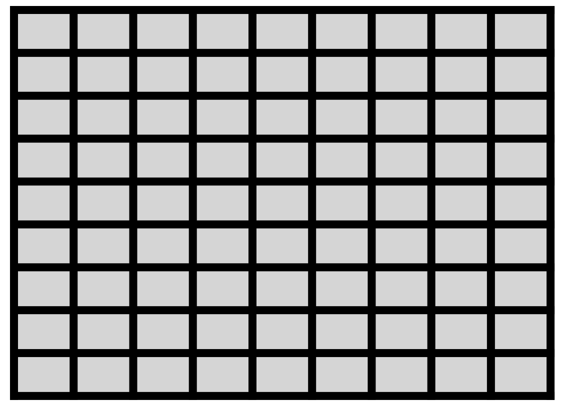
$$Multihead(Q, K, V) = Concat(C_1, C_2, C_3)W^o$$

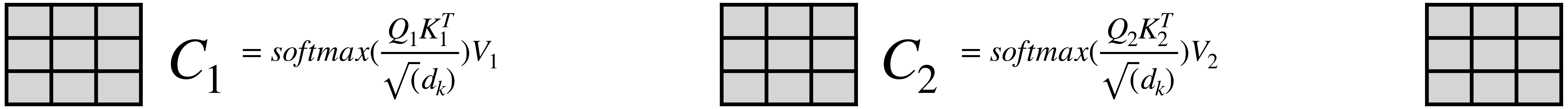
$C_1 \quad C_2 \quad C_3$

$W^o \in \mathbb{R}^{d_{model} \times d_{model}}$



$\times \quad = \quad C \in \mathbb{R}^{T_x \times d_{model}}$



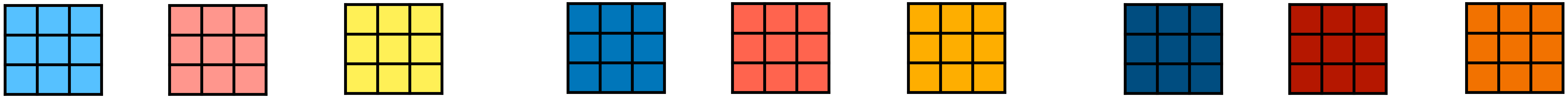


$C_1 = softmax(\frac{Q_1 K_1^T}{\sqrt(d_k)})V_1$

$C_2 = softmax(\frac{Q_2 K_2^T}{\sqrt(d_k)})V_2$

$C_3 = softmax(\frac{Q_3 K_3^T}{\sqrt(d_k)})V_3$

$Q_1 = X_{epe}W_1^q \quad K_1 = X_{epe}W_1^k \quad V_1 = X_{epe}W_1^v \quad Q_2 = X_{epe}W_2^q \quad K_2 = X_{epe}W_2^k \quad V_2 = X_{epe}W_2^v \quad Q_3 = X_{epe}W_3^q \quad K_3 = X_{epe}W_3^k \quad V_3 = X_{epe}W_3^v$

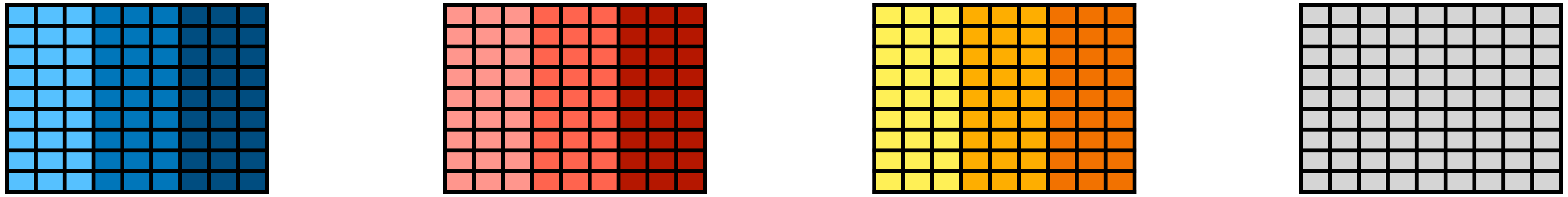


Number of heads

$h = 3$

$d_k = \frac{d_{model}}{h} = 3$

$W^q \in \mathbb{R}^{d_{model} \times d_{model}} \quad W^k \in \mathbb{R}^{d_{model} \times d_{model}} \quad W^v \in \mathbb{R}^{d_{model} \times d_{model}} \quad W^o \in \mathbb{R}^{d_{model} \times d_{model}}$



$W_1^q \quad W_2^q \quad W_3^q \qquad W_1^k \quad W_2^k \quad W_3^k \qquad W_1^v \quad W_2^v \quad W_v^k$

$T_x = 3$

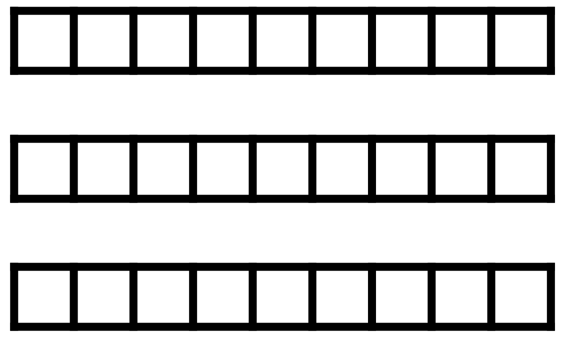
(sequence length)

$d_{model} = 9$

Lucas

will

travel



$X_{epe} \in \mathbb{R}^{T_x \times d_{model}}$

UFV

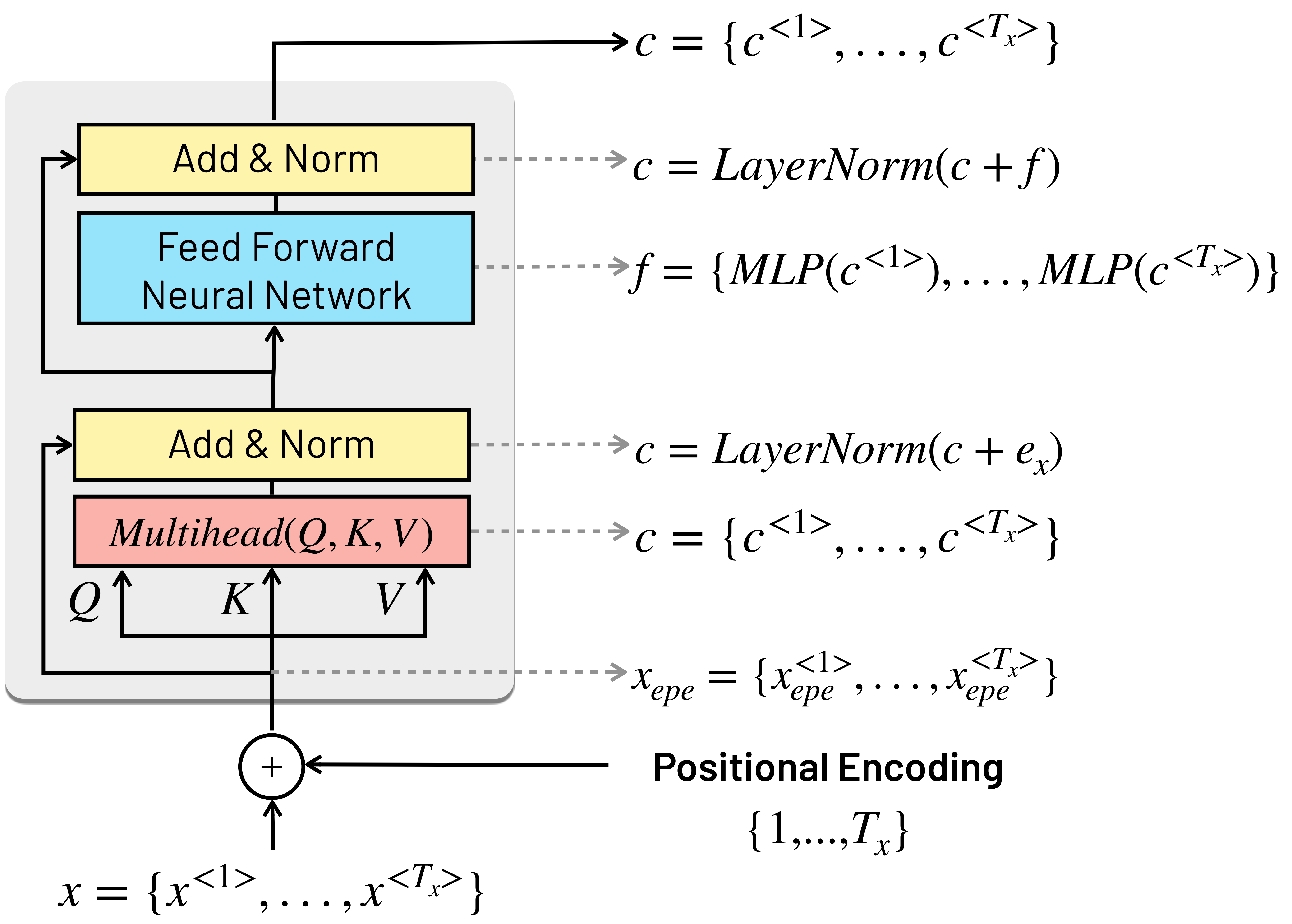
# Encoder

**Input**: a sequence $x = \{x^{<1>}, \ldots, x^{<T_x>}\}$

**Ouput**: a contextual representation $C = \{c^{<1>}, \ldots, c^{<T_x>}\}$ of $x$

The encoder applies a ***Multihead* Layer** followed by a ***Feed Forward Neural Network*** (MLP).

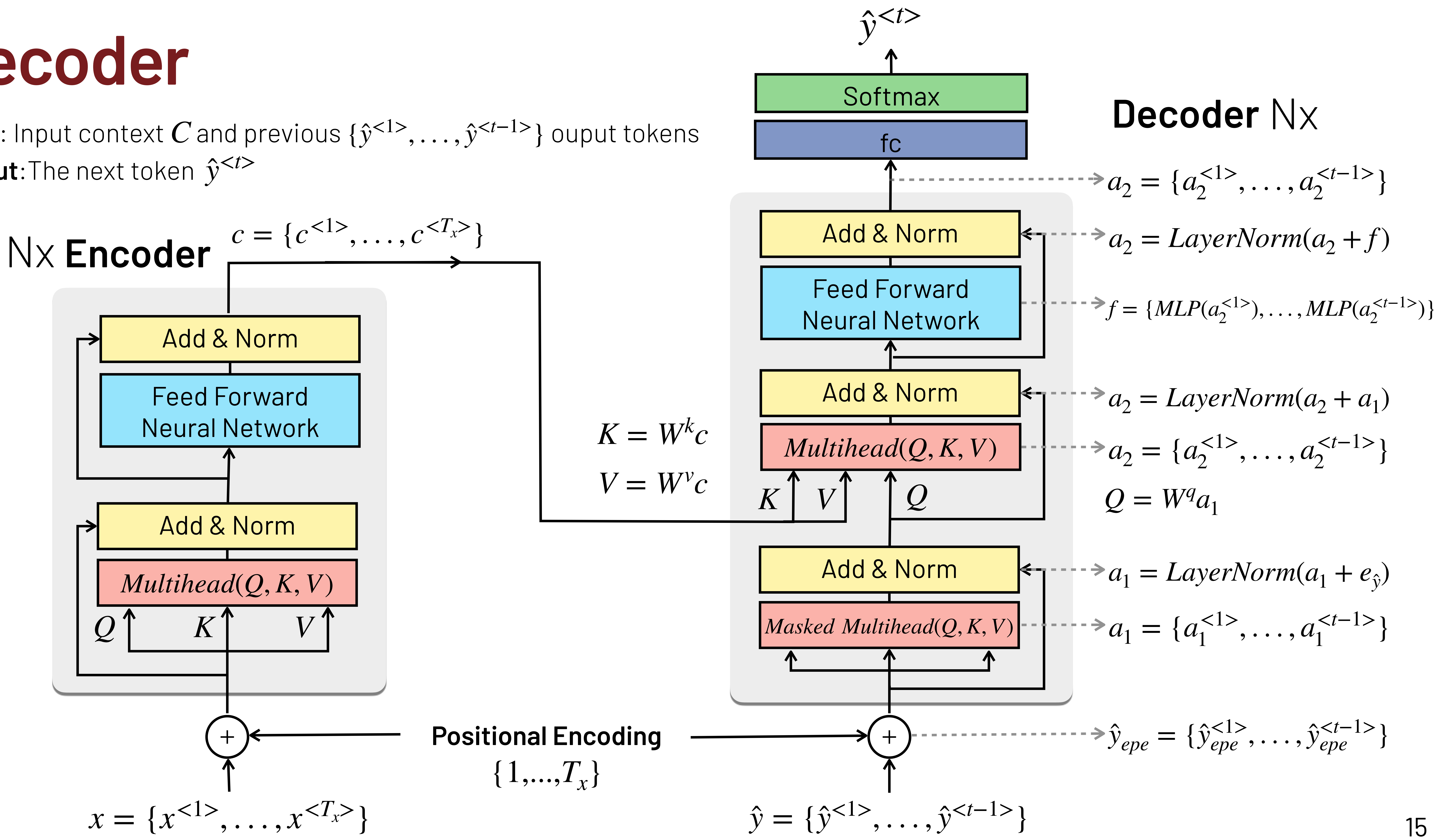
Both are normalized with Layer Norm (***Norm***) and conneced with a residual connection (***Add***)



# Decoder

**Input**: Input context $C$ and previous $\{\hat{y}^{<1>},...,\hat{y}^{<t-1>}\}$ ouput tokens

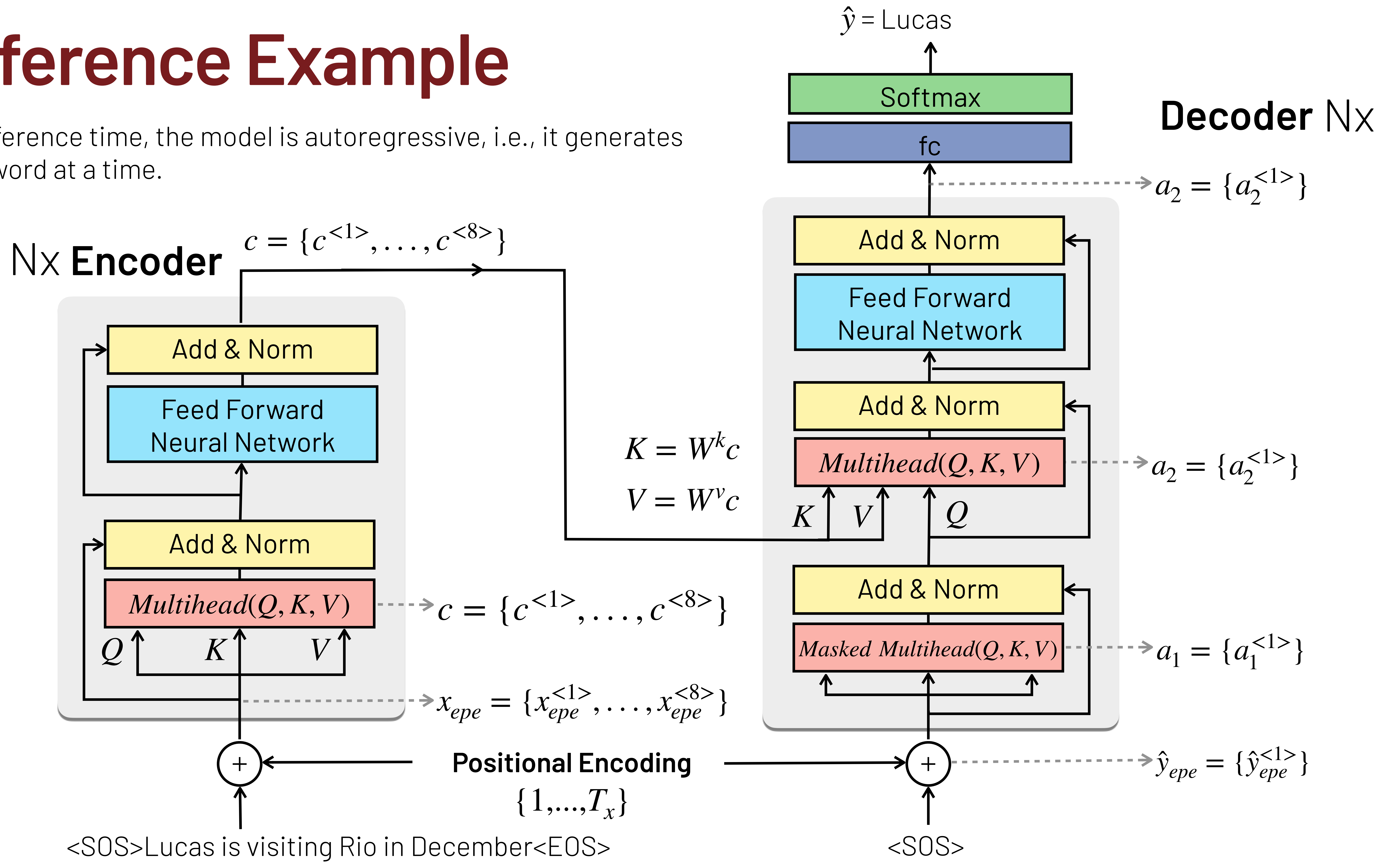
**Output**: The next token $\hat{y}^{<t>}$




UFV

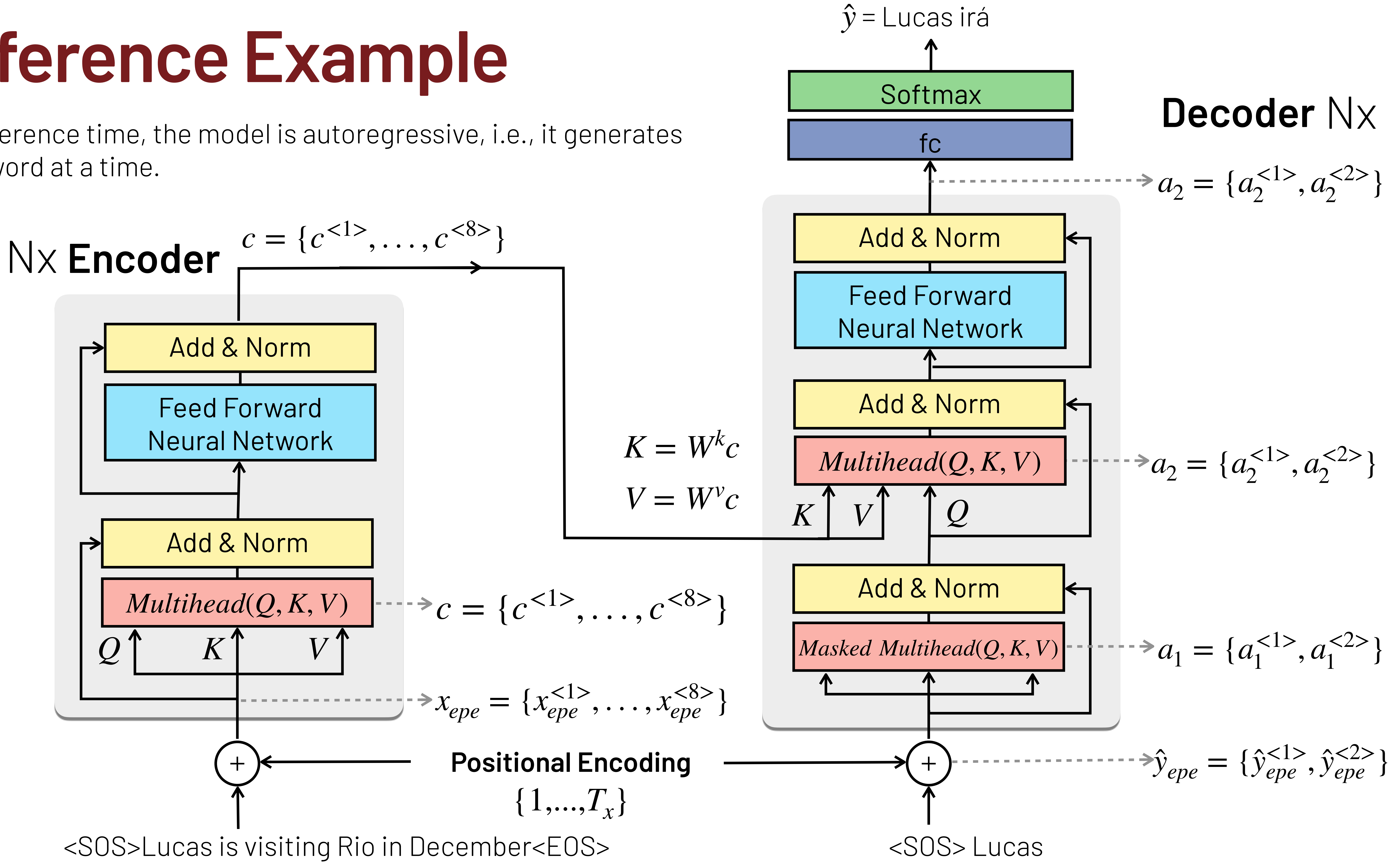
# Inference Example

At inference time, the model is autoregressive, i.e., it generates one word at a time.



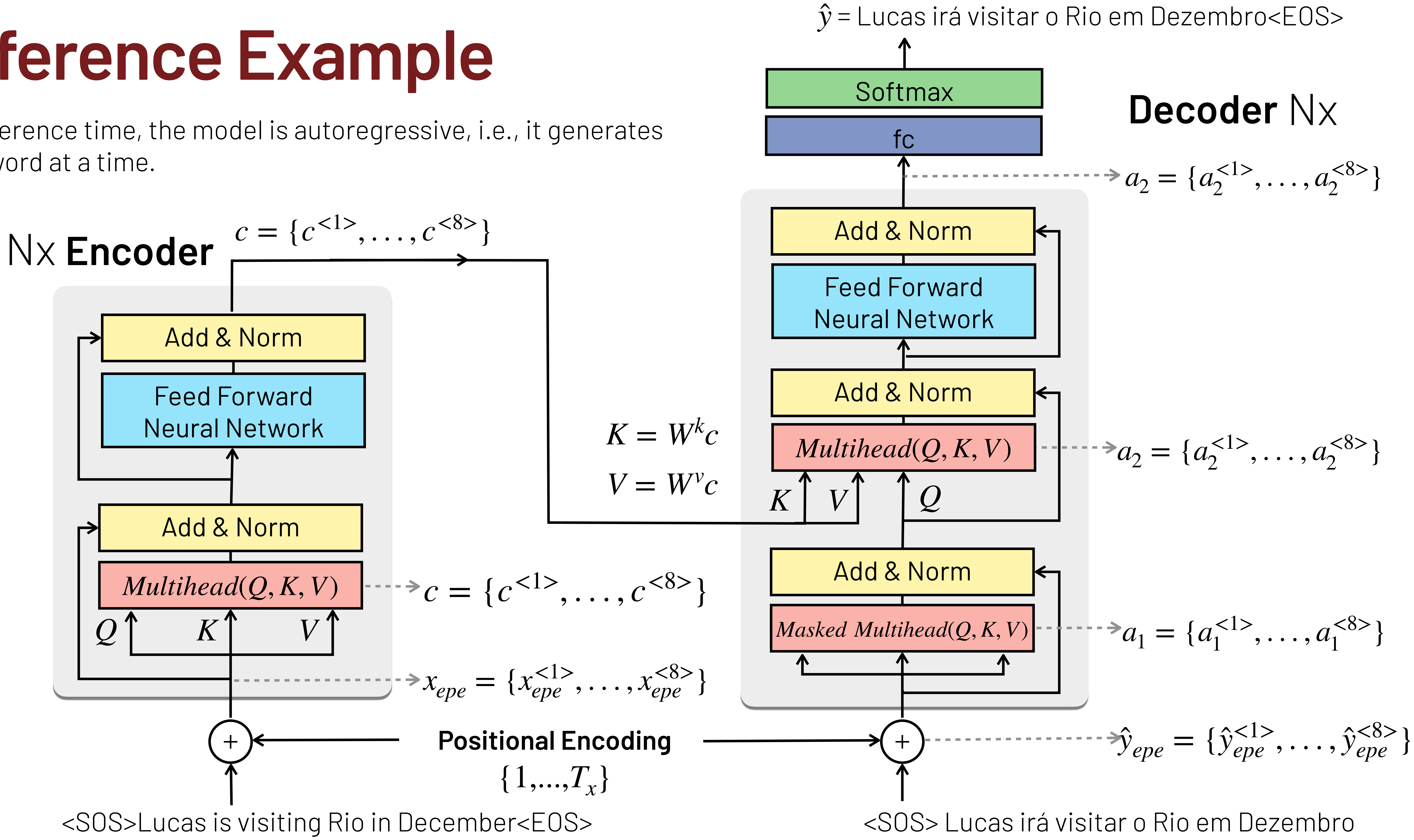
# Inference Example

At inference time, the model is autoregressive, i.e., it generates one word at a time.



# Inference Example

At inference time, the model is autoregressive, i.e., it generates one word at a time.



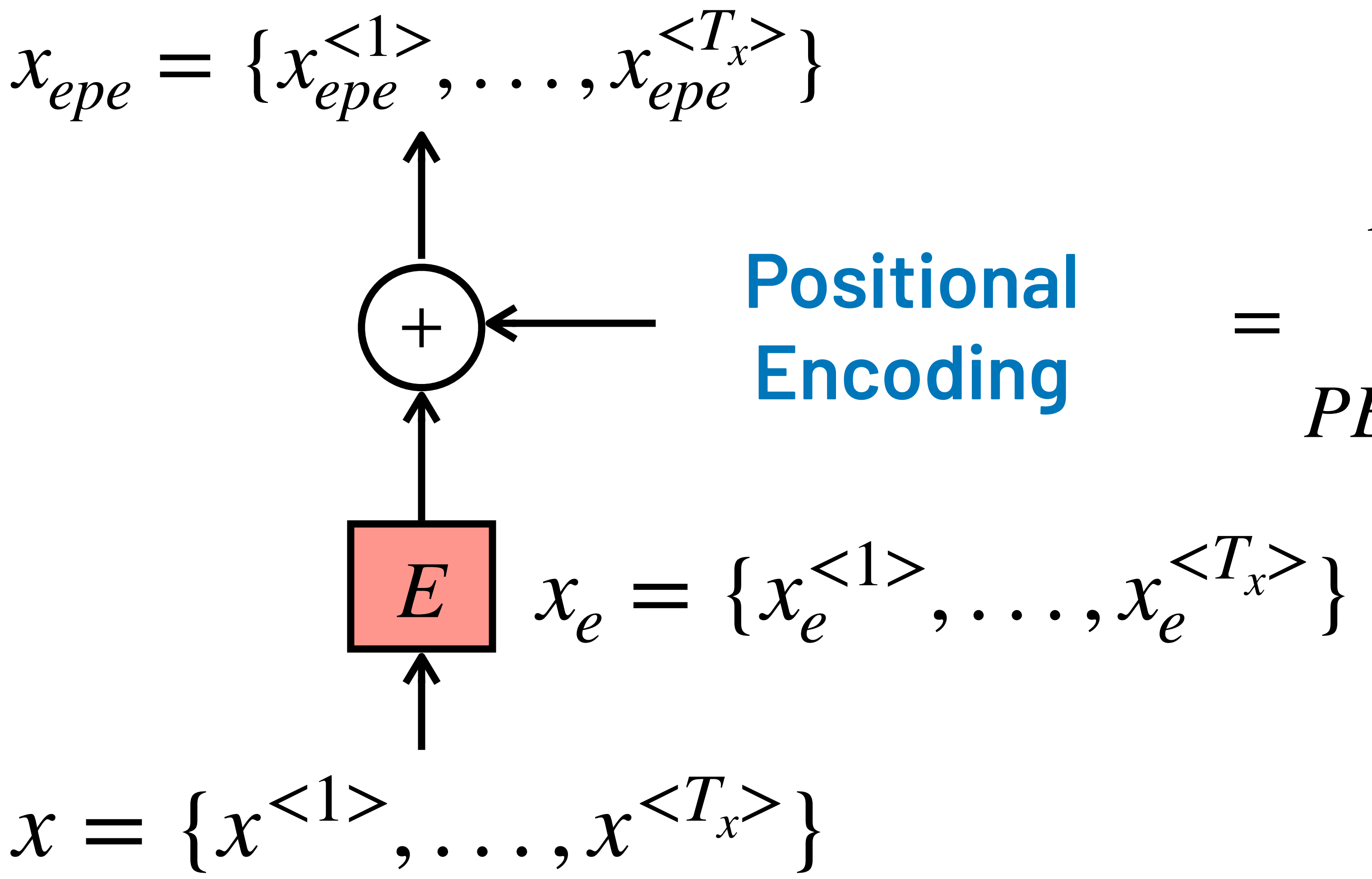

UFV

# Positional Encoding

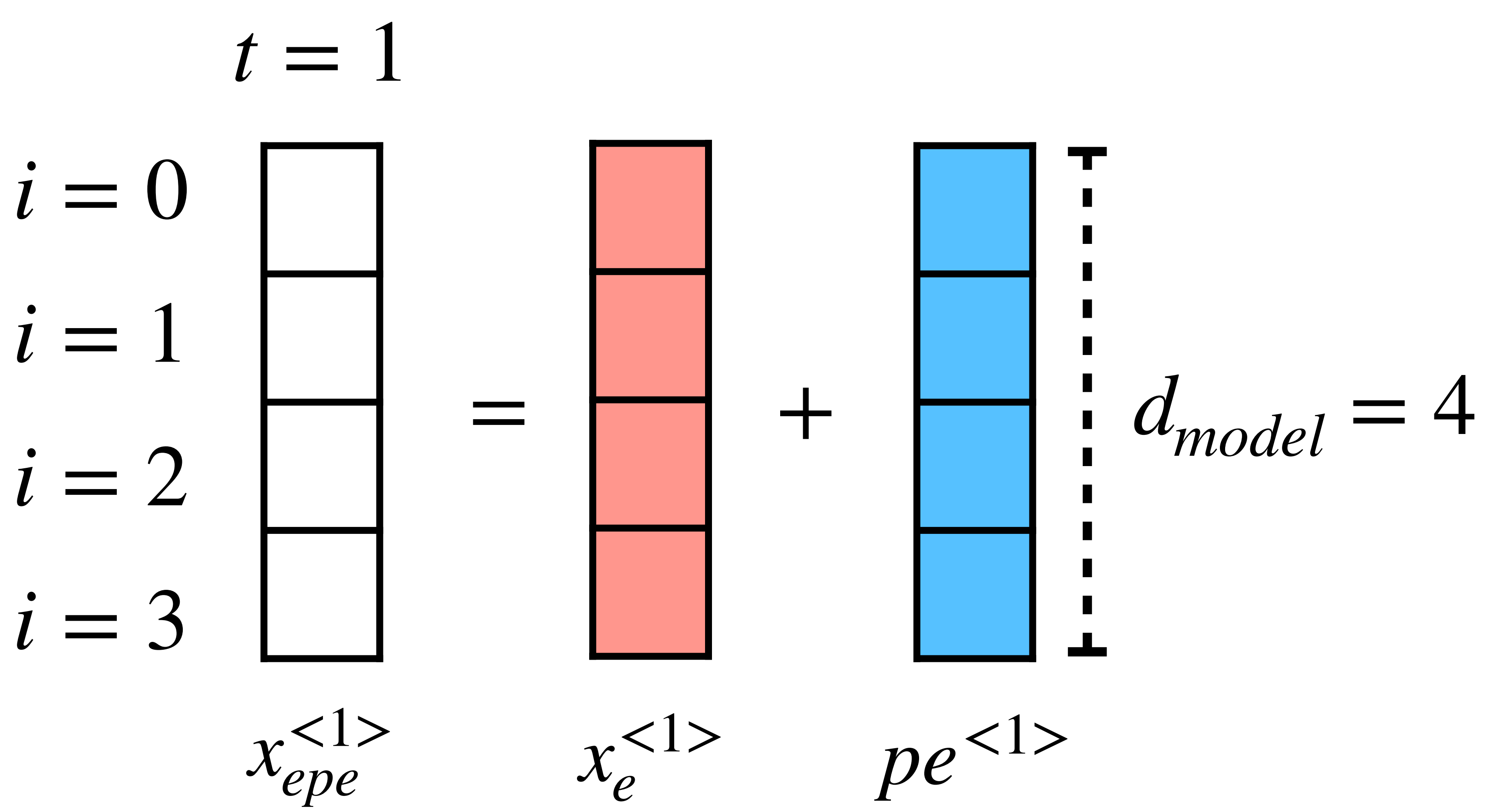
The self-attention mechanism does not consider the position of the words.

$$\boxed{C} = Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{(d_k)}})V$$

To add this information to the learned contextual representation $C$, both encoder and decoder add  an positional information to each element $x^{<t>}$ of the input $x = \{x^{<1>}, \ldots, x^{<T_x>}\}$




$$= \begin{cases} PE_{(t,2i)} = sin(\frac{t}{10000^{\frac{2i}{d}}}) \\ PE_{(t,2i+1)} = cos(\frac{t}{10000^{\frac{2i}{d}}}) \end{cases}$$




UFV

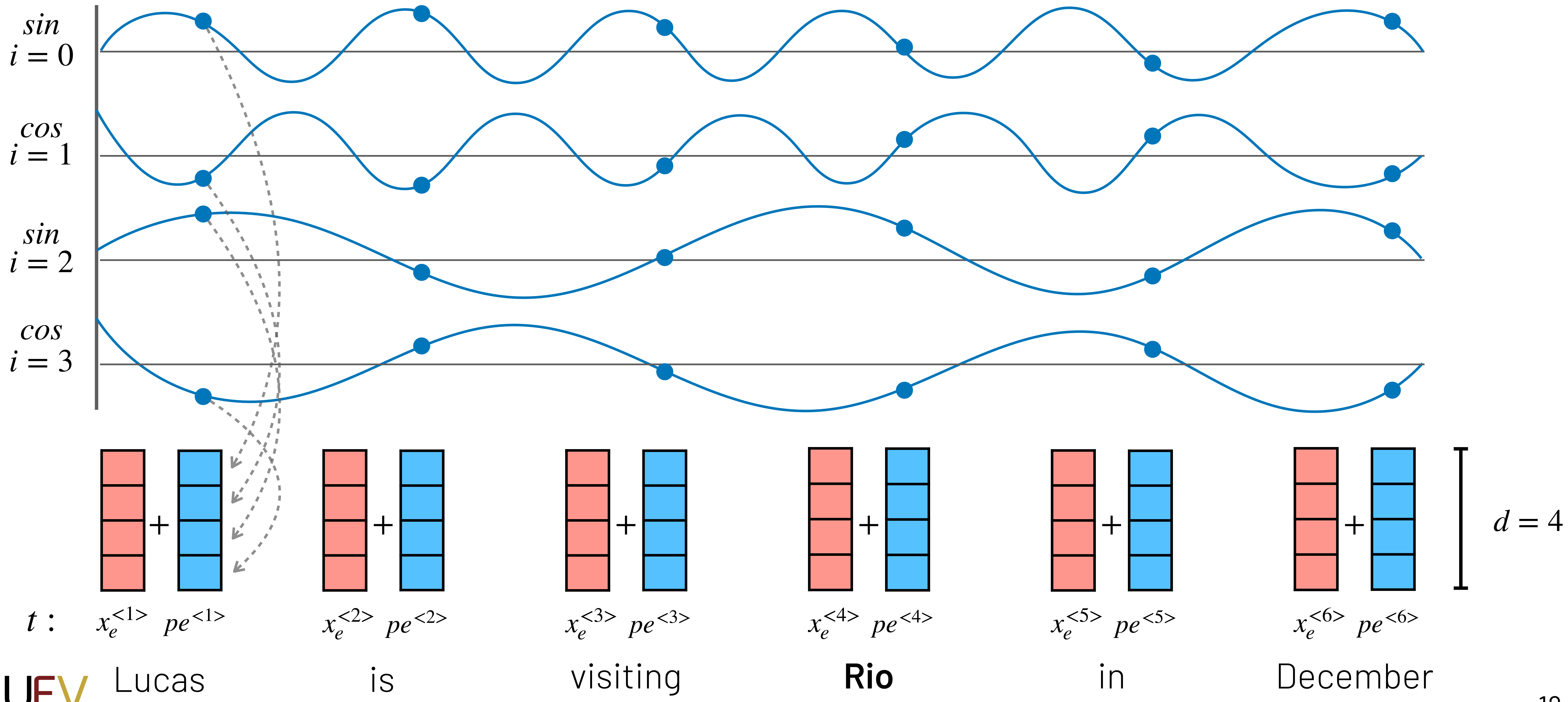
# Positional Encoding

if $i$ is even

$$PE_{(t,2i)} = sin(\frac{t}{10000^{\frac{2i}{d}}})$$

if $i$ is odd

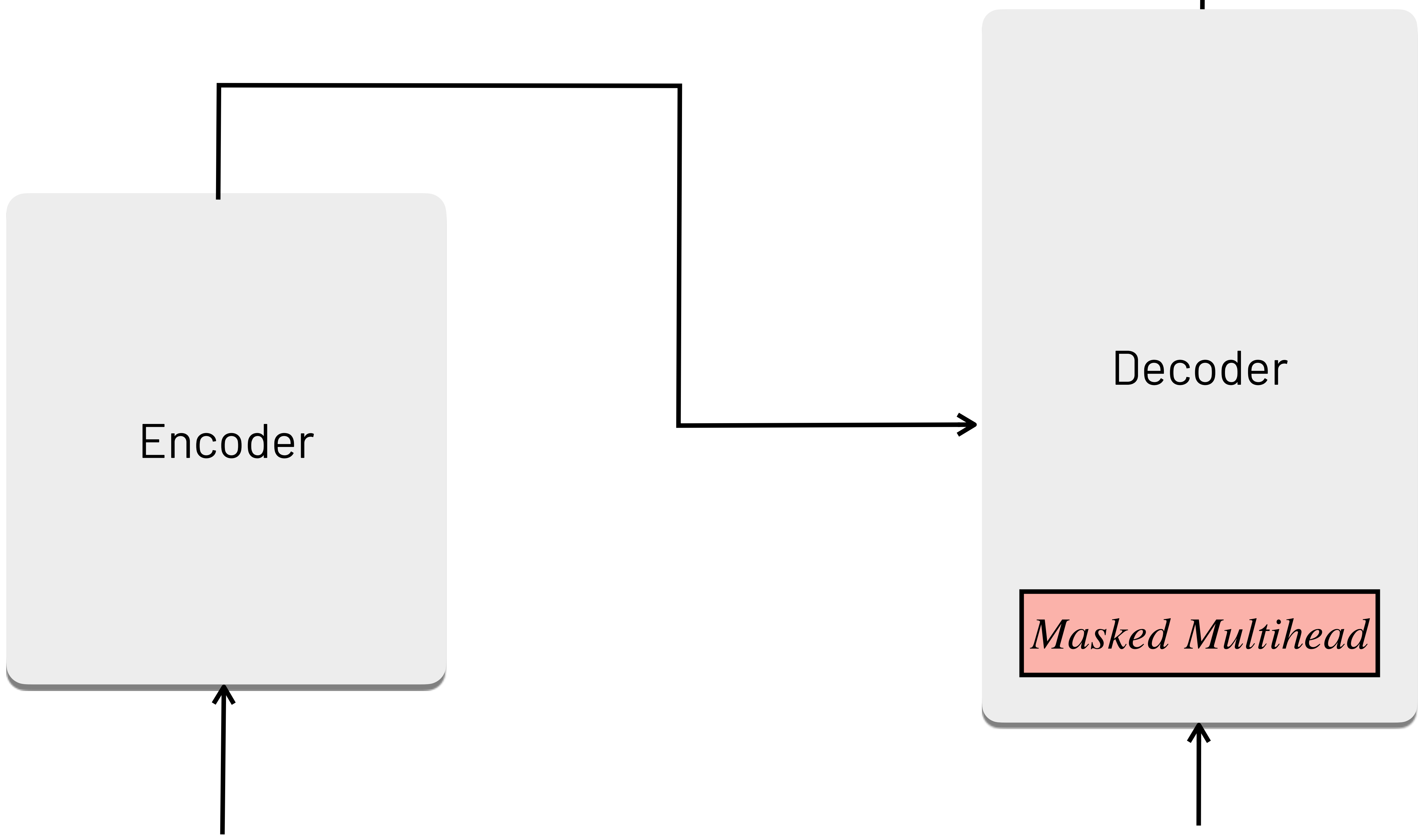
$$PE_{(t,2i+1)} = cos(\frac{t}{10000^{\frac{2i}{d}}})$$



# Training with Masked Multi-head Attention

$\hat{y}^{<1>}$ $\hat{y}^{<2>}$ $\hat{y}^{<3>}$ $\hat{y}^{<4>}$ $\hat{y}^{<5>}$

$\hat{y}$ = Lucas irá visitar o ____




- During training, the model **does not** produce $\hat{y}$ one token at at time;
- Instead, the output $\hat{y}$ is produced once, in parallel with vectorization.
- Thus, the full target sentence $y = \{y^{<1>}, \ldots, y^{<T_y>}\}$ is given as input to the Decoder, which uses a **mask** to predict $\hat{y}^{<t>}$, so it does not attent to the future tokens $(\hat{y}^{<t'>}, t' > t)$. For example, for $t = 5$:

$x$ = <SOS>Lucas is visiting Rio in December <EOS>

$y$ = <SOS> Lucas irá visitar o Rio em Dezembro

$y^{<1>}$ $y^{<2>}$ $y^{<3>}$ $y^{<4>}$ $y^{<5>}$ $y^{<6>}$ $y^{<7>}$ $y^{<8>}$

# Training with Masked Multi-head Attention

Attention weights (normalized by row) of work $i$ to word $j$ after mask application

$$Q \times K^T = \frac{\begin{bmatrix}31 & 27 & 22\\ 27 & 42 & 31\\ 22 & 31 & 37\end{bmatrix}}{\sqrt{(d_k)}} + \underbrace{\begin{bmatrix}0 & -inf & -inf\\ 0 & 0 & -inf\\ 0 & 0 & 0\end{bmatrix}}_{\textbf{Mask}} = softmax\left(\frac{\begin{bmatrix}31 & -inf & -inf\\ 27 & 42 & -inf\\ 22 & 31 & 37\end{bmatrix}}{\sqrt{(d_k)}}\right) = \begin{bmatrix}.1 & .0 & .0\\ .3 & .7 & .0\\ .3 & .3 & .4\end{bmatrix}$$

$$Q = X_{epe} \times W^q \in \mathbb{R}^{d_e \times d_q} \qquad K = X_{epe} \times W^k \in \mathbb{R}^{d_e \times d_k} \qquad V = X_{epe} \times W^v \in \mathbb{R}^{d_e \times d_v}$$

The attention layer receives the embedded sequence $X_{epe}$ as input

$X_{epe} \in \mathbb{R}^{T_x \times d_e}$

Lucas

will

travel

$$d_e = d_q = d_k = d_v = 3$$

The sizes of key and query have to be the same. But embeddings and value typically also have same sizes.

UFV

# Next Lecture

**L17**: Transformers (Part II)

Case studies of transformers: BERT and GPT