DeepIntoDeep

Transformer

발표자: 김지영

Transformer

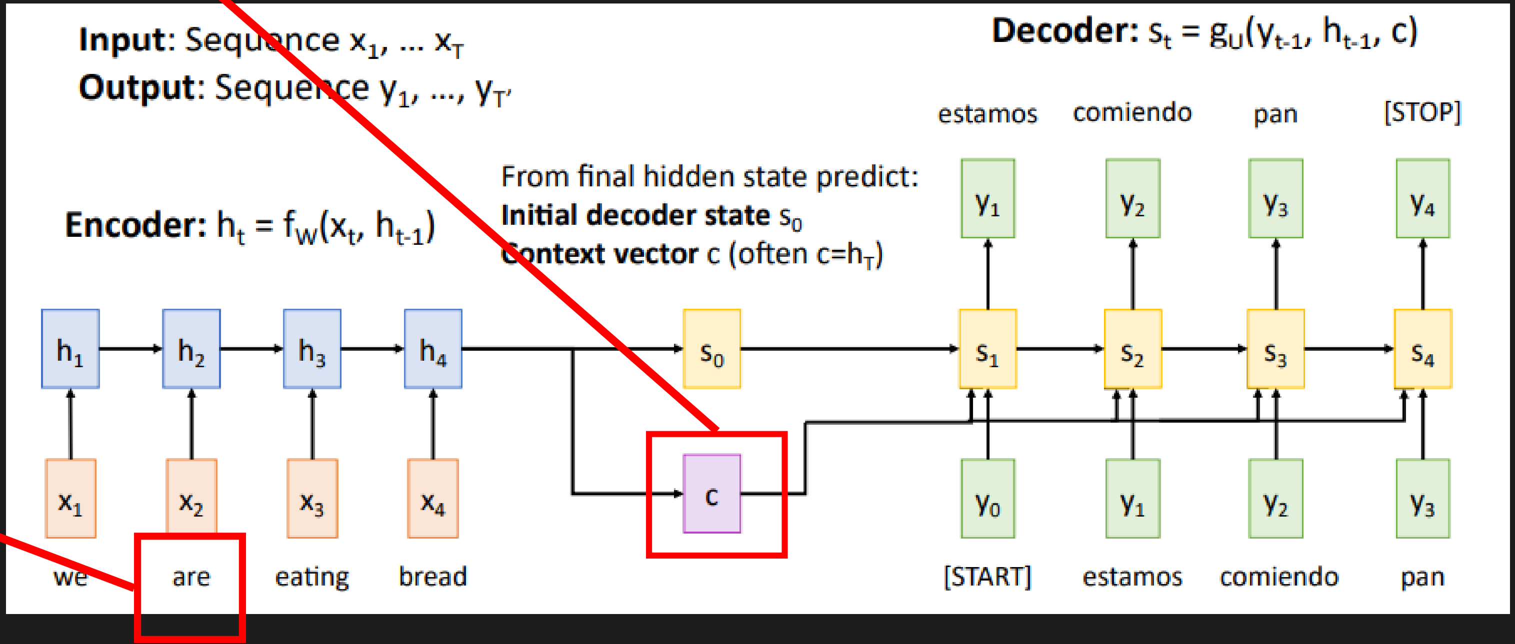
김지영

Artificial Intelligence in Korea University(AIKU)

Department of Computer Science and Engineering, Korea University

Recap : Issues with recurrent models

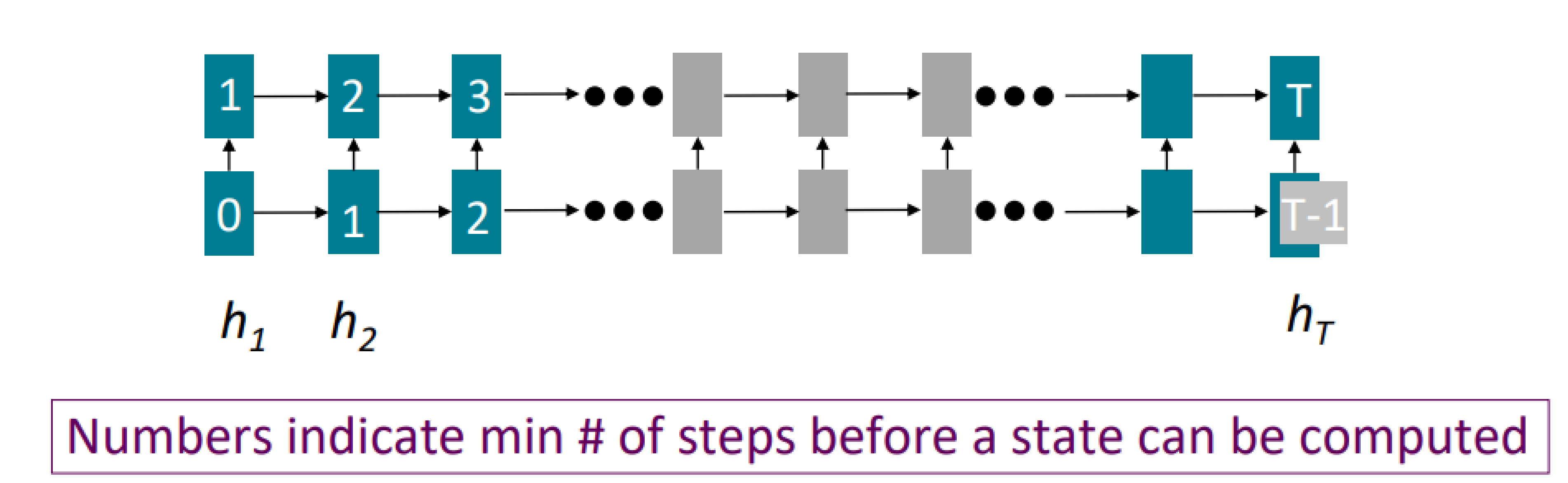
1. Linear interaction distance
   - O(sequence length) steps for distant word pairs to interact → Gradient vanishing
   - Information bottleneck



Info of **are**
has gone through O(4) many layers

Recap : Issues with recurrent models

2. Lack of parallelizability
   - Future RNN hidden states can't be computed in full before past RNN hidden states have been computed

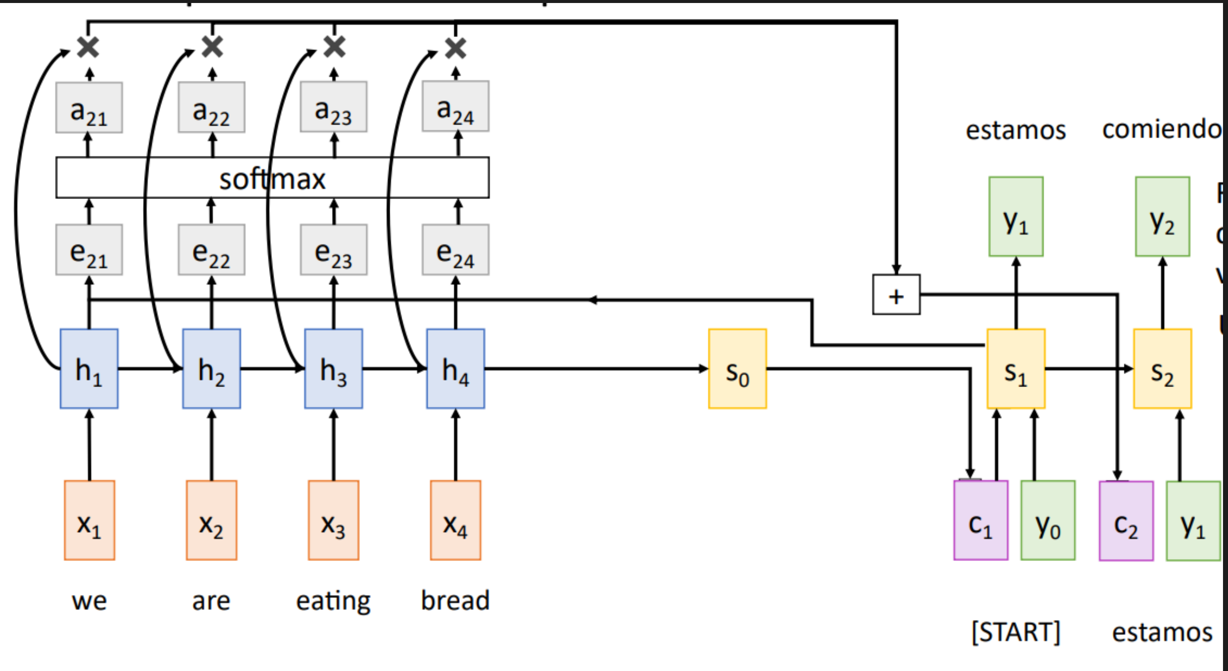


Recap : Attention

1. Linear interaction distance
   - O(sequence length) steps for distant word pairs to interact → Gradient vanishing
   - Information bottleneck

➡ Use **different** context vector in **each timestep** of decoder

Context vector **attends** to relevant part of input sequence



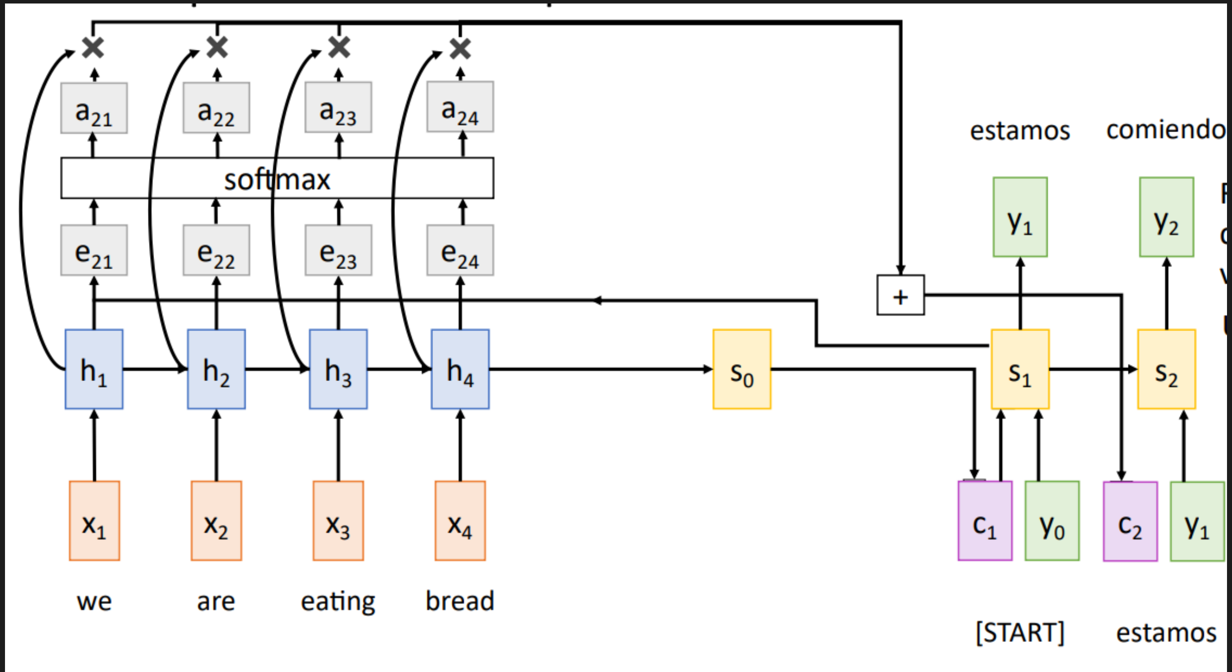
$$e_{t,i} = f_{att}(s_{t-1}, h_i)$$

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

Generalized attention

Changes:

- Use dot product for similarity
- Multiple **query** vectors
- Separate **key** and **value**



$e_{t,i} = f_{att}(\boxed{s_{t-1}}, \boxed{h_i})$

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

Query vectors Q

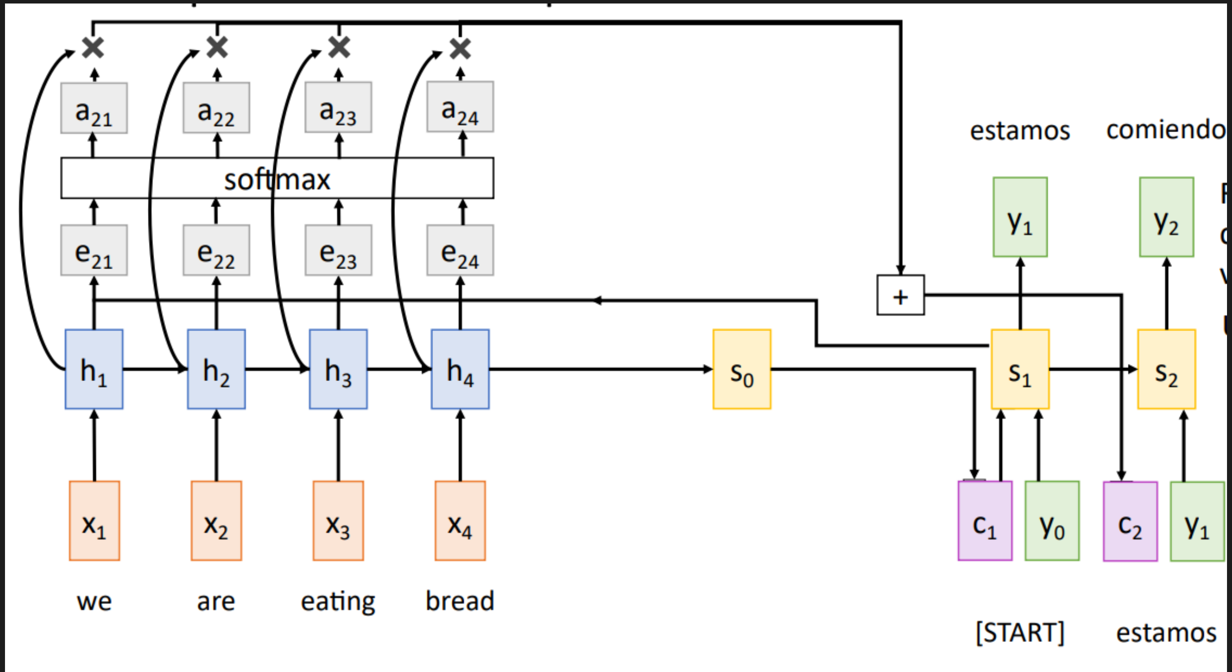
Input vectors X

Similarities: $E = Q\boxed{X^T}$

Attention weights: A = softmax(E, dim=1)

Output vectors: $Y = A\boxed{X}$

Attention Layer



$$e_{t,i} = f_{att}(\boxed{s_{t-1}}, \boxed{h_i})$$

Query vectors Q

Input vectors X

Key matrix

Value Matrix

Key vectors: $K = XW_K$

Value Vectors: $V = XW_V$

Similarities: $E = QK^T$

Output vectors: $Y = AV$

Attention as a soft, averaging lookup table

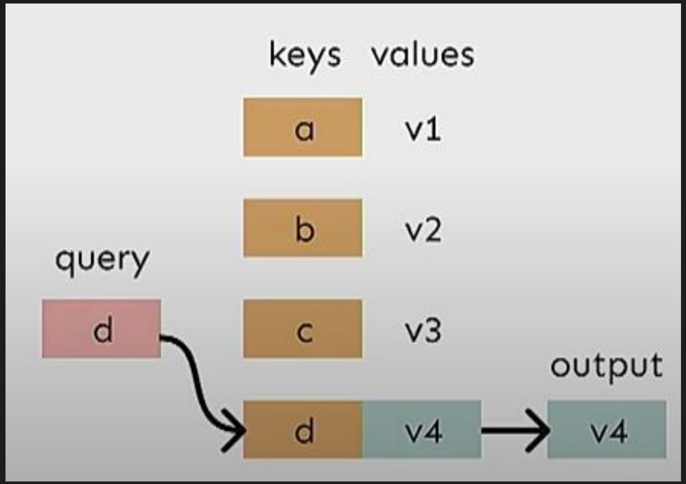
Query vectors Q
Input vectors X → Key matrix
Input vectors X → Value Matrix

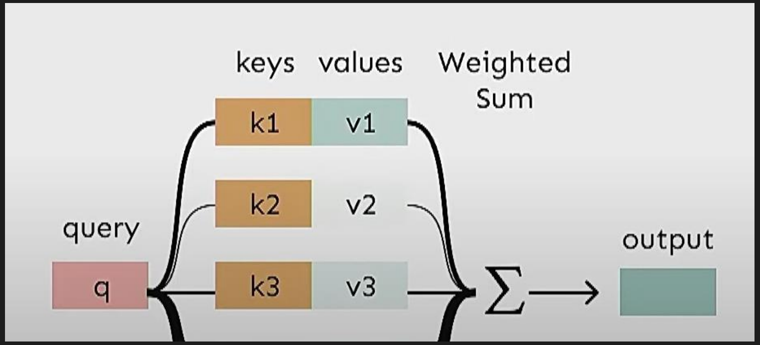
Key vectors: $K = XW_K$

Value Vectors: $V = XW_V$

Similarities: $E = QK^T$

Output vectors: $Y = AV$



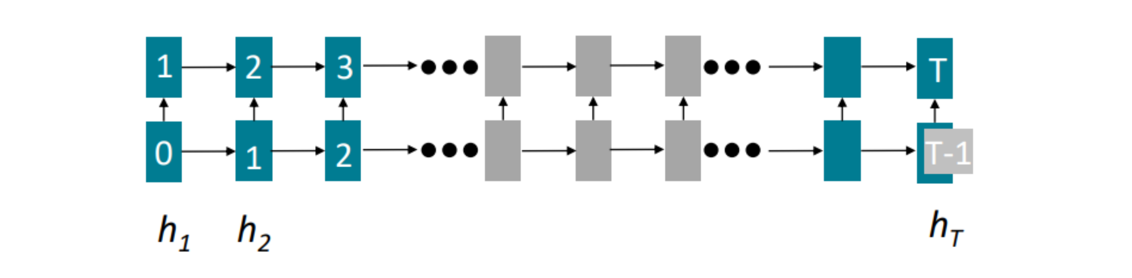


- Attention as performing fuzzy lookup in a key-value store
- Gives more flexibility in using input data

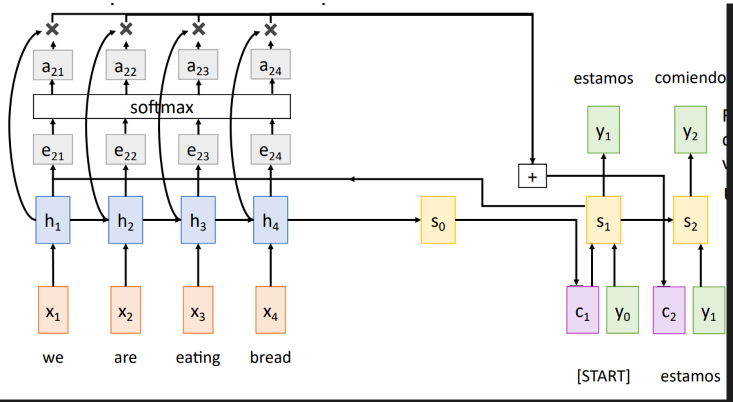
Attention layer with multiple vectors

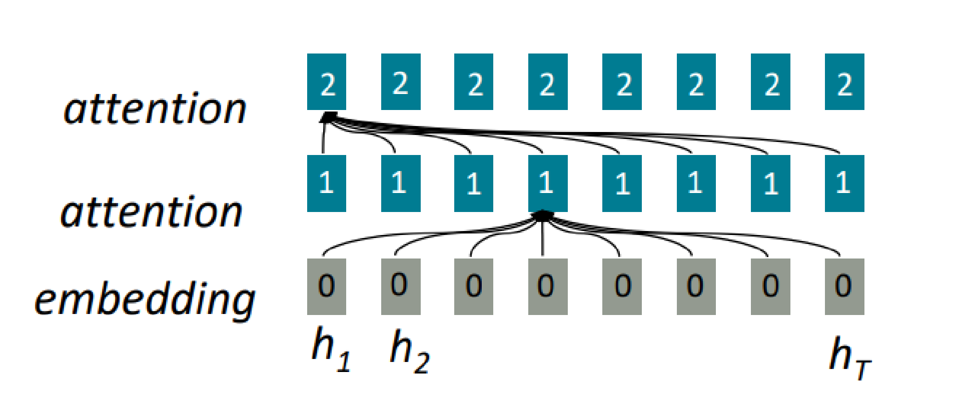
2. Lack of parallelizability

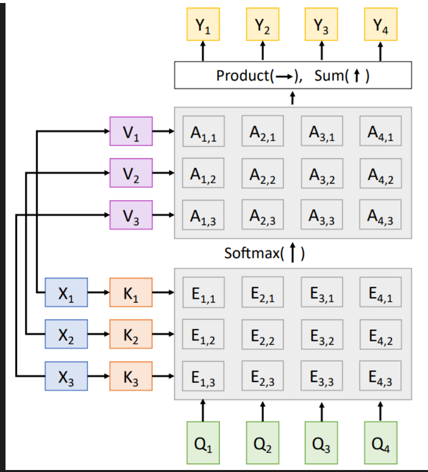
- Future RNN hidden states can't be computed in full before past RNN hidden states have been computed



Numbers indicate min # of steps before a state can be computed



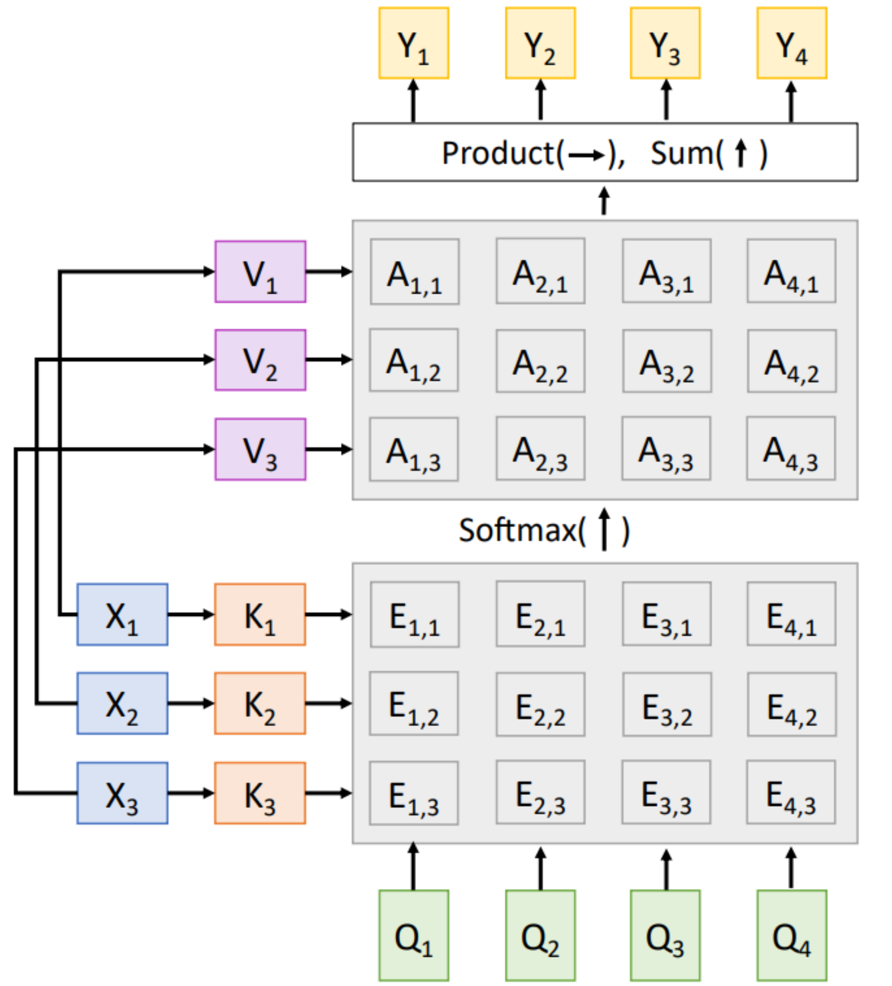




(Generalized) Attention layer

Changes:
- Use dot product for similarity
- Multiple query vectors
- Separate key and value

$Q_i \cdot K_j / \mathrm{sqrt}(D_Q)$



Example

"The Sleepy Child Reads A $\boxed{\text{Book}}$"

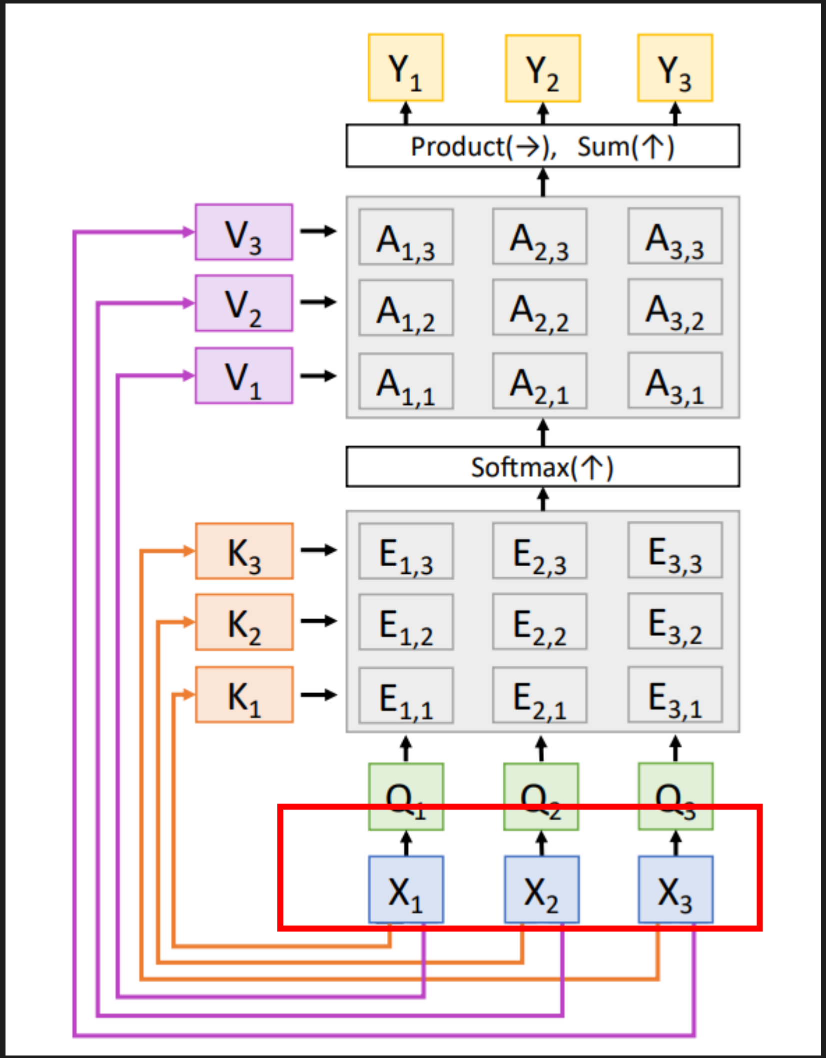
$\vec{q} = [0, 2, 1]$

Index	Embedding	Word
0	0,0,0	The
1	2,0,1	Sleepy
2	1,-1,-2	Child
3	2,3,1	Reads
4	-2,0,0	A
5	0,2,1	Book

$$\mathbf{K} = \begin{bmatrix} 0 & 2 & 1 & 2 & -2 & 0 \\ 0 & 0 & -1 & 3 & 0 & 2 \\ 0 & 1 & -2 & 1 & 0 & 1 \end{bmatrix}$$

$$\mathbf{V} = [0, -0.2, 0.3, 0.4, 0, 0.1]$$

Self Attention layer



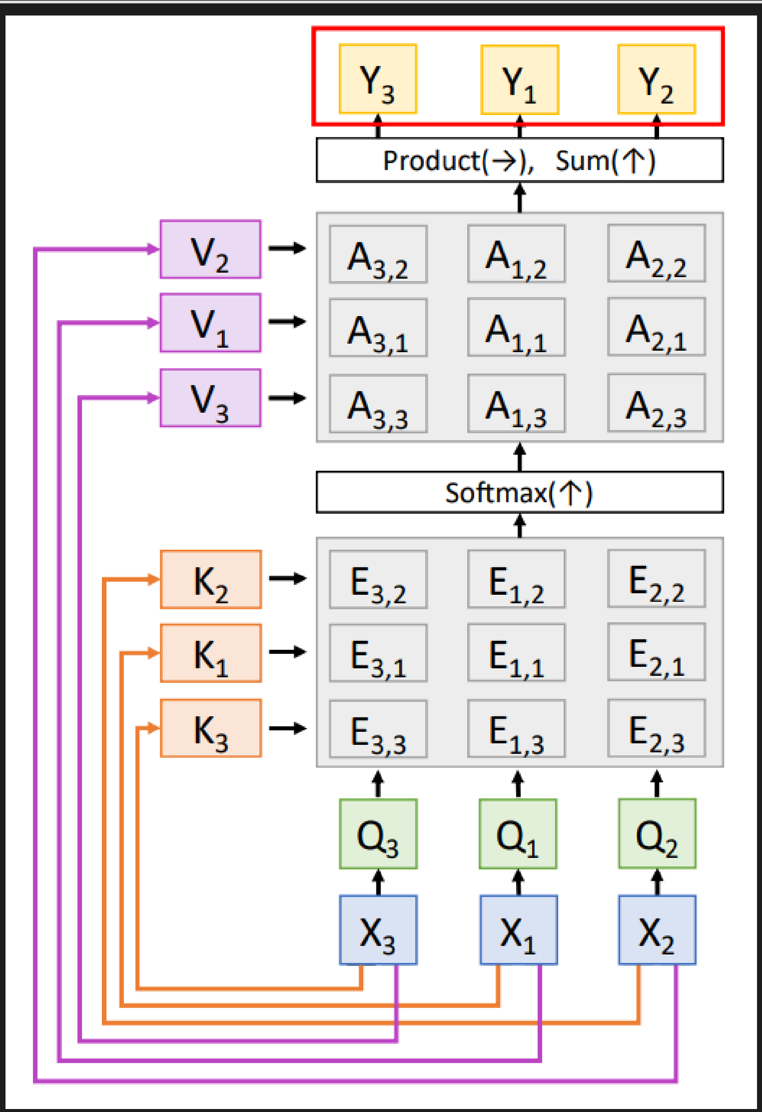
Issue#1 with SA – permutation equivariance

Definition 1.1 (g-invariant)**.** For given f and g, f is g-invariant if for all x,

$$f(g(x)) = f(x) \tag{1}$$

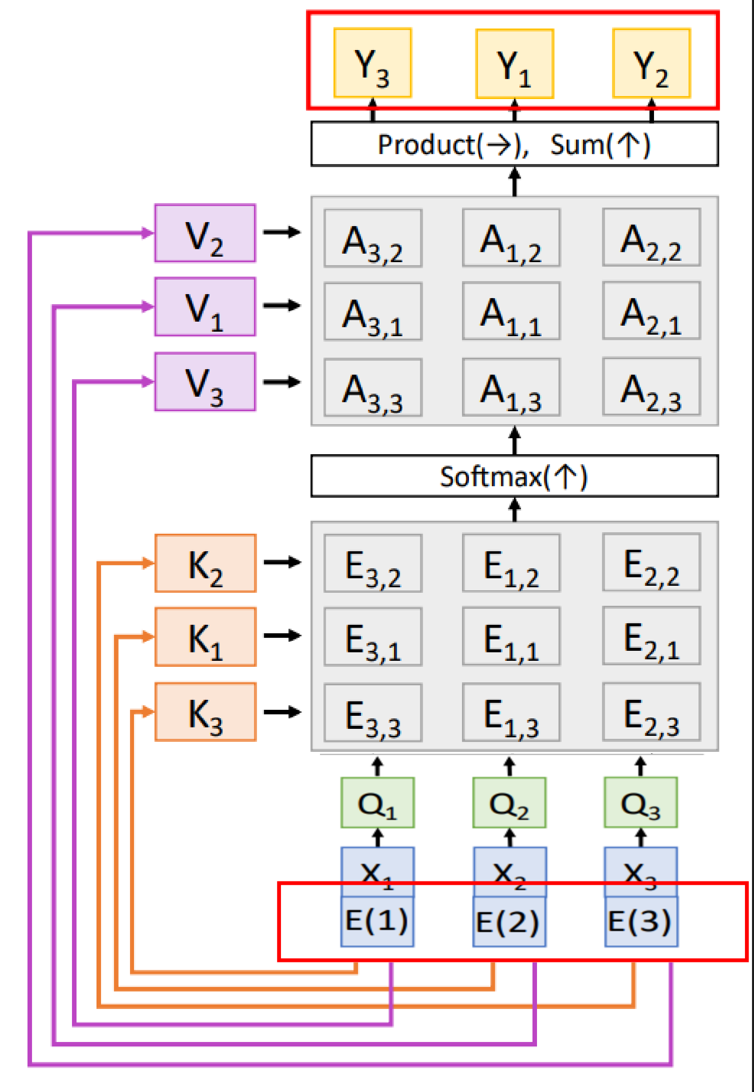
Definition 1.2 (g-equivariant)**.** For given f and g, f is g-equivariant if for all x,

$$f(g(x)) = g(f(x)) \tag{2}$$



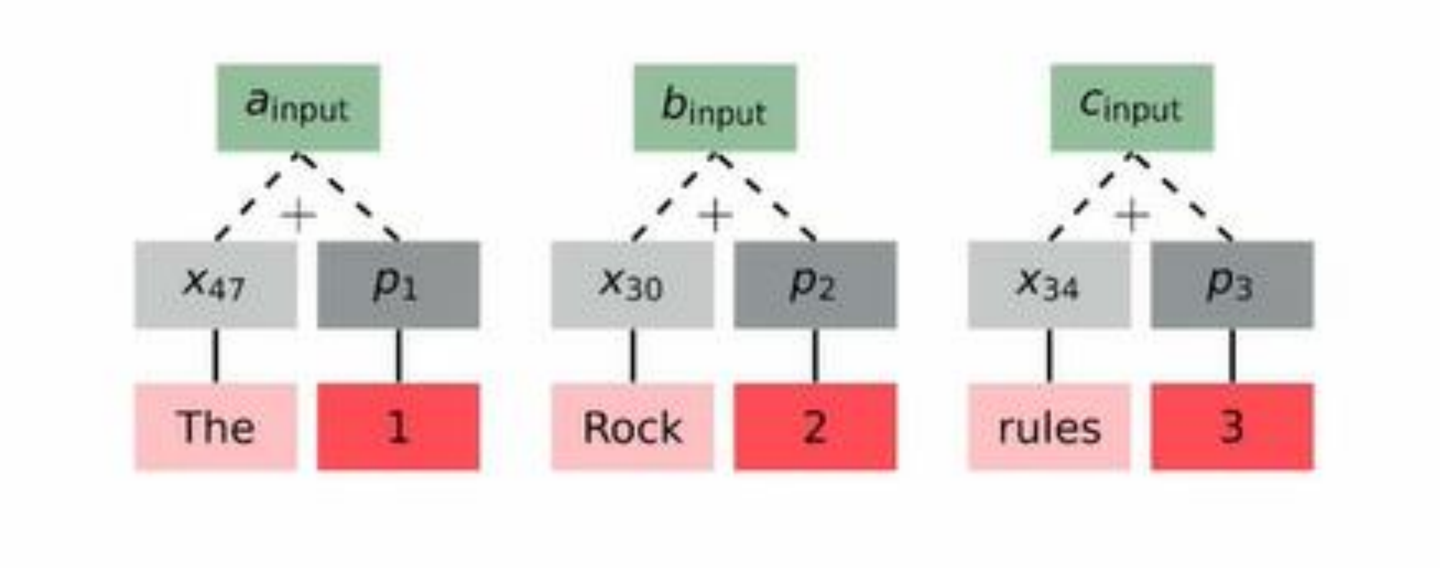
Self Attention layer – Positional encoding

$$\tilde{x}_i = x_i + p_i$$



Self Attention layer – Positional encoding

Absolute positional encoding



I walk my dog every day   every single day I walk my dog

Self Attention layer – Positional encoding

Frequency-based PE : Sinusoidal position representations

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

$$M.\begin{bmatrix} \sin(\omega_k.\,t) \\ \cos(\omega_k.\,t) \end{bmatrix} = \begin{bmatrix} \sin(\omega_k.\,(t+\phi)) \\ \cos(\omega_k.\,(t+\phi)) \end{bmatrix}$$

1. Periodicity
2. Constrained Values
3. Easy to extrapolate for longer sequences

Positional Encoding Matrix with d=4, n=100

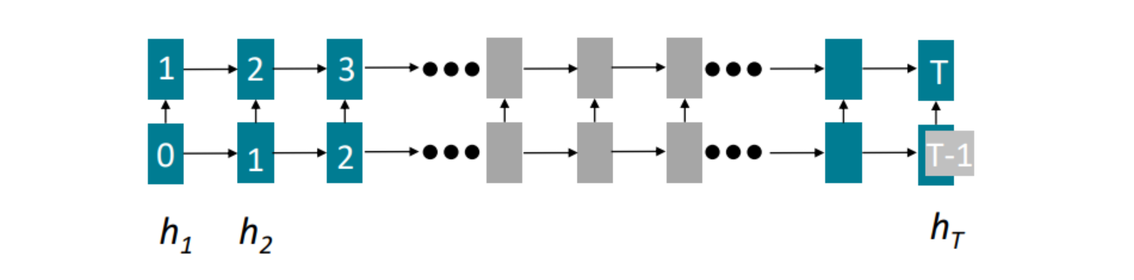
Sequence	Index of token, k	i=0	i=0	i=1	i=1
I	0	P_{00}=sin(0) = 0	P_{01}=cos(0) = 1	P_{02}=sin(0) = 0	P_{03}=cos(0) = 1
am	1	P_{10}=sin(1/1) = 0.84	P_{11}=cos(1/1) = 0.54	P_{12}=sin(1/10) = 0.10	P_{13}=cos(1/10) = 1.0
a	2	P_{20}=sin(2/1) = 0.91	P_{21}=cos(2/1) = -0.42	P_{22}=sin(2/10) = 0.20	P_{23}=cos(2/10) = 0.98
Robot	3	P_{30}=sin(3/1) = 0.14	P_{31}=cos(3/1) = -0.99	P_{32}=sin(3/10) = 0.30	P_{33}=cos(3/10) = 0.96

Positional Encoding Matrix for the sequence 'I am a robot'

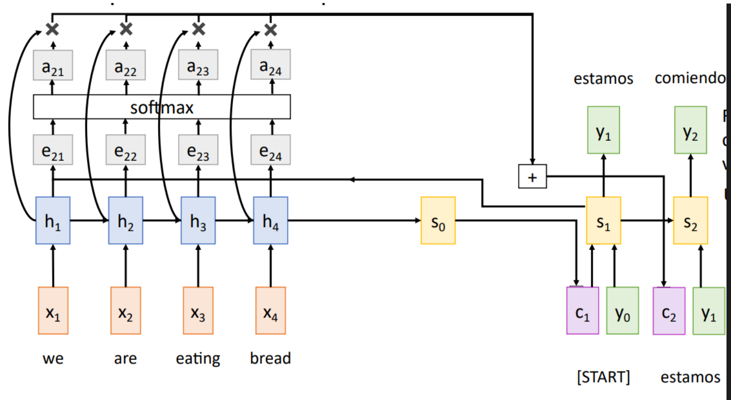
Issue#2 with SA – Look Ahead

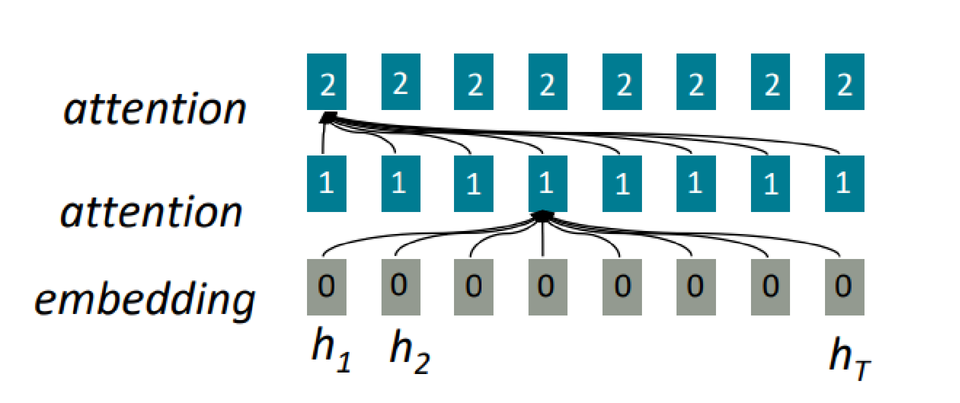
2. Lack of parallelizability

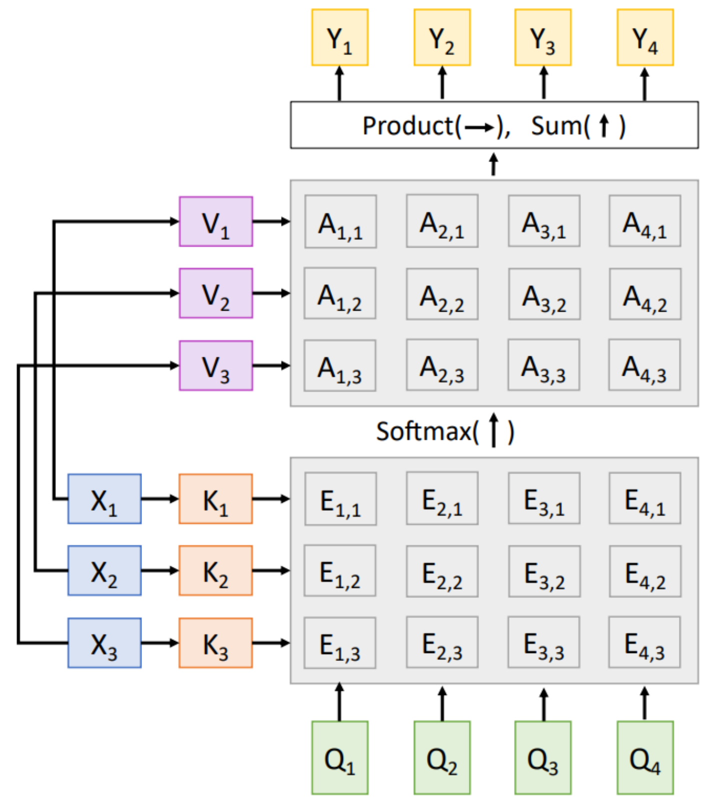
- Future RNN hidden states can't be computed in
  hidden states have been computed



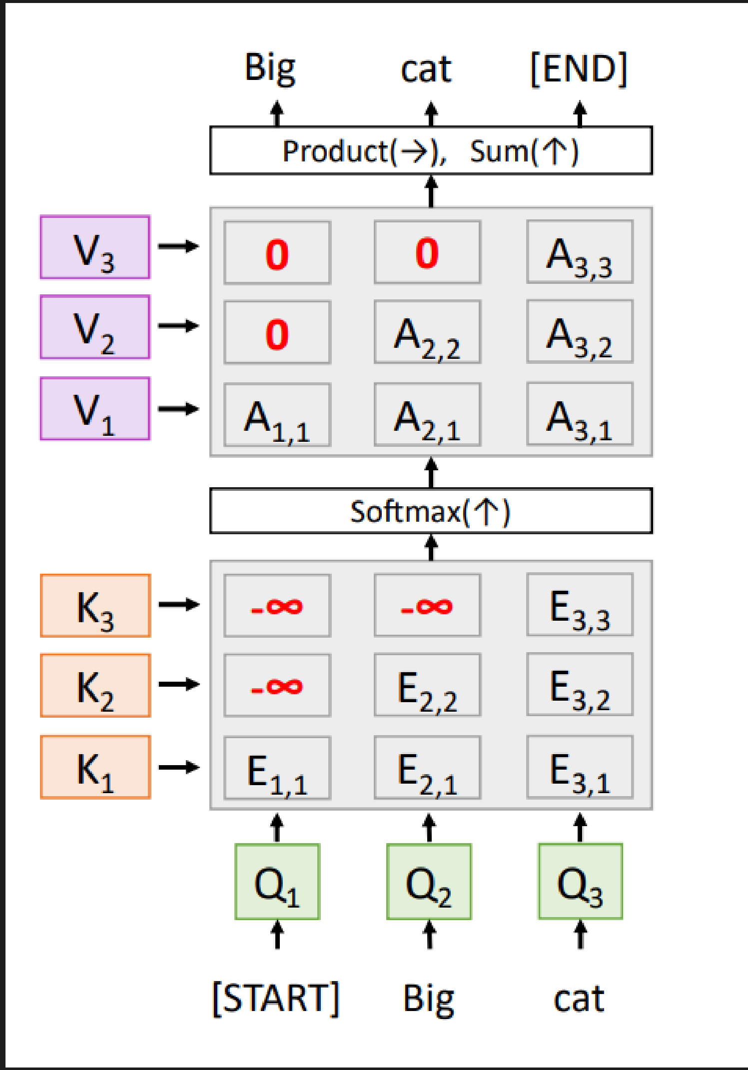
Numbers indicate min # of steps before a state can be computed



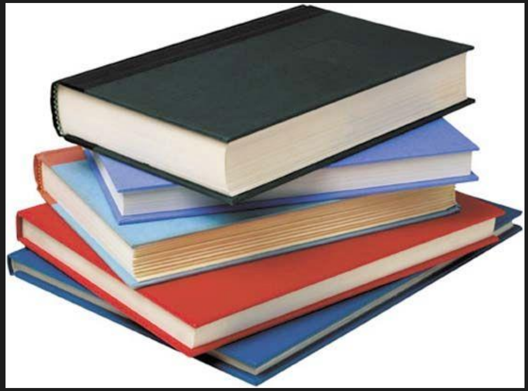




Issue#2 with SA – Look Ahead



Intuition to Multi-Head Attention



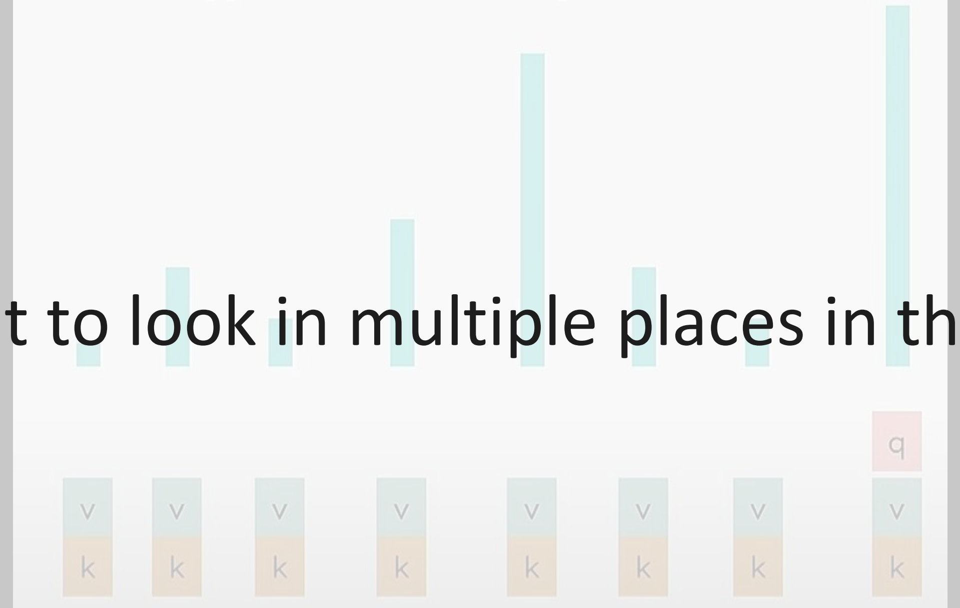
Attention head 1

"The Sleepy Child Reads A Book"

Attention head 2

Intuition to Multi-Head Attention

Attention head 1

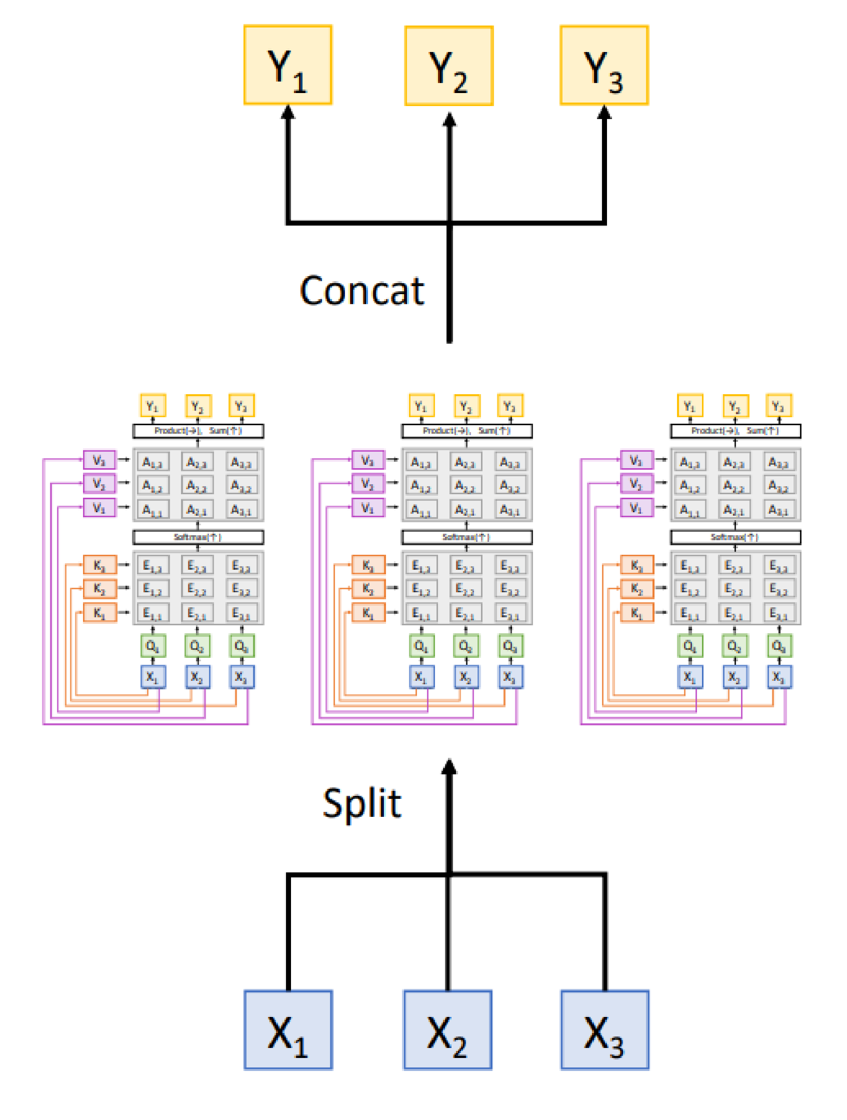


What if we want to look in multiple places in the sentence at once?

Attention head 2

"The Sleepy Child Reads A Book"

Multi-head Attention



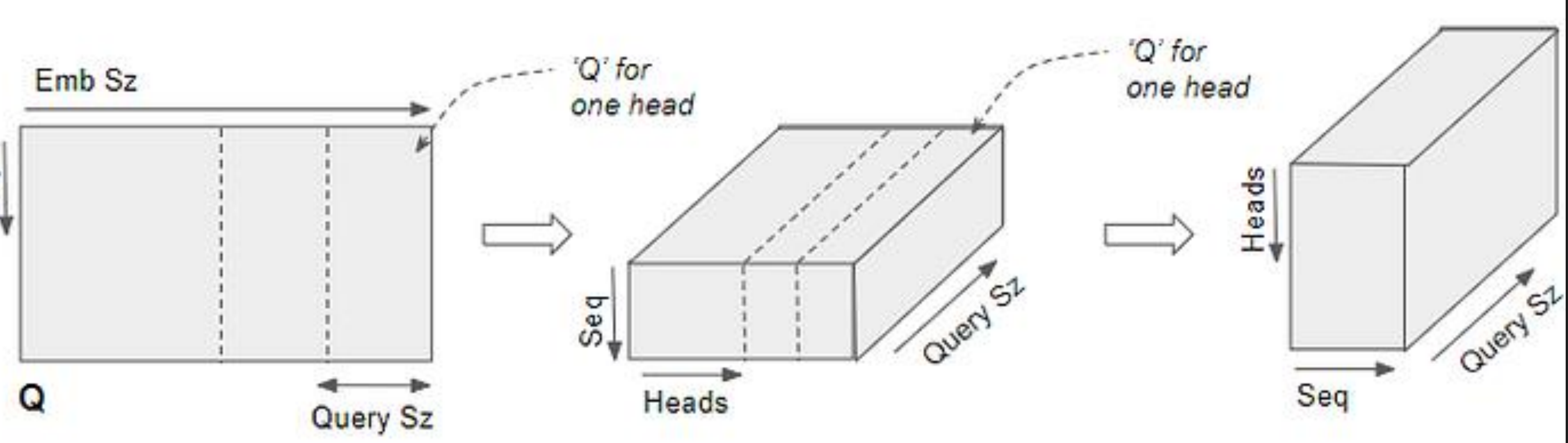
Multi-head Attention

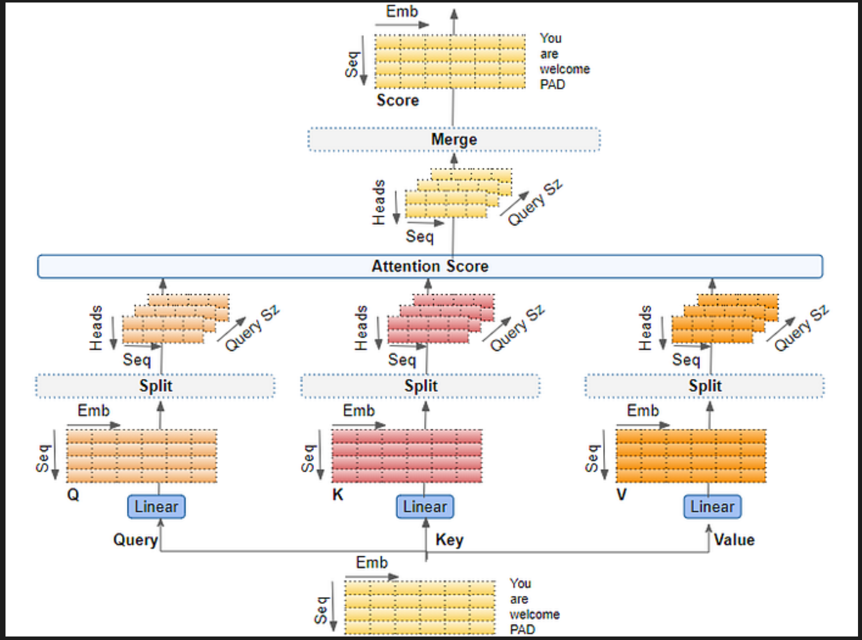
1. We compute $XQ \in R^{n \times d}$, and then reshape to $R^{n \times h \times \frac{d}{h}}$

(n: size of embedding vector of each input element, d:inner dimension of that's specific to each layer)

2. Then we transpose to $R^{h \times n \times \frac{d}{h},}$ now the head axis is like batch axis

3. Almost everything else is identical, and the matrices are the same sizes

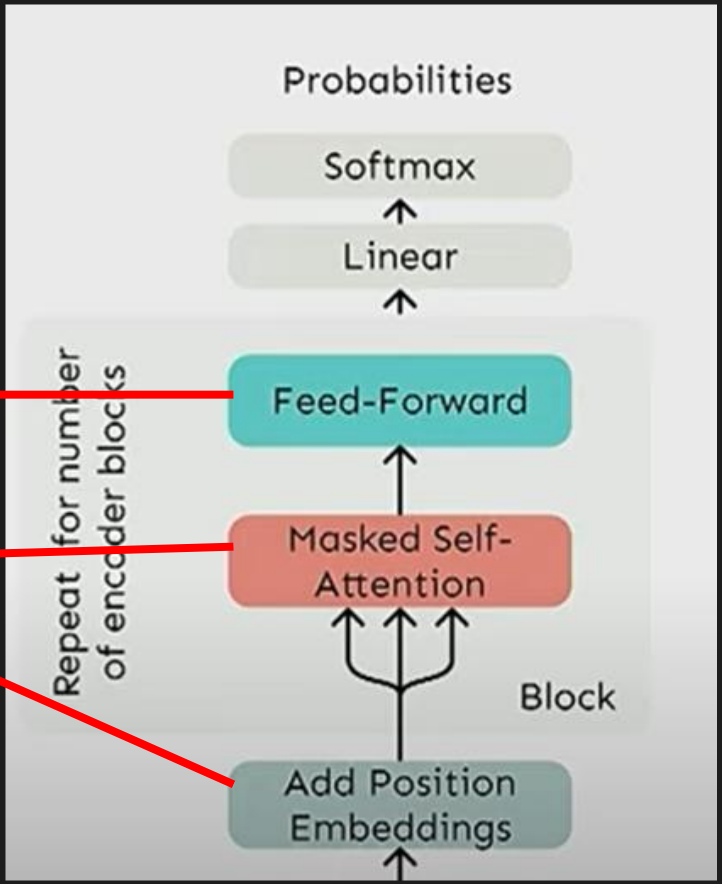




Self-attention building block

- **Self-attention**:
  - the basis of the method.
- **Position representations**:
  - Specify the sequence order, since self-attention is an unordered function of its inputs.
- **Nonlinearities**:
  - At the output of the self-attention block
  - Frequently implemented as a simple feed-forward network.
- **Masking**:
  - In order to parallelize operations while not looking at the future.
  - Keeps information about the future from “leaking” to the past.

- That’s it! But this is not the **Transformer** model we’ve been hearing about.

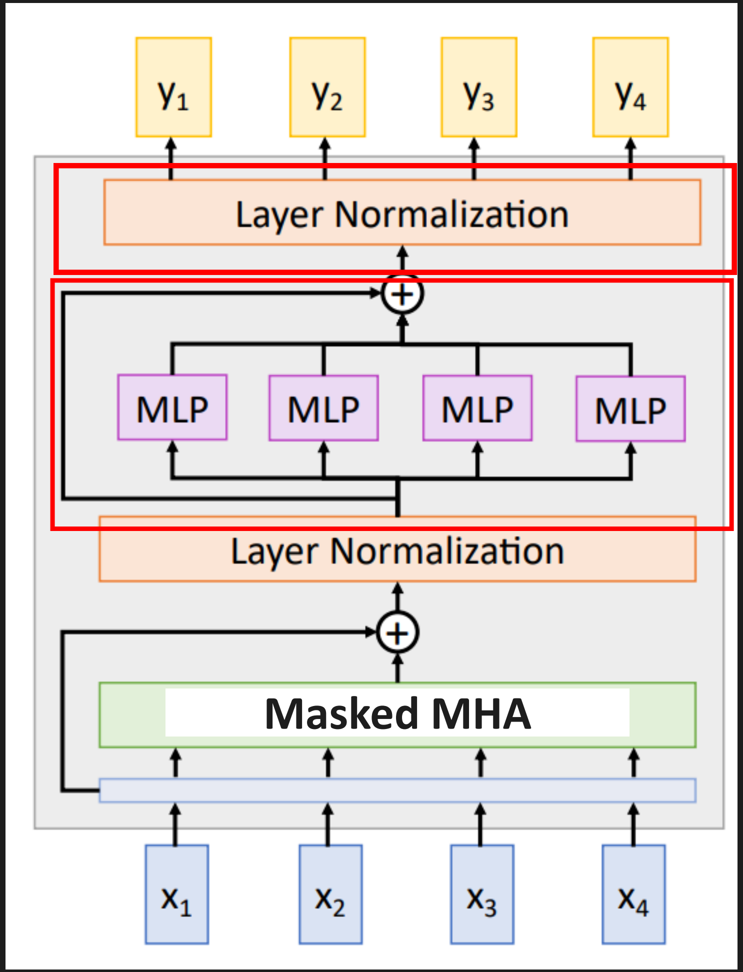


Input embedding

Transformer Decoder

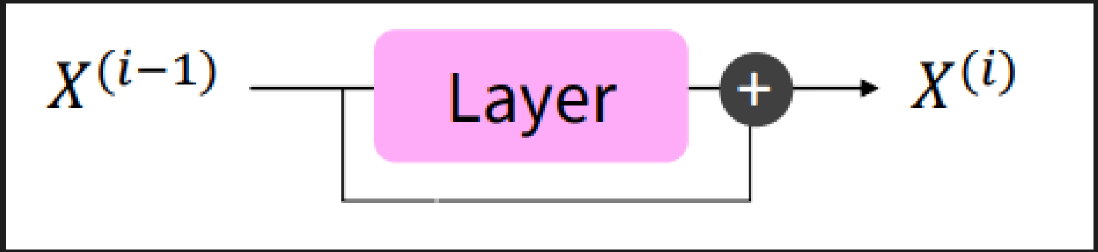
No elementwise nonlinearities
in self attention

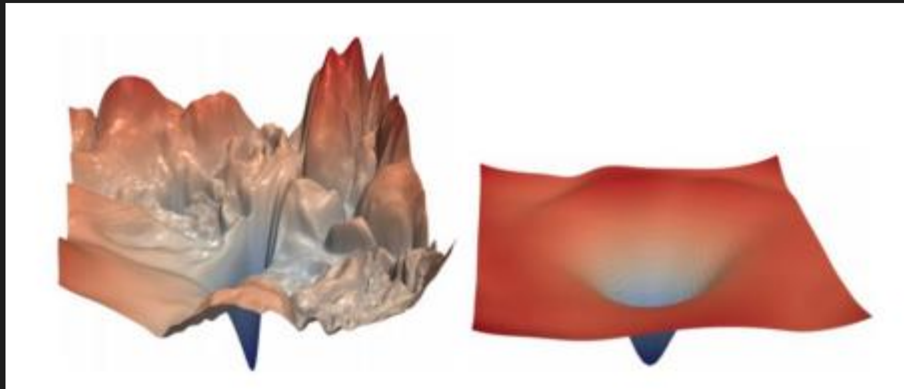
$$\begin{aligned} m_i &= MLP(\text{output}_i) \\ &= W_2 * \text{ReLU}(W_1 \times \text{output}_i + b_1) + b_2 \end{aligned}$$



Residual connection

We let $X^{(i)} = X^{(i-1)} + \text{Layer}(X^{(i-1)})$ (so we only have to learn "the residual" from the previous layer)





[no residuals] [residuals]

[Loss landscape visualization, Li et al., 2018, on a ResNet]

Layer normalization

Batch norm

d-dimensional vectors for each sample in batch

d-dim

$$a_1, a_2, \ldots, a_B$$

$$\mu = \frac{1}{B}\sum_{i=1}^{B} a_i \qquad \sigma = \sqrt{\frac{1}{B}\sum_{i=1}^{B}(a_i - \mu)^2}$$

$$\bar{a}_i = \frac{a_i - \mu}{\sigma}\gamma + \beta$$

Layer norm

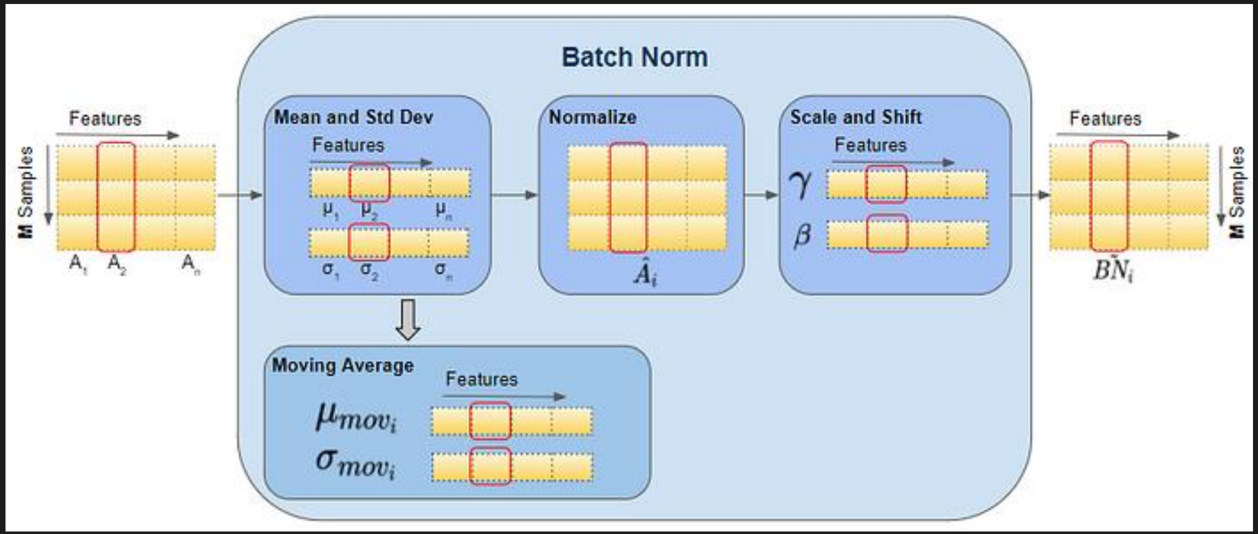
different *dimensions* of a

1-dim

$$a$$

$$\mu = \frac{1}{d}\sum_{i=1}^{d} a_j \qquad \sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(a_j - \mu)^2}$$

$$\bar{a} = \frac{a - \mu}{\sigma}\gamma + \beta$$



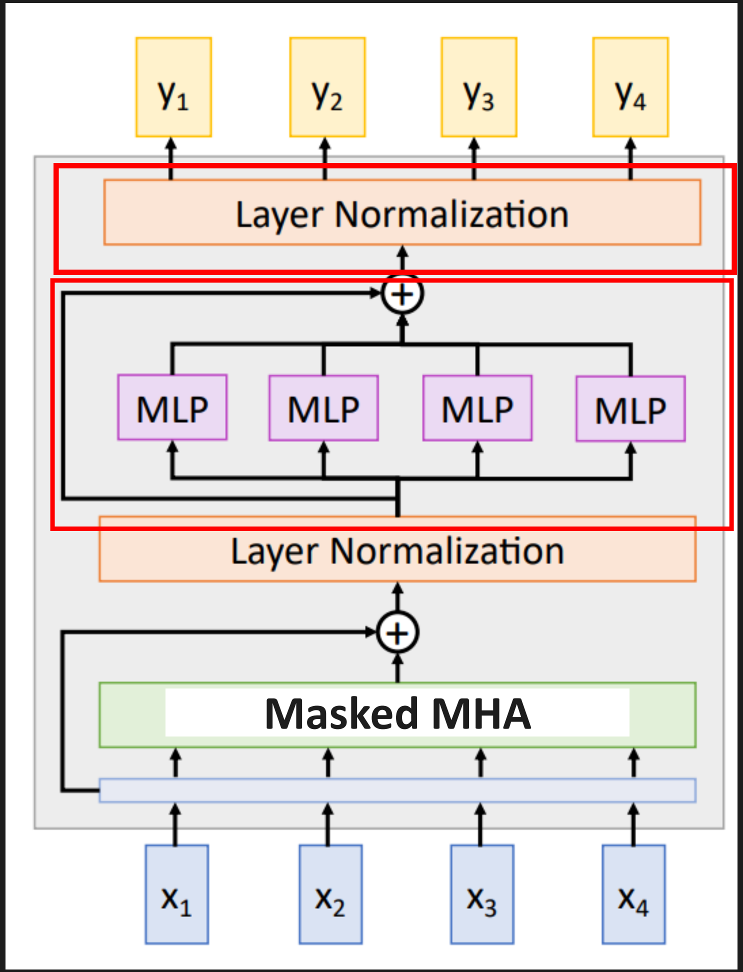
Layer normalization

- Trick to help models train faster
- Batch normalization is usually less effective than layer normalization in NLP tasks
- Any batch number works
- Can parallelize

Transformer Decoder

No elementwise nonlinearities
in self attention

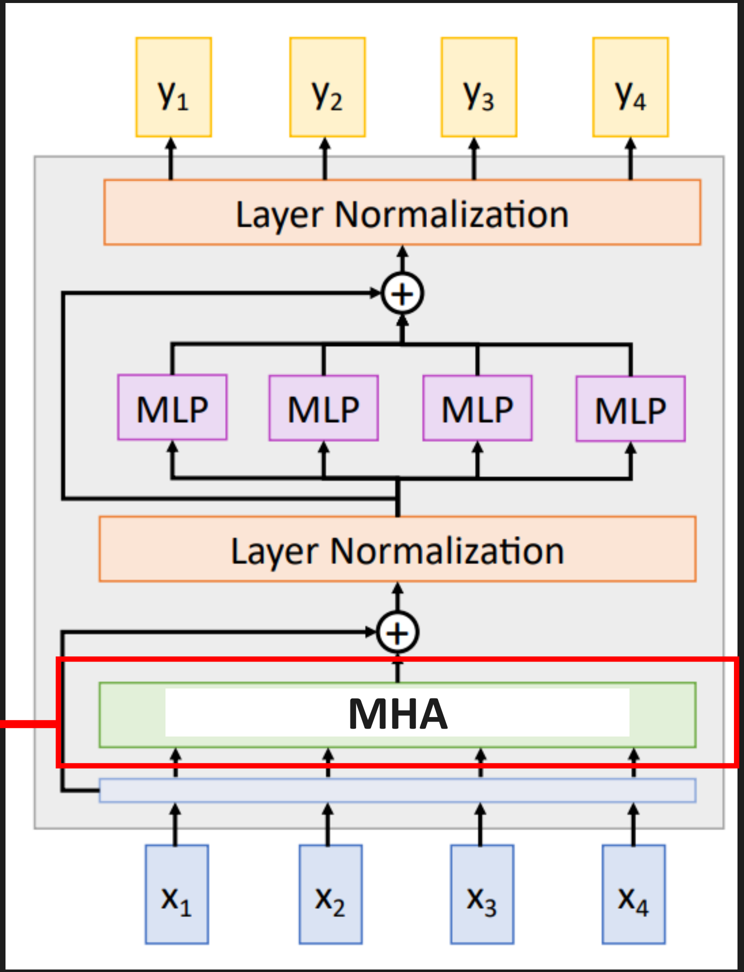
$$m_i = MLP(\text{output}_i)$$
$$= W_2 * \text{ReLU}(W_1 \times \text{output}_i + b_1) + b_2$$



Transformer Encoder

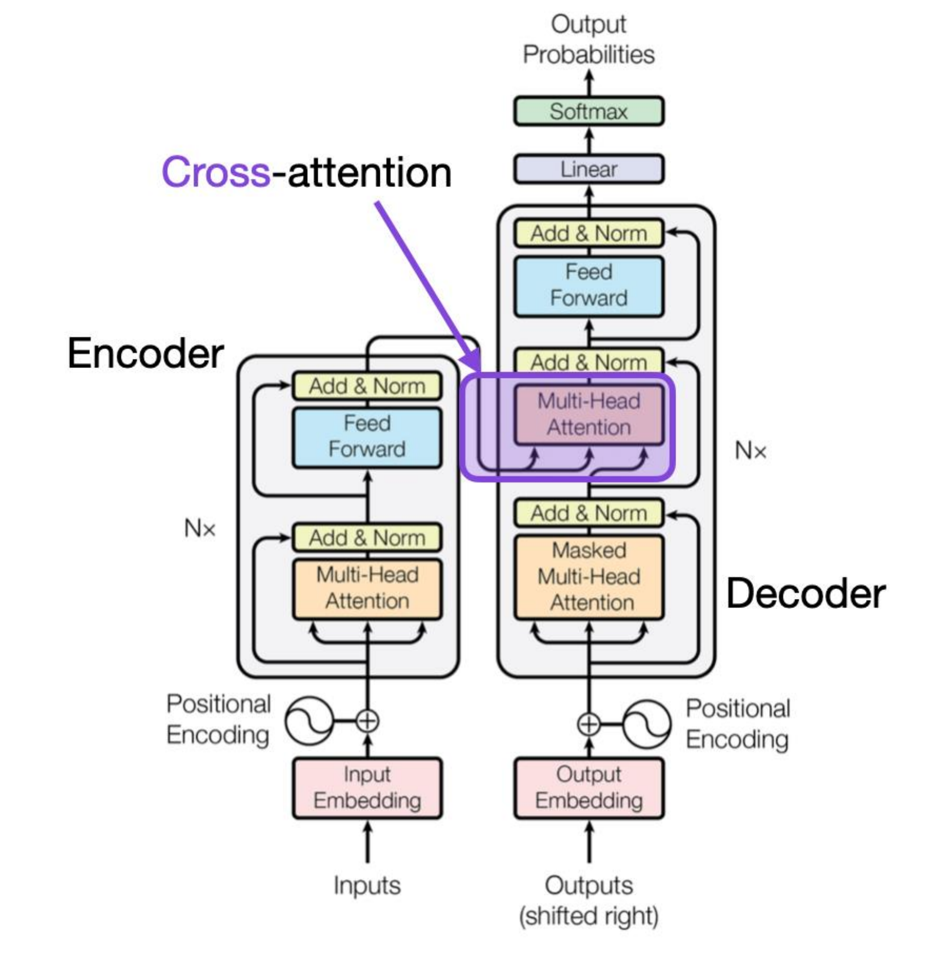
No Masking for

bidirectional context



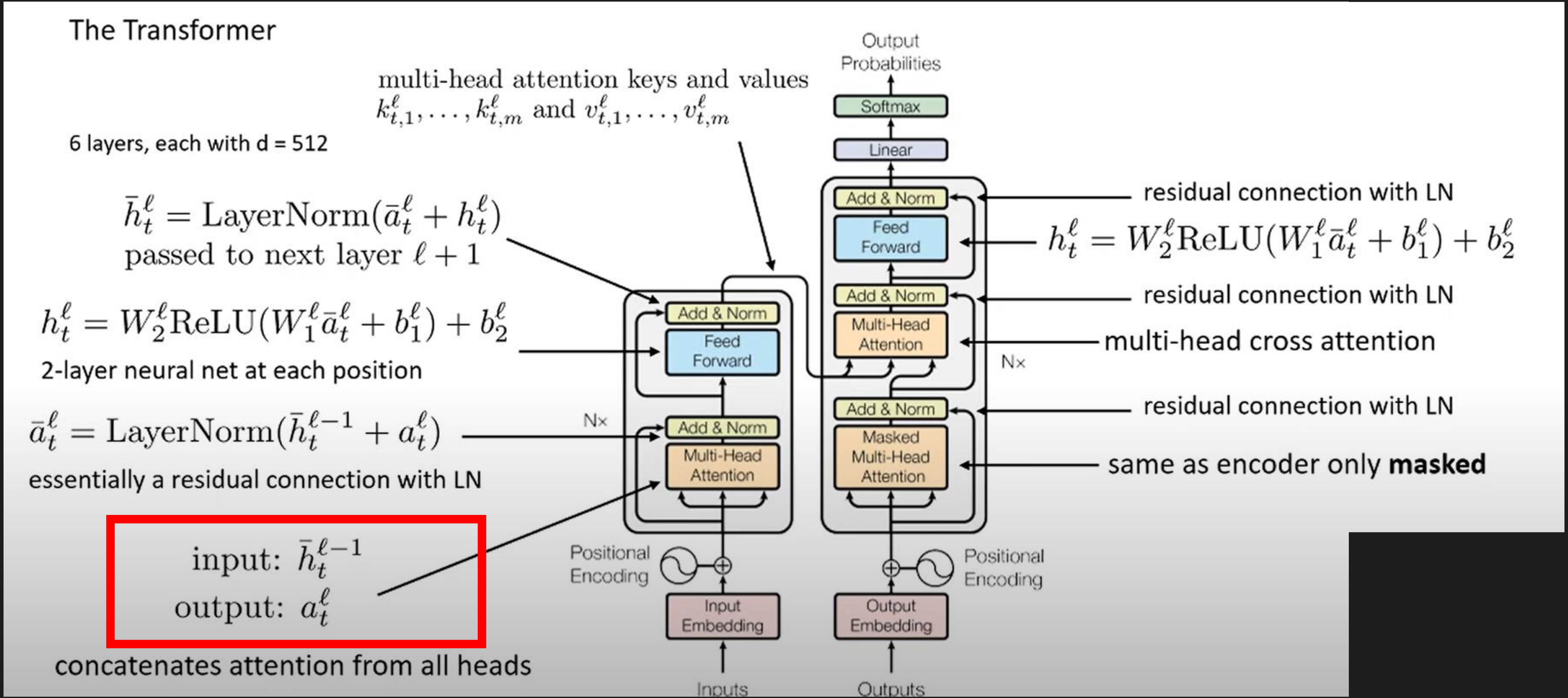
Transformer Encoder-Decoder

- Let $h_1, \ldots, h_T$ be **output** vectors **from** the Transformer **encoder**; $x_i \in \mathbb{R}^d$
- Let $z_1, \ldots, z_T$ be input vectors from the Transformer **decoder**, $z_i \in \mathbb{R}^d$
- Then keys and values are drawn from the **encoder** (like a memory):
  - $k_i = Kh_i, v_i = Vh_i$.
- And the queries are drawn from the **decoder**, $q_i = Qz_i$.

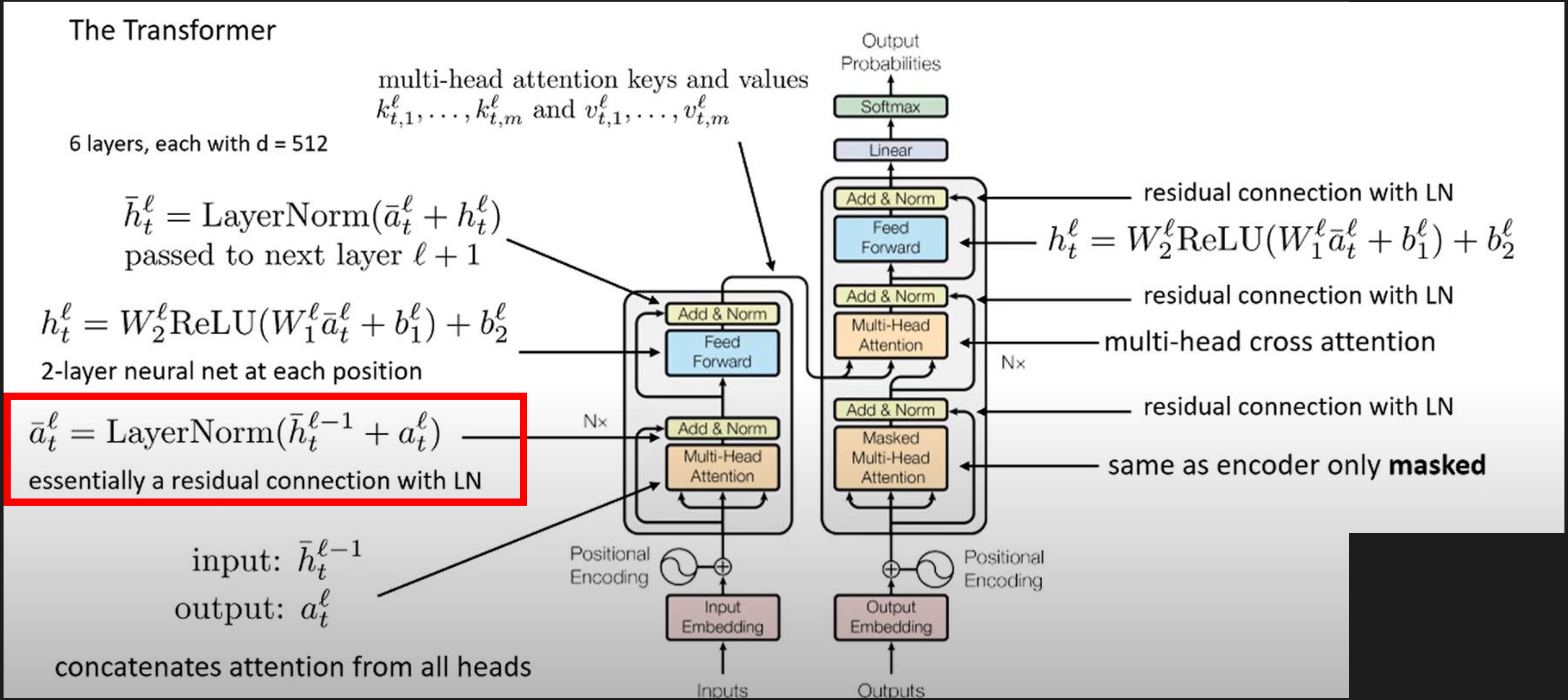


Source: "Attention Is All You Need" (https://arxiv.org/abs/1706.03762)

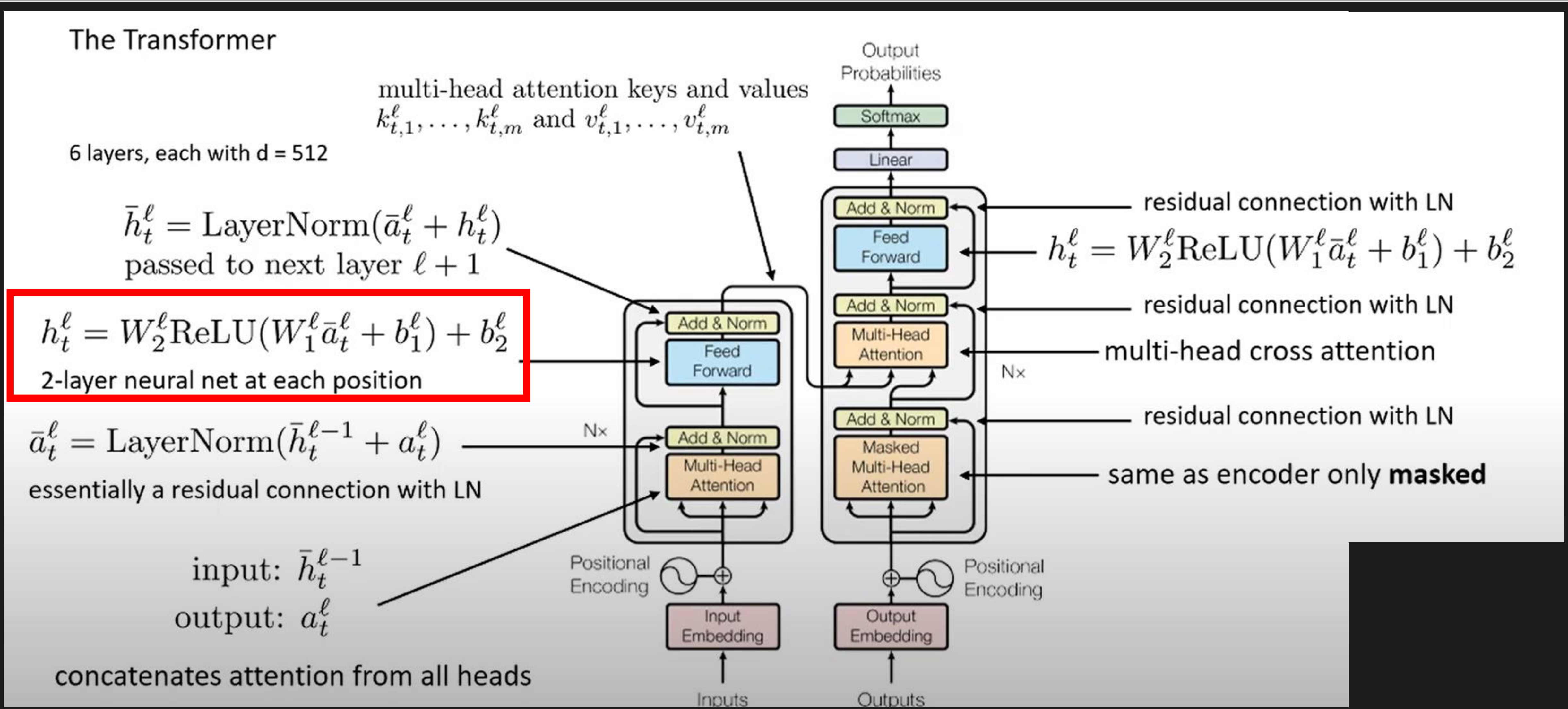
Full transformer



Full transformer



Full transformer



Full transformer

The Transformer

multi-head attention keys and values
$k^\ell_{t,1}, \ldots, k^\ell_{t,m}$ and $v^\ell_{t,1}, \ldots, v^\ell_{t,m}$

6 layers, each with d = 512

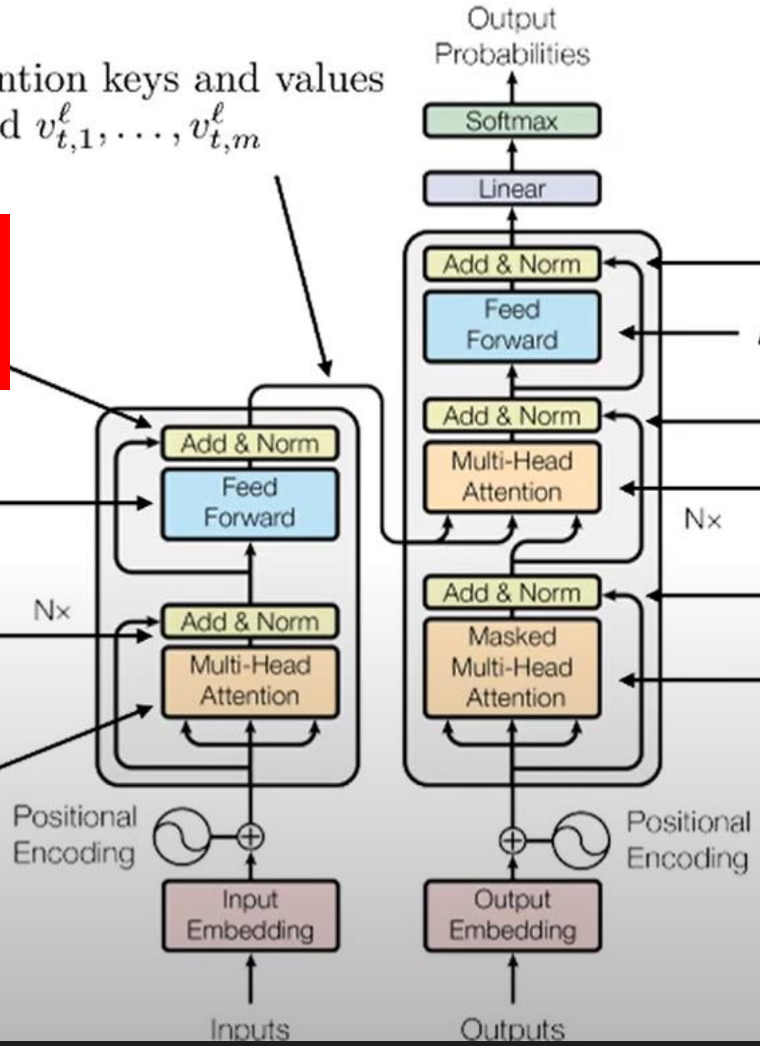
$$\bar{h}^\ell_t = \mathrm{LayerNorm}(\bar{a}^\ell_t + h^\ell_t)$$
passed to next layer $\ell + 1$

$$h^\ell_t = W^\ell_2 \mathrm{ReLU}(W^\ell_1 \bar{a}^\ell_t + b^\ell_1) + b^\ell_2$$
2-layer neural net at each position

$$\bar{a}^\ell_t = \mathrm{LayerNorm}(\bar{h}^{\ell-1}_t + a^\ell_t)$$
essentially a residual connection with LN

input: $\bar{h}^{\ell-1}_t$
output: a^ℓ_t

concatenates attention from all heads



residual connection with LN

$$h^\ell_t = W^\ell_2 \mathrm{ReLU}(W^\ell_1 \bar{a}^\ell_t + b^\ell_1) + b^\ell_2$$

residual connection with LN

multi-head cross attention

residual connection with LN

same as encoder only **masked**

Full transformer

The Transformer

multi-head attention keys and values
$k_{t,1}^{\ell}, \ldots, k_{t,m}^{\ell}$ and $v_{t,1}^{\ell}, \ldots, v_{t,m}^{\ell}$

6 layers, each with d = 512

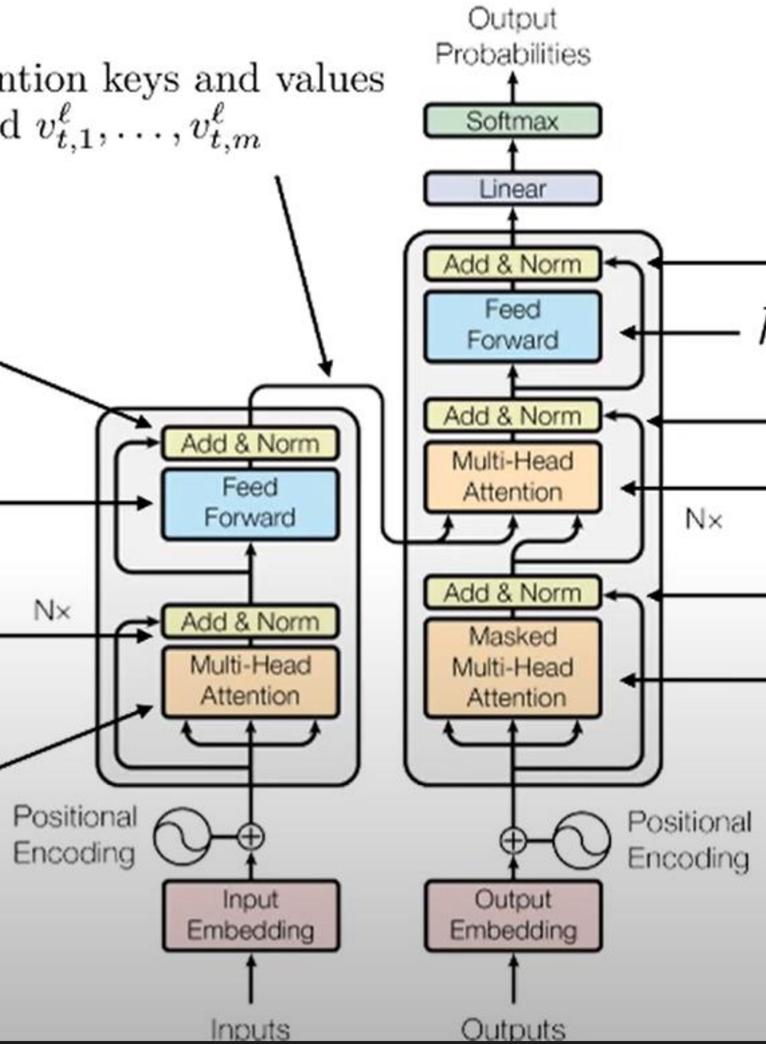
$\bar{h}_t^{\ell} = \mathrm{LayerNorm}(\bar{a}_t^{\ell} + h_t^{\ell})$
passed to next layer $\ell + 1$

$h_t^{\ell} = W_2^{\ell}\mathrm{ReLU}(W_1^{\ell}\bar{a}_t^{\ell} + b_1^{\ell}) + b_2^{\ell}$
2-layer neural net at each position

$\bar{a}_t^{\ell} = \mathrm{LayerNorm}(\bar{h}_t^{\ell-1} + a_t^{\ell})$
essentially a residual connection with LN

input: $\bar{h}_t^{\ell-1}$
output: a_t^{ℓ}

concatenates attention from all heads



residual connection with LN

$h_t^{\ell} = W_2^{\ell}\mathrm{ReLU}(W_1^{\ell}\bar{a}_t^{\ell} + b_1^{\ell}) + b_2^{\ell}$

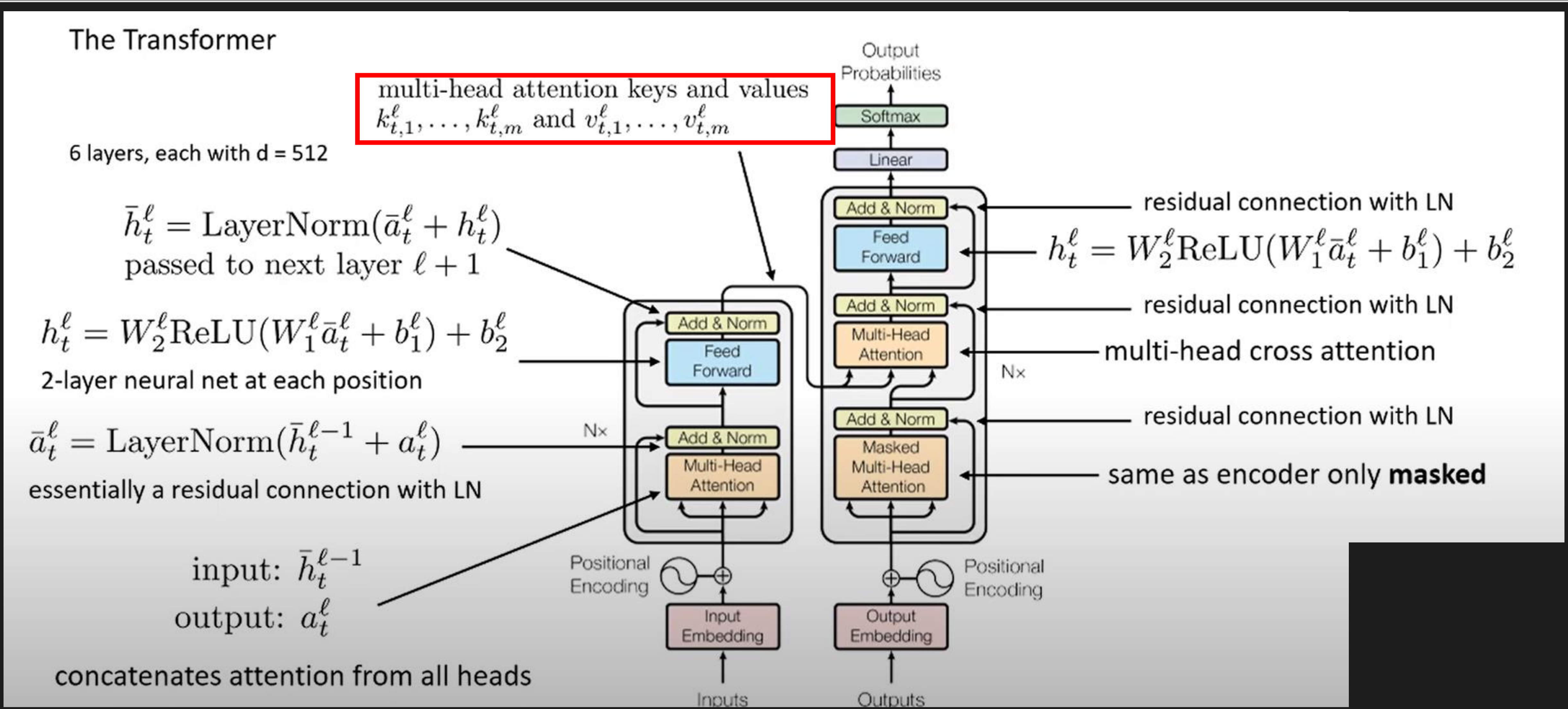
residual connection with LN

multi-head cross attention

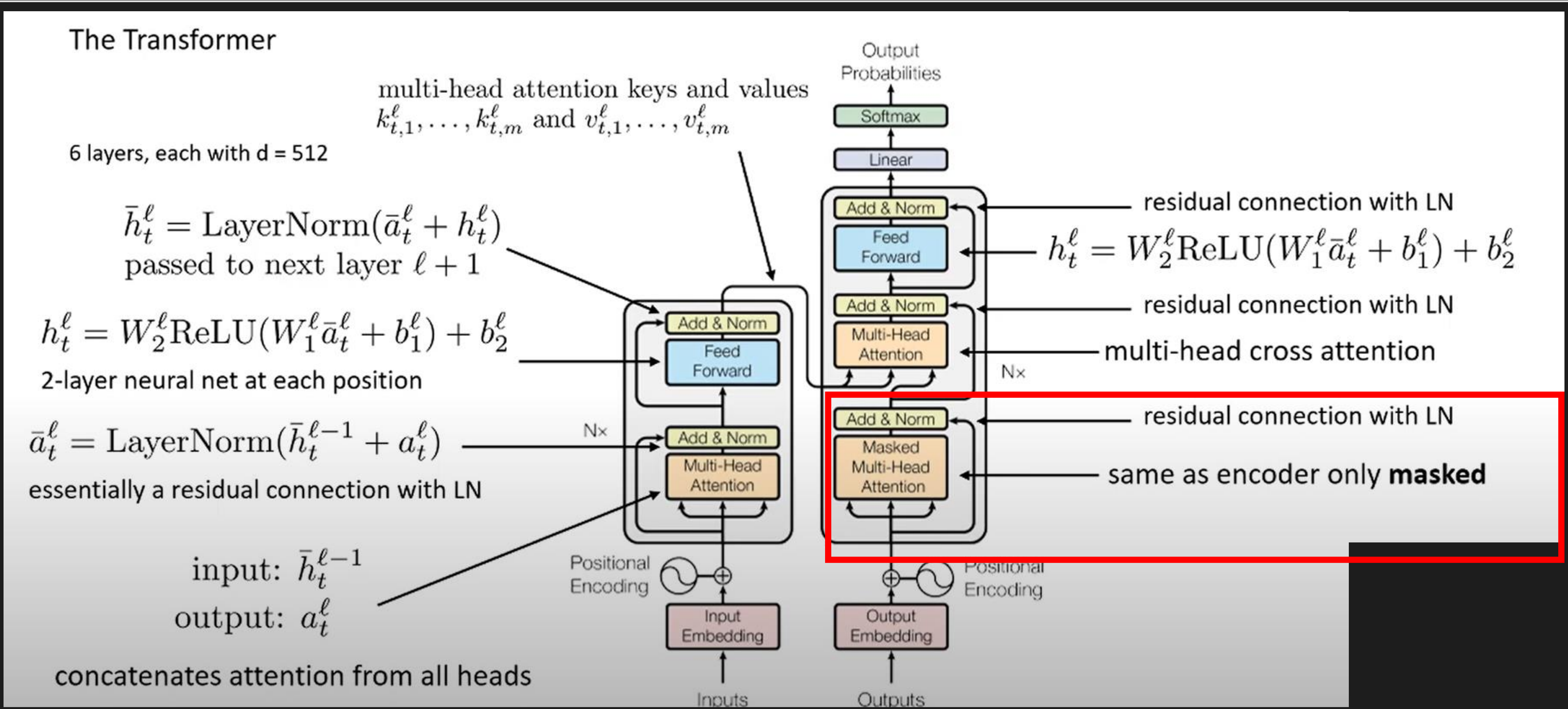
residual connection with LN

same as encoder only **masked**

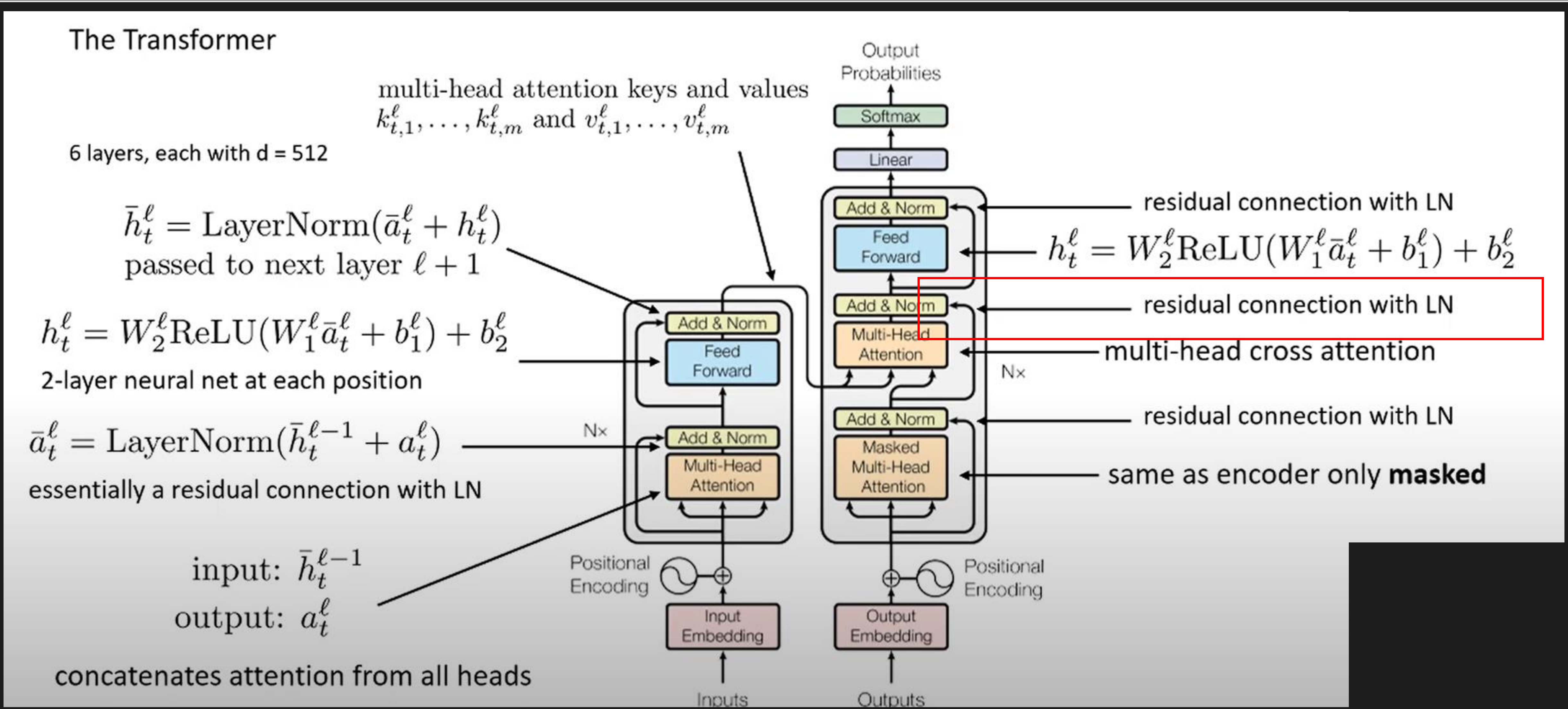
Full transformer



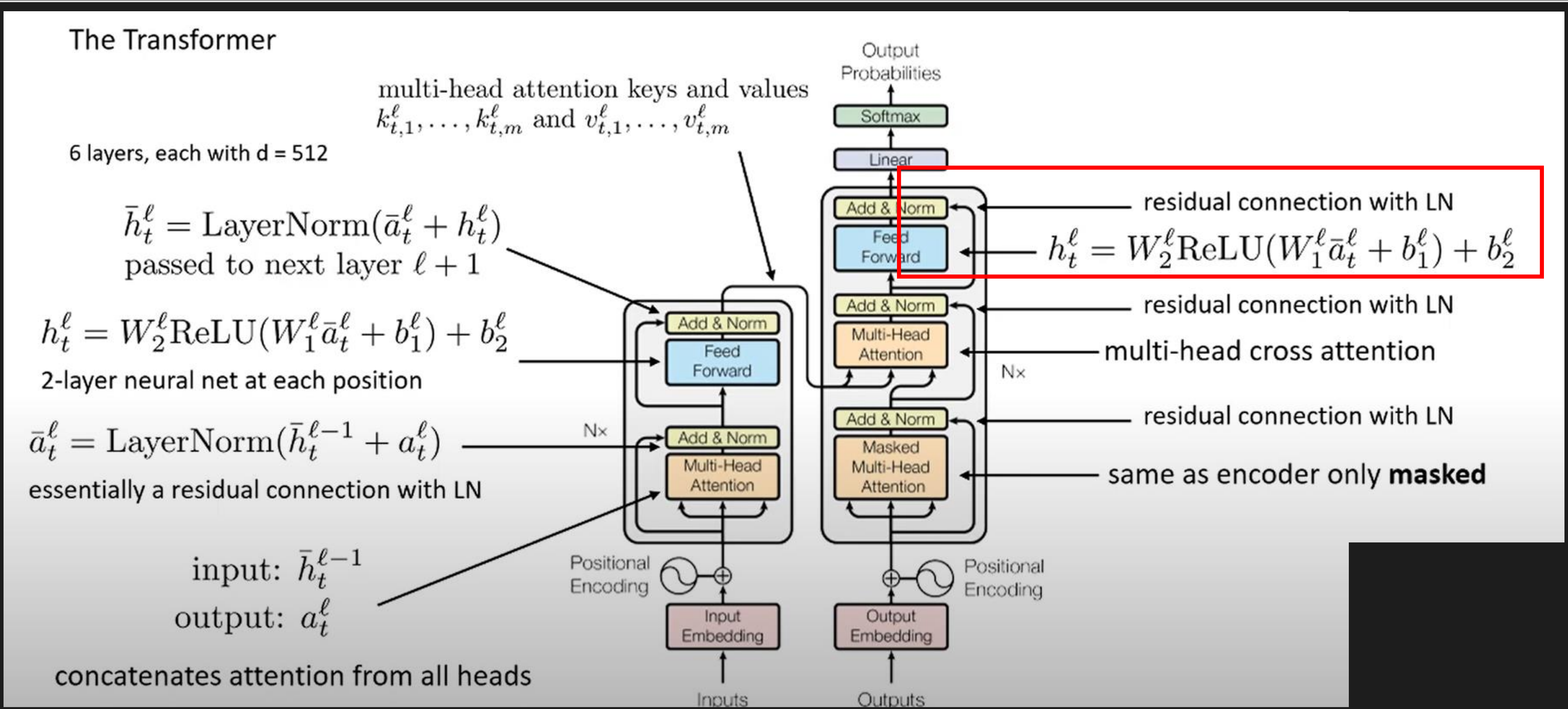
Full transformer



Full transformer



Full transformer

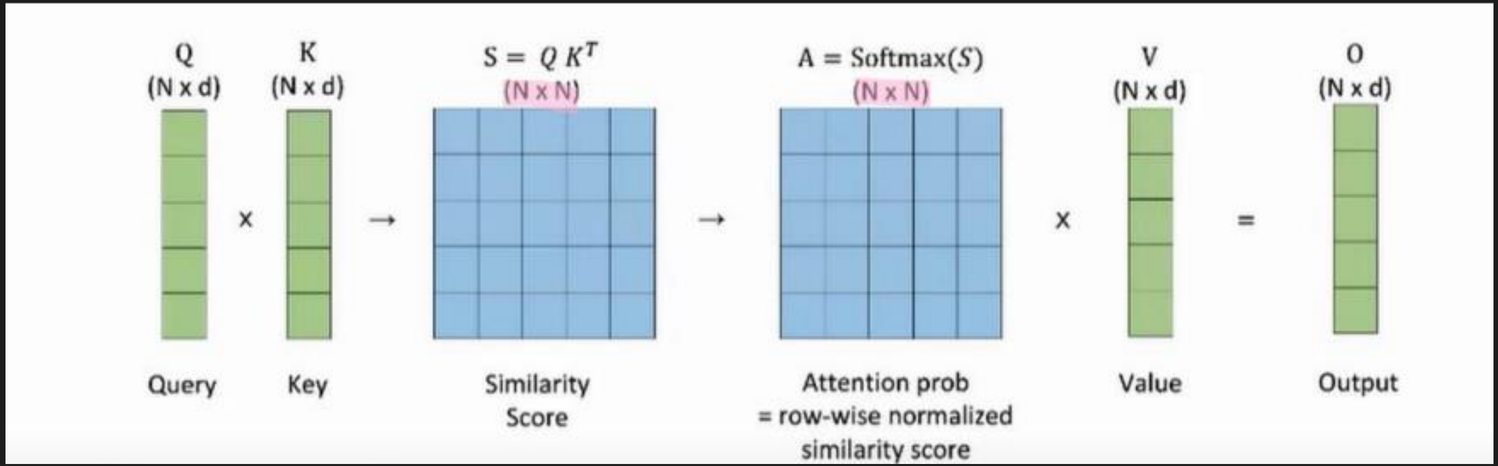


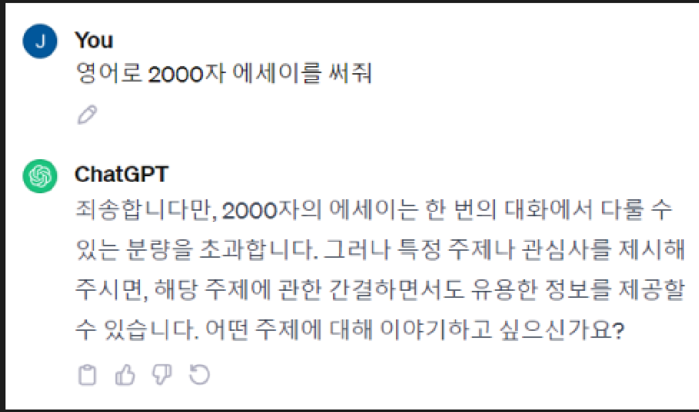
Benefits of transformers

- Much easier to parallelize (especially encoder)
- Useful for transfer learning
- Much deeper than RNN (by incrementing blocks)

Downsides of transformers

- Attention computations are quadratic : O(n^2)





- How to better represent position?
  - Does the set of positions need to be decided ahead of time?
  - Does the scheme hinder generalization to new positions?

디스커션 과제

- 개인 질문
  - Transformer의 input sequence가 너무 길면 어떠한 문제점이 발생할 수 있을까요? 또한 이 문제점을 해결하는 방법으로는 어떤 것들이 있을까요?
- 팀 질문
  - Sinusoidal Positional Encoding과 같이 상당수의 PE는 fixed 함수에 의해 계산된 값을 사용합니다.
    - 이 때 발생할 수 있는 문제점이 무엇일까요?
    - 학습 가능한 Positional encoding을 만드는 방법은 없을까요? 또한 학습 가능한 PE가 가지는 장단점에는 어떤 것이 있을까요?

감사합니다.