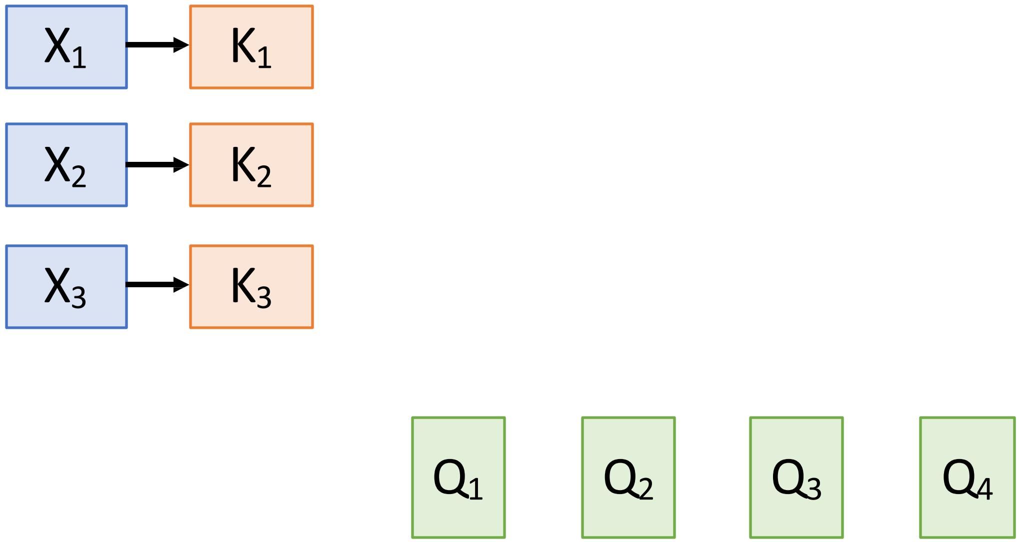
Attention Layer

<u>Inputs</u>:
Query vectors: $\mathbf{Q}$ (Shape: N_Q x D_Q)
Input vectors: $\mathbf{X}$ (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)

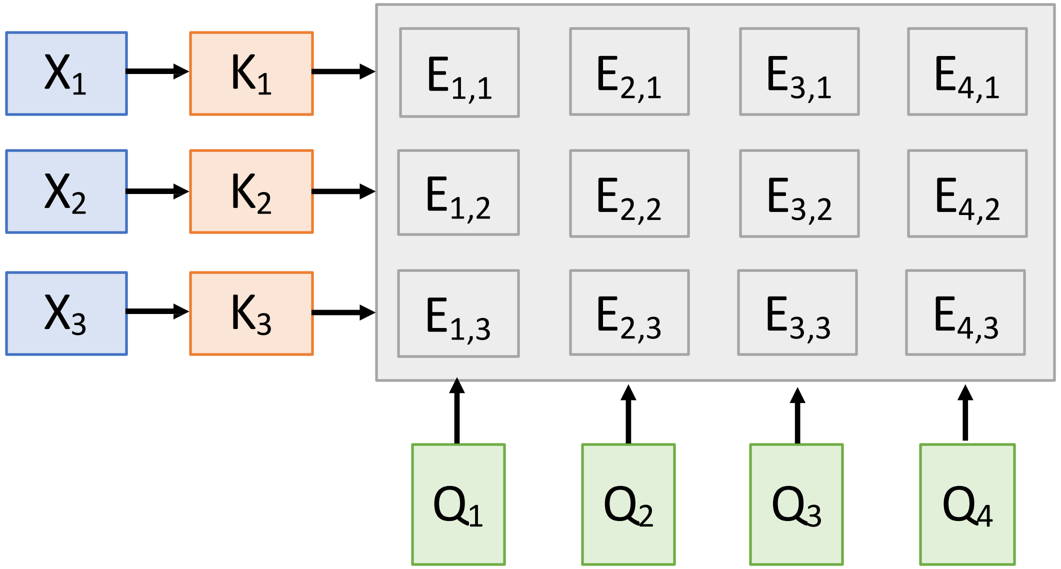
<u>Computation</u>:
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{QK}^T$ (Shape: N_Q x N_X) $E_{i,j} = \mathbf{Q_i} \cdot \mathbf{K_j} / \mathrm{sqrt}(D_Q)$
Attention weights: $A = \mathrm{softmax}(E, \mathrm{dim}=1)$ (Shape: N_Q x N_X)
Output vectors: $Y = A\mathbf{V}$ (Shape: N_Q x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$



Attention Layer

Inputs:
Query vectors: $\mathbf{Q}$ (Shape: N_Q x D_Q)
Input vectors: $\mathbf{X}$ (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)

Computation:
Key vectors: $\mathbf{K} = \mathbf{X}\mathbf{W_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{X}\mathbf{W_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{Q}\mathbf{K}^T$ (Shape: N_Q x N_X) $E_{i,j} = \mathbf{Q_i} \cdot \mathbf{K_j} / \mathrm{sqrt}(D_Q)$
Attention weights: $A = \mathrm{softmax}(E, \mathrm{dim}=1)$ (Shape: N_Q x N_X)
Output vectors: $Y = A\mathbf{V}$ (Shape: N_Q x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$



Attention Layer

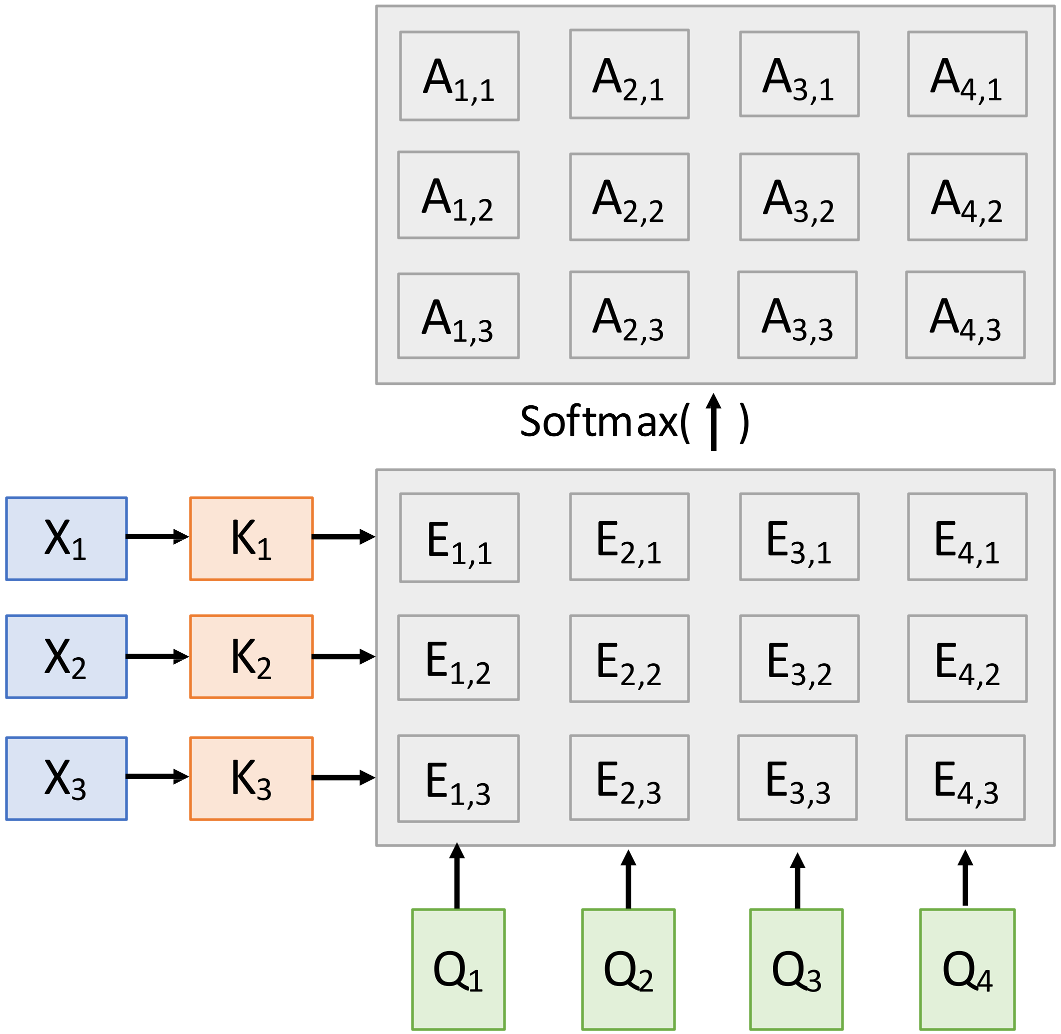
Inputs:

Query vectors: **Q** (Shape: N_Q x D_Q)

Input vectors: **X** (Shape: N_X x D_X)

Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)

Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)



Computation:

Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)

Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)

Similarities: $E = \mathbf{QK^T}$ (Shape: N_Q x N_X) $E_{i,j} = \mathbf{Q_i} \cdot \mathbf{K_j} / \mathrm{sqrt}(D_Q)$

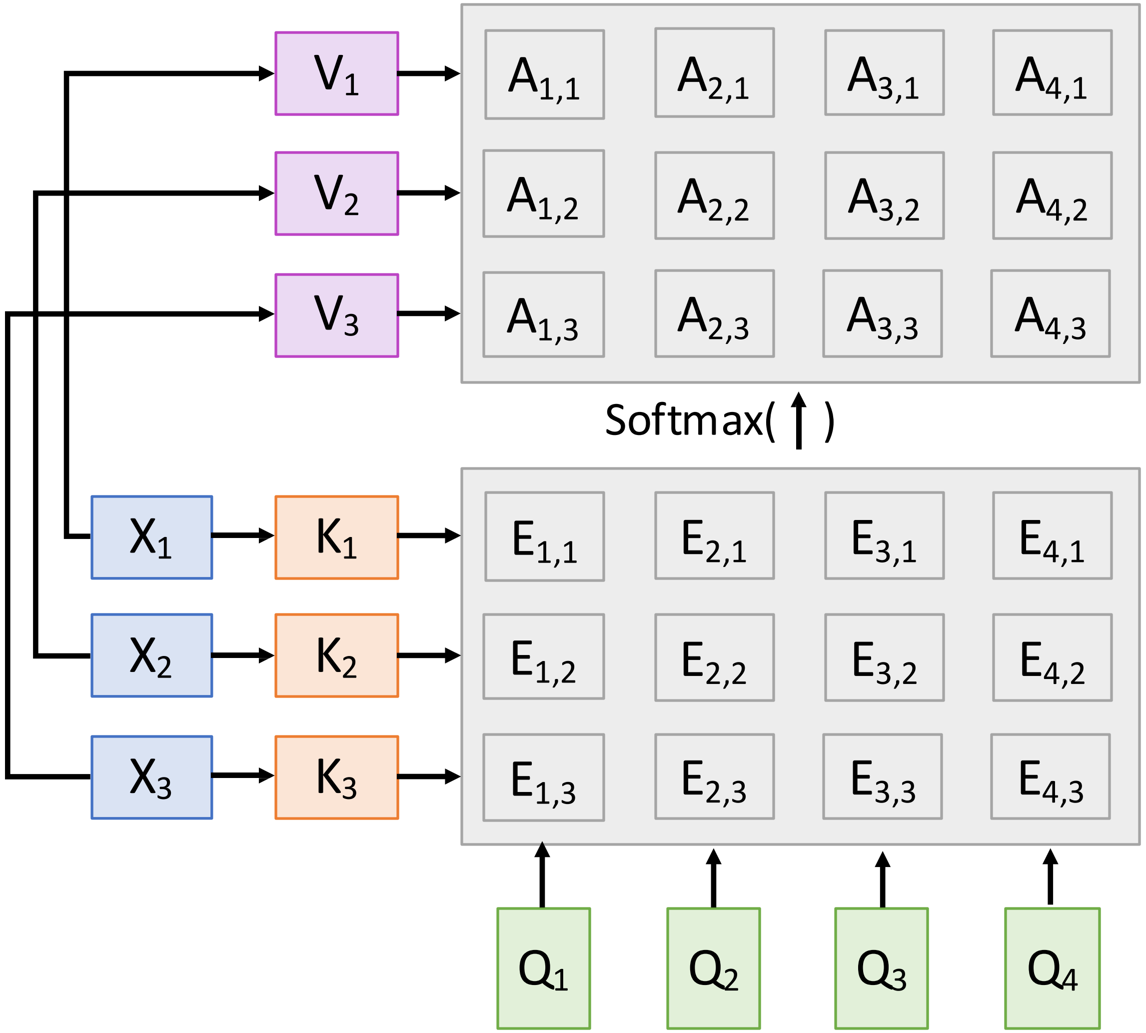
Attention weights: $A = \mathrm{softmax}(E)$ (Shape: N_Q x N_X)

Output vectors: $Y = A\mathbf{V}$ (Shape: N_Q x D_V) $Y_i = \sum_j A_{i,j} \mathbf{V_j}$

Attention Layer

Inputs:
Query vectors: Q (Shape: $N_Q \times D_Q$)
Input vectors: X (Shape: $N_X \times D_X$)
Key matrix: W_K (Shape: $D_X \times D_Q$)
Value matrix: W_V (Shape: $D_X \times D_V$)

Computation:
Key vectors: $K = XW_K$ (Shape: $N_X \times D_Q$)
Value Vectors: $V = XW_V$ (Shape: $N_X \times D_V$)
Similarities: $E = QK^T$ (Shape: $N_Q \times N_X$) $E_{i,j} = Q_i \cdot K_j / \text{sqrt}(D_Q)$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_Q \times N_X$)
Output vectors: $Y = AV$ (Shape: $N_Q \times D_V$) $Y_i = \sum_j A_{i,j} V_j$

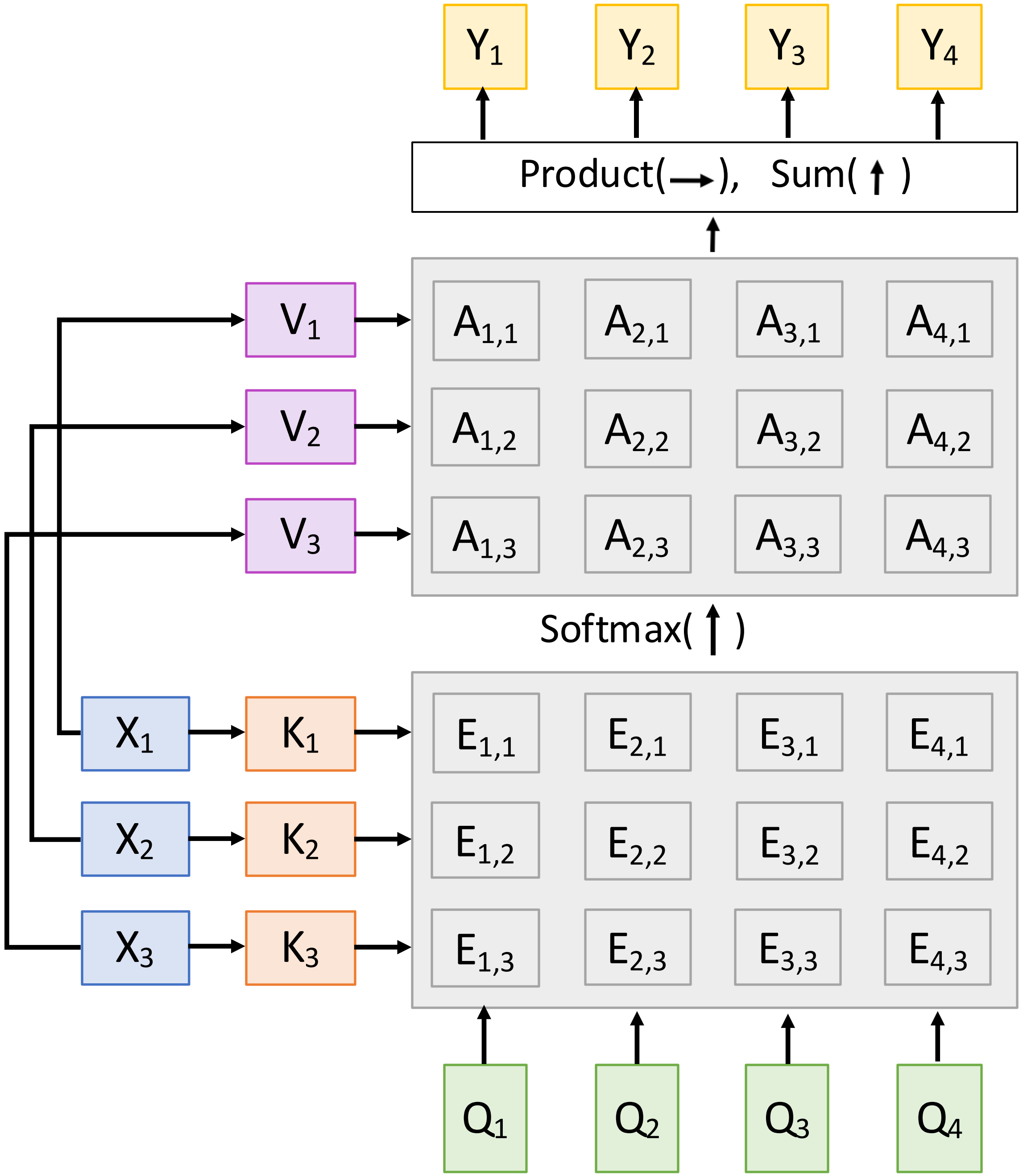


Monsoon,

Attention Layer

Inputs:
Query vectors: Q (Shape: N_Q x D_Q)
Input vectors: X (Shape: N_X x D_X)
Key matrix: W_K (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)

Computation:
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value Vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T$ (Shape: N_Q x N_X) $E_{i,j} = Q_i \cdot K_j / \mathrm{sqrt}(D_Q)$
Attention weights: $A = \mathrm{softmax}(E, \mathrm{dim}=1)$ (Shape: N_Q x N_X)
Output vectors: $Y = AV$ (Shape: N_Q x D_V) $Y_i = \sum_j A_{i,j} V_j$



Self-Attention Layer

One **query** per **input vector**

- Inputs:
- Input vectors: X (Shape: $N_X \times D_X$) Key matrix: W_K (Shape: $D_X \times D_Q$)
- Value matrix: W_V (Shape: $D_X \times D_V$) Query matrix: W_Q (Shape: $D_X \times D_Q$)

- Computation:
- Query vectors: $Q = XW_Q$
- Key vectors: $K = XW_K$ (Shape: $N_X \times D_Q$)
- Value Vectors: $V = XW_V$ (Shape: $N_X \times D_V$)
- Similarities: $E = QK^T$ (Shape: $N_X \times N_X$) $E_{i,j} = Q_i \cdot K_j$ /

X_1 X_2 X_3

Monsoon,

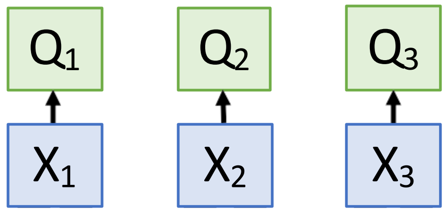
Self-Attention Layer

One **query** per **input vector**

- Inputs:
- Input vectors: X

(Shape: N_X x D_X) Key matrix:
W_K (Shape: D_X x D_Q) Value matrix:
W_V (Shape: D_X x D_V) Query matrix:
W_Q (Shape: D_X x D_Q)

- Computation:
- Query vectors: $Q = XW_Q$
- Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
- Value Vectors: $V = XW_V$ (Shape: N_X x D_V)
- Similarities: $E = QK^T$ (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j$

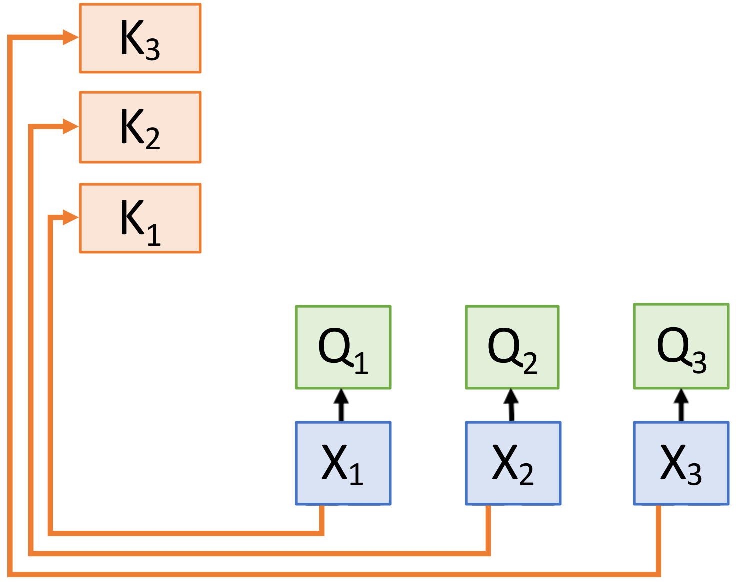


Self-Attention Layer

One **query** per **input vector**

- Inputs:
- Input vectors: X (Shape: N_X x D_X) Key matrix: W_K (Shape: D_X x D_Q)
- Value matrix: W_V (Shape: D_X x D_V)
- Query matrix: W_Q (Shape: D_X x D_Q)

- Computation:
- Query vectors: $Q = XW_Q$
- Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
- Value Vectors: $V = XW_V$ (Shape: N_X x D_V)
- Similarities: $E = QK^T$ (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j$ / sqrt(D_Q) Attention weights: A = softmax(E, dim=1) (Shape: N_X x



Monsoon,

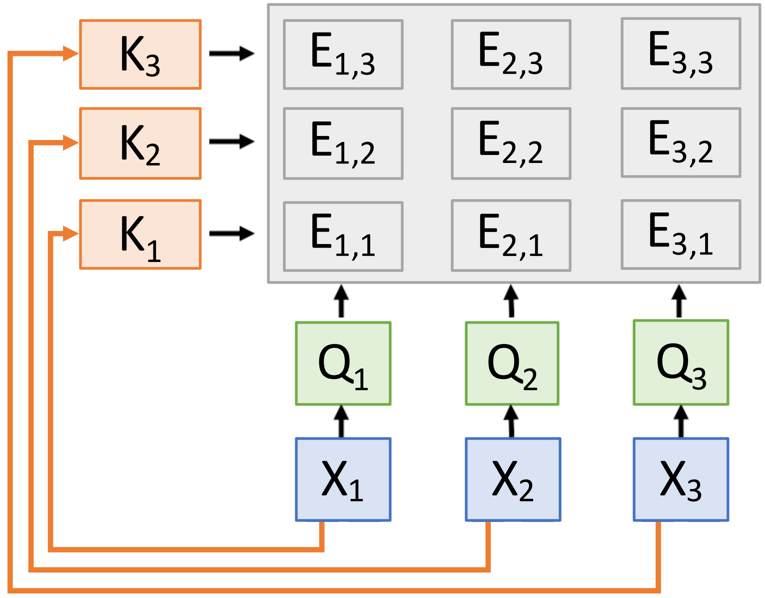
Self-Attention Layer

One **query** per **input vector**

- Inputs:
- Input vectors: X

(Shape: N_X x D_X) Key matrix:
W_K (Shape: D_X x D_Q) Value matrix:
W_V (Shape: D_X x D_V) Query matrix:
W_Q (Shape: D_X x D_Q)

- Computation:
- Query vectors: $Q = XW_Q$
- Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
- Value Vectors: $V = XW_V$ (Shape: N_X x D_V)
- Similarities: $E = QK^T$ (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j$



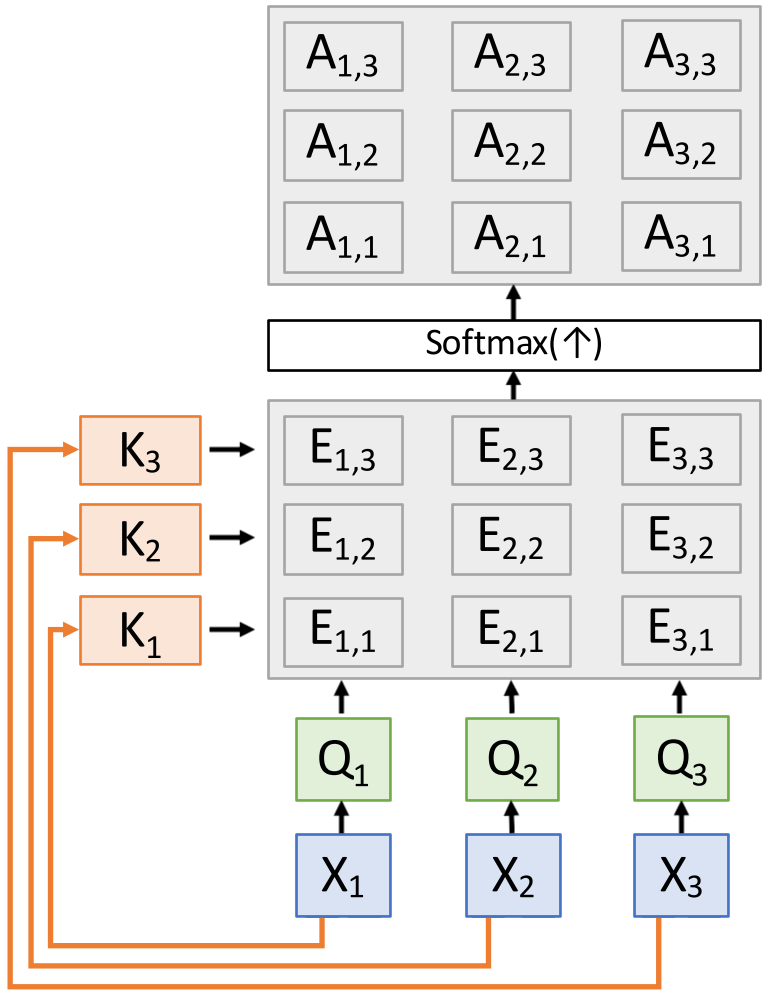
Self-Attention Layer

One **query** per **input vector**

- Inputs:
- Input vectors: X

(Shape: N_X x D_X) Key matrix:
W_K (Shape: D_X x D_Q) Value matrix:
W_V (Shape: D_X x D_V) Query matrix:
W_Q (Shape: D_X x D_Q)

- Computation:
- Query vectors: $Q = XW_Q$
- Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
- Value Vectors: $V = XW_V$ (Shape: N_X x D_V)
- Similarities: $E = QK^T$ (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j$



Monsoon,

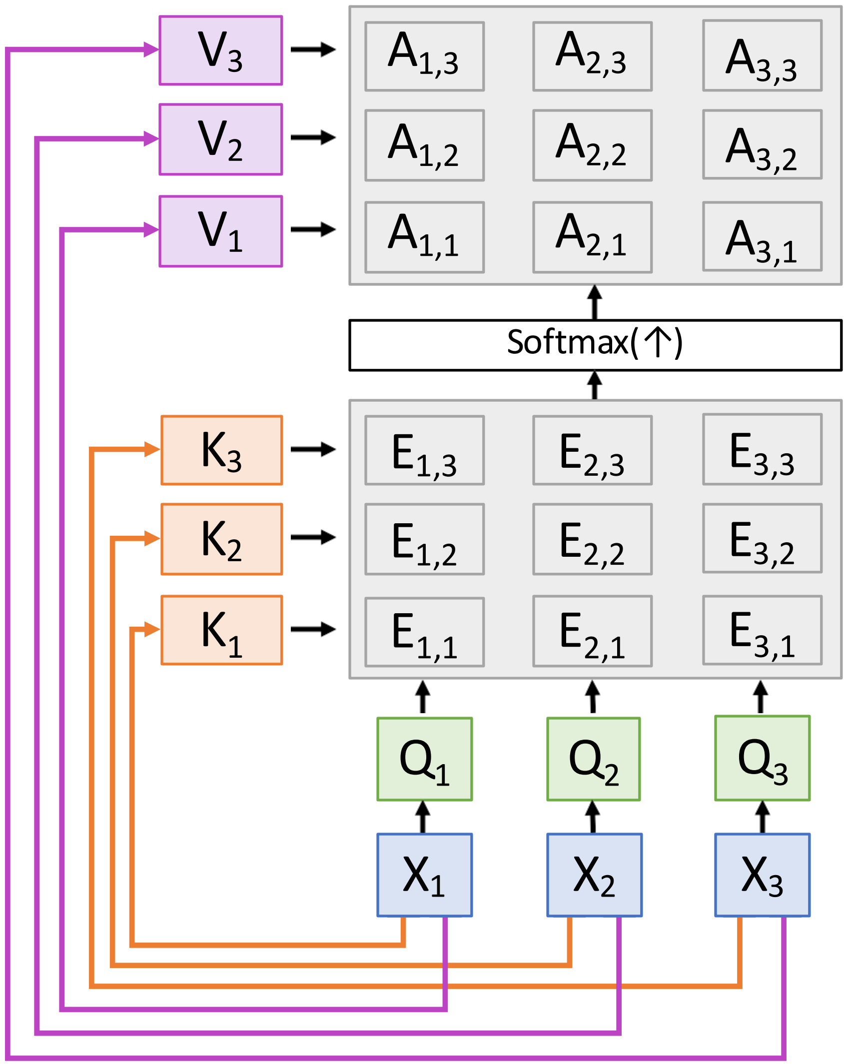
Self-Attention Layer

One **query** per **input vector**

- Inputs:
- Input vectors: X

(Shape: N_X x D_X) Key matrix:
W_K (Shape: D_X x D_Q) Value matrix:
W_V (Shape: D_X x D_V) Query matrix:
W_Q (Shape: D_X x D_Q)

- Computation:
- Query vectors: $Q = XW_Q$
- Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
- Value Vectors: $V = XW_V$ (Shape: N_X x D_V)
- Similarities: $E = QK^T$ (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j$



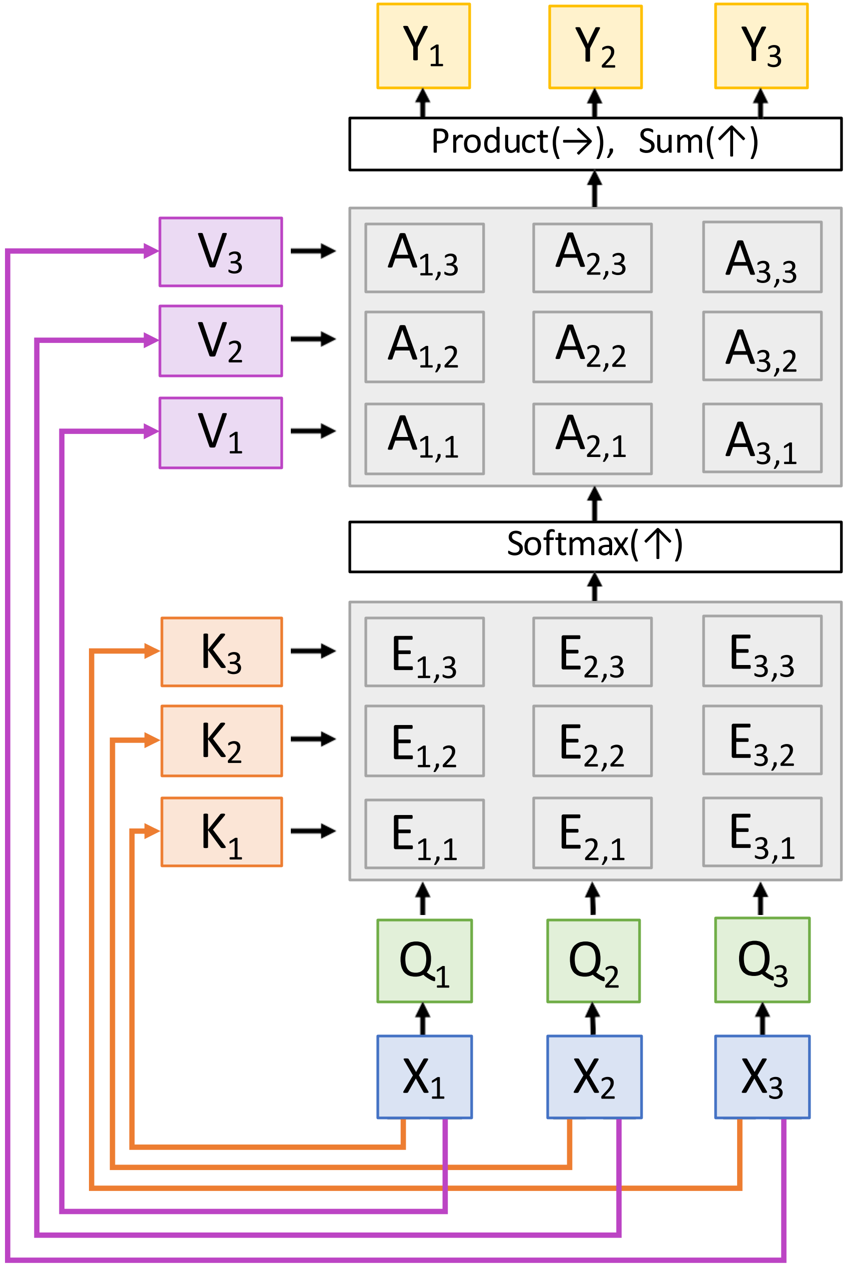
Monsoon,

Self-Attention Layer

One **query** per **input vector**

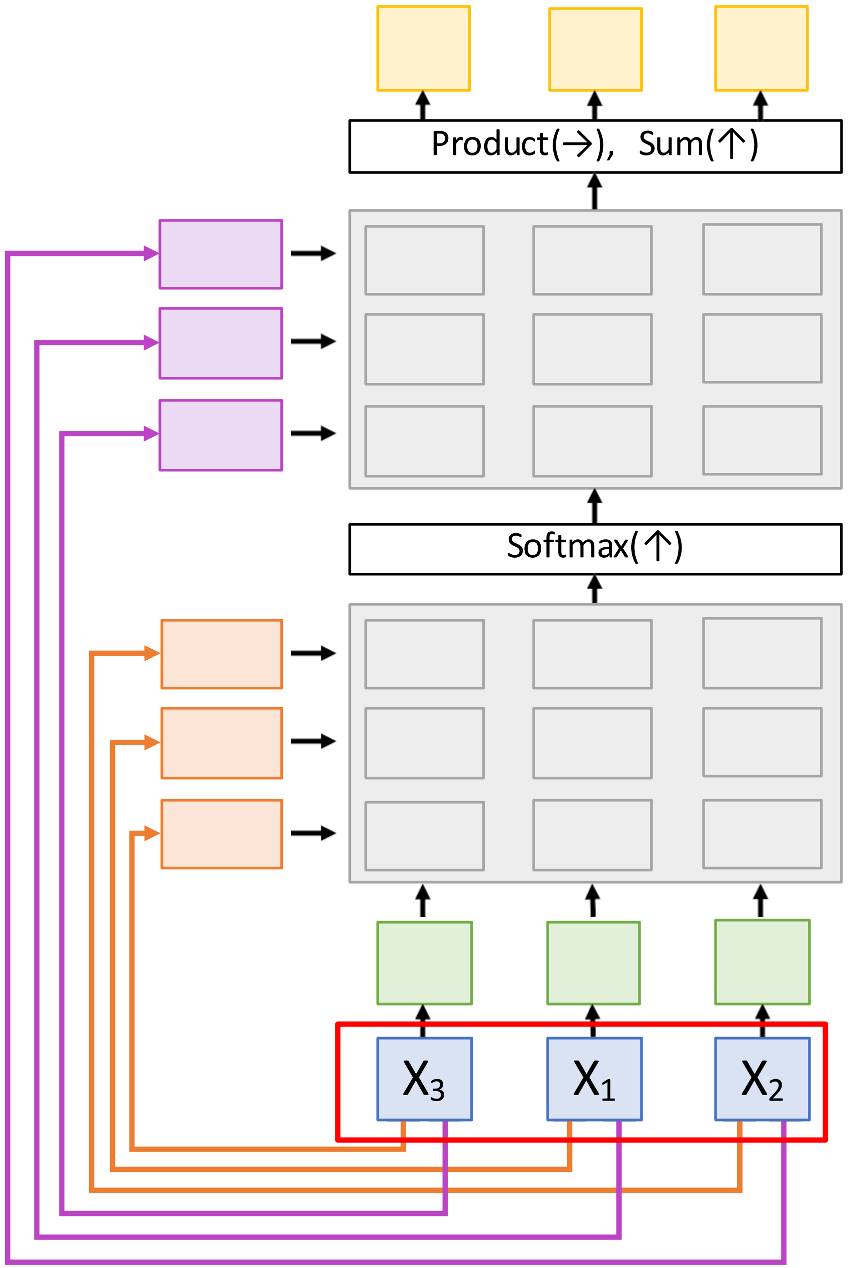
<u>**Inputs**</u>:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

<u>**Computation**</u>:
Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{QK}^T$ (Shape: N_X x N_X) $E_{i,j} = \mathbf{Q_i} \cdot \mathbf{K_j} / \mathrm{sqrt}(D_Q)$
Attention weights: $A = \mathrm{softmax}(E, \mathrm{dim}=1)$ (Shape: N_X x N_X)
Output vectors: $Y = A\mathbf{V}$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$



Self-Attention Layer

Consider **permuting** the input vectors:



Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

Computation:
Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{QK^T}$ (Shape: N_X x N_X) $E_{i,j} = \mathbf{Q_i} \cdot \mathbf{K_j} / \mathrm{sqrt}(D_Q)$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: $Y = A\mathbf{V}$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$

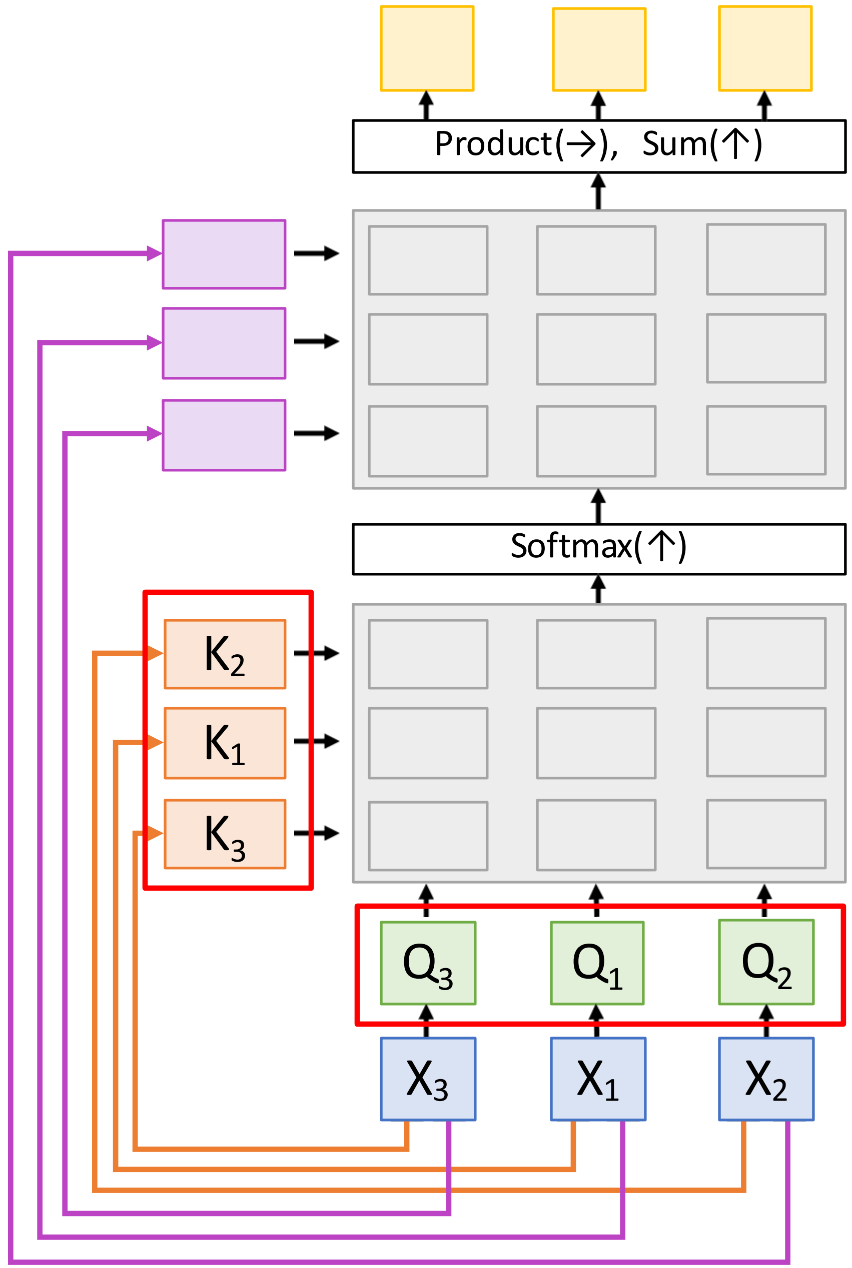
Self-Attention Layer

Consider **permuting** the input vectors:

Queries and Keys will be the same, but permuted

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: **W_K** (Shape: D_X x D_Q)
Value matrix: **W_V** (Shape: D_X x D_V)
Query matrix: **W_Q** (Shape: D_X x D_Q)

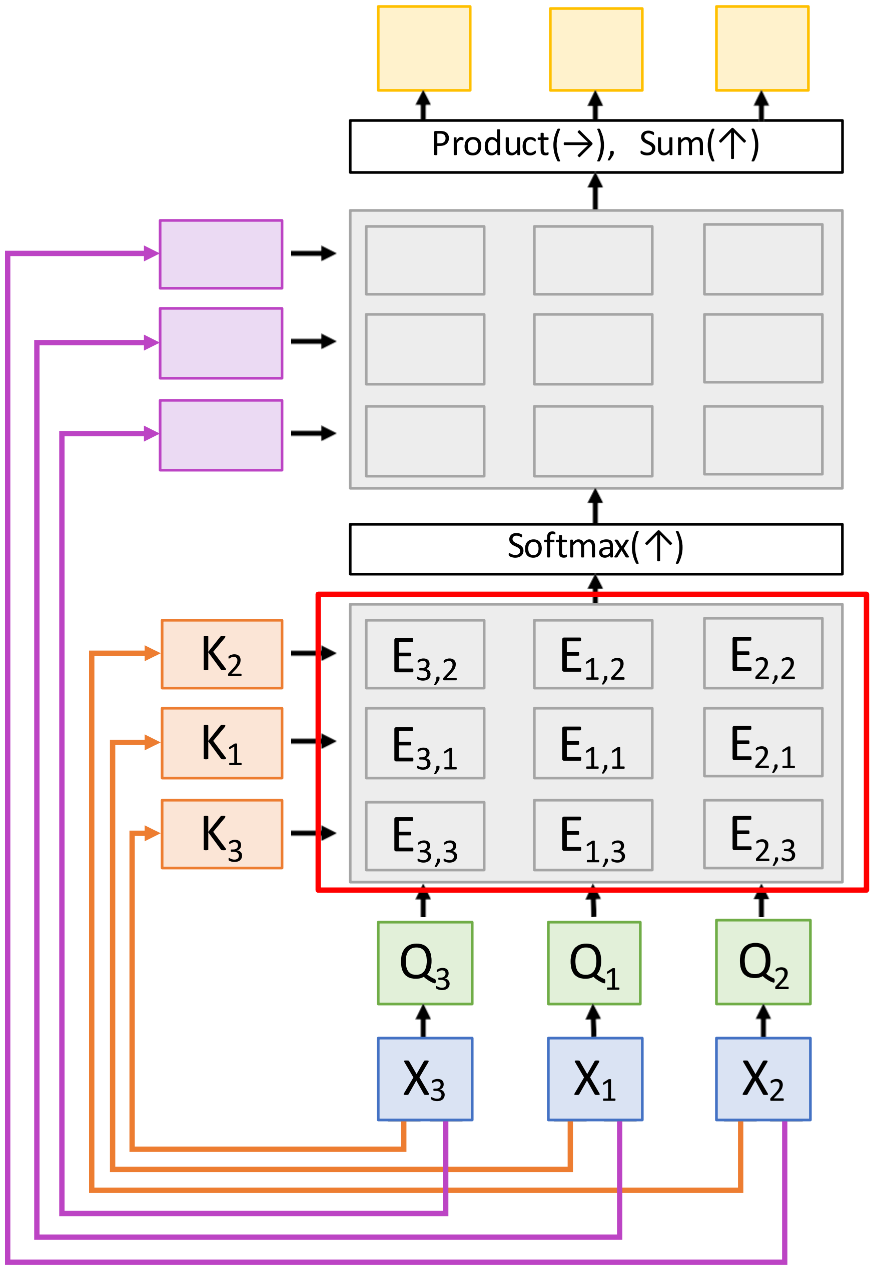
Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value Vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T$ (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j / \mathrm{sqrt}(D_Q)$
Attention weights: $A = \mathrm{softmax}(E, \mathrm{dim}=1)$ (Shape: N_X x N_X)
Output vectors: $Y = AV$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$



Self-Attention Layer

Consider **permuting** the input vectors:

Similarities will be the same, but permuted



Inputs:

Input vectors: X (Shape: N_X x D_X)

Key matrix: W_K (Shape: D_X x D_Q)

Value matrix: W_V (Shape: D_X x D_V)

Query matrix: W_Q (Shape: D_X x D_Q)

Computation:

Query vectors: $Q = XW_Q$

Key vectors: $K = XW_K$ (Shape: N_X x D_Q)

Value Vectors: $V = XW_V$ (Shape: N_X x D_V)

Similarities: $E = QK^T$ (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j / \mathrm{sqrt}(D_Q)$

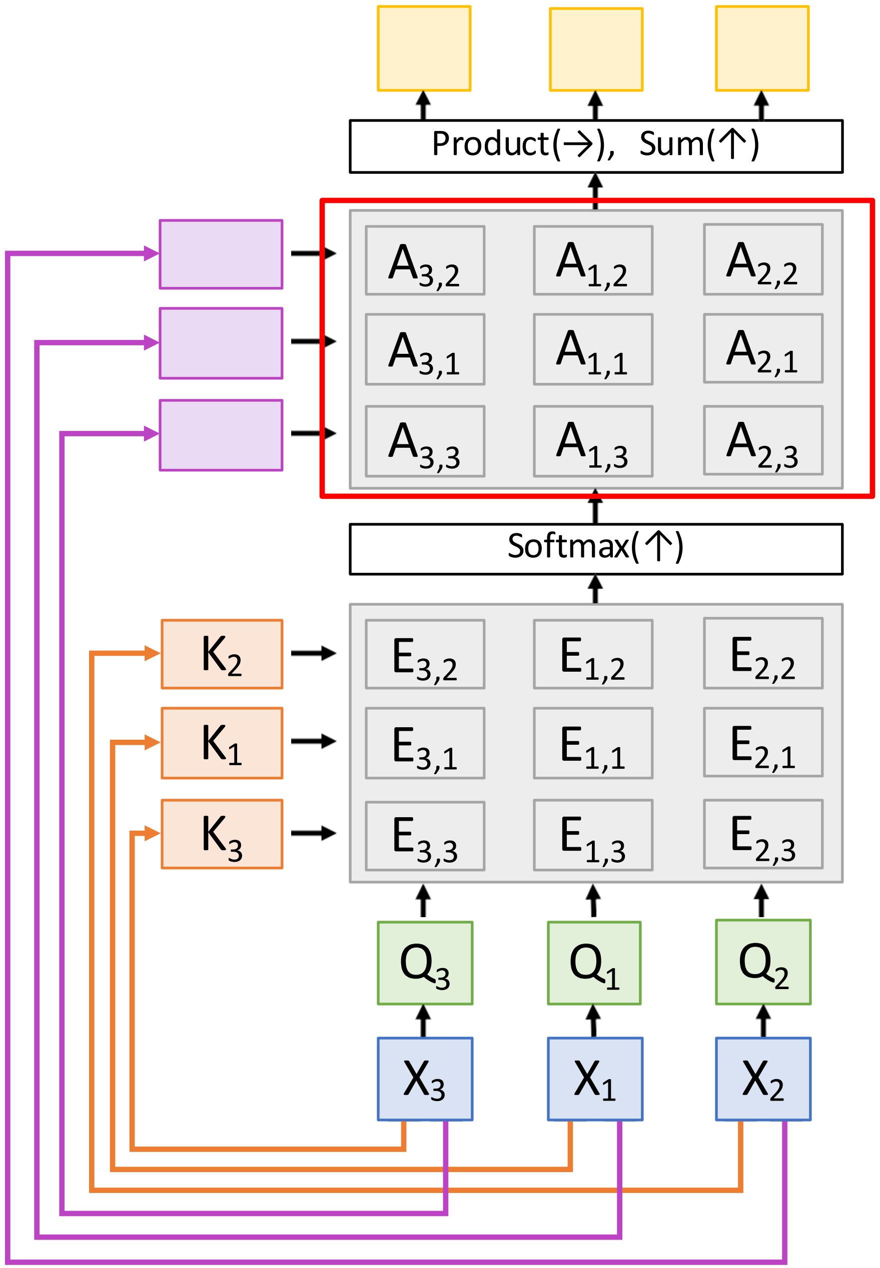
Attention weights: $A = \mathrm{softmax}(E, \mathrm{dim}=1)$ (Shape: N_X x N_X)

Output vectors: $Y = AV$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$

Self-Attention Layer

Consider **permuting** the input vectors:

Attention weights will be the same, but permuted



<u>**Inputs**</u>:

Input vectors: **X** (Shape: N_X x D_X)

Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)

Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)

Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

<u>**Computation**</u>:

Query vectors: **Q** = $\mathbf{XW_Q}$

Key vectors: **K** = $\mathbf{XW_K}$ (Shape: N_X x D_Q)

Value Vectors: **V** = $\mathbf{XW_V}$ (Shape: N_X x D_V)

Similarities: E = $\mathbf{QK}^T$ (Shape: N_X x N_X) $E_{i,j} = \mathbf{Q_i} \cdot \mathbf{K_j} / \text{sqrt}(D_Q)$

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

Output vectors: Y = A**V** (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} \mathbf{V_j}$

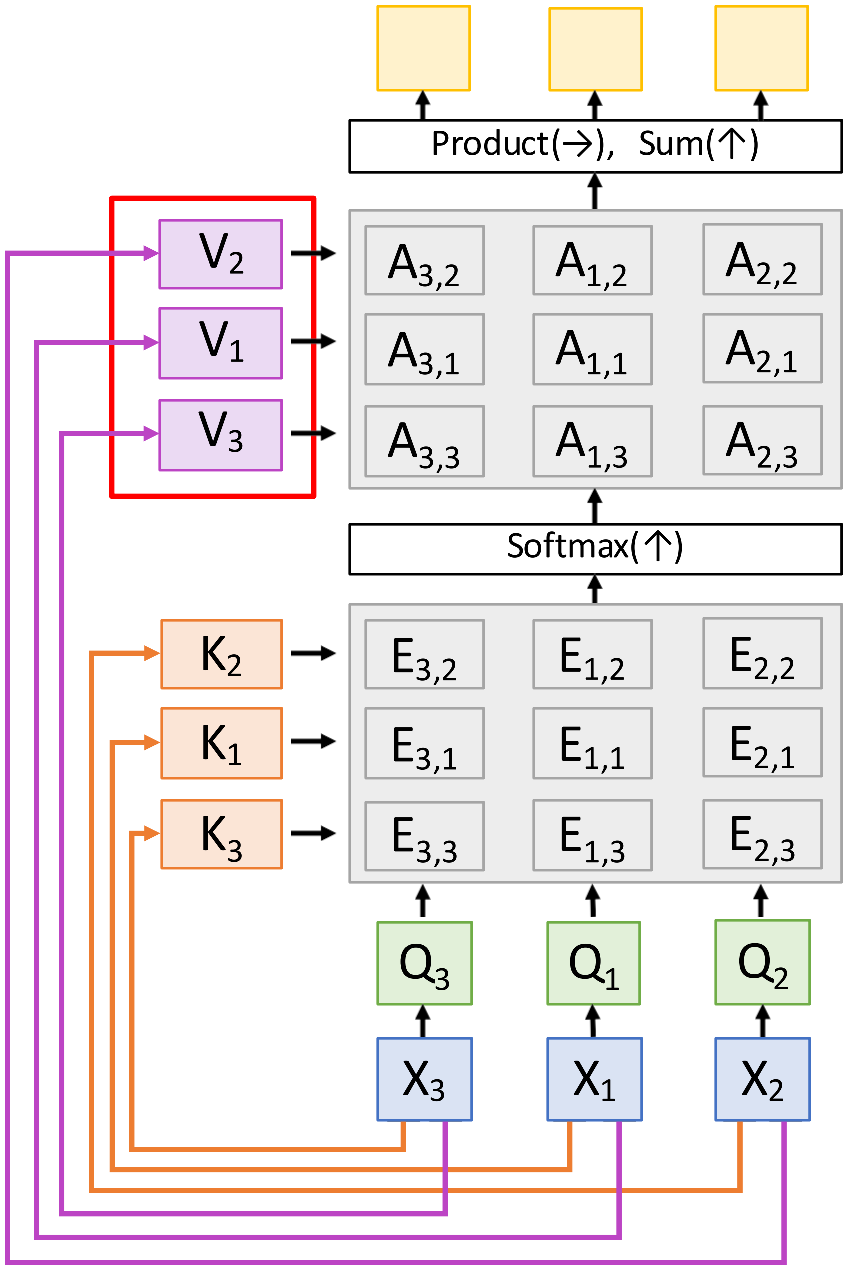
Self-Attention Layer

Consider **permuting** the input vectors:

Values will be the same, but permuted

<u>**Inputs**</u>:
Input vectors: X (Shape: $N_X \times D_X$)
Key matrix: W_K (Shape: $D_X \times D_Q$)
Value matrix: W_V (Shape: $D_X \times D_V$)
Query matrix: W_Q (Shape: $D_X \times D_Q$)

<u>**Computation**</u>:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: $N_X \times D_Q$)
Value Vectors: $V = XW_V$ (Shape: $N_X \times D_V$)
Similarities: $E = QK^T$ (Shape: $N_X \times N_X$) $E_{i,j} = Q_i \cdot K_j / \mathrm{sqrt}(D_Q)$
Attention weights: $A = \mathrm{softmax}(E, \mathrm{dim}=1)$ (Shape: $N_X \times N_X$)
Output vectors: $Y = AV$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} V_j$



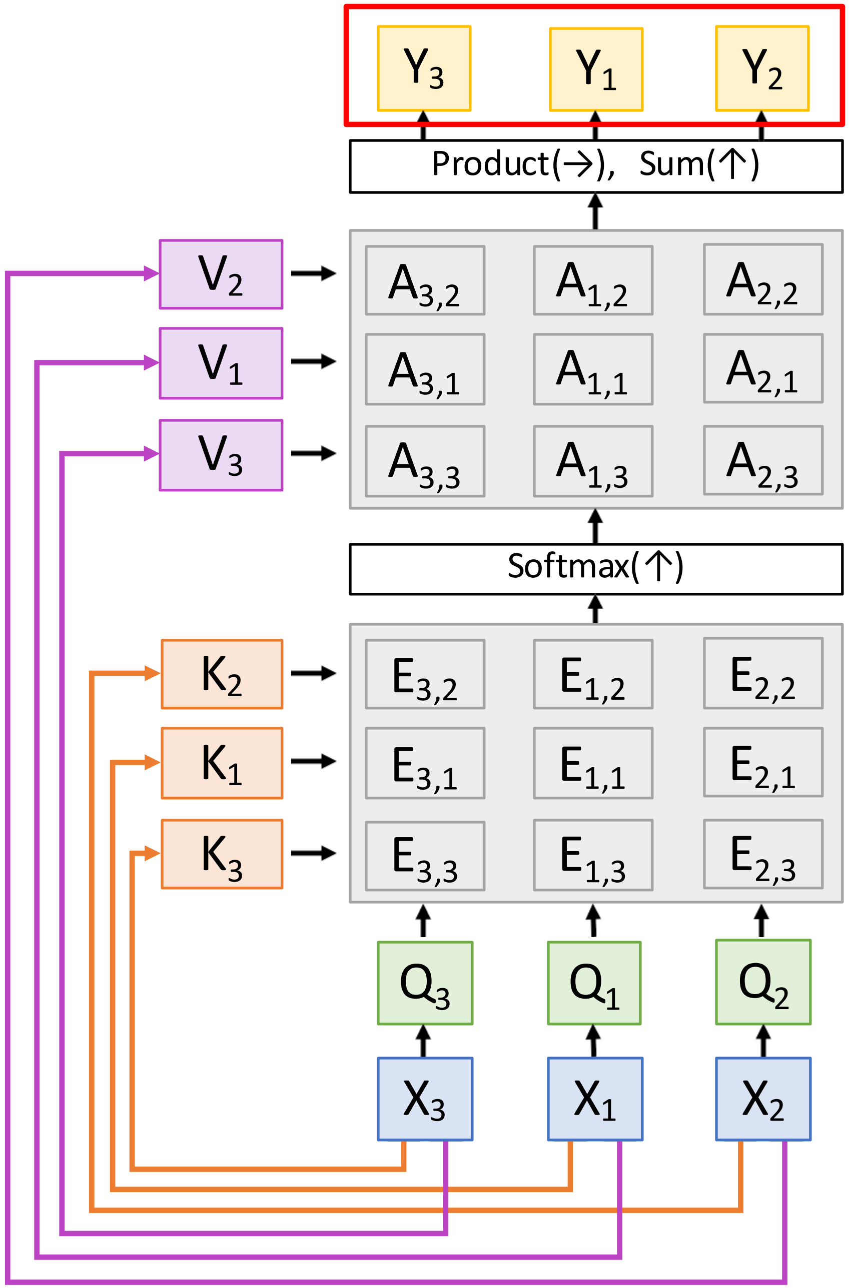
Self-Attention Layer

Consider **permuting** the input vectors:

Outputs will be the same, but permuted

<u>**Inputs**</u>:
Input vectors: **X** (Shape: $N_X \times D_X$)
Key matrix: $\mathbf{W_K}$ (Shape: $D_X \times D_Q$)
Value matrix: $\mathbf{W_V}$ (Shape: $D_X \times D_V$)
Query matrix: $\mathbf{W_Q}$ (Shape: $D_X \times D_Q$)

<u>**Computation**</u>:
Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: $N_X \times D_Q$)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: $N_X \times D_V$)
Similarities: $E = \mathbf{QK}^T$ (Shape: $N_X \times N_X$) $E_{i,j} = \mathbf{Q_i} \cdot \mathbf{K_j} / \mathrm{sqrt}(D_Q)$
Attention weights: $A = \mathrm{softmax}(E, \mathrm{dim}=1)$ (Shape: $N_X \times N_X$)
Output vectors: $Y = A\mathbf{V}$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$



Self-Attention Layer

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: **W_K** (Shape: D_X x D_Q)
Value matrix: **W_V** (Shape: D_X x D_V)
Query matrix: **W_Q** (Shape: D_X x D_Q)

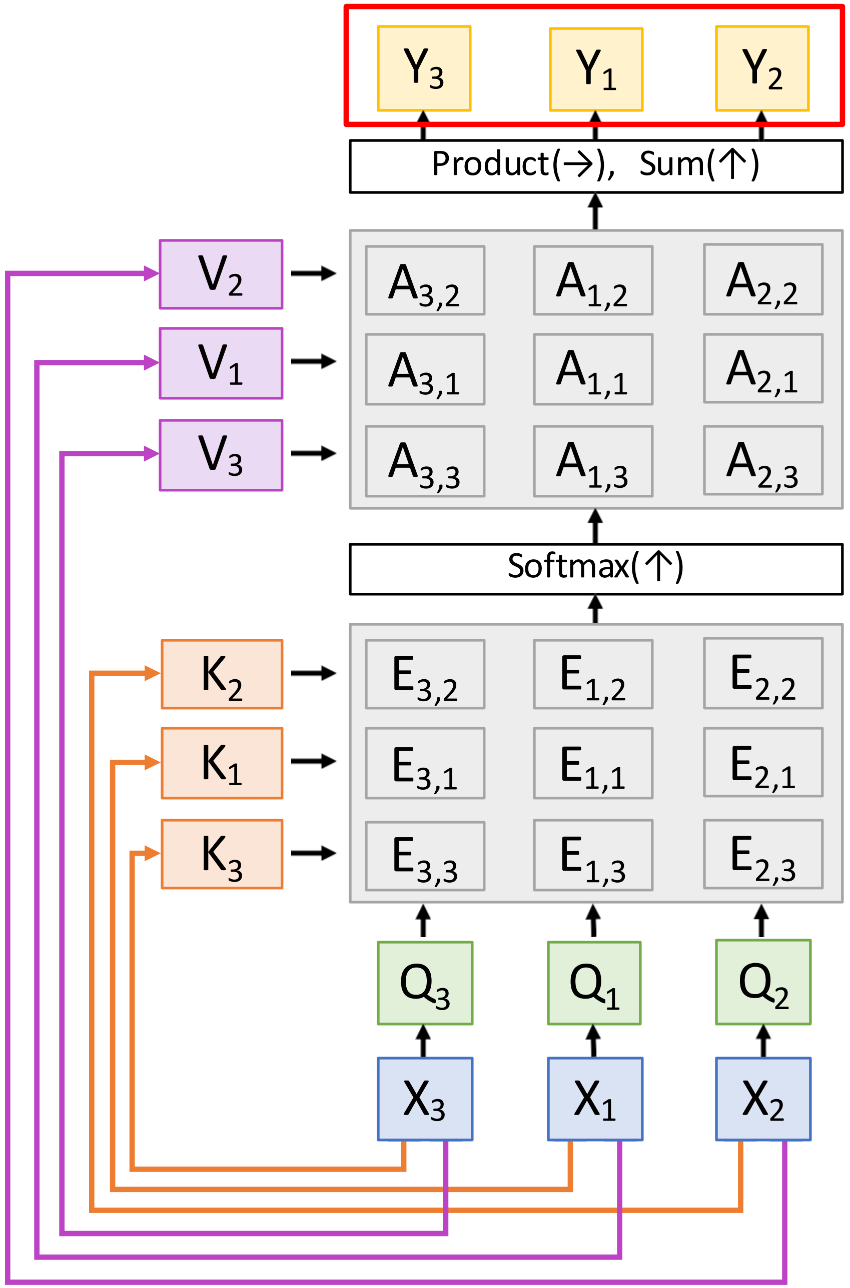
Computation:
Query vectors: **Q** = **XW_Q**
Key vectors: **K** = **XW_K** (Shape: N_X x D_Q)
Value Vectors: **V** = **XW_V** (Shape: N_X x D_V)
Similarities: E = QK^T (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j / \mathrm{sqrt}(D_Q)$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: Y = AV (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$

Consider **permuting** the input vectors:

Outputs will be the same, but permuted

Self-attention layer is **Permutation Equivariant** f(s(x)) = s(f(x))

Self-Attention layer works on **sets** of vectors

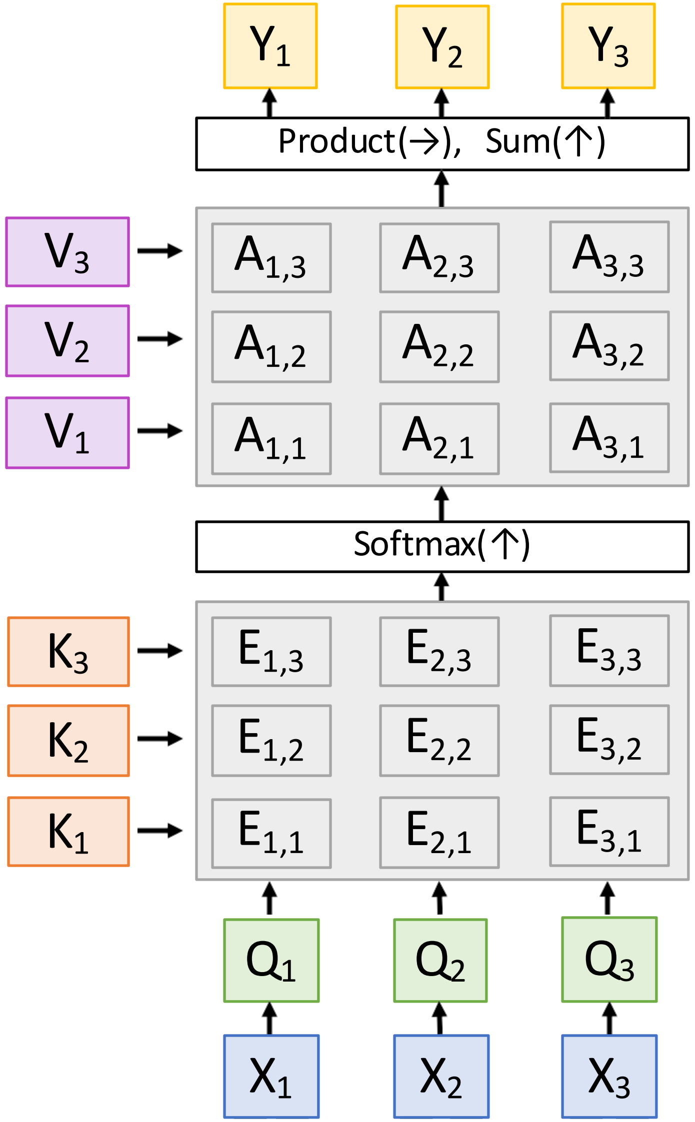


Self-Attention Layer

Self attention doesn't "know" the order of the vectors it is processing!

Inputs:
Input vectors: $\mathbf{X}$ (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

Computation:
Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{QK^T}$ (Shape: N_X x N_X) $E_{i,j} = \mathbf{Q_i} \cdot \mathbf{K_j} / \text{sqrt}(D_Q)$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: N_X x N_X)
Output vectors: $Y = A\mathbf{V}$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$



Self-Attention Layer

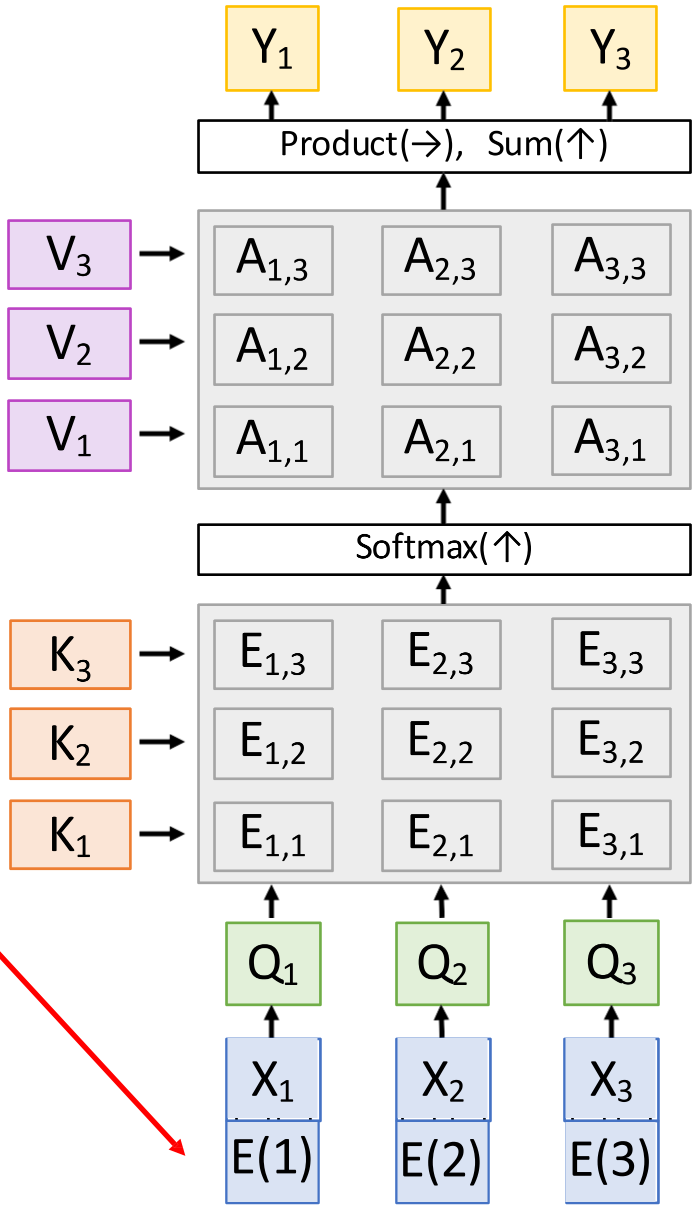
Self attention doesn’t “know” the order of the vectors it is processing!

In order to make processing position-aware, concatenate input with **positional encoding**

E can be learned lookup table, or fixed function

<u>Inputs</u>:
Input vectors: X (Shape: N_X x D_X)
Key matrix: W_K (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)
Query matrix: W_Q (Shape: D_X x D_Q)

<u>Computation</u>:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value Vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T$ (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j / \mathrm{sqrt}(D_Q)$
Attention weights: $A = \mathrm{softmax}(E, \mathrm{dim}=1)$ (Shape: N_X x N_X)
Output vectors: $Y = AV$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$

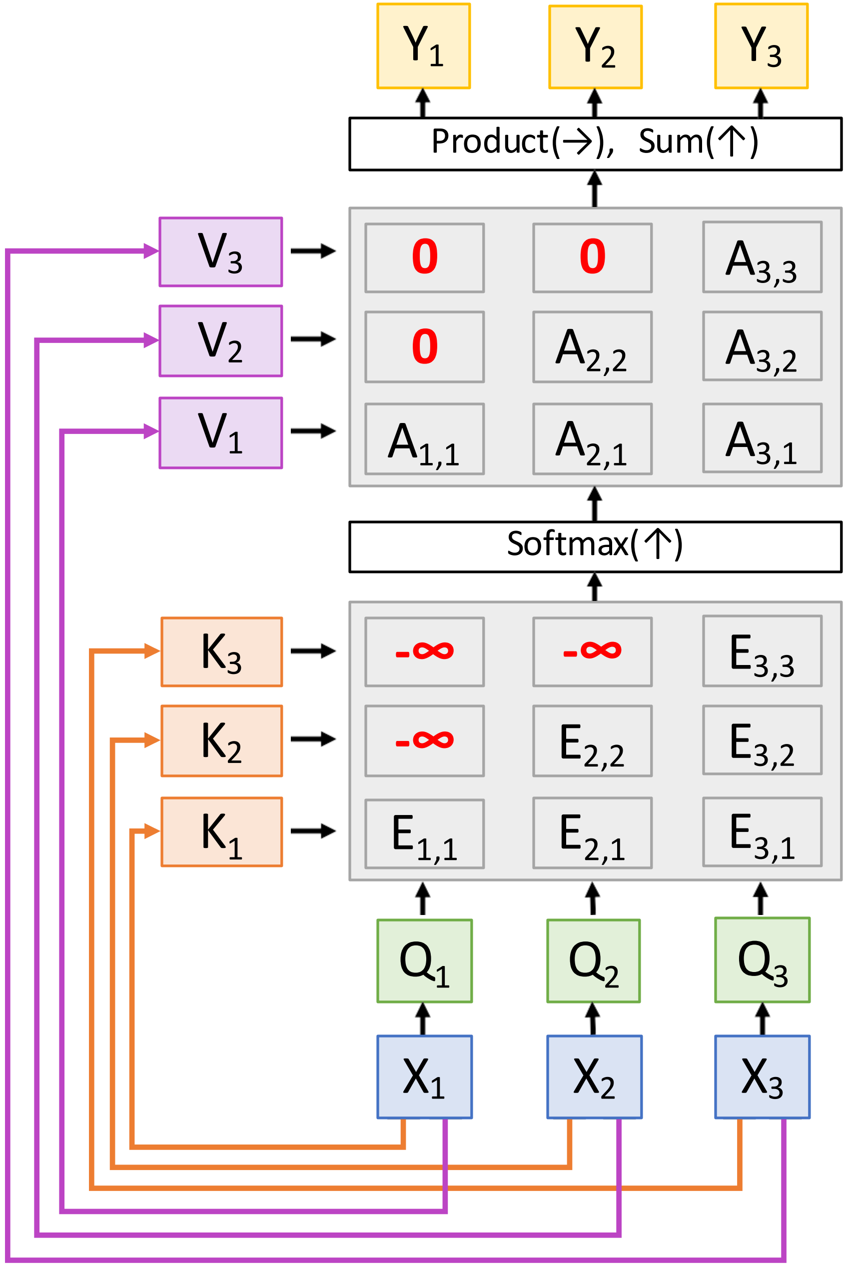


Masked Self-Attention Layer

Don't let vectors "look ahead" in the sequence

Inputs:
Input vectors: X (Shape: N_X x D_X)
Key matrix: W_K (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)
Query matrix: W_Q (Shape: D_X x D_Q)

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value Vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T$ (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j / \mathrm{sqrt}(D_Q)$
Attention weights: $A = \mathrm{softmax}(E, \mathrm{dim}=1)$ (Shape: N_X x N_X)
Output vectors: $Y = AV$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$

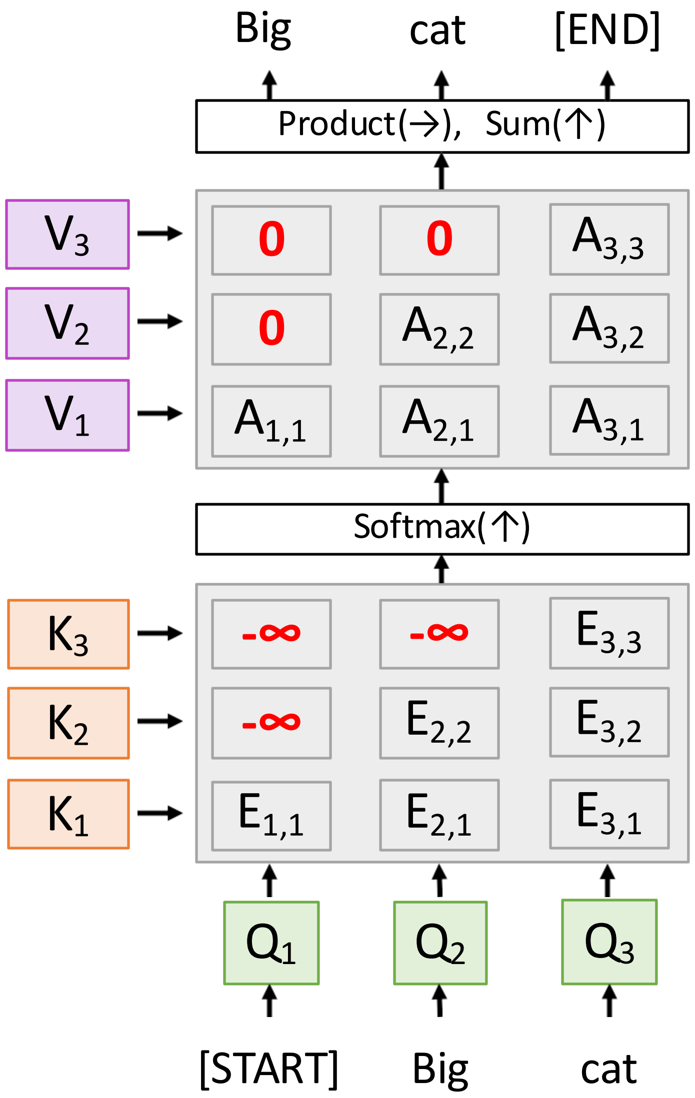


Masked Self-Attention Layer

Don't let vectors "look ahead" in the sequence
Used for language modeling (predict next word)

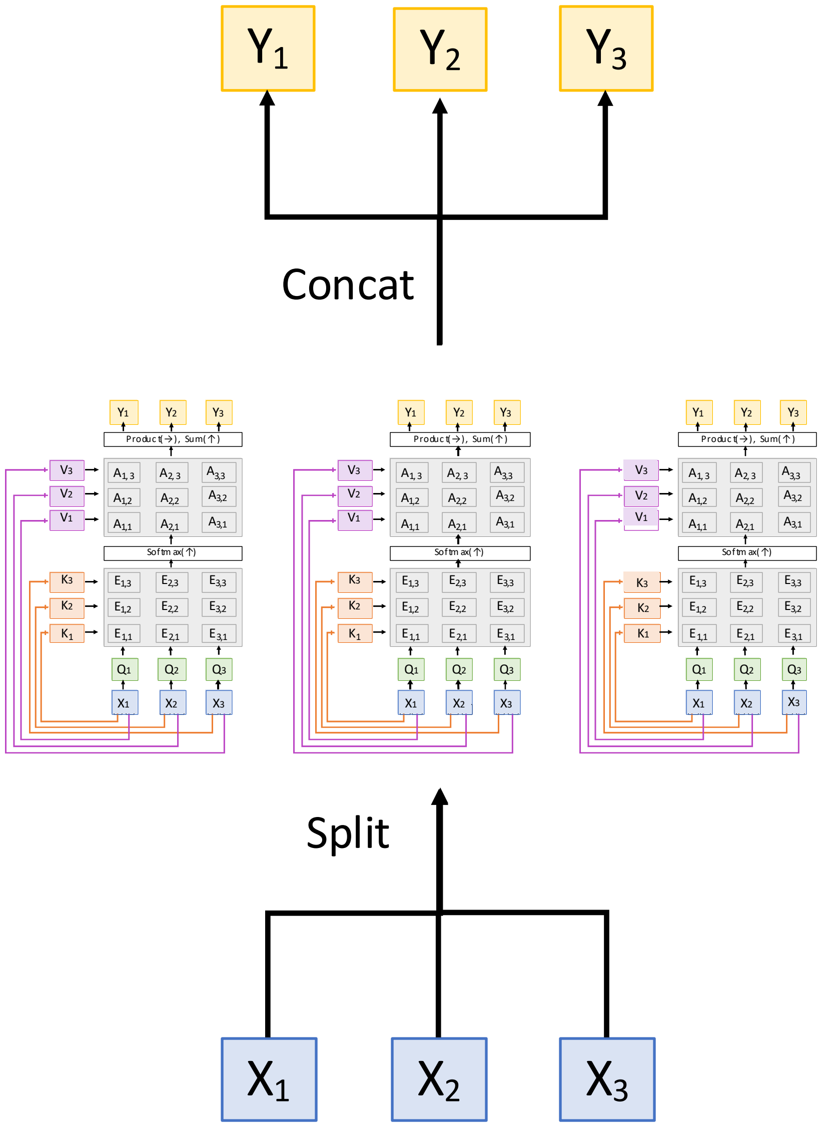
<u>Inputs</u>:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: W_K (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)
Query matrix: W_Q (Shape: D_X x D_Q)

<u>Computation</u>:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value Vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T$ (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j / \mathrm{sqrt}(D_Q)$
Attention weights: $A = \mathrm{softmax}(E, \mathrm{dim}=1)$ (Shape: N_X x N_X)
Output vectors: $Y = AV$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$



Multihead Self-Attention Layer

Use H independent
"Attention Heads" in parallel



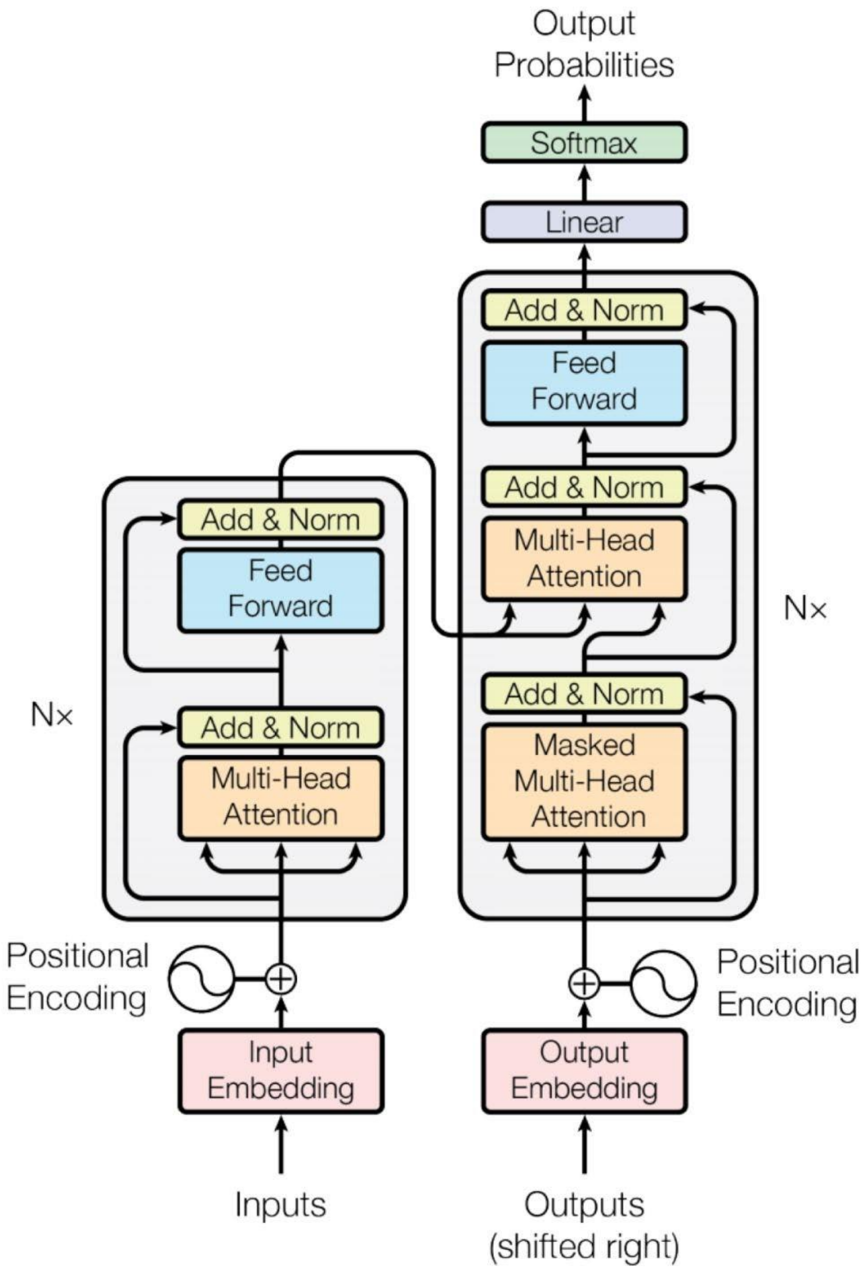
Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

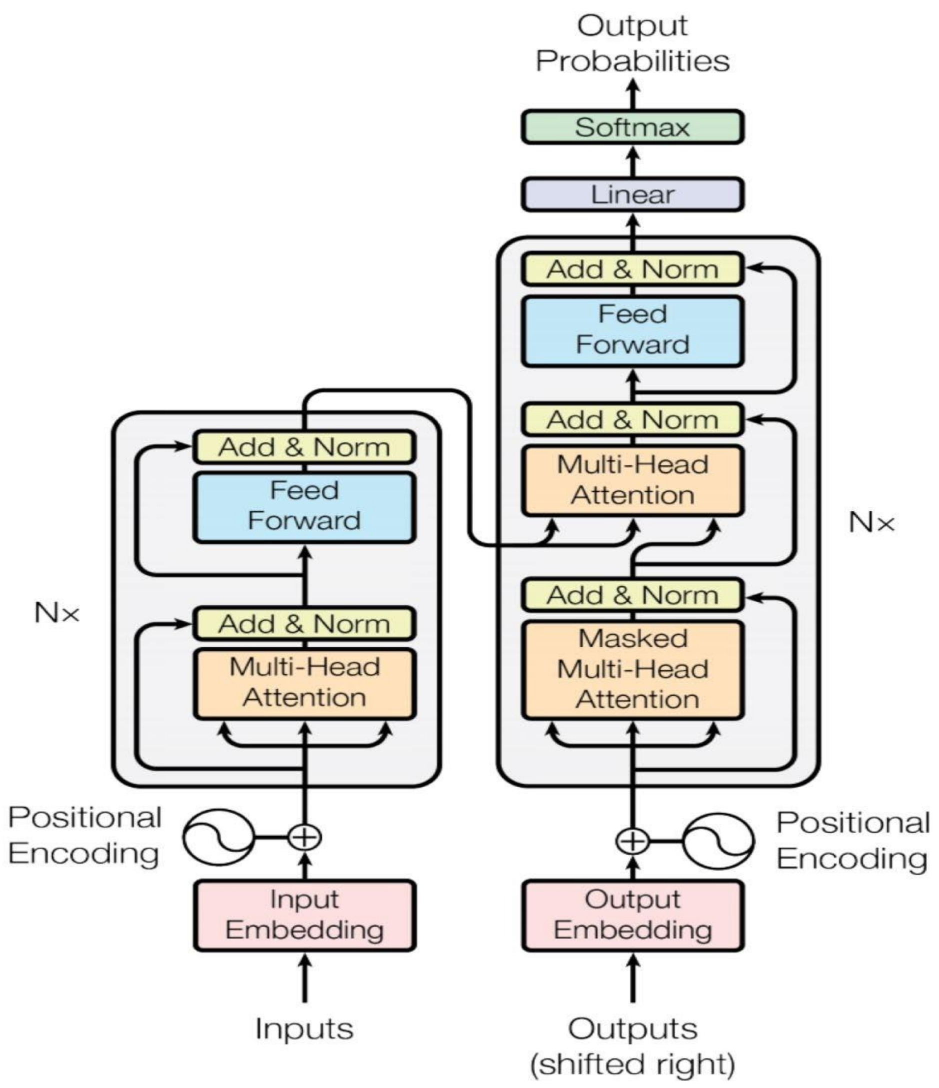
Computation:
Query vectors: $\mathbf{Q} = \mathbf{XW_Q}$
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)
Similarities: $E = \mathbf{QK}^T$ (Shape: N_X x N_X) $E_{i,j} = \mathbf{Q_i} \cdot \mathbf{K_j} / \text{sqrt}(D_Q)$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: $Y = A\mathbf{V}$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$

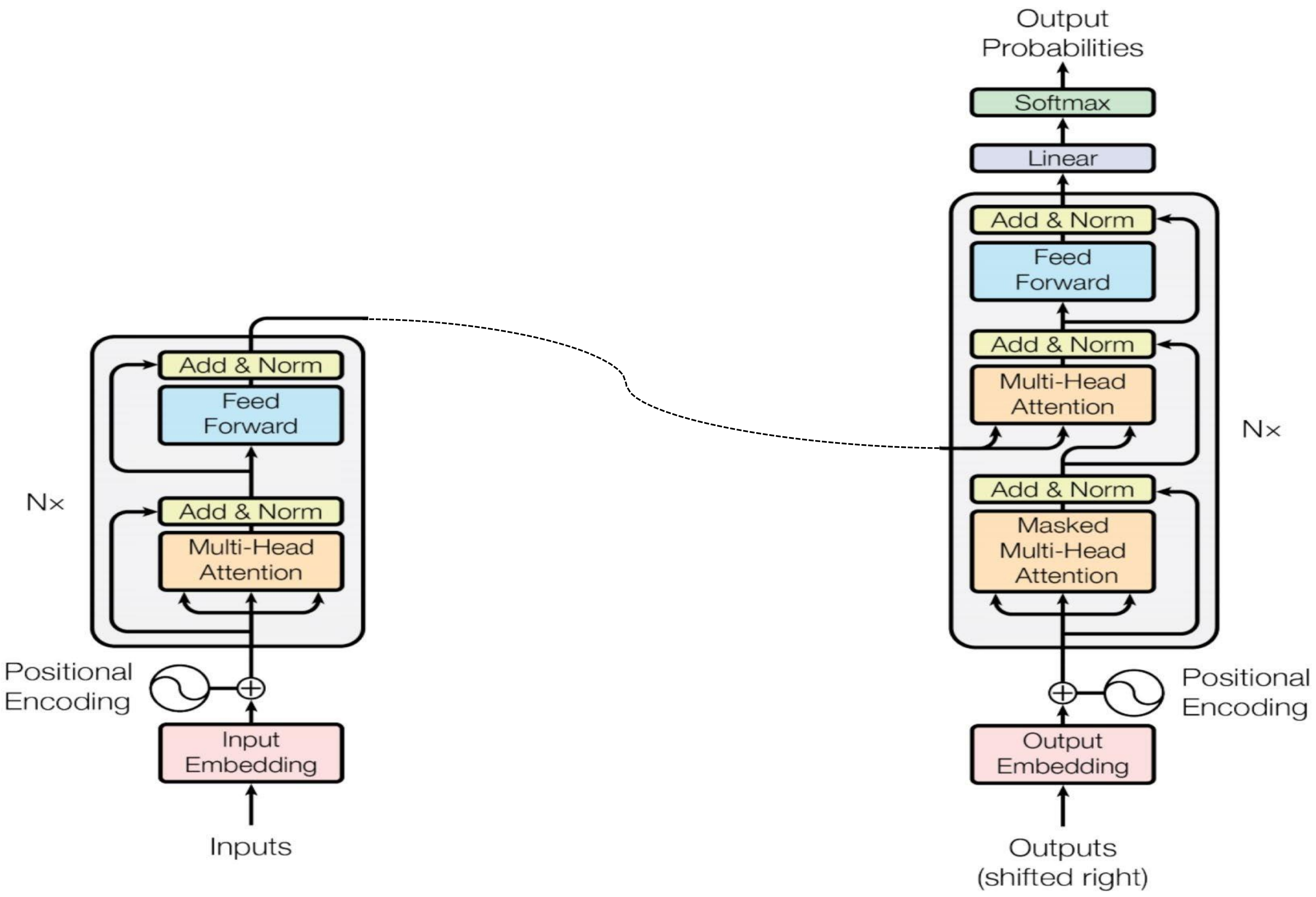
Monsoon,

Vision Transformers

Attention Is All You Need (2017)



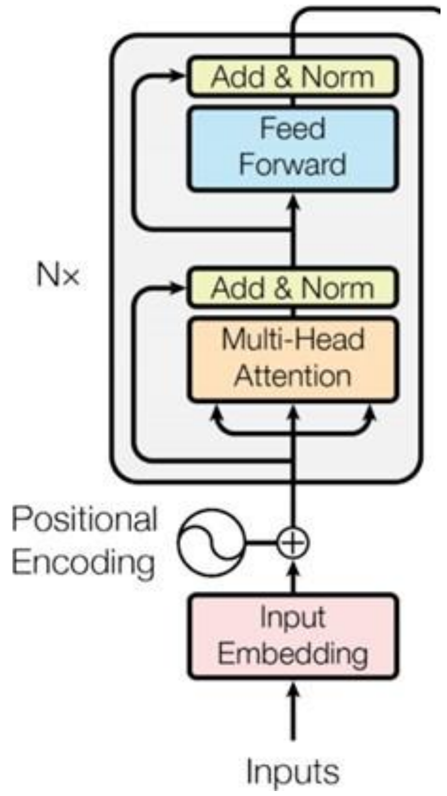

Output
Probabilities
Softmax
Linear
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Add & Norm
Masked
Multi-Head
Attention
Nx
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Nx
Positional
Encoding
Positional
Encoding
Input
Embedding
Output
Embedding
Inputs
Outputs
(shifted right)



Encoder Decoder

Encoder

The encoder part of the transformer embeds the input sequence of n words $X \in \mathbb{R}^{d \times n}$ into context vectors with the attention mechanism.



Encoder

- The encoder consists of two main components: Self-Attention and Feedforward Neural Network (FFN).
- **Self-Attention:**
  - Input: Matrix X
  - Linear Transformations to generate Query (Q), Key (K), and Value (V) matrices:

$$Q = W_Q^T X, \quad K = W_K^T X, \quad V = W_V^T X$$

  - Compute attention output Z using the formula:

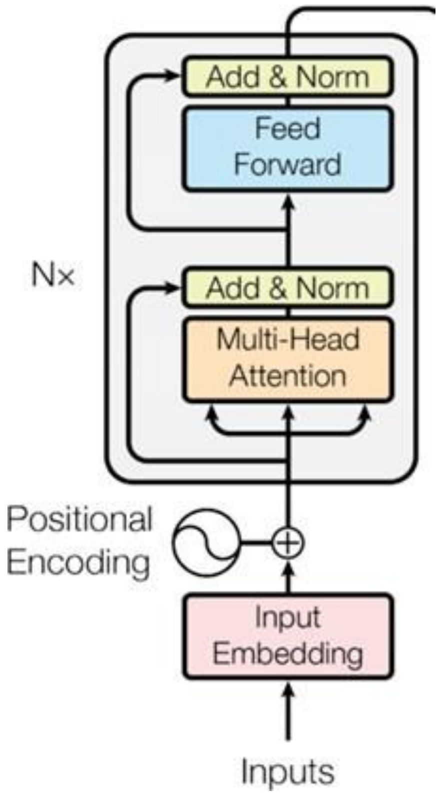
$$Z = V\mathrm{softmax}\left(\frac{Q^T K}{\sqrt{p}}\right)$$

  - Residual Connection:

$$X + Z$$

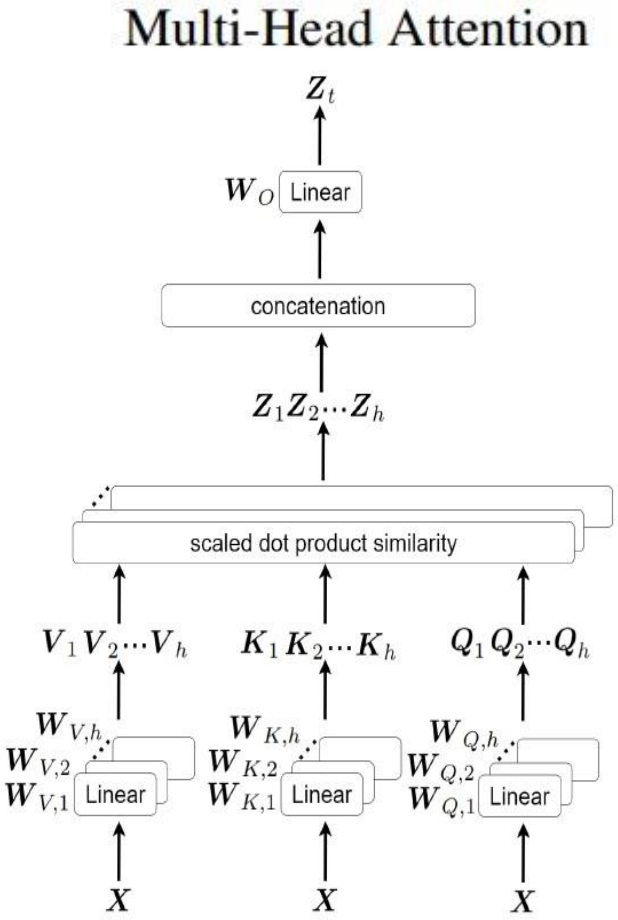
  - Normalization:

$$(X + Z)$$



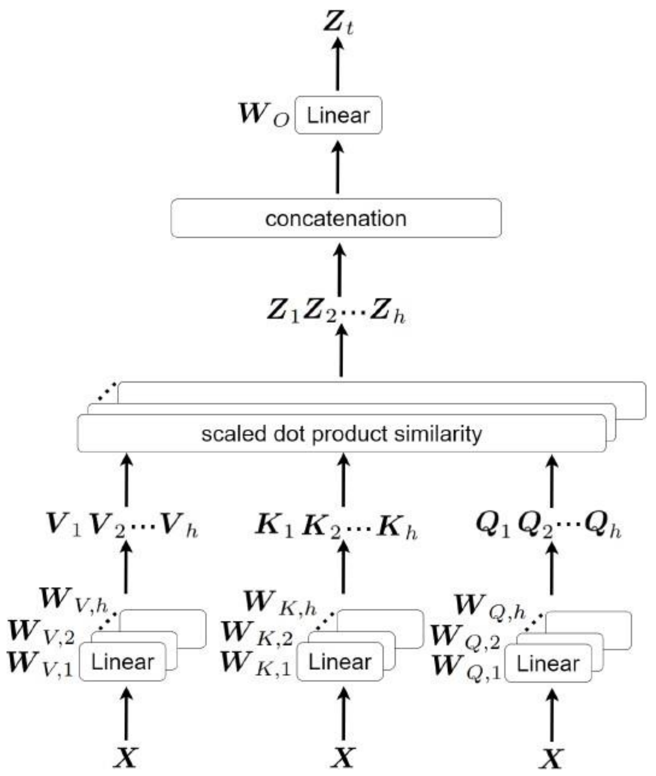
Multi-Headed Attention

$$Q_1 = {W_Q^1}^T X$$
$$K_1 = {W_K^1}^T X$$
$$V_1 = {W_V^1}^T X$$
$$Z_1 = V \mathrm{softmax}\left(\frac{1}{\sqrt{p}} Q_1^T K_1\right)$$
$$\vdots$$
$$Q_h = {W_Q^h}^T X$$
$$K_h = {W_K^h}^T X$$
$$V_h = {W_V^h}^T X$$
$$Z_h = V \mathrm{softmax}\left(\frac{1}{\sqrt{p}} Q_h^T K_h\right)$$



Multi-Headed Attention

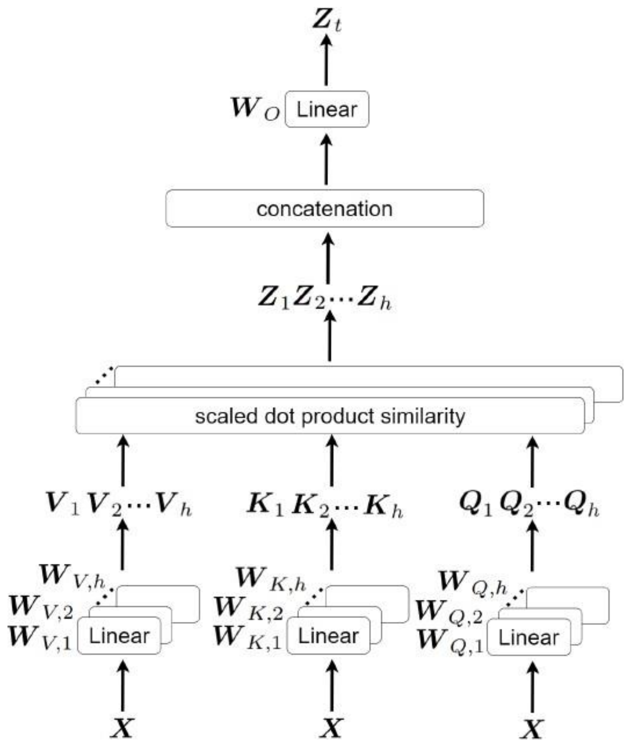
$$\begin{pmatrix} Z_1 \\ Z_2 \\ \vdots \\ Z_h \end{pmatrix} = \text{Concat}(head_1, \ldots, head_h)$$



Multi-Headed Attention

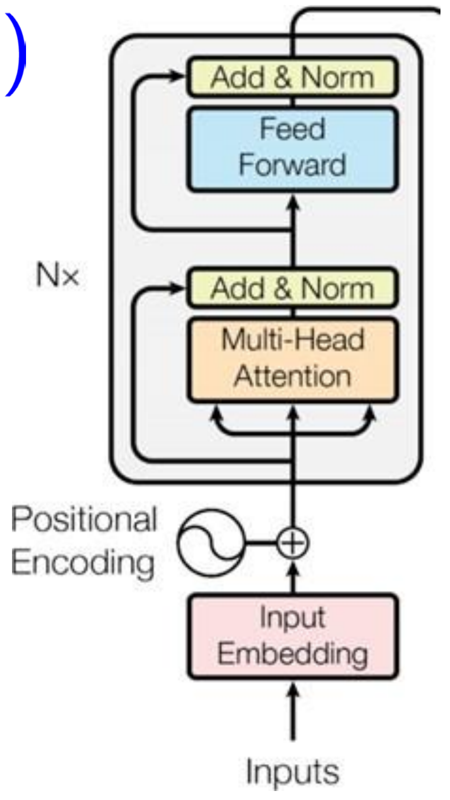
$$\begin{pmatrix} Z_1 \\ Z_2 \\ \vdots \\ Z_h \end{pmatrix} = \text{Concat}(head_1, \ldots, head_h)$$

$$Z = \text{MultiHead}(Q, K, V) = {W_0}^T \begin{pmatrix} Z_1 \\ Z_2 \\ \vdots \\ Z_h \end{pmatrix}$$



Multi-Headed Attention

$$\text{MultiHead}(Q, K, V) = {W_O}^T \text{Concat}(head_1, \ldots, head_h)$$



Structure of the Feed Forward Network

- Linear Layer 1
- ReLU Activation
- Linear Layer 2

$$FFN(x) = W_2^T \max(0, W_1^T X + b_1) + b_2$$

Two linear transformations with ReLU activation in between.

Application of FFN to Each Position

- The Feed Forward Network (FFN) is applied independently to each position in the input sequence.
- Despite individual processing, all positions share the same set of weights and biases in the FFN.
- Key Points:
  - Shared parameters ensure consistency in processing across all positions.
  - Enables the model to generalize learnings from one position to all positions.
  - Facilitates parallel processing of the sequence, enhancing computational efficiency.

Global vs Local

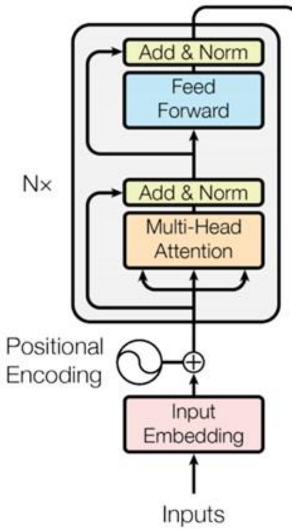
- Attention Mechanism:
- **Global Understanding**: Captures relationships among different positions in the sequence.
- **Context Aggregation**: Spreads relevant information across the sequence, enabling each position to see a broader context.

Global vs Local

- Attention Mechanism:
- **Global Understanding**: Captures relationships among different positions in the sequence.
- **Context Aggregation**: Spreads relevant information across the sequence, enabling each position to see a broader context.
- Feed-Forward Networks (FFN):
- **Local Processing**: While attention looks across the entire sequence, FFN zooms back in to process each position independently.
- **Individual Refinement**: Enhances the representation of each position based on its own value, refining the information gathered so far.

Encoder

If the output of the FFN is denoted by R, then a residual connection is established from the output of the previous layer $(X + Z)$ to the output of the FFN, resulting in $(X + Z) + R$.This will be normalized $((X + Z) + R)$to form the output of the encoder.



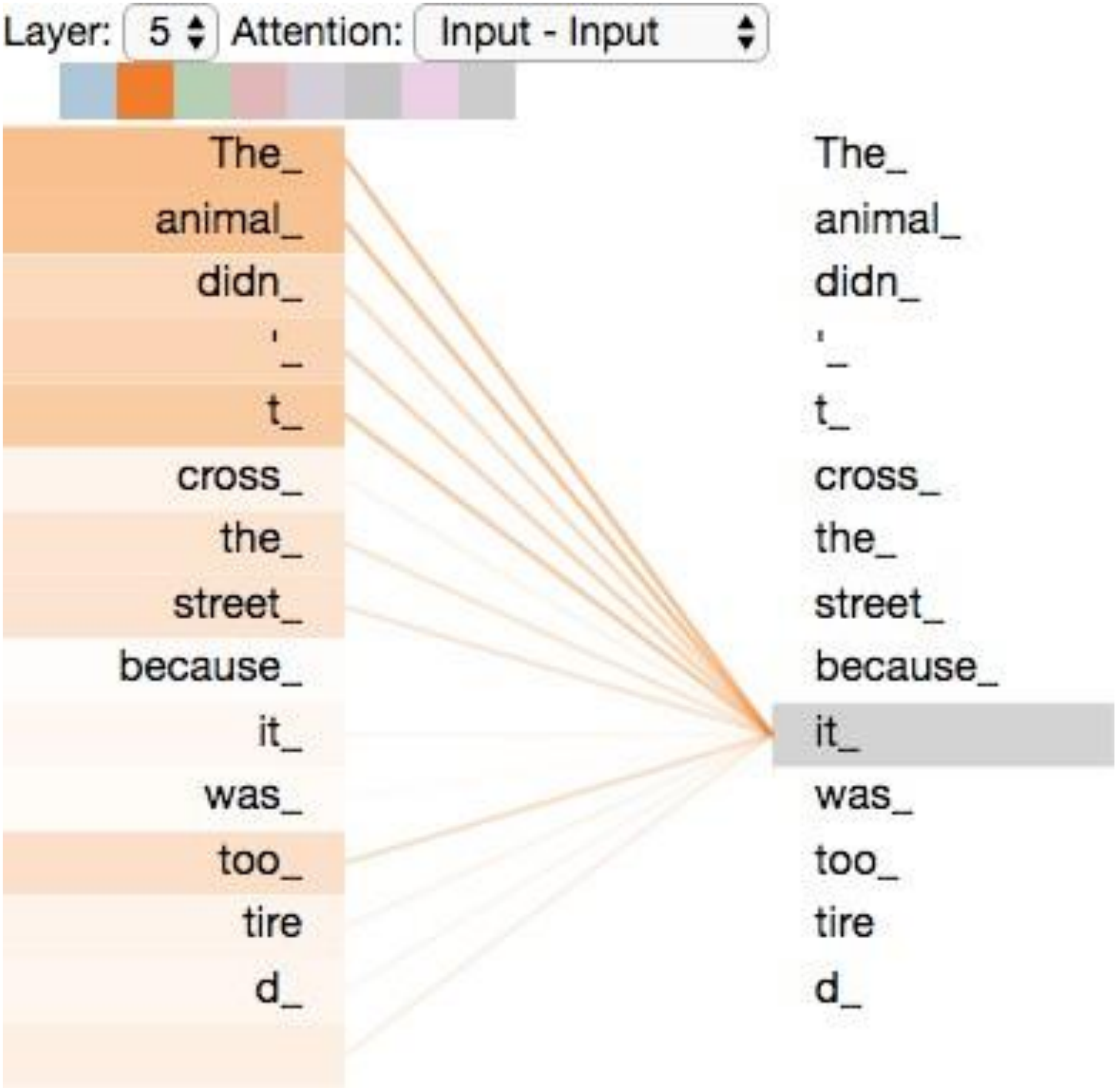


Figure: Jay Alammarz

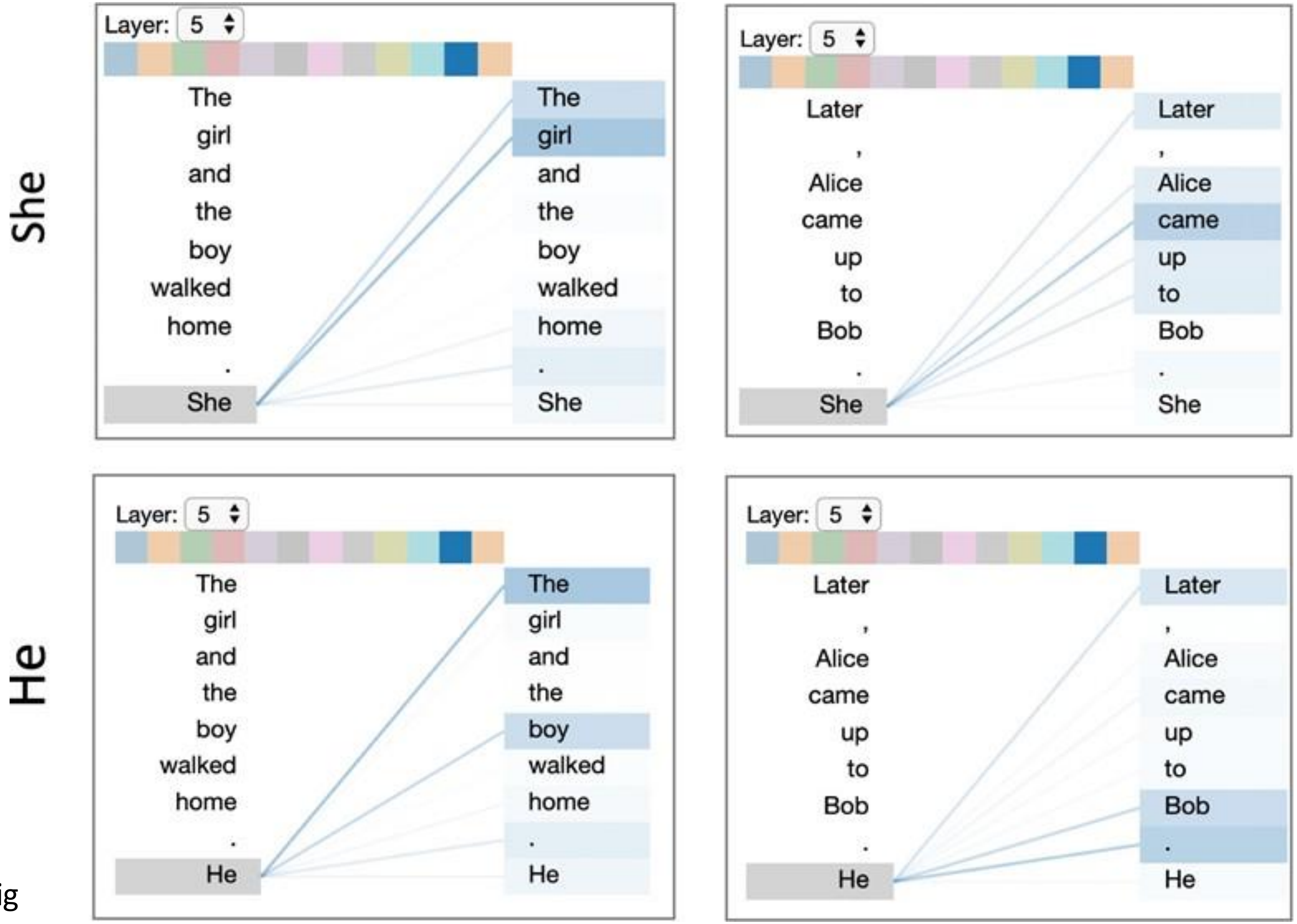
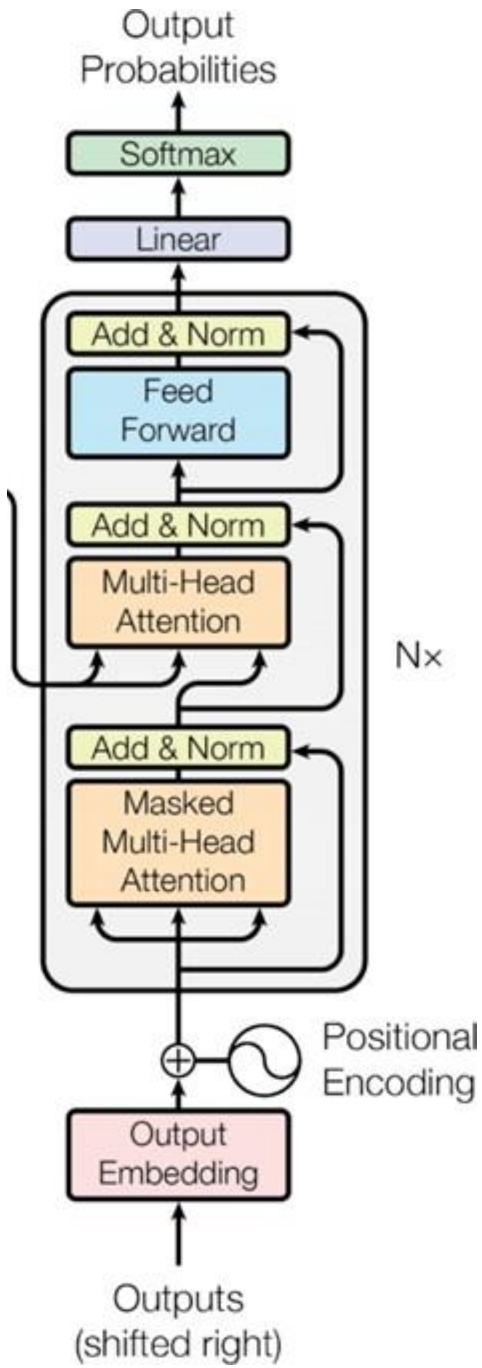


Figure: Jesse Vig

Decoder

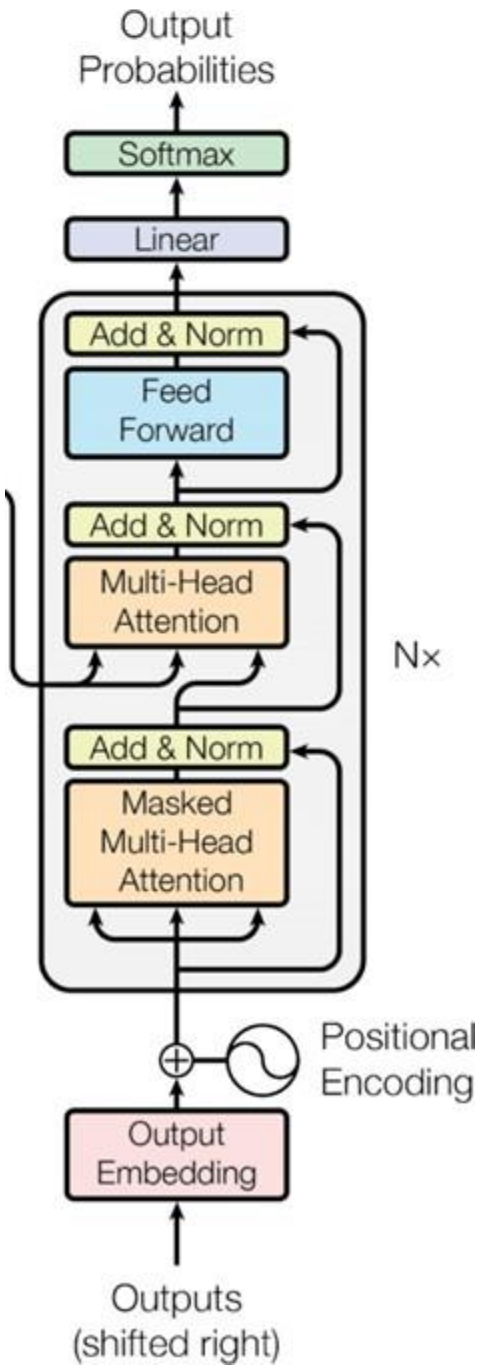
- **Masked self attention**.



Decoder

- **Masked self attention.**

$$Z := \mathrm{Attention}(Q, K, V) = V\mathrm{softmax}\left(\frac{1}{\sqrt{p}}(Q^\top K)\right),$$



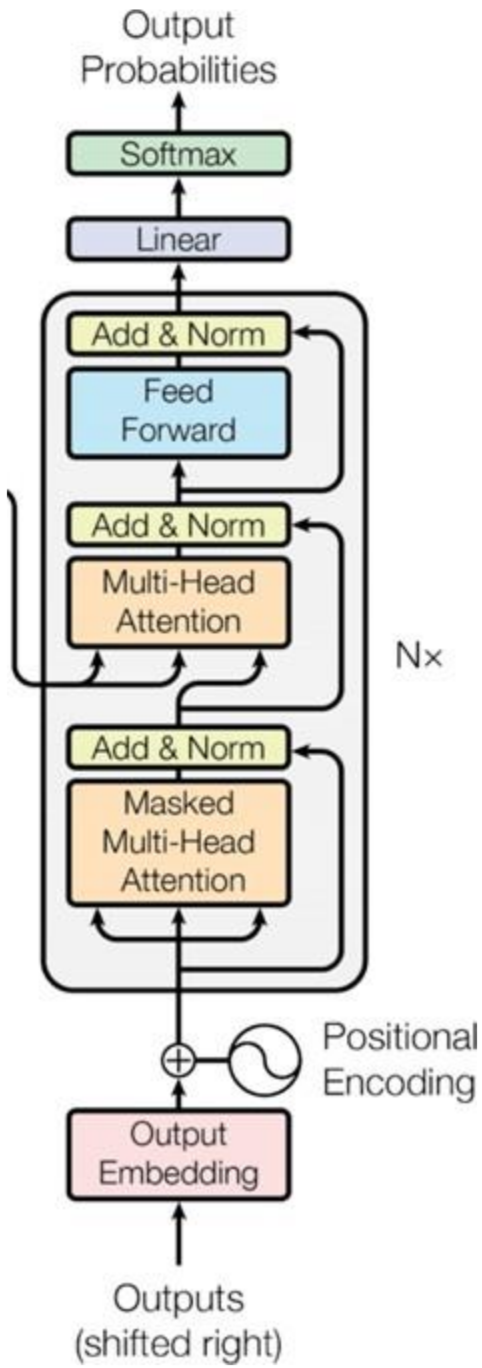
Decoder

- **Masked self attention.**

$$Z := \mathrm{maskedAttention}(Q, K, V) = V\mathrm{softmax}\left(\frac{1}{\sqrt{p}}(Q^\top K + M)\right),$$

where the mask matrix $M \in \mathbb{R}^{n\times n}$ is:

$$M(i,j) := \begin{cases} 0 & \text{if } j \leq i, \\ -\infty & \text{if } j > i. \end{cases}$$

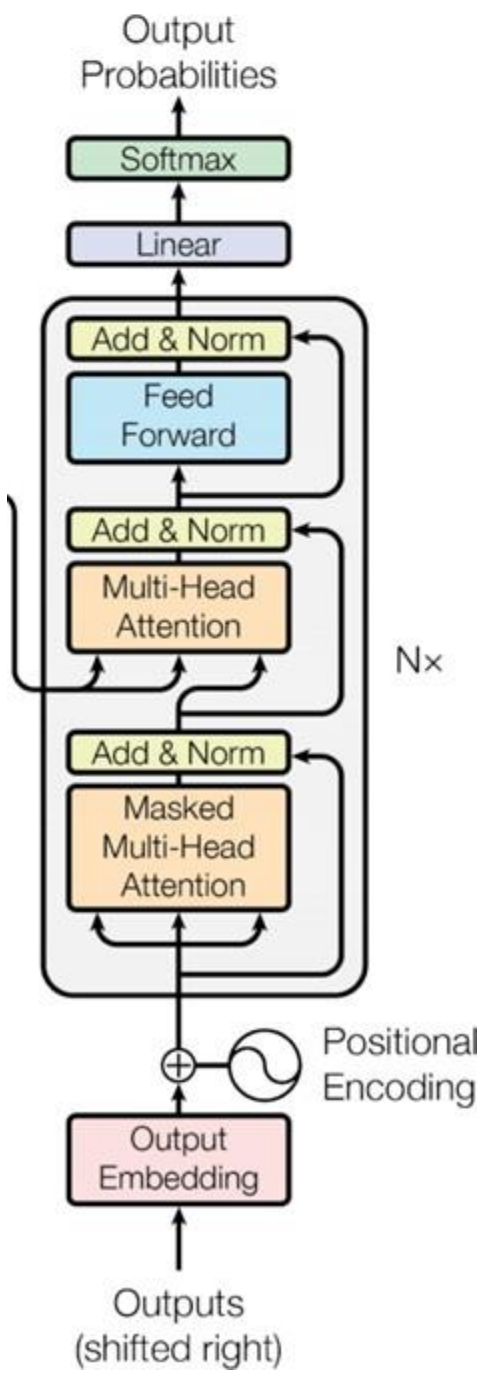


Decoder

- **Cross Attention**
- Cross attention allows each position in one sequence to attend over all positions in another sequence.

•**Query (Q)**: Originates from a position in the first sequence, i.e. the output of a previous layer in the decoder.

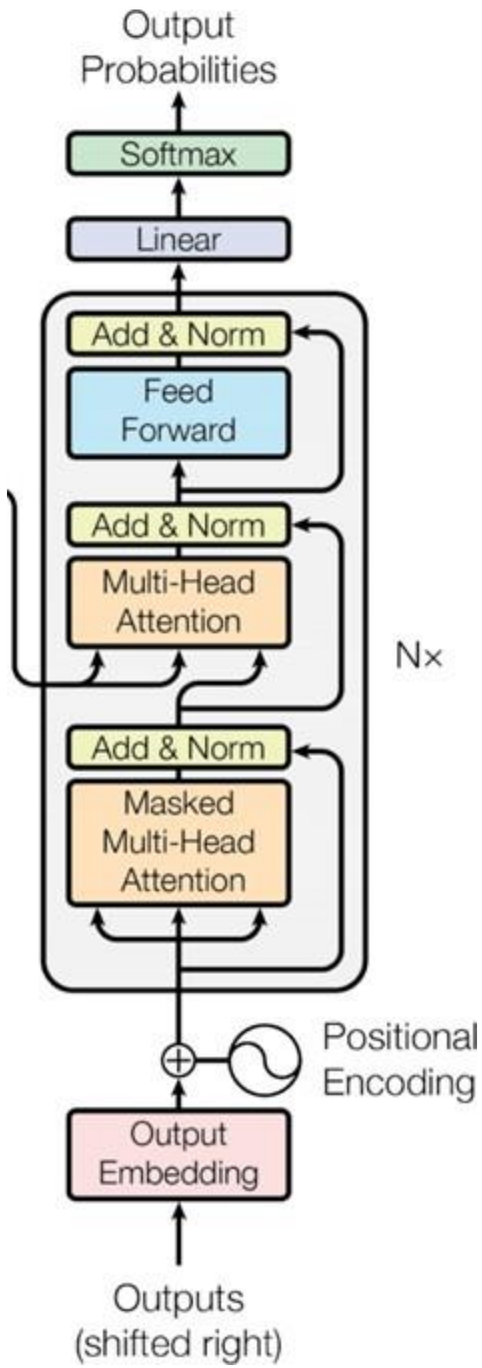
•**Memory Keys (K) and Values (V)**: Both come from all positions in the second sequence, i.e. the output of the encoder.



Decoder

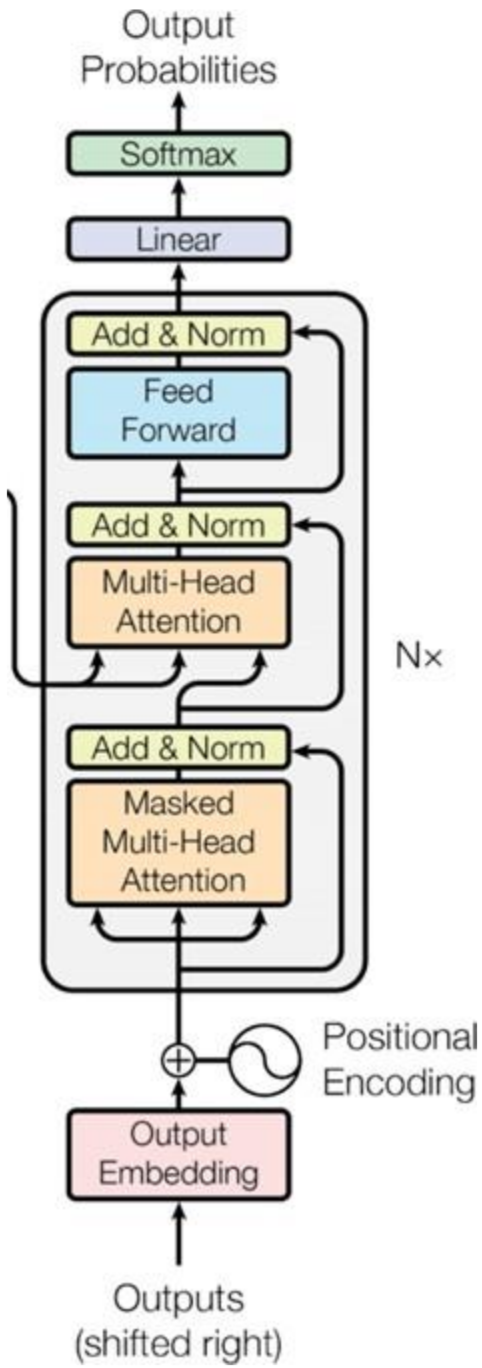
- Masked self attention.

- Cross attention layer is like what attention does in sequence-to-sequence models.



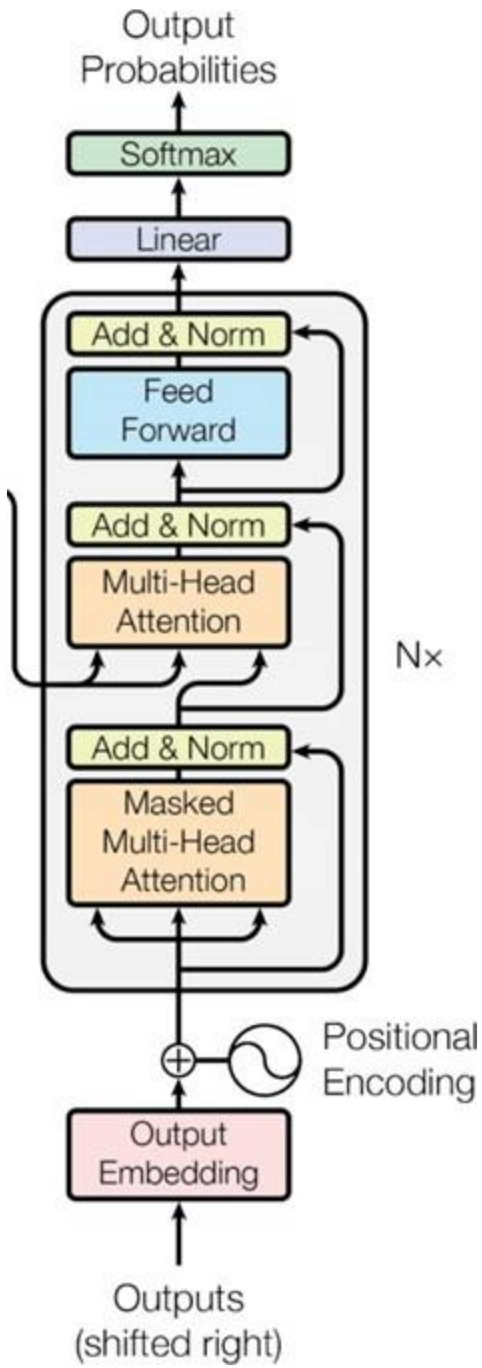
Decoder

- Masked self attention.
- Cross attention attention layer is like what attention does in sequence-to-sequence models.
- It helps the decoder emphasize on relevant parts of the input.



Decoder

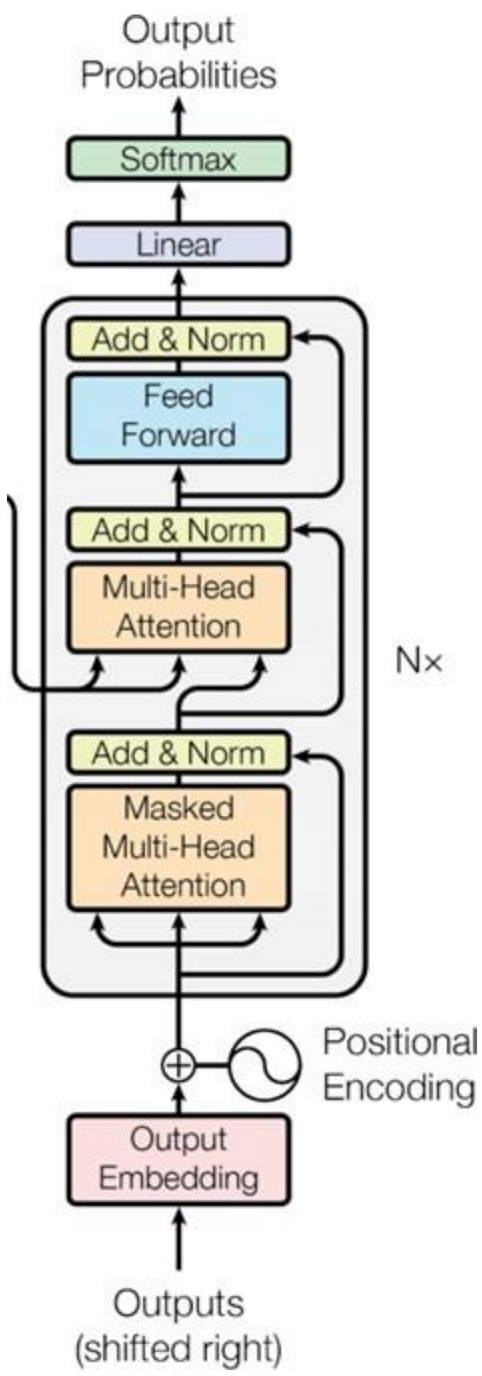
- Masked self attention.
- Cross attention attention layer is like what attention does in sequence-to-sequence models.
- It helps the decoder emphasize on relevant parts of the input.
- The same feed-forward network is applied to each position.



From Feedforward Network to Word Prediction

Linear Projection:

- Primary Role: Adjusting dimensionality.
- The linear layer serves to change the dimensionality of the feedforward network's output to match the size of the vocabulary.
- This ensures that the output has a dimension corresponding to every word in the dictionary.

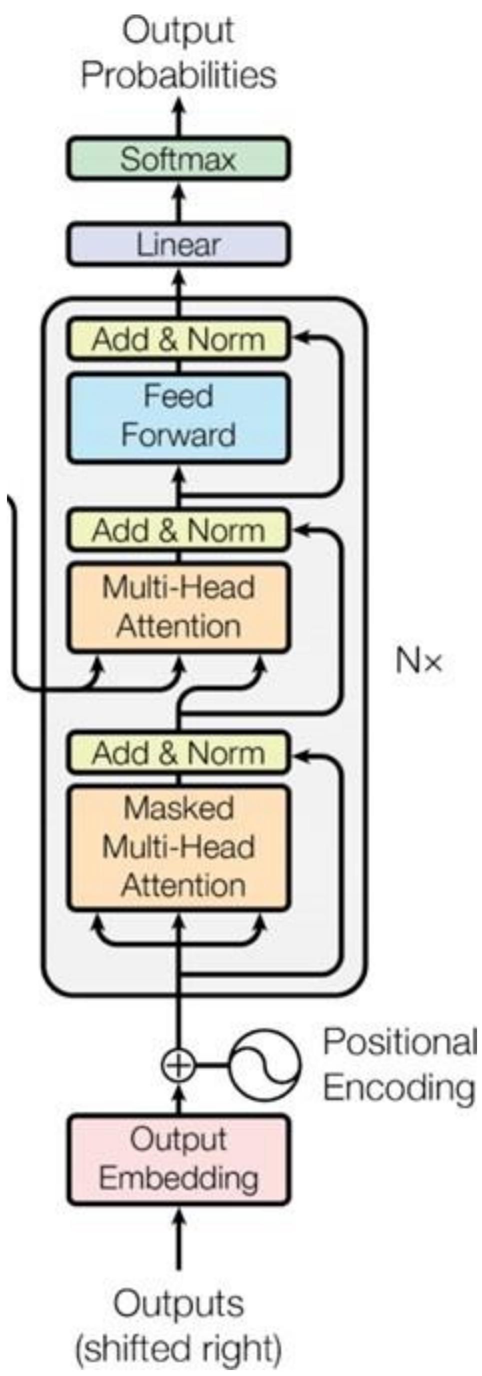


From Feedforward Network to Word Prediction

Softmax Activation:

•This function transforms the linear layer's output into probabilities.

•Representing the likelihood of a respective word being the next word in the sequence.



Positional Encoding

- Problem: no recurrence and no convolution, the model has no sense of the sequence.

Positional Encoding

- Problem: no recurrence and no convolution, the model has no sense of the sequence.

- We need a way to account for the order of the tokens in the sequence.

Positional Encoding

- Problem: no recurrence and no convolution, the model has no sense of the sequence.

- We need a way to account for the order of the tokens in the sequence.

- Solution: Adds a vector accounting for the position to each input embedding.

Positional Encoding

$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{p}}}\right)$$

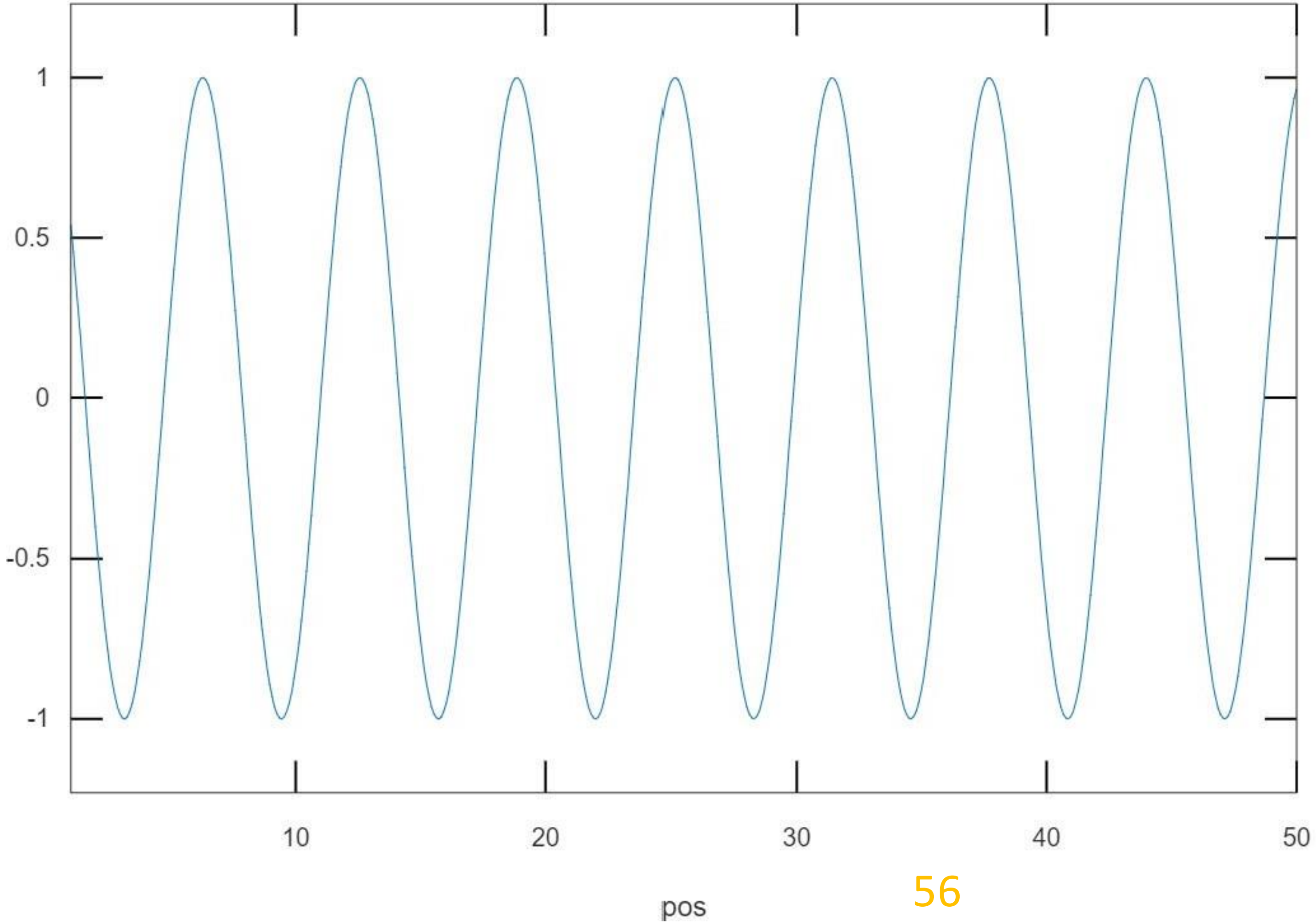
p is the dimension to which we project

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{p}}}\right)$$

Positional Encoding

$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{p}}}\right)$$

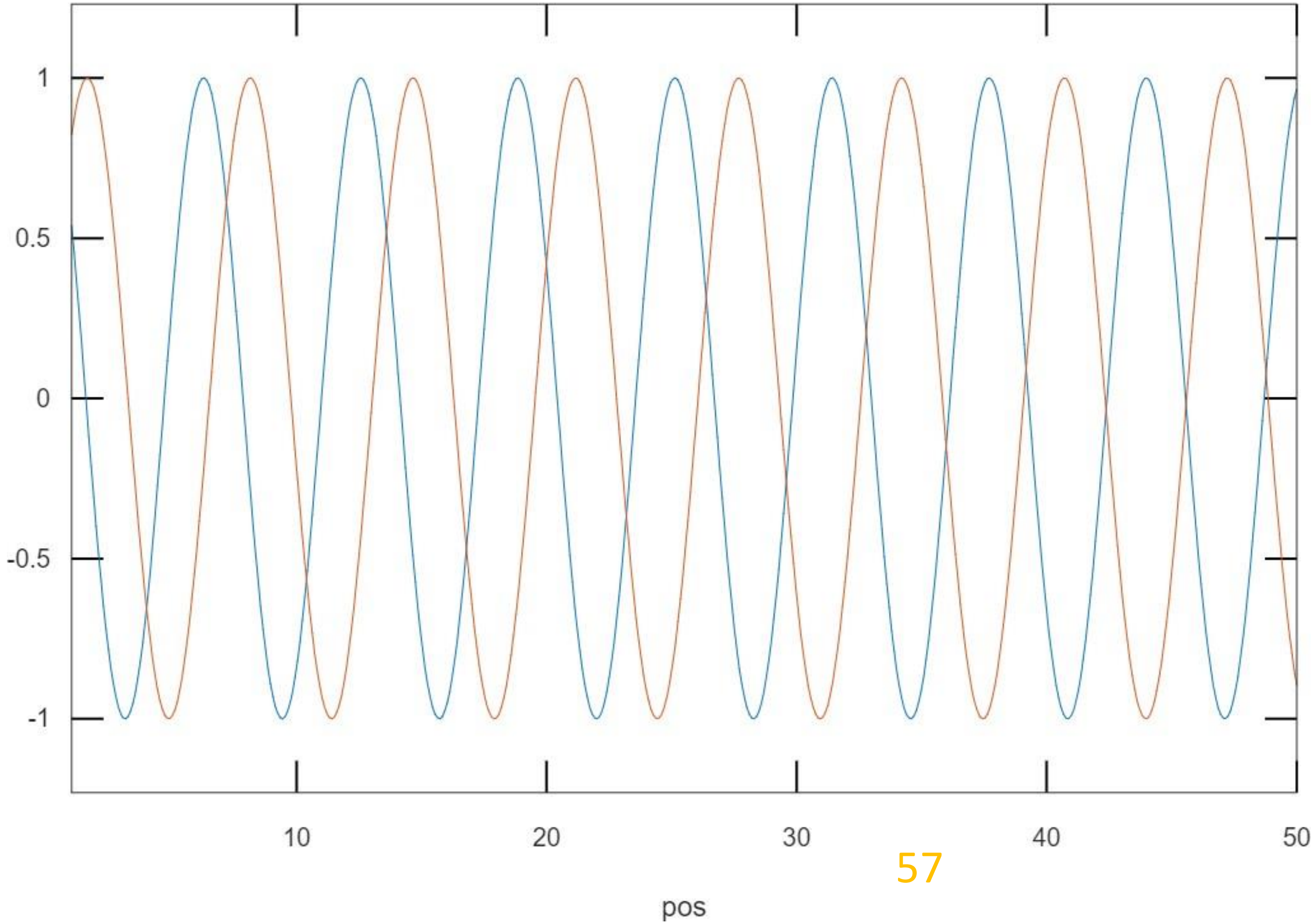
Positional Encoding



$i = 0$

$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{p}}}\right)$$

Positional Encoding



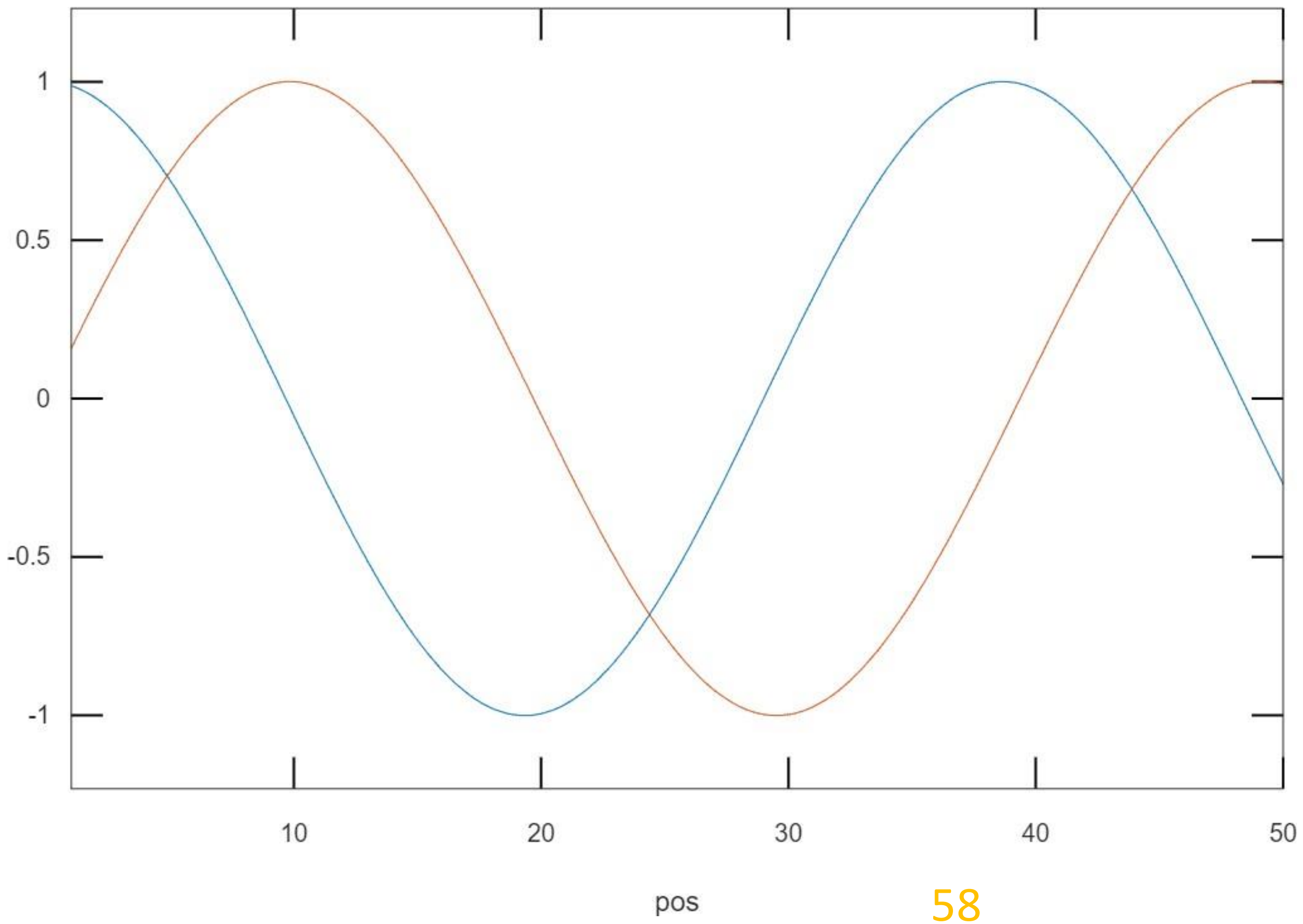
$i = 0$

$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{p}}}\right)$$

$i = 1$

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{p}}}\right)$$

Positional Encoding

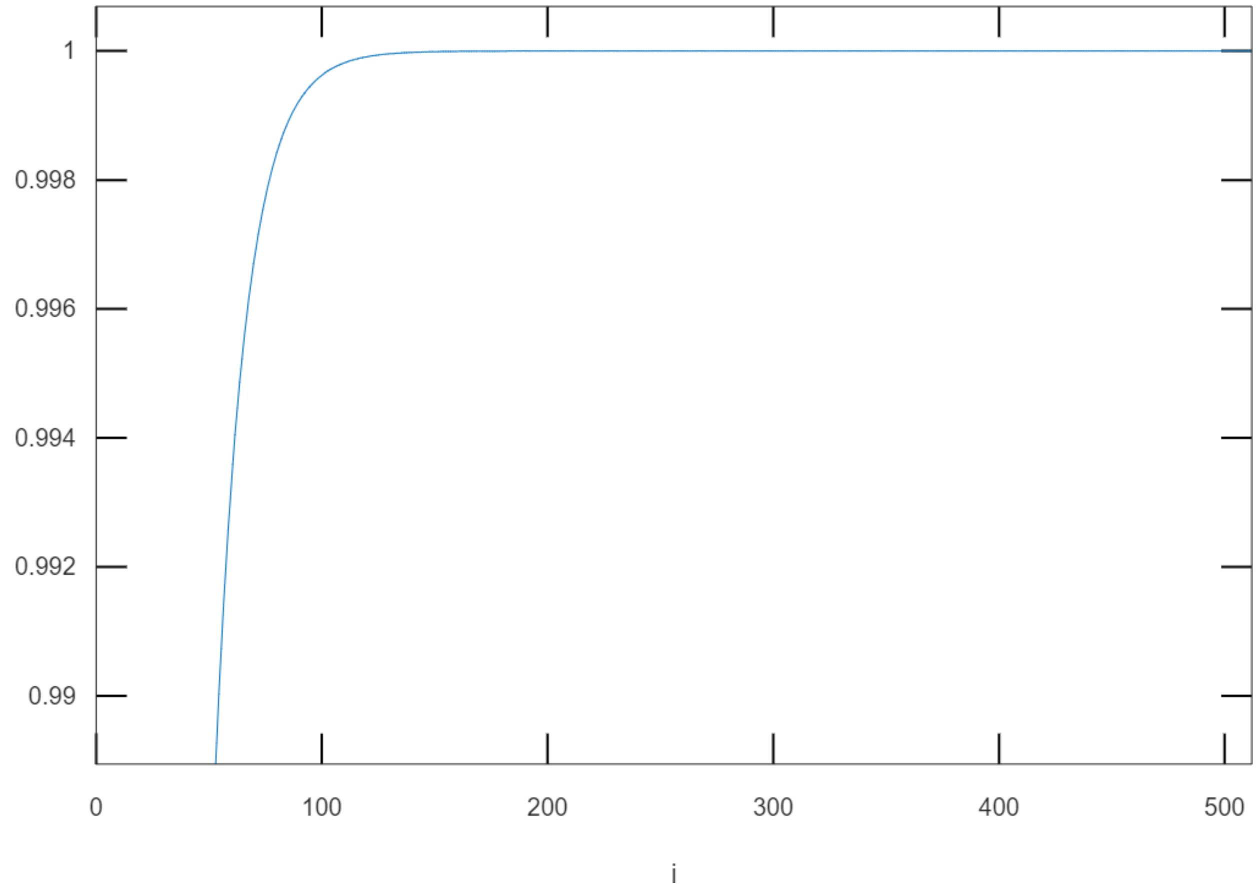


$i = 50$

$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{p}}}\right)$$

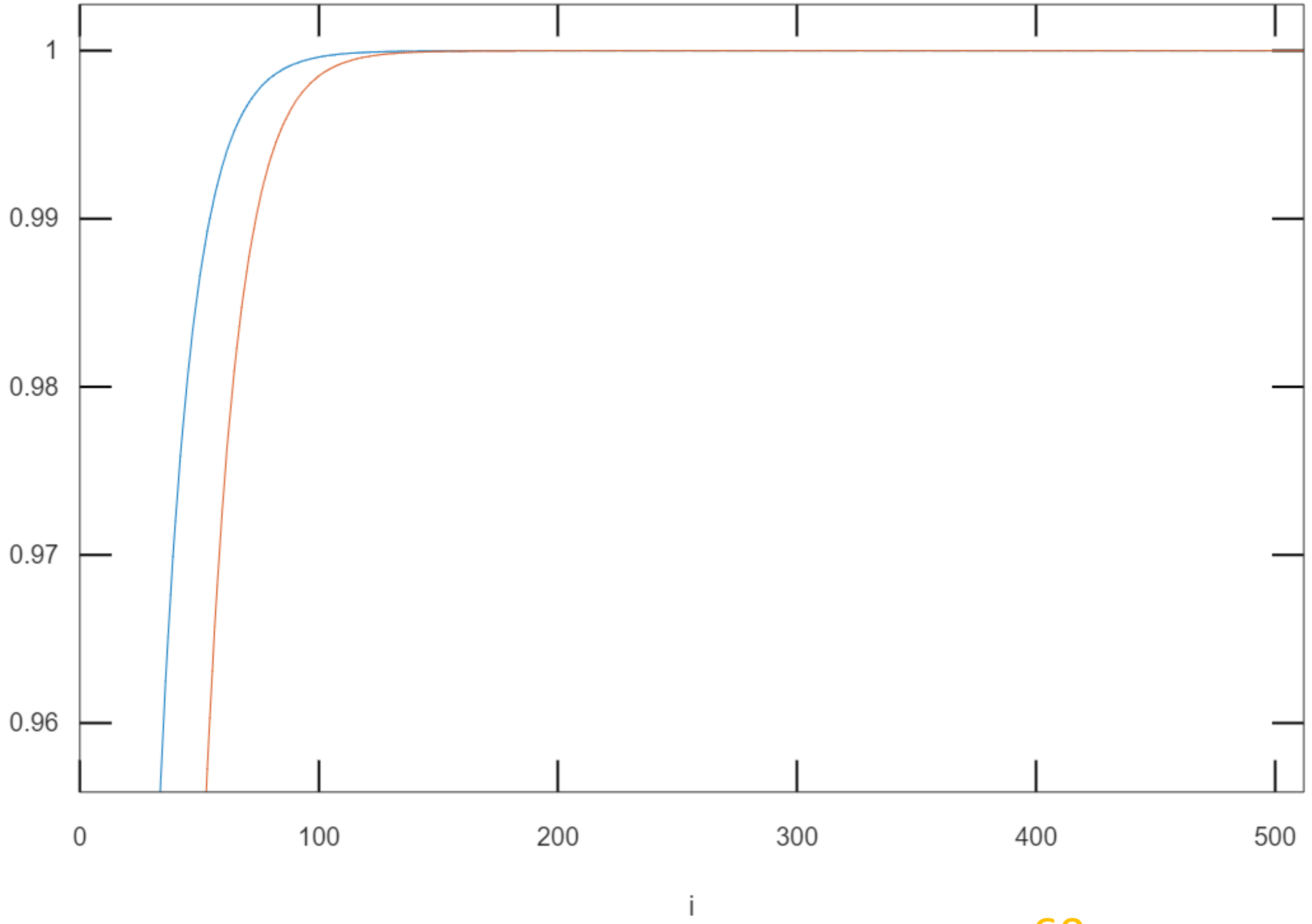
$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{p}}}\right)$$

Positional Encoding



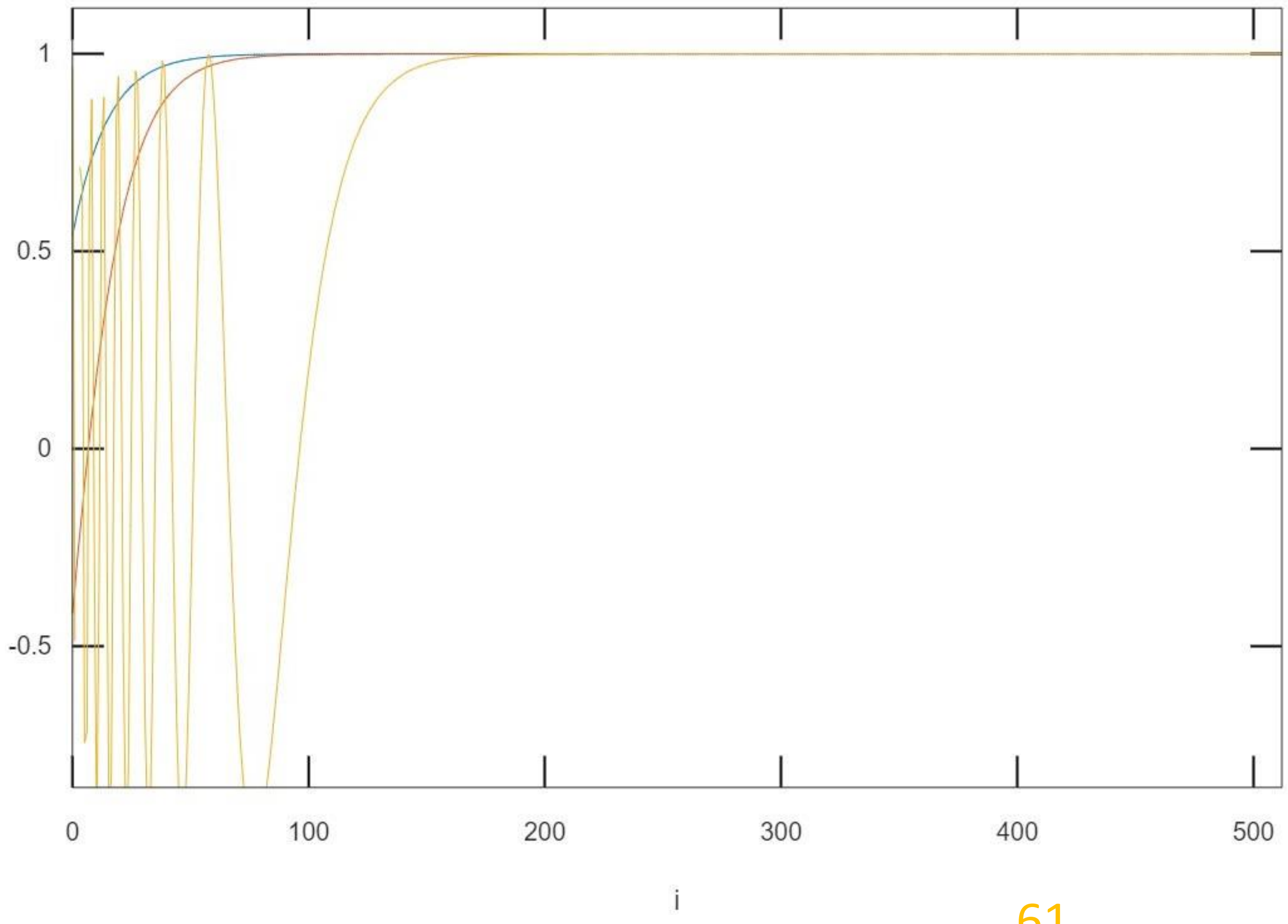
$$pos = 1$$

$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{p}}}\right)$$



$pos = 2$

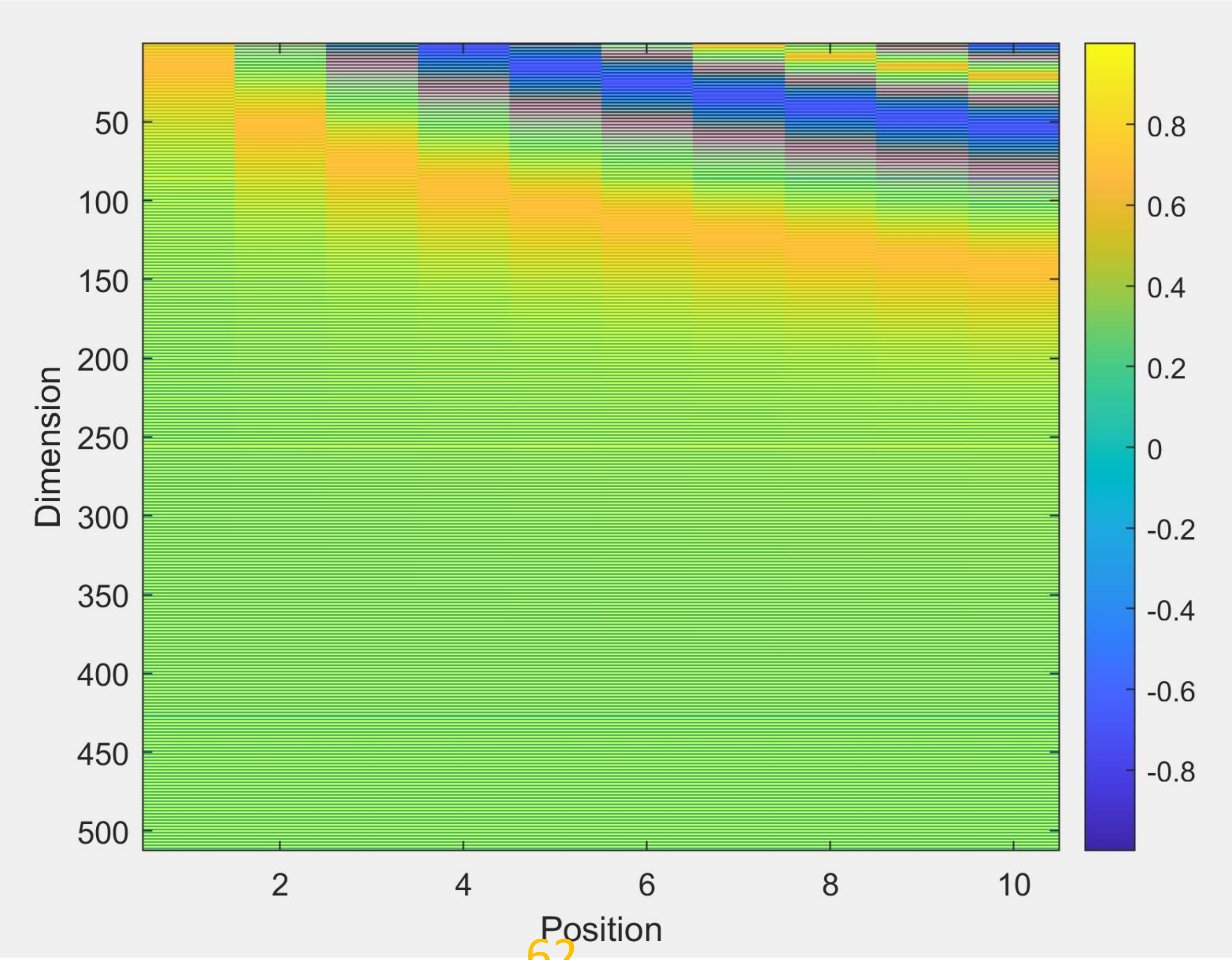
$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{p}}}\right)$$



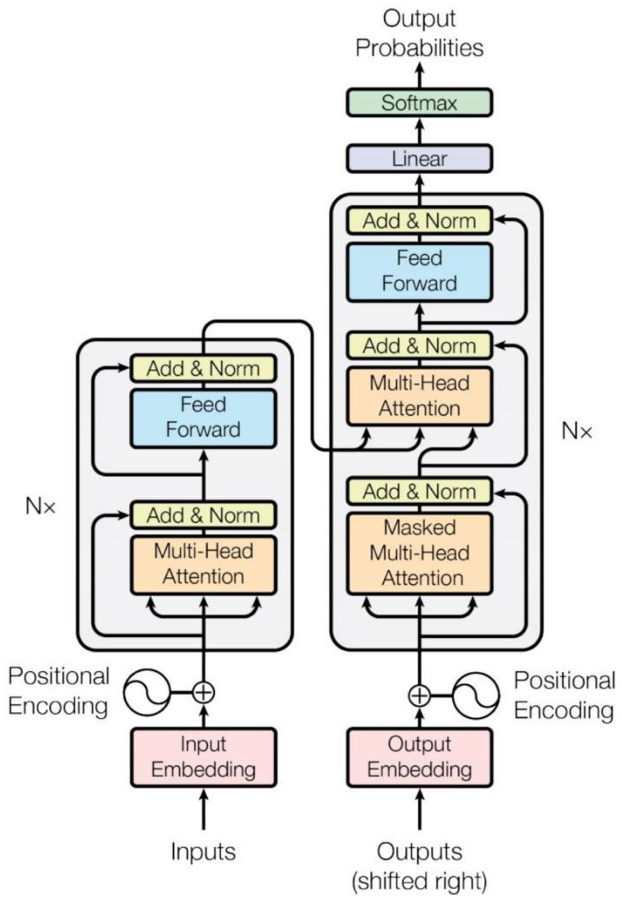
pos =50

$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{p}}}\right)$$

Positional Encoding Visualization



BERT,
GPT



Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

Generative Pre-trained Transformer (GPT)

- Improving Language Understanding by Generative Pre-Training (2018)

Alec Radford

Karthik Narasimhan

Tim Salimans

Ilya Sutskever

Bidirectional Encoder Representations from Transformers (BERT)

- BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (2018)

Jacob Devlin

Ming-Wei Chang

Kenton Lee

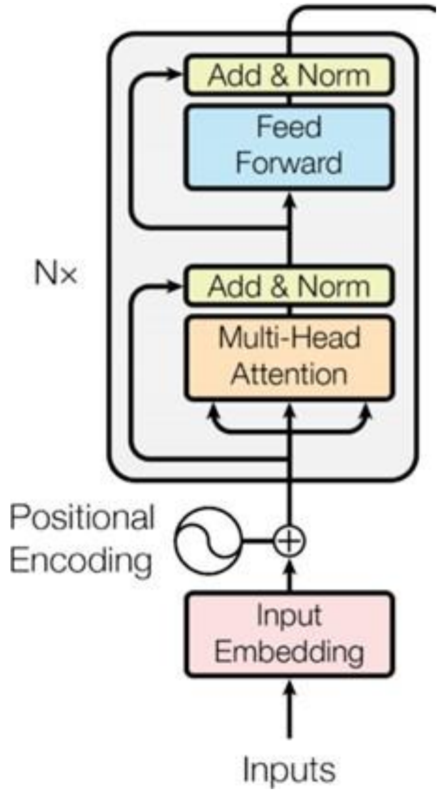
Kristina Toutanova

BERT and GPT

- The GPT is built using transformer decoder blocks.

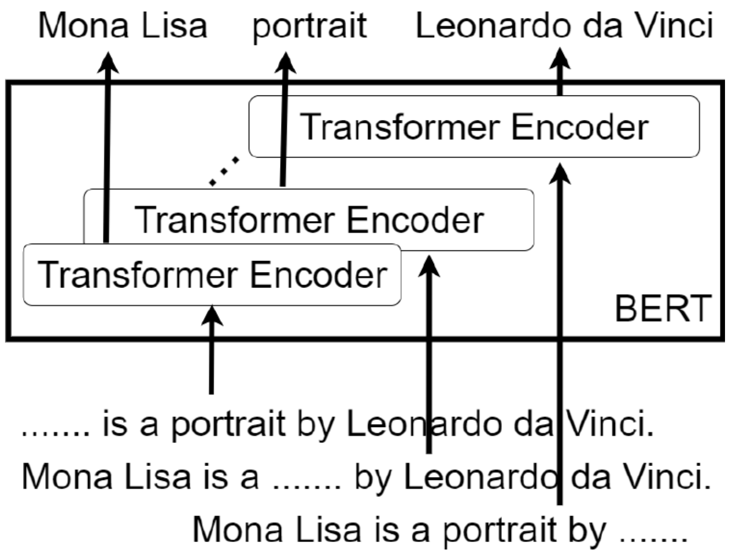
- BERT is built using transformer encoder blocks.

BERT


Add & Norm
Feed
Forward
Nx
Add & Norm
Multi-Head
Attention
Positional
Encoding
Input
Embedding
Inputs

Masked language model

- Masks words in the input and asks the model to predict the missing word.



BERT: Bidirectional language model

- Masks words in the input and asks the model to predict the missing word.

- Additional task: Given two sentences (A and B), is B likely to be the sentence that follows A, or not?

BERT

- BERT is designed to pretrain bidirectional representations from unlabeled text.

BERT

- BERT is designed to pretrain bidirectional representations from unlabeled text.

- Jointly conditioning on both left and right context.

BERT

- BERT is designed to pretrain bidirectional representations from unlabeled text.

- Jointly conditioning on both left and right context.

- The pre-trained BERT model can be finetuned with just one additional output layer.

BERT

- BERT is designed to pretrain bidirectional representations from unlabeled text.
- Jointly conditioning on both left and right context.
- The pre-trained BERT model can be finetuned with just one additional output layer.
- It creates state-of-the-art models for a wide range of tasks, such as question answering and language inference.

[CLS] Token in BERT

- The [CLS] token is prepended to the input text and travels through the Transformer layers alongside other tokens.
- All tokens, including [CLS], gather contextual information from the entire sequence due to the self-attention mechanism.
- For sentence-level tasks, the final hidden state of the [CLS] token is used as the sentence representation.
- During fine-tuning on a specific task, the model learns to imbue the [CLS] token with a meaningful representation of the entire sentence, optimized for that task.
- Example Usage: In classification tasks, the [CLS] token representation is fed into a classifier to determine the sentence's class.

BERT

BERT is basically a trained Transformer Encoder stack

1. **Transformer**:
    1. Encoder Layers: 6
    2. FFNN Hidden Layer Units: 512
    3. Attention Heads: 8
2. **BERT Base**:
    1. Encoder Layers: 12
    2. FFNN Hidden Layer Units: 768
    3. Attention Heads: 12
    4. Total Parameters: 110 million
3. **BERT Large**:
    1. Encoder Layers: 24
    2. FFNN Hidden Layer Units: 1024
    3. Attention Heads: 16
    4. Total Parameters: 340 million

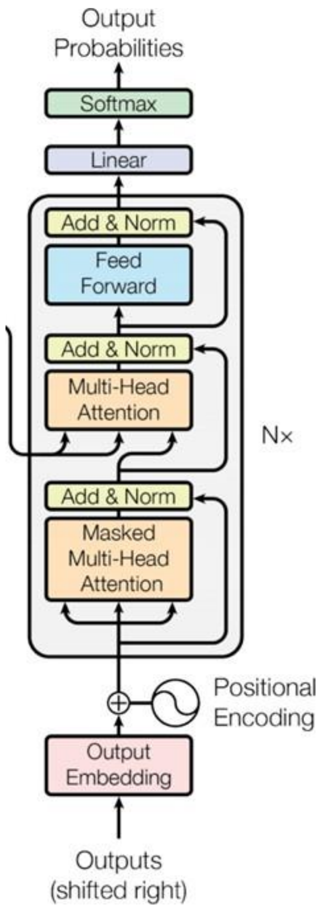
BERT

- **RoBERTa**:
  - Optimizes BERT's training process by using more data, larger batch sizes, and longer training times, resulting in improved performance on NLP tasks.
- **TinyBERT**:
  - A smaller and faster version of BERT designed for resource-constrained environments, retaining competitive performance with significantly fewer parameters.

BERT

1. **Multilingual BERT**:
   1. Trained on 104 different languages, capable of "zero-shot" adaptation to a new language domain.
2. **Domain Specific BERT Variants**:
   1. BioBERT: Retrained on a biomedical corpus.
   2. SciBERT: Trained on over one million published articles.
   3. BERTweet: A RoBERTa model trained on 850 million tweets.
   4. FinBERT: Adapted to the financial domain.

GPT


Output
Probabilities
Softmax
Linear
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Nx
Add & Norm
Masked
Multi-Head
Attention
Positional
Encoding
Output
Embedding
Outputs
(shifted right)

GPT

Predict the next word, given all of the previous words

- **Architecture**:
  - Stack of Transformer decoder blocks.
  - No encoder, hence no cross-attention module.
  - Components: Positional encoding, masked multihead self-attention, and feedforward network.
- **Directionality**:
  - Only considers previous (left) words in attention, not bidirectional like BERT.
  - Utilizes masked multihead self-attention for this purpose.

GPT 1

- Released: 2018
- Parameters: 117 Million
- Layers: 12
- Training Data: Books1 Corpus (7,000 unpublished books)
- Focus: Unsupervised pre-training, Transformer architecture, large-scale language modeling

GPT
2

- Released: 2019
- Parameters: ~1.5 Billion
- Layers: 48
- Training Data: 40GB (English)
- Focus: Transformer architecture, self-attention mechanism

GPT 3

- Released: 2020
- Parameters: 175 Billion
- Layers: 175
- Training Data: 570GB (Multilingual)
- Focus: Few-shot learning, prompt engineering, Python support

GPT
4

- Release: Not Yet
- Parameters: ~100 Trillion (speculative)
- Layers: Unknown
- Training Data: Larger, more diverse (speculative)
- Focus: GPT-4 is known to be a multimodal model, capable of processing both text and image inputs to generate text outputs. Advanced few-shot learning, improved NLU and NLG, reasoning and inference