24. Transformers

STA3142 Statistical Machine Learning

Kibok Lee
Assistant Professor of
Applied Statistics / Statistics and Data Science

Jun 11, 2024

** Slides adapted from EECS498/598 @ Univ. of Michigan by Justin Johnson*

The Rest of the Course Schedule

- **6/4 Tue**: 22. Generative Models
- **6/6 Thu**: 23. Recurrent Neural Networks (We have a class!)
- **6/9 Tue**: 24. Transformers & 25. Reinforcement Learning
- **6/13 Thu**: 26. ML Advice
- **6/14 Fri**: Final Assignment Deadline

Assignment 5 (Final Exam Replacement)

- Due **Friday 6/14, 11:59pm**
- Topic: Convolutional Neural Networks
  - Derive gradients for NN layers
  - Implement layers for CNNs
  - Train a CNN classifier for MNIST digit recognition
- Please read the instruction carefully!
  - Submit one pdf and one zip file separately
  - Write your code only in the designated spaces
  - Do not import additional libraries
  - …
- If you feel difficult, consider to take option 2.

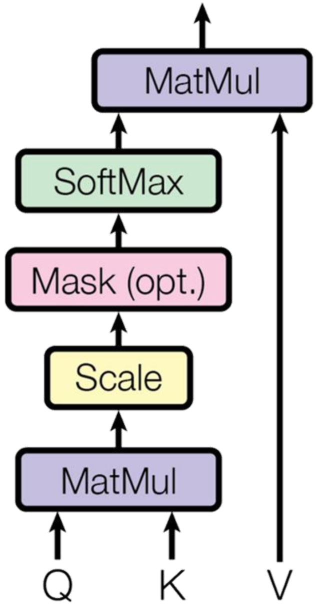
Attention

- Mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors
- Example: YouTube video search
  - Query: text prompt
  - Key: meta-information (video title, description, …)
  - Value: videos
- Given a query, find the best matching key and return the corresponding value
  - Sort by matching scores in video search
  - Soft matching for attention mechanism
    - Return weighted sum of values; weights are normalized matching scores

Attention

- Mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors

- Scaled Dot-Product Attention:



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

Caution: In the following slides, weight matrix multiplications mostly come with bias addition but omitted

Vaswani et al, “Attention is all you need”, NeurIPS 2017

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:

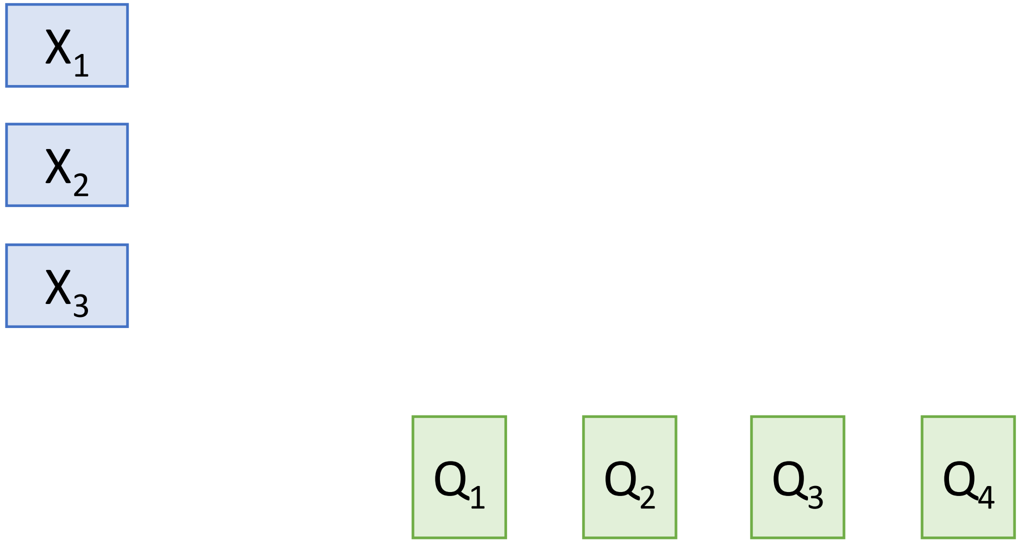
Key vectors: $\mathbf{K} = \mathbf{X}\mathbf{W_K}$ (Shape: N_X x D_Q)

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

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

Attention weights: A = softmax(E, 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:

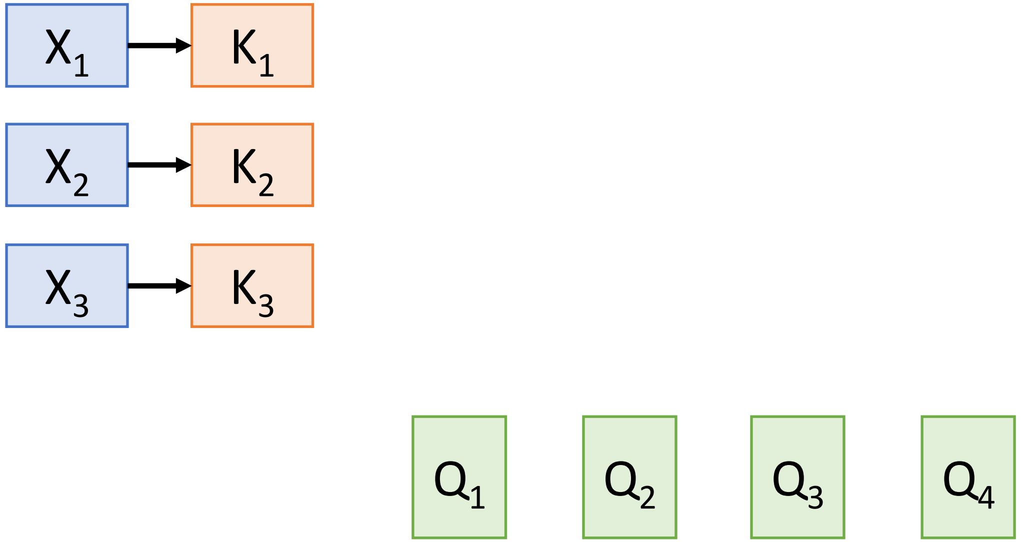
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)

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

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

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

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



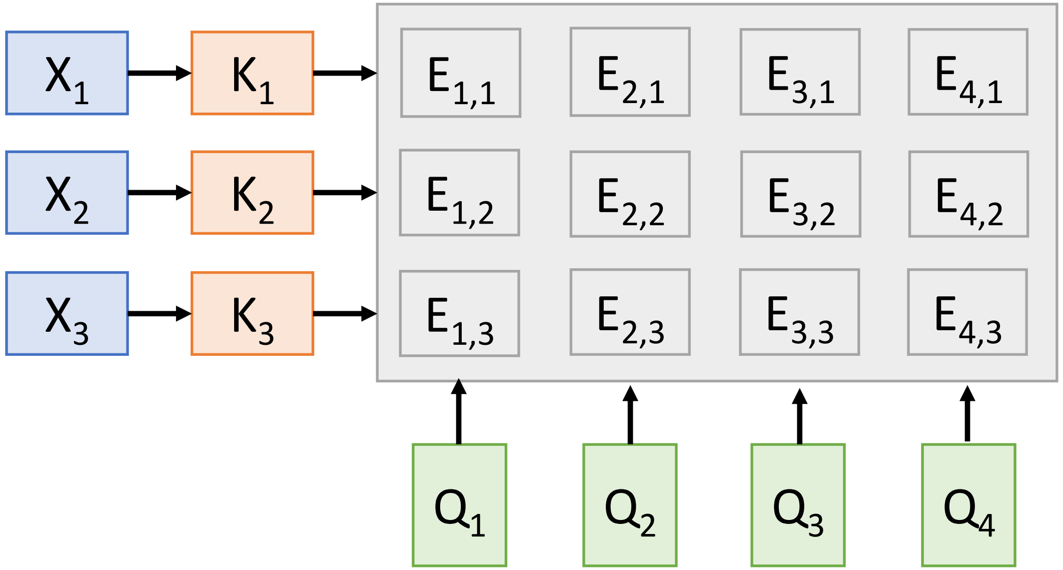
Attention Layer

Inputs:

Query vectors: **Q** (Shape: N_Q x D_Q)
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: **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 = $QK^T / \sqrt{D_Q}$ (Shape: N_Q x N_X) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_Q x N_X)
Output vectors: Y = AV (Shape: N_Q x D_V) $Y_i = \sum_j A_{i,j} V_j$



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:

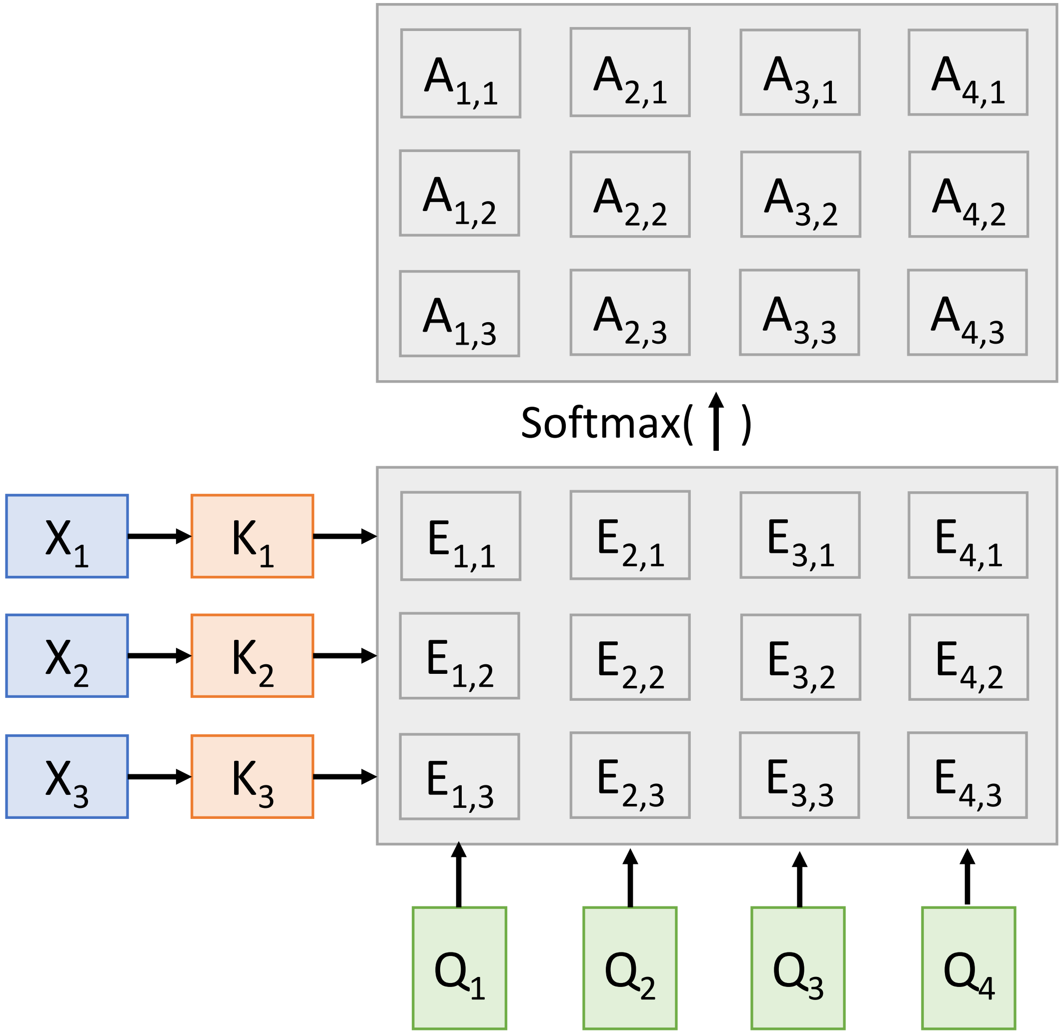
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)

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

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

Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (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:

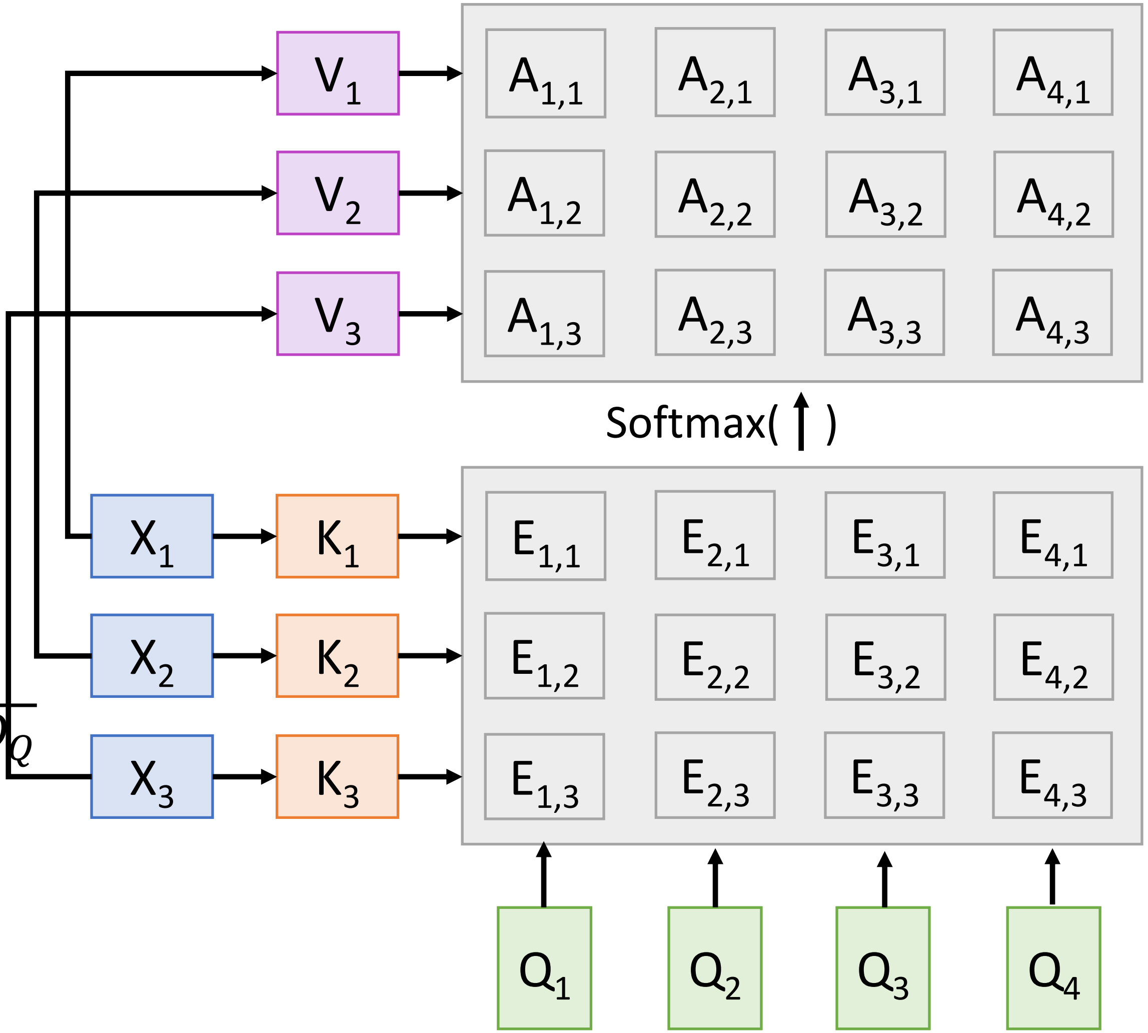
Key vectors: **K** = $\mathbf{XW_K}$ (Shape: N_X x D_Q)

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

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

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

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



V_1
V_2
V_3
$A_{1,1}$ $A_{2,1}$ $A_{3,1}$ $A_{4,1}$
$A_{1,2}$ $A_{2,2}$ $A_{3,2}$ $A_{4,2}$
$A_{1,3}$ $A_{2,3}$ $A_{3,3}$ $A_{4,3}$
Softmax(↑)
X_1
X_2
X_3
K_1
K_2
K_3
$E_{1,1}$ $E_{2,1}$ $E_{3,1}$ $E_{4,1}$
$E_{1,2}$ $E_{2,2}$ $E_{3,2}$ $E_{4,2}$
$E_{1,3}$ $E_{2,3}$ $E_{3,3}$ $E_{4,3}$
Q_1 Q_2 Q_3 Q_4

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:

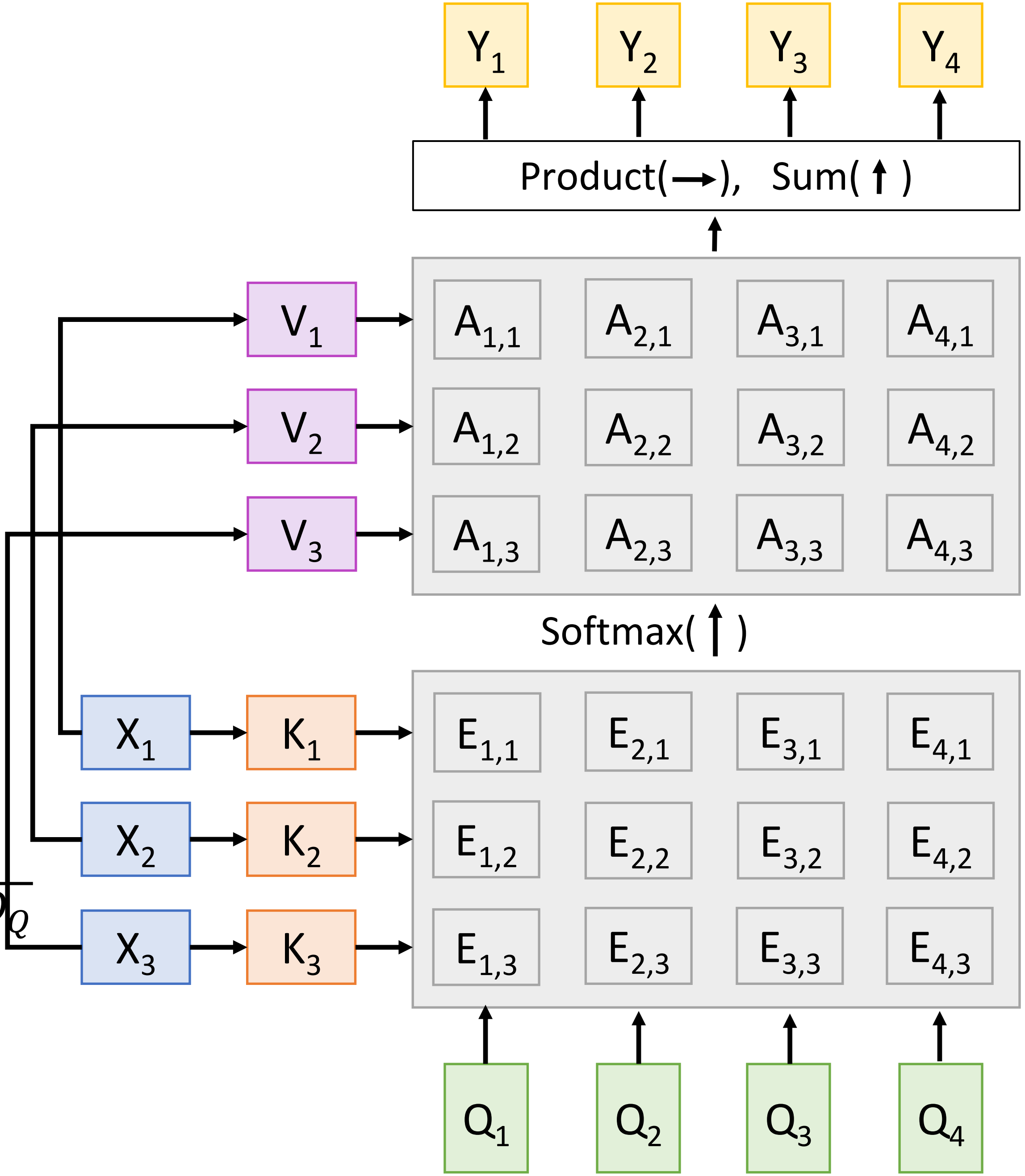
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)

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

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

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

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

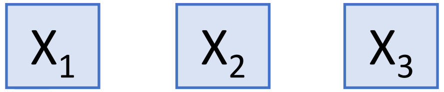


Self-Attention Layer

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

Inputs:
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: $E = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: N_Q x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$
Attention weights: A = softmax(E, dim=1) (Shape: N_Q x N_X)
Output vectors: $Y = A\mathbf{V}$ (Shape: N_Q x D_V) $Y_i = \sum_j A_{i,j} \mathbf{V_j}$



Self-Attention Layer

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

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: $\mathbf{Q} = \mathbf{XW_Q}$

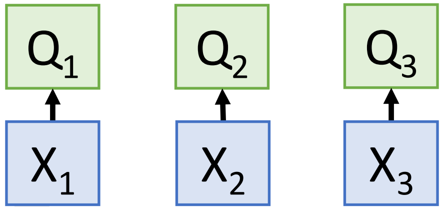
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)

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

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

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) $Y_i = \sum_j A_{i,j} \mathbf{V_j}$

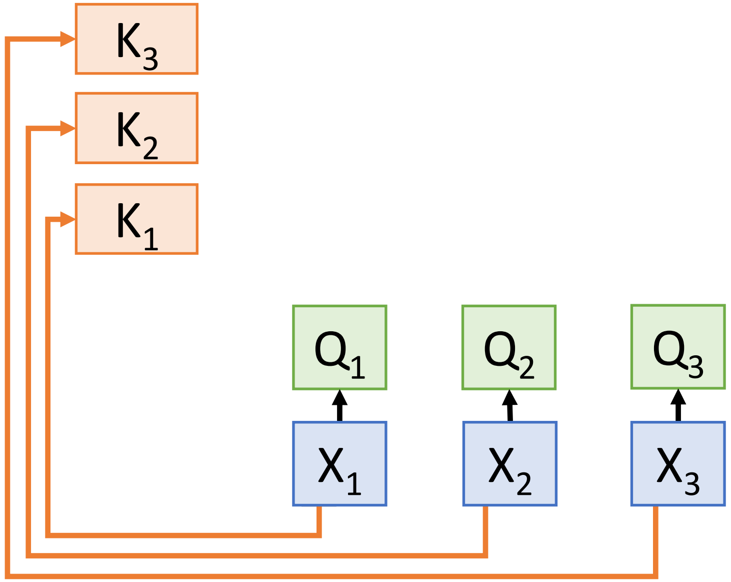


Self-Attention Layer

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

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

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

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: $\mathbf{Q} = \mathbf{XW_Q}$

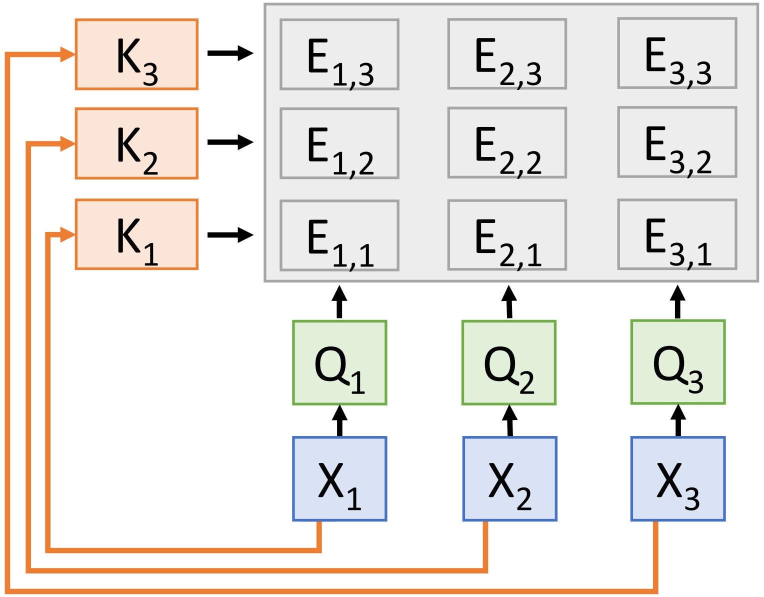
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)

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

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

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

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**

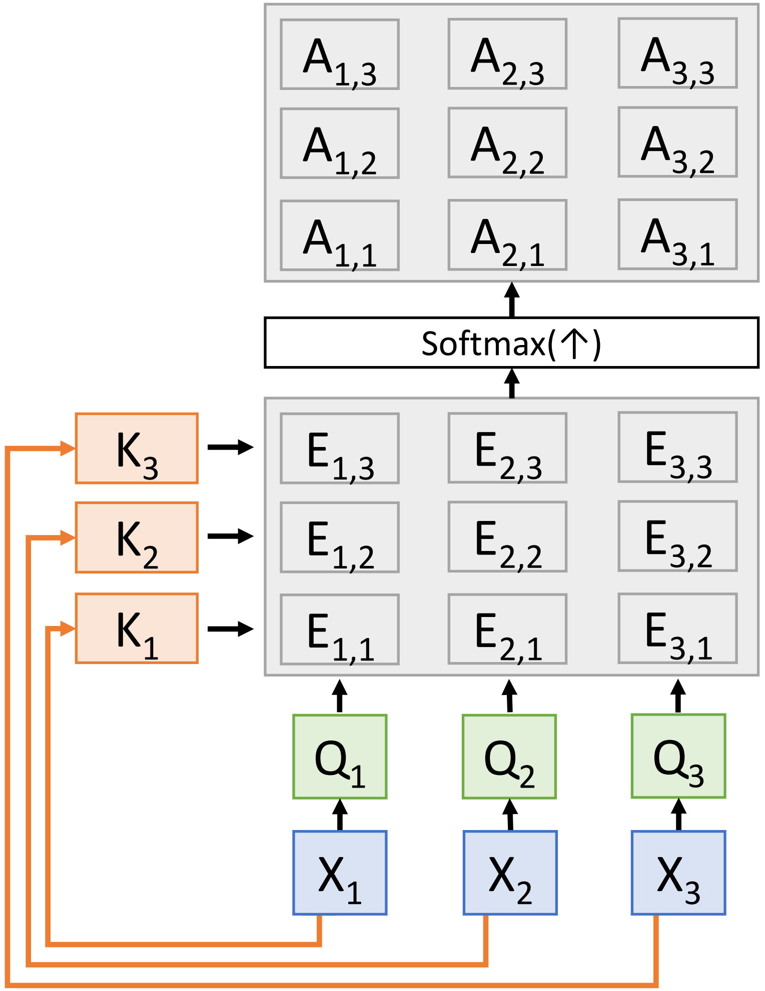
Key vectors: **K** = **XW_K** (Shape: N_X x D_Q)

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

Similarities: E = $QK^T / \sqrt{D_Q}$ (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 N_X)

Output vectors: Y = AV (Shape: N_X 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 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: $Q = XW_Q$

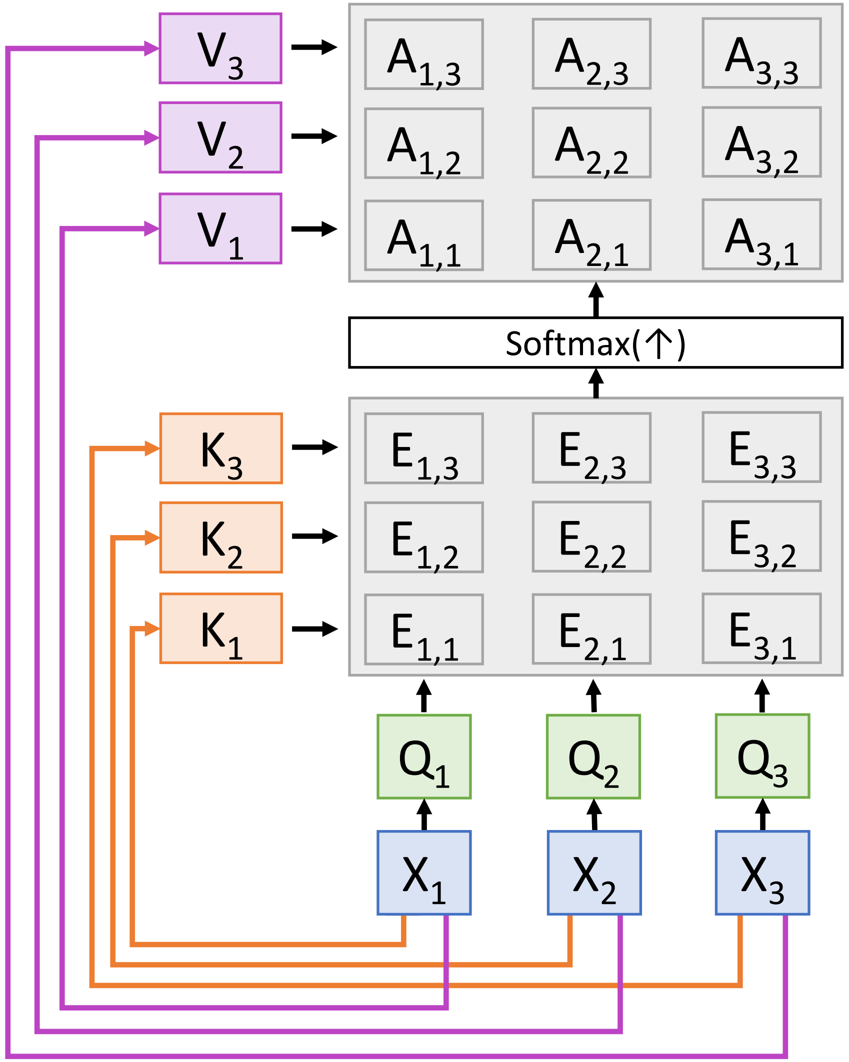
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)

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

Similarities: $E = QK^T / \sqrt{D_Q}$ (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 N_X)

Output vectors: $Y = AV$ (Shape: N_X 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 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**

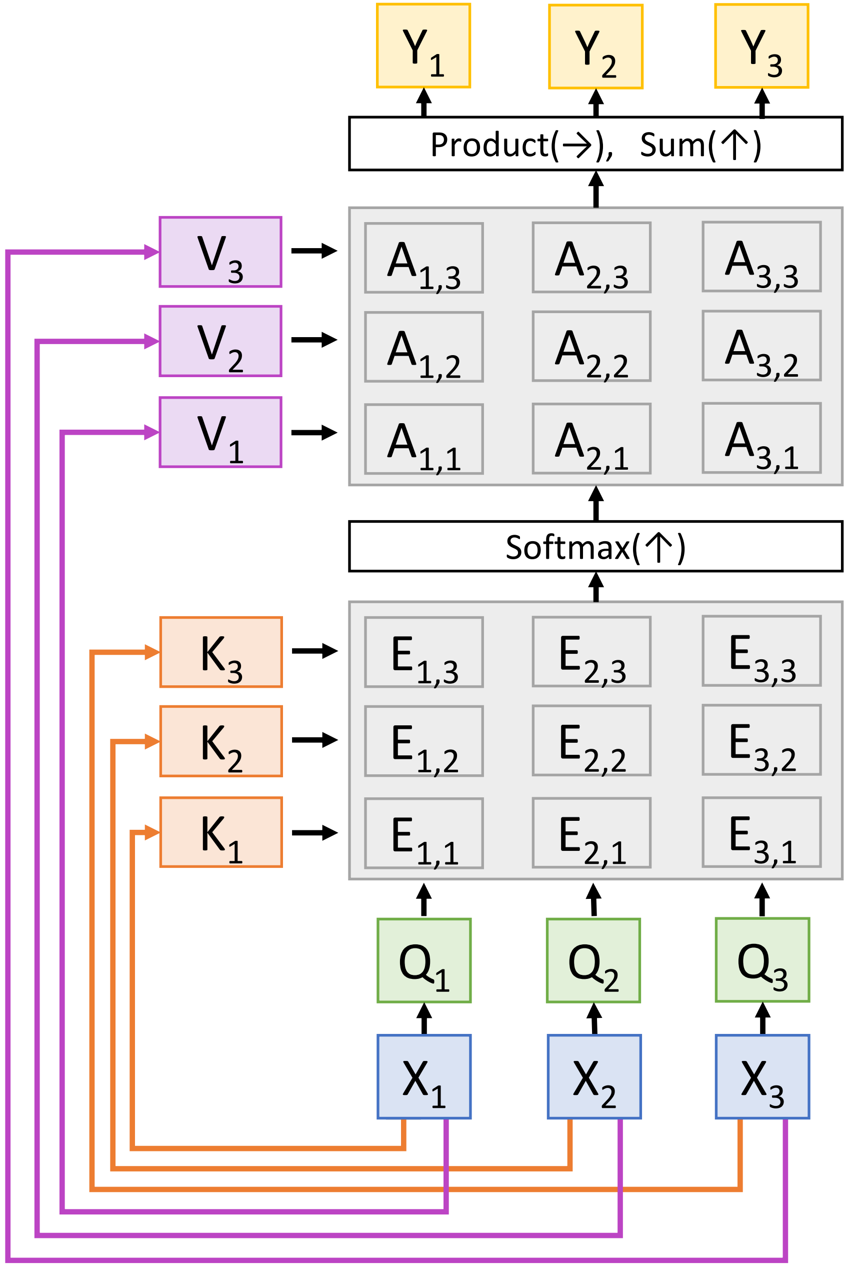
Key vectors: **K** = **XW_K** (Shape: N_X x D_Q)

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

Similarities: $E = QK^T / \sqrt{D_Q}$ (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 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:

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: $\mathbf{Q} = \mathbf{XW_Q}$

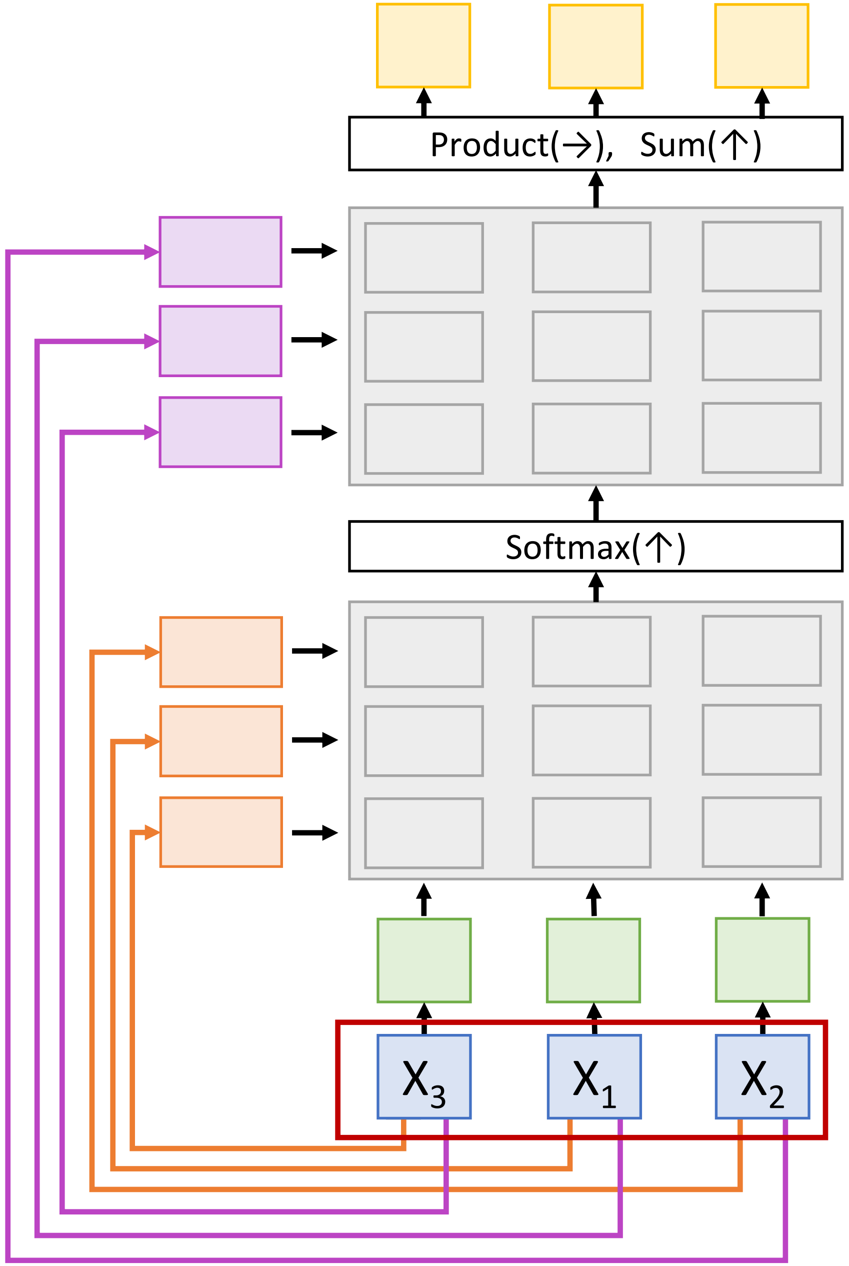
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)

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

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

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: $\mathbf{Q} = \mathbf{X}\mathbf{W_Q}$

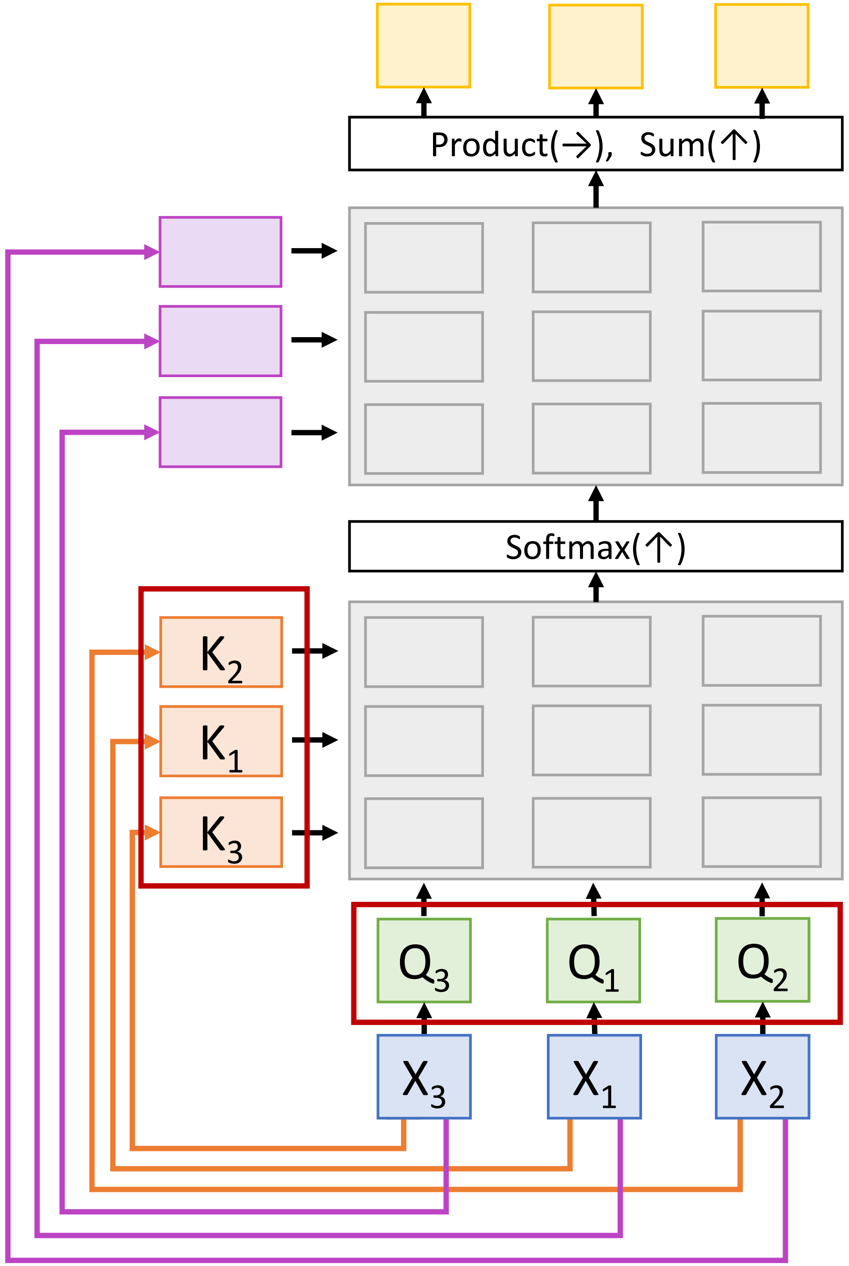
Key vectors: $\mathbf{K} = \mathbf{X}\mathbf{W_K}$ (Shape: N_X x D_Q)

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

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

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

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)

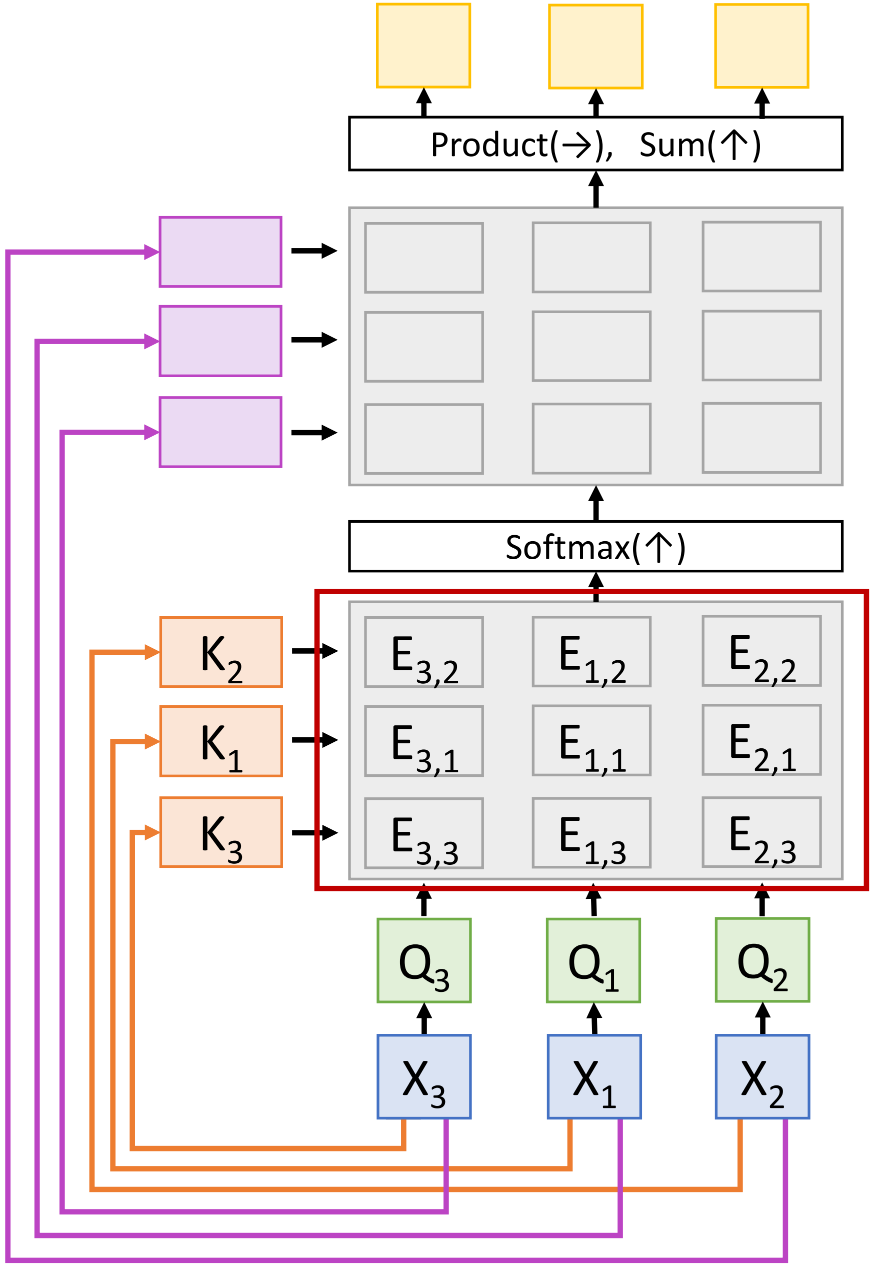
Similarities: $E = \mathbf{QK^T} / \sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$

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) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$

Consider **permuting** the input vectors:

Similarities will be the same, but permuted



Self-Attention Layer

Consider **permuting** the input vectors:

Attention weights 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: $Q = XW_Q$

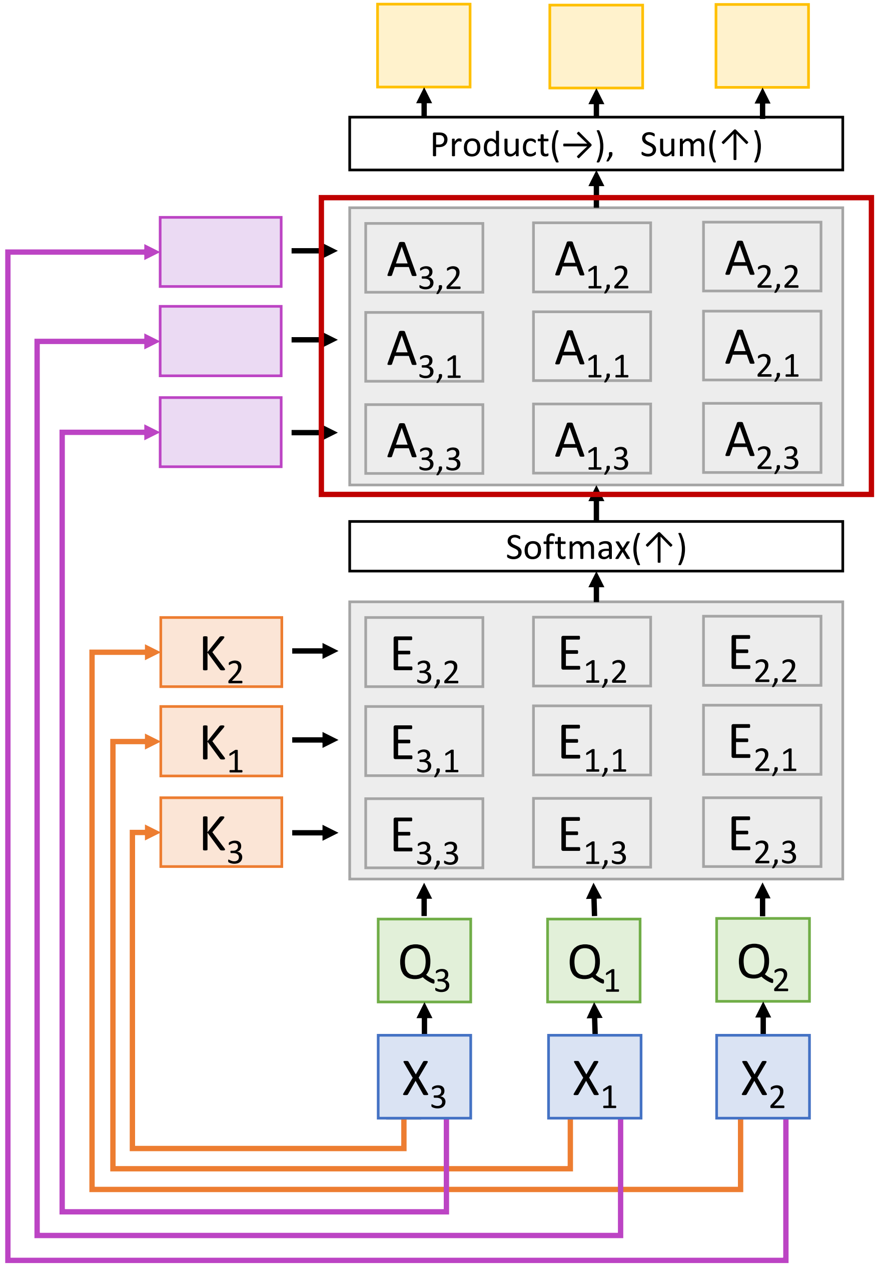
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)

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

Similarities: $E = QK^T / \sqrt{D_Q}$ (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 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:

Values will be the same, but permuted

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: $\mathbf{Q} = \mathbf{XW_Q}$

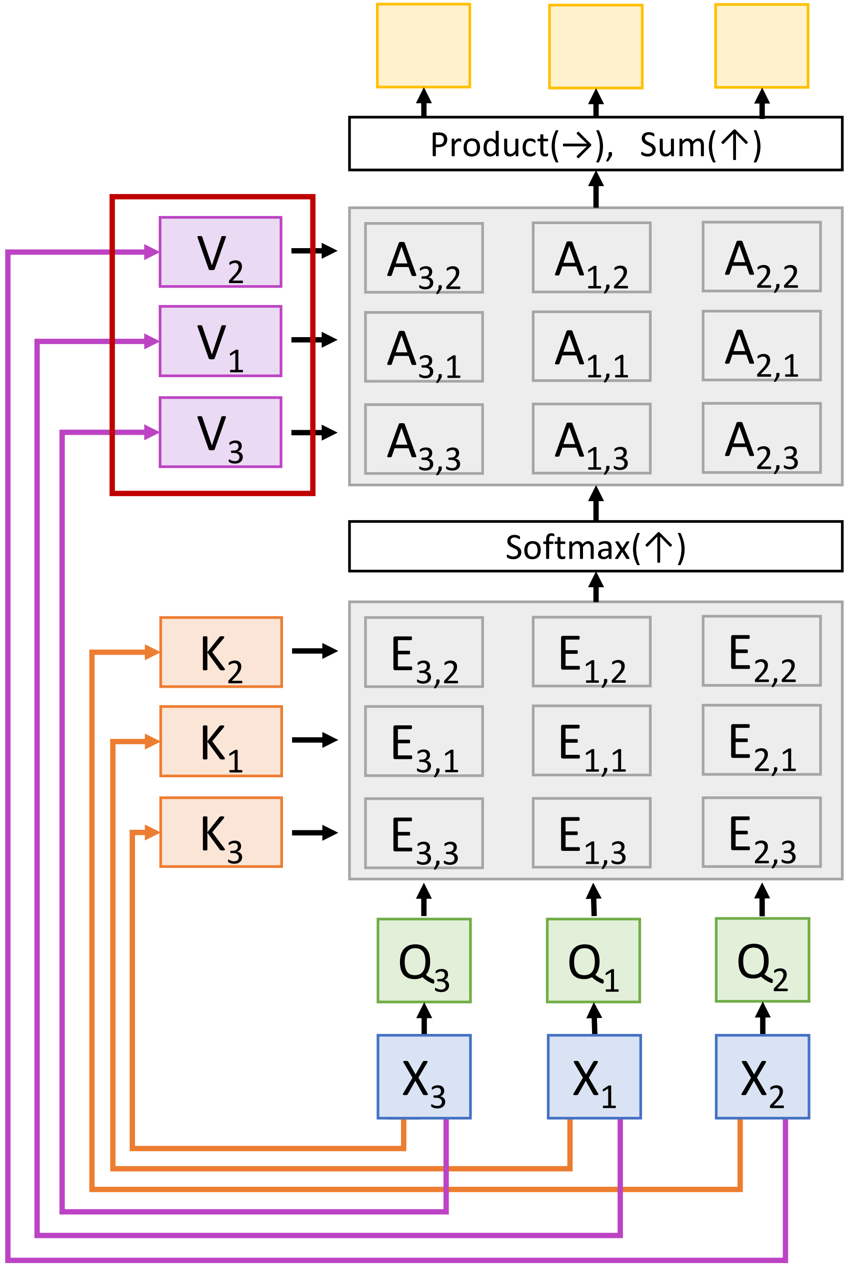
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)

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

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

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

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

Input vectors: $\mathbf{X}$ (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: $E = \mathbf{QK}^T / \sqrt{D_Q}$ (Shape: $N_X \times N_X$) $E_{i,j} = (\mathbf{Q_i} \cdot \mathbf{K_j}) / \sqrt{D_Q}$

Attention weights: A = softmax(E, dim=1) (Shape: $N_X \times N_X$)

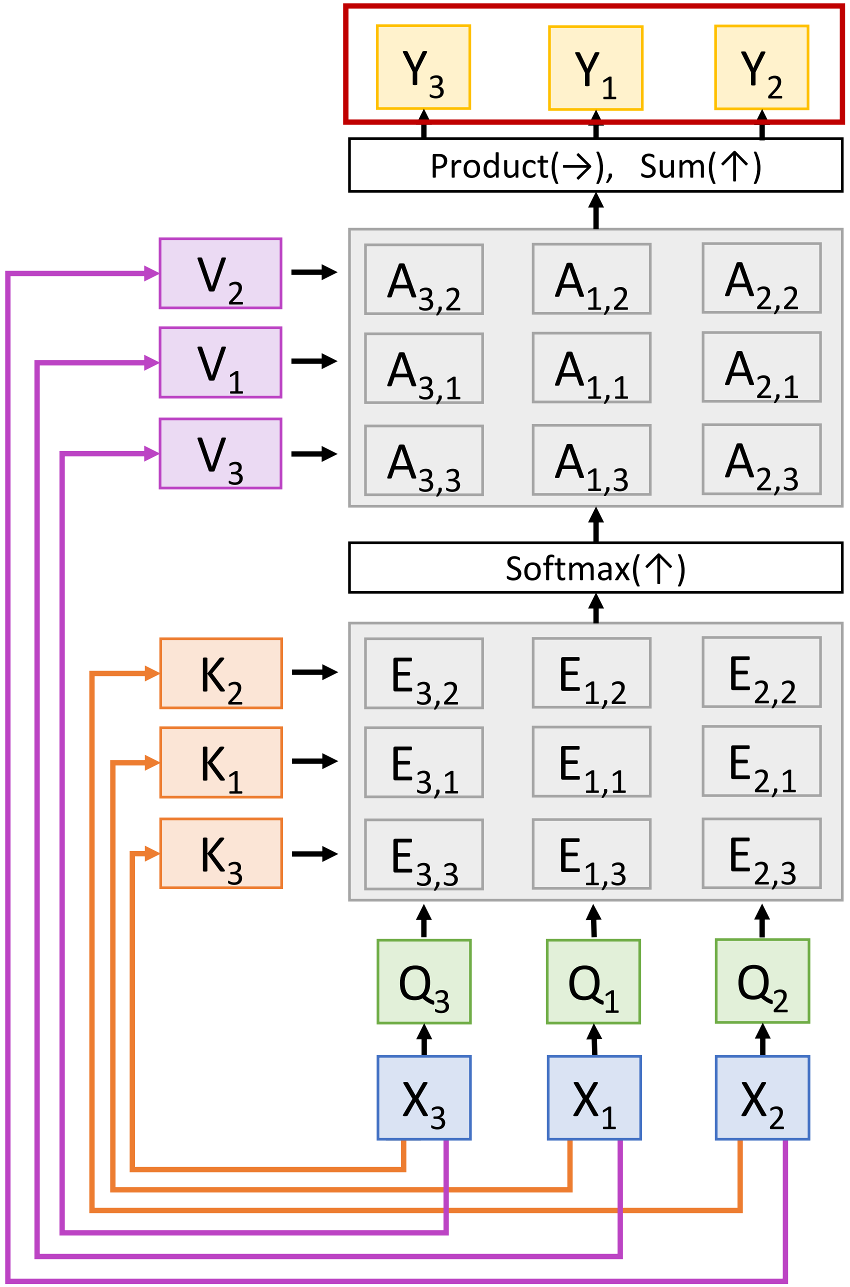
Output vectors: $Y = A\mathbf{V}$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} \mathbf{V_j}$

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: $\mathbf{Q} = \mathbf{XW_Q}$

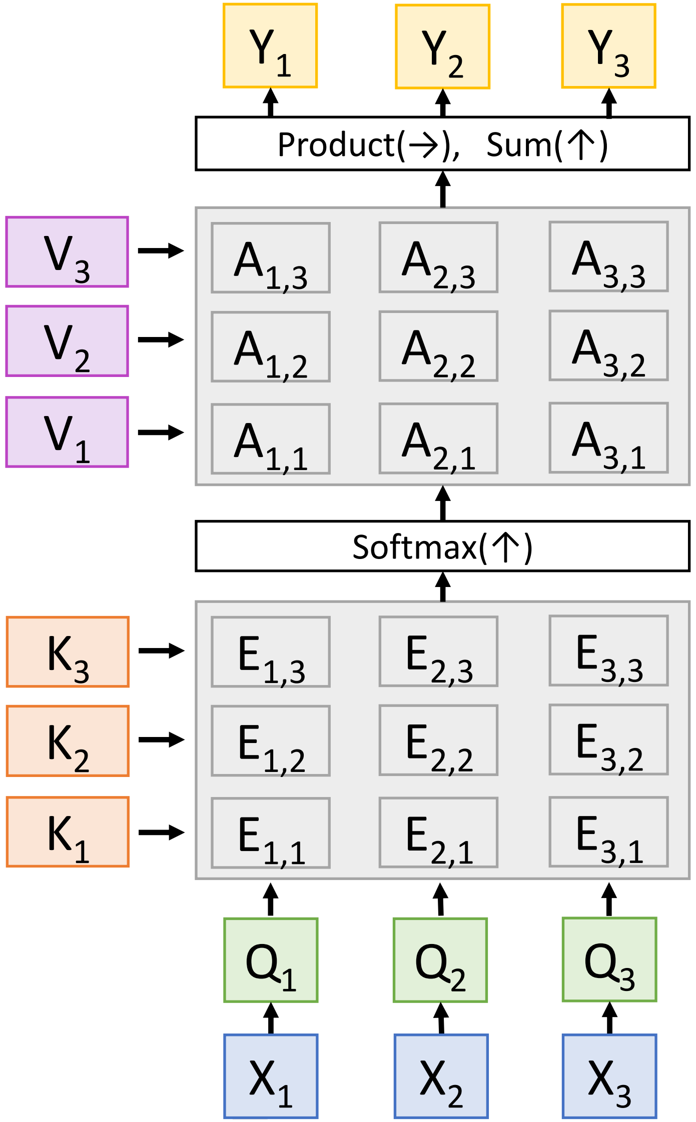
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)

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

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

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

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** $/\sqrt{D_Q}$ (Shape: N_X x N_X) $E_{i,j} = (Q_i \cdot K_j) / \sqrt{D_Q}$

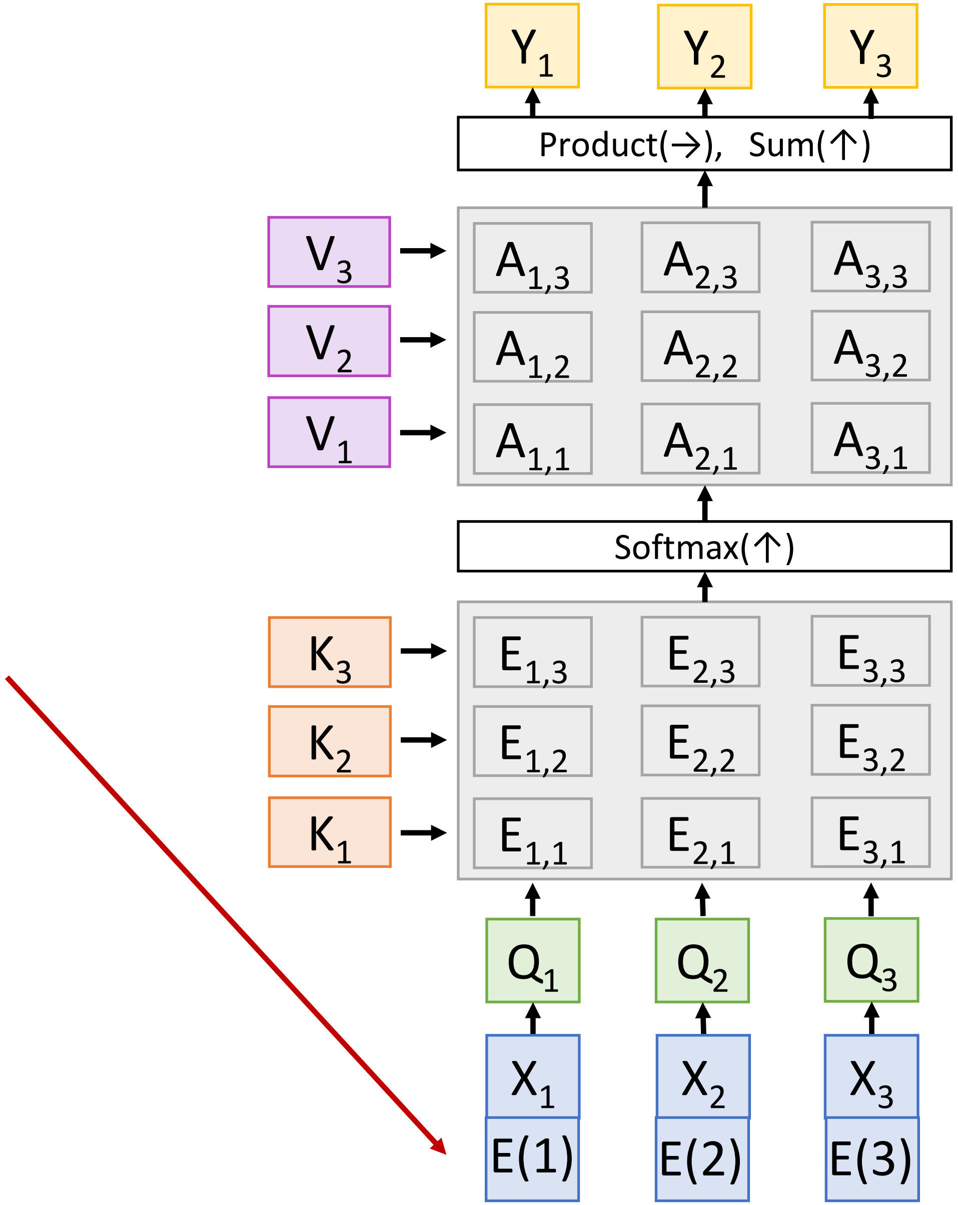
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)

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

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

In order to make processing position-aware, concatenate or add **positional encoding** to the input

E is either learnable or fixed (sinusoidal wave)



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: $\mathbf{W_K}$ (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: $\mathbf{Q} = \mathbf{X}\mathbf{W_Q}$

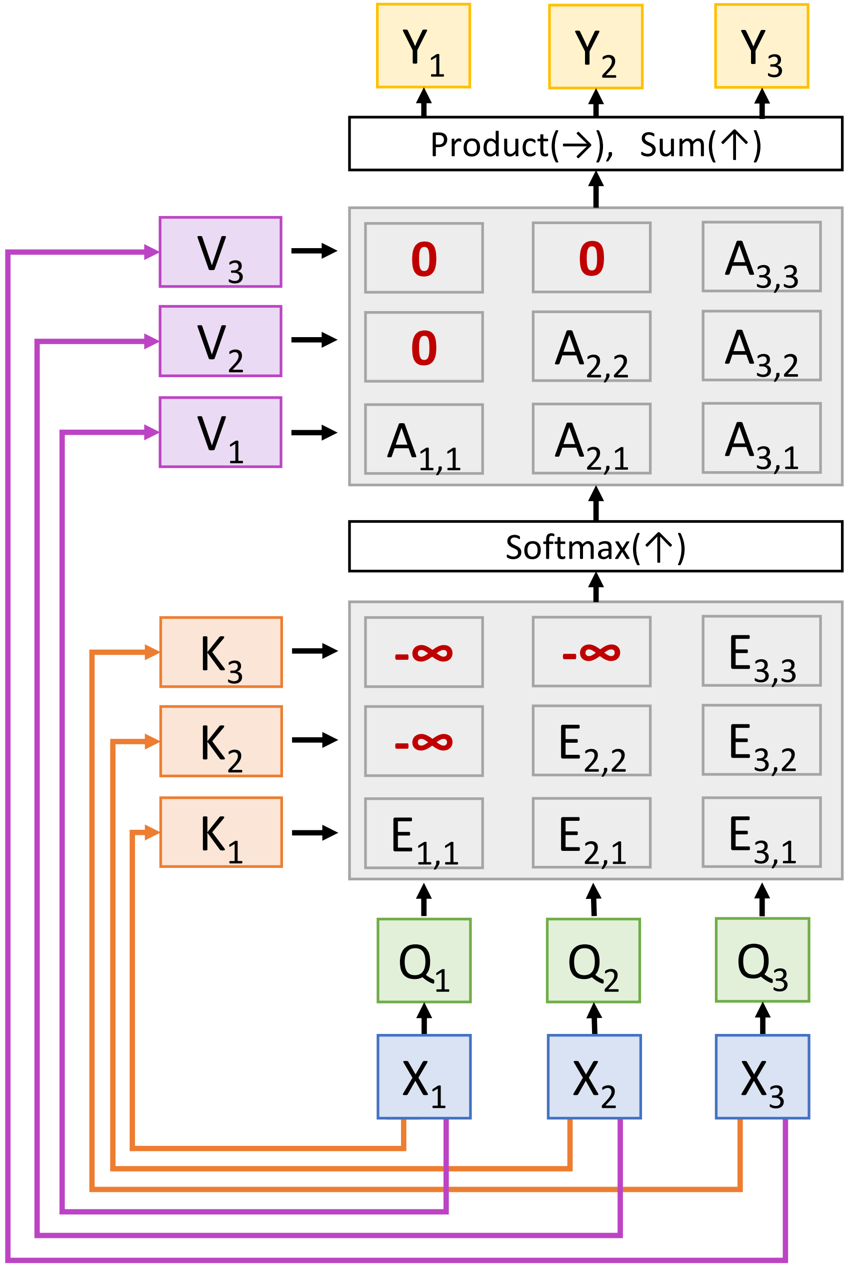
Key vectors: $\mathbf{K} = \mathbf{X}\mathbf{W_K}$ (Shape: N_X x D_Q)

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

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

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) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$

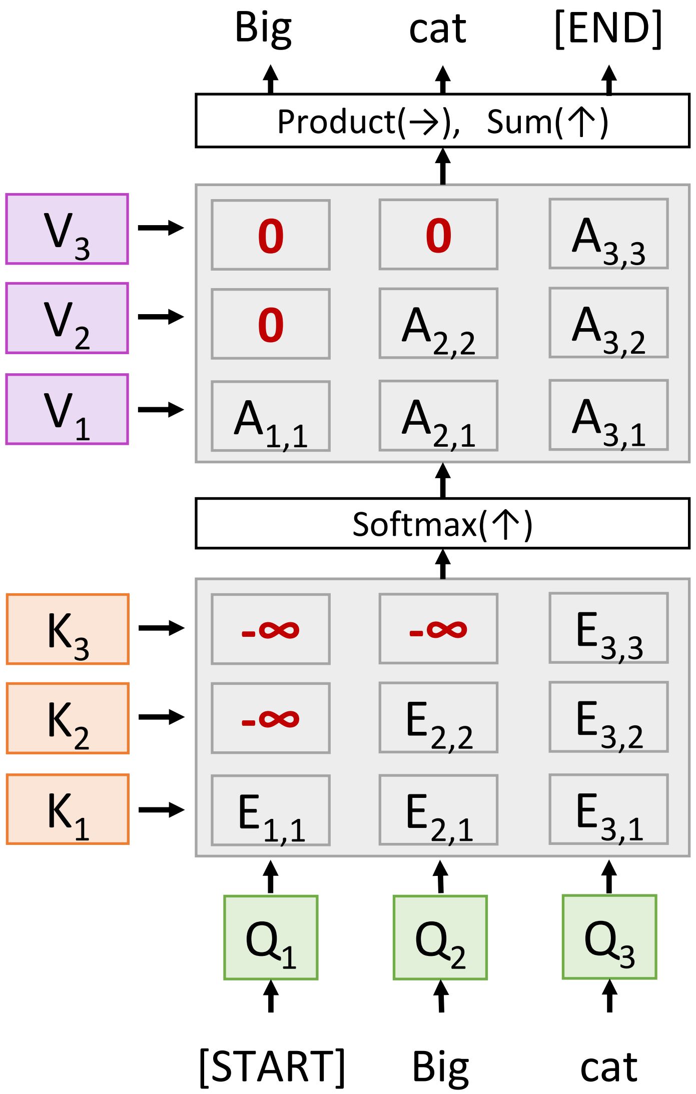


Masked Self-Attention Layer

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

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



Multihead Self-Attention

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)

Use H independent "Attention Heads" in parallel

Computation:

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

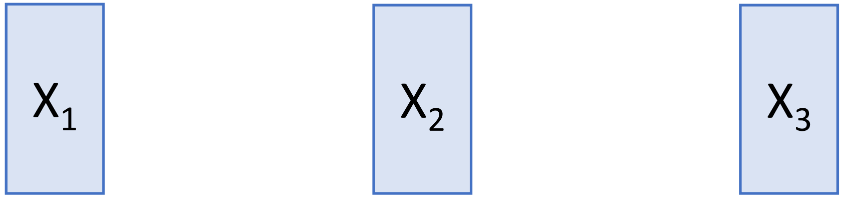
Key vectors: $\mathbf{K} = \mathbf{XW_K}$ (Shape: N_X x D_Q)

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

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

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) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$



Multihead Self-Attention

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)

Use H independent "Attention Heads" in parallel

Computation:

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

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

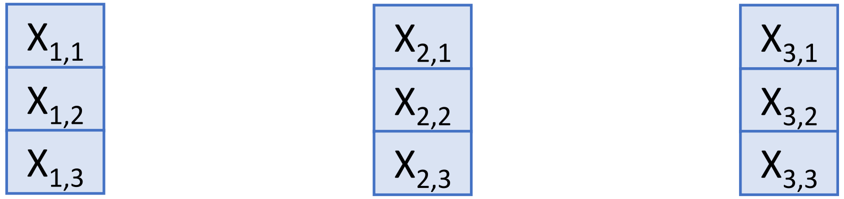
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: N_X x D_V)

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

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) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$

Split



Multihead Self-Attention

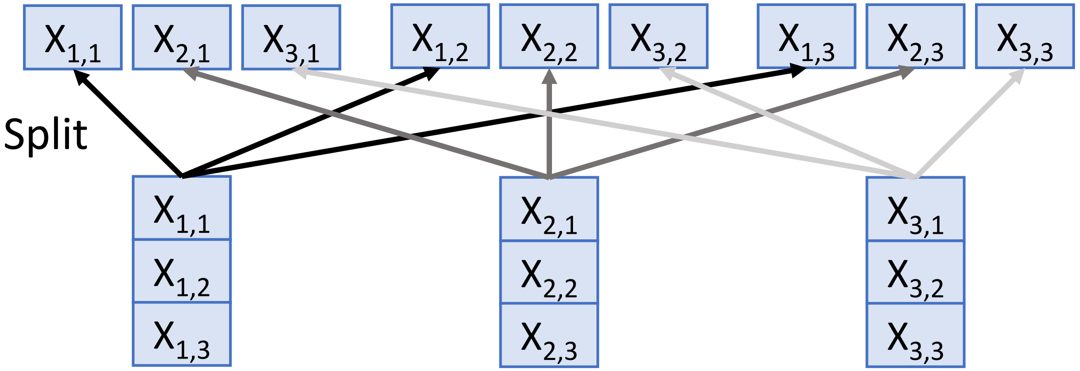
Inputs:

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)

Use H independent “Attention Heads” in parallel

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: $E = QK^T / \sqrt{D_Q}$ (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 N_X)
Output vectors: $Y = AV$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$



Multihead Self-Attention

Run self-attention in parallel on each set of input vectors (different weights per head)

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)

Use H independent "Attention Heads" in parallel

Computation:

Query vectors: $\mathbf{Q} = \mathbf{X}\mathbf{W_Q}$

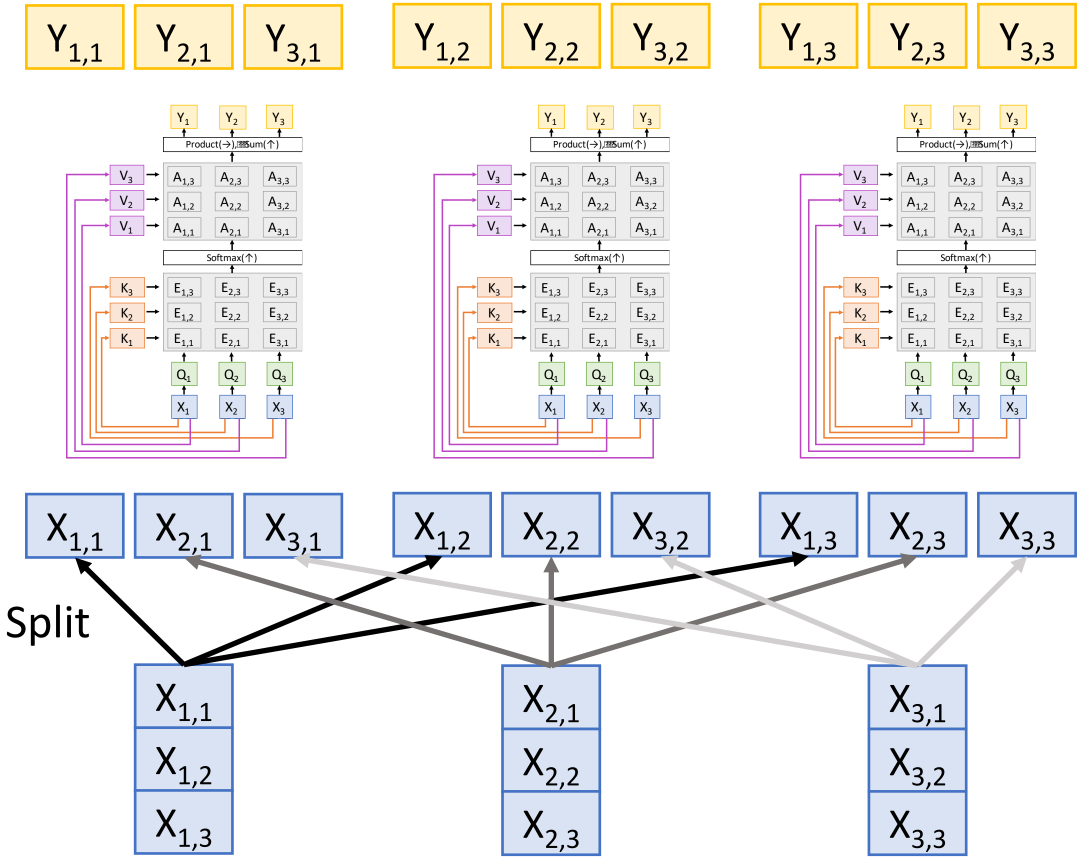
Key vectors: $\mathbf{K} = \mathbf{X}\mathbf{W_K}$ (Shape: N_X x D_Q)

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

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

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) $Y_i = \sum_j A_{i,j}\mathbf{V_j}$



Multihead Self-Attention

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

Input vectors: $\mathbf{X}$ (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: $N_X \times D_Q$)

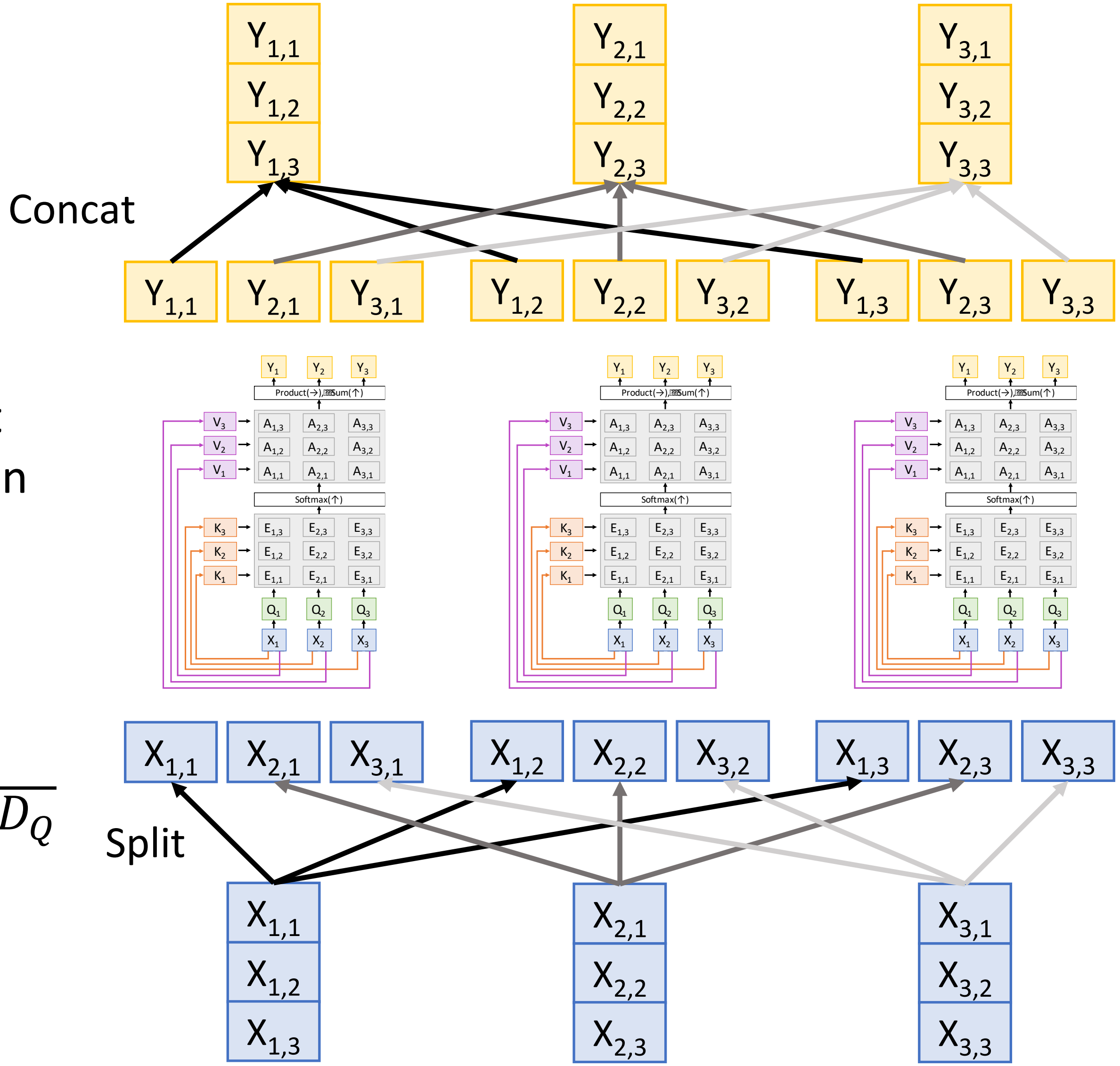
Value Vectors: $\mathbf{V} = \mathbf{XW_V}$ (Shape: $N_X \times D_V$)

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

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

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

Use H independent "Attention Heads" in parallel


$Y_{1,1}$
$Y_{1,2}$
$Y_{1,3}$
$Y_{2,1}$
$Y_{2,2}$
$Y_{2,3}$
$Y_{3,1}$
$Y_{3,2}$
$Y_{3,3}$
Concat
$Y_{1,1}$ $Y_{2,1}$ $Y_{3,1}$ $Y_{1,2}$ $Y_{2,2}$ $Y_{3,2}$ $Y_{1,3}$ $Y_{2,3}$ $Y_{3,3}$
$X_{1,1}$ $X_{2,1}$ $X_{3,1}$ $X_{1,2}$ $X_{2,2}$ $X_{3,2}$ $X_{1,3}$ $X_{2,3}$ $X_{3,3}$
Split
$X_{1,1}$
$X_{1,2}$
$X_{1,3}$
$X_{2,1}$
$X_{2,2}$
$X_{2,3}$
$X_{3,1}$
$X_{3,2}$
$X_{3,3}$

Multihead Self-Attention

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)

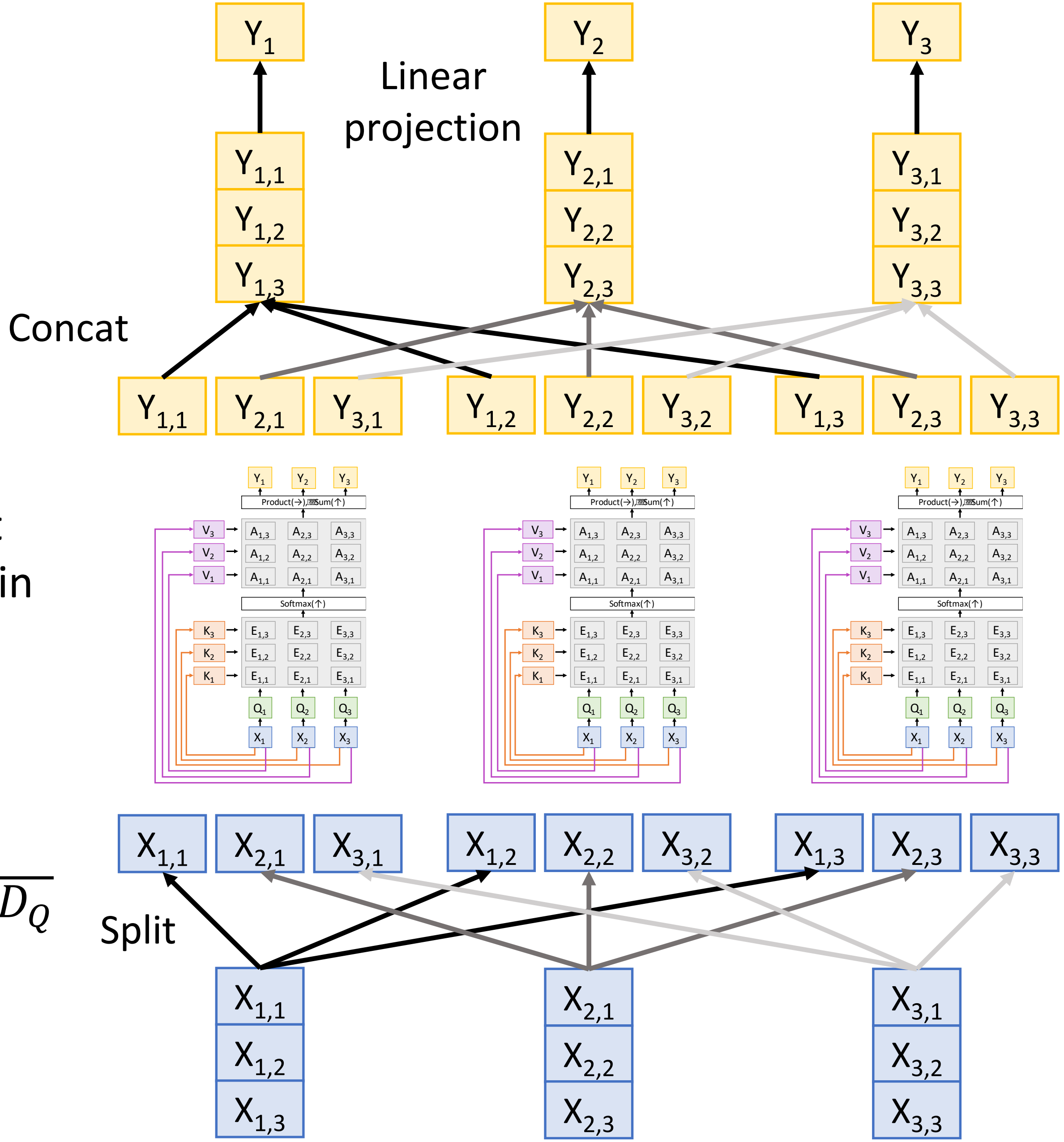
Value Vectors: $V = XW_V$ (Shape: N_X x D_V)

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

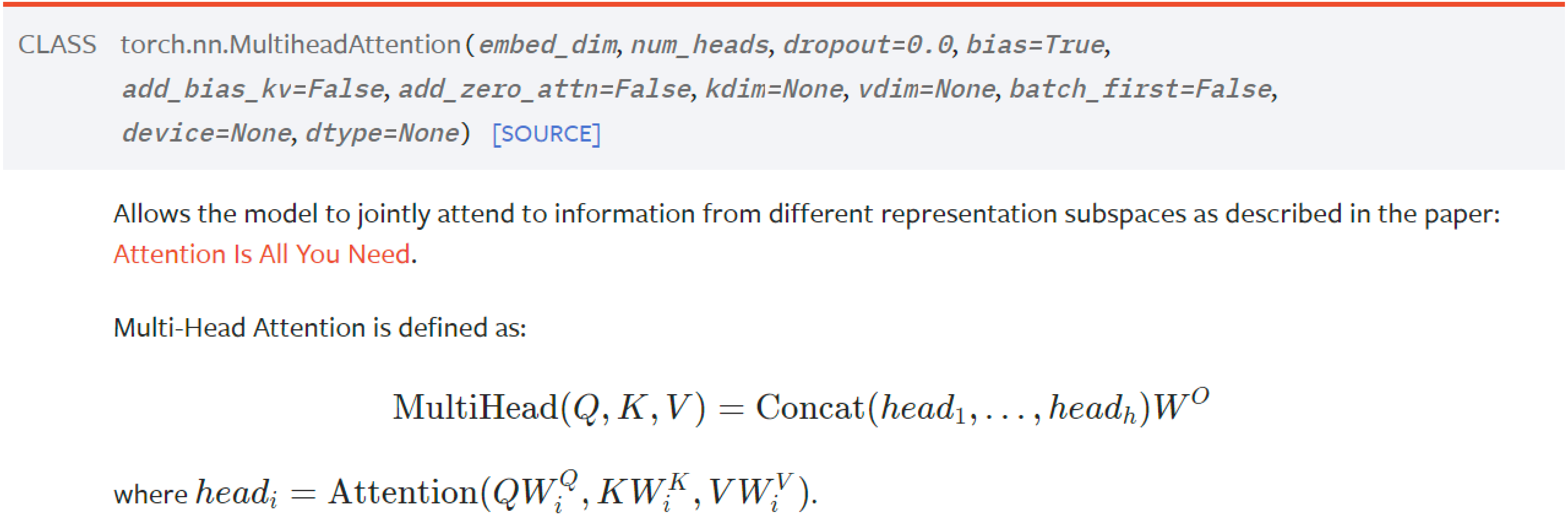
Attention weights: $A = \mathrm{softmax}(E, \mathrm{dim}=1)$ (Shape: N_X x N_X)

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

Use H independent "Attention Heads" in parallel



PyTorch MutiheadAttention Layer



CLASS torch.nn.MultiheadAttention(*embed_dim*, *num_heads*, *dropout=0.0*, *bias=True*, *add_bias_kv=False*, *add_zero_attn=False*, *kdim=None*, *vdim=None*, *batch_first=False*, *device=None*, *dtype=None*) [SOURCE]

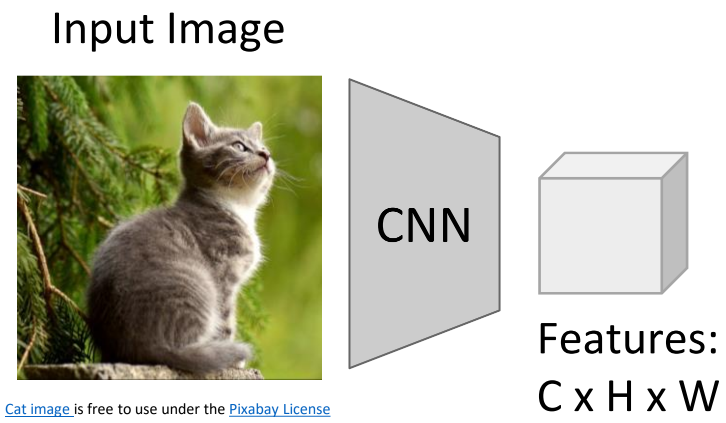
Allows the model to jointly attend to information from different representation subspaces as described in the paper: Attention Is All You Need.

Multi-Head Attention is defined as:

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(head_1, \ldots, head_h)W^O$$

where $head_i = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$.

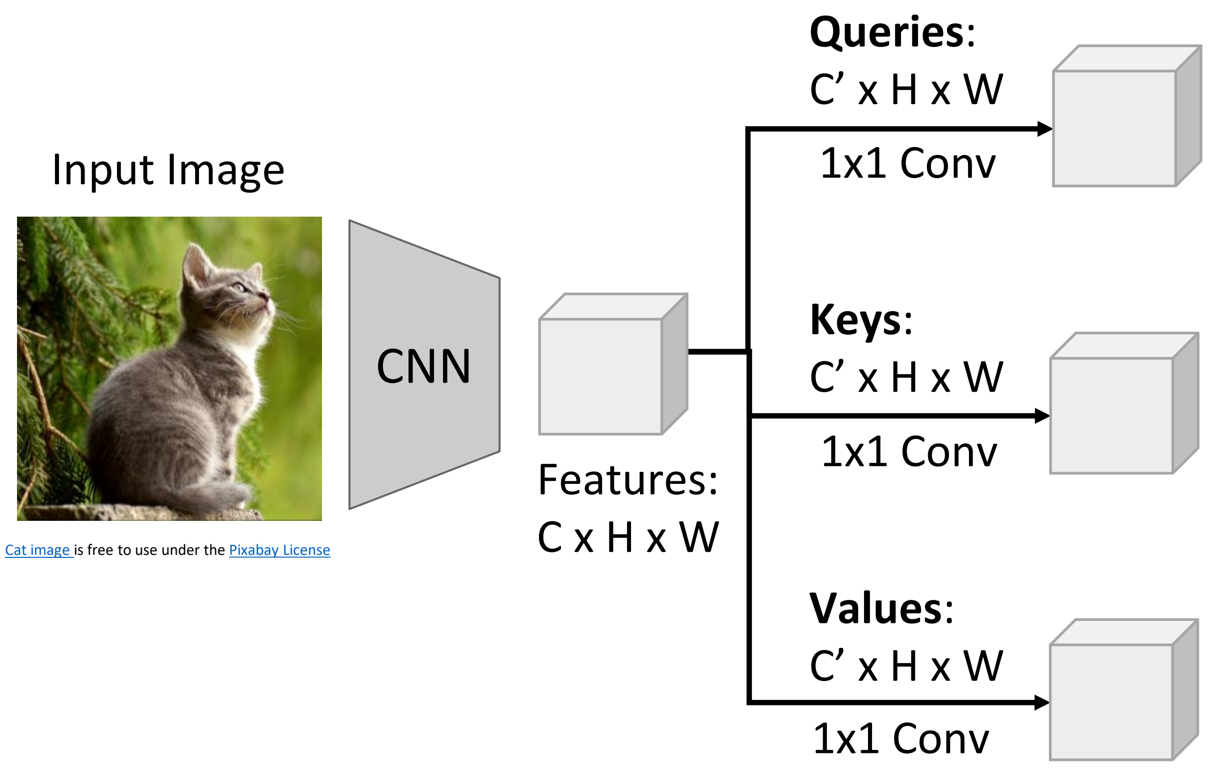
Example: CNN with Self-Attention



Cat image is free to use under the Pixabay License

Zhang et al, "Self-Attention Generative Adversarial Networks", ICML 2018

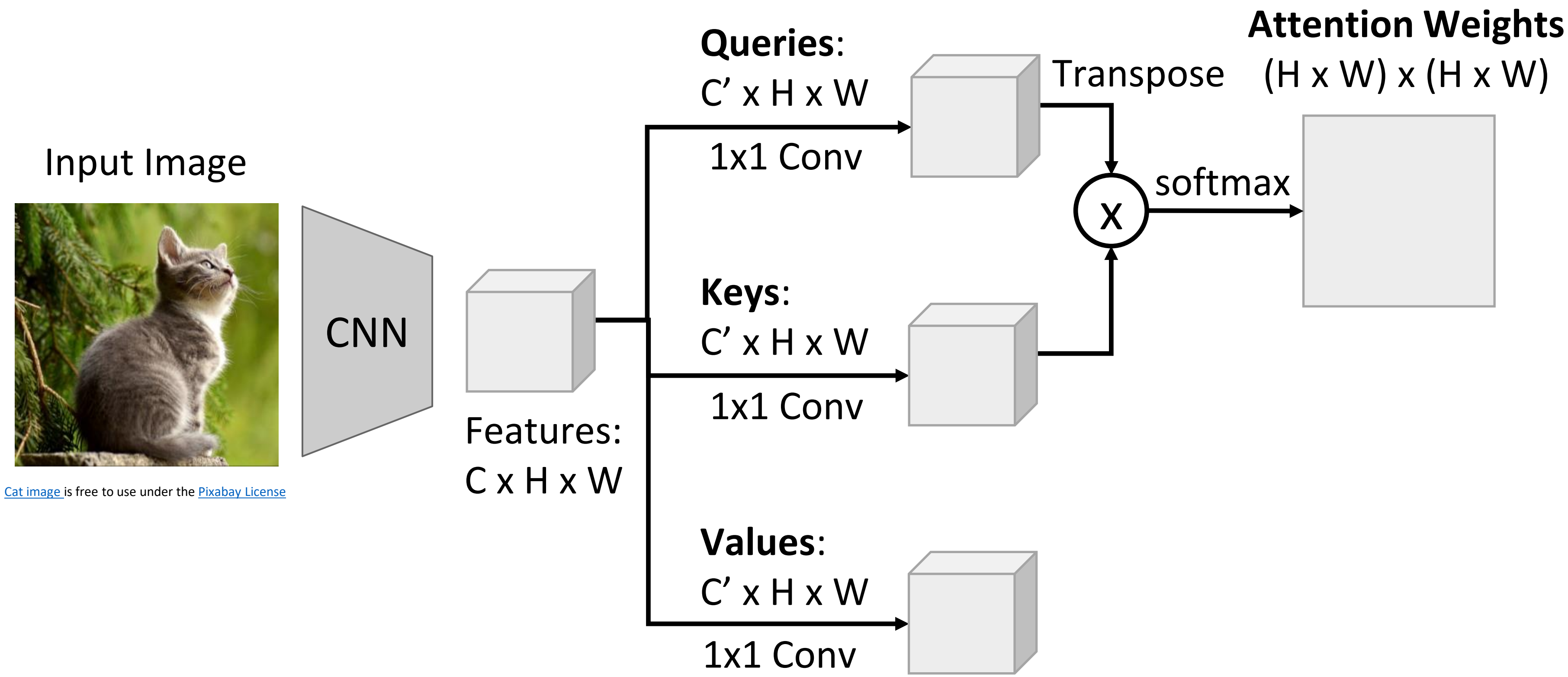
Example: CNN with Self-Attention



Cat image is free to use under the Pixabay License

Zhang et al, “Self-Attention Generative Adversarial Networks”, ICML 2018

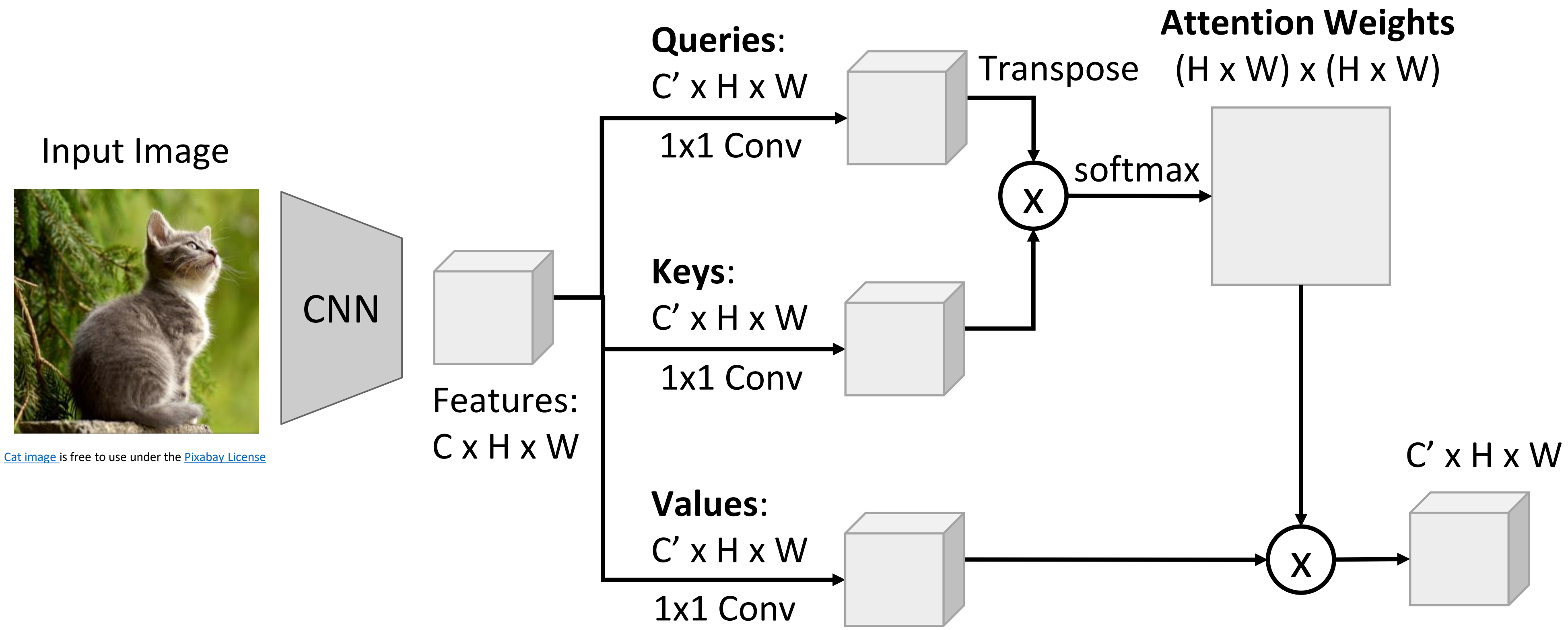
Example: CNN with Self-Attention



Cat image is free to use under the Pixabay License

Zhang et al, “Self-Attention Generative Adversarial Networks”, ICML 2018

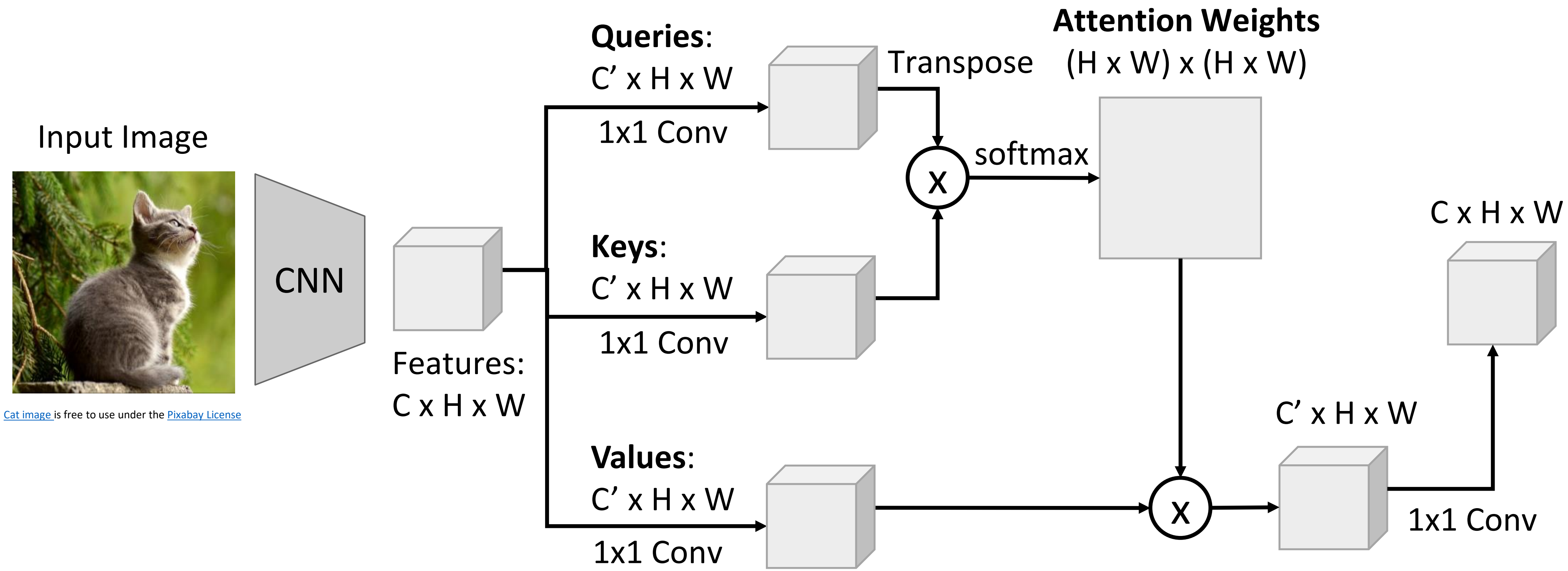
Example: CNN with Self-Attention



Cat image is free to use under the Pixabay License

Zhang et al, “Self-Attention Generative Adversarial Networks”, ICML 2018

Example: CNN with Self-Attention



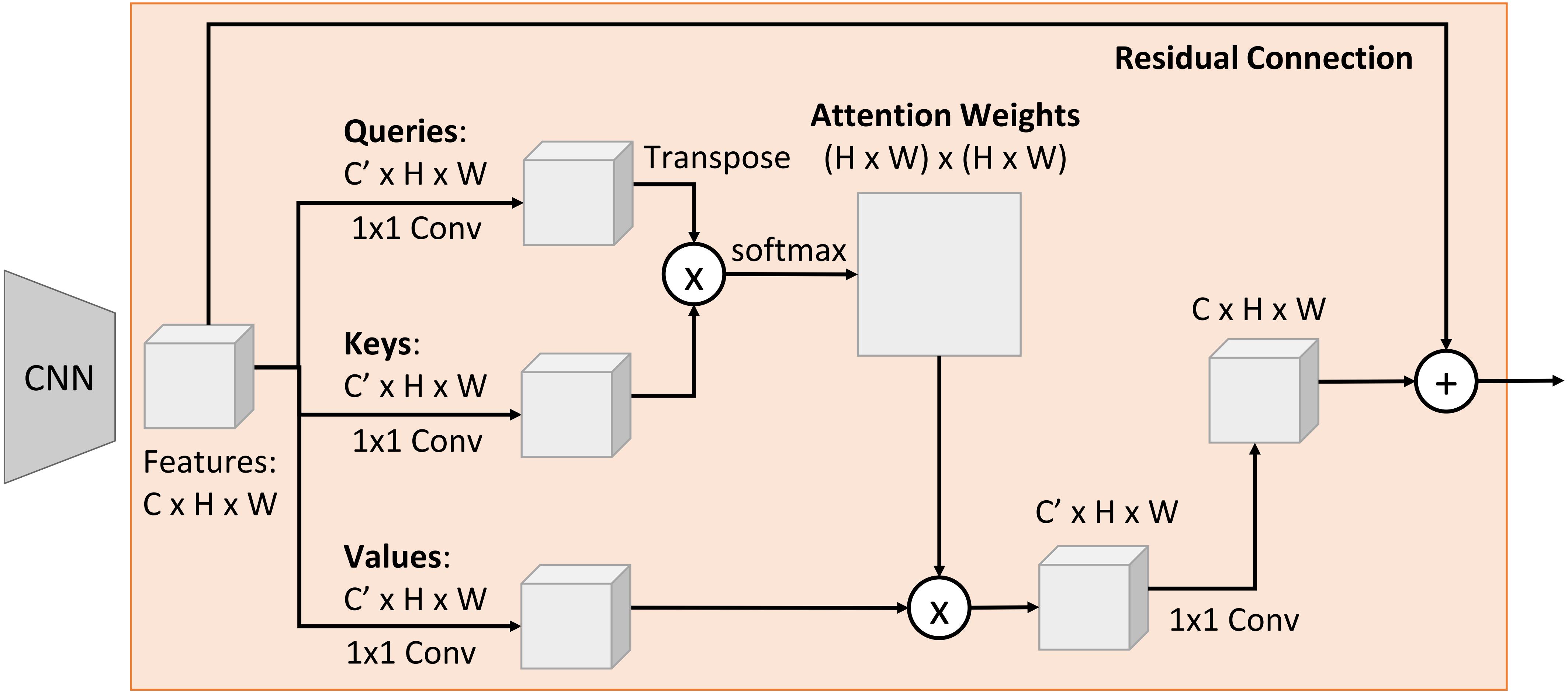
Cat image is free to use under the Pixabay License

Zhang et al, “Self-Attention Generative Adversarial Networks”, ICML 2018

Example: CNN with Self-Attention

Input Image

Cat image is free to use under the Pixabay License



CNN

Features:
C x H x W

Queries:
C’ x H x W
1x1 Conv

Keys:
C’ x H x W
1x1 Conv

Values:
C’ x H x W
1x1 Conv

Transpose

x

softmax

Attention Weights
(H x W) x (H x W)

x

C’ x H x W
1x1 Conv

C x H x W

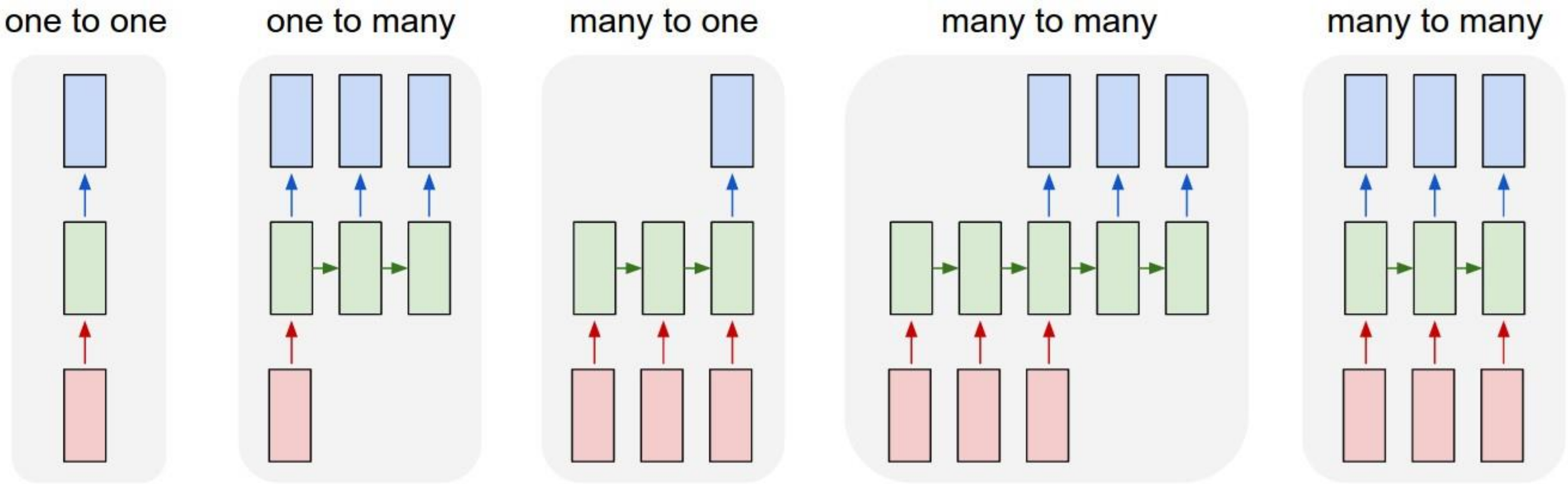
+

Residual Connection

Self-Attention Module

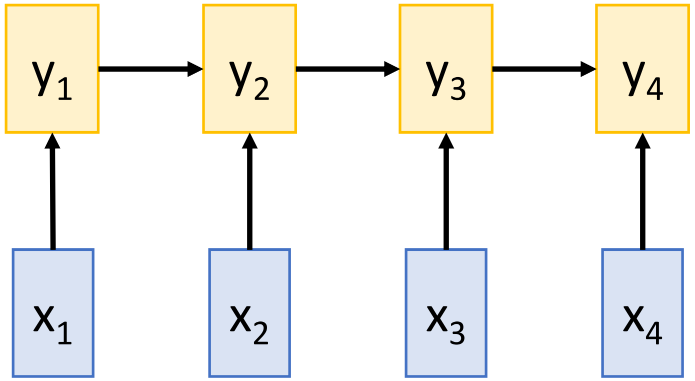
Zhang et al, “Self-Attention Generative Adversarial Networks”, ICML 2018

Recall: Recurrent Neural Networks



Three Ways of Processing Sequences

Recurrent Neural Network

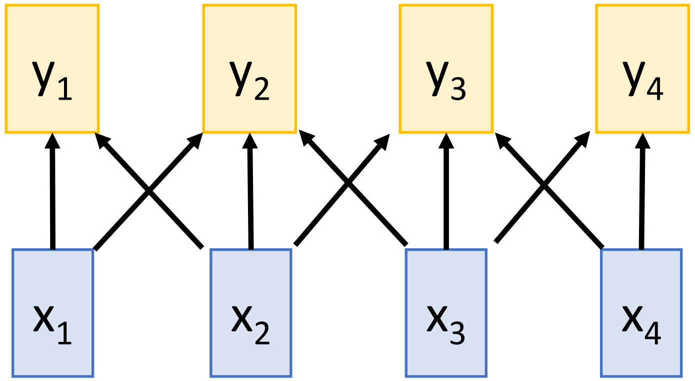


Works on **Ordered Sequences**

(+) Good at long sequences: After one RNN layer, h_T "sees" the whole sequence

(-) Not parallelizable: need to compute hidden states sequentially

1D Convolution

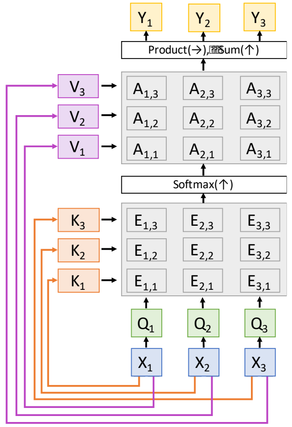


Works on **Multidimensional Grids**

(-) Bad at long sequences: Need to stack many conv layers for outputs to "see" the whole sequence

(+) Highly parallel: Each output can be computed in parallel

Self-Attention



Works on **Sets of Vectors**

(-) Good at long sequences: after one self-attention layer, each output "sees" all inputs!

(+) Highly parallel: Each output can be computed in parallel

(-) Very memory intensive

Three Ways of Processing Sequences

Recurrent Neural Network

1D Convolution

Self-Attention

Attention is all you need

Vaswani et al, NeurIPS 2017

Works on **Ordered Sequences**

(+) Good at long sequences: After one RNN layer, h_T "sees" the whole sequence

(-) Not parallelizable: need to compute hidden states sequentially

Works on **Multidimensional Grids**

(-) Bad at long sequences: Need to stack many conv layers for outputs to "see" the whole sequence

(+) Highly parallel: Each output can be computed in parallel

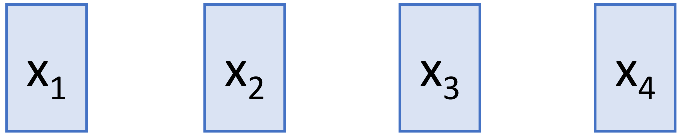
Works on **Sets of Vectors**

(-) Good at long sequences: after one self-attention layer, each output "sees" all inputs!

(+) Highly parallel: Each output can be computed in parallel

(-) Very memory intensive

The Transformer

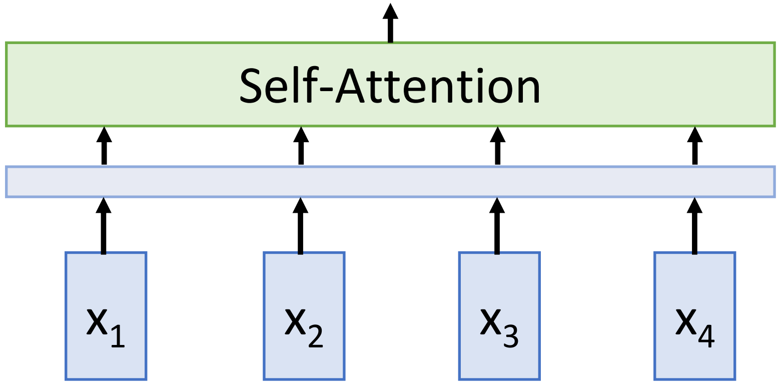


x_1 x_2 x_3 x_4

Vaswani et al, “Attention is all you need”, NeurIPS 2017

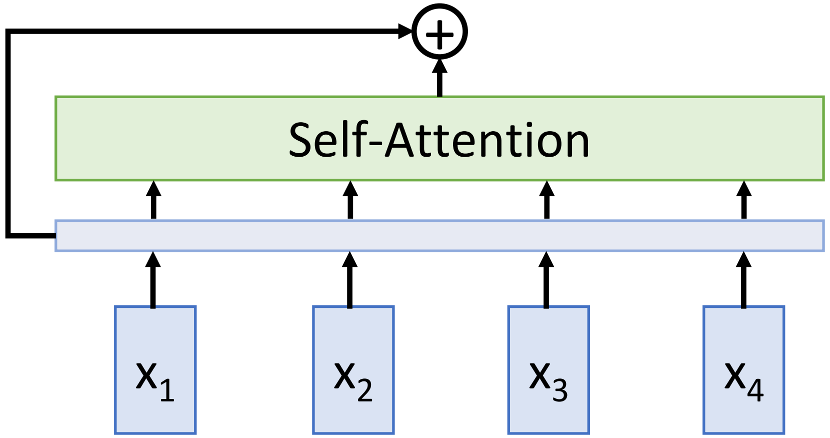
The Transformer

All vectors interact with each other



Vaswani et al, “Attention is all you need”, NeurIPS 2017

The Transformer



Residual connection

All vectors interact with each other

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

The Transformer

Recall **Layer Normalization**:

Given $h_1, \ldots, h_N$ (Shape: D)

scale: γ (Shape: D)

shift: β (Shape: D)

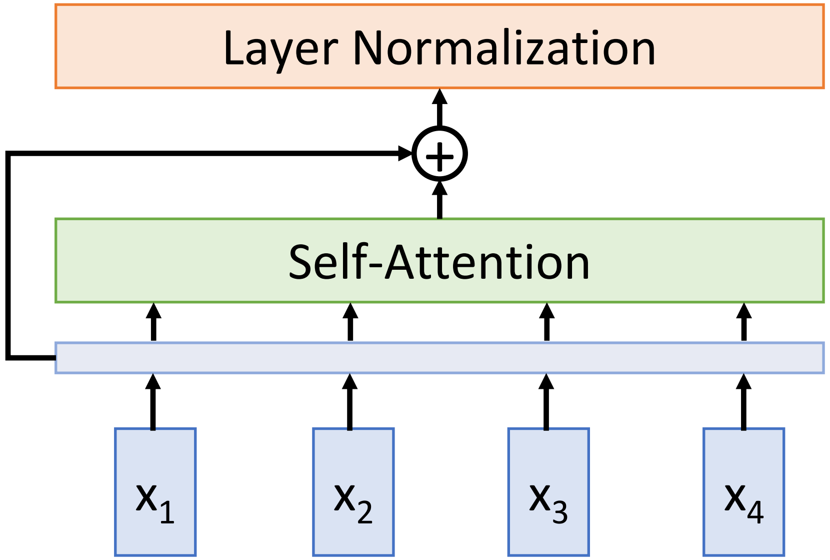
$\mu_i = (\sum_j h_{i,j})/D$ (scalar)

$\sigma_i = (\sum_j (h_{i,j} - \mu_i)^2/D)^{1/2}$ (scalar)

$z_i = (h_i - \mu_i) / \sigma_i$

$y_i = \gamma * z_i + \beta$

Ba et al, 2016



Residual connection

All vectors interact with each other

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

The Transformer

Recall **Layer Normalization**:

Given $h_1, \ldots, h_N$ (Shape: D)

scale: γ (Shape: D)

shift: β (Shape: D)

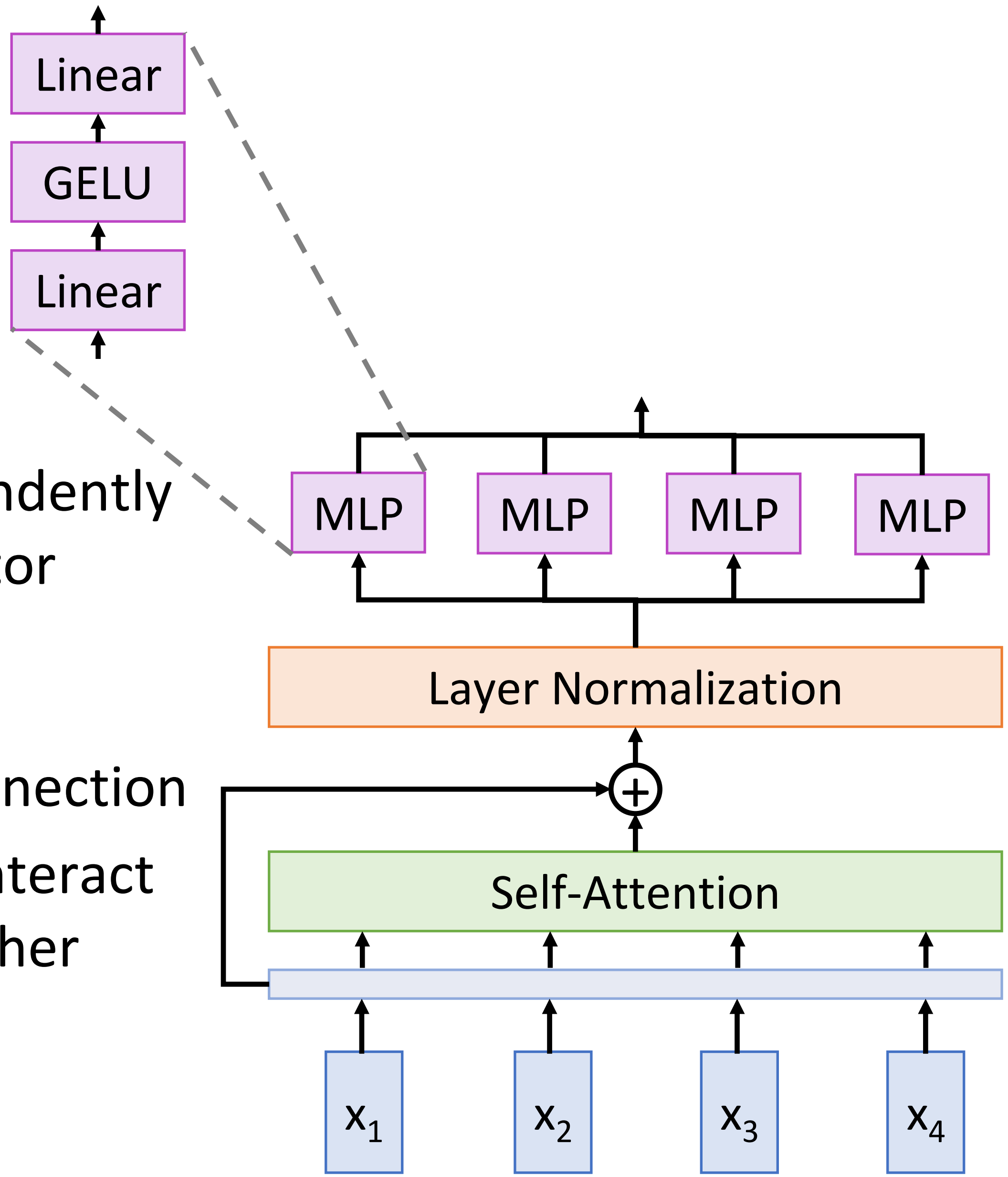
$\mu_i = (\sum_j h_{i,j})/D$ (scalar)

$\sigma_i = (\sum_j (h_{i,j} - \mu_i)^2/D)^{1/2}$ (scalar)

$z_i = (h_i - \mu_i) / \sigma_i$

$y_i = \gamma * z_i + \beta$

Ba et al, 2016



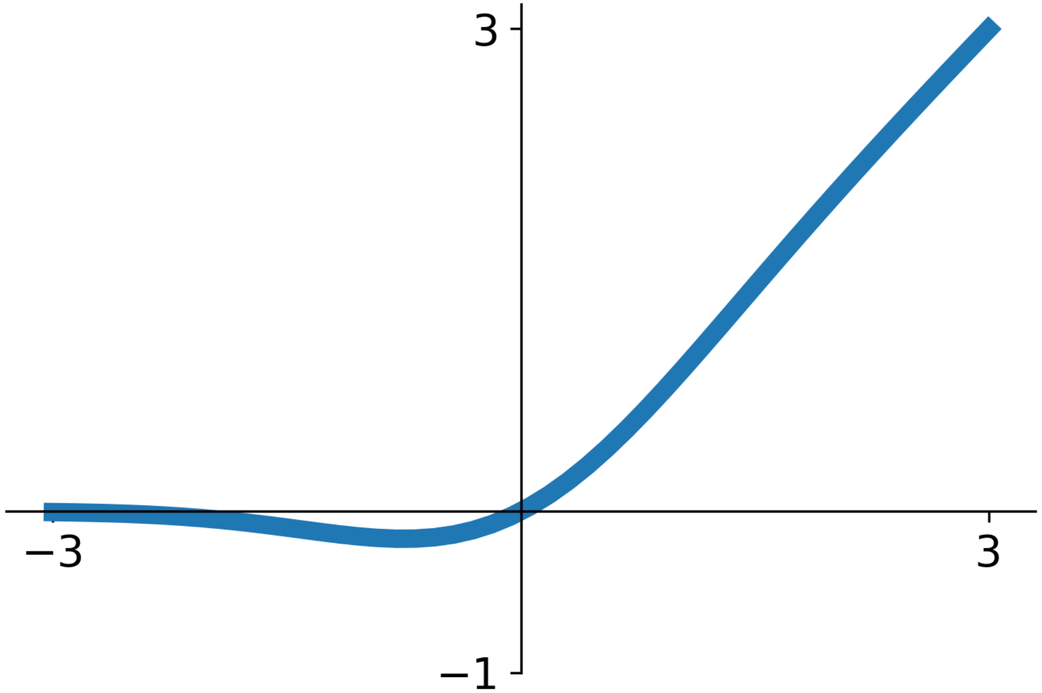
MLP independently on each vector

Residual connection

All vectors interact with each other

Vaswani et al, “Attention is all you need”, NeurIPS 2017

Activation Function: Gaussian Error Linear Unit (GELU)



$$X \sim N(0,1)$$

$$gelu(x) = xP(X \le x) = \frac{x}{2}\left(1 + \mathrm{erf}\left(x/\sqrt{2}\right)\right)$$

$$\approx x\sigma(1.702x)$$

- **Idea**: Multiply input by 0 or 1 at random; large values more likely to be multiplied by 1, small values more likely to be multiplied by 0 (data-dependent dropout)
- Take expectation over randomness
- Very common in Transformers (BERT, GPT, ViT)
- Cf. **Swish** (or **SiLU**): $x\sigma(x)$

Hendrycks and Gimpel, Gaussian Error Linear Units (GELUs), 2016
Ramachandran et al, Swish: a Self-Gated Activation Function, 2017

The Transformer

Recall **Layer Normalization**:

Given $h_1, \ldots, h_N$ (Shape: D)

scale: γ (Shape: D)

shift: β (Shape: D)

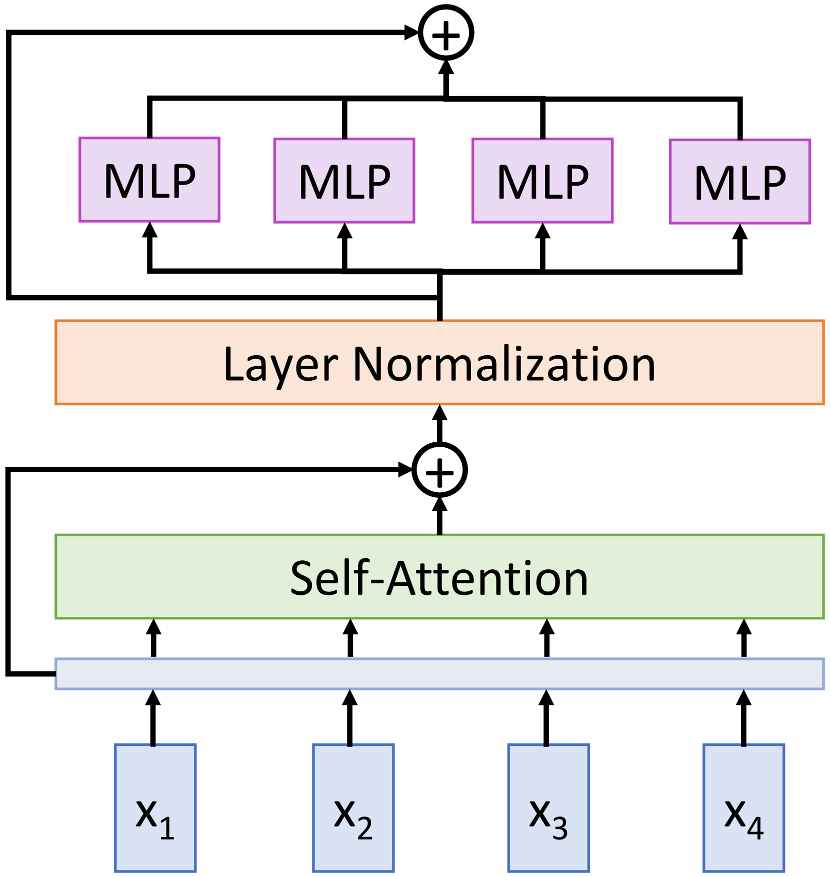
$\mu_i = (\sum_j h_{i,j})/D$ (scalar)

$\sigma_i = (\sum_j (h_{i,j} - \mu_i)^2/D)^{1/2}$ (scalar)

$z_i = (h_i - \mu_i) / \sigma_i$

$y_i = \gamma * z_i + \beta$

Ba et al, 2016



Residual connection

MLP independently on each vector

Residual connection

All vectors interact with each other

Vaswani et al, “Attention is all you need”, NeurIPS 2017

The Transformer

Recall **Layer Normalization**:

Given $h_1, \ldots, h_N$ (Shape: D)

scale: γ (Shape: D)

shift: β (Shape: D)

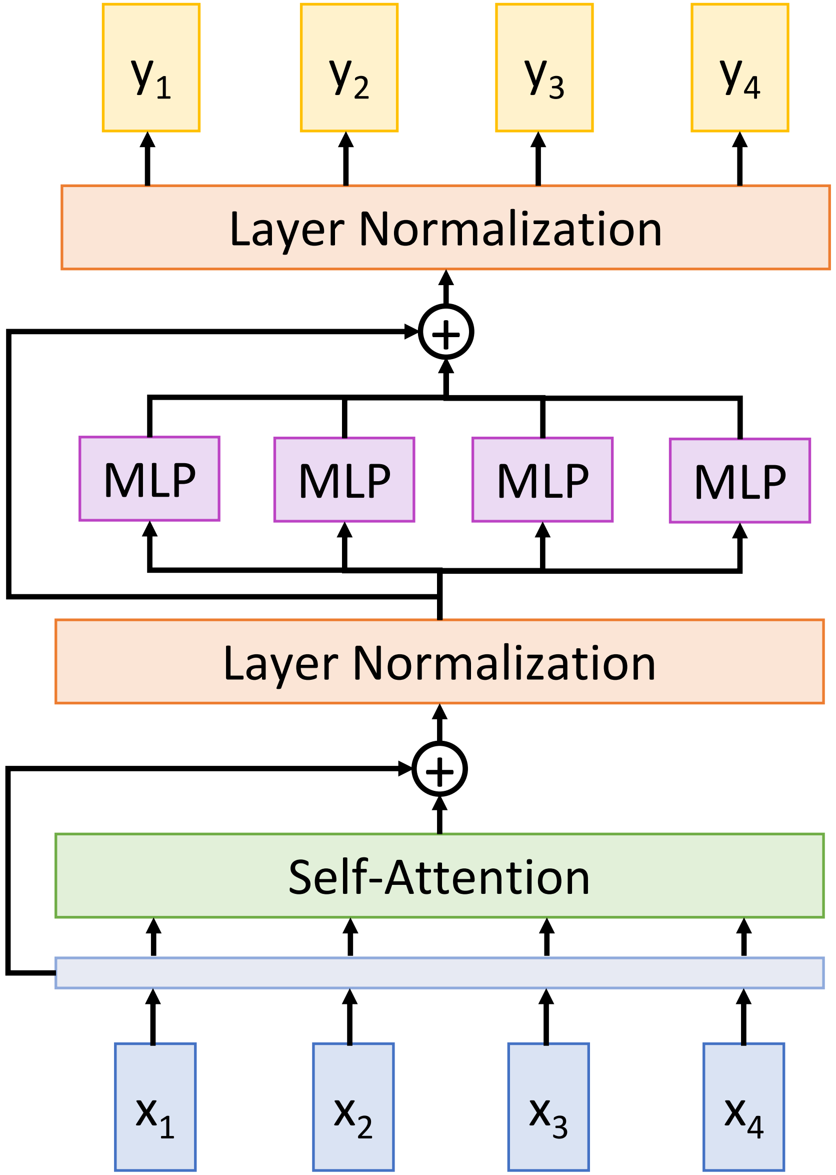
$\mu_i = (\sum_j h_{i,j})/D$ (scalar)

$\sigma_i = (\sum_j (h_{i,j} - \mu_i)^2/D)^{1/2}$ (scalar)

$z_i = (h_i - \mu_i) / \sigma_i$

$y_i = \gamma * z_i + \beta$

Ba et al, 2016



Residual connection

MLP independently on each vector

Residual connection

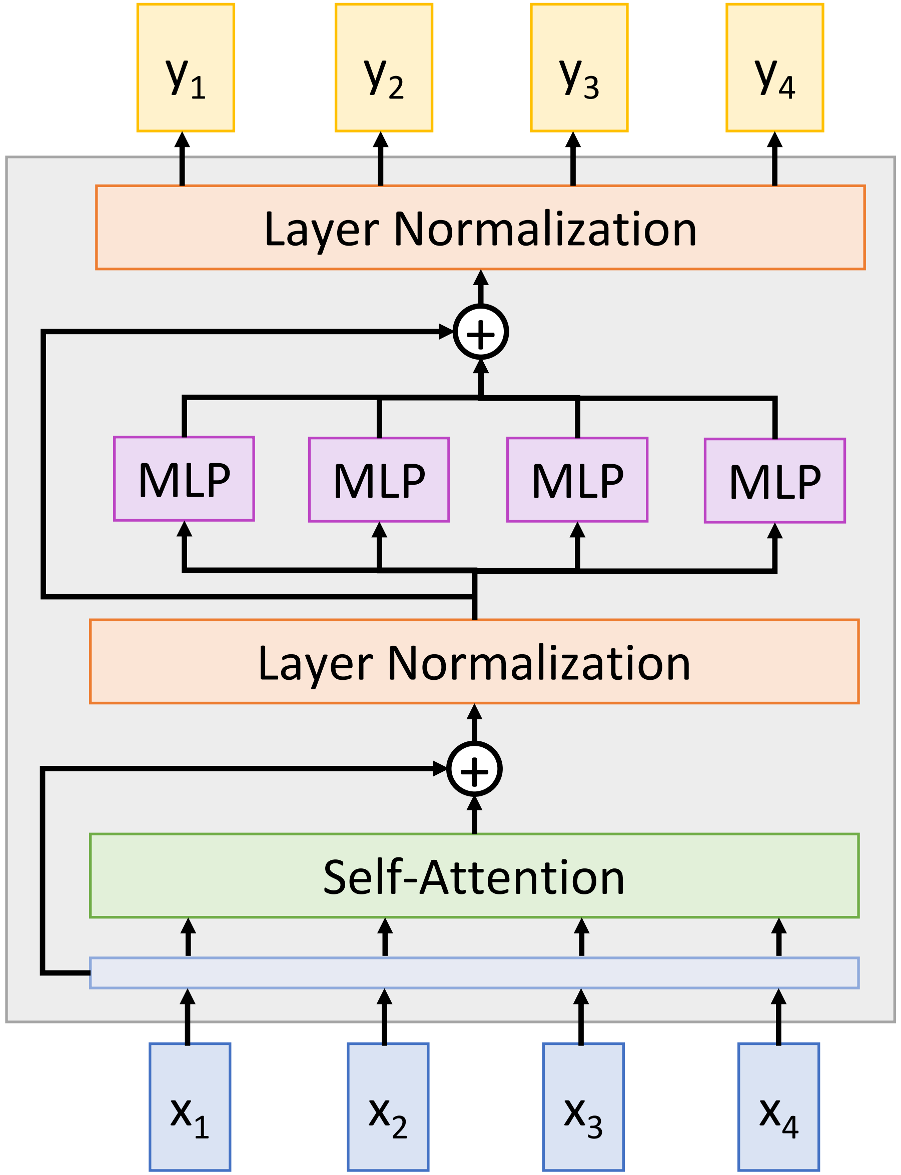
All vectors interact with each other

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

The Transformer

<u>Transformer Block</u>:
Input: Set of vectors x
Output: Set of vectors y

Hyperparameters:
- Number of blocks
- Number of heads per block
- Width (channels per head, FFN width)



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

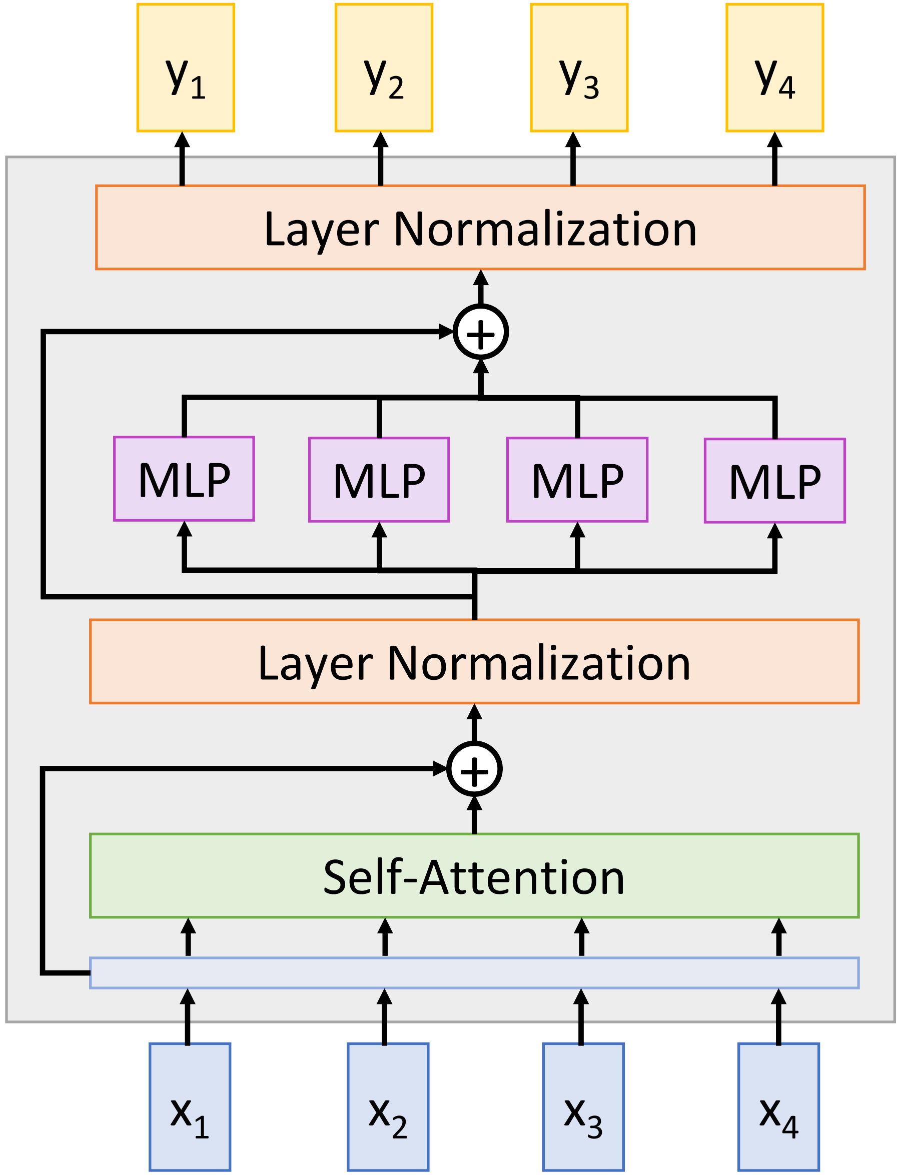
The Transformer

Transformer Block:
Input: Set of vectors x
Output: Set of vectors y

Self-attention is the only interaction between vectors!

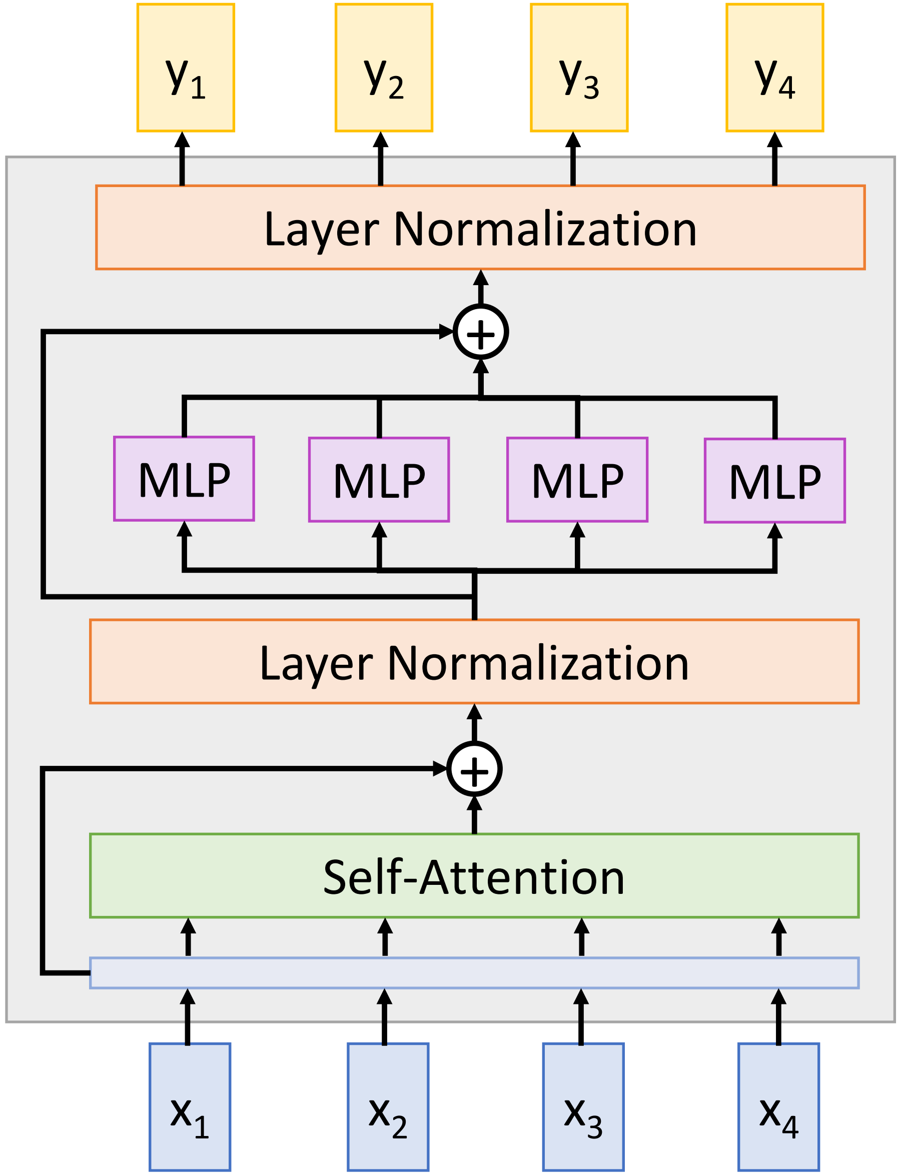
Layer norm and MLP work independently per vector

Highly scalable, highly parallelizable



Vaswani et al, “Attention is all you need”, NeurIPS 2017

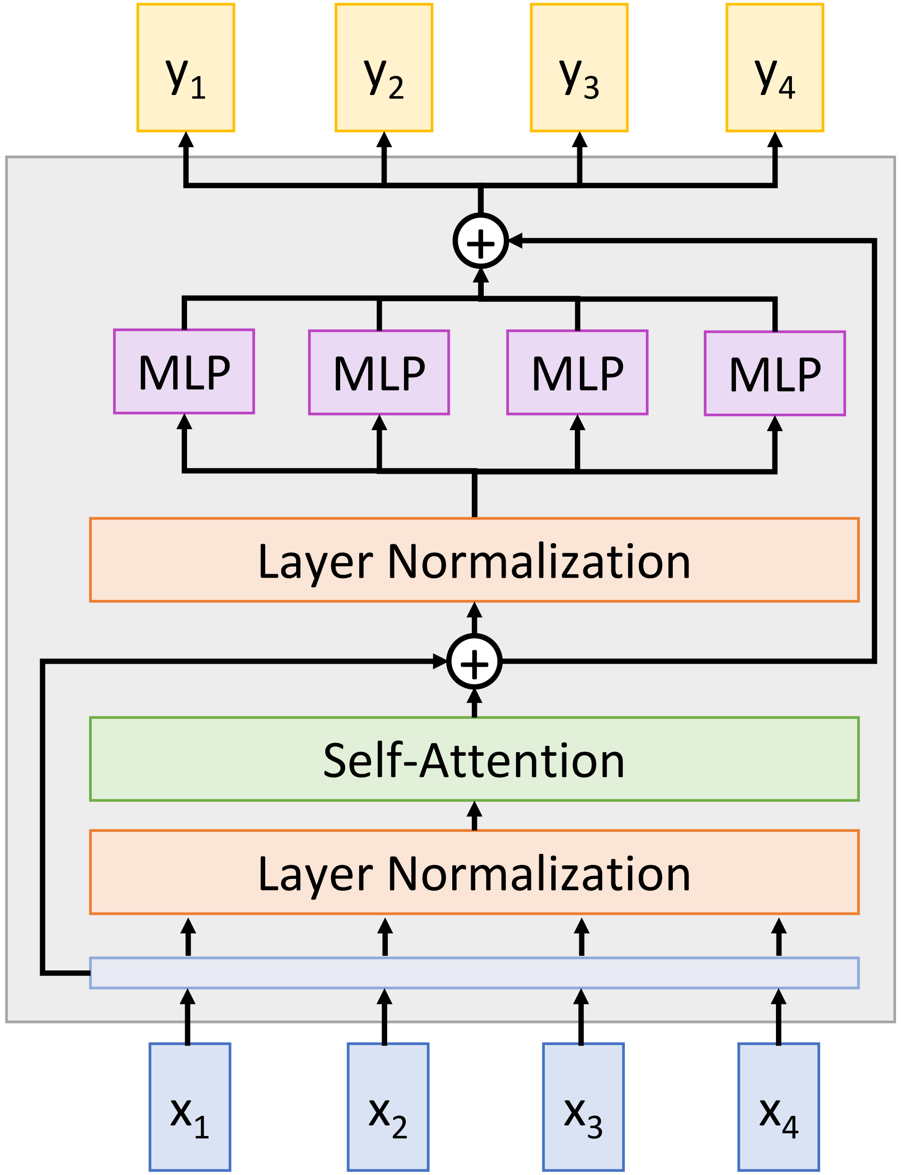
Post-Norm Transformer



Layer normalization is
after residual connections

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

Pre-Norm Transformer



Layer normalization is **inside** residual connections

Gives more stable training, commonly used in practice

Baevski & Auli, "Adaptive Input Representations for Neural Language Modeling", arXiv 2018

The Transformer

Transformer Block:
Input: Set of vectors x
Output: Set of vectors y

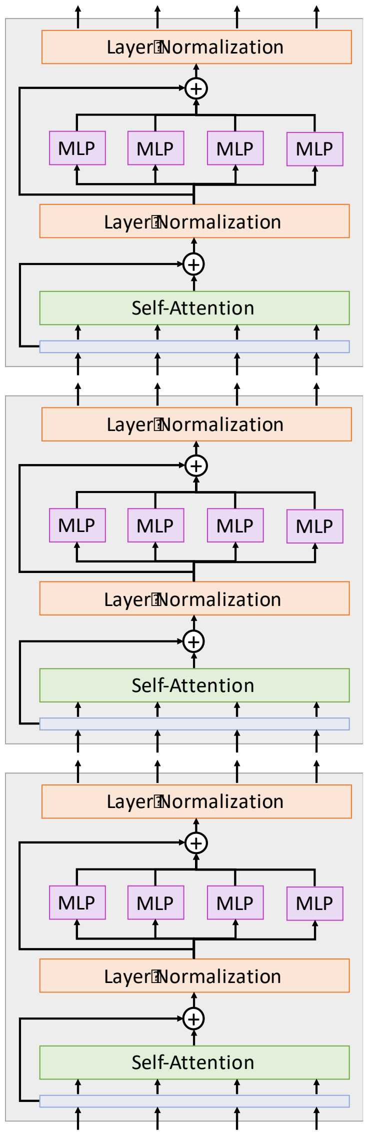
Self-attention is the only interaction between vectors!

Layer norm and MLP work independently per vector

Highly scalable, highly parallelizable

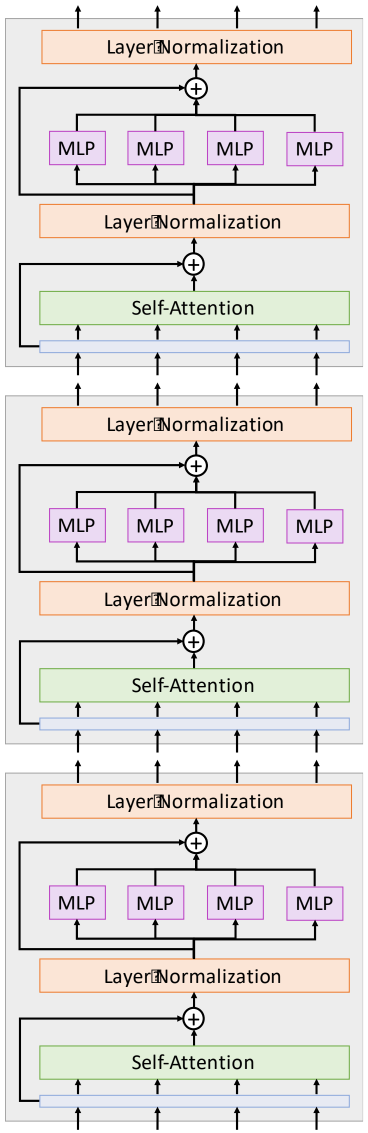
A **Transformer** is a sequence of transformer blocks

Vaswani et al:
12 blocks, D_Q=512, 6 heads



Vaswani et al, “Attention is all you need”, NeurIPS 2017

The Transformer: Transfer Learning



“ImageNet Moment for Natural Language Processing”

Pre-training:
Download a lot of text from the internet

Train a giant Transformer model for language modeling

Fine-tuning:
Fine-tune the Transformer on your own NLP task

Devlin et al, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", EMNLP 2018

Scaling up Transformers

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)

Vaswani et al, “Attention is all you need”, NeurIPS 2017

Scaling up Transformers

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)
BERT-Base	12	768	12	110M	13 GB	
BERT-Large	24	1024	16	340M	13 GB	

Devlin et al, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", EMNLP 2018

Scaling up Transformers

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)
BERT-Base	12	768	12	110M	13 GB	
BERT-Large	24	1024	16	340M	13 GB	
XLNet-Large	24	1024	16	~340M	126 GB	512x TPU-v3 (2.5 days)
RoBERTa	24	1024	16	355M	160 GB	1024x V100 GPU (1 day)

Yang et al, XLNet: Generalized Autoregressive Pretraining for Language Understanding", 2019
Liu et al, "RoBERTa: A Robustly Optimized BERT Pretraining Approach", 2019

Scaling up Transformers

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)
BERT-Base	12	768	12	110M	13 GB	
BERT-Large	24	1024	16	340M	13 GB	
XLNet-Large	24	1024	16	~340M	126 GB	512x TPU-v3 (2.5 days)
RoBERTa	24	1024	16	355M	160 GB	1024x V100 GPU (1 day)
GPT-2	48	1600	?	1.5B	40 GB	

Radford et al, "Language models are unsupervised multitask learners", 2019

Scaling up Transformers

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)
BERT-Base	12	768	12	110M	13 GB	
BERT-Large	24	1024	16	340M	13 GB	
XLNet-Large	24	1024	16	~340M	126 GB	512x TPU-v3 (2.5 days)
RoBERTa	24	1024	16	355M	160 GB	1024x V100 GPU (1 day)
GPT-2	48	1600	?	1.5B	40 GB	
Megatron-LM	72	3072	32	8.3B	174 GB	512x V100 GPU (9 days)

Shoeybi et al, "Megatron-LM: Training Multi-Billion Parameter Languge Models using Model Parallelism", 2019

Scaling up Transformers

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)
BERT-Base	12	768	12	110M	13 GB	
BERT-Large	24	1024	16	340M	13 GB	
XLNet-Large	24	1024	16	~340M	126 GB	512x TPU-v3 (2.5 days)
RoBERTa	24	1024	16	355M	160 GB	1024x V100 GPU (1 day)
GPT-2	48	1600	?	1.5B	40 GB	
Megatron-LM	72	3072	32	8.3B	174 GB	512x V100 GPU (9 days)
Turing-NLG	78	4256	28	17B	?	256x V100 GPU

Microsoft, "Turing-NLG: A 17-billion parameter language model by Microsoft", 2020

Scaling up Transformers

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)
BERT-Base	12	768	12	110M	13 GB	
BERT-Large	24	1024	16	340M	13 GB	
XLNet-Large	24	1024	16	~340M	126 GB	512x TPU-v3 (2.5 days)
RoBERTa	24	1024	16	355M	160 GB	1024x V100 GPU (1 day)
GPT-2	48	1600	?	1.5B	40 GB	
Megatron-LM	72	3072	32	8.3B	174 GB	512x V100 GPU (9 days)
Turing-NLG	78	4256	28	17B	?	256x V100 GPU
GPT-3	96	12,288	96	175B	694 GB	?

Brown et al, "Language Models are Few-Shot Learners", arXiv 2020

Scaling up Transformers

$3,768,320 on Google Cloud (eval price)

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)
BERT-Base	12	768	12	110M	13 GB	
BERT-Large	24	1024	16	340M	13 GB	
XLNet-Large	24	1024	16	~340M	126 GB	512x TPU-v3 (2.5 days)
RoBERTa	24	1024	16	355M	160 GB	1024x V100 GPU (1 day)
GPT-2	48	1600	?	1.5B	40 GB	
Megatron-LM	72	3072	32	8.3B	174 GB	512x V100 GPU (9 days)
Turing-NLG	78	4256	28	17B	?	256x V100 GPU
GPT-3	96	12,288	96	175B	694 GB	?
Gopher	80	16,384	128	280B	10.55 TB	4096x TPUv3 (38 days)

Rae et al, "Scaling Language Models: Methods, Analysis, & Insights from Training Gopher", arXiv 2021

Scaling up Transformers

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)
BERT-Base	12	768	12	110M	13 GB	
BERT-Large	24	1024	16	340M	13 GB	
XLNet-Large	24	1024	16	~340M	126 GB	512x TPU-v3 (2.5 days)
RoBERTa	24	1024	16	355M	160 GB	1024x V100 GPU (1 day)
GPT-2	48	1600	?	1.5B	40 GB	
Megatron-LM	72	3072	32	8.3B	174 GB	512x V100 GPU (9 days)
Turing-NLG	78	4256	28	17B	?	256x V100 GPU
GPT-3	96	12,288	96	175B	694 GB	?
Gopher	80	16,384	128	280B	10.55 TB	4096x TPUv3 (38 days)
PaLM	118	18,432	256	540B	?	6144x TPUv4 (38 days)

Chowdhery et al, "PaLM: Scaling Language Modeling with Pathways", arXiv 2022

Scaling up Transformers

Specifications are veiled!

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)
BERT-Base	12	768	12	110M	13 GB	
BERT-Large	24	1024	16	340M	13 GB	
XLNet-Large	24	1024	16	~340M	126 GB	512x TPU-v3 (2.5 days)
RoBERTa	24	1024	16	355M	160 GB	1024x V100 GPU (1 day)
GPT-2	48	1600	?	1.5B	40 GB	
Megatron-LM	72	3072	32	8.3B	174 GB	512x V100 GPU (9 days)
Turing-NLG	78	4256	28	17B	?	256x V100 GPU
GPT-3	96	12,288	96	175B	694 GB	?
Gopher	80	16,384	128	280B	10.55 TB	4096x TPUv3 (38 days)
PaLM	118	18,432	256	540B	?	6144x TPUv4 (38 days)
GPT-4	?	?	?	? (1T?)	?	?

OpenAI, "GPT-4 Technical Report", arXiv 2023

Scaling up Transformers

Specifications are veiled!

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)
BERT-Base	12	768	12	110M	13 GB	
BERT-Large	24	1024	16	340M	13 GB	
XLNet-Large	24	1024	16	~340M	126 GB	512x TPU-v3 (2.5 days)
RoBERTa	24	1024	16	355M	160 GB	1024x V100 GPU (1 day)
GPT-2	48	1600	?	1.5B	40 GB	
Megatron-LM	72	3072	32	8.3B	174 GB	512x V100 GPU (9 days)
Turing-NLG	78	4256	28	17B	?	256x V100 GPU
GPT-3	96	12,288	96	175B	694 GB	?
Gopher	80	16,384	128	280B	10.55 TB	4096x TPUv3 (38 days)
PaLM	118	18,432	256	540B	?	6144x TPUv4 (38 days)
GPT-4	?	?	?	? (1T?)	?	?
PaLM 2	?	?	?	?	?	?

Generated Text from GPT-3

Bold text: Input prompt written by humans
Italics: Completion by GPT-3

Title: United Methodists Agree to Historic Split
Subtitle: Those who oppose gay marriage will form their own denomination
Article:

Brown et al, "Language Models are Few-Shot Learners", arXiv 2020

Generated Text from GPT-3

Bold text: Input prompt written by humans
Italics: Completion by GPT-3

Title: United Methodists Agree to Historic Split
Subtitle: Those who oppose gay marriage will form their own denomination
Article: *After two days of intense debate, the United Methodist Church has agreed to a historic split - one that is expected to end in the creation of a new denomination, one that will be "theologically and socially conservative," according to The Washington Post. The majority of delegates attending the church's annual General Conference in May voted to strengthen a ban on the ordination of LGBTQ clergy and to write new rules that will "discipline" clergy who officiate at same-sex weddings. But those who opposed these measures have a new plan: They say they will form a separate denomination by 2020, calling their church the Christian Methodist denomination. The Post notes that the denomination, which claims 12.5 million members, was in the early 20th century the "largest Protestant denomination in the U.S.," but that it has been shrinking in recent decades. The new split will be the second in the church's history. The first occurred in 1968, when roughly 10 percent of the denomination left to form the Evangelical United Brethren Church. The Post notes that the proposed split "comes at a critical time for the church, which has been losing members for years," which has been "pushed toward the brink of a schism over the role of LGBTQ people in the church." Gay marriage is not the only issue that has divided the church. In 2016, the denomination was split over ordination of transgender clergy, with the North Pacific regional conference voting to ban them from serving as clergy, and the South Pacific regional conference voting to allow them.*

GPT-3: Programming by prompt

Bold text: Input prompt written by humans
Italics: Completion by GPT-3

Poor English input: I eated the purple berries.
Good English output: I ate the purple berries.
Poor English input: Thank you for picking me as your designer. I'd appreciate it.
Good English output: Thank you for choosing me as your designer. I appreciate it.
Poor English input: The mentioned changes have done. or I did the alteration that you requested. or I changed things you wanted and did the modifications.
Good English output: The requested changes have been made. or I made the alteration that you requested. or I changed things you wanted and made the modifications.
Poor English input: I'd be more than happy to work with you in another project.
Good English output:

Brown et al, "Language Models are Few-Shot Learners", arXiv 2020

GPT-3: Programming by prompt

Input / Output examples

Test example

Italics: Completion by GPT-3

Poor English input: I eated the purple berries.

Good English output: I ate the purple berries.

Poor English input: Thank you for picking me as your designer. I’d appreciate it.

Good English output: Thank you for choosing me as your designer. I appreciate it.

Poor English input: The mentioned changes have done. or I did the alteration that you requested. or I changed things you wanted and did the modifications.

Good English output: The requested changes have been made. or I made the alteration that you requested. or I changed things you wanted and made the modifications.

Poor English input: I’d be more than happy to work with you in another project.

Good English output:

Brown et al, "Language Models are Few-Shot Learners", arXiv 2020

GPT-3: Programming by prompt

Input / Output examples

Test example

Italics: Completion by GPT-3

Poor English input: I eated the purple berries.

Good English output: I ate the purple berries.

Poor English input: Thank you for picking me as your designer. I’d appreciate it.

Good English output: Thank you for choosing me as your designer. I appreciate it.

Poor English input: The mentioned changes have done. or I did the alteration that you requested. or I changed things you wanted and did the modifications.

Good English output: The requested changes have been made. or I made the alteration that you requested. or I changed things you wanted and made the modifications.

Poor English input: I’d be more than happy to work with you in another project.

Good English output: *I'd be more than happy to work with you on another project.*

Brown et al, "Language Models are Few-Shot Learners", arXiv 2020

https://www.gwern.net/GPT-3#successes

GPT-3: Literary Style Transfer

Topic: humor: literary parodies and homages, pastiches, style parodies.
Parodies of the fantasy novel series *Harry Potter* in the style of various famous authors:
By Ernest Hemingway:
"It was a cold day on Privet Drive. A child cried. Harry felt nothing.

https://www.gwern.net/GPT-3#successes

GPT-3: Literary Style Transfer

Topic: humor: literary parodies and homages, pastiches, style parodies.
Parodies of the fantasy novel series *Harry Potter* in the style of various famous authors:
By Ernest Hemingway:
"It was a cold day on Privet Drive. A child cried. Harry felt nothing. He was dryer than dust. He had been silent too long. He had not felt love. He had scarcely felt hate. Yet the Dementor's Kiss killed nothing. Death didn't leave him less dead than he had been a second before. It wasn't about living or dying really. It wasn't about death. It was about the Dark Lord, his creator, his engenderer. He was not resurrected. He was created again. He was whole again in the courtyard of Malfoy Manor."

https://www.gwern.net/GPT-3#successes

GPT-3: Literary Style Transfer

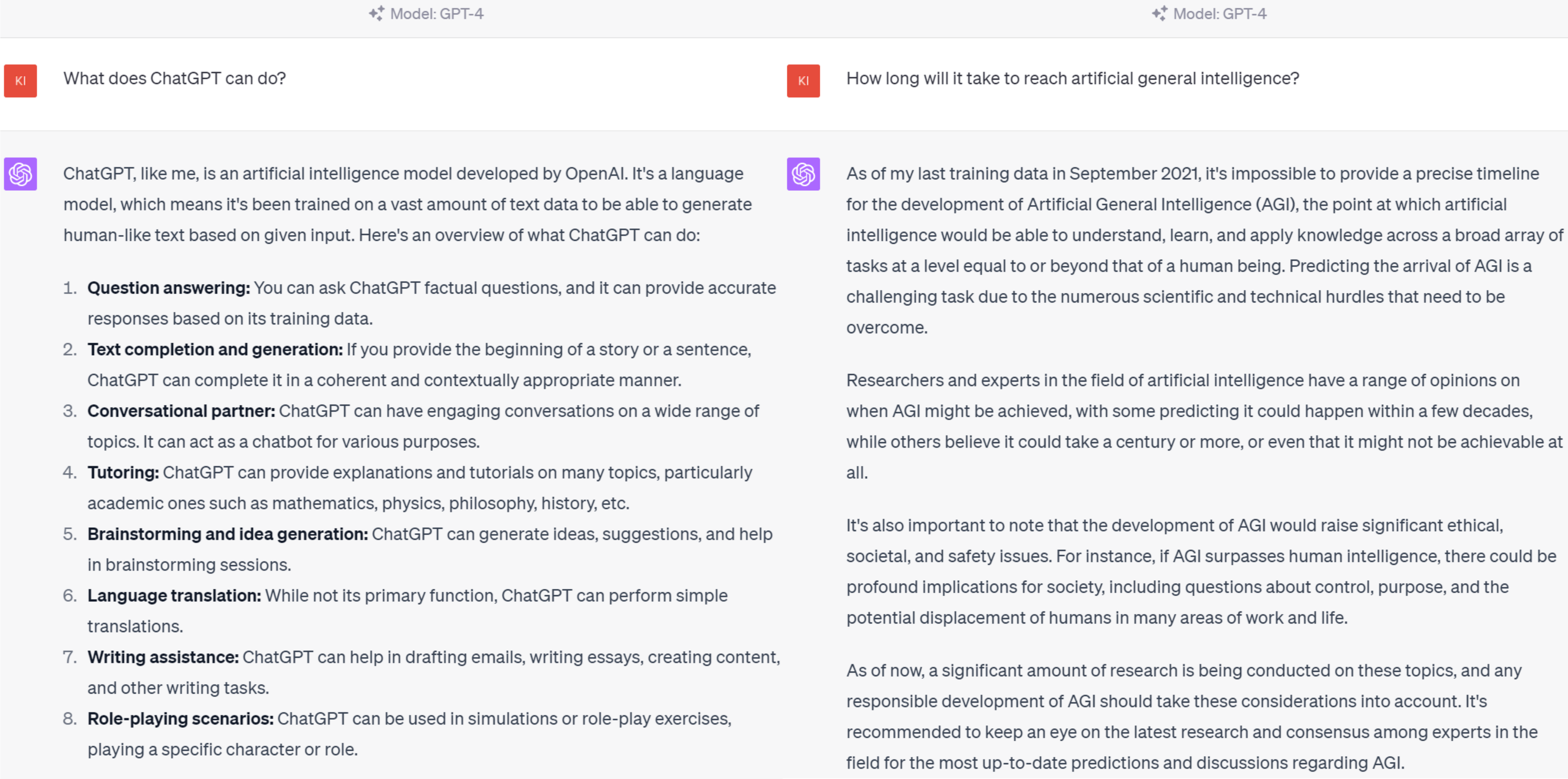
Topic: humor: literary parodies and homages, pastiches, style parodies.
Parodies of the fantasy novel series *Harry Potter* in the style of various famous authors:
By Ernest Hemingway:
"It was a cold day on Privet Drive. A child cried. Harry felt nothing. He was dryer than dust. He had been silent too long. He had not felt love. He had scarcely felt hate. Yet the Dementor's Kiss killed nothing. Death didn't leave him less dead than he had been a second before. It wasn't about living or dying really. It wasn't about death. It was about the Dark Lord, his creator, his engenderer. He was not resurrected. He was created again. He was whole again in the courtyard of Malfoy Manor."
By Jane Austen:
"It is a truth universally acknowledged, that a broken Harry is in want of a book. This he knows to his cost; pressing that wretched nose against the window of a bookstore in Diagon Alley the last thing before he goes into hiding for several years, where upon, he goes straight to Azkaban."

https://www.gwern.net/GPT-3#successes

GPT-3: Literary Style Transfer

Topic: humor: literary parodies and homages, pastiches, style parodies.
Parodies of the fantasy novel series *Harry Potter* in the style of various famous authors:
By Ernest Hemingway:
"It was a cold day on Privet Drive. A child cried. Harry felt nothing. He was dryer than dust. He had been silent too long. He had not felt love. He had scarcely felt hate. Yet the Dementor's Kiss killed nothing. Death didn't leave him less dead than he had been a second before. It wasn't about living or dying really. It wasn't about death. It was about the Dark Lord, his creator, his engenderer. He was not resurrected. He was created again. He was whole again in the courtyard of Malfoy Manor."
By Jane Austen:
"It is a truth universally acknowledged, that a broken Harry is in want of a book. This he knows to his cost; pressing that wretched nose against the window of a bookstore in Diagon Alley the last thing before he goes into hiding for several years, where upon, he goes straight to Azkaban."
By Arthur Conan Doyle:
"Harry pushed at the swinging doors of the bookshop hard, and nearly knocked himself unconscious. He staggered in with his ungainly package, his cheeks scarlet with cold and the shame of having chosen the wrong month to go Christmas shopping. The proprietor of the store, however, didn't cast even a cursory glance at him, being far more interested in an enormous hunk of wood lying in the middle of the floor, which certainly looked like a gallows. Yes, the proprietor said to a reedy old man wearing a bowler hat and a forlorn expression that made Harry want to kick him, I can rent you such a gallows for a small fee."

ChatGPT: Artificial Intelligence Chatbot



What does ChatGPT can do?

ChatGPT, like me, is an artificial intelligence model developed by OpenAI. It's a language model, which means it's been trained on a vast amount of text data to be able to generate human-like text based on given input. Here's an overview of what ChatGPT can do:

1. **Question answering:** You can ask ChatGPT factual questions, and it can provide accurate responses based on its training data.
2. **Text completion and generation:** If you provide the beginning of a story or a sentence, ChatGPT can complete it in a coherent and contextually appropriate manner.
3. **Conversational partner:** ChatGPT can have engaging conversations on a wide range of topics. It can act as a chatbot for various purposes.
4. **Tutoring:** ChatGPT can provide explanations and tutorials on many topics, particularly academic ones such as mathematics, physics, philosophy, history, etc.
5. **Brainstorming and idea generation:** ChatGPT can generate ideas, suggestions, and help in brainstorming sessions.
6. **Language translation:** While not its primary function, ChatGPT can perform simple translations.
7. **Writing assistance:** ChatGPT can help in drafting emails, writing essays, creating content, and other writing tasks.
8. **Role-playing scenarios:** ChatGPT can be used in simulations or role-play exercises, playing a specific character or role.

How long will it take to reach artificial general intelligence?

As of my last training data in September 2021, it's impossible to provide a precise timeline for the development of Artificial General Intelligence (AGI), the point at which artificial intelligence would be able to understand, learn, and apply knowledge across a broad array of tasks at a level equal to or beyond that of a human being. Predicting the arrival of AGI is a challenging task due to the numerous scientific and technical hurdles that need to be overcome.

Researchers and experts in the field of artificial intelligence have a range of opinions on when AGI might be achieved, with some predicting it could happen within a few decades, while others believe it could take a century or more, or even that it might not be achievable at all.

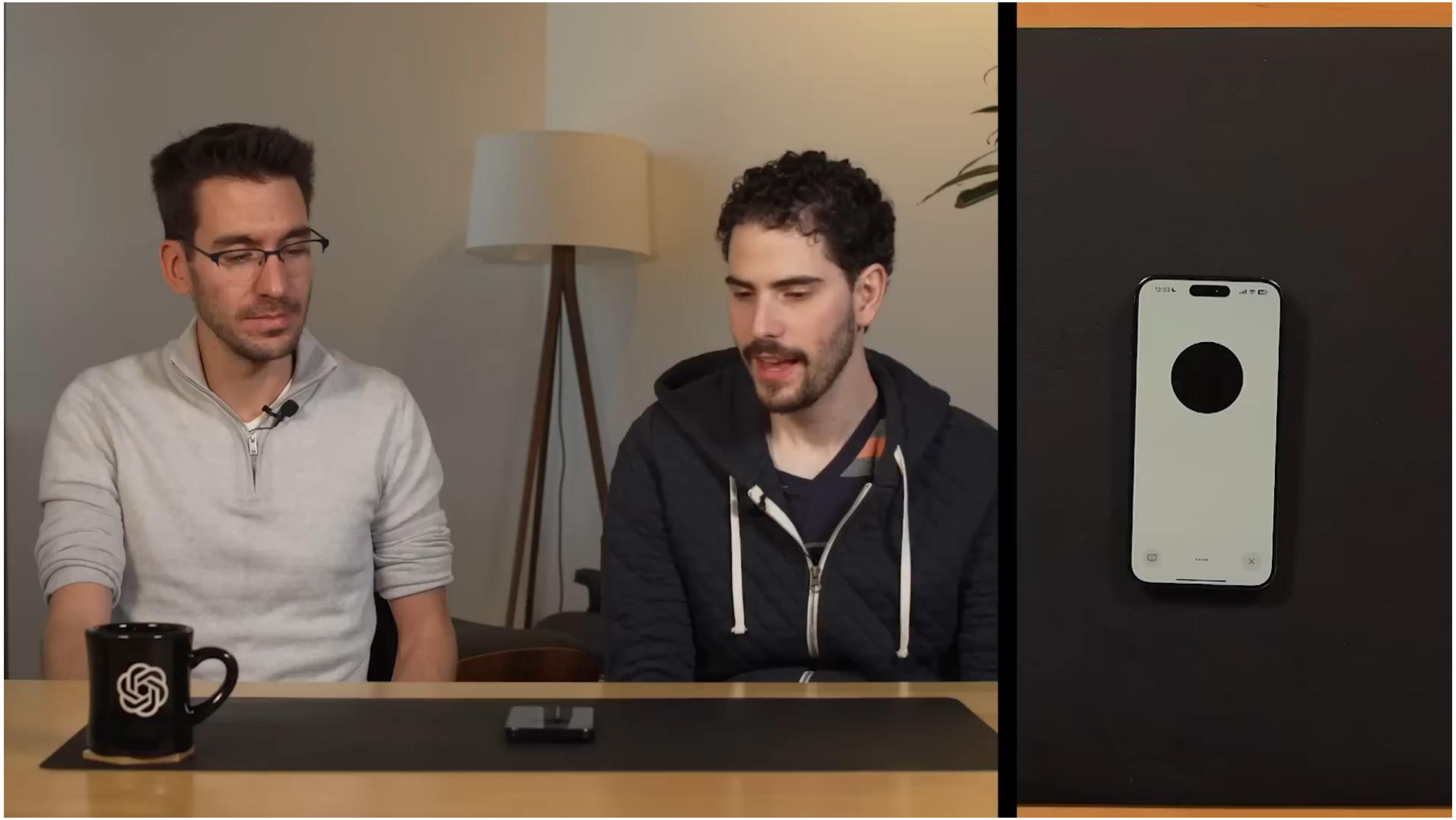
It's also important to note that the development of AGI would raise significant ethical, societal, and safety issues. For instance, if AGI surpasses human intelligence, there could be profound implications for society, including questions about control, purpose, and the potential displacement of humans in many areas of work and life.

As of now, a significant amount of research is being conducted on these topics, and any responsible development of AGI should take these considerations into account. It's recommended to keep an eye on the latest research and consensus among experts in the field for the most up-to-date predictions and discussions regarding AGI.

GPT-4o: Reasoning across audio, vision, and text

https://www.youtube.com/watch?v=vgYi3Wr7v_g

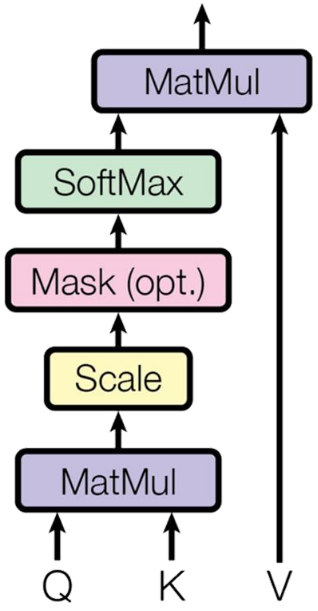
GPT-4o: Realtime translation



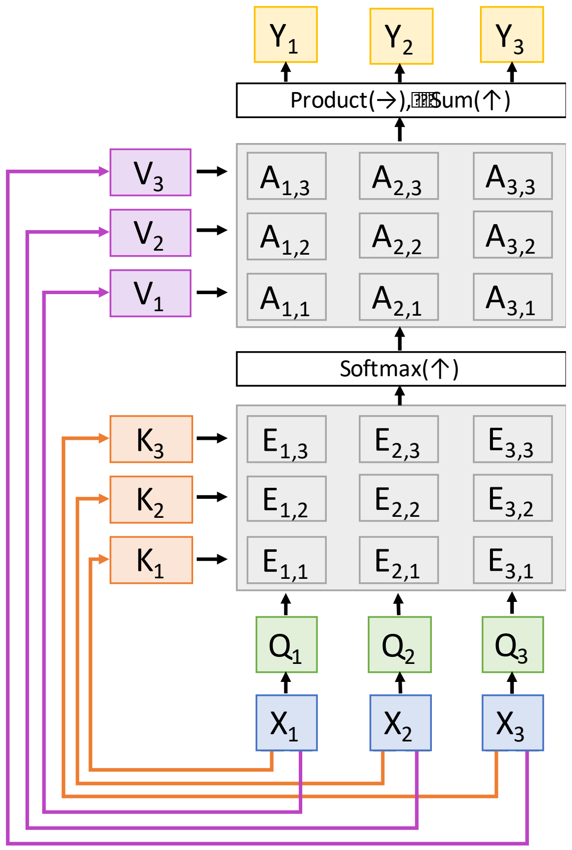
https://www.youtube.com/watch?v=WzUnEfiIqP4

Summary

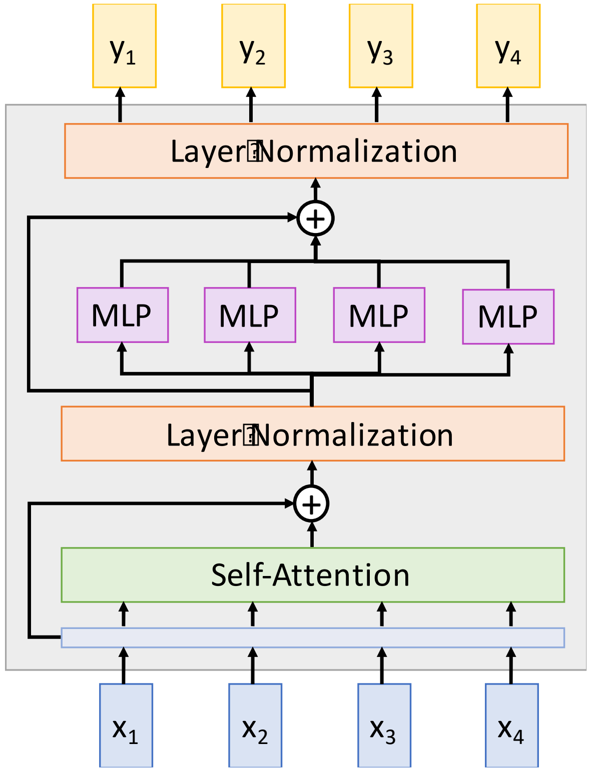
Attention mechanism: mapping a query and a set of key-value pairs to an output (= attention map)



Generalized **Self-Attention** is new, powerful neural network primitive



Transformers are a new neural network model that only uses attention



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

Next: Reinforcement Learning