C. Attention and Transformers

C.1 Origins of attention

C.2 Attention as a standalone operator

C.3 Transformer architectures

C.4 Transformers across different data structures

C.5 Training transformers

We now specialize the general attention operator into two key forms, Multi-head attention and Self-attention. Then we will see that stacking these with *feed-forward layers*, *residual connections*, and *normalization* gives the *Transformer* architecture.

Tiam Heydari, PhD
Postdoctoral Fellow, UBC

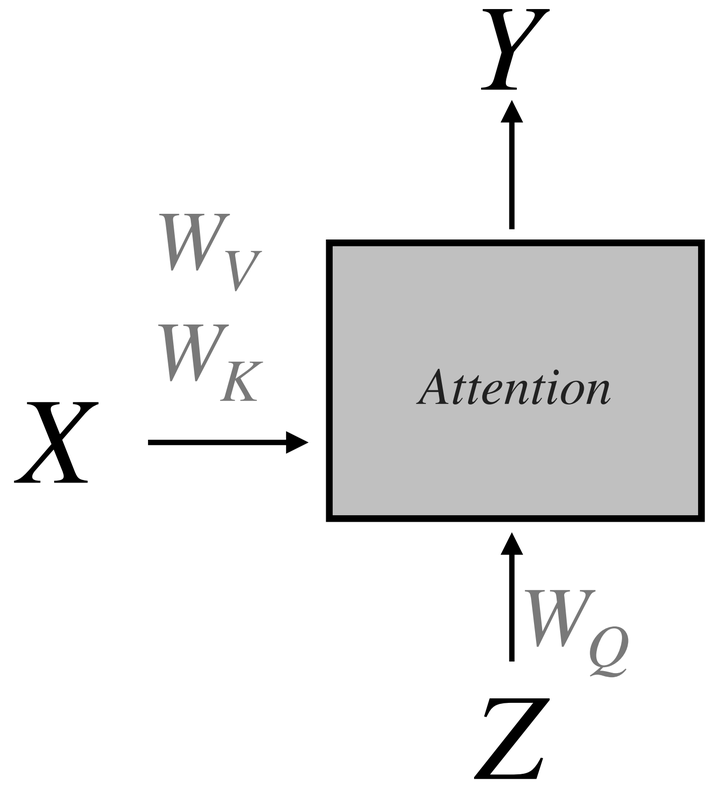
1) Multi-head attention

In part II of this lecture, we have seen that attention operator allows each element of Z to softly reads from all elements of X and update itself. And result is called Y

$$X = \begin{bmatrix} x_1^\top \\ \vdots \\ x_T^\top \end{bmatrix} \quad Z = \begin{bmatrix} z_1^\top \\ \vdots \\ z_{T'}^\top \end{bmatrix}$$

$$Q = ZW_Q \qquad W_Q \in \mathbb{R}^{d_z \times d_k}$$
$$K = XW_K \qquad W_K \in \mathbb{R}^{d_x \times d_k}$$
$$V = XW_V \qquad W_V \in \mathbb{R}^{d_x \times d_v}$$

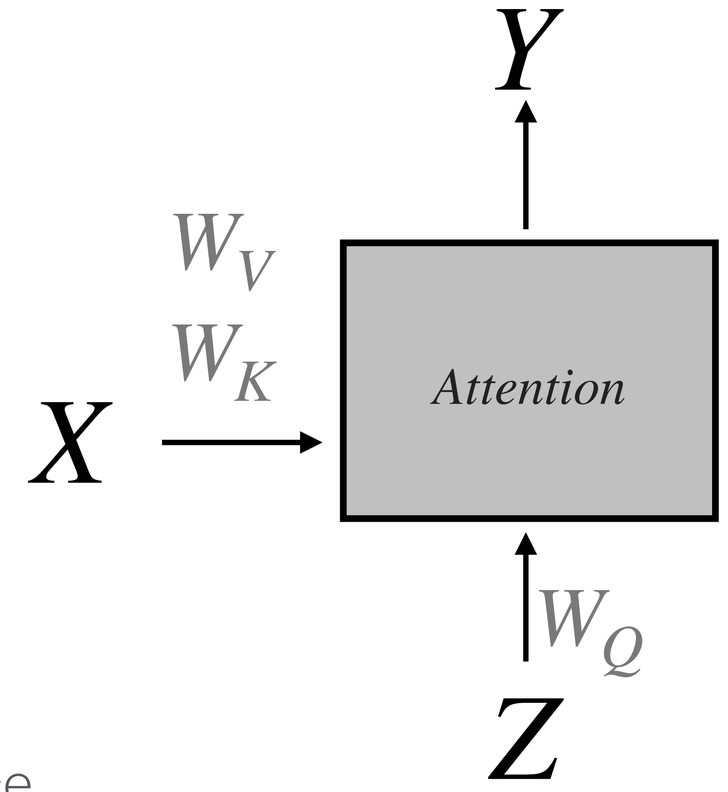
$$Y = \text{Attention}(Q, K, V) = \text{softmax}_{rowwise}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$



Motivation for multi-head Attention

- one **attention block** produces one specific mixing pattern over the values to produce output Y

- this is useful, but maybe too restrictive and not very stable



In other words:

- A single attention head gives one similarity function and one weighted mixture

- But the same input may contain different kinds of relations at once

- So instead of one attention, we want to run several (k) attentions *in parallel*

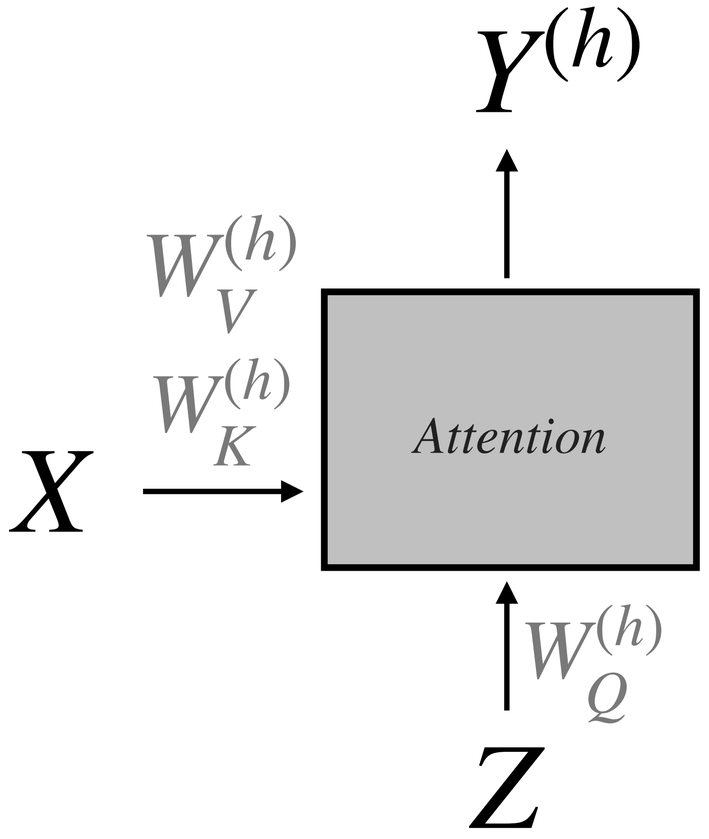
- Let each head learn its own W_Q, W_K, W_V

Defining one attention head, h:

$$X = \begin{bmatrix} x_1^\top \\ \vdots \\ x_T^\top \end{bmatrix} \quad Z = \begin{bmatrix} z_1^\top \\ \vdots \\ z_{T'}^\top \end{bmatrix}$$

$$Q^{(h)} = ZW_Q^{(h)},$$
$$K^{(h)} = XW_K^{(h)}$$
$$V^{(h)} = XW_V^{(h)}$$

$$Y^{(h)} = \text{Attention}\left(Q^{(h)}, K^{(h)}, V^{(h)}\right)$$



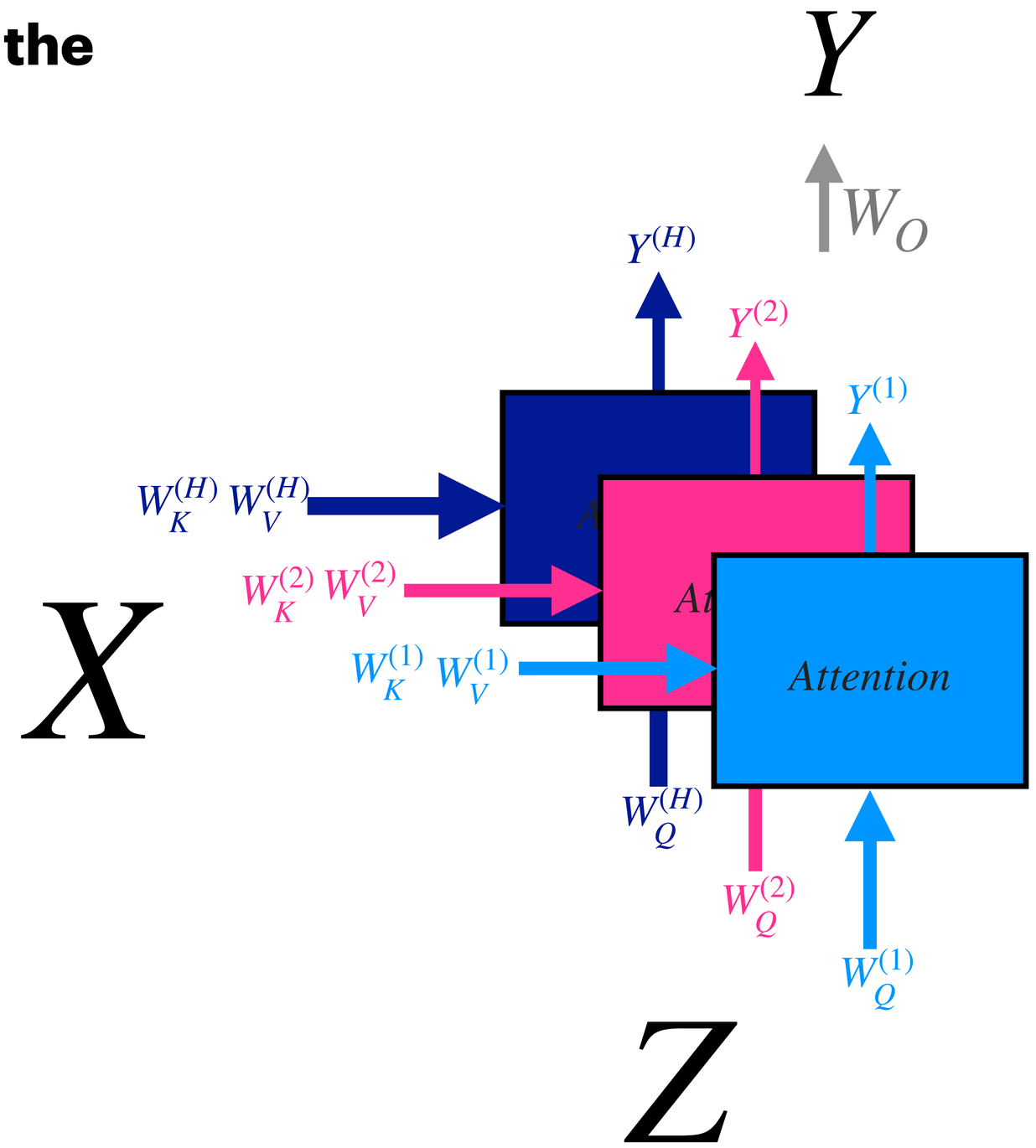
Now we can have multiple heads at the same time

$$X = \begin{bmatrix} x_1^\top \\ \vdots \\ x_T^\top \end{bmatrix} \qquad Z = \begin{bmatrix} z_1^\top \\ \vdots \\ z_{T'}^\top \end{bmatrix}$$

$$Q^{(h)} = ZW_Q^{(h)},$$
$$K^{(h)} = XW_K^{(h)}$$
$$V^{(h)} = XW_V^{(h)}$$

$$Y^{(h)} = \text{Attention}\left(Q^{(h)}, K^{(h)}, V^{(h)}\right)$$

$$Y = \text{Concat}\left(Y^{(1)}, Y^{(2)}, \ldots, Y^{(H)}\right) W_O$$

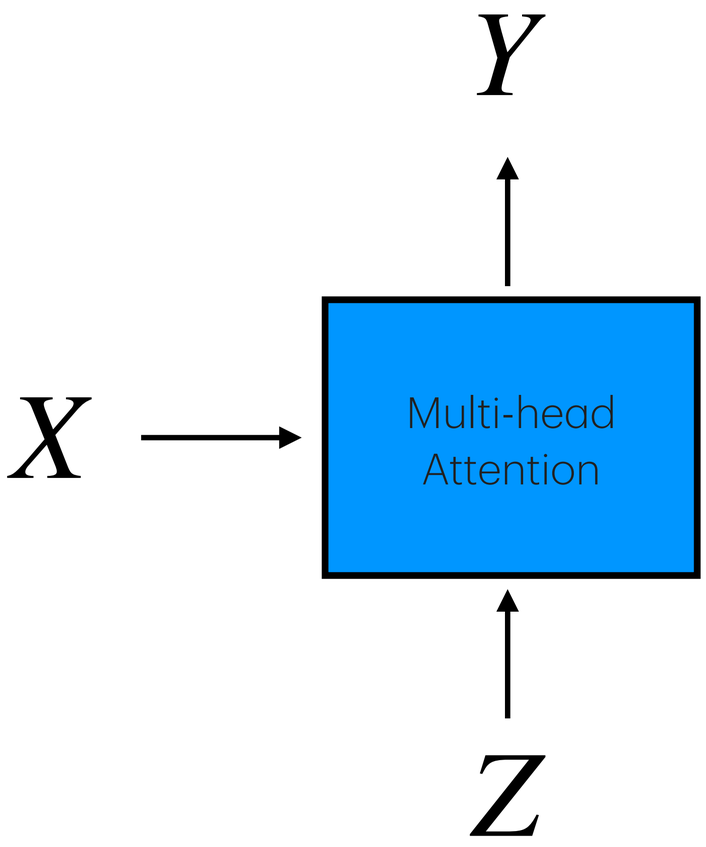


$$X = \begin{bmatrix} x_1^\top \\ \vdots \\ x_T^\top \end{bmatrix} \quad Z = \begin{bmatrix} z_1^\top \\ \vdots \\ z_{T'}^\top \end{bmatrix}$$

$$Q^{(h)} = ZW_Q^{(h)},$$
$$K^{(h)} = XW_K^{(h)}$$
$$V^{(h)} = XW_V^{(h)}$$

$$Y^{(h)} = \text{Attention}\left(Q^{(h)}, K^{(h)}, V^{(h)}\right)$$

$$Y = \text{Concat}\left(Y^{(1)}, Y^{(2)}, \ldots, Y^{(H)}\right) W_O$$



Multi-head attention simply replaces one attention operator with several attention operators in parallel, each with its own softmax weighting pattern.

2) Self-attention

from general attention to self-attention

- In general attention, Queries come from Z, while Keys/Values come from X

- In self-attention, these come from the same set of representations So we set:

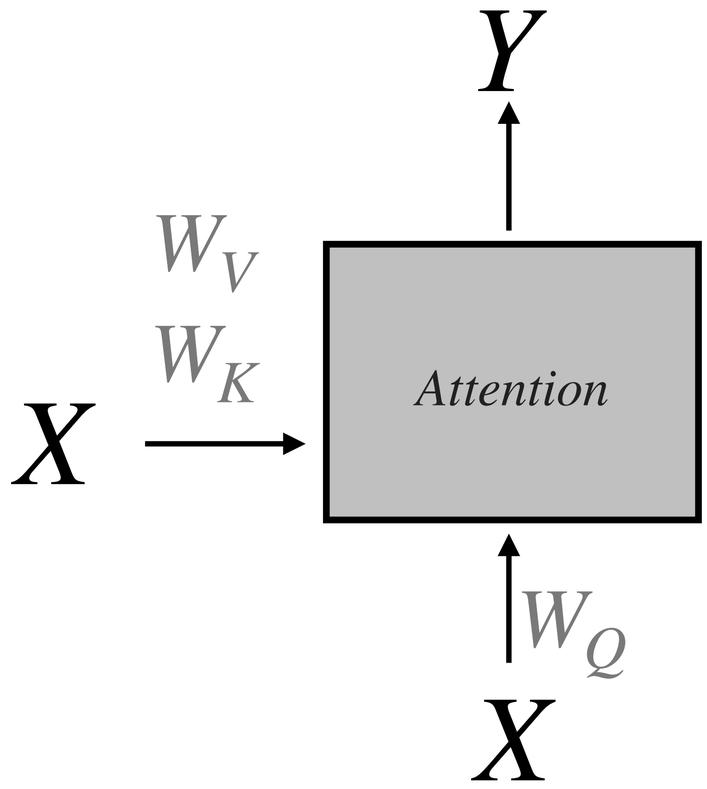
$$Z = X$$

- Self-attention in matrix form:

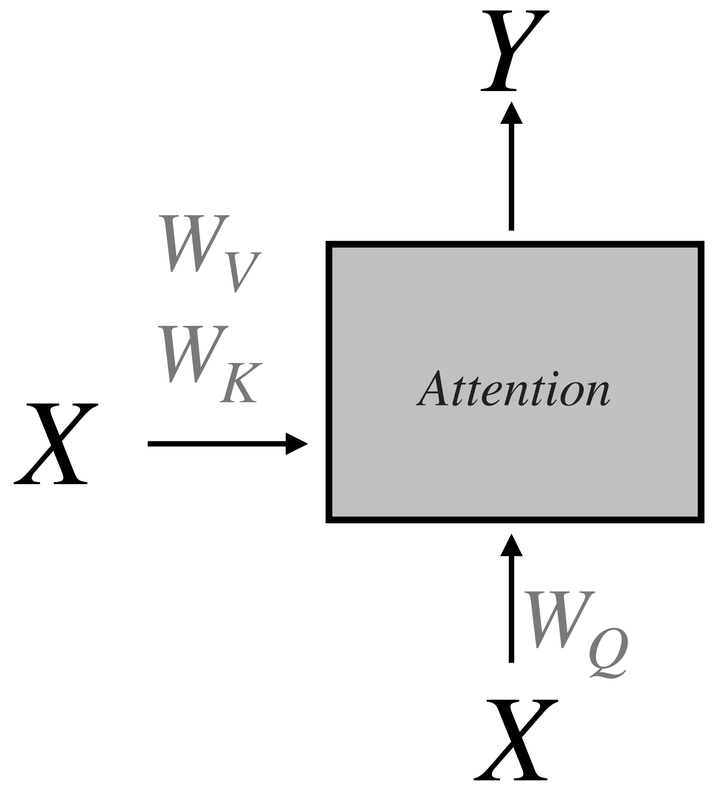
$$X = \begin{bmatrix} x_1^\top \\ \vdots \\ x_T^\top \end{bmatrix}$$

$$Q = XW_Q \qquad W_Q \in \mathbb{R}^{d_z \times d_k}$$
$$K = XW_K \qquad W_K \in \mathbb{R}^{d_x \times d_k}$$
$$V = XW_V \qquad W_V \in \mathbb{R}^{d_x \times d_v}$$

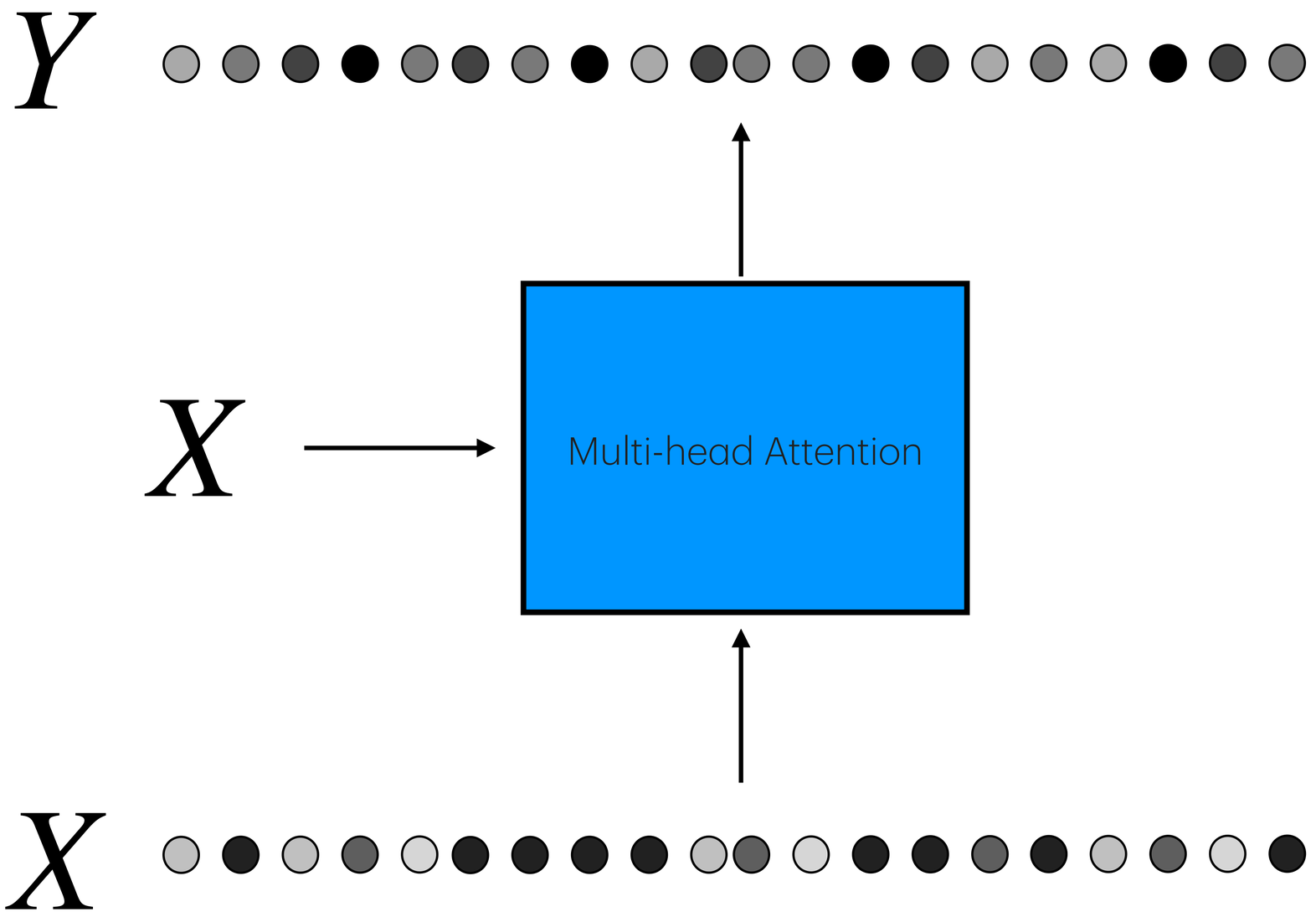
$$Y = \text{Attention}(Q, K, V) = \text{softmax}_{rowwise}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$



- What does self-attention do?
- Each element of X becomes a Query, a Key, and a Value
- So each element can read from all other elements in the same set
- This updates every element using context from the whole set

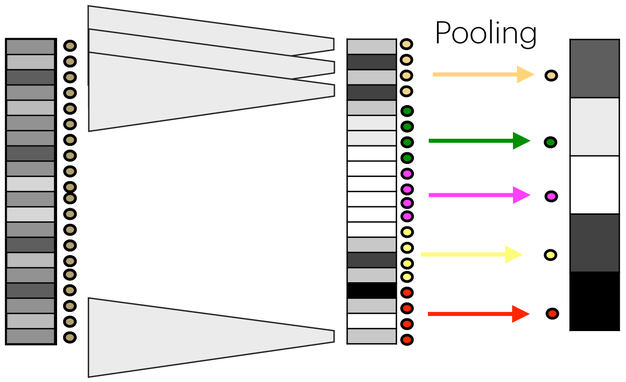


Self-attention lets each element of X update itself by softly reading from all elements, including itself.



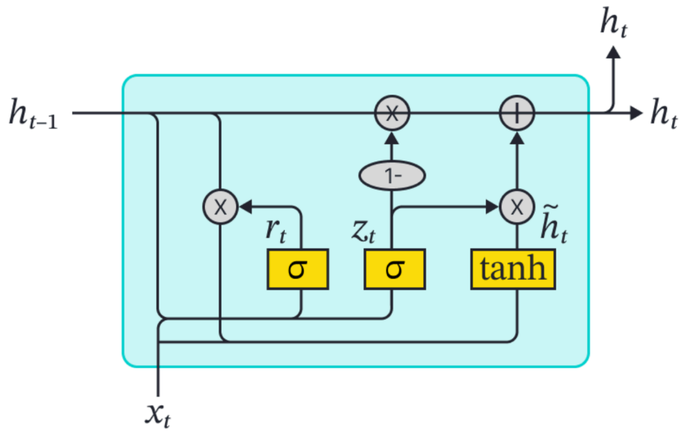
Let's put attention in a more general context, and compare it with some other operators:

Convolutional Block



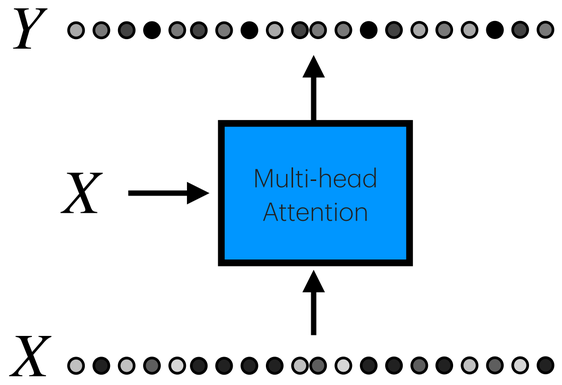
- **Best natural domain**: grids and locally ordered domains
- **Built-in bias**: locality + weight sharing
- **Symmetry view**: translation equivariant
- **Strength:** efficient, parallel computing, strong local inductive bias
- **Weakness:** long-range interactions need many layers / large receptive field

Recurrent Networks



- **Best natural domain**: ordered sequences
- **Built-in bias**: sequential update, directionality / causality
- **Symmetry view**: order matters explicitly
- **Strength**: natural for autoregressive sequence processing
- **Weakness**: hard to parallelize across time, information buttleneck

Self-Attention



- **Best natural domain**: sets of vectors; also sequences or grids once positional structure is added.
- **Built-in bias:** content-based interaction between any pair
- **Symmetry view**: permutation equivariant (without positional encoding)
- **Strength:** direct long-range interactions, **high parallelism**
  Weakness: dense attention is expensive for long inputs, since it builds the full pairwise score matrix

3) Transformer Architecture

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Transformer is an architecture that puts Self-Attention as the core! Very simple

Output

layer normalization normalizes each vector element for stability

Residual connection

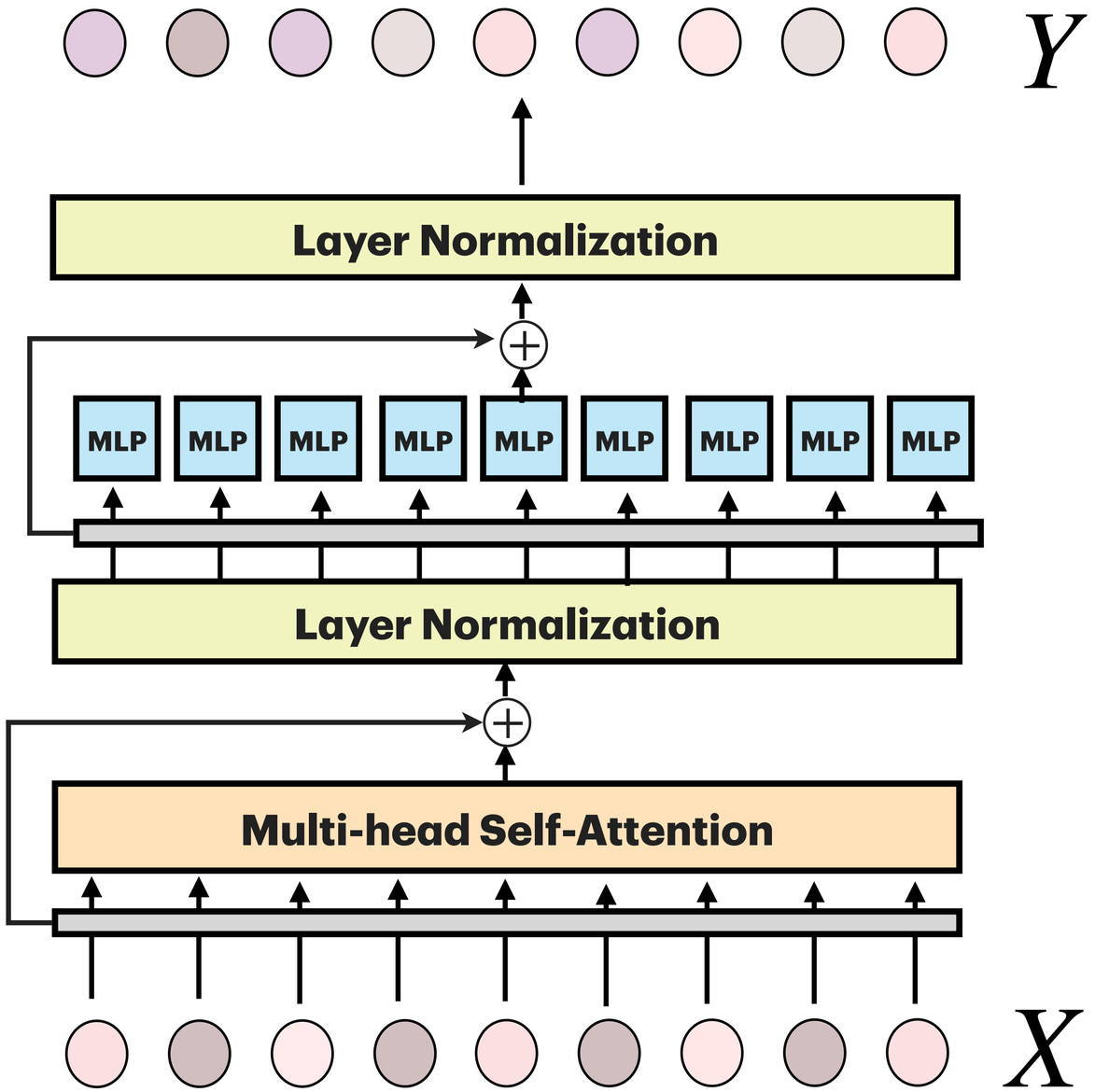
MLP independently on each vector element

layer normalization normalizes each vector element for stability

Residual connection

All elements (vectors) interact through multi-head self-attention

Input

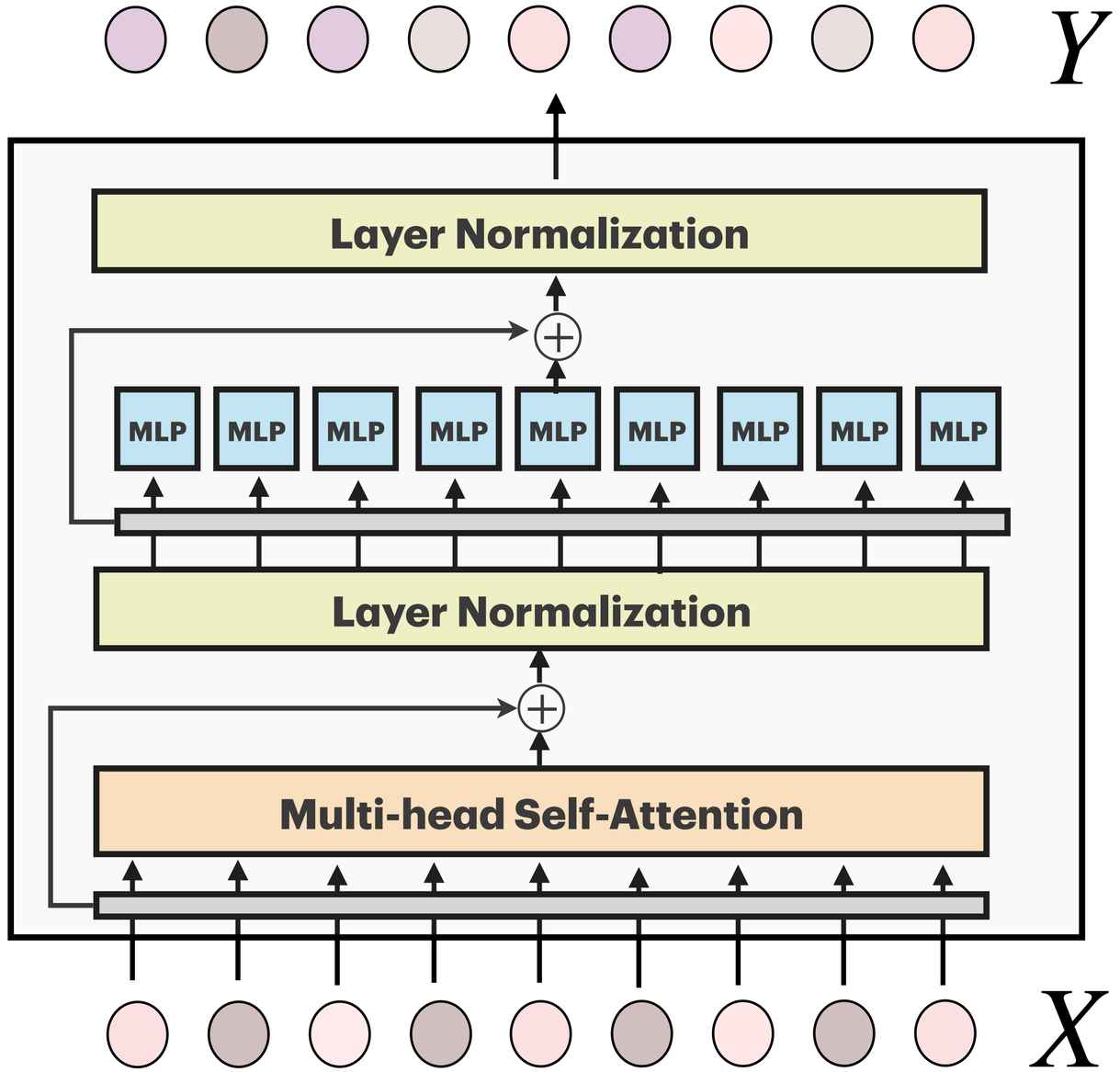


A Transformer Block:

Input: set/sequence X

Output: updated set/sequence Y

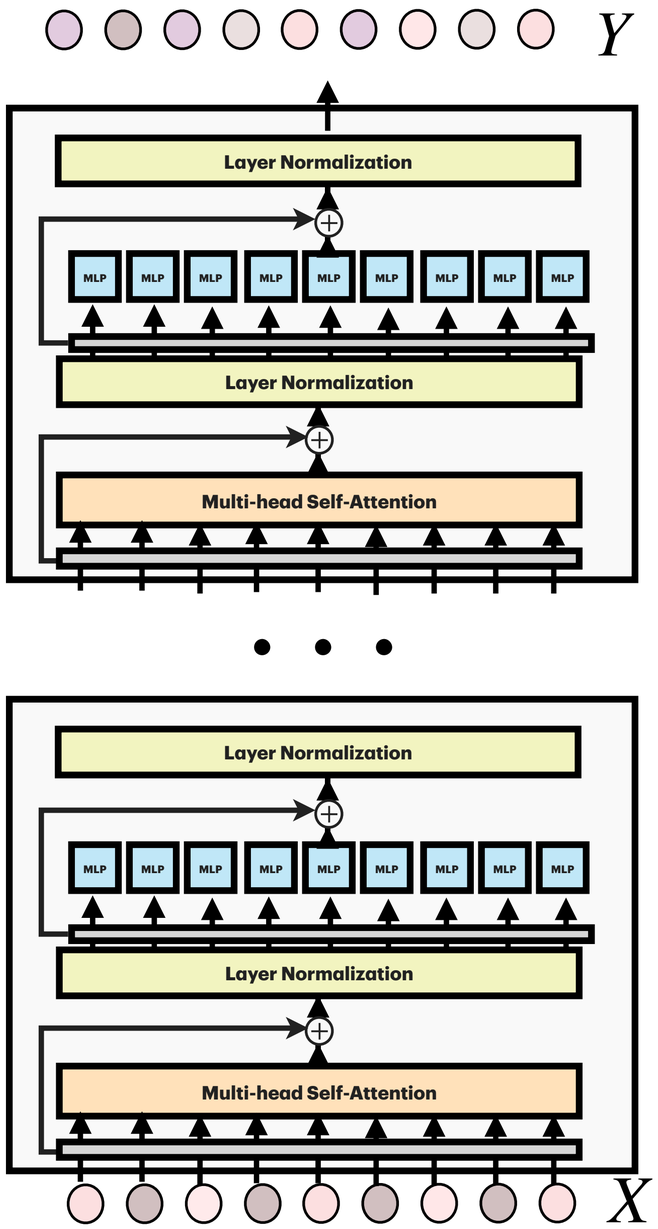
- Multi-head self-attention: the only step in the block where different tokens directly interact with each other
- LayerNorm and MLP: operate on each token independently, with shared weights across positions
- Residual connections: preserve information and help optimization
- Highly parallelizable: most of the computation is large matrix multiplications!



A Transformer is a sequence of Transformer blocks:

Input: set/sequence X

Output: updated set/sequence Y



- Transformers have not changed **much**** since 2017
- They have just become much larger!:

- Original Transformer (Vaswani et al., 2017) 12 blocks, (D=1024), (H=16), (N=512) 213M parameters

- GPT-2 (Radford et al., 2019) 48 blocks, (D=1600), (H=25), (N=1024) 1.5B parameters

- GPT-3 (Brown et al., 2020) 96 blocks, (D=12288), (H=96), (N=2048) 175B parameters

**

- LayerNorm: **Ba et al., 2016** normalize each vector using its own features.
- Original Transformer: **Vaswani et al., 2017** introduced the standard Transformer block.
- Pre-Norm: **Baevski & Auli, 2018** early use of LayerNorm before attention/ MLP.
- Why Pre-Norm: **Xiong et al., 2020** explained its training stability advantages.
- RMSNorm: **Zhang & Sennrich, 2019** simpler norm based on root-mean-square only.

Attention Is All You Need