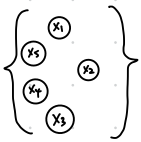
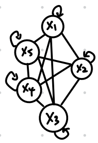
We can think about a set in 2 different ways.

(1)



Graph without any edges

(2)



Message Passing

: Allow every node talk to every other node

: This is the core Principle behind transformer networks

Graph that is fully connected

Quick Recap

: Within linear algebra, each permutation defines a $|V| \times |V|$ matrix.

· Such matrices are called permutation matrices

· They have exactly one 1 in every row and column, zeroes elsewhere

Ex)

$$P_{(2,4,1,3)}X = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} - X_1 - \\ - X_2 - \\ - X_3 - \\ - X_4 - \end{bmatrix} = \begin{bmatrix} - X_2 - \\ - X_4 - \\ - X_1 - \\ - X_3 - \end{bmatrix}$$

Learning on Sets

① Setup

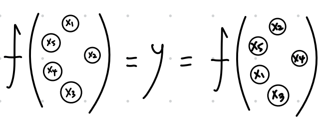
For now, assume our graph has no edges (i.e., $\Omega = V$, the set of nodes)

Let $X_i \in \mathbb{R}^k$ be the features of node i. (i.e., our feature space is $C = \mathbb{R}^k$)

We can stack these features into a node feature matrix of shape $|V| \times k$. $X = (X_1, \ldots, X_N)^T$

· i^{th} row of X corresponds to X_i

* Permutation invariant operator is an operator (F), that if we apply F to our set or a permuted version of a set, we're getting the same output.



Symmetry group G: n-element Permutation group Σ_n

Group element $g \in G$: Permutation

Permutation invariance

Want : function $f(X)$ over sets that will not depend on the order

Equivalently: Applying a permutation matrix shouldn't modify result

$f(X)$ is permutation invariant if, for all permutation matrices P: $f(PX) = f(X)$

Permutation Equivariance:

· Permutation invariant models are good for set-level outputs. If we would like to answer at the node level, we need Permutation equivariant models.

· Permutation equivariant functions: $F(PX) = PF(X)$

↳ It doesn't matter if we do permute before F or later F.

Deep Sets

: $f(X) = \phi\left(\bigoplus_{i \in V} \psi(x_i)\right)$

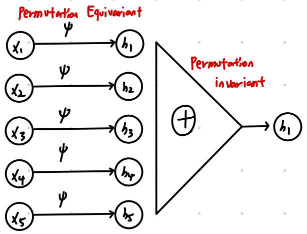
where ϕ, ψ are (learnable) functions, e.g. MLPs

$\bigoplus$ denotes any permutation-invariant operator.

General blueprint for learning on sets

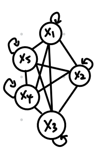
: One way we can enforce locality in equivariant set functions is through a shared function ψ applied to every node in isolation: $h_i = \psi(x_i)$, and stack h_i into a matrix $H = F(X)$

Pictorial View



Note: ψ : a shared function to every node in isolation
And, there's no message passing

Now, (2) · Sets as fully connected graphs



Message Passing: $f(x_i) = \phi\left(x_i, \square_{j \in N_i} \psi(x_i, x_j)\right)$

$\psi(x_i, x_j)$: Message Passing between x_i and x_j

$\square_{j \in N_i}$: Message Aggregation

$\phi(x_i, z)$: Node feature update

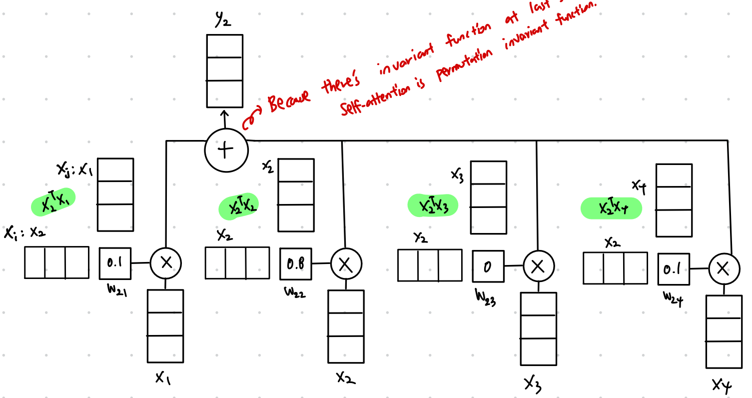
Basic Self-Attention

- Input: Sequence of tensors $x_1, x_2, \dots, x_t$
- Output: Sequence of tensors, each one a weighted sum of the input sequence : $y_1, y_2, \dots, y_t$

  $y_i = \sum_j w_{ij} x_j$
- W in self-attention is not a learned weight, but a function of x_i and x_j : $w'_{ij} = x_i^T x_j$
- W must sum to 1 over j

  : $w_{ij} = \frac{\exp w'_{ij}}{\sum_j \exp w'_{ij}}$: Softmax (we're applying softmax to the nodes in same sequence)

Pictorial View of basic self attention



Because there's invariant function at last step, Self-attention is permutation invariant function.

: $x_1 \sim x_4$ are in same sequence (block)

$$y_2 = \sum_j W_{2j} x_j = W_{21}x_1 + W_{22}x_2 + W_{23}x_3 + W_{24}x_4$$

$$= \frac{\exp(x_2^T x_1)}{\sum_j \exp(x_2^T x_j)} x_1 + \frac{\exp(x_2^T x_2)}{\sum_j \exp(x_2^T x_j)} x_2 + \frac{\exp(x_2^T x_3)}{\sum_j \exp(x_2^T x_j)} x_3 + \frac{\exp(x_2^T x_4)}{\sum_j \exp(x_2^T x_j)} x_4 = 0.1x_1 + 0.8x_2 + 0 + 0.1x_4$$

☆ $x_i^T x_j$: Notion of similarity between x_i and x_j

Therefore, W_{2j} tells how important x_j is to x_2. And y_2 is computed based on its relationship with other nodes.

this is "Attention"!

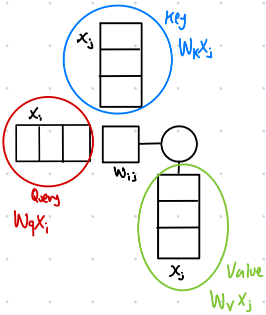
Basic Self Attention

- No learned weights
- Order of sequence doesn't affect result of computations

Query, Key, Value

: Every input vector x_i is used in 3 ways:

- Query: Compared to every other vector to compute attention weights for its own output y_i
- Key: Compared to every other vector to compute attention weight w_{ij} for output y_j
- Value: Summed with other vectors to form the result of the attention weighted sum

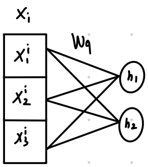


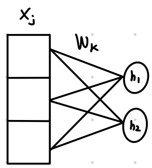
: We can process each input vector to fulfill the three roles with matrix multiplication

Learning the matrices → learning attention

$$q_i = W_q x_i \qquad w'_{ij} = q_i^T k_j$$
$$k_i = W_k x_i \qquad w_{ij} = \text{Softmax}(w'_{ij})$$
$$v_i = W_v x_i \qquad y_i = \sum_j w_{ij} v_j$$

W_q, W_K, W_v : Trainable matrices, such as MLP





And, $w'_{ij} = q_i^T k_j$

$w_{ij} = \text{Softmax}(w'_{ij})$: Scalar value.

$y_i = \sum_j w_{ij} v_j$

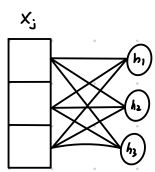
$$W_q : \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$$

$$q_i = W_q x_i = \underset{2\times 3}{\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}} \underset{3\times 1}{\begin{bmatrix} x_1^i \\ x_2^i \\ x_3^i \end{bmatrix}} = \underset{2\times 1}{\begin{bmatrix} h_1 \\ h_2 \end{bmatrix}_q}$$

$$W_K : \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix}$$

$$k_j = W_K x_j = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} \begin{bmatrix} x_1^j \\ x_2^j \\ x_3^j \end{bmatrix} = \underset{2\times 1}{\begin{bmatrix} h_1 \\ h_2 \end{bmatrix}_K}$$

★ W_q and W_K must share same size, but not W_v.



$$W_K : \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$

$$v_j = W_K x_j = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \begin{bmatrix} x_1^j \\ x_2^j \\ x_3^j \end{bmatrix} : 3\times 1$$

$y_i = \sum_j w_{ij} v_j$

$X : (B, T, C)$

self.key(C, h) → B, T, H

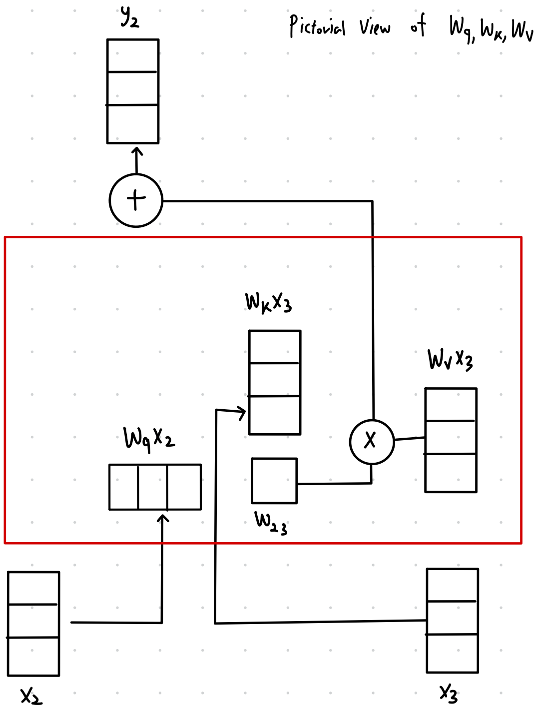
self.query(C, h) → B, T, H

$W = Q @ K^T$ $\left([[[\text{—}], [\text{—}], \vdots [\text{—}]] \right)^T$

B × T × H   B × H × T

: B × T × T

Pictorial View of W_q, W_K, W_V



Multi-head attention

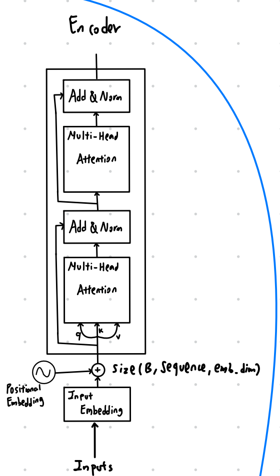
- Multiple "heads" of attention just means learning different sets of W_q, W_K, and W_V matrices simultaneously.
- Implemented as just a single matrix

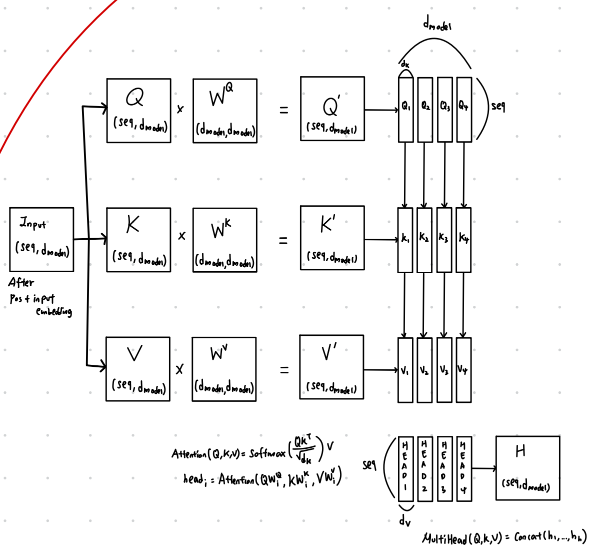
Code: `nn.ModuleList([Head(...) for _ in range(num_heads)])`

And when you return, you need concatenate them.

```
out = torch.cat([h(X) for h in self.heads], dim=-1)
```

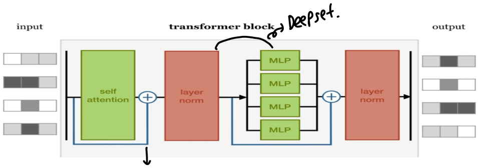
Pictorial View of MultiHead

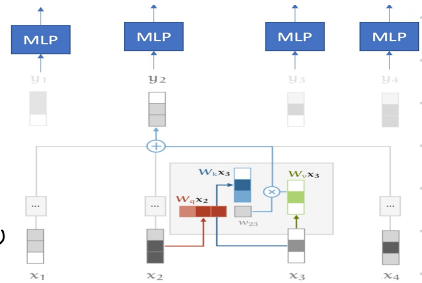




Transformer

- Self-attention layer → Layer normalization → Dense Layer





- $\begin{matrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{matrix} + \begin{matrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{matrix} = \begin{matrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{matrix} \rightarrow \text{layer norm}(z) \rightarrow \begin{matrix} \text{normalized } z_1 \\ \text{" } z_2 \\ \text{" } z_3 \\ \text{" } z_4 \end{matrix} + \begin{matrix} \text{MLP(normalized } z_1) \\ \vdots \\ \text{MLP(normalized } z_4) \end{matrix}$

However, there are problems that order of sequence affect the result of computation, such as NLP.

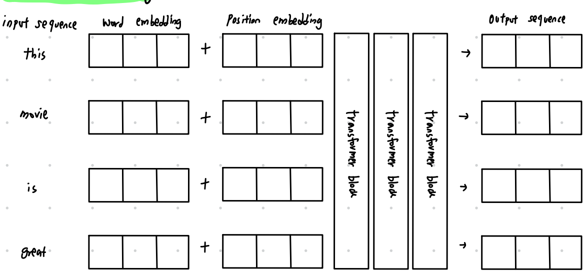
: Let's encode each vector with position

Text, Signals, and Images are not sets.

: There is an inherent ordering on most domains we deal with

Yoda is a Jedi master! ≠ a Jedi master Yoda is!

Position embedding



In "Attention is All you need" Paper,

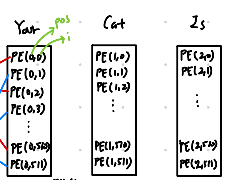
$$\vec{P}_t^{(i)} = f(t)^{(i)} := \begin{cases} \sin(w_k \cdot t) & \text{if } i = 2k \\ \cos(w_k \cdot t) & \text{if } i = 2k+1 \end{cases} \qquad \vec{P}_t = \begin{bmatrix} \sin(w_1 \cdot t) \\ \cos(w_1 \cdot t) \\ \vdots \\ \sin(w_{d/2} \cdot t) \\ \cos(w_{d/2} \cdot t) \end{bmatrix}_{d \times 1}$$

where d be the encoding dimension ($d \overset{\equiv}{_2} 0$) (embedding)

where $w_k = \dfrac{1}{10000^{2k/d}}$

Example)

Sentence 1:



$PE(Pos, 2i) = \sin\dfrac{Pos}{10000^{\frac{2i}{d}}}$

$PE(Pos, 2i+1) = \cos\dfrac{Pos}{10000^{\frac{2i}{d}}}$

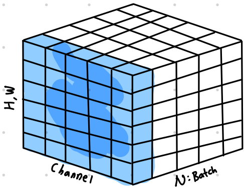
$d \in \mathbb{R}^{511 \times 1}$

$\vec{P}_1 \quad \vec{P}_2 \quad \vec{P}_3$

Why trigonometric functions?

: Trigonometric functions like sin and cos naturally represent a pattern that the model can recognize as continuous.

Layer Normalization



Layer Normalization

Layer Norm: Calculate μ, σ^2 of Channel per each batch.

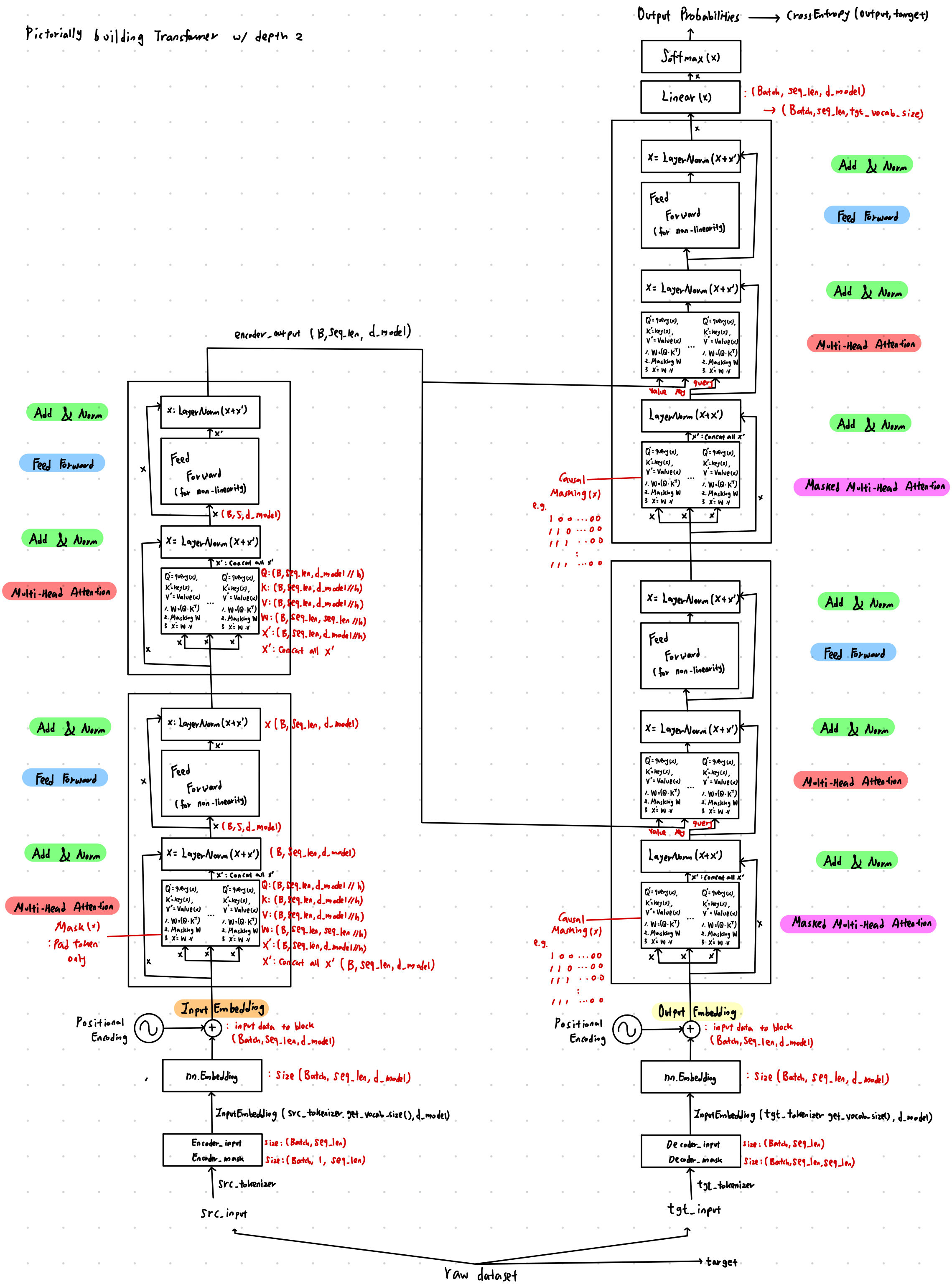
Example of layer norm:

Batches of 3 items:

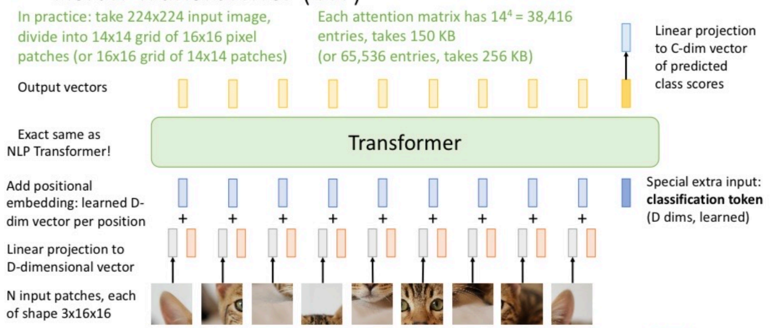
Item ①	Item ②	Item ③
50.147	1990.2	183.7
3314.8	688.3	174.878
...	...	...
...	...	...
8941.2	27.4	10923.7
1994.7	94.10	1004.88
μ_1	μ_2	μ^3
σ_1^2	σ_2^2	σ_3^2

$$\Rightarrow \quad \hat{x}_j = \frac{x_j - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

Pictorially building Transformer w/ depth 2



Vision Transformer (ViT)



Dosovitskiy et al, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, ICLR 2021