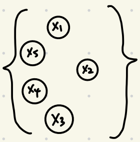
Set-Structured data

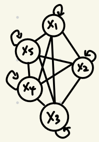
We can think about a set in 2 different ways.

(1) Set



Graph without any edges

(2)



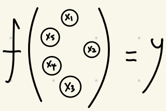
Graph that is fully connected

Message Passing
: Allow every node talk to every other node

: This is the core Principle behind transformer networks

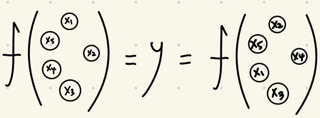
If you want to do message passing in a first scenario (1), where there's no connection between the nodes, you can immediately see there won't be any messages passing between the nodes.

Hence, we're essentially applying a nonlinear function on top of those elements of our sets. And it's precisely the idea of deepsets.



So, what we want for learning on sets is a permutation invariant or permutation equivarient operator.

* 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.

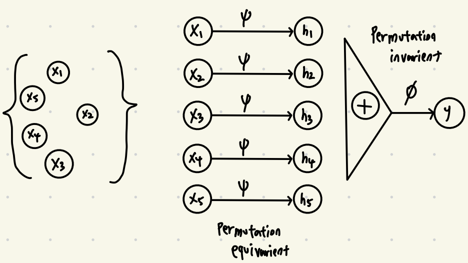


When do we want to get a permutation invariant function or neural network? And when would we need a permutation equivariant?

: Permutation invariant: If the problem is a set classification

Permutation equivarient: If the problem is a node classification or regression

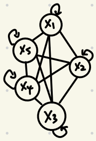
General Architecture of Deepsets



No message passing!

Attention & Set transformers

Sets as fully connected graphs : Allow every node talk to every other node.



Revisit "Message Passing"

$$f(x_i) = \phi\left(x_i, \boxed{}_{j \in N_i} \psi(x_i, x_j)\right)$$

Permutation invariant aggregation operator. e.g. Sum, average

new feature of node i

Learnable functions

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

$\boxed{}_{j \in N_i}$: Message Aggregation

$\phi(x_i, z)$: Node feature Update.

Since we transformed set to a fully connected graph, N_i is all nodes.

Attention : Specific mechanism to allow us to do message passing.

Basic self-attention

- Input: Sequence of tensors : $x_1, x_2, \dots, x_t$ and $x_i \in \mathbb{R}^d$
- Output: Sequence of tensors, each one a weighted sum of the input sequence.

  $y_1, y_2, \dots, y_t$ and $y_i = \sum_j w_{ij} x_j$

So, self-attention is an operation that goes and updates the features, x_i, by a simple linear combination of all other neighbors

The way that weights (w_{ij}) are calculated has a very specific structure.

Structure : W_{ij} in self attention are not learned weights, but a function of x_i and x_j $\rightarrow$ $w'_{ij} = \underbrace{x_i^T x_j}_{\text{linear kernel.}}$, hence w'_{ij} tells how similar between x_i and x_j

$$W_{ij} = \frac{\exp w'_{ij}}{\sum_j \exp w'_{ij}}$$ softmax

: must sum to 1 over j, which means convex combination.

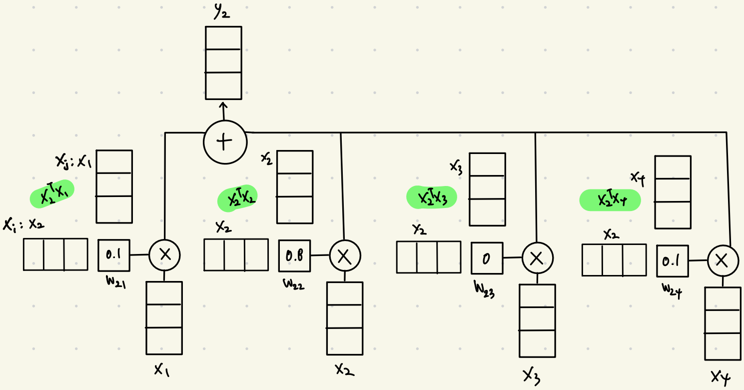
$\hookrightarrow$ linear combination that weights sum up to 1.

Therefore, essentially it's giving me a probability for each node.

And, this probability is the notion of attention.

$W_{ij} x_j$: Node i is going to pay attention to node j by W_{ij}

Pictorial depiction of Self attention



① Calculate w'

② Calculate softmax of w'

Is this a permutation invariant or equivariant function?

: Self-attention is a Permutation equivariant operation.

b.t.w, what's difference between equivariant and invariant?

: Equivariant is when I apply my permutation into my input, the output is also be permuted

ex) In CNN, if we translate the input, the output of the convolution is going to be also translated.

: Invariant : If I translate the inputs, the output is not going to be changed.

ex) When we want to classify a cat, and if you translate the cat, the output class shouldn't change.

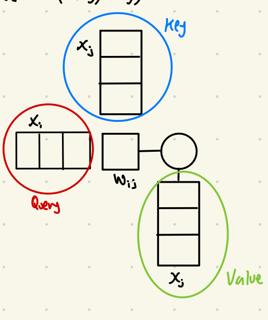
However, if we want to segment a cat, and if you translate the cat, your segmentation must translate as well.

Basic self attention

: No learned weights → Let's transform to make W learnable

Order of the sequence doesn't affect result of computations

We need Query, Key, and Value Concept.



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

· Learning matrices → Learning attention

Learnable matrices

$q_i = W_q x_i \qquad k_i = W_k x_i \qquad v_i = W_v x_i$

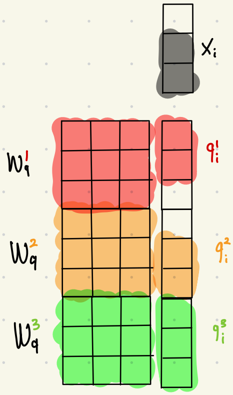
$w'_{ij} = q_i^T k_j$

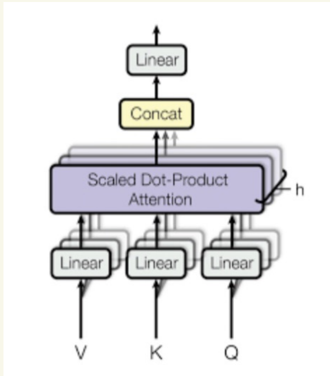
$w_{ij} = \mathrm{Softmax}(w'_{ij})$

$y_i = \sum_j w_{ij} v_j$

Multi-head attention

: Multiple "heads" of attention just means learning different sets of W_q, W_k, W_v matrices simultaneously.





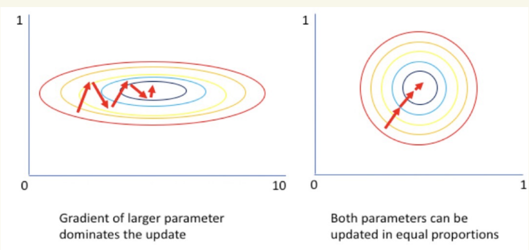
Transformer : Block of neural networks

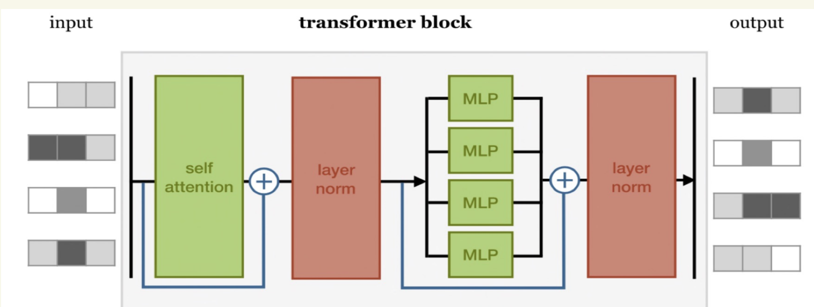
: Self-attention layer → Layer normalization → Dense layer.

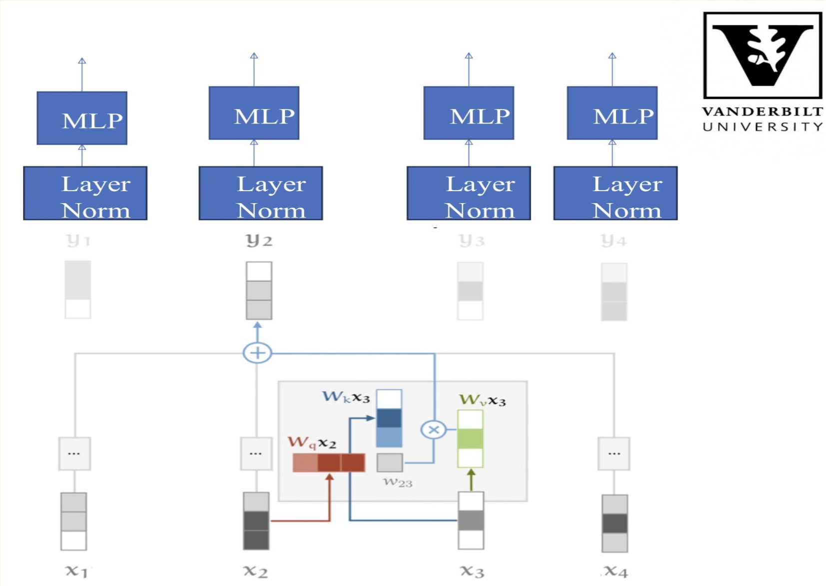
• Layer normalization : It's used for better optimization, so this allows us to optimize the network way more efficiently.

$$y_i = \lambda\left(\frac{x_i-\mu}{\sqrt{\sigma^2+\varepsilon}}\right)+\beta$$

Why? : Neural network layers work best when input vectors have uniform mean and standard deviation in each dimension







Positional Encoding

Sequence		Index of token
I	$\rightarrow$	0
am	$\rightarrow$	1
a	$\rightarrow$	2
Robot	$\rightarrow$	3

Positional Encoding Matrix

P_{00}	P_{01}	...	P_{0d}
P_{10}	P_{11}	...	P_{1d}
P_{20}	P_{21}	...	P_{2d}
P_{30}	P_{31}	...	P_{3d}

Idea: Use sines and cosines with different frequency values to encode position

Let t be the desired position in an input sentence, $\vec{p}_t \in \mathbb{R}^d$ be its corresponding encoding, and d be the encoding dimension. Then, $f: \mathbb{N} \to \mathbb{R}^d$ will be the function that produces the output vector $\vec{p}_t$ and

$$\vec{p}_t^{(i)} = f(t)^{(i)} := \begin{cases} \sin(w_k \cdot t), & \text{if } i = 2k \\ \cos(w_k \cdot t), & \text{if } i = 2k+1 \end{cases}$$

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

$$\vec{p}_t = \begin{bmatrix} \sin(w_1 \cdot t) \\ \cos(w_1 \cdot t) \\ \sin(w_2 \cdot t) \\ \cos(w_2 \cdot t) \\ \vdots \\ \sin(w_{d/2} \cdot t) \\ \cos(w_{d/2} \cdot t) \end{bmatrix}$$