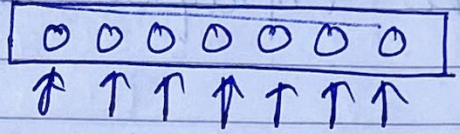
* Transformer Network.

Although LSTMs' overcome the problem of RNNs, they're still sequential models

increased complexity →

RNN → GRU → LSTM

- ↑ Attention + CNN processing.
  - Self Attention
  - Multi Head Attention



* Self Attention:

$A(q, k, v)$ = attention-based vector representation of a word

↳ calculate for each word $A^{<1>}, \ldots, A^{<6>}$.

$$A(q, K, V) = \sum_i \frac{\exp(q \cdot k^{<i>})}{\sum_j \exp(q \cdot k^{<j>})} v^{<i>}$$

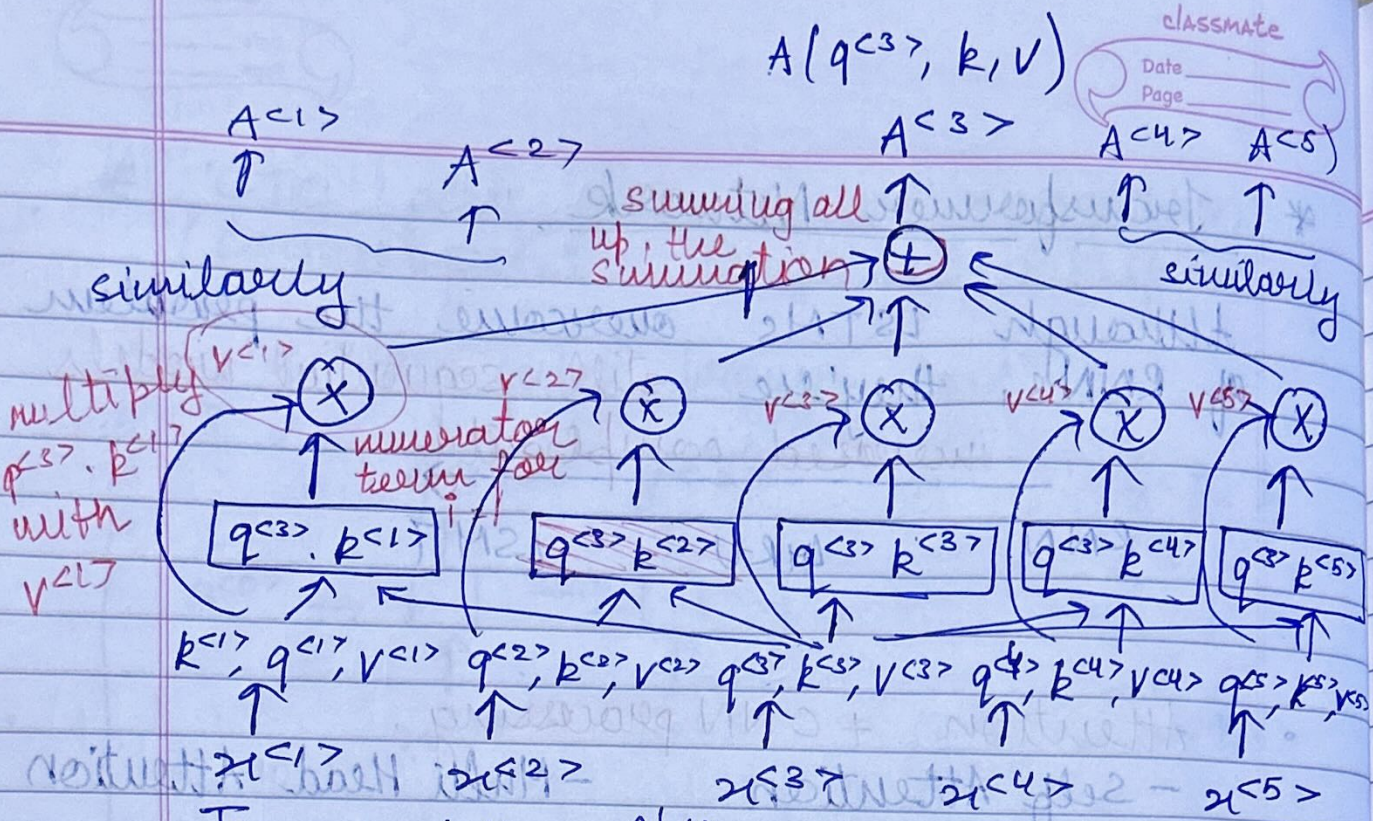
Main difference from RNN attention:

$q^{<>}$, $k^{<>}$, $v^{<>}$,

query: question you get to ask. Eg: what's happening there? (Africa)

key: most appropriate answer to query, visits within $A^{<3>}$ (there).

value: allows representation to plug in, how visite should be represented (there).



Jane visite l'Afrique en septembre.

Query (Q)	Key (K)	Value (V)
$q^{<1>}$	$k^{<1>}$	$V^{<1>}$
$q^{<2>}$	$k^{<2>}$	$V^{<2>}$
$q^{<3>}$	$k^{<3>}$	$V^{<3>}$
$q^{<4>}$	$k^{<4>}$	$V^{<4>}$
$q^{<5>}$	$k^{<5>}$	$V^{<5>}$

$$q^{<3>} = W^Q \times x^{<3>}$$
$$k^{<3>} = W^K \times x^{<3>}$$
$$V^{<3>} = W^V \times x^{<3>}$$

Scaled Dot Product Attention:

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

(compressed/vectorised way): another way of writing the formula on the previous page

$\sqrt{d_k}$ term to scale the dot product.

(for loop (for understanding) over self attention)

* Multi - Head Attention :

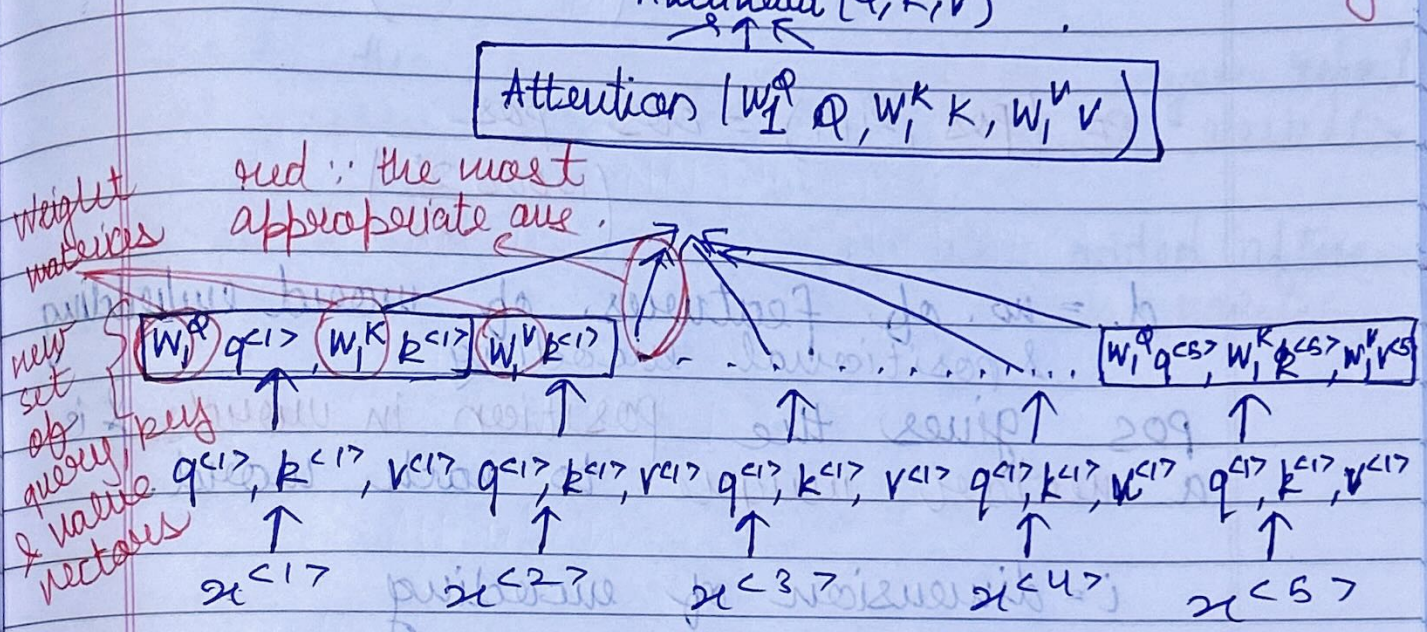
(multiple questions for every word)

Multi Head $(Q, K, V) = \text{concat}(\text{head}_1, \text{head}_2 \ldots \text{head}_h) W_o$

$\text{head}_i = \text{Attention}(W_i^Q Q, W_i^K K, W_i^V V)$

h = no. of heads

→ All the heads can be computed parallely.



Intuition : W_1^Q, W_1^K, W_1^V ⇒ help to ask & answer the question what's happening.

W_1^Q, W_1^K, W_1^V were the weights for one question (or one self attention example), similarly we have W_2^Q, W_2^K, W_2^V to answer when, then W_3^Q, W_3^K, W_3^V to answer who, and so on.

Also, note that $q^{<i>}, k^{<i>}, v^{<i>}$ here are not same as self attention ($W^Q \times x^{<>}$), here they're actually equal to word embedding $x^{<>}$. They're treated different due to architecture of transformer.

i = dimension of encoding
→ repeats twice for sin & cos

* Transformer : i to i, k = helper index to calculate i

Positional Embeddings :

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

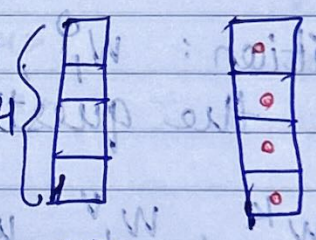
$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

d = no. of features of word embedding & positional encoding

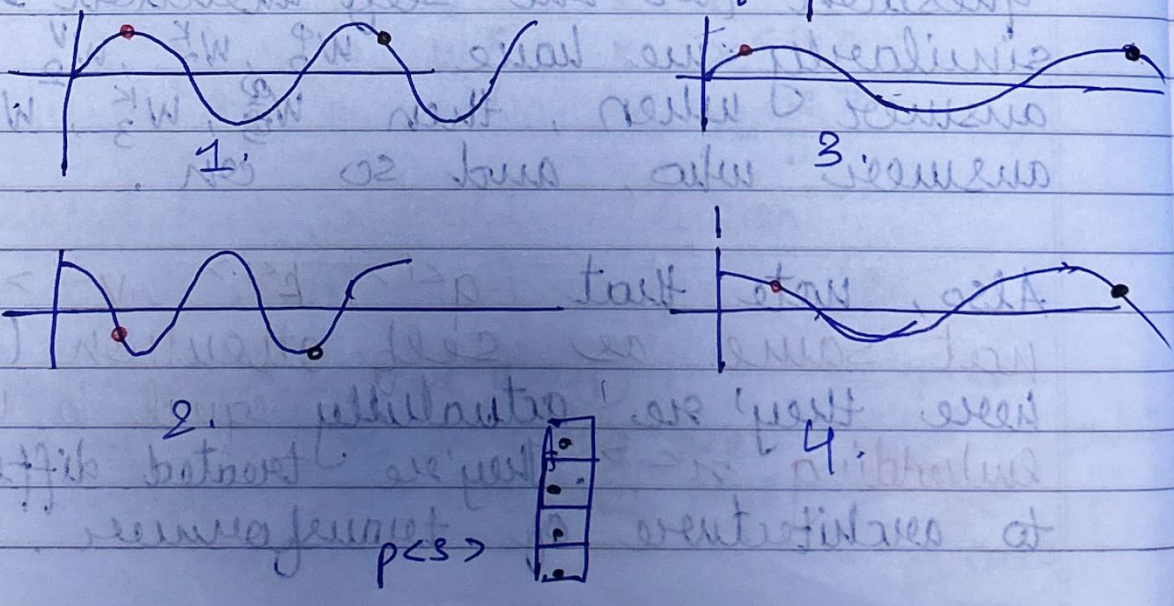
pos gives the position in words & is a vector unique to each word.

i = dimension of encoding

1. $k = 2i = 0$ $i = 0$.
2. $k = 2i + 1 = 1$ $i = 0$
3. $k = 2i = 2$ $i = 1$
4. $k = 2i + 1 = 3$ $i = 1$



= input word encoder



Input for decoder :

1. <SOS> (∵ <SOS> & <EOS> are predefined).
2. Jane
3. Jane visits
   ⋮

- Both the encoder & decoder layer repeat N times to give the optimum result.
- Add & Norm ~~nor~~ layers are added after every layer to act as Batch-Norm.