TRANSFORMER

Why? (not Convolution Networks)

- CNN is good for Structured patterns: (like images)
  - Fixed input
  - Fixed structure
- However, it is not good for unstructured patterns (languages)

⇒ Transformer

Transformer Architecture

1. Attention:

$q \in \mathbb{R}^c$

$K = [k_1, k_2, \dots] \in \mathbb{R}^c$

$V = [v_1, v_2, \dots] \in \mathbb{R}^c$

Attention $\rightarrow$ $o = \sum \alpha_i v_i$,

where $\alpha_i = \dfrac{e^{\frac{q^T k_i}{\sqrt{c}}}}{\sum_j e^{\frac{q^T k_j}{\sqrt{c}}}}$

$\text{softmax}\left(\dfrac{q^T K}{\sqrt{c}}\right)$ <---

Basically:

- For each word, a "query" is made on all words $[K \rightarrow V]$ in the current context window (sentence, paragraph, chunk...) and return the relationship matrix that tells us "how much this word relates to all words"

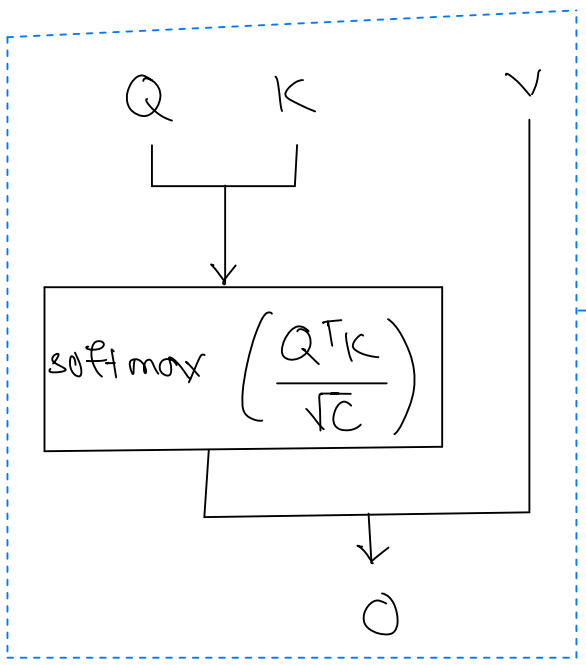
For example: Consider this sentence "cat chased mouse"

"cat chased mouse" $\xrightarrow{\text{Tokenized}}$ $X^T = \begin{bmatrix} x_0^T \\ x_1^T \\ x_2^T \end{bmatrix}$ "cat" "chased" "mouse"

$\downarrow$ Embedding (explained later)

$e(X^T) = \begin{bmatrix} e(x_0^T) \\ e(x_1^T) \\ e(x_2^T) \end{bmatrix}$ $\}$ m, underbrace c

$\downarrow$ Attention



- Init $Q^T = K^T = W^T = e(X^T)$
- Calculate relationship matrix: (attention matrix)

$O = \text{Attention}(Q, K, V)$

$= \text{softmax}\left(\dfrac{Q^T K}{\sqrt{C}}\right) \cdot V^T$

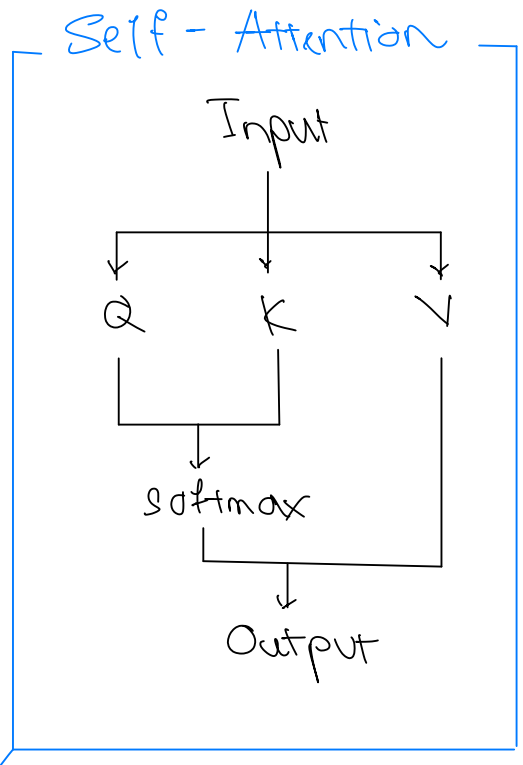
$Q \in \mathbb{R}^{n \times c}$

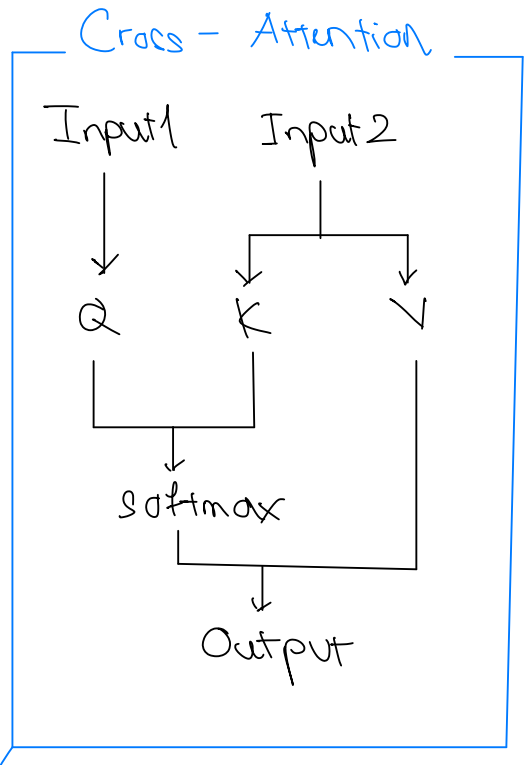
$K \in \mathbb{R}^{n \times c}$

$V \in \mathbb{R}^{n \times c}$

return matrix $\begin{bmatrix} a_{00} & & \\ & \ddots & \\ & & a_{mn} \end{bmatrix}$

• Different types of Attention





↳ Used in NLP:

- Understand the relationship between words

↳ Used in Advanced CV task:

- In task like object detection, the "query" is not coming from the image, but outside of it

When to choose which? Intuition

→ Think of it like this, Attention's goal is to capture context using the question (query) and answers (keys, values) mechanism:

- In NLP task: the questions and answers come from the same source i.e "How is "cat" relates to "mouse" " in the sentence "cat chased mouse".
- In CV task: the questions and answers are not from the same source. We don't ask "How is this pixel relates to the other pixels?", instead we ask "Where is the car?" (assumming we are building a car detection model)

Issues with traditional Attention:

For any token, it will always "attend" to itself more than other tokens.

Because: $\text{Attention}(X) = \text{softmax}\left(\frac{X^T X}{\sqrt{C}}\right) X^T = \begin{bmatrix} \alpha_{11} & \cdots & \alpha_{1N} \\ \vdots & \ddots & \\ \alpha_{N1} & & \alpha_{NN} \end{bmatrix} \begin{bmatrix} x_1^T \\ \vdots \\ x_N^T \end{bmatrix}$

And: $x_i^T x_j \leq \frac{1}{2}\left(x_i^T x_i + x_j^T x_j\right)$

$\leq \max\left(x_i^T x_i , x_j^T x_j\right)$

$\Rightarrow \quad \alpha_{ij} \leq \max\left(\alpha_{ii} , \alpha_{jj}\right)$

So: . Diagonal in relationship matrix will always be greater than off-diagonals.

Proof: $x_i^T x_j \leq \frac{1}{2}\left(x_i^T x_i + x_j^T x_j\right)$

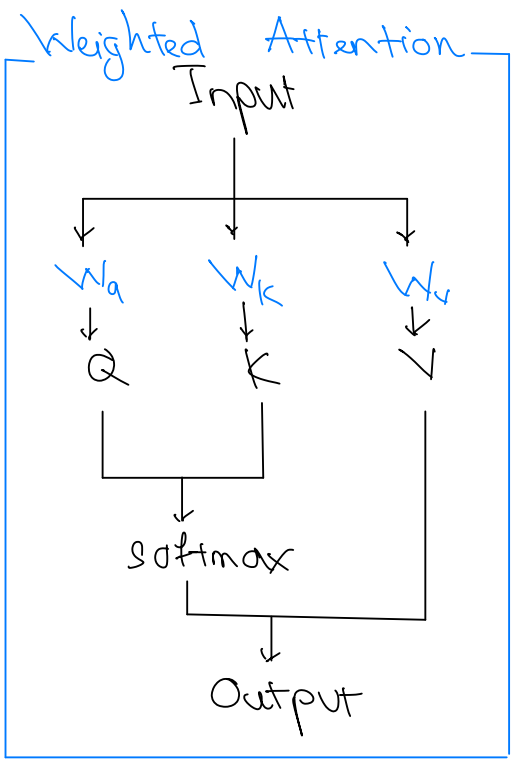
. Cosine distance: $x_i^T x_j = |x_i| \cdot |x_j| \cdot \cos\theta \quad (0° \leq \theta \leq 90°)$

$\leq |x_i| \cdot |x_j|$

$\leq \frac{1}{2}\left(|x_i|^2 + |x_j|^2\right)$ <AM-GM inequality>

$= \frac{1}{2}\left(x_i^T x_i + x_j^T x_j\right)$

Solution: Attention with weights



$\text{Attention}(X; \{W_q, W_k, W_v\})$

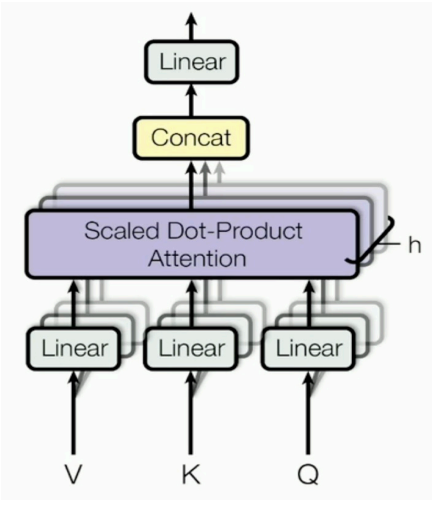
$= \text{Attention}(XW_q , XW_k , XW_v)$

$= \text{softmax}\left(\frac{(XW_q)^T (XW_k)}{\sqrt{d_k}}\right)(XW_v)$

$\Rightarrow$ These weights allow Attention to focus more on others than just itself.

Multi-head Attention

If we use just 1 attention (relationship) matrix, then we won't be able to capture different aspects of the text. So, multi-head attention is used, its basically just concatenate many attention matrices



Recap on Attention

- Always use attention with weights
- Self-attention with weights generalizes a 1×1 2D Conv
- Multi-head attention with h heads generalizes a $\sqrt{h} \times \sqrt{h}$ 2D Conv

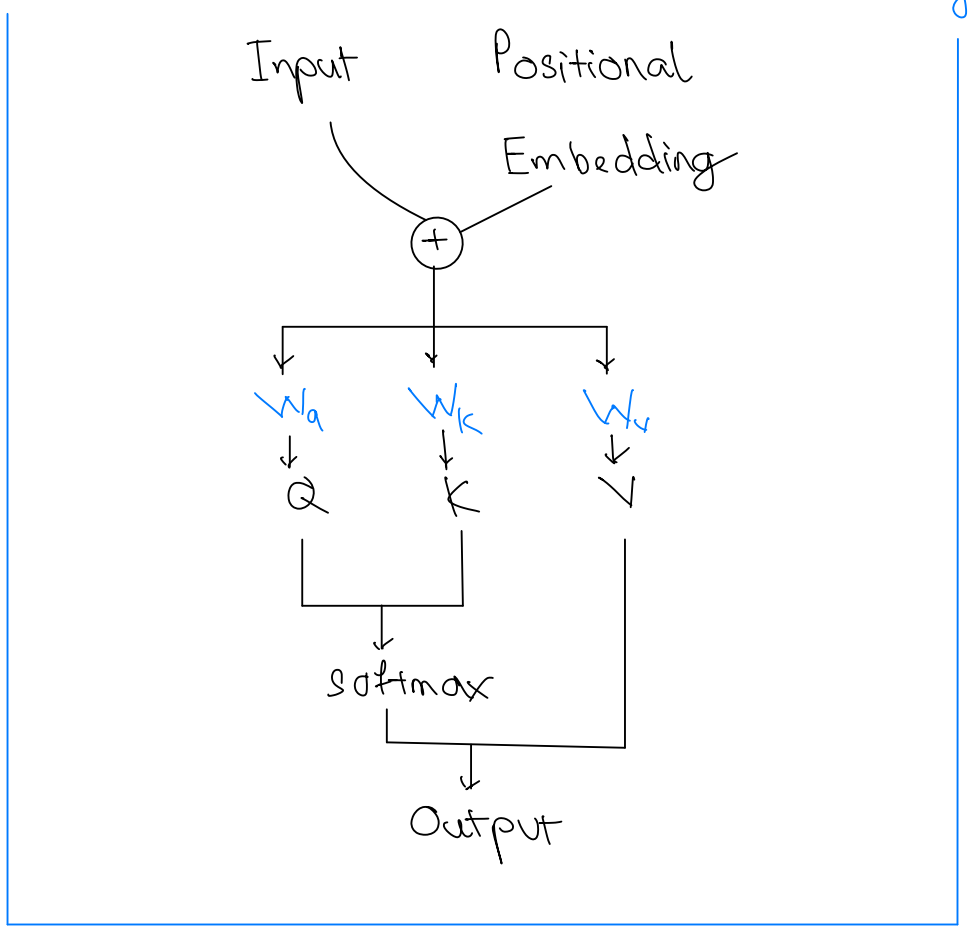
  ⟶ Always use multi-head attention

2. Positional Embeddings:

So far we talked about Attention as a **Set Operator**, meaning if we apply the same permutation matrix to Q, K, V, we'd get the same result. This might not be an issue in scenario where order doesn't matter like "predict the number of power outages in the future, given a set of reported power outages in the past".

However, tasks that involve **Natural Language**, **Speech**, **Images** **require positional embeddings** (changing the sequence of elements could resulted in completely different outcome)

Attention with Positional Embedding

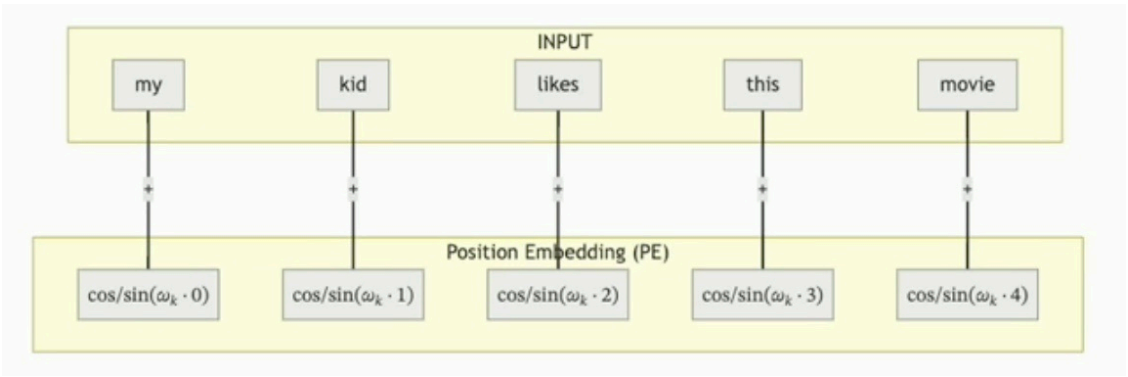


Types of Positional Embeddings

1. Absolute embedding:

Basically indexing the input

2. Sinusoidal Positional Embedding: (kinda absolute)



- Use sine & cosine functions with varied frequencies:

$$PE^{(i)} = \begin{cases} \sin(\omega_k \cdot t) & \text{if } i = 2k \\ \cos(\omega_k \cdot t) & \text{if } i = 2k+1 \end{cases}$$

$i: 0 \to d$

$$\omega_k = \frac{1}{1000^{2k/d}}$$

position

Pros:

- "Kind of" absolute
- Position represented by frequecies

Example: Consider sentence "cat chases mouse"

```python
# Let's make a simple version with 3 frequency pairs
def get_position_encoding(pos, d_model=6):
    embedding = []

    # First frequency pair (fastest waves)
    embedding.append(sin(pos / 1))     # dimension 0
    embedding.append(cos(pos / 1))     # dimension 1

    # Second frequency pair (medium waves)
    embedding.append(sin(pos / 100))   # dimension 2
    embedding.append(cos(pos / 100))   # dimension 3

    # Third frequency pair (slowest waves)
    embedding.append(sin(pos / 10000))  # dimension 4
    embedding.append(cos(pos / 10000))  # dimension 5

    return embedding

# Let's see what each word gets:
cat_pos    = get_position_encoding(0)  # position 0
chases_pos = get_position_encoding(1)  # position 1
mouse_pos  = get_position_encoding(2)  # position 2

# Approximate values:
cat_pos    ≈ [0.00,  1.00,  0.00,  1.00,  0.00,  1.00]
chases_pos ≈ [0.84,  0.54,  0.01,  1.00,  0.00,  1.00]
mouse_pos  ≈ [0.91, -0.42,  0.02,  1.00,  0.00,  1.00]
```

fast waves | medium waves | slow waves

Intuition:

Works like a radar, where:

- fast waves can give exact position, but dies early
- slow waves are not as precise, but live very long (reach very far)

- Words to the right is rotated from word to the left:

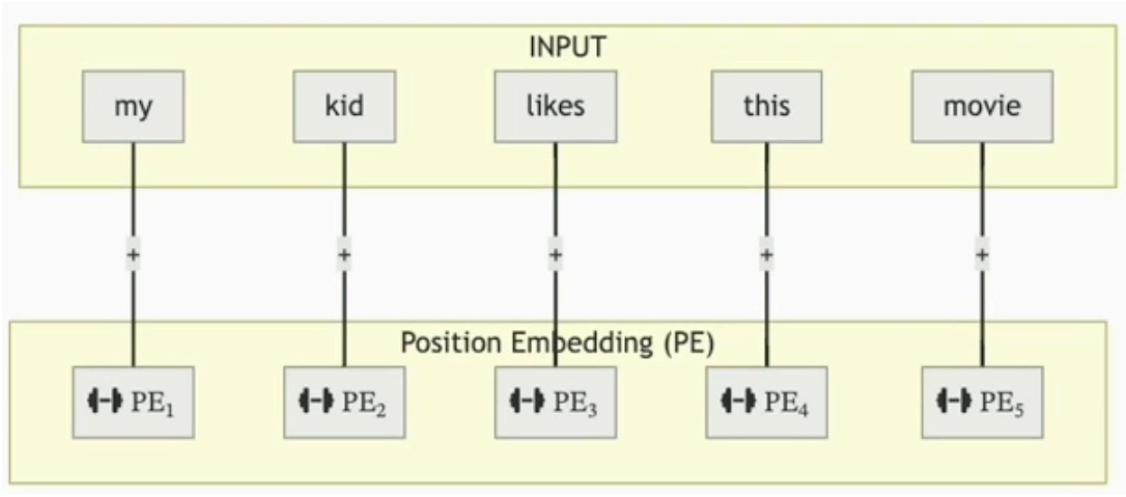
  "cat": [0.00, 1.00] ⟶ "chases": [0.84, 0.54]

- Basically, if the neural network see the word "chases", it

can: check the fast waves to know its right after "cat",
check the medium waves to know its in the middle of the sentence,
check the slow waves to know its relative position in the document.

3. Learnable Positional Embedding

- Attach weights to each token and learn it.
- Use when frequency info is not obvious
- Only used if training data is longer than test data



Example: Vision Transformer

4a. T5 - bias Positional Embedding: (relative)

- Learn the relationship between elements in the sequence
- Instead of trying to assign an absolute value for an element, it learn "how far" any pair of elements to each other.
- Good for when we don't know the length of training or test sequence

$$O = \text{softmax}\left(\frac{(XW_q)^T XW_K + B}{\sqrt{C}}\right)(XW_V)^T$$

Learnable bias:

$$B = \begin{bmatrix} b_0 & b_{-1} & \cdots & b_{-N+1} \\ b_1 & b_0 & \ddots & b_{-N+2} \\ \vdots & \ddots & \ddots & \vdots \\ b_{N-1} & \cdots & \cdots & b_0 \end{bmatrix}$$

4b. Alibi Positional Embedding (relative)

Same as T5 - Bias but instead of learning the full weights, we only learn the scaled(α) of the difference of a pair elements

$$O = \text{softmax}\left(\frac{(XW_q)^T XW_K + B}{\sqrt{C}}\right)(XW_v)^T$$

Learnable Bias

$$B = \alpha \begin{bmatrix} 0 & 0 & \cdots & 0 \\ -1 & 0 & & \\ \vdots & & \ddots & \\ -N+1 & -N+2 & & 0 \end{bmatrix}$$

5. Rotary Positional Embedding : RoPE

- Current SOTA
- Both absolute and relative
- Fist, RoPE use the same intuition as Sinusoidal Embedding to find the absolute encoding. Instead of adding sinusoidal value, it rotates the word directly using a rotation matrix. Then, the moment this rotated word is fed to Attention, the positional embedding will be relative

Step-by-step:

1. Calculate the rotation matrix for each position (same as Sinusoidal)

$$\theta = pos \cdot \left(\frac{n}{1000^{2k/d}}\right)$$

$$R(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

For example, if I have a word embedding of 4 dimensions at position 5, then:

```
# Let's say we have a word embedding of dimension d=4
word_embedding = [x0, x1, x2, x3]

# For position pos=5, we calculate two rotation matrices:
# First pair (dims 0,1) - faster rotation
theta1 = 5 * (1/10000)^(0/d)  # bigger angle
R1 = [cos(theta1) -sin(theta1)]
     [sin(theta1)  cos(theta1)]

# Second pair (dims 2,3) - slower rotation
theta2 = 5 * (1/10000)^(2/d)  # smaller angle
R2 = [cos(theta2) -sin(theta2)]
     [sin(theta2)  cos(theta2)]

# Apply rotations to pairs:
[x0_rot, x1_rot] = R1 @ [x0, x1]
[x2_rot, x3_rot] = R2 @ [x2, x3]

rotated_embedding = [x0_rot, x1_rot, x2_rot, x3_rot]
```

→ do the same for other embeddings

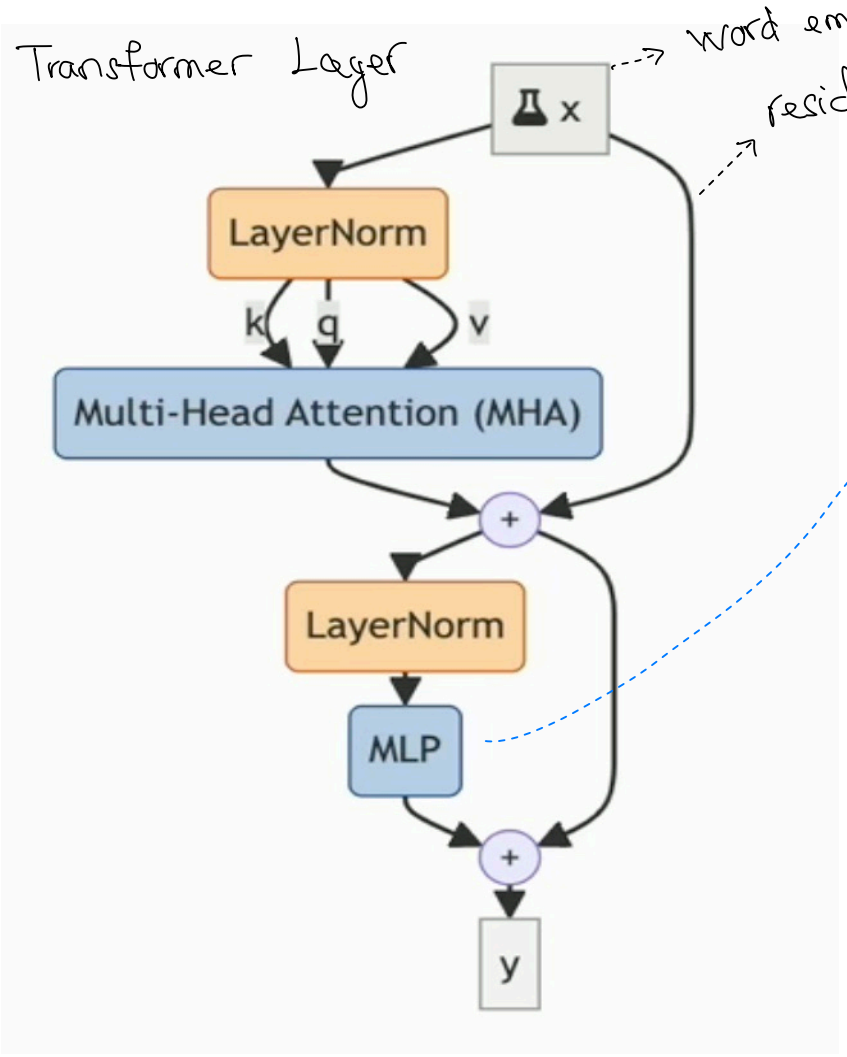
2. Feed the rotated embedding through Attention

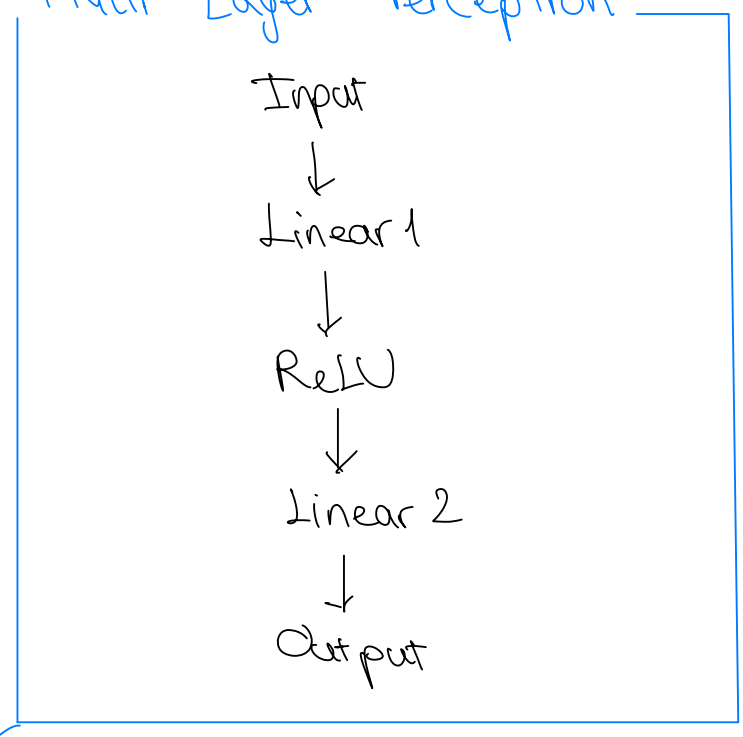
Consider just 1 rotated query and 1 rotated key at 2 positions i and j :

$$\begin{aligned} \text{attention} &\sim q_i^T k_j \\ &\sim \left(R(\theta_i)\, x_i\right)^T R(\theta_j)\, x_j \\ &\sim R(\theta_i)^T R(\theta_j) \\ &\sim R(\theta_i - \theta_j) \end{aligned}$$

⇒ This gives us the relative position based on how much it rotates

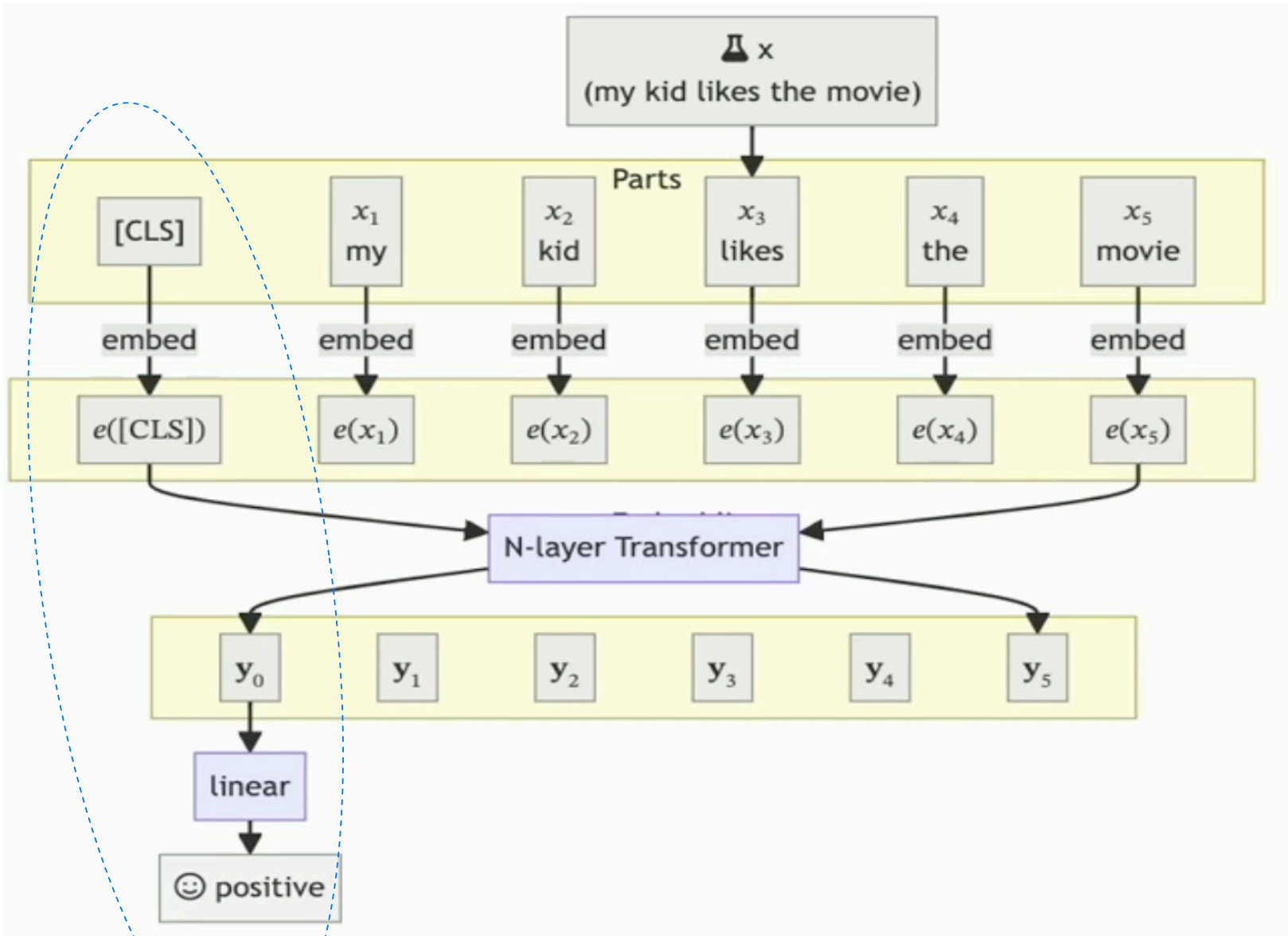
Transformer Architecture (finally)





Purpose: Computational efficient, the alternative is more attention layers (which is more expensive)

That is it, now if we project to more word embeddings and more transformer layers, we would have something like this



Task specific input: Sentiment analysis

Examples:

- [Positive] My kid likes this movie
- [Negative] My kid doesn't like this movie

The first input

Why place it right next to other words and not after?

Because attention mechanism will output attention score $\text{softmax}\left(\frac{q^T K}{\sqrt{c}}\right)V^T$ that show how much the "sentiment token" attend to other words.