

# Introduction To Transformers

- 1) RNN / LSTM / GRU RNN
- 2) Encoder Decoder Architecture
- 3) ATTENTION MECHANISM
- 4) TRANSFORMERS

① Why Transformers?

② Architecture of Transformers?

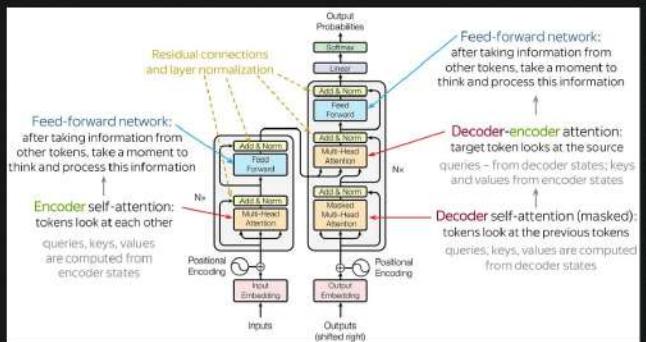
③ SELF ATTENTION  $\rightarrow Q, K, V$

④ Positional Encoding

⑤ Multi Head ATTENTION

⑥ Combining the Working of Transformers

## Architecture.



Generative AI  $\rightarrow$  LM, Multimodel

BERT, GPT  $\leftarrow$

OpenAI  $\rightarrow$  ChatGPT

GPT-4.0

## ① What And Why $\rightarrow$ Transformers

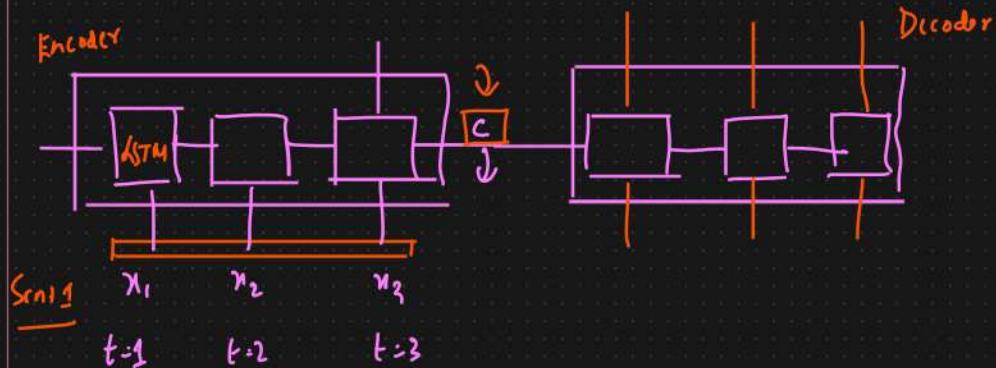
Transformers in natural language processing (NLP) are a type of deep learning model that use self-attention mechanisms to analyze and process natural language data. They are encoder-decoder models that can be used for many applications, including machine translation.  $\Rightarrow$  Seq2Seq Task

Eg: Language Translation  $\rightarrow$  Google Translation

English  $\rightarrow$  French

if  $\Rightarrow$  Many  $\rightarrow$  O/p: Many.  $\{$  Length of the sentence  $\}$ .

### Encoder - Decoder



Sentence Length  $\uparrow \uparrow$

Beam Score  $\downarrow \downarrow$

Length Sentence  $\uparrow \uparrow$

### 3.1 DECODER: GENERAL DESCRIPTION

In a new model architecture, we define each conditional probability in Eq. (2) as:

$$p(y_i | y_1, \dots, y_{i-1}, s_i, c_i) = g(y_{i-1}, s_i, c_i), \quad (4)$$

where  $s_i$  is an RNN hidden state for time  $i$ , computed by

$$s_i = f(s_{i-1}, y_{i-1}, c_i).$$

It should be noted that unlike the existing encoder-decoder approach (see Eq. (2)), here the probability is conditioned on a distinct context vector  $c_i$  for each target word  $y_i$ .

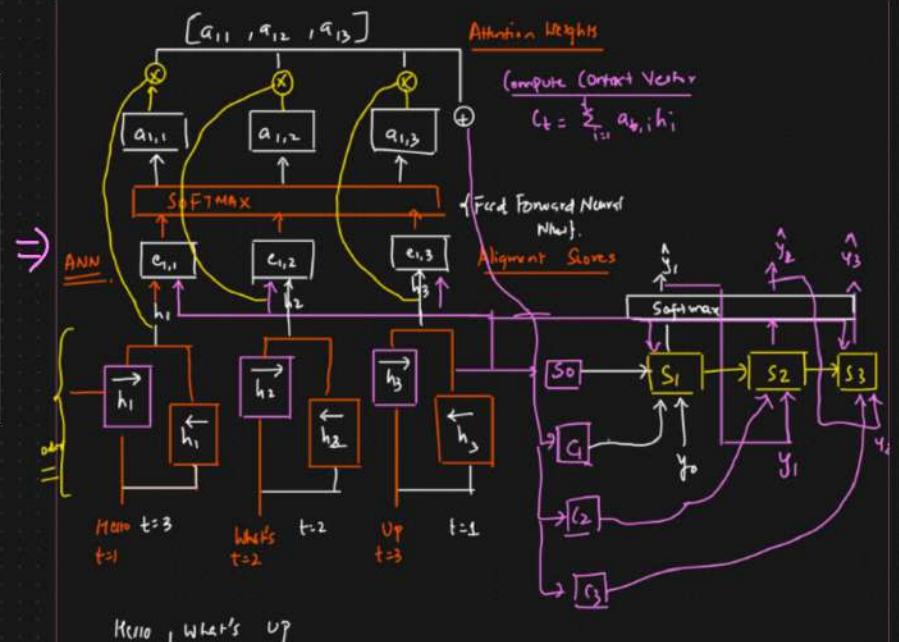
The context vector  $c_i$  depends on a sequence of annotations ( $h_1, \dots, h_{T_s}$ ) to which an encoder maps the input sentence. Each annotation  $h_i$  contains information about the whole input sequence with a strong focus on the parts surrounding the  $i$ -th word of the input sequence. We explain in detail how the annotations are computed in the next section.

The context vector  $c_i$  is, then, computed as a weighted sum of these annotations  $h_j$ :

$$c_i = \sum_{j=1}^{T_s} \alpha_{ij} h_j. \quad (5)$$

**Additional Context → Decoder**

**long Sentence Accuracy ↑**



Attention Mechanism

① Parallelly we cannot send all the words in a sentence → Scalable

DATASET → Huge → Scalable With Respect to Training.

TRANSFORMERS \* LSTM RNN

↳ Self Attention Module ← All the words will be parallelly sent to encoder.

↓

Positional Encoding

Transformer ↑ DATASET → Amazing SOTA ← NLP

Transfer Learning → MultiModal Task → NLP + Image ←

Transformers ÷ AI Space → SOTA Model →

↓ ↗ ↓

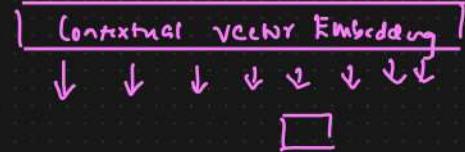
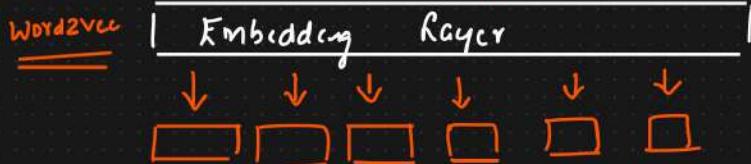
BERT GPT → Transfer learning → SOTA Models → DALLE → Generative AI

↑  
Train huge Data

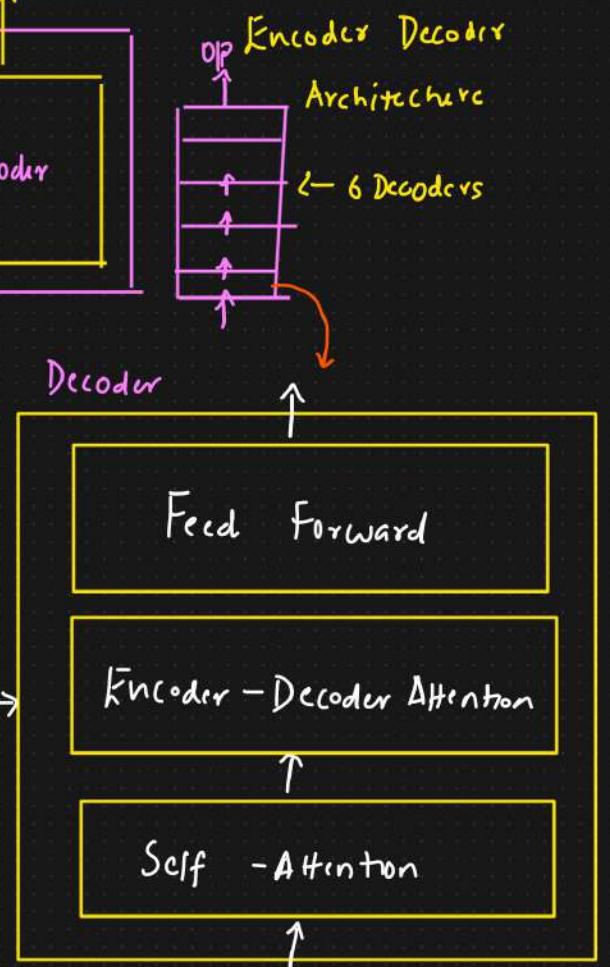
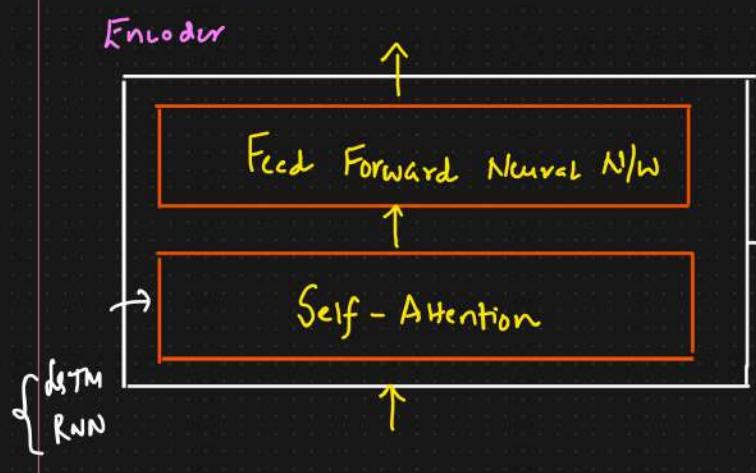
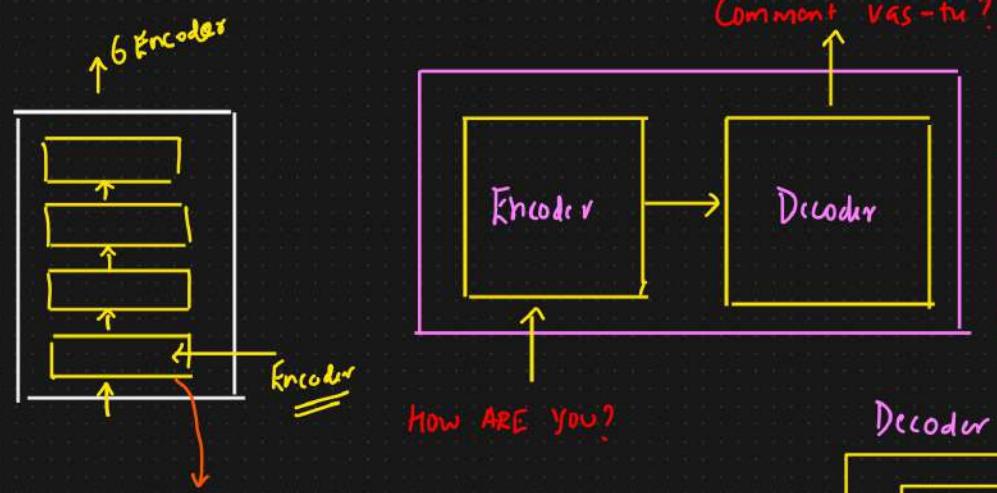
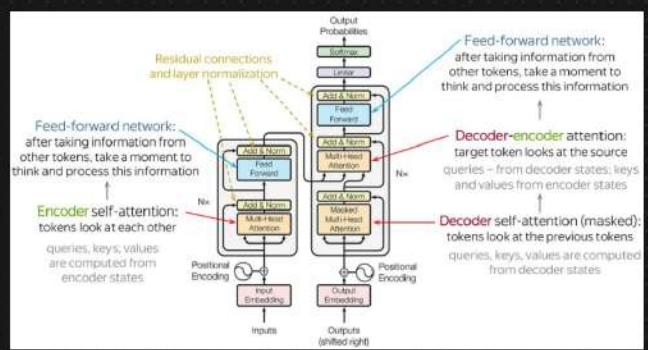
AI M's

## ② Contextual Embedding → Self Attention → Contextual Vector

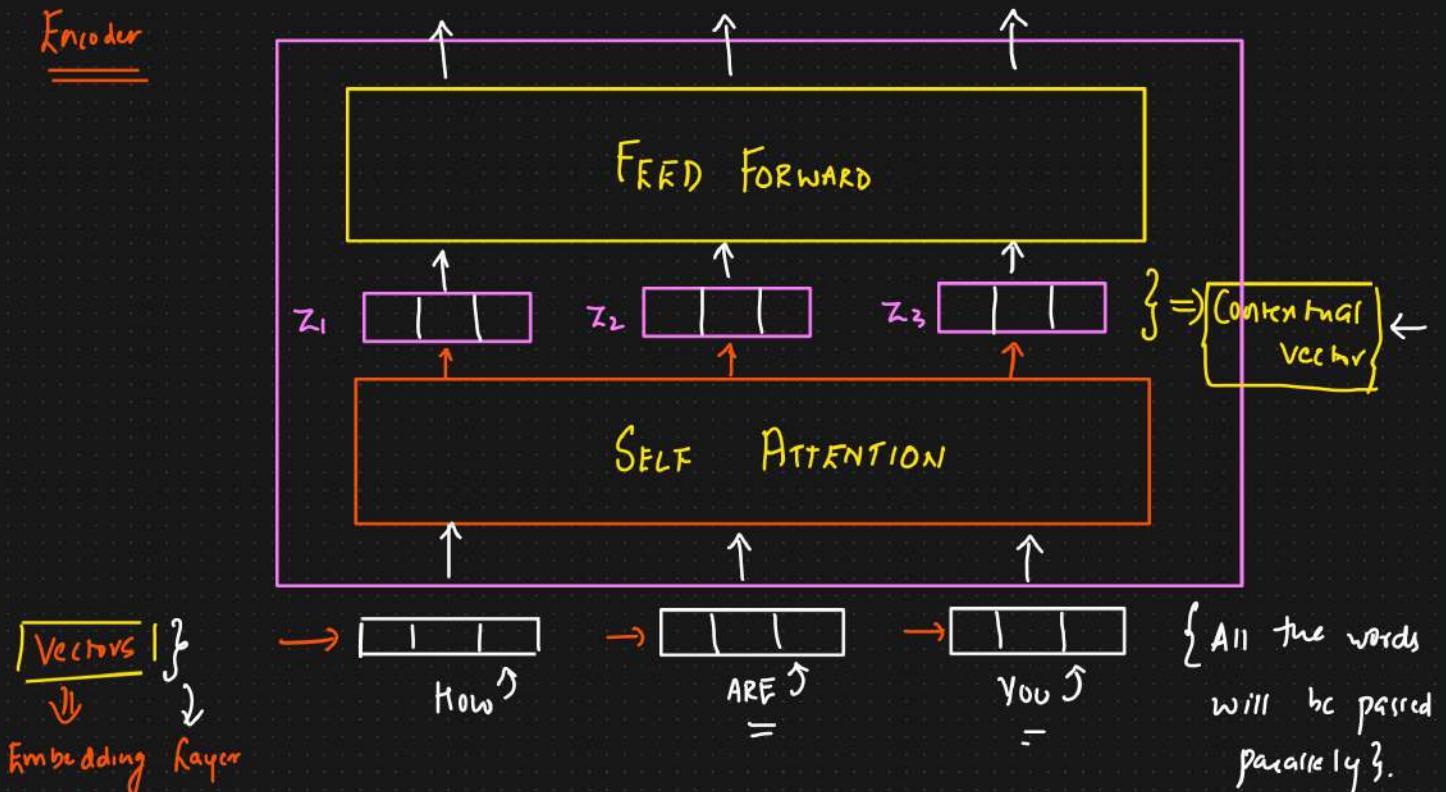
Eg: My name is KRISH = And I play CRICKET



## ② Basic Transformer Architecture {Seq2Seq Task} → Language Translation {Eng → French}

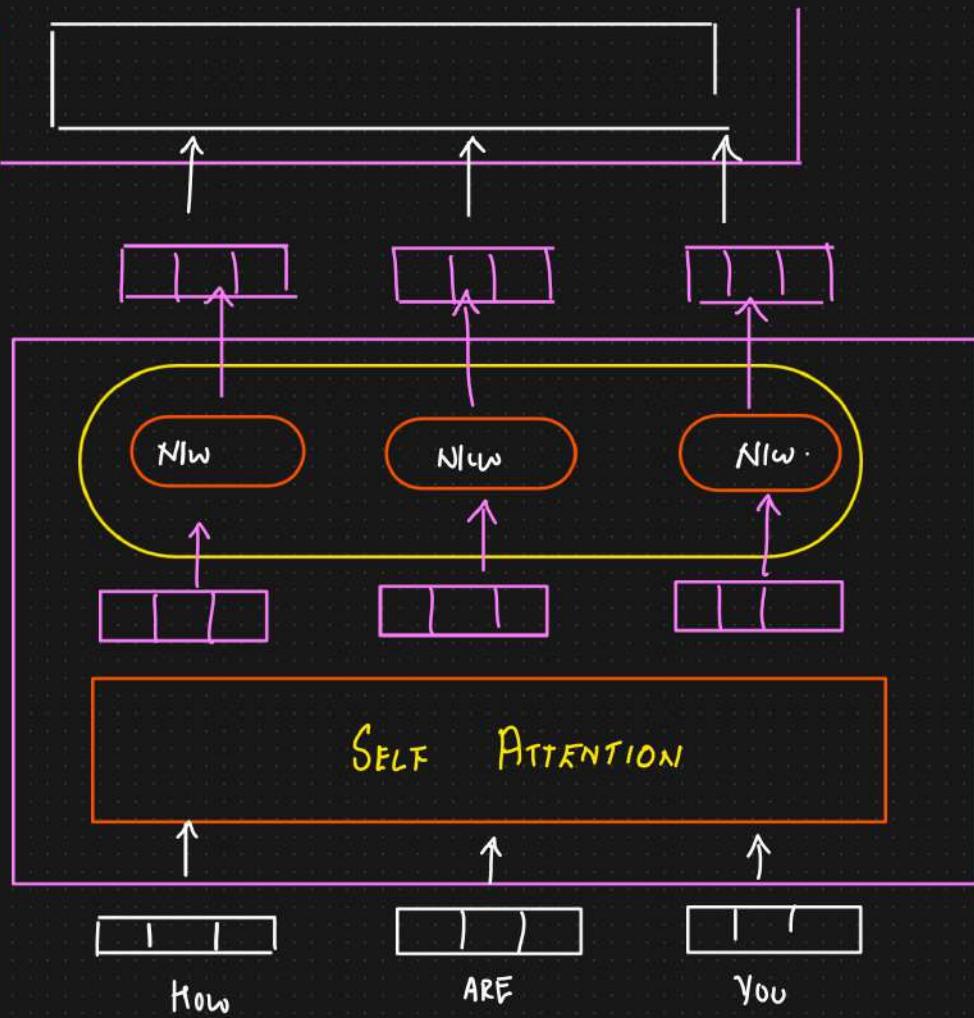


## Encoder



## Encoder 2

## Encoder



## Self Attention At a Higher Level

Eg: The cat sat on the mat, the cat lay on the rug.

Word Embedding  
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

SELF ATTENTION

↓  
The. . . . .

The

→ [Cat] → [Cat] → [ | | | ]  
1 → [Sat] Sat

{ Contextual Embedding }

Rank 2 → on  
= 3 → the  
mat

mat → [ - | - | - ]



### Self Attention In Detail

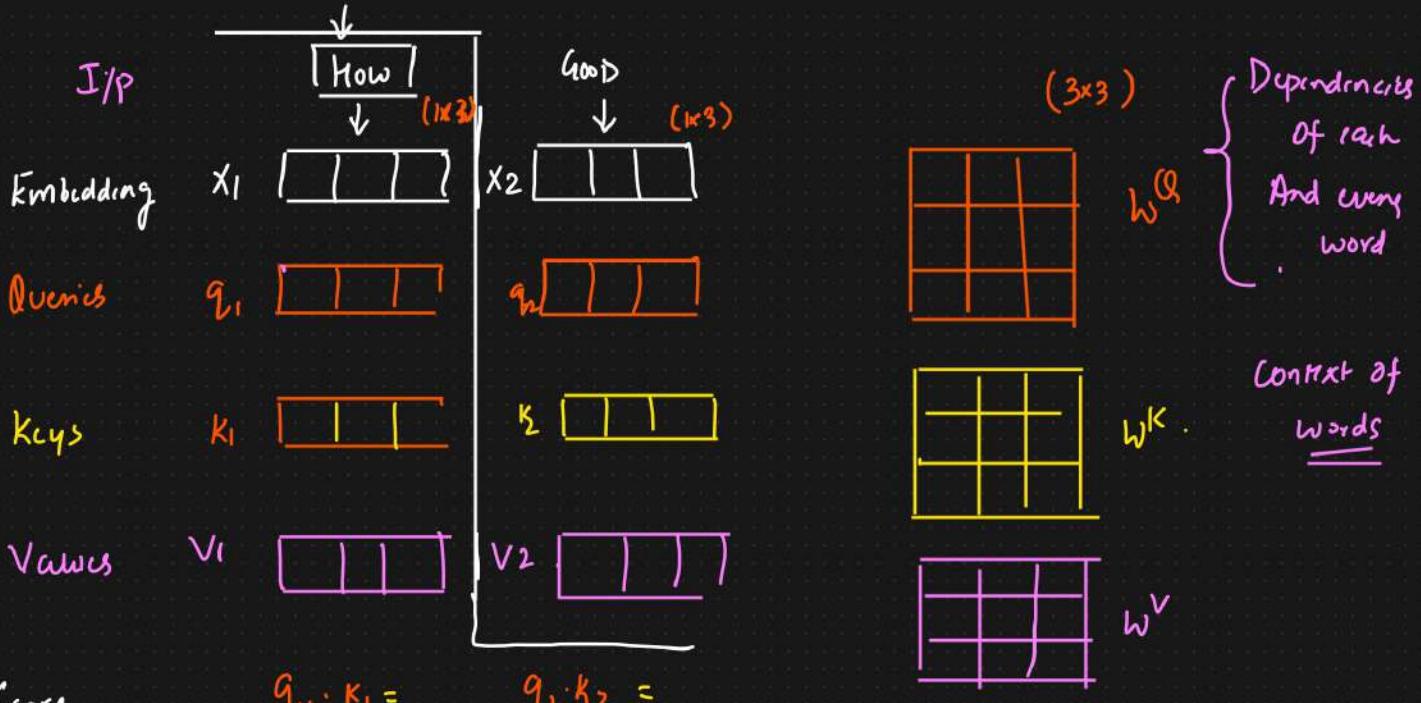
- ① To Create 3 vectors from each of the encoder i/p. Query vector, Key vector, Value vector.  $\Rightarrow$  Contextual Embedding

Eg: YT  $\Rightarrow$  Search keywords  $\Rightarrow \{ \text{Query} \}$ .

Query  $\rightarrow \{ \text{Key} \} \xrightarrow{\text{Topic}} \text{Tags}$   
 $\xrightarrow{\text{Description}} \text{Title} \xrightarrow{\text{Title}} \text{Value} \Rightarrow \text{i/p Video}$

- ② Second step in calculating self attention is to calculate the Score.

The Score determines how much focus to place on the other part of the sentence.



$$\text{Score} \quad q_1 \cdot k_1 = \quad q_1 \cdot k_2 =$$

④ How much focus to place on other part of the I/P Sentence as we encode a word at certain position

Divide  $\sqrt{dk}$

More Stable  $\left\{ \begin{array}{l} \text{gradients} \\ \text{variance} \end{array} \right\}$

Dimension = 3      Dimension  $\uparrow = T12$

$\downarrow$       O/p  $\uparrow\uparrow\uparrow$

$\frac{\text{variance}}{\sqrt{dk}} \approx \frac{\text{variance}}{\sqrt{dk}}$

SoftMax

$\frac{[0-1]}{[0-1].}$

$0.88 + 0.12 = 1$

Softmax  $\Rightarrow$  The Softmax score determines how much each word will be expressed at this position

X      X      X

Value Vectors       $v_1 \begin{bmatrix} | & | \end{bmatrix}$        $v_2 \cdot \begin{bmatrix} | & | & | \end{bmatrix}$

$0.88 \begin{bmatrix} | & | & | \end{bmatrix}$

$\frac{0.88}{0.88 + 0.12} + \frac{0.12}{0.88 + 0.12}$

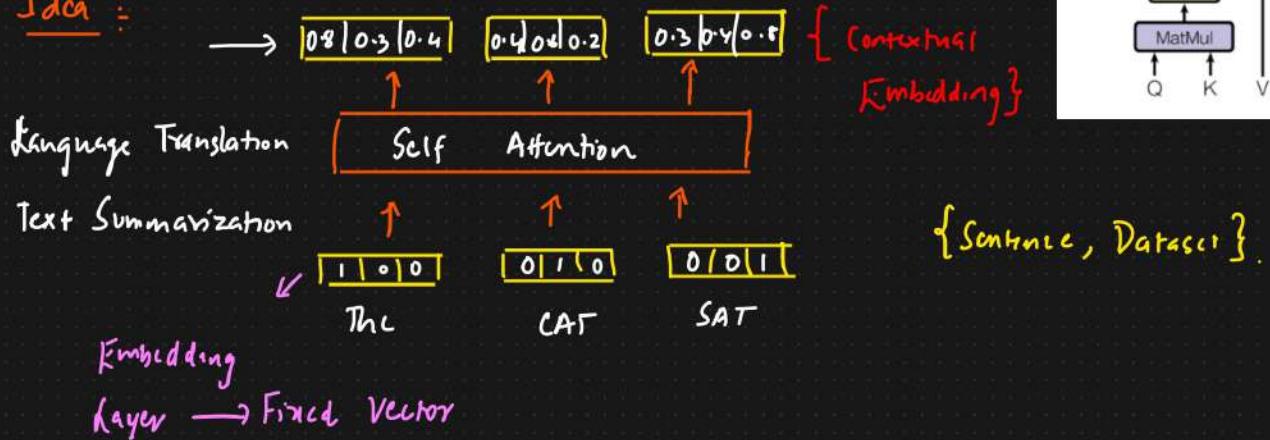
$\downarrow \quad \downarrow$

Contextual Vector  $\rightarrow \begin{bmatrix} | & | & | \end{bmatrix}$        $\rightarrow \begin{bmatrix} | & | & | \end{bmatrix}$

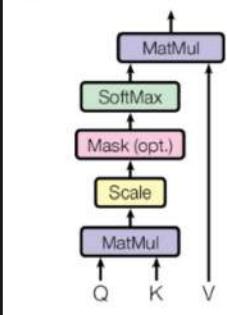
## Self Attention At Higher And Detailed Level

Self-attention, also known as scaled dot-product attention, is a crucial mechanism in the transformer architecture that allows the model to weigh the importance of different tokens in the input sequence relative to each other.

Idea :



Scaled Dot-Product Attention



i) Inputs: Queries, Keys, And Values

Model  $\rightarrow$  Queries, Keys And Values

## 1. Query Vectors (Q):

Role: Query vectors represent the token for which we are calculating the attention. They help determine the importance of other tokens in the context of the current token.

Importance:

**Focus Determination:** Queries help the model decide which parts of the sequence to focus on for each specific token. By calculating the dot product between a query vector and all key vectors, the model assesses how much attention to give to each token relative to the current token.

**Contextual Understanding:** Queries contribute to understanding the relationship between the current token and the rest of the sequence, which is essential for capturing dependencies and context.

## 2. Key Vectors (K):

Role: Key vectors represent all the tokens in the sequence and are used to compare with the query vectors to calculate attention scores.

Importance:

**Relevance Measurement:** Keys are compared with queries to measure the relevance or compatibility of each token with the current token. This comparison helps in determining how much attention each token should receive.

**Information Retrieval:** Keys play a critical role in retrieving the most relevant information from the sequence by providing a basis for the attention mechanism to compute similarity scores.

## 3. Value Vectors (V):

Role: Value vectors hold the actual information that will be aggregated to form the output of the attention mechanism.

Importance:

**Information Aggregation:** Values contain the data that will be weighted by the attention scores. The weighted sum of values forms the output of the self-attention mechanism, which is then passed on to the next layers in the network.

**Context Preservation:** By weighting the values according to the attention scores, the model preserves and aggregates relevant context from the entire sequence, which is crucial for tasks like translation, summarization, and more.

$$\text{Input Sequence} = ["\text{The}", "(\text{AT})", "\text{SAT}"]$$

Embedding Size = 4

$Q, K, V \Rightarrow 4$

$$\xrightarrow{\text{L}} \boxed{\text{S} \text{ H} \text{ A} \text{ T}} \rightarrow \boxed{\text{O} \text{ P} \text{ Z} \text{ O} \text{ S}}$$

① Token Embedding

$$E_{\text{The}} = [1 \ 0 \ 1 \ 0]$$

$$E_{\text{SAT}} = [1, 1, 1, 1]$$

$$E_{\text{CAT}} = [0 \ 1 \ 0 \ 1]$$

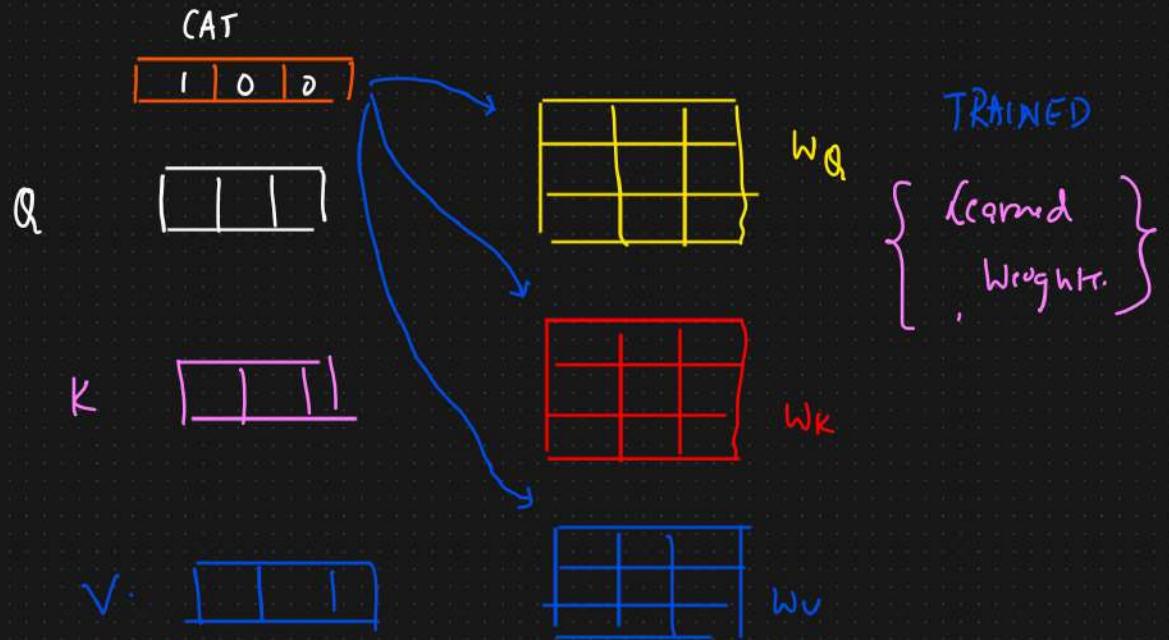
$\downarrow$   
Sentence, Dataset

1 0 1 0

1 0 1 0

## ② Linear Transformation

We create  $Q, K, V$  by multiplying the embeddings by learned weights matrices  $W_Q$ ,  $W_K$  and  $W_V$ .



Let's consider

$$W_Q = W_K = W_V = I \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \leftarrow$$

$$Q_{\text{Trained}} = [1 \ 0 \ 1 \ 0] \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [1 \ 0 \ 1 \ 0]$$

$$K_{\text{Trained}} = [1 \ 0 \ 1 \ 0]$$

$$V_{\text{Trained}} = [1 \ 0 \ 1 \ 0]$$

$$\textcircled{1} \quad Q_{\text{Trained}} = K_{\text{Trained}} = V_{\text{Trained}} = [1 \ 0 \ 1 \ 0]$$

$$\textcircled{2} \quad Q_{\text{CAT}} = K_{\text{CAT}} = V_{\text{CAT}} = [0 \ 1 \ 0 \ 1]$$

$$\textcircled{3} \quad Q_{\text{SAT}} = K_{\text{SAT}} = V_{\text{SAT}} = [1, 1, 1, 1]$$

### ③ Compute Attention Scores

$$\begin{bmatrix} 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

The

$$\text{Score}(Q_{\text{The}}, K_{\text{The}}) = [1 \ 0 \ 1 \ 0] \cdot [1 \ 0 \ 1 \ 0]^T = 2$$

$$\text{Score}(Q_{\text{The}}, K_{\text{CAT}}) = [1 \ 0 \ 1 \ 0] \cdot [0 \ 1 \ 0 \ 1]^T = 0$$

$$\text{Score}(Q_{\text{The}}, K_{\text{SAT}}) = [1 \ 0 \ 1 \ 0] \cdot [1 \ 1 \ 1 \ 1]^T = 2$$

For the token CAT

$$\text{Score}(Q_{\text{CAT}}, K_{\text{The}}) = [0 \ 1 \ 0 \ 1] \cdot [1 \ 0 \ 1 \ 0]^T = 0$$

$$\text{Score}(Q_{\text{CAT}}, K_{\text{CAT}}) = [0 \ 1 \ 0 \ 1] \cdot [0 \ 1 \ 0 \ 1]^T = 2$$

$$\text{Score}(Q_{\text{CAT}}, K_{\text{SAT}}) = [0 \ 1 \ 0 \ 1] \cdot [1 \ 1 \ 1 \ 1]^T = 2.$$

For the Token SAT

$$\left\{ \begin{array}{l} \text{Score}(Q_{\text{SAT}}, K_{\text{The}}) = [1 \ 1 \ 1 \ 1] \cdot [1 \ 0 \ 1 \ 0]^T = 2 \\ \text{Score}(Q_{\text{SAT}}, K_{\text{CAT}}) = 2 \\ \text{Score}(Q_{\text{SAT}}, K_{\text{SAT}}) = 4 \end{array} \right.$$

### ④ Scaling =

We take up the scores and scale down by dividing the scores by the

$$\sqrt{d_K} \Rightarrow d_K = 4 \quad \sqrt{d_K} = 2.$$

Scaling in the attention mechanism is crucial to prevent the dot product from growing too large.  $\Rightarrow$  Ensure stable gradients during Training

$d_K$  is large  $\rightarrow$

① Gradient Exploding

② Softmax Saturation  $\{\curvearrowright\}$   $\rightarrow$  Vanishing Gradient Problem.

$$Q = [2 \ 3 \ 4 \ 1] \quad K_1 = [1 \ 0 \ 1 \ 0] \quad K_2 = [0 \ 1 \ 0 \ 1]$$

Without Scaling

$$Q \cdot K_1^T = 2 \times 1 + 3 \times 0 + 4 \times 1 + 1 \times 0 = 2 + 4 = 6.$$

$$Q \cdot K_2^T = 2 \times 0 + 3 \times 1 + 4 \times 0 + 1 \times 1 = 0 + 3 + 0 + 1 = 4$$

\* Score  $[6, 4] \Rightarrow$  Scaling Not Applied

$$\text{Softmax}([6, 4]) = \left[ \frac{e^6}{e^6 + e^4}, \frac{e^4}{e^6 + e^4} \right] = \left[ \frac{e^6}{e^6(1 + e^{-2})}, \frac{e^4}{e^4(e^2 + 1)} \right]$$

① Property of Softmax  $w_Q, w_K, w_V$

$$[[\underline{10}, \underline{1}]] = [0.99, \underline{0.01}]$$

$$= \left[ \frac{1}{(1 + e^{-2})}, \frac{1}{(e^2 + 1)} \right]$$

Dot product = Large values  $\approx [0.88, 0.12]$ .

Most of the attention weight is assigned to the first key vector, very little to the second vector,

With Scaling

① Compute Scaled Dot Product

$$[6, 4] \Rightarrow \text{Scale} \Rightarrow \left[ \frac{6}{\sqrt{2}}, \frac{4}{\sqrt{2}} \right] = [3, 2]. \quad \begin{matrix} \sqrt{dk} \uparrow & \text{dimension} \uparrow \\ \text{same} & \text{variance} \uparrow \end{matrix}$$

$$\text{Softmax}([3, 2]) = \left[ \frac{e^3}{e^3 + e^2}, \frac{e^2}{e^3 + e^2} \right] = \left[ \frac{e^3}{e^3(1 + e^{-1})}, \frac{e^2}{e^2(e^1 + 1)} \right] = [0.73, 0.27] \Rightarrow \text{Attention weights}$$

(4) Here, the attention weights are more balanced compared to the unscaled case

#### Summary of Importance

**Stabilizing Training:** Scaling prevents extremely large dot products, which helps in stabilizing the gradients during backpropagation, making the training process more stable and efficient.

**Preventing Saturation:** By scaling the dot products, the softmax function produces more balanced attention weights, preventing the model from focusing too heavily on a single token and ignoring others.

**Improved Learning:** Balanced attention weights enable the model to learn better representations by considering multiple relevant tokens in the sequence, leading to better performance on tasks that require context understanding.

**Scaling ensures that the dot products are kept within a range that allows the softmax function to operate effectively, providing a more balanced distribution of attention weights and improving the overall learning process of the model.**

(4) Scaling =  $\sqrt{dk} = \sqrt{4} \Rightarrow 2$  Similarly Scaling will be done for all other tokens.

Scaled-Score  $(Q_{The}, K_{The}) = 2/2 = 1$

Scaled-Score  $(Q_{The}, K_{CAT}) = 0/2 = 0$

Scaled-Score  $(Q_{The}, K_{SAT}) = 2/2 = 1$

(5) Apply Softmax

ATTENTION WEIGHTS "The" =  $\text{Softmax}([1, 0, 1]) = [0.4223, 0.1554, 0.4223]$

ATTENTION WEIGHTS "CAT" =  $\text{Softmax}([0, 2, 2]) = [0.1554, 0.4223, 0.4223]$

ATTENTION WEIGHTS "SAT" =  $\text{Softmax}([2, 2, 4]) = [0.2119, 0.2119, 0.5762]$

(6) Weight Sum of Values

We multiply the attention weights by corresponding value vectors

For the Token The =

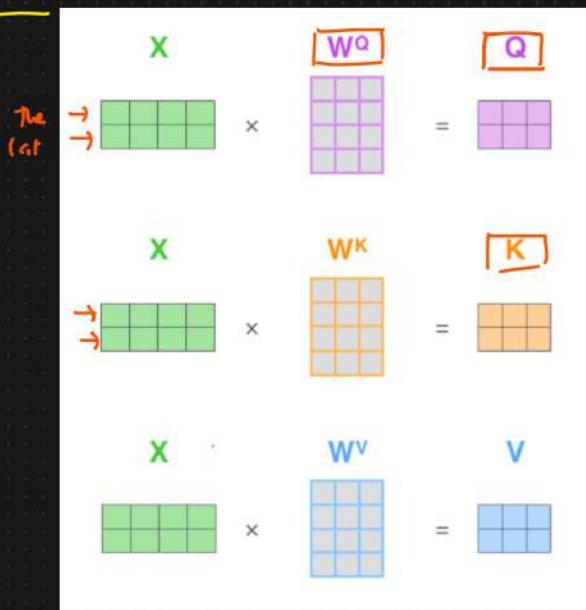
$$\begin{aligned}\text{Output}_{(\text{The})} &= 0.4223 * V_{\text{The}} + 0.1554 * V_{\text{CAT}} + 0.4223 * V_{\text{Sal.}} \\ &= 0.4223 [1 \ 0 \ 1 0] + 0.1554 [0 \ 1 0 1] + 0.4223 [1 \ 1 \ 1] \\ &= [0.4223, 0, 0.4223, 0] + [0, 0.1554, 0, 0.1554] + [0.4223, 0.4223, \\ &\quad 0.4223, 0.4223] \\ &= [1.2669, 0.9999, 1.2669, 0.9999].\end{aligned}$$

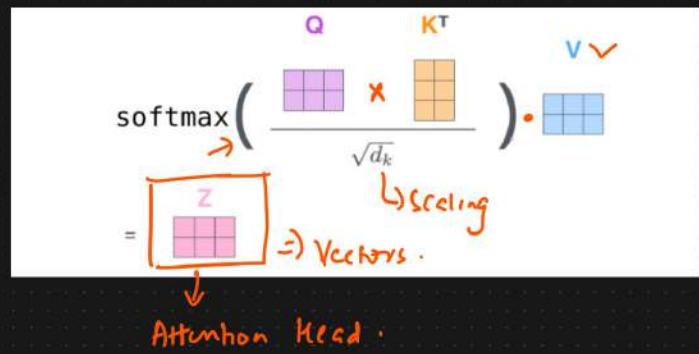
↓ Contextual  
vector

The 1 1 0 1 1 0  $\Rightarrow$  Self Attention  $\Rightarrow [1.2669, 0.9999, 1.2669, 0.9999]$ .

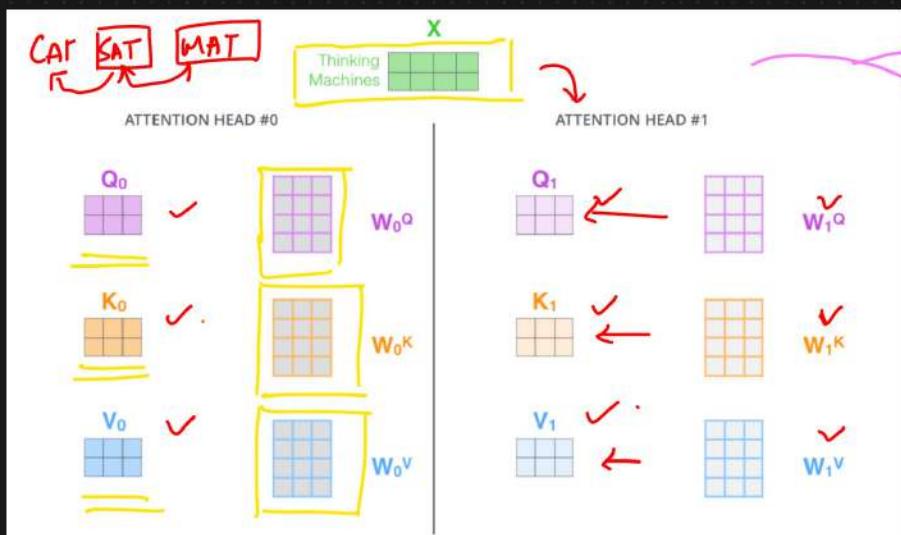
①  $\hookrightarrow Q, K, V [w^Q, w^K, w^V]$   
②  $\hookrightarrow \text{Attention Score}$   
③  $\hookrightarrow \text{Scaled}$   
④  $\hookrightarrow \text{Softmax}$   
⑤  $\hookrightarrow \text{Weighted Sum of Value } (\text{Softmax} \times V)$

## ① Multi Head Attention

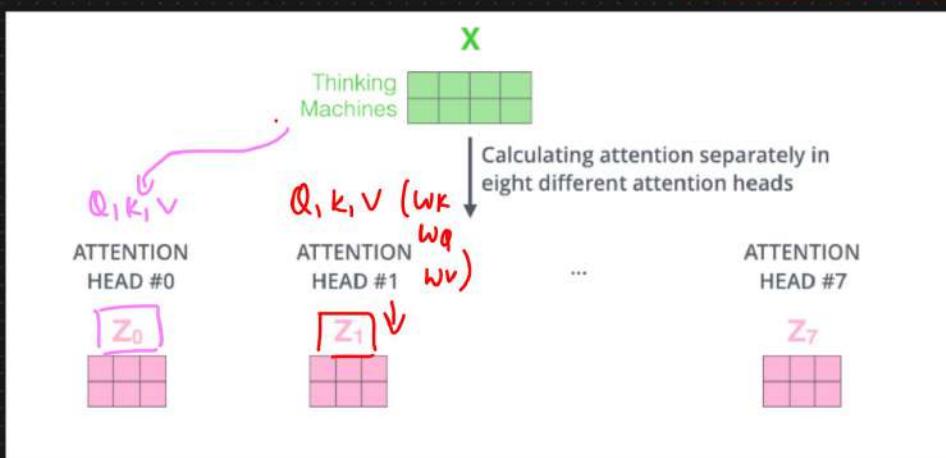




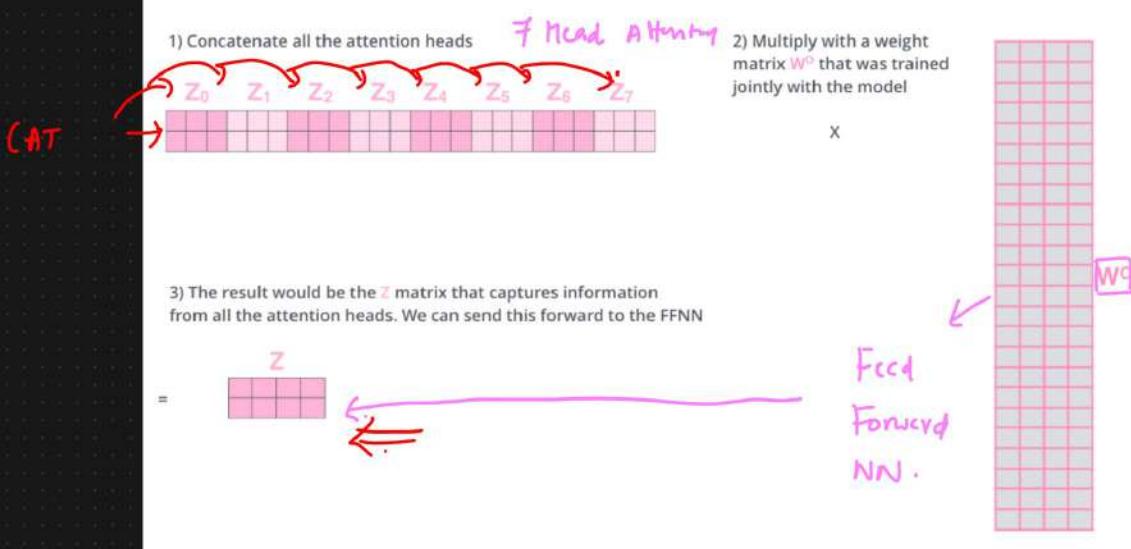
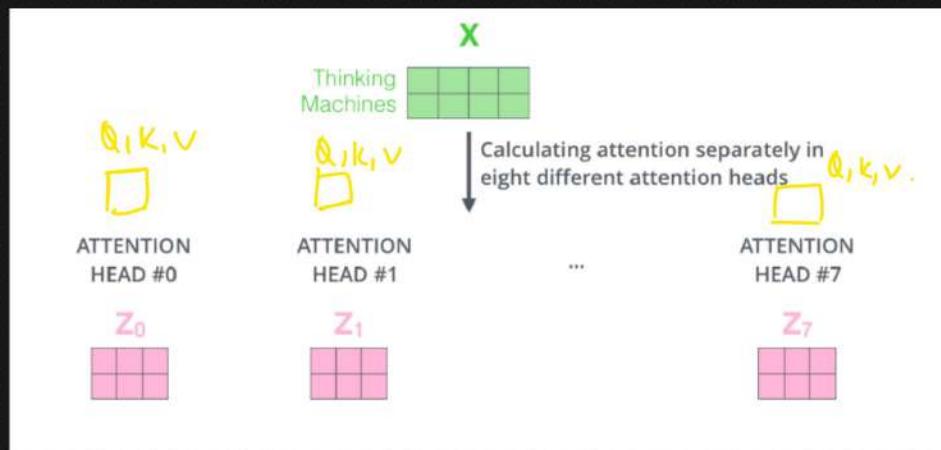
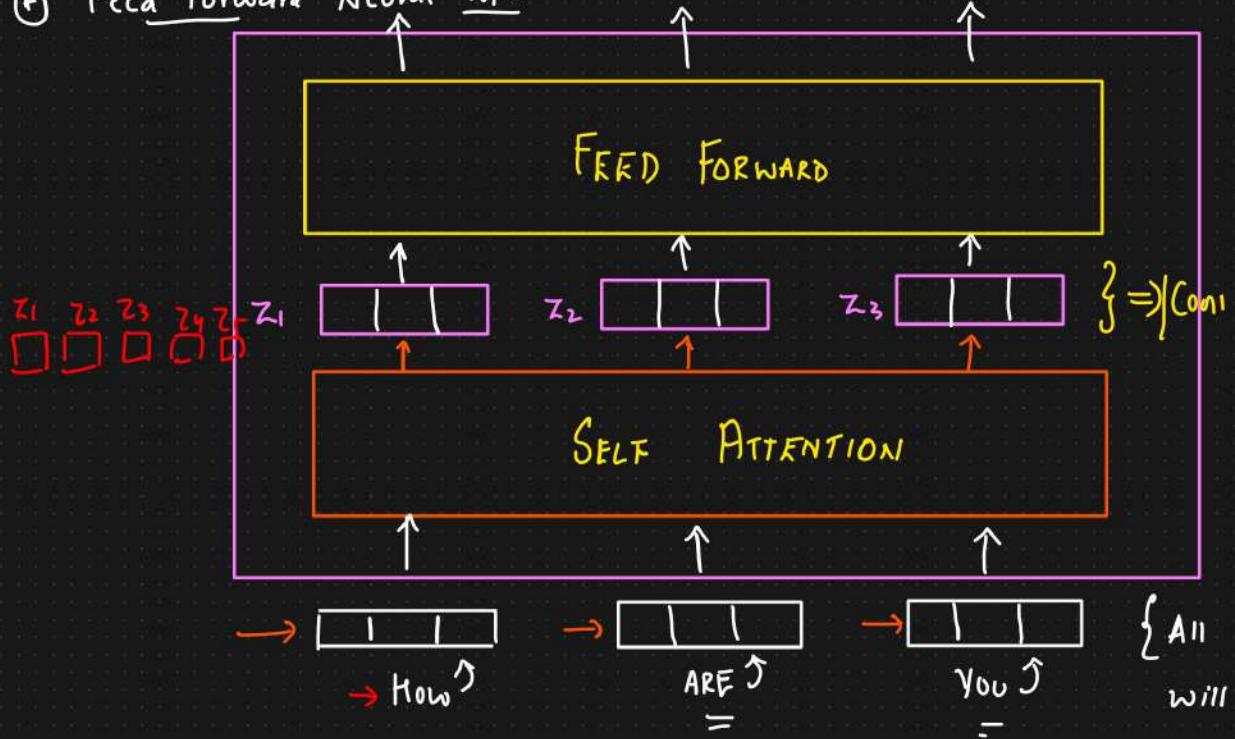
## → Self Attention with Multi Heads

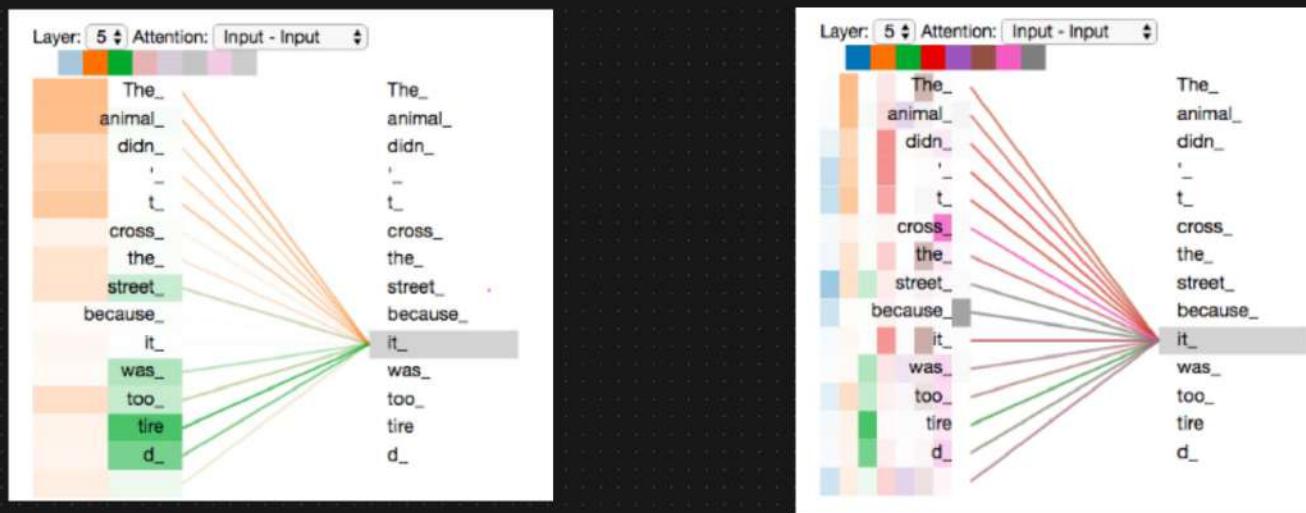
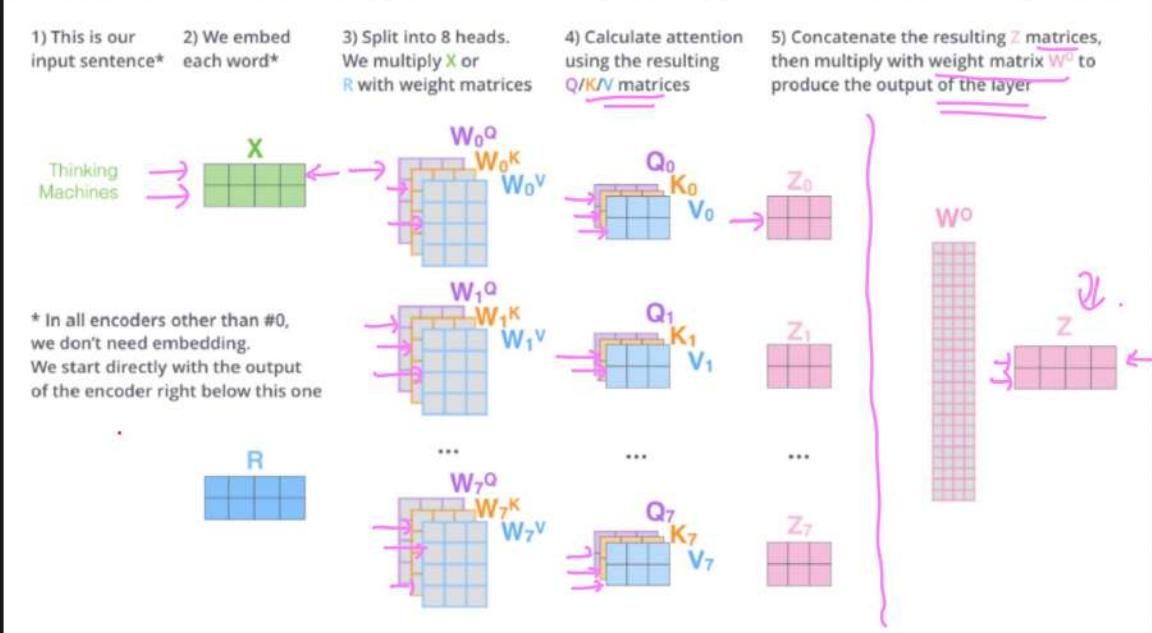


Score, Softmax  $\times V$   
 $\downarrow$   $\downarrow$   $\downarrow$   $\downarrow$   $\downarrow$   
 $\mathbf{Z}_0$   $\mathbf{Z}_1$   $\dots$   $\mathbf{Z}_n$   
 Vectors      Multi Head Attention

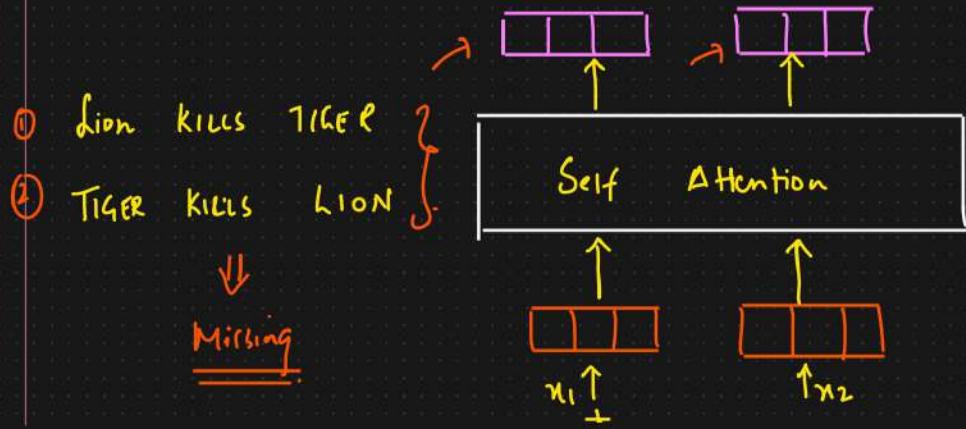


## ⑥ Feed Forward Neural Network





## ① Positional Encoding - Representing Order of Sequence



### Advantage

- ① Word Tokens it can process parallelly

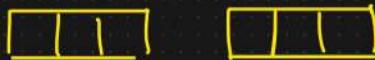


### DRAW BACK

Lack the sequential structure of the words {order}

Attention IS All

YOU NEED



Positional  
Encoded  
Vector

BOOK, JOURNAL

Novels

↳ 1 lakh  
words



Backpropagation



## Types of Position Encoding

1) Sinusoidal Position Encoding →

2) Learned Positional Encoding ⇒ Positional Encoding Are learned during Training ←

① Sinusoidal Positional Encoding : It uses Sinc And Cosine functions

of different frequencies to create positional encodings

Formula :



$$P.E(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

Where pos is the position  
 $i$  is the dimension

$$P.E(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right).$$

$d_{model}$  is the dimensionality  
of the embeddings.

Eg: The Cat Sat

$$\text{The} \rightarrow [0.1 \ 0.2 \ 0.3 \ 0.4]$$



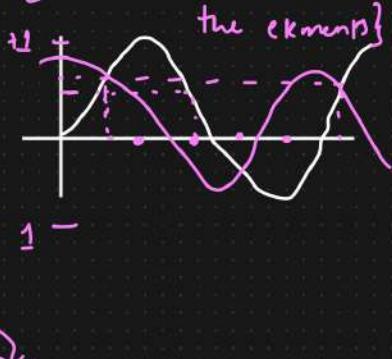
$$\text{CAT} \rightarrow [0.5 \ 0.1 \ 0.7 \ 0.8]$$



$$\text{SAT} \rightarrow [0.9 \ 1.0 \ 1.1 \ 1.2]$$



{ Miss the order of  
the elements }



$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \quad PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

For our example  $d_{model} = 4$

For position  $pos = 0$

$$PE(0,0) = \sin\left(\frac{0}{10000^0/4}\right) = \sin(0) = 0$$

$$PE(0,1) = \cos\left(\frac{0}{10000^1/4}\right) = \cos(0) = 1$$

$$PE(0,2) = \sin\left(\frac{0}{10000^2/4}\right) = \sin(0) = 0$$

$$PE(0,3) = \cos\left(\frac{0}{10000^3/4}\right) = 1$$

$$PE = [0, 1, 0, 1]$$

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

For  $pos = 1$

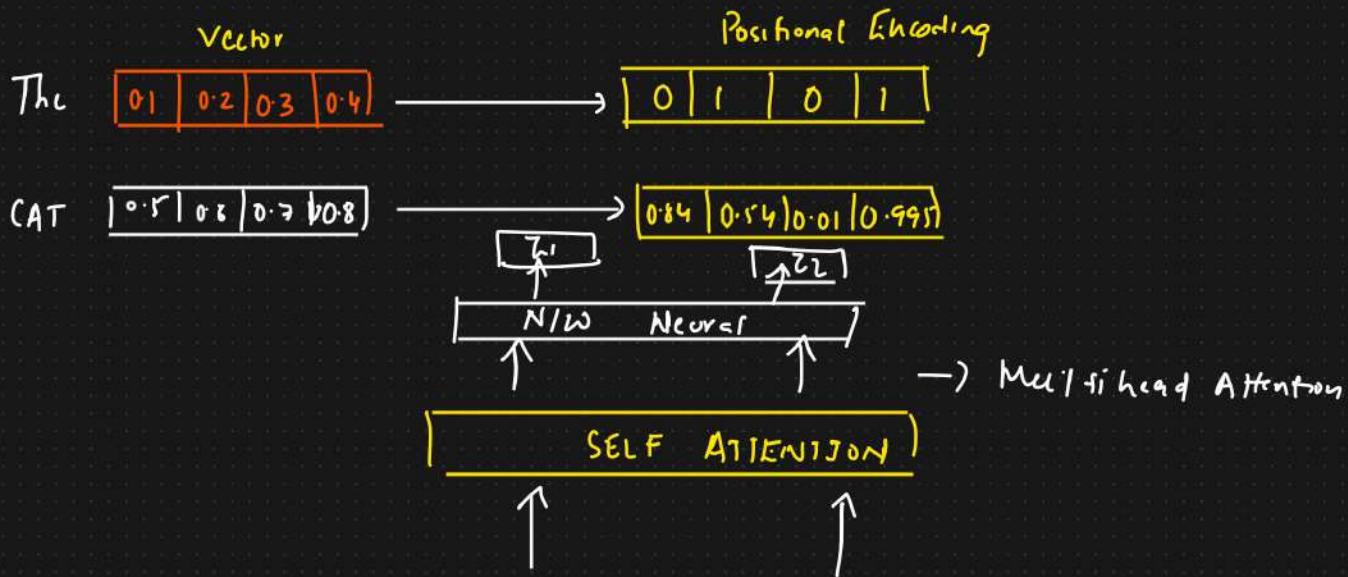
$$PE(1,0) = \sin\left(\frac{1}{10000^0/4}\right) = \sin(1) = 0.8415$$

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

$$PE(1,1) = \cos\left(\frac{1}{10000^1/4}\right) \approx 0.5403$$

$$PE(1,2) = \sin\left(\frac{1}{10000^2/4}\right) \approx 0.01$$

$$PE(1,3) = \cos \approx 0.99995$$

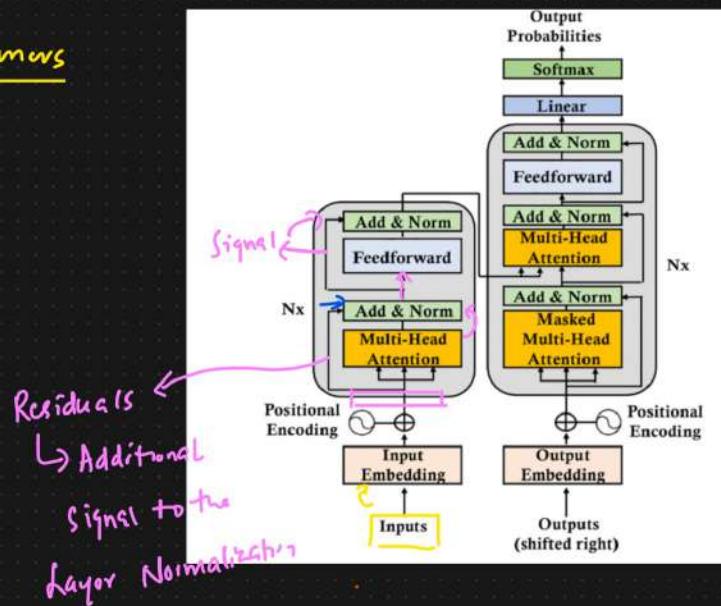


$$\begin{array}{c}
 \text{Positional} \\
 \text{Encoding} \\
 \leftarrow \boxed{0.84 \mid 0.54 \mid 0.01 \mid 0.39} \\
 + \\
 \boxed{0.5 \mid 0.6 \mid 0.7 \mid 0.8} \\
 \uparrow \\
 \text{CAT}
 \end{array}
 \quad
 \begin{array}{c}
 \boxed{0 \mid 1 \mid 0 \mid 1} \leftarrow \text{Positional Encoding} \\
 + \\
 \boxed{0.1 \mid 0.2 \mid 0.3 \mid 0.4} \\
 \uparrow \\
 \text{The}
 \end{array}$$

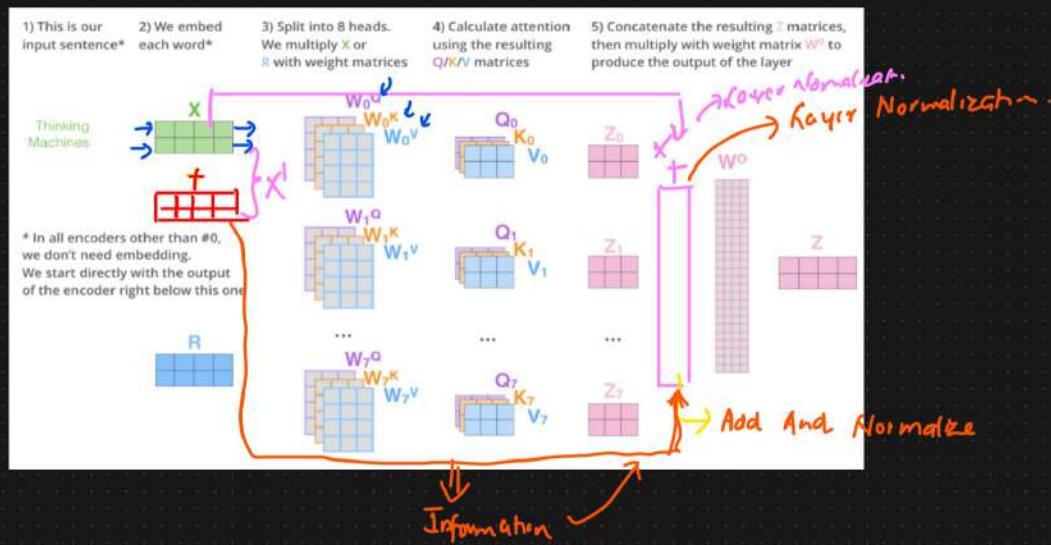
## ④ Layer Normalization In Transformers

### Transformers

### Residuals

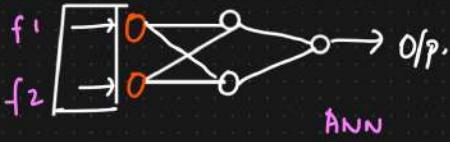


- ① Self Attention layer
  - ② Multi head Attention
  - ③ Positional Encoding
  - ④ Layer Normalization
- ADD AND Normalize**



### Normalization

→ Batch Normalization  
→ Layer Normalization



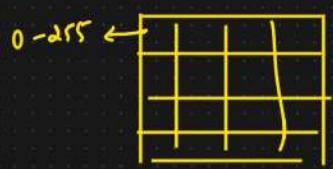
$f_1$	$f_2$	
House Size	No. of Rooms	Price
1200	2	45
1500	3	70

## Normalization : Standard Scaling

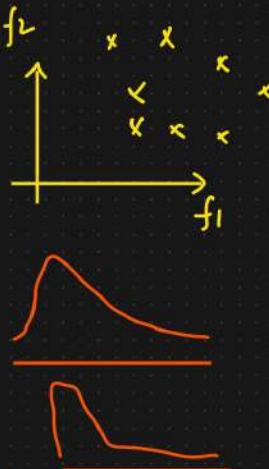
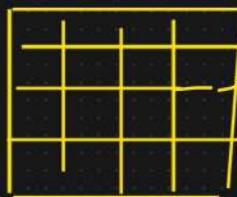
$\mu, \sigma \leftarrow | 2000 |$

$$\text{z-score} = \frac{x_i - \mu}{\sigma} \Rightarrow \{\mu=0, \sigma=1\} \quad f_1 \rightarrow f_1' \Leftarrow \mu=0, \sigma=1$$

## Deep learning : I/P Image $\Rightarrow$ Min Max Scaler



$\Rightarrow$  Min Max Scaler  $\Rightarrow$



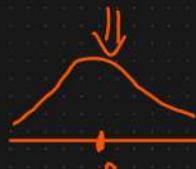
## Advantages

- ① Improved Training Stability

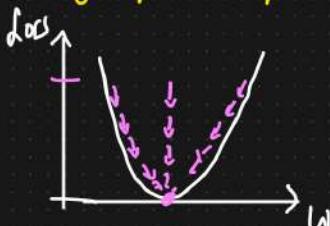


Vanishing and exploding Gradient problem

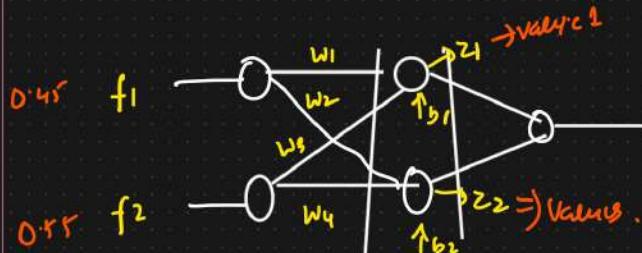
$$\begin{cases} \mu=0 \\ \sigma=1 \end{cases}$$



- ② Faster Convergence



$\Rightarrow$  Back propagation  $\Rightarrow$  Stable update.



$f_1$

$$\begin{matrix} \text{House size} \\ 0.45 \\ 0.60 \\ - \end{matrix}$$

$f_2$

$$\begin{matrix} \text{Rooms} \\ 0.55 \\ 0.20 \\ - \end{matrix}$$

Normalized (Standardized)

$$\begin{matrix} z_1 \\ z_2 \end{matrix} = \begin{matrix} - \\ - \end{matrix}$$

$$z_1 = \sigma[(0.45 \cdot w_1 + 0.55 \cdot w_3) + b_1] = \text{Value 1} \quad \text{Does this distribution}$$

$$z_2 = \sigma[(0.45 \cdot w_2 + 0.55 \cdot w_4) + b_2] = \text{Value 2} \quad \begin{matrix} \mu=0, \sigma=1 \\ \mu_1, \sigma_1 \\ \mu_2, \sigma_2 \end{matrix}$$

## Batch Normalization VS Layer Normalization

$f_1$

$f_2$

$\tau_1$

$\tau_2$

$$\begin{matrix} - \\ - \\ - \\ - \end{matrix}$$

$\mu_1, \sigma_1$

$\mu_2, \sigma_2$

$$\begin{matrix} - \\ - \\ - \end{matrix}$$

$\mu_3, \sigma_3$

$\mu_4, \sigma_4$

$$\begin{matrix} - \\ - \\ - \end{matrix}$$

$\mu_1, \sigma_1$

$\mu_2, \sigma_2$

$$\begin{matrix} - \\ - \\ - \end{matrix}$$

$\mu_3, \sigma_3$

$\mu_4, \sigma_4$

$$\begin{matrix} - \\ - \\ - \end{matrix}$$

$f_1$

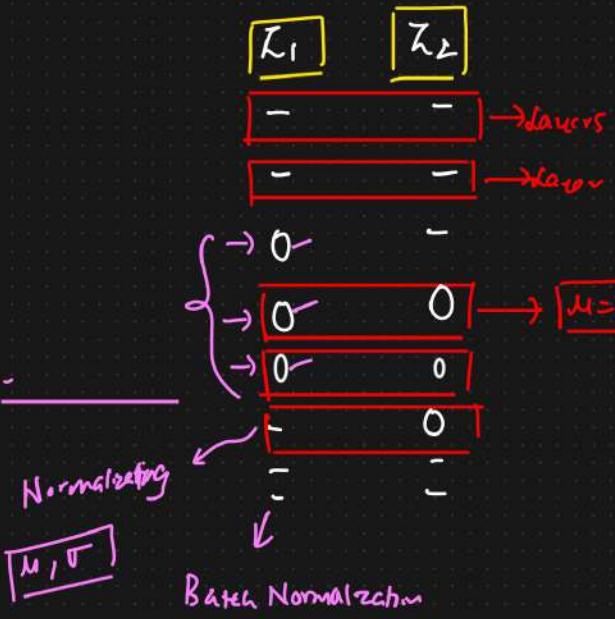
$f_2$

$$\begin{matrix} - \\ - \\ - \end{matrix}$$

Layer

Normalization

$\gamma, \beta \rightarrow$  learnable parameters



Normalization

$\Downarrow$

$\gamma, \beta$

$$z_1 = \Gamma [w_1^T x + b_1]$$

$$y = \underline{\underline{\gamma}} \left[ \frac{z_1 - \mu_1}{\sigma_1} \right] + \underline{\underline{\beta}}$$

Normalized  $\Downarrow$   
Scale And Shift parameters

i) "CAT" =  $[2.0, 4.0, 6.0, 8.0] \leftarrow \{ \text{Values} \}$

2) Parameters =  $\gamma = [1.0, 1.0, 1.0, 1.0] \rightarrow \text{Learned Scale}$   
 $\beta = [0.0, 0.0, 0.0, 0.0] \rightarrow \text{Shift}$   $\Downarrow$   $\Rightarrow$  Scale And Shift param

i) Compute the mean

$$z_{score} = \frac{x_i - \mu}{\sigma}$$

$$\mu = \frac{1}{4} (2.0 + 4.0 + 6.0 + 8.0)$$

$$= \frac{20.0}{4} = 5.0$$

ii) Compute the variance ( $\sigma^2$ )

$$\sigma^2 = \frac{1}{4} [(2.0 - 5.0)^2 + (4.0 - 5.0)^2 + (6.0 - 5.0)^2 + (8.0 - 5.0)^2] = 5.0$$

iii) Normalize the i/p

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} \quad \epsilon = 1e^{-5} \Rightarrow \text{Avoid division by 0}$$

$$\sqrt{\sigma^2 + \epsilon} = \sqrt{5.0 + 1e^{-5}} \approx \sqrt{5.00001} = 2.236$$

$$\hat{x}_1 = \frac{2.0 - 5.0}{2.236} \approx -1.34 \quad \text{Normalized vector}$$

$$\hat{x}_2 = \frac{4.0 - 5.0}{2.236} \approx -0.45 \quad \hat{x} = [-1.34, -0.45, 0.45, 1.34]$$

$$\hat{x}_3 = \frac{6.0 - 5.0}{2.236} \approx 0.45$$

$$\hat{x}_4 = \frac{8.0 - 5.0}{2.236} \approx 1.34.$$

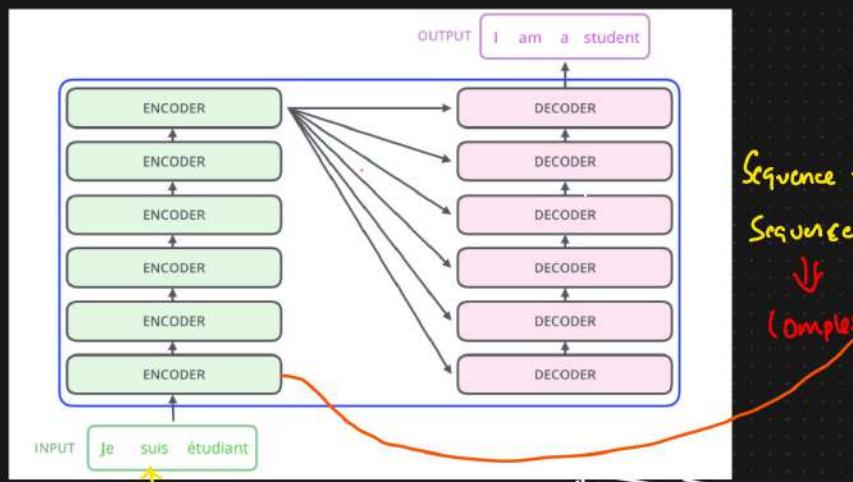
#### 4) Scale And Shift

$$y_i = r_i \hat{x}_i + \beta_i$$

$$r = [1.0, 1.0, 1.0, 1.0] \quad \beta = [0.0, 0.0, 0.0, 0.0].$$

$$y = [-1.34, -0.45, 0.45, 1.34].$$

#### ④ Encoder Architecture [Research Paper]



INPUT: Je suis étudiant  
 OUTPUT: I am a student  
 Encoder: 6 layers  
 Decoder: 6 layers  
 Sequence to Sequence ↴ (complex)

Diagram annotations:  
 - 'Feed Forward NN.' is connected to the input embeddings.  
 - '(Add and Norm)' is connected to the multi-head attention layer.  
 - 'Multi Head Attention' is connected to the input embeddings.  
 - 'Text Embeddings + Positional Encoding' is connected to the input embeddings.  
 - 'J/I/P Sequence' is connected to the input embeddings.  
 - 'Residuals' is connected to the input embeddings.  
 - 'Every word = 512.' is connected to the input embeddings.  
 - 'K = 64' is connected to the multi-head attention layer.  
 - 'V = 64' is connected to the multi-head attention layer.  
 - 'D\_h = 64' is connected to the feed forward layer.  
 - '811' is connected to the output probabilities.

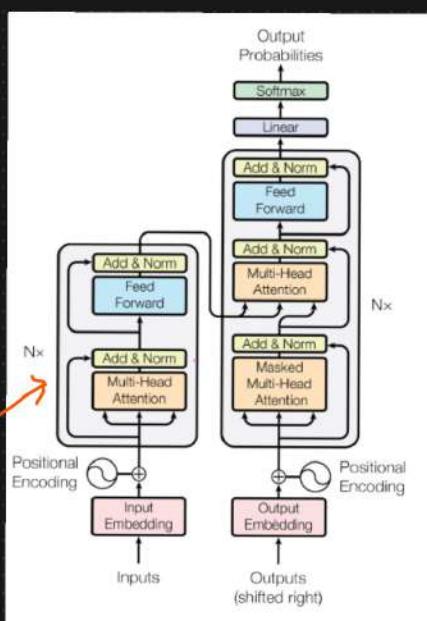
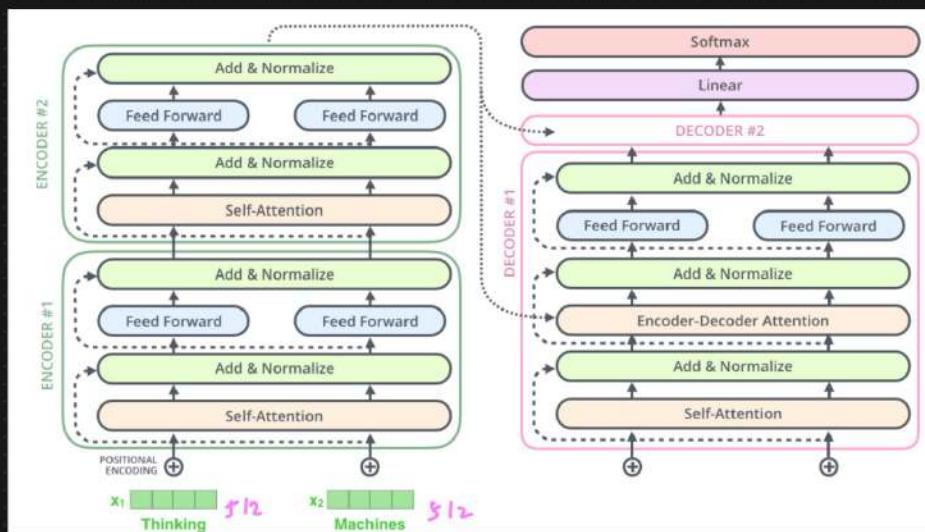
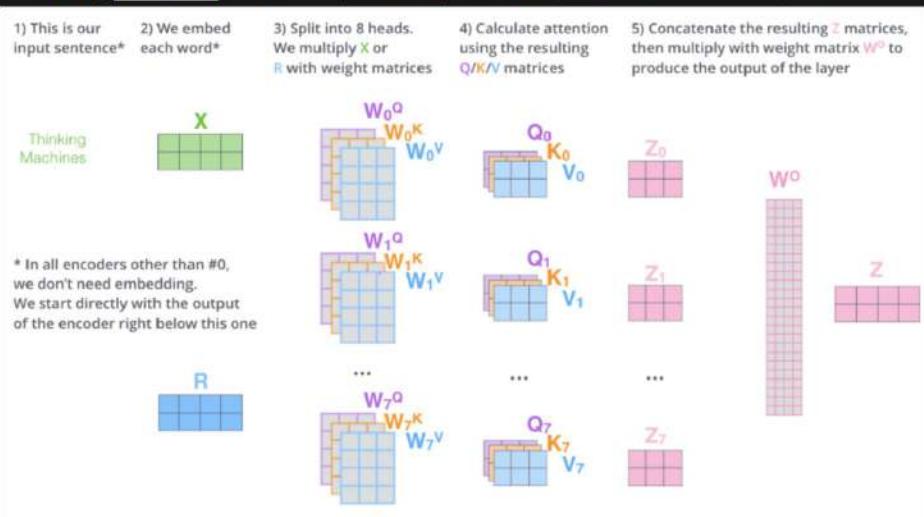


Figure 1: The Transformer - model architecture.

## Self Attention to Feed Forward Neural N/W



① Residual connection : Skip connection NN.

i) Addressing the Vanishing Gradient Problem

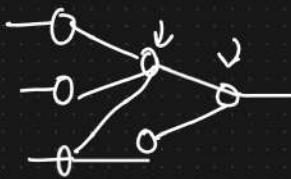
Residuals : Residual connection create a short paths for gradients to flow directly through the n/w. Gradient remains sufficiently large.

2) Improve Gradient flow

Convergence will be faster.

3) Enables Training of Deeper Networks.

## ④ Feed Forward NN



inner function problem

{Non linear functions}.

## ① Adding Non Linearity

② Processing Each position Independently.

Self Attention → (capture relationships)

FFN → Each token representation independently



AFFN =>

Transforming those representation further

and allows the model to learn

Richer Representation.

③ FFN → Decoder => Adds Depth to the Model.

Depth ↑↑ => More Learnings → DATA

## ⑤ Decoders In Transformers

3 main Components

The transformer decoder is responsible for generating the output sequence one token at a time, using the encoder's output and the previously generated tokens.

① Masked Multi Head Self Attention ✓.

② Multi Head Attention (Encoder Decoder Attention)

③ Feed Forward Neural Network.

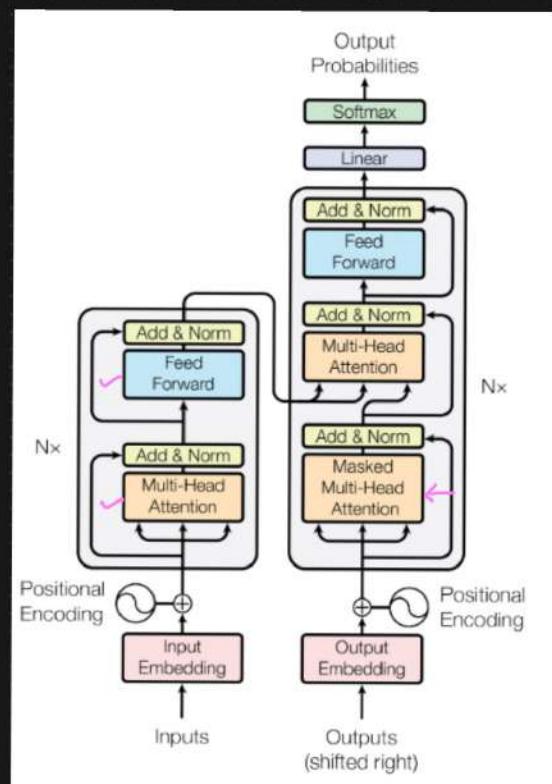
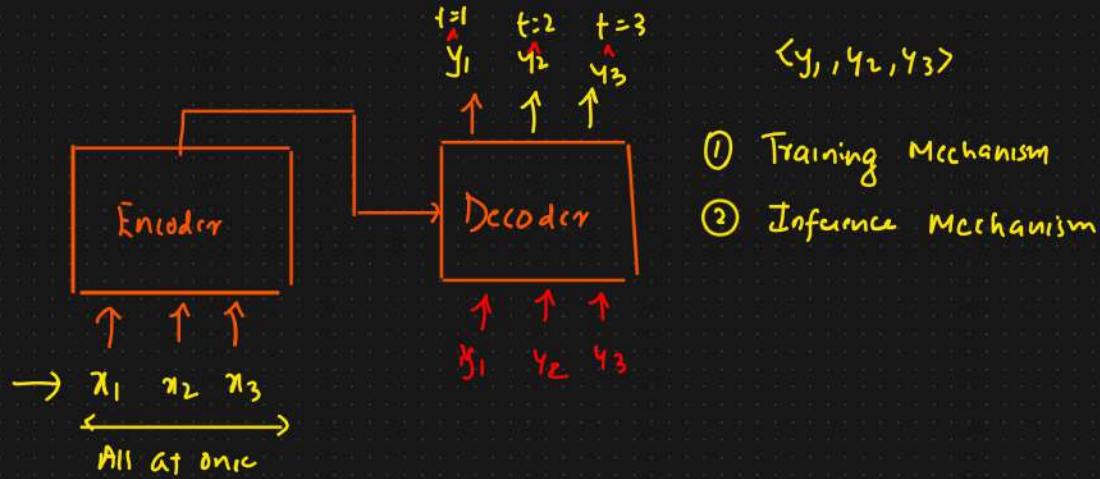


Figure 1: The Transformer - model architecture.



## ④ Masked Multi Head Self Attention

- ① I/P Embedding And Positional Embedding ✓ → Zero padding → Sequence length equal
- ② Linear Projection for Q,K,V
- ③ Scaled Dot Product Attention
- ④ Mask Application } => Try to understand the imp.
- ⑤ Multi Head Attention
- ⑥ Concatenation And Final Linear Projection
- ⑦ Residual Connection And Layer Normalization

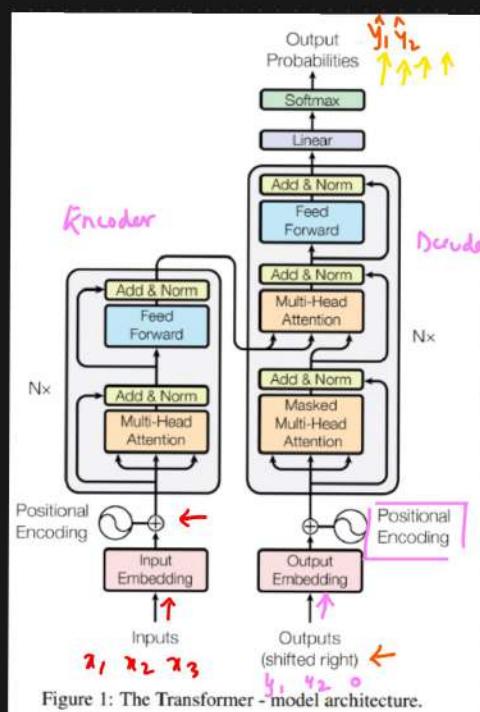


Figure 1: The Transformer -model architecture.



Masked

Multi Head  
Attention.

$$[4 \ 5 \ \frac{6}{\cancel{6}} \ 7] \quad [1 \ 2 \ \underline{3}]$$

$$[1 \ 2 \ 3 \ 0]$$

$\downarrow$   
4 dimension vector

### i) Input Embedding and Positional Encoding

Output Embedding [STEP 1]

$$\begin{bmatrix} [0.1, 0.2, 0.3, 0.4], \\ [0.5, 0.6, 0.7, 0.8], & \text{+ PE} = 0 \\ [0.9, 1.0, 1.1, 1.2], \\ [0.0, 0.0, 0.0, 0.0] \end{bmatrix}$$

Step 2 : Linear Projection for Q, K, and V

$$WQ = WK = WV = I$$

Create query (Q), key (K) and value (V) vectors

$Q = \text{Output Embedding} + W_Q = \text{Output Embedding}$

" "  $+ WK = " "$

" "  $+ WV = " "$

$$Q = K = V = \begin{bmatrix} [0.1, 0.2, 0.3, 0.4], \\ [0.5, 0.6, 0.7, 0.8], \\ [0.9, 1.0, 1.1, 1.2], \\ [0.0, 0.0, 0.0, 0.0] \end{bmatrix}, \begin{bmatrix} 0.1 \\ 0.2 \\ 0.3 \\ 0.4 \end{bmatrix} K^T \begin{bmatrix} 0.5 \\ 0.6 \\ 0.7 \\ 0.8 \end{bmatrix} \begin{bmatrix} 0.9 \\ 1.0 \\ 1.1 \\ 1.2 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

### ③ Scaled Dot Product Attention Calculation

$$\text{Score} = Q * K^T / \sqrt{dk}$$

$$= Q * K^T / 2$$

$$\begin{bmatrix} 0.1 * 0.1 + 0.2 * 0.2 + 0.3 * 0.3 + 0.4 * 0.4, \\ 0.1 * 0.5 + 0.2 * 0.6 + 0.3 * 0.7 + 0.4 * 0.8, \\ 0.1 * 0.9 + 0.2 * 1.0 + 0.3 * 1.1 + 0.4 * 1.2 \\ 0.1 * 0 + 0.2 * 0 + 0.3 * 0 + 0.4 * 0 \end{bmatrix}$$

Score :  $\left[ \begin{bmatrix} 0.3, 0.7, 1.1, 0.0 \\ 0.7, 1.9, 3.1, 0.0 \\ 1.1, 3.1, 5.1, 0.0 \\ 0.0, 0.0, 0.0, 0.0 \end{bmatrix} \right]$

## ④ Masked Application

It helps manage the structure of the sequences being processed  
And ensures the model's behavior correctly during training And Inference

### Reasons

- ① Handling Variable length Sequences with Padding MASK

#### Purpose

- ① To handle sequences of different length in batch
- ② To ensure that padding tokens, which are added to make sequences of uniform length, do not affect the model prediction

Eg:  $\text{inp} \leftarrow \text{Sequence 1} [1, 2, 3] \rightarrow \text{O/p } y_1, y_2, 0, 0, 0, 0$

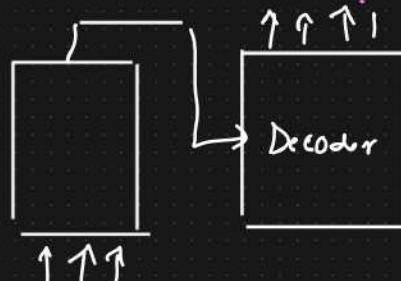
$\text{O/p} \leftarrow \text{Sequence 2} [4, 5, \boxed{0}]$  0 is the padding token  
 $\uparrow \rightarrow$  Influence the Attention Mechanism  
 $\rightarrow 100.$   $\Downarrow$   
 lead to Incorrect or biased predictions.

A padding mask  $\Rightarrow$  The tokens are ignored.

MASKING  $\rightarrow$  Padding Mask }  $\rightarrow$  Padding Mask  $\left[ \begin{array}{ccc} 1 & 1 & 1 \\ 1 & 1 & 0 \end{array} \right]$   
 $\rightarrow$  Look Ahead Mask .

## ② Look Ahead Mask → Maintain Auto Regressive Property

- ① To ensure that each position in the decoder output sequence can only attend to the previous position, but no future position.



How Anyo

- ② Sequence → Language Modelling, Transition

Eg:  $[4, 5, 0]$  →  $[1, 1, 0]$  → 1D MASK.

Token 1 attends  
Attention ← to token 1, 2  
Mechanism 1, 2

$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Convert 1D to 2D mask  
For each token in the sequence,  
the mask should indicate  
which tokens it can attend  
to.

## \* Look Ahead Mask → Decoder Output

$$\left[ \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \right]$$

## A) Combine Padding And Looking Ahead Mask

Element wise multiplication of 2 mask

Comming Mask =  $\begin{bmatrix} [1, 0, 0] \\ [1, 1, 0] \\ [0, 0, 0] \end{bmatrix}$

④ MASK

Score :  $\begin{bmatrix} [0.3, 0.7, 1.1, 0.0] \\ [0.7, 1.9, 3.1, 0.0] \\ [1.1, 3.1, 5.1, 0.0] \\ [0.0, 0.0, 0.0, 0.0] \end{bmatrix}$

Look Ahead Mask

$\begin{bmatrix} [1, 0, 0, 0] \\ [1, 1, 0, 0] \\ [1, 1, 1, 0] \\ [1, 1, 1, 1] \end{bmatrix}$

Padding Masking [extended to 2D Format]

$$\begin{bmatrix} [1, 1, 1, 0] \\ [1, 1, 1, 0] \\ [1, 1, 1, 0] \\ [0, 0, 0, 0] \end{bmatrix}$$

Combined Mask : Look Ahead Mask + Padding Mask

$$\begin{bmatrix} [1*1, 0*1, 0*1, 0*0] \\ [1*1, 1*1, 0*1, 0*0] \\ [1*1, 1*1, 1*1, 0*0] \\ [1*1, 1*1, 1*1, 1*0] \end{bmatrix} = \begin{bmatrix} [1, 0, 0, 0] \\ [1, 1, 0, 0] \\ [1, 1, 1, 0] \\ [1, 1, 1, 0] \end{bmatrix}, \Rightarrow \begin{bmatrix} [1, -\infty, -\infty, -\infty] \\ [1, 1, -\infty, -\infty] \\ [1, 1, 1, -\infty] \\ [1, 1, 1, -\infty] \end{bmatrix}$$

Masked Score

$$\begin{bmatrix} [0.3, -\infty, -\infty, -\infty] \\ [0.7, 1.9, -\infty, -\infty] \\ [1.1, 3.1, 5.1, -\infty] \\ [0.0, 0.0, 0.0, -\infty] \end{bmatrix}$$

Zero out the influence when the softmax is applied.

Attention weight.



## Softmax

$$\begin{aligned}\text{Softmax Score} &= \text{Softmax}(\text{Masked Scores}) \\ &= \left[ \left[ 1.0, 0.0, 0.0, 0.0 \right], \right. \\ &\quad \left[ 0.3, 0.7, 0.0, 0.0 \right], \\ &\quad \left[ 0.1, 0.3, 0.6, 0.0 \right], \\ &\quad \left. \left[ 1.0, 0.0, 0.0, 0.0 \right] \right]\end{aligned}$$

## Weight Sum of Value

Attention O/p = Softmax Scores \* V.

## Masking

Masking in the transformer architecture is essential for several reasons. It helps manage the structure of the sequences being processed and ensures the model behaves correctly during training and inference. Here are the key reasons for using masking:

### 1. Handling Variable-Length Sequences with Padding Mask

#### Purpose

To handle sequences of different lengths in a batch.

To ensure that padding tokens, which are added to make sequences of uniform length, do not affect the model's predictions.

### 2. Maintaining Autoregressive Property with Look-Ahead Mask

#### Purpose

To ensure that each position in the decoder output sequence can only attend to previous positions and itself, but not future positions.

This is crucial for sequence generation tasks like language modeling and translation, where the model should not have access to future tokens when predicting the current token.

# ④ Encoder Decoder Multi Head Attention

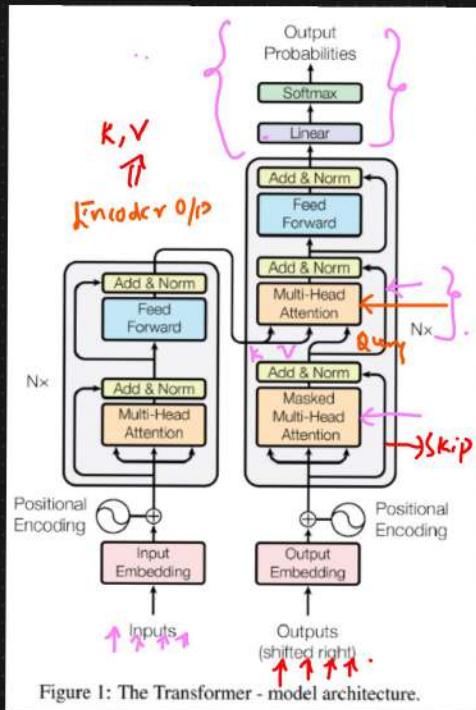


Figure 1: The Transformer - model architecture.

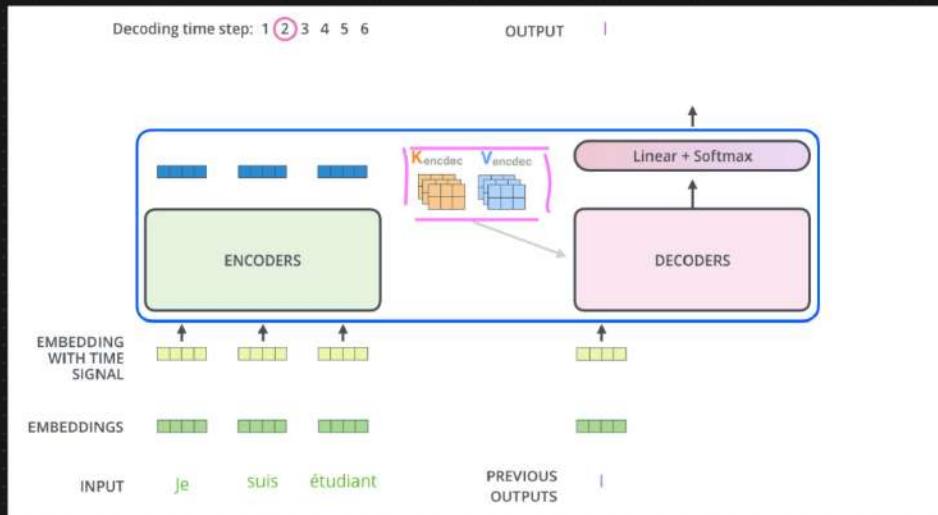
- ① Encoder O/p  $\rightarrow$  Set of Attention vector  $K$  &  $V$
- ② Masked Multihead  $\rightarrow$  Attention vector  $Q$  {Query Vector}

These are to be used by each decoder in its

"Encoder-decoder" Attention layer  
 $\downarrow$

**Skip Connection** Helps the Decoder to focus on appropriate places in the i/p Sequence

Dataflow  
 i/p                      o/p  
 $\langle x_1, x_2, x_3 \rangle$      $\langle y_1, y_2, y_3 \rangle$ .



## ④ The Final Linear And Softmax Layer { Vectors → o/p Word } { BLOG } ⇒ Transformers

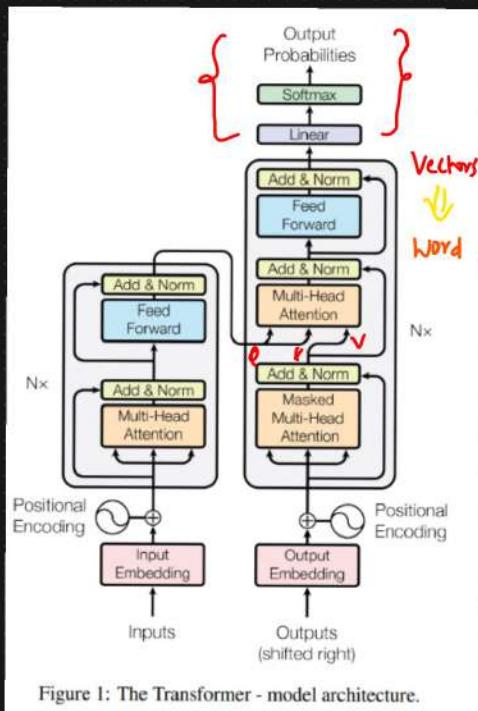
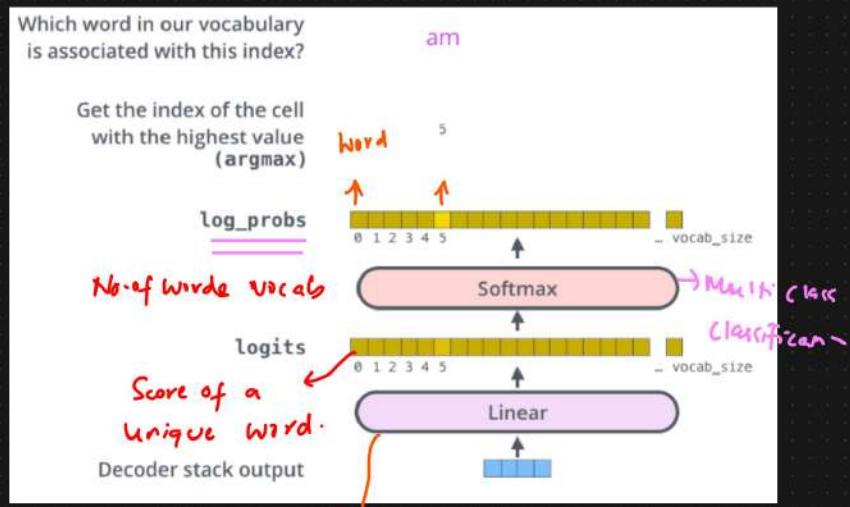


Figure 1: The Transformer - model architecture.



linear ⇒ The linear layer is a simple fully connected neural net that projects the vector produced by the stack of Decoder ⇒ logits vector ⇒

Model ⇒ 10,000 ⇒ Vocabulary ⇒ logits vector = 10000 cells wide

② Softmax layer turns those scores into probabilities (all add up to 1.0).

The cell with the highest probability is chosen, and the word associated with it is produced as the o/p. ⇒ time stamp.

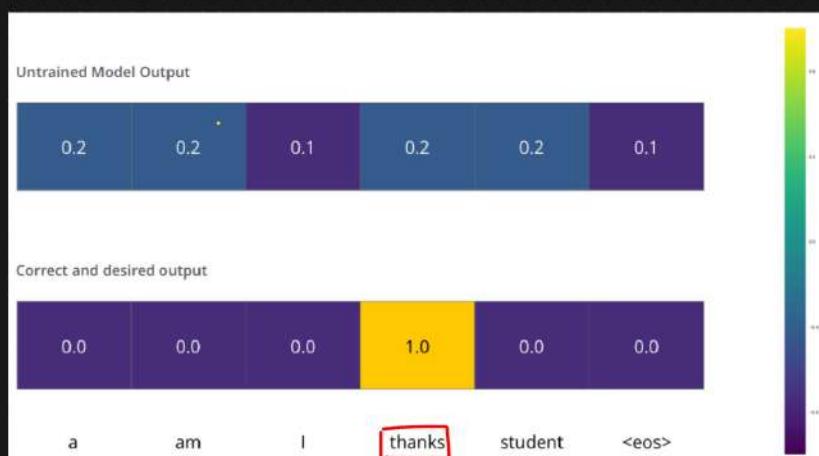
## Recap of Training

Output Vocabulary						
WORD	a	am	I	thanks	student	<eos>
INDEX	0	1	2	3	4	5

Output Vocabulary						
WORD	a	am	I	thanks	student	<eos>
INDEX	0	1	2	3	4	5
One-hot encoding of the word "am"						
0.0	1.0	0.0	0.0	0.0	0.0	0.0

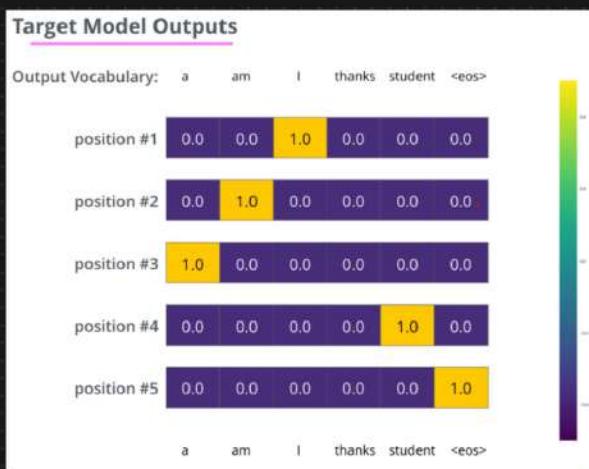
Merci → Thanks

I am a student <eos>



⇒ Loss function ↓.

Back Propagation



yp      D/p  
I am a student  
↓      ↓      ↓      ↓      ↓  
OHE    OHE    OHE    OHE    OHE

6      6  
a, am, i, student  
Loss

