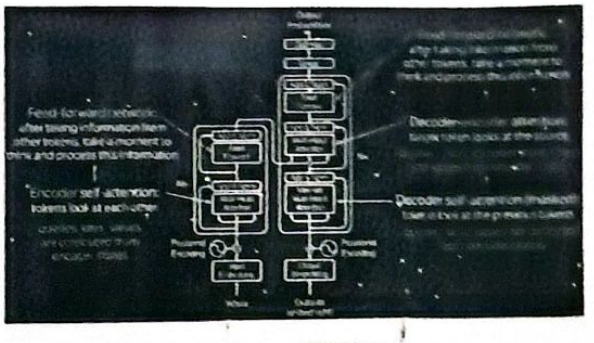
TRANSFORMERS

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 → Q, K, V
   - ④ Positional Encoding
   - ⑤ Multi Head ATTENTION
   - ⑥ Combining the WORKing of Transformers

Architecture



Generative AI → LLM, Multimodel

BERT, GPT ←

↓

OpenAI → ChatGPT

↓

GPT-4o

① What And Why → 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. ⇒ Seq2Seq Task

Eg: Language Translation → Google Translation

English → French

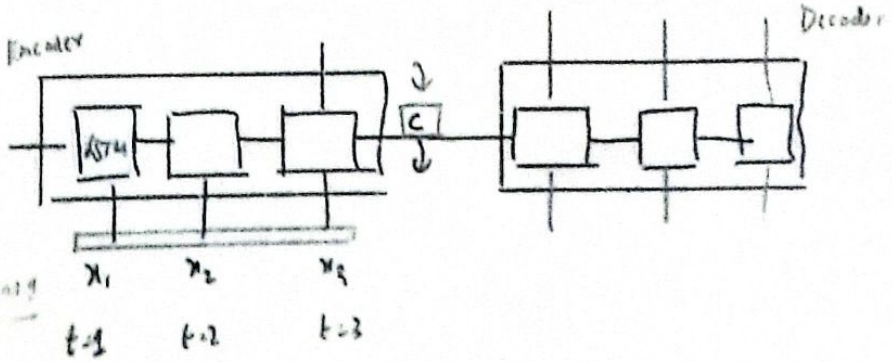
i/p ⇒ Many → o/p: Many. {Length of the sentence}

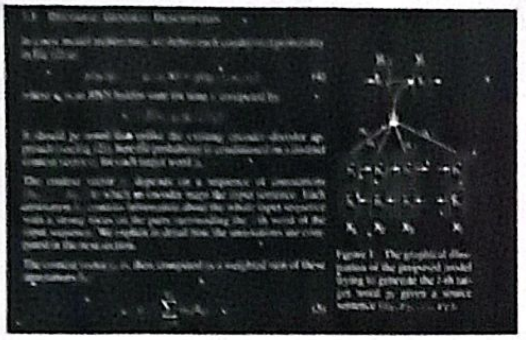
Encoder - Decoder

Sentence Length ↑↑

Bleau Score ↓↓

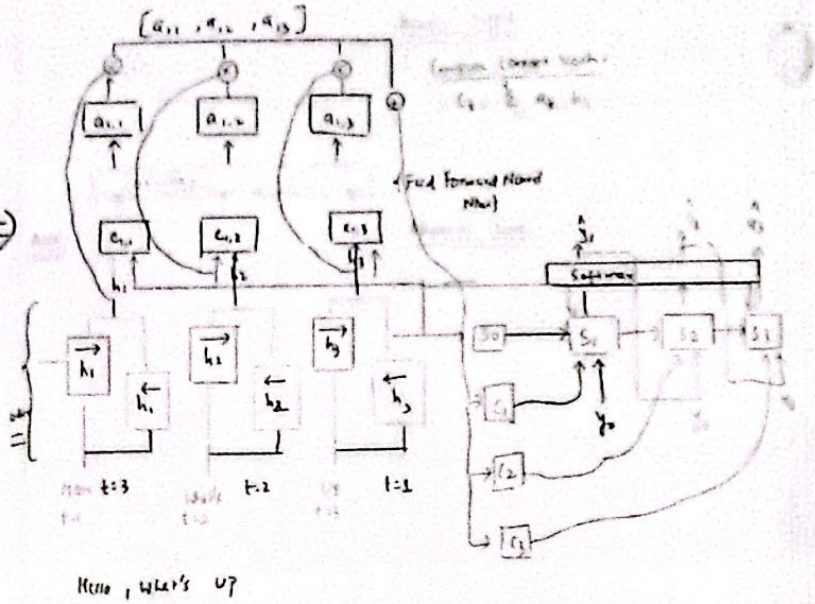
Length Sentence ↑↑





Additional Context ⟶ Decoder

long Sentence Accuracy ↑↑



Hello, What's up?

Attention Mechanism ↓ (Problem)

① Parallely ● 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 parallely sent to encoder.

⇩

Positional Encoding

State of the Art

✓ Transformer ↑↑ DATASET ⟶ Amazing SOTA ← NLP

✓ Transfer Learning ⟶ MultiModal Task ⟶ NLP + Image ←

● Transformers ÷ AI Space ⟶ SOTA Model ⟶

↓ ↓

⟶ Transfer Learning ⟶ SOTA Models ⟶ DALL E } LLM's, Generative AI

↑

Train Huge Data

① Contextual Embedding → Self Attention

Contextual Vector

Eg: My name is KRISH And I play CRICKET

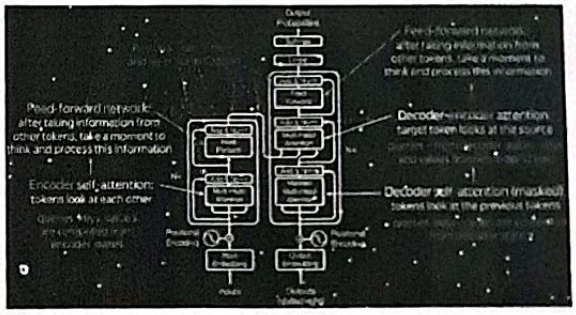
Relational Vector

Word2Vec

Embedding Layer

Contextual vector Embedding

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



I/p → The TRANSFORMER → O/p.

multiple encoder stacking step

6 Encoder → 6 is not used in Research Papers

Comment yes-no?

Encoder Decoder

Encoder

Decoder

Encoder

Inside each Encoder

How are you?

Inside each Decoder

O/p Encoder Decoder Architecture

6 Decoders

Feed Forward Neural N/W

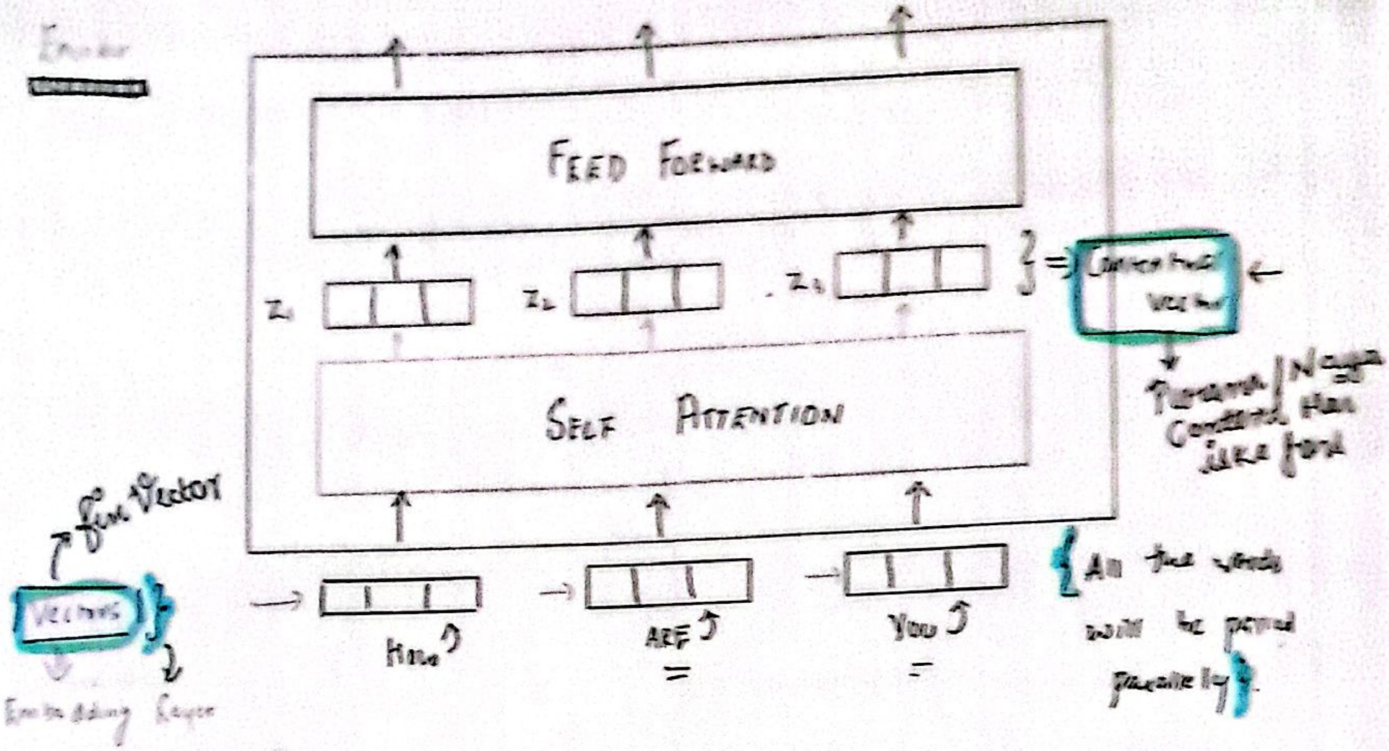
Self - Attention

LSTM RNN

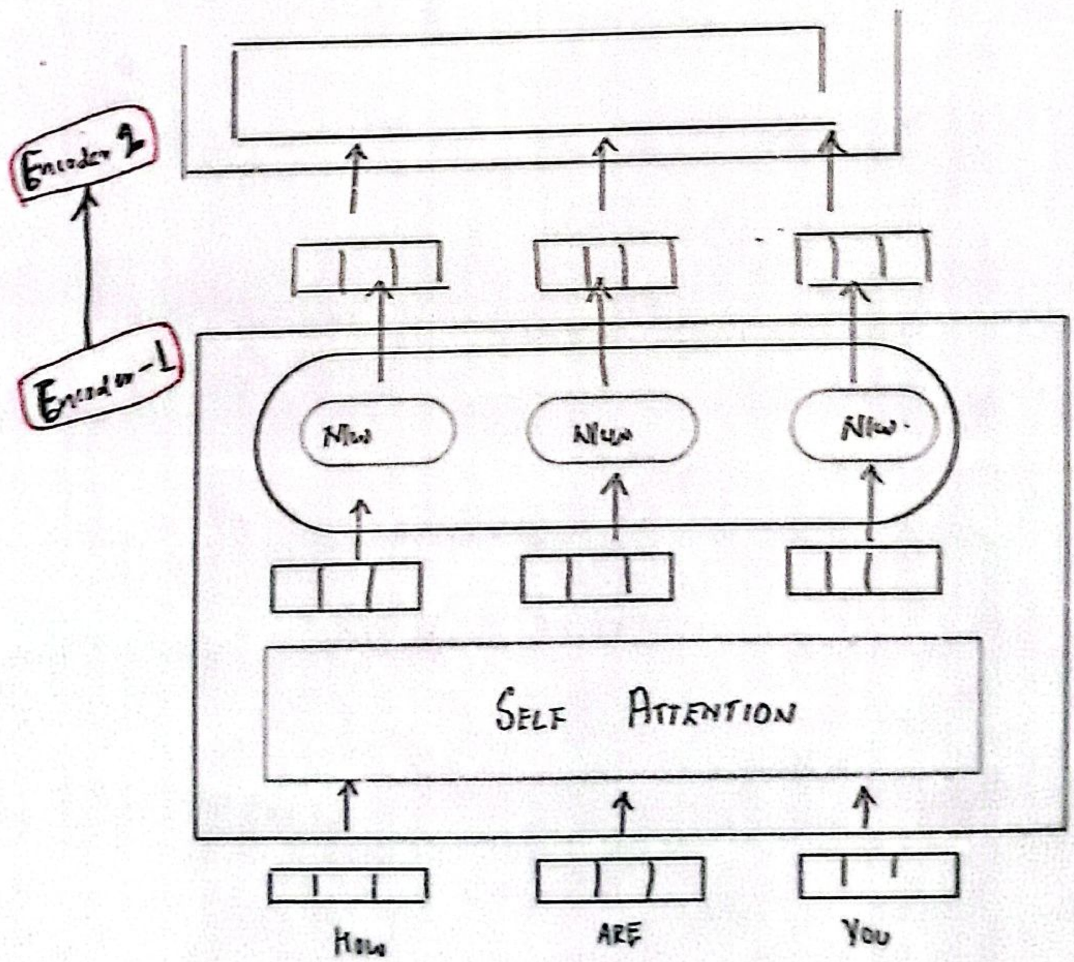
Feed Forward

Encoder - Decoder Attention

Self - Attention

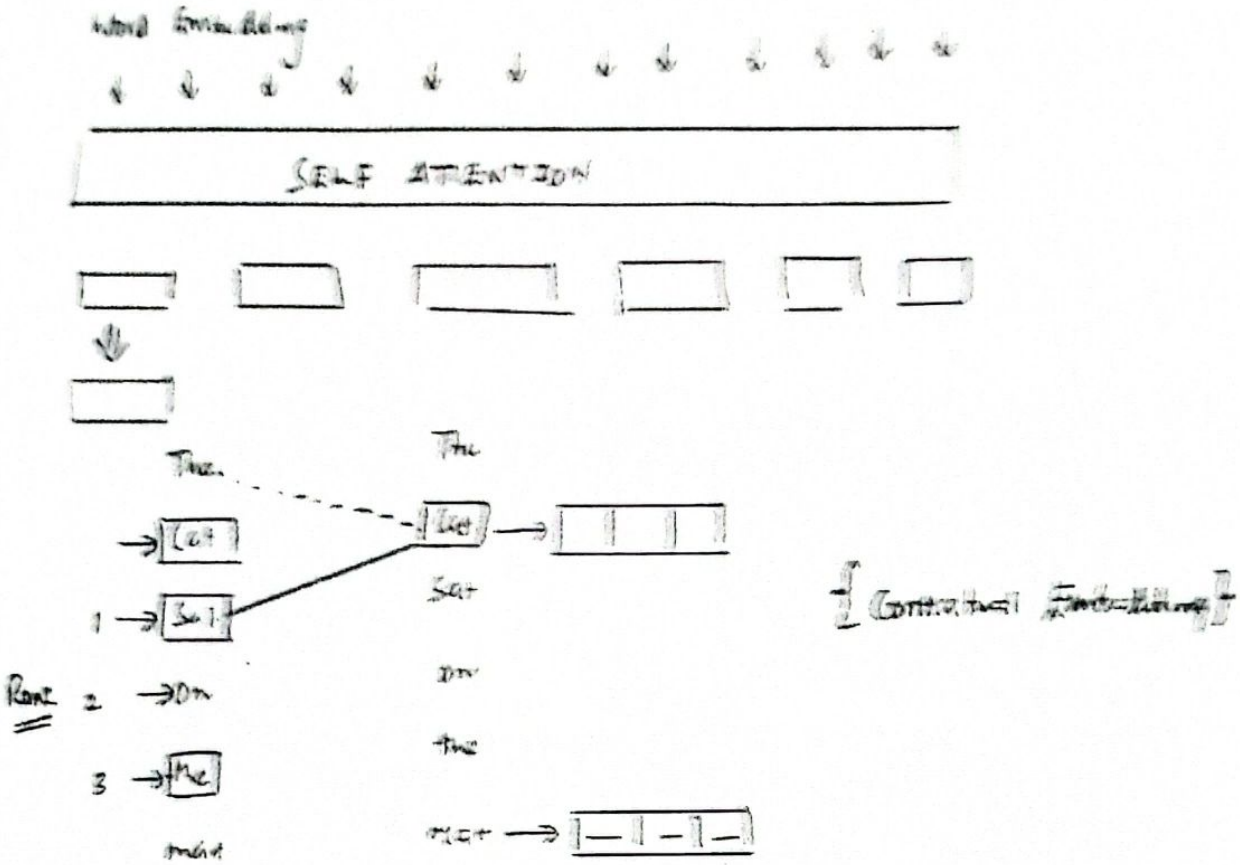


FEED FORWARD
SELF ATTENTION
Vectors
Embedding Layer
How
ARE
You
Contextual vector
All the words will be passed parallely



Encoder 2
Encoder-1
SELF ATTENTION
How
ARE
You

Self Attention At a Higher Level



Self Attention In Details

① To create 3 vectors from each of the encoder I/P - Query vector, key vector, value vector ⇒ Contextual Embedding

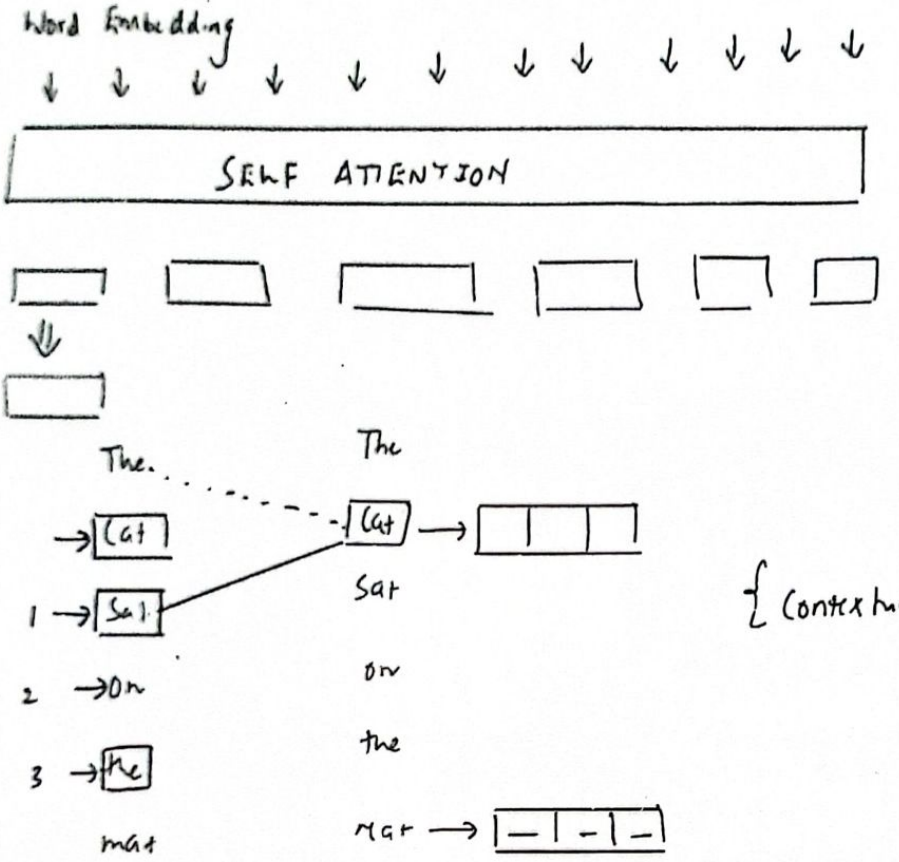
Topics

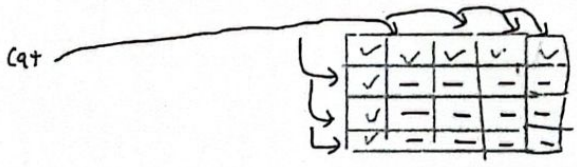
e.g. "IT" ⇒ Search keywords ⇒ [Query]

Query → [Key] → Tags, Description, Title

② Second step in calculating self attention is to calculate the dot

Self Attention At a Higher Level





Self Attention In Detail

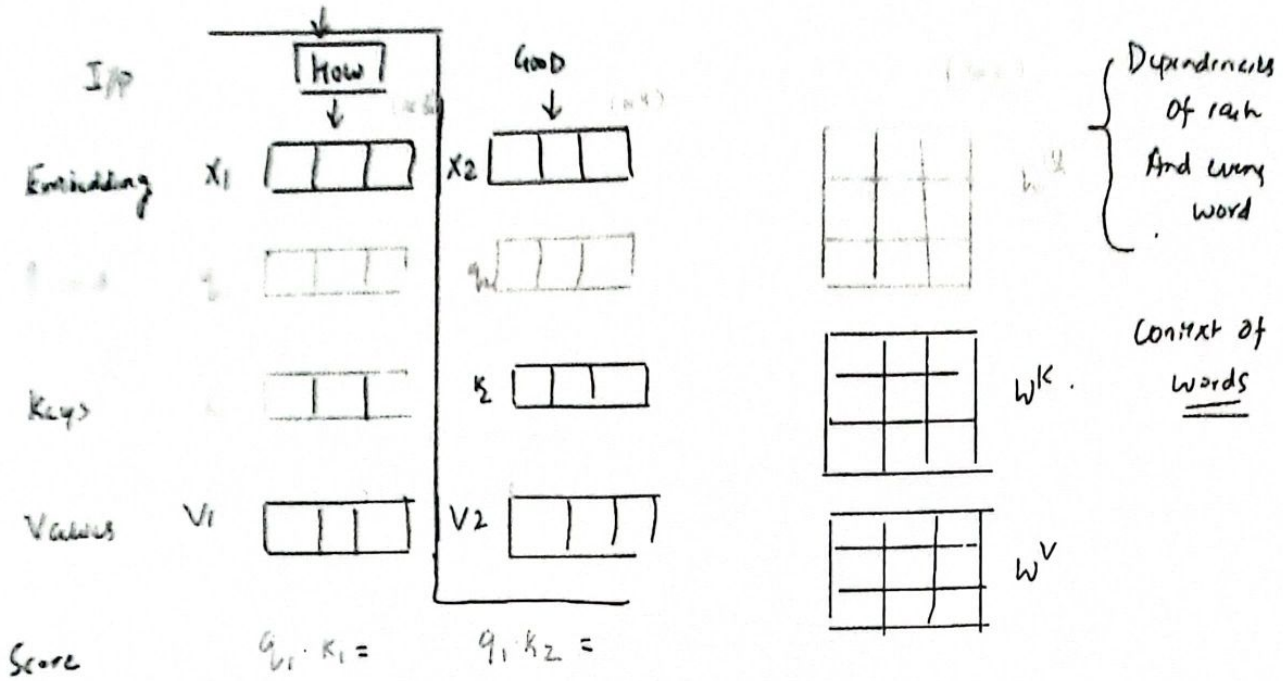
① To create 3 vectors from each of the encoder i/p · Query vector, Key vector, value vector. ⇒ Contextual Embedding

Eg. YT ⇒ Search keywords (Topic's) ⇒ {Query}.

Query ⟶ {Key} → Tags, Description, Title ⇒ Value ⇒ O/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



④ How much Focus to place on other part of the i/p Sentence as we encode a word at certain position

Divide $\sqrt{dk}$ — —

Dimension=3 Dimension ↑↑↑ 512

o/p ↑↑↑

↓

More stable gradients

Variance / $\sqrt{dk}$ ≃ Variance ↑↑↑ ⇒ / $\sqrt{dk}$

Softmax

[0-1] [0-1].

0.88 + 0.12 = 1

Softmax ⇒ The Softmax score determines how much each word will be expressed at this position

X X X

Value Vectors V_1 V_2 . [1 | 0.4 | 1]

0.88 [1 | 1 | 1]

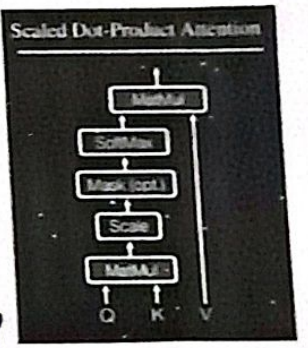
[0.88 | 0.88 | 0.88] + [0.12 | 0.6 | 0.12]

↓

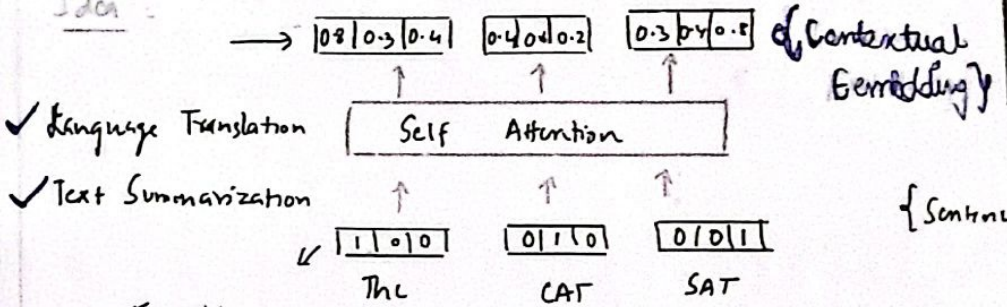
Contextual → Z_1 Z_2 →

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 :



✓ Language Translation

✓ Text Summarization

{Sentence, Dataset}.

Embedding Layer ⟶ Fixed Vector

1) Inputs :	Queries (Q), Keys (K), And Values (V)
Model ⟶	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:

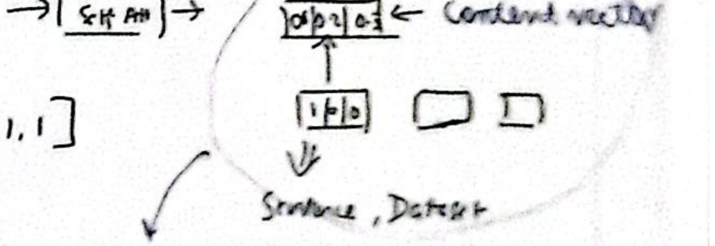
Information Aggregation: Values contain the data that will be weighted by the attention scores. The weighted sum of values forms the output of the self-attention mechanism, which is then passed on to the next layers in the network.

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.

Input Sequence = ["The", "CAT", "SAT"]

Let's assume Embedding size = 4

Q, K, V => 4



① Token Embedding

E_{The} = [1 0 1 0]

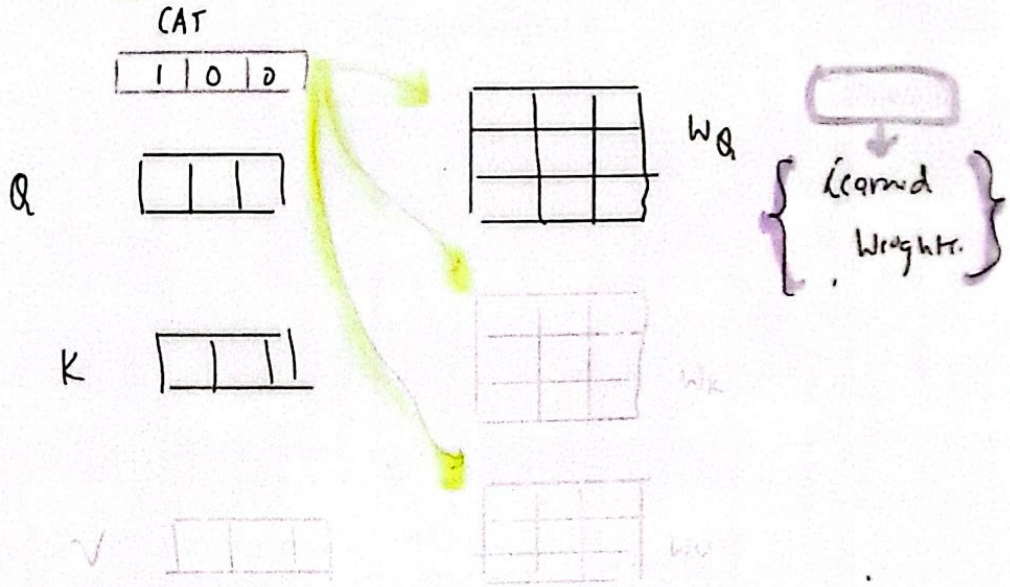
E_{CAT} = [0 1 0 1]

E_{Sat} = [1, 1, 1, 1]

Listen, we have vectors jisko context vector banana hai, for that with diff-diff dataset & sentence the context vector will change because of their mutual connections, so to solve this we came up with (Self Attention) & Q, K, V to generate this context vector

② Linear Transformation

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



Lets consider

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

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

$$K_{The} = [1\ 0\ 1\ 0] \cdot$$

$$V_{The} = [1\ 0\ 1\ 0] \cdot$$

① $Q_{The} = K_{The} = V_{The} = [1\ 0\ 1\ 0]$

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

③ $Q_{SAT} = K_{SAT} = V_{SAT} = [1, 1, 1, 1]$

Why we do
→ To know how much attention to give to each token relative to the current token

③ Compute Attention Scores

The

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

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

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

For the token CAT

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

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

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

For the Token SAT

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

$\text{Score}(Q_{SAT}, K_{CAT}) = 2$

$\text{Score}(Q_{SAT}, K_{SAT}) = 4$

④ Scaling ÷

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

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

↳ dimension of key vector

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

↳ Here dot product is B/w query & key vector.

If d_k is large →

1. Gradient Exploding
2. Softmax Saturation

other token ⇒ weights ≈ 0

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

Without Scaling

$Q \cdot K_1^T = 2\times1 + 3\times0 + 4\times1 + 1\times0 = 2 + 4 = 6.$

$Q \cdot K_2^T = 2\times0 + 3\times1 + 4\times0 + 1\times1 = 0 + 3 + 0 + 1 = 4$

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

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

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

Property of Softmax

$([10, 1]) = [0.99, 0.01]$ $\approx [0.88, 0.12]$.

Jab aapan inn value ko use krenge during BP propagation the weight will not update Bcoz of small value

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

With Scaling

$\sqrt{d_k} = \sqrt{4} = 2$ $\Rightarrow$ Variance

Via Research variance of dot product increase in same proportion as Dimension 1 es 2, 3

dimension ↑ Variance ↑↑

$\frac{\text{same}}{\sqrt{d_k}\uparrow}$

① Compute Scaled Dot Product

$[6, 4] \Rightarrow \text{Scale} \Rightarrow \left[\frac{6}{2}, \frac{4}{2}\right] = [3, 2]$

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

$= [0.73, 0.27]$ $\Rightarrow$ 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 $= \sqrt{d_k} = \sqrt{4} \Rightarrow 2$

Similarly scaling will be done for all other Tokens

$\text{Scaled\_Score}(Q_{The}, K_{The}) = 2/2 = 1$

$\text{Scaled\_Score}(Q_{The}, K_{CAT}) = 0/2 = 0$

$\text{Scaled\_score}(Q_{The}, K_{SAT}) = 2/2 = 1$

5 Apply Softmax

Contextual embedding vector

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

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

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

6 Weight Sum of Values

We multiply the attention weights by corresponding value vectors

For the Token The =

$$Output_{(The)} = 0.4223 * V_{The} + 0.1554 * V_{CAT} + 0.4223 * V_{Sat}$$

$$= 0.4223 \begin{bmatrix} 1 & 0 & 1 & 0 \end{bmatrix} + 0.1554 * \begin{bmatrix} 0 & 1 & 0 & 1 \end{bmatrix} + 0.4223 * \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}$$

$$= [0.4223, 0, 0.4223, 0] + [0, 0.1554, 0, 0.1554] + [0.4223, 0.4223, 0.4223, 0.4223]$$

$$= [1.2669, 0.9999, 1.2669, 0.9999].$$

The → [1 | 0 | 1 | 0] ⇒ Self Attention ⇒ [1.2669, 0.9999, 1.2669, 0.9999]. (Contextual vector)

① ↳ Q, K, V $[W^Q, W^K, W^V]$ → these are trained W

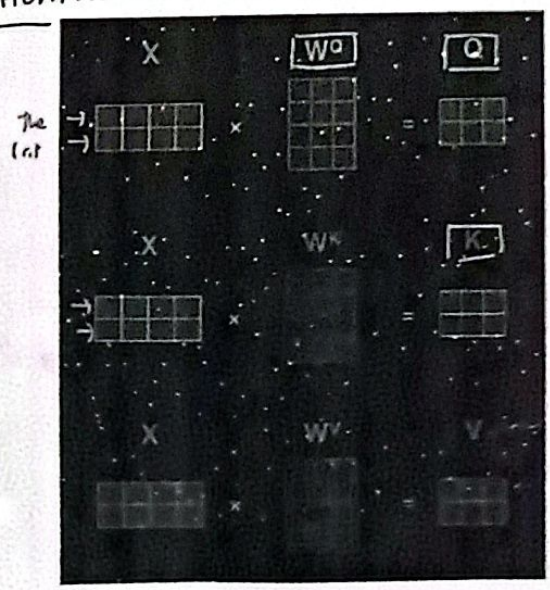
② ↳ Attention Score

③ ↳ Scaled

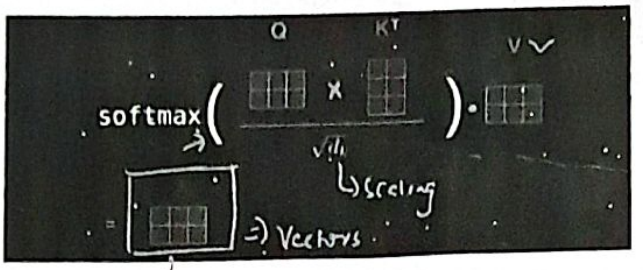
④ ↳ Softmax

⑤ ↳ Weighted Sum of Value (Softmax × V)

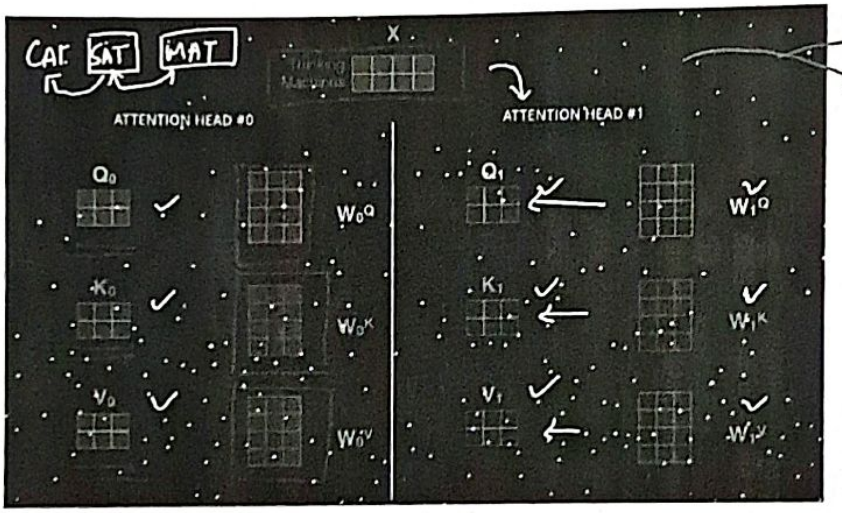
④ Multi Head Attention



for every word we get separate Q, K, V vector



→ Self Attention with Multi Heads



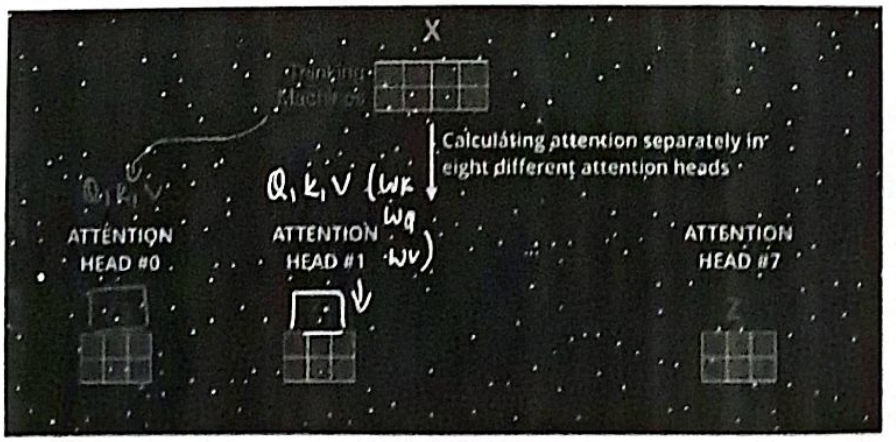
It expands Models ability to focus on different position of token

Score, Softmax × V

Z0 → Co Vector → contextualised

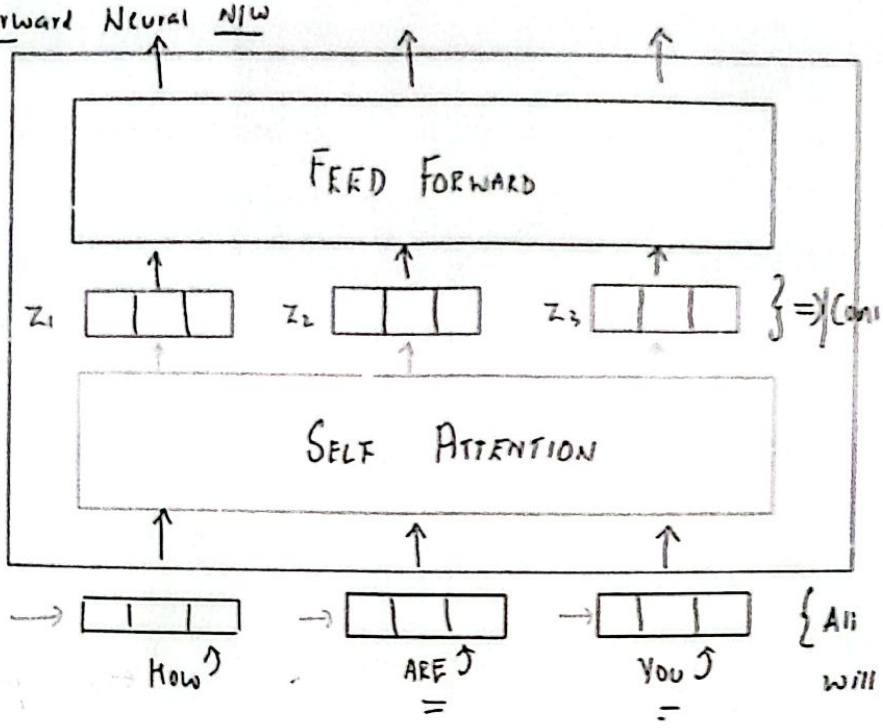
Z1

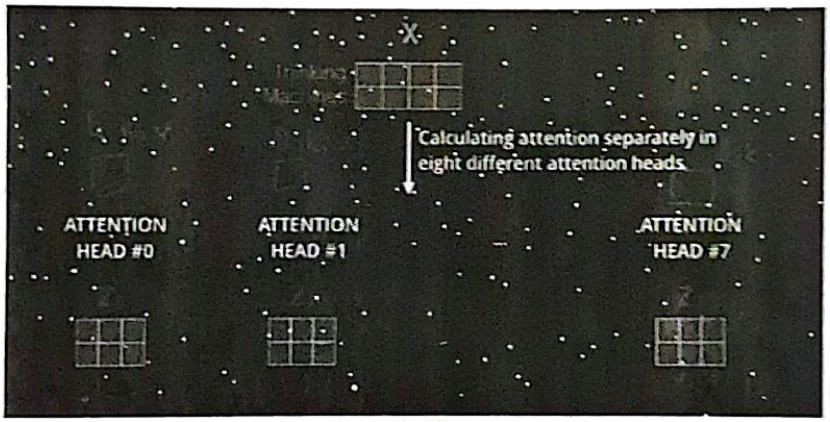
Multi Head Attention

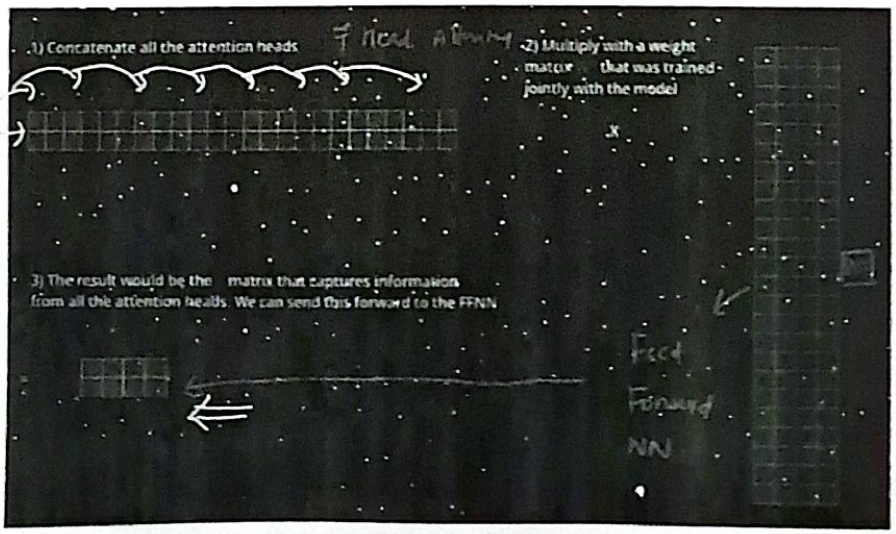


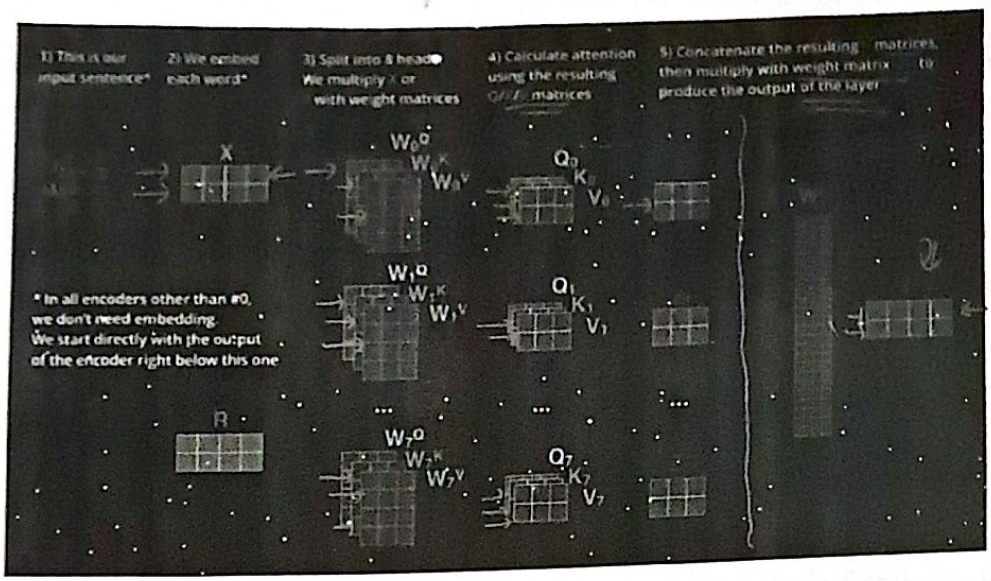
Z | Z | Z

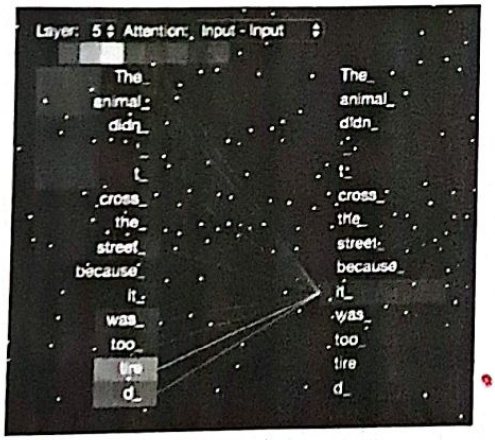
(6) Feed Forward Neural NW

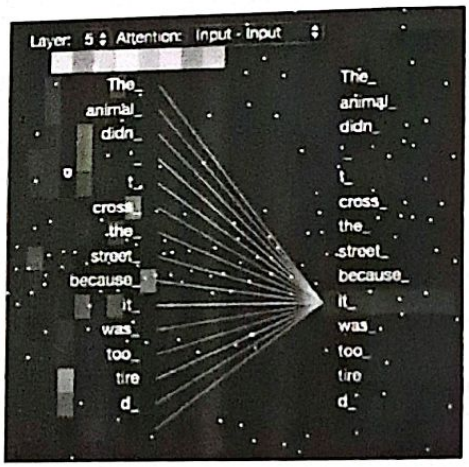










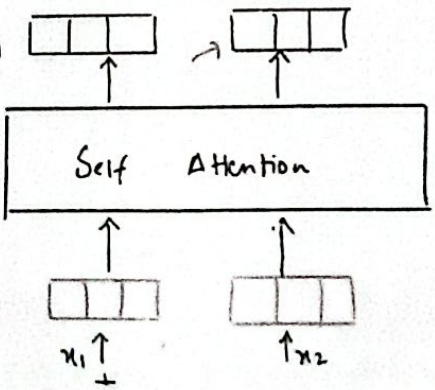


⑨ Positional Encoding – Representing Order of Sequence

① LION KILLS TIGER
② TIGER KILLS LION

⇓

Missing



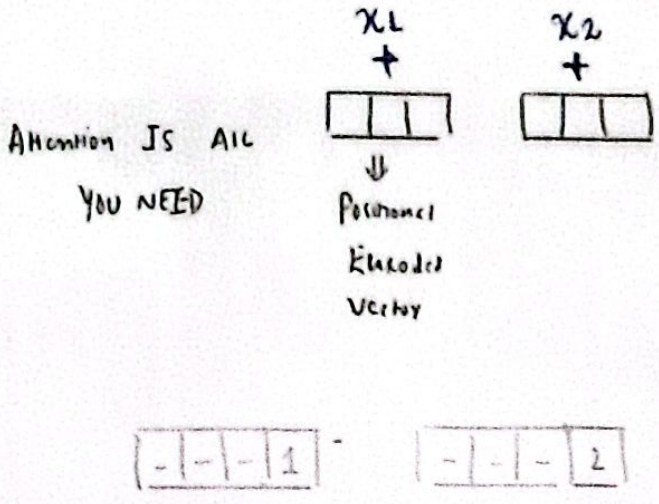
Advantage

① Word Tokens it can process parallely

⇓

DRAWBACK

Lack the sequential structure of the words {order}

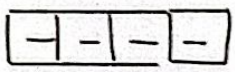


Types of Positon Encoding

1. Sinsusoidal Positon Encoding → In Research papers
2. Learned positional Encoding ⇒ Positonal Encoding Are learned during Training ⇐

① Sinsusoidal Positonal Encoding : It uses sine and cosine functions of different frequencies to create positonal encodings

Formula :



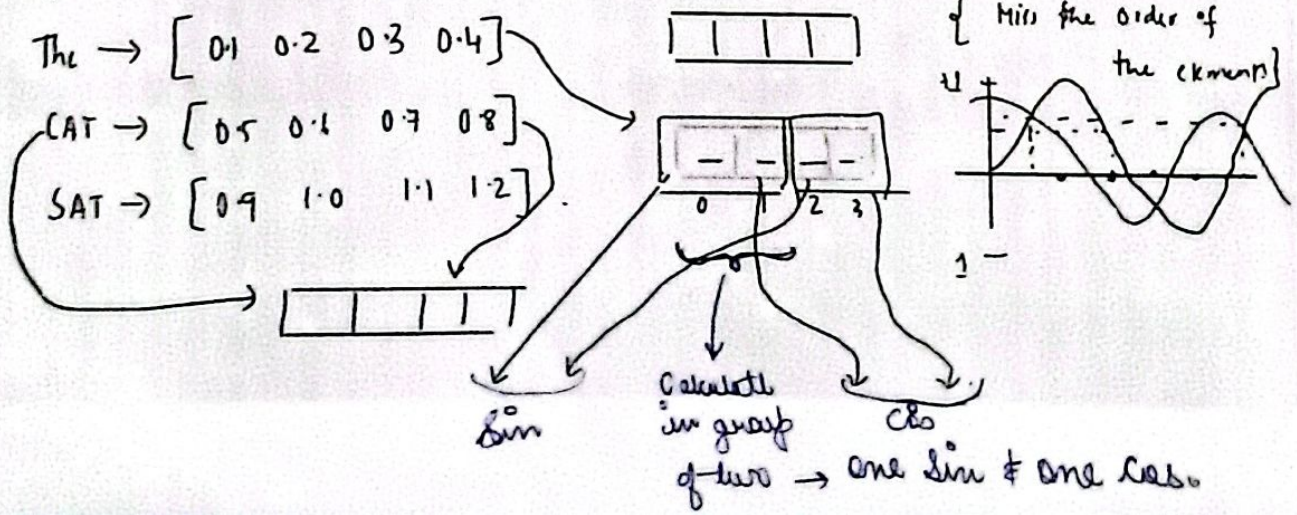
$$P.E(pos, 2i) = \sin\left(\frac{Pos}{10000^{2i/dmodel}}\right)$$

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

Where, pos → is the positon
i → is the dimension
dmodel → is the dimensionality Of the embeddings.

Eg : The Cat Sat

The → [0.1 0.2 0.3 0.4]
CAT → [0.5 0.6 0.7 0.8]
SAT → [0.9 1.0 1.1 1.2]



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

For our example $d_{model} = 4$

For position $pos = 0$ → THE ka liya

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

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

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

$PE(0,3) = \cos(0) = 1$ (Alternating)

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

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

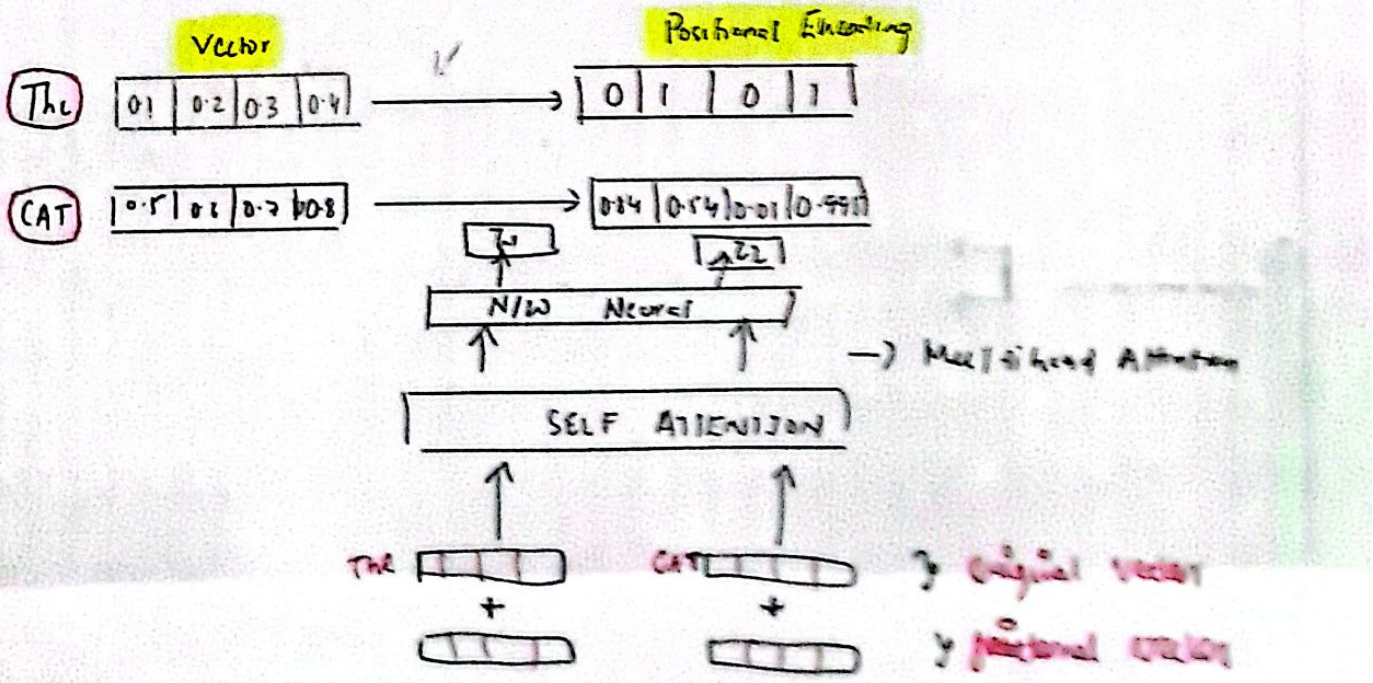
For $pos = 1$ → CAT ka liya

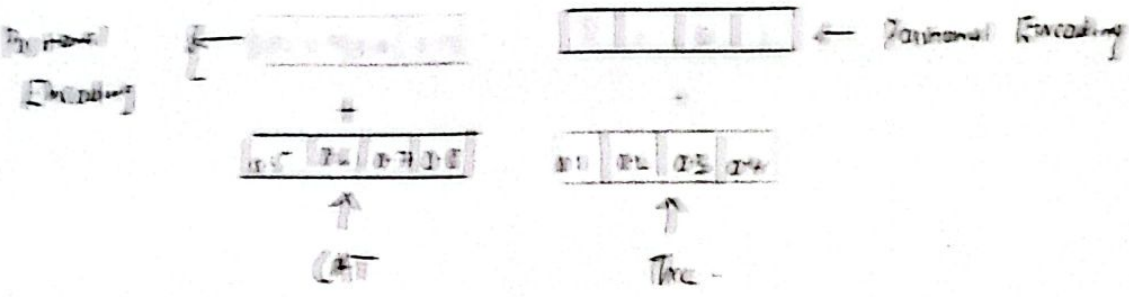
$PE(1,0) = \sin\left(\frac{1}{10000^{0/4}}\right) = \sin(1) = 0.8415$

$PE(1,1) = \cos\left(\frac{1}{10000^{2/4}}\right) \approx 0.5403$

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

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





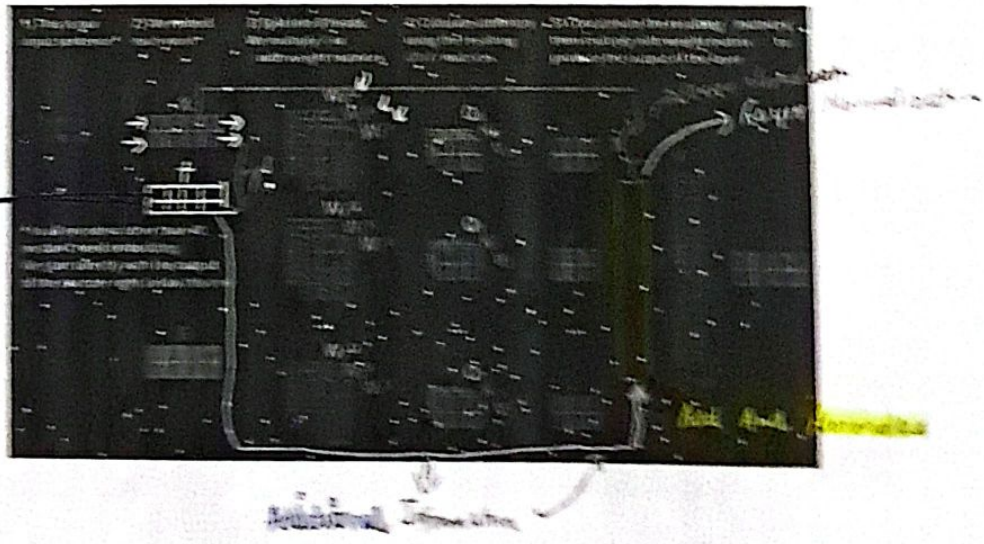
Transformers

Transformers

Residuals

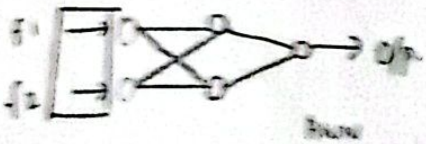
1. Self Attention layer
2. Multi head Attention
3. Positional Encoding
4. Layer Normalization

ADD AND Normalize



Normalization

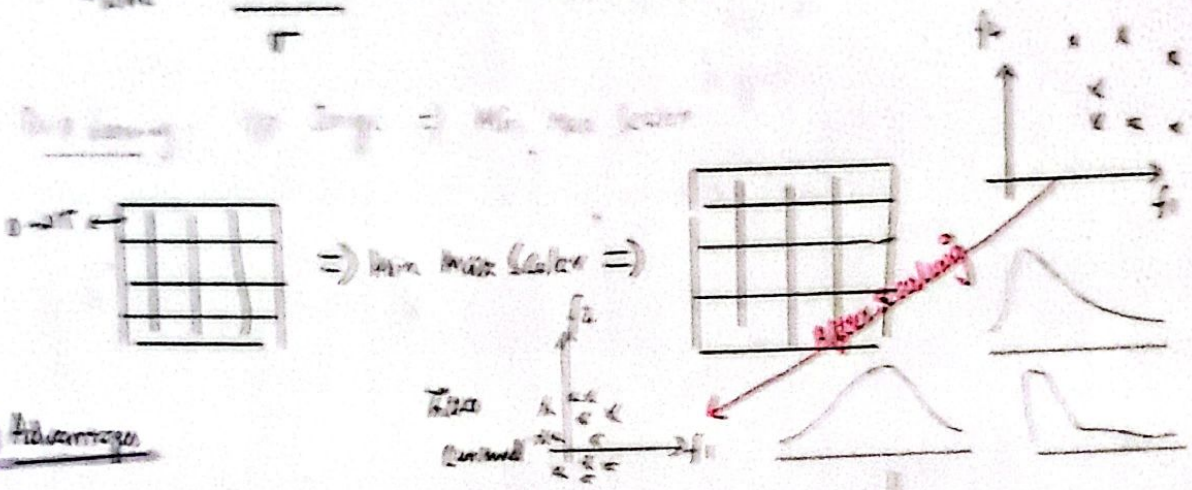
- Batch Normalization
- Layer Normalization



Normalization : Standard Scaling

$$z_{score} = \frac{x - \mu}{\sigma} \Rightarrow$$

$\mu = 0, \sigma = 1$ $\quad f_1 \longrightarrow f_1' \leftarrow \mu = 0, \sigma = 1$



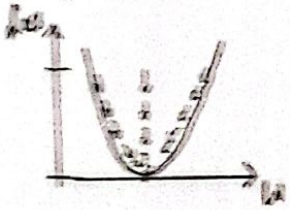
Advantages

1. Improved Training Stability $\left\{ \begin{matrix} \mu = 0 \\ \sigma = 1 \end{matrix} \right\}$ $\mu = 0, \sigma = 1$

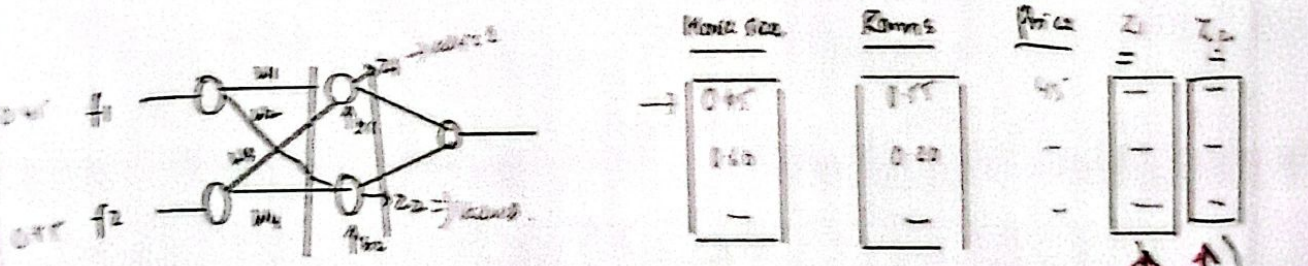
↓

Vanishing and Exploding Gradient problem

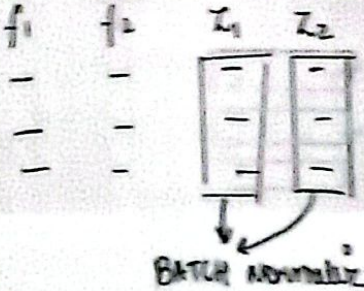
2. Faster Convergence



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



Batch Normalization VS Layer Normalization



Gamma → γ, Beta → β

γ, β → learnable parameters

Z_1	Z_2	
–	–	→ layer
–	–	→ layer
→ 0	–	
→ 0	0	→ (mean, $\sigma = 0$)
→ 0	0	
–	0	
–	–	
–	–	

Normalizing → μ, σ

Batch Normalization

Normalization ⇓ γ, β

$Z_1 = \sigma[W_1^T X + b_1]$

$Y = \gamma\left[\frac{Z_1 - \mu_1}{\sigma_1}\right] + \beta$

(γ, β: learnable parameters)

Scale And Shift parameters

Example to Understand this Theory

① "CAT" = [2.0, 4.0, 60.0, 8.0] ← {Vectors} (not normalized)

② Parameters = γ = [1.0, 1.0, 1.0, 1.0] → Learned scale

β = [0.0, 0.0, 0.0, 0.0] → Shift

} ⇒ Scale And Shift param

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

(i) Compute the mean

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

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

(ii) Compute the variance (σ^2)

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

(iii) Normalize the i/p

$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$

$\epsilon = 1e^{-5}$ ⇒ Avoid division by 0

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

$$\hat{x}_1 = \frac{2.0 - 5.0}{2.236} \approx -1.34$$

$$\hat{x}_2 = \frac{4.0 - 5.0}{2.236} \approx -0.45$$

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

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

Normalized vector

$$\hat{x} = [-1.34, -0.45, 0.45, 1.34]$$

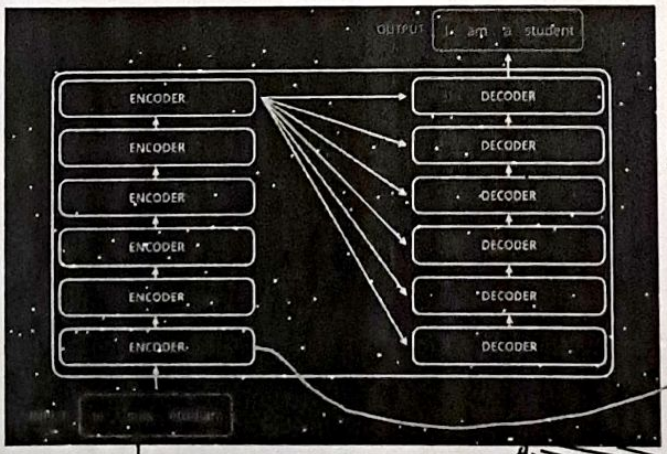
(4) Scale And Shift

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

$$\gamma = [1.0, 1.0, 1.0, 1.0] \quad \beta = [0.0, 0.0, 0.0, 0.0]$$

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

(5) Encoder Architecture [Research Paper]



Sequence to Sequence

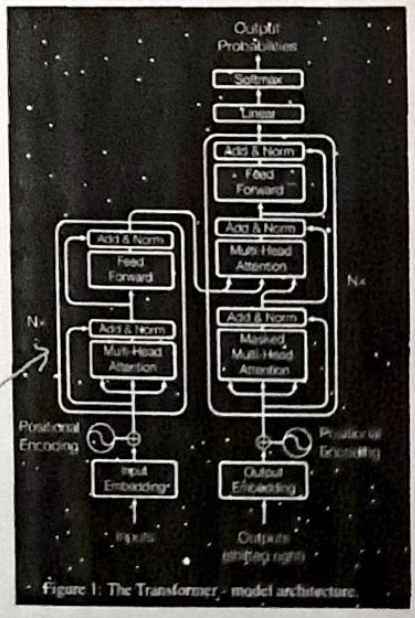


Figure 1: The Transformer - model architecture.

Feed Forward NN. → 0 0 0 0 0 0 0 0 0 0 0 0

(Add and Norm) → • • • • •

Multi Head Attention → $Z_1, Z_2, Z_3, Z_4, Z_5, Z_6, Z_7, Z_8$ {Research paper}.

Residuals

Text Embeddings + Positional Encoding ⇒ 512 → {Research paper}.

I/p Sequence

every word = 512.

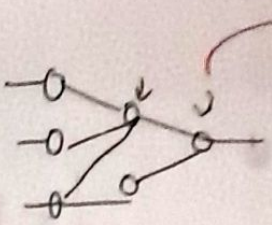
$Q_1 = 64$

$K = 64$

$V = 64$

$\sqrt{d_k} = 8$

⑥ Feed Forward NN Use



drives function problem

from linear functions

↓

With feed forward NN

① Adding Non linearity

② Processing Each position Independently.

Self Attention → Capture relationships B/w one another

FFN → Each token representation Independently.

↓

Transforming these representation further

and allows the model to learn

Richer Representation

} AdvN ⇒

③ FFN → Deeper ⇒ Adds Depth to the Model.

Depth ↑↑ ⇒ More learnings → DATA

⑦ 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 (Encode Decoder Attention)

③ Feed Forward Neural Network.

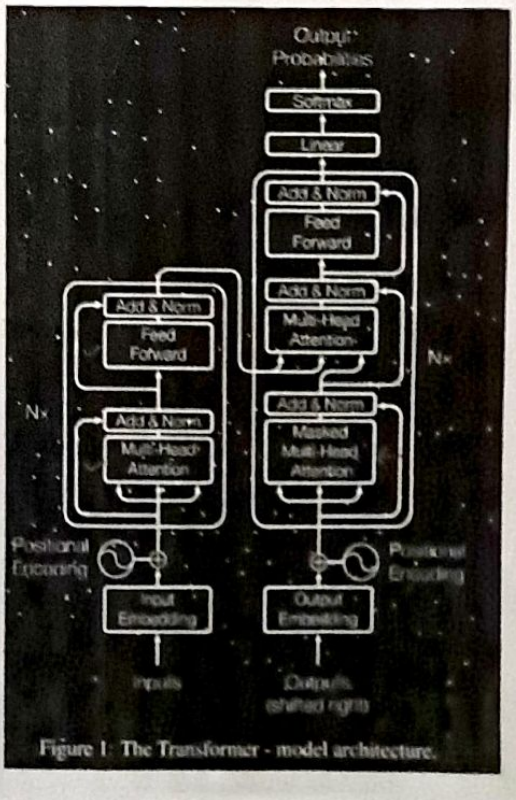
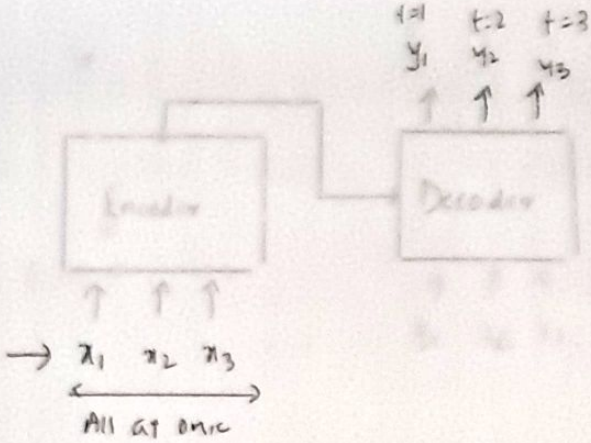


Figure 1: The Transformer - model architecture.



$\langle y_1, y_2, y_3 \rangle$

① Training Mechanism
② Inference Mechanism

① 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

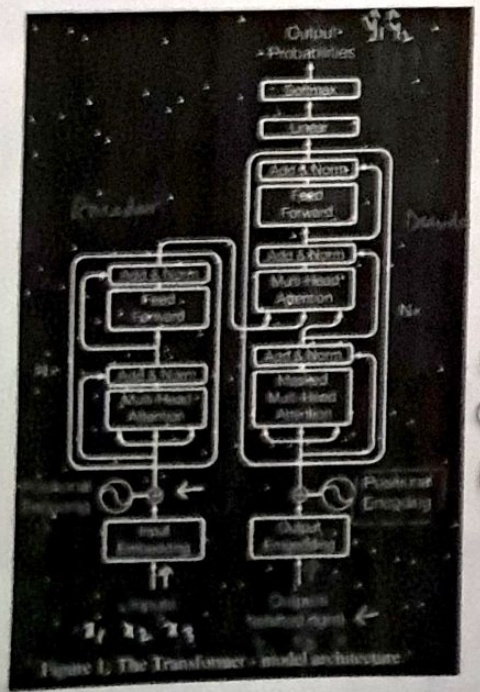


Figure 1: The Transformer - model architecture

Dataset

Eng	Hindi
$\langle x_1, x_2, x_3 \rangle$	$\langle y_1, y_2 \rangle$ ←
	↓
	$\langle y_1, y_2, 0 \rangle$
	↓ Zero padding

① Linear Projections Q, K, V
② Scaled Dot Product Attention
③ Mask Application → Look Ahead mask
→ Padding Mask

Masked Multi Head Attention

[1 2 3 0]

4 dimension vector

) Input Embedding and Positional Encoding

Output Embedding [STEP 1]

[[0.1, 0.2, 0.3, 0.4],
[0.5, 0.6, 0.7, 0.8], + PE → 0
[0.9, 1.0, 1.1, 1.2],
[0.0 0.0 0.0 0.0]]

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

$W_Q = W_K = W_V = I$

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

Q = Output Embedding * W_Q = Output Embedding
K = " * W_K = " "
V = " * W_V = " "

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

③ Scaled Dot Product Attention Calculation

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

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

[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]

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

ⓐ Masked Application

It helps manage the structure of the sequences being processed and ensures the models behaves correctly during training and Inferencing

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: i/p ← Sequence 1 [1, 2, 3]

o/p ← Sequence 2 [4, 5, 0] 0 is the padding token

↑ → Influence the Attention Mechanism

⇓

lead to Incorrect or biased predictions.

o/p → 4₁ 4₂ 0 0 0 0 0

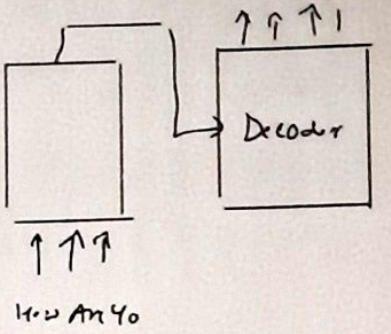
→ 100.

A padding mask ⇒ The tokens Are ignored.

Masking → Padding Mask, Look Ahead Mask.

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

Mask → Maintain Auto Regressive Property



t each position in the decoder
nce can only attend to
position, but no future position

Language Modelling, Translation

, 0] → [1, 1, 0] → 1D Mask.

Convert 1D to 2D Mask

Token 1 attends to token 1, 2

1, 2

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

For each token in the sequence the mask should indicate which tokens it can attend to

Mask → Decoder output

0]
0]
1]]

Padding And Looking Ahead Mask

ise multiplication of 2 mask

[[1, 0, 0]
[1, 1, 0]
[0, 0, 0]]

④ Mask

Scores : [[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]]

Look Ahead Mask

[[1 0 0 0]
[1 1 0 0]
[1 1 1 0]
[1 1 1 1]]

Padding Masking [extended to 2D Format]

[[1, 1, 1, 0]
[1, 1, 1, 0]
[1, 1, 1, 0]
[0, 0, 0, 0]]

Combined Mask : Look Ahead Mask * Padding Mask

[[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]]

=

[[1, 0, 0, 0],
[1, 1, 0, 0],
[1, 1, 1, 0],
[1, 1, 1, 0]]

⇒

[[1, -∞, -∞, -∞],
[1, 1, -∞, -∞],
[1, 1, 1, -∞]
[1, 1, 1, -∞]]

Masked Score

[[0.3, -∞, -∞, -∞]
[0.7, 1.9, -∞, -∞]
[1.1, 3.1, 5.1, -∞]
[0.0, 0.0, 0.0, -∞]]

Zero out the influence when the Softmax is applied.

Attention weights.

⑧ Softmax

Softmax Score = Softmax (Masked Scores)

$$= \begin{bmatrix} [1.0, 0.0, 0.0, 0.0], \\ [0.3, 0.7, 0.0, 0.0], \\ [0.1, 0.3, 0.6, 0.0], \\ [1.0, 0.0, 0.0, 0.0] \end{bmatrix}$$

⑨ Weigth 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.

ulti Head Attention

① Encoder O/p → set of Attention vector K & V

② Masked Multihead → Attention vector Q {Query Vector}

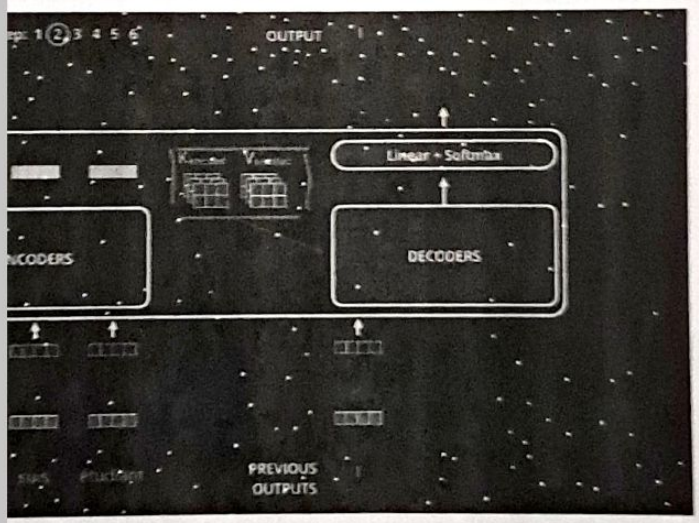
These are to be used by each decoder in its "Encoder-decoder" Attention layer

⇓

Dataset

i/p $\langle x_1, x_2, x_3 \rangle$

o/p $\langle y_1, y_2, y_3 \rangle$.



⑥ The Final Linear And Softmax Layer

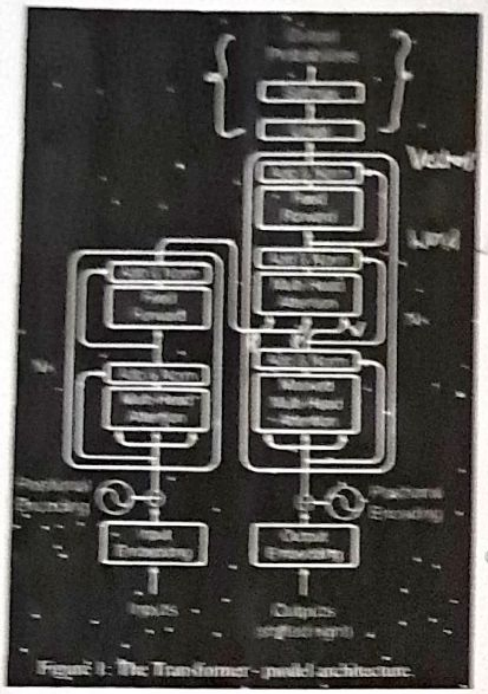
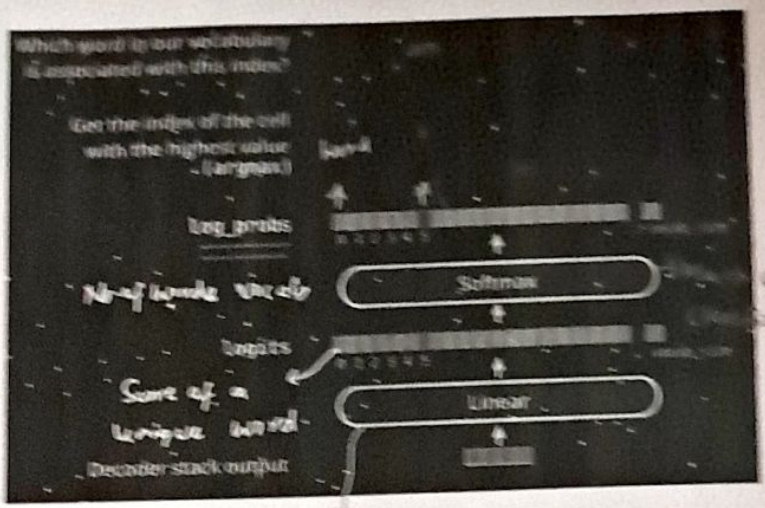


Figure 1: The Transformer - model architecture.

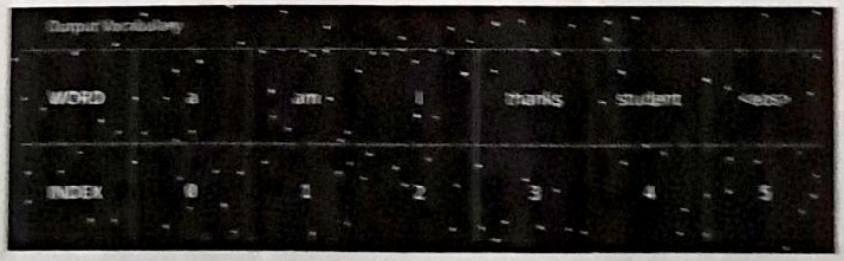


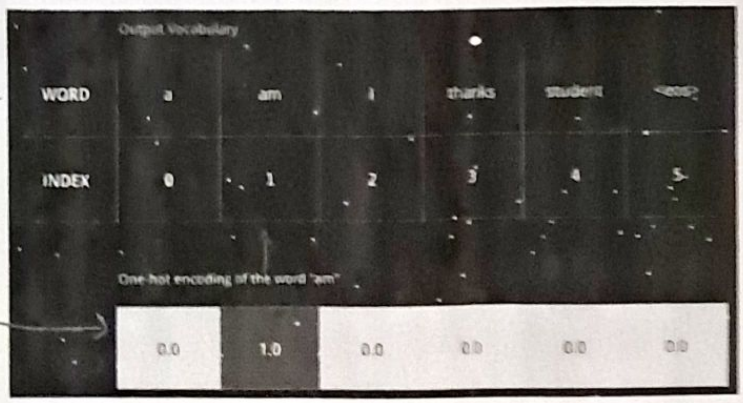
Linear ⇒ The Linear layer is a simple fully connected neural network that projects the vector produced by the stack of decoder ⇒ logits vector
⇒

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

⑦ Softmax layer turns those scores into probabilities (all add upto 1.0).
The cell with the highest probability is chosen, and the word associated with it is produced as the o/p. ⇒ time step.

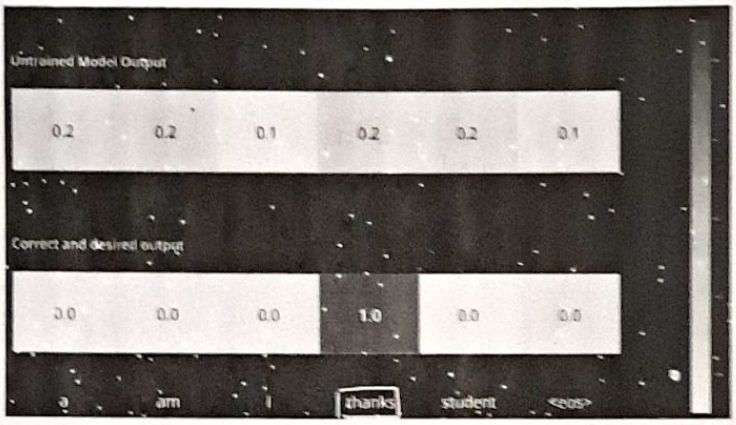
Recap of Training



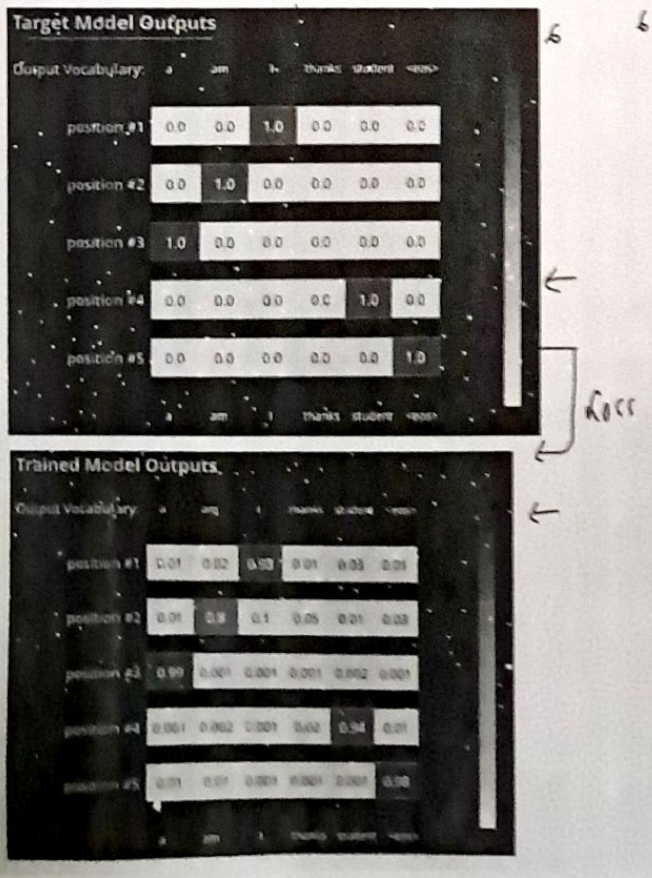


ONE ←

Model → Target



⇒ Back Propagation



Loss

I/P

a, am, i, student

O/P

I am a student

ONE ONE ONE ONE

NLP in Machine Learning

ROADMAP

Bert
↓
Transformers
↓
Text preprocessing → word Embedding → D.L → tensorflow & keras
↓
Neural N/w — RNN, LSTM RNN, GRU RNN → Deep Learning
↓
ML — I/p text → vector — Text preprocessing 3 → word2vec, Avgword2vec
I/p text → vector — Text preprocessing 2 → BOW, TFIDF, Unigram
Cleaning the I/p — Text preprocessing → Tokenization, Lemmatization
↓
NLTK & Spacy

Tokenization in NLP

① Corpus → Paragraph
② Documents → Sentences
③ Vocabulary → Unique word
④ words

Tokenization

"My name is Krish and I have a interest in teaching ML, NLP and DL. I am also a Youtuber"

↓

Tokens (Sentences)

① My name is _ _ _ _ _ DL
② I am also a Youtuber

↓

Tokenization

↓

words

[I like to drink Apple Juice. My friend likes mango Juice]

⇓ Tokenization

11 words — Token (sentence)

① I like to drink Apple Juice
② My friend likes (s) Mango Juice

↓ If Remove

→ Unique words → 10 word

↓

Vocabulary

↓ & (s) remove

9 words

- NLTK V/S spacy in Vs Code.
- Stemming
  - ↳ Word to its word stem
  - ↳ root word known as lemma
- Lemmatization (POS Tag → part of speech, eg → Noun, Verb)
  - ↳ To solve problem come with stemming
  - ↳ WordNet lemmatizer class se check krke replace krta hai
- Stopwords
  - ↳ Available for many languages
  - ↳ eg → In english → They are → i, me, my, and, are, why