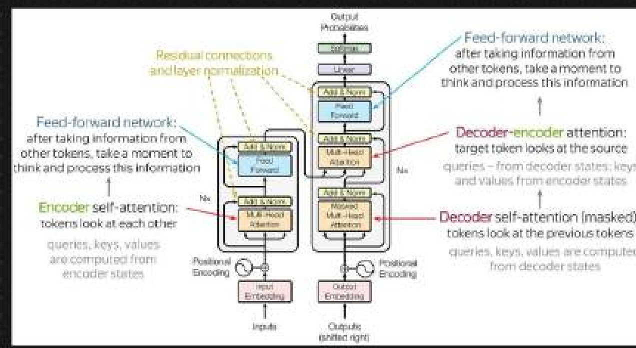
Introduction To Transformers

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 ←
↓
Open AI ⟶ 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

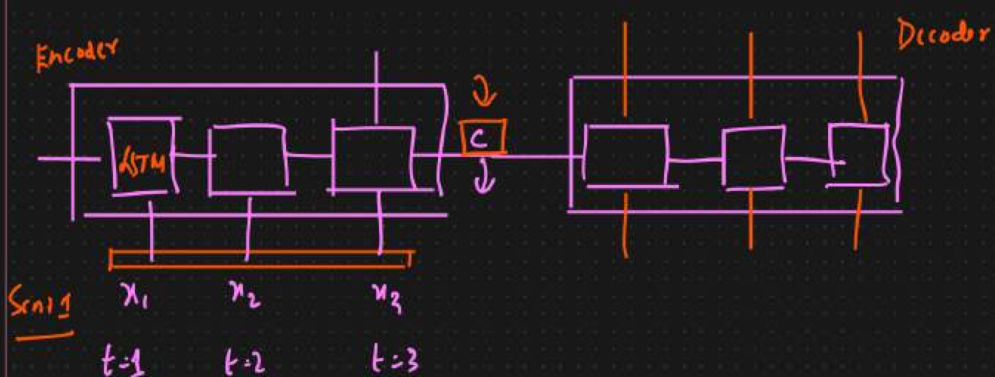
English ⟶ French

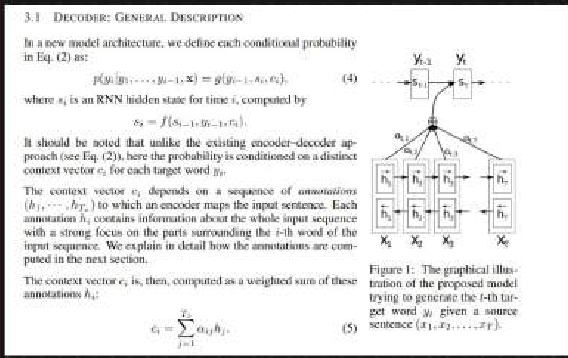
i/p ⇒ Many ⟶ o/p: Many. {Length of the Sentence}.

Sentence Length ↑↑   Bleau Score ⇓

Length Sentence ↑↑.

Encoder - Decoder





3.1 DECODER: GENERAL DESCRIPTION

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

$$p(y_i | y_1, \ldots, y_{i-1}, \mathbf{x}) = g(y_{i-1}, s_i, c_i), \tag{4}$$

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.

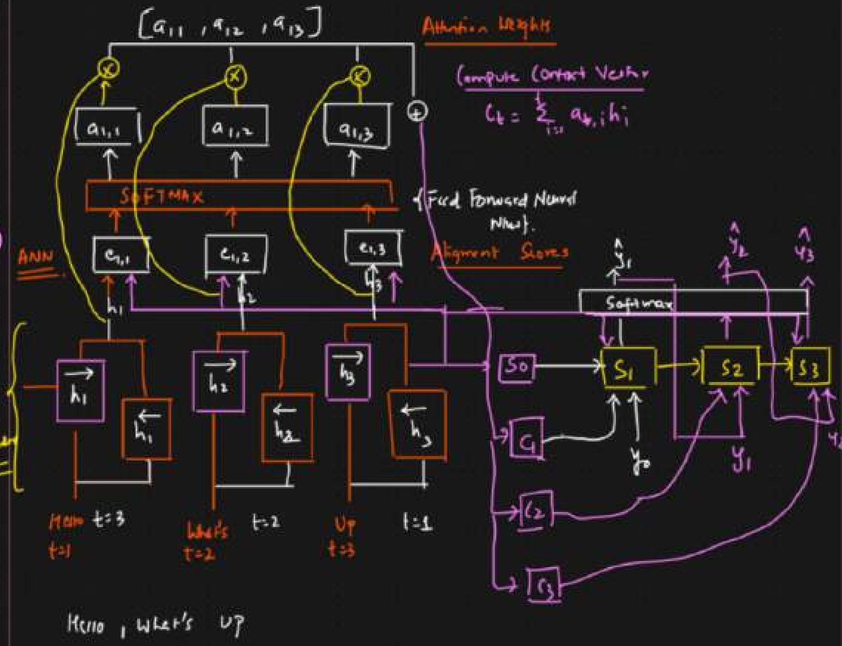
The context vector c_i depends on a sequence of *annotations* $(h_1, \cdots, h_{T_x})$ to which an encoder maps the input sentence. Each annotation h_i contains information about the whole input sequence with a strong focus on the parts surrounding the i-th word of the input sequence. We explain in detail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

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j. \tag{5}$$

Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

⇒



Additonal Context ⟶ Decoder

Long Sentence. Accuracy ↑↑.

Attention Mechanism

① Parallely we cannot send all the words in a sentence ⟶ $\boxed{\text{Scalable}}$

DATASET ⟶ Huge ⟶ Scalable with Respect to Training.

TRANSFORMERS ≠ LSTM RNN

↳ Self Attention Module ⟵ All the words will be parallely sent to encoder.

⇓

Positional Encoding

Transformer ↑↑ DATASET ⟶ Amazing SOTA ⟵ NLP

Transfer Learning ⟶ MultiModal Task ⟶ NLP + Image ⟵

Transformers : AI Space ⟶ SOTA Model ⟶

BERT GPT ⟶ Transfer Learning ⟶ SOTA Models ⟶ DALLE } LLM's, Generative AI

↑

Train Huge Data

② Contextual Embedding → Self Attention

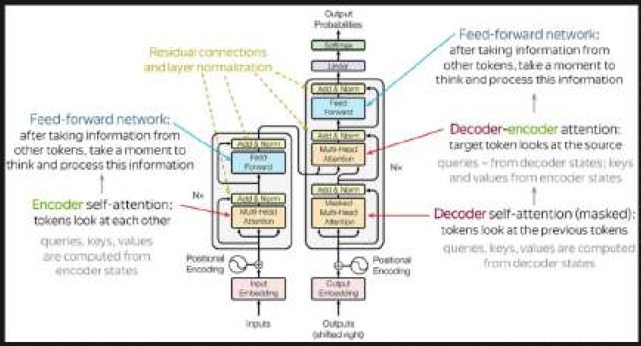
Contextual Vectors

Eg: My name is KRISH And I play CRICKET

Word2vec | Embedding Layer |

| Contextual vector Embedding |

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



I/p → The TRANSFORMER → O/p.

↑ 6 Encoder

Encoder

Comment vas-tu?

Encoder → Decoder

HOW ARE YOU?

O/P Encoder Decoder Architecture

← 6 Decoders

Decoder

Encoder

Feed Forward Neural N/W

Self-Attention

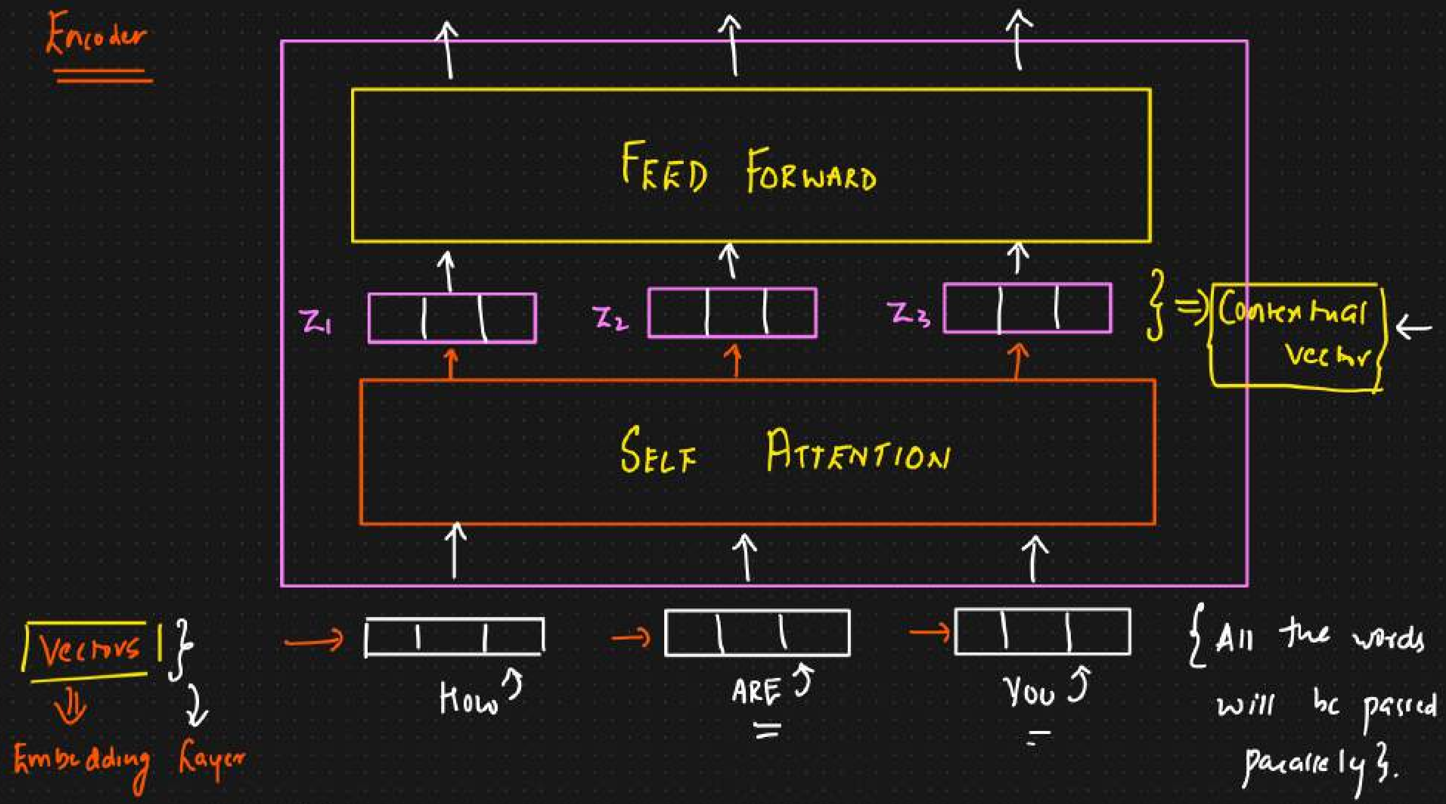
{LSTM
RNN

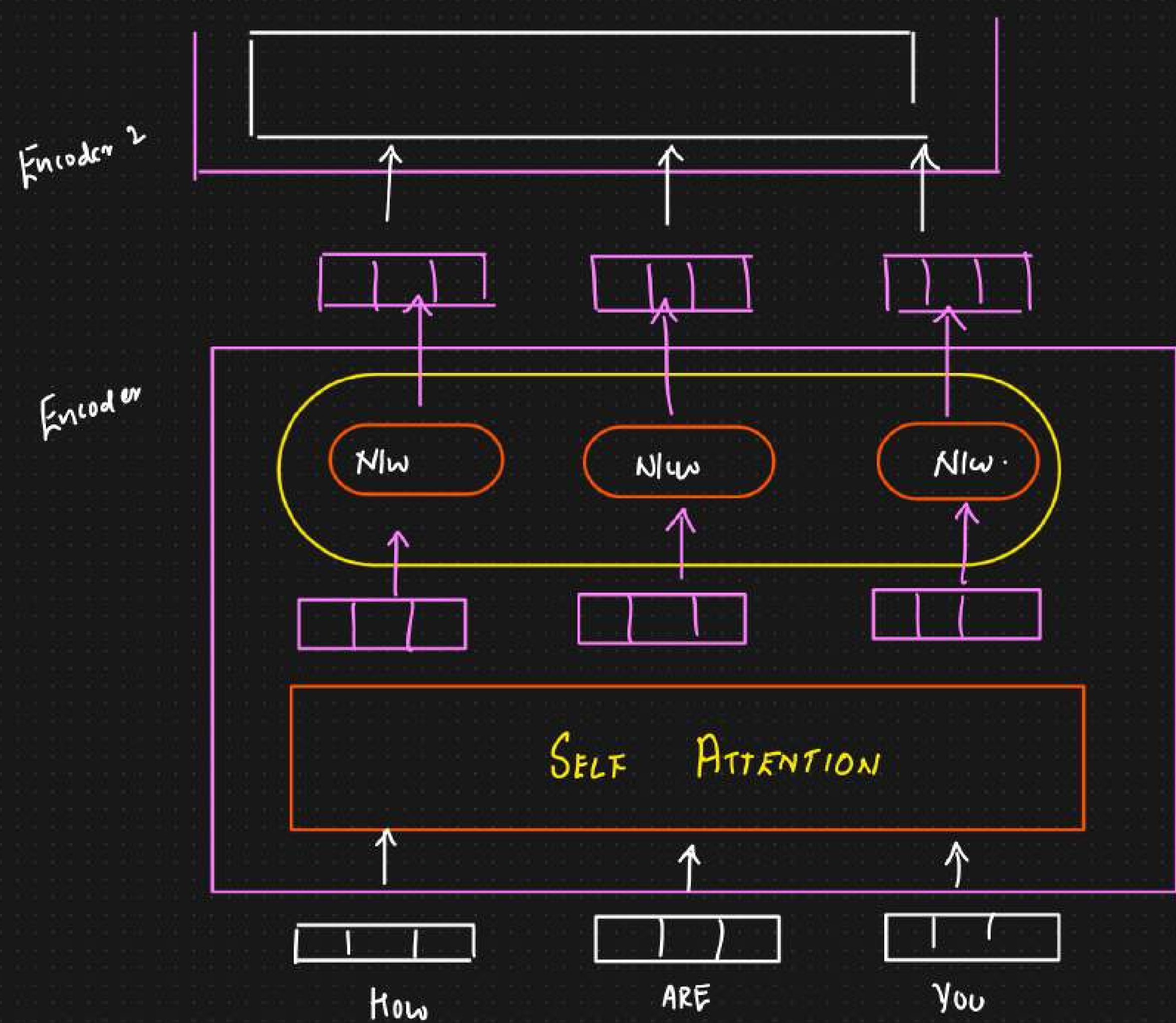
Feed Forward

Encoder - Decoder Attention

Self - Attention

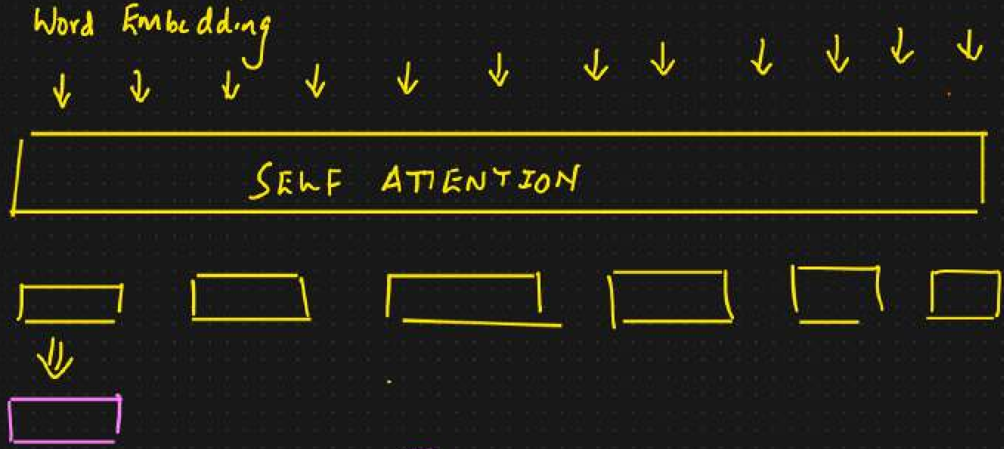
Encoder

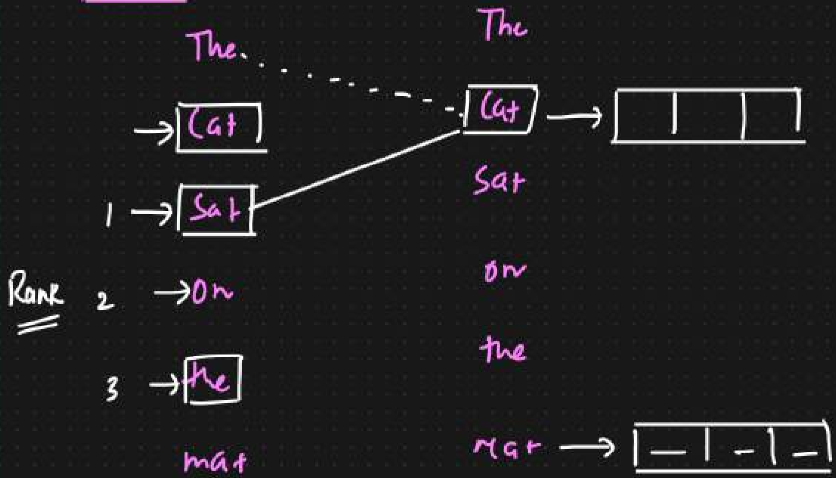

FEED FORWARD
Z_1
Z_2
Z_3
Contextual vector
SELF ATTENTION
Vectors
Embedding layer
How
ARE
You
{All the words will be passed parallely}.


Encoder 2
Encoder
N/w
N/w
N/w
SELF ATTENTION
How
ARE
You

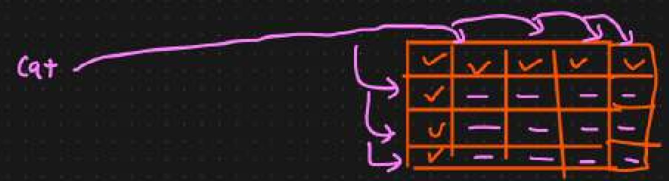
Self Attention At a Higher Level

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





{Contextual Embedding}



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 (Topi'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 sentence.

I/P — How (1x3) — GOOD (1x3)

Embedding	X_1	X_2
Queries	q_1	q_2
Keys	k_1	k_2
Values	V_1	V_2

(3x3)

W^Q — Dependencies of each And every word

W^K — Context of words

W^V

Score $\quad q_1 \cdot k_1 = \quad q_1 \cdot k_2 =$

* 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 gradients }

Dimension = 3 → $\frac{\text{Variance}}{\sqrt{dk}} \approx \frac{\text{Variance} \uparrow\uparrow\uparrow}{\sqrt{dk}}$

Dimension ↑↑ 512

O/P ↑↑↑

Variance ↑↑↑ ⇒

SoftMax $\quad$ [0-1] $\quad$ [0-1].

0.88 + 0.12 = 1

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

X $\quad$ X $\quad$ X

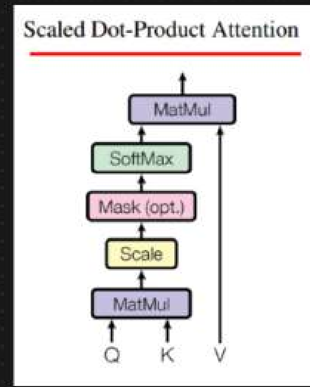
Value Vectors $\quad V_1$ [1 | 1 | 1] $\quad V_2$. [1 | 0.5 | 1]

0.88 [1 | 1 | 1]

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

Contextual vector → Z_1 $\quad Z_2$

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 :

→ | 0.8 | 0.3 | 0.4 | | 0.4 | 0.8 | 0.2 | | 0.3 | 0.4 | 0.8 | { Contextual Embedding }

Language Translation   Self Attention

Text Summarization

| 1 | 0 | 0 | | 0 | 1 | 0 | | 0 | 0 | 1 | { Sentence, Dataset }.

The   CAT   SAT

Embedding Layer → Fixed Vector

1) Inputs: Queries, Keys, And Values

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"]

Q, K, V $\Rightarrow$ 4

Embedding size = 4

$\rightarrow \boxed{\text{Self Att}} \rightarrow$

0.1 | 0.2 | 0.3

① Token Embedding

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

$E_{sat} = [1, 1, 1, 1]$

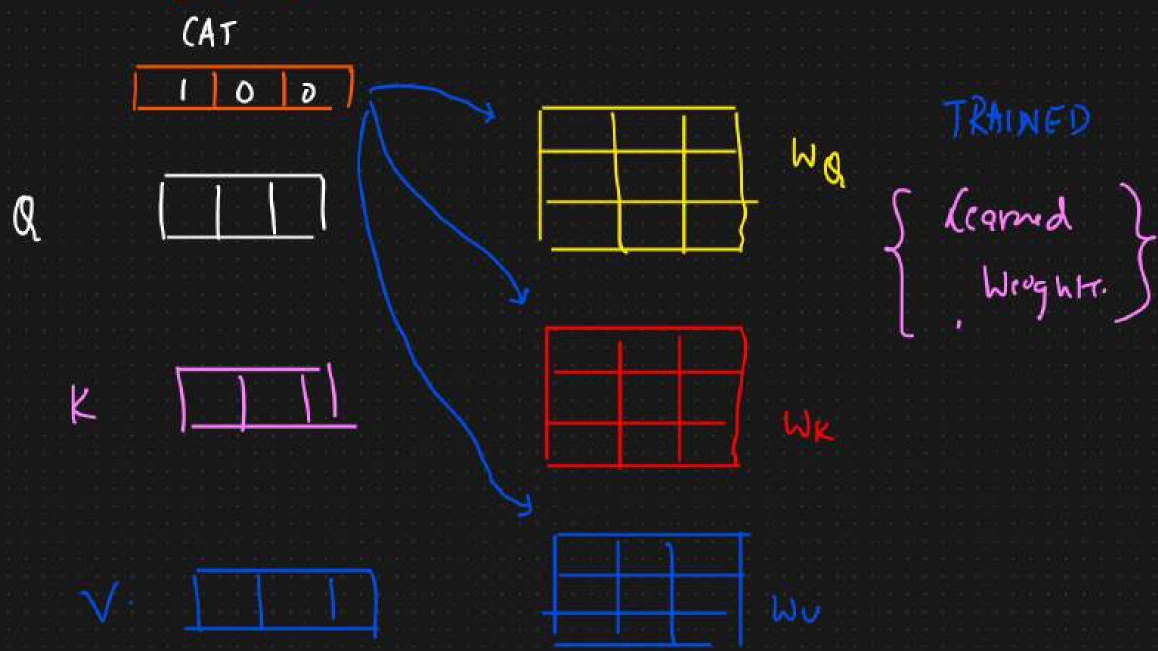
1 | 0 | 0

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

Sentence, Dataset

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

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

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

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

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

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

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

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

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

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

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

④ Scaling :

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

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

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 $\longrightarrow$

① Gradient Exploding

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

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

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 $[6, 4]$ ⇒ Scaling Not Applied

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

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

$$\approx [0.88, 0.12].$$

(*) Property of Softmax W_Q, W_K, W_V

$$([10, 1]) = [0.99, \boxed{0.01}]$$

Dot product = Large values

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

With Scaling

$\sqrt{d_k} = \sqrt{4} = \sqrt{2}$ ⇒ Variance 2, 3

① Compute Scaled Dot Product

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

dimension ↑ Variance ↑ 2

$\sqrt{d_k}$ ↑ same

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

$$= [0.73, 0.27]$$ ⇒ Attention Weights

(*) 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

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

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

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

6) Weight Sum of Values

We multiply the attention weights by corresponding value vectors

For the Token The =

$$\text{Output}_{(\text{The})} = 0.4223 * V_{\text{The}} + 0.1554 * V_{\text{CAT}} + 0.4223 \cdot V_{\text{Sat}}$$

$$= 0.4223\,[1\ 0\ 1\ 0] + 0.1554 * [0\ 1\ 0\ 1] + 0.4223 * [1\ 1\ 1\ 1]$$

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

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

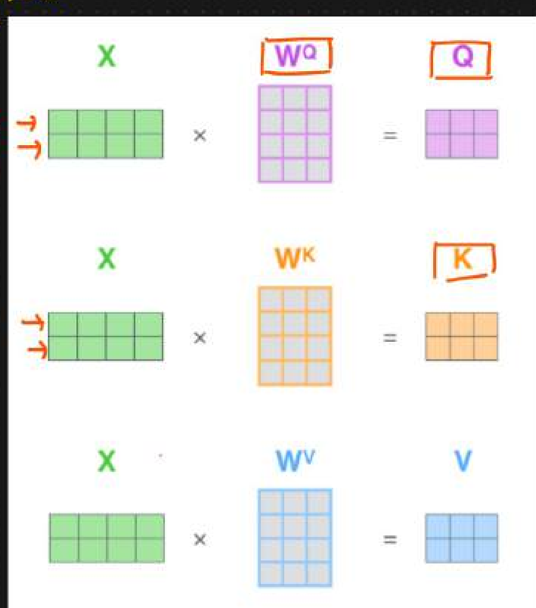
Contextual vector

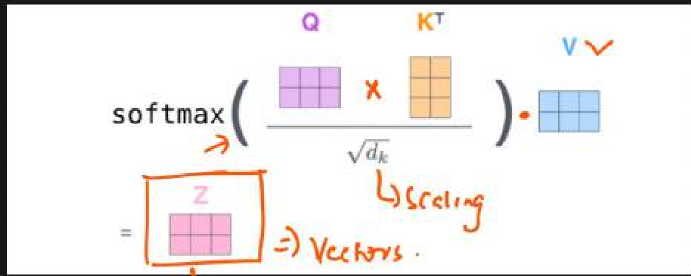
$$\text{The}\ \boxed{1\,|\,0\,|\,1\,|\,0} \Rightarrow \text{Self Attention} \Rightarrow [1.2669, 0.9999, 1.2669, 0.9999].$$

1. ↳ Q, K, V $[W^Q, W^K, W^V]$
2. ↳ Attention Score
3. ↳ Scaled
4. ↳ Softmax
5. ↳ Weighted Sum of Value (Softmax × V).

(4) Multi Head Attention

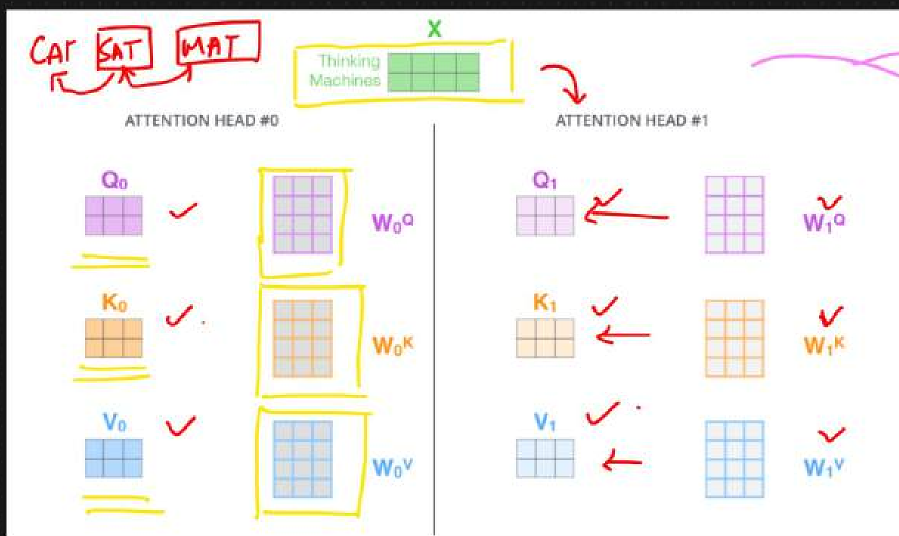
The
Cat





Attention Head.

→ Self Attenton with Multi Heads



It expands model's ability to focus on different position of tokens.

Score, Softmax X V

[- - -]

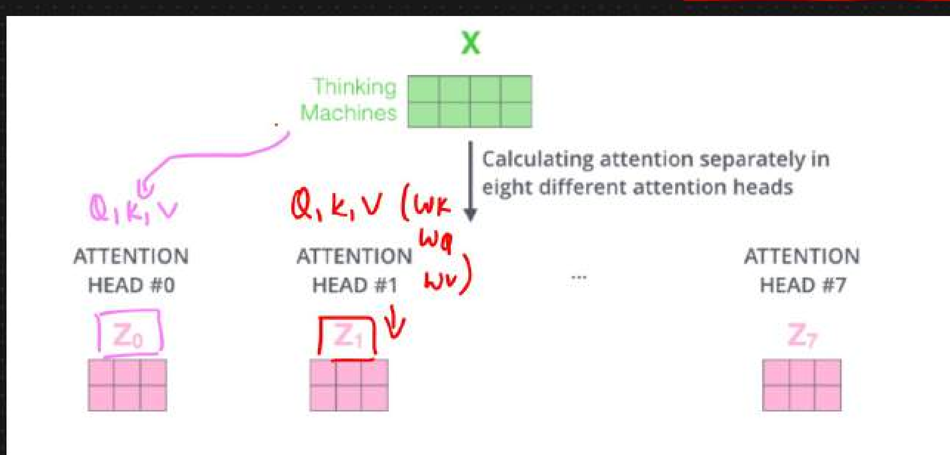
Z0

Vectors

[- - -]

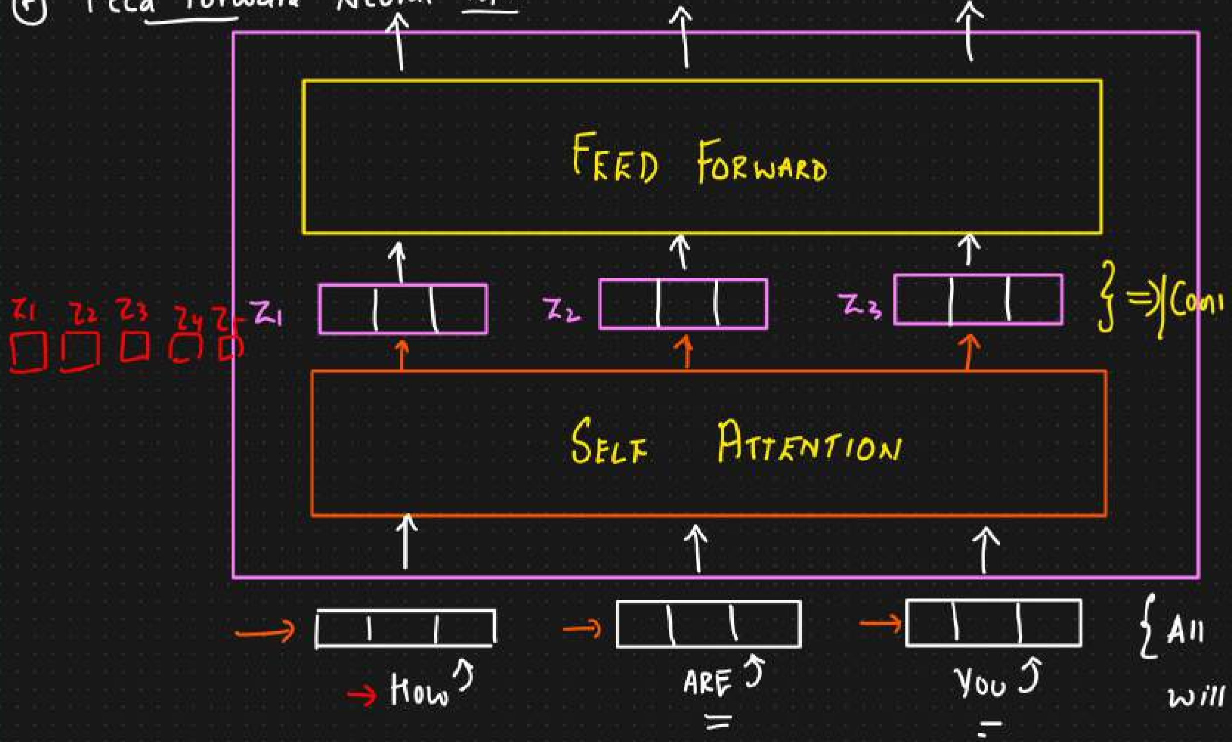
Z1

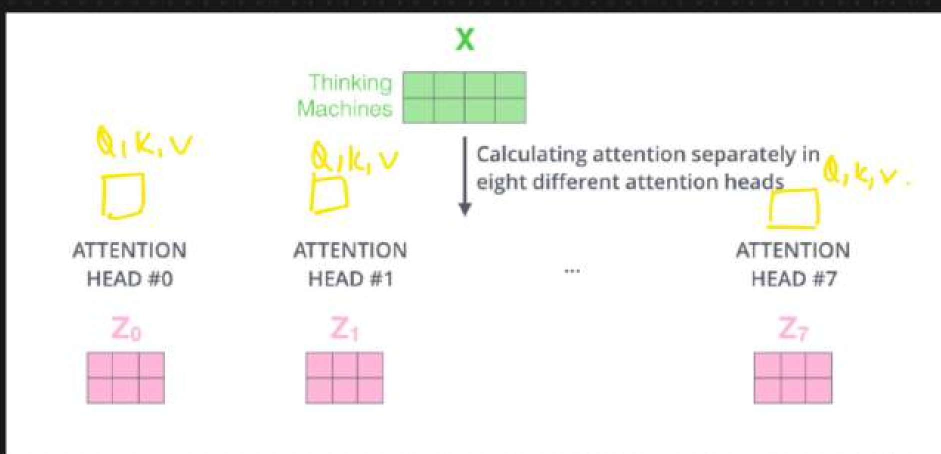
Multi Head Attention

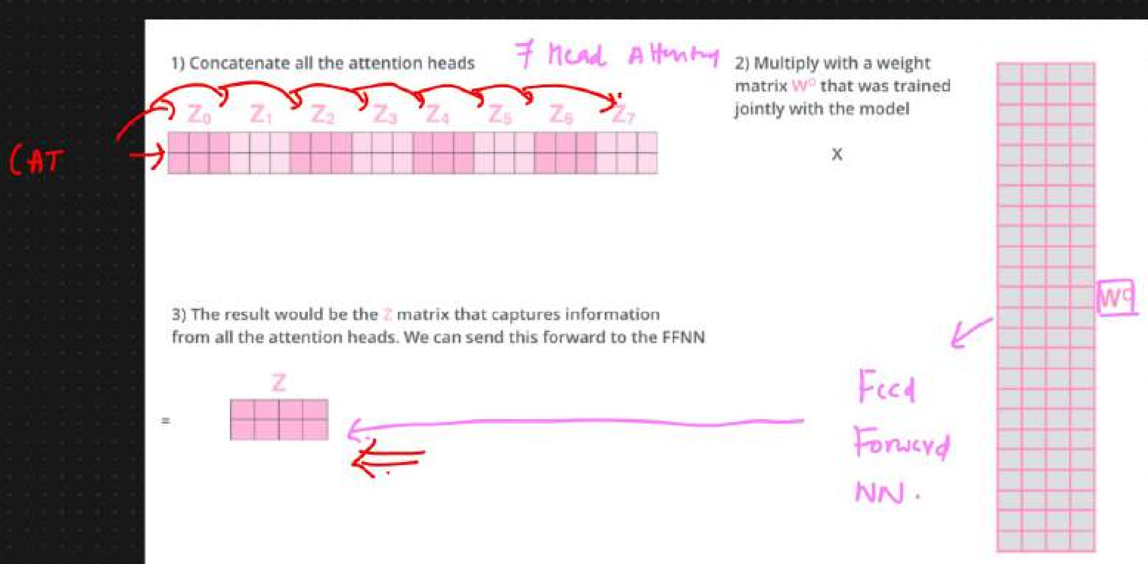


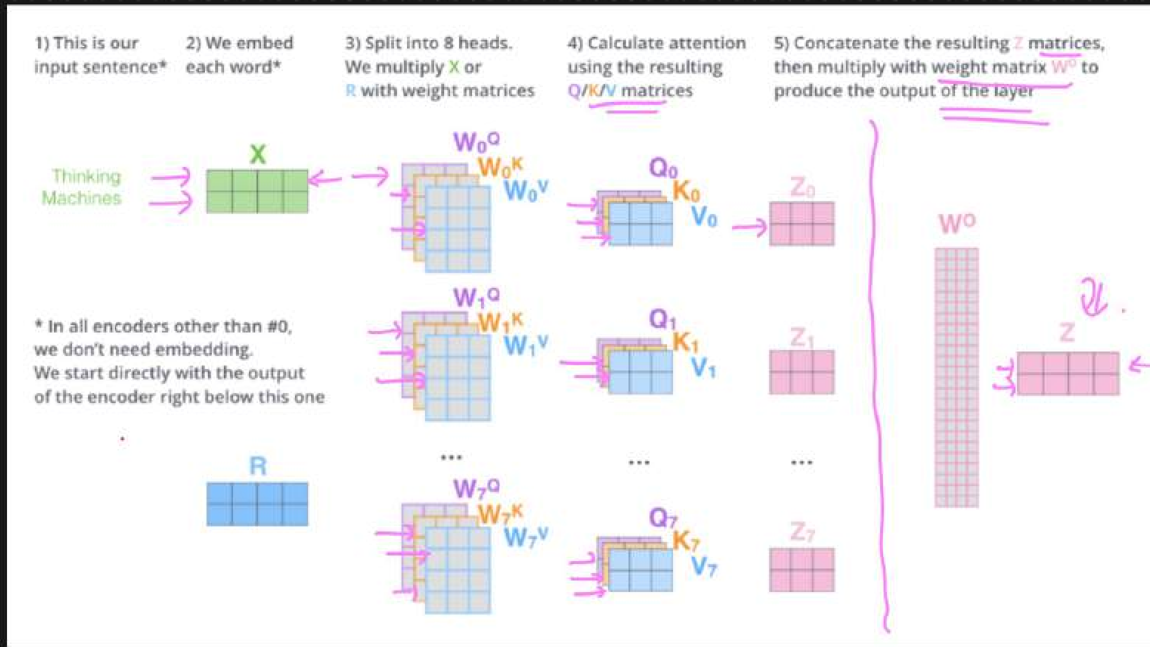
Z Z Z

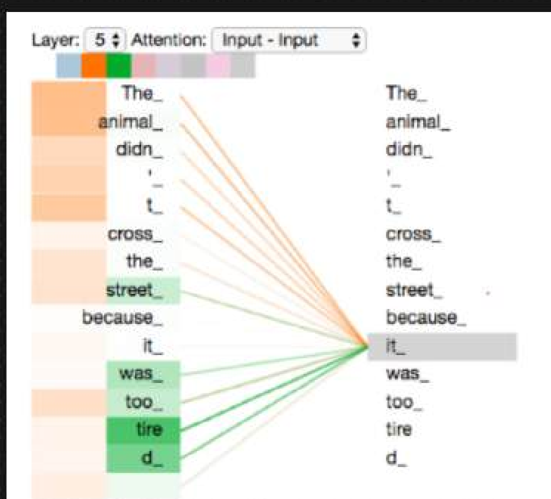
(f) Feed Forward Neural N/W

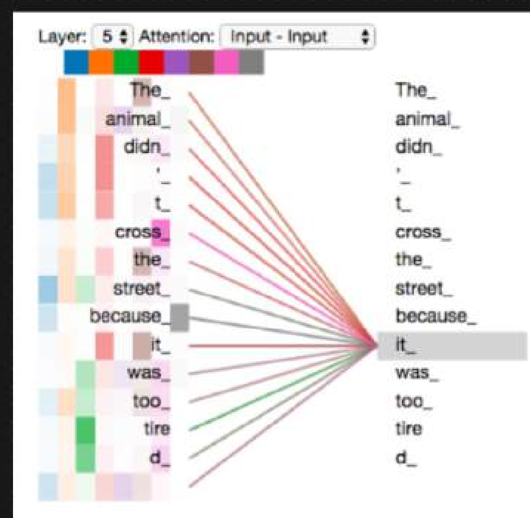












(1) Positional Encoding – Representing Order of Sequence

Advantage

(1) Word Tokens it can process parallelly

⇓

DRAW BACK

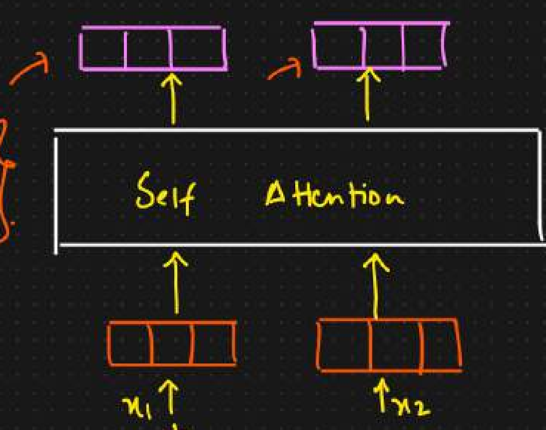
Lack the sequential structure of the words {order}

(1) Lion kills Tiger

(2) Tiger kills Lion

⇓

Missing



Attention Is All You NEED

Positional Encoded Vector

Book, Journal Novel
↳ 1 lakh words }.
⇓
Back propogation

- | - | - | 1   - | - | - | 2   - | - | - | 3 ←

Types of Position Encoding

✓1) Sinsusoidal Position Encoding ⟶

2) Learned positional Encoding ⇒ Positional Encoding Are learned during Training ⇐

① Sinsusoidal Positional Encoding : It uses sine and Cosine functions of different frequencies to create positional encodings

Formula :

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

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

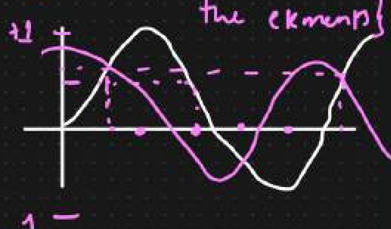
Where pos is the position
i is the dimension
dmodel is the dimensionality of the embeddings.

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].

0 1 2 3

{ Miss the order of the elements}



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

For our example $d_{model} = 4$

For position $pos = 0$

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

$$PE(0,1) = \cos\left(\frac{0}{-}\right) = \cos(0) = \underline{1}$$

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

$$P.E(0,3) = \cos(0) = \underline{1}$$

$P.E = [0,1,0,1]$

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

For $pos = 1$

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

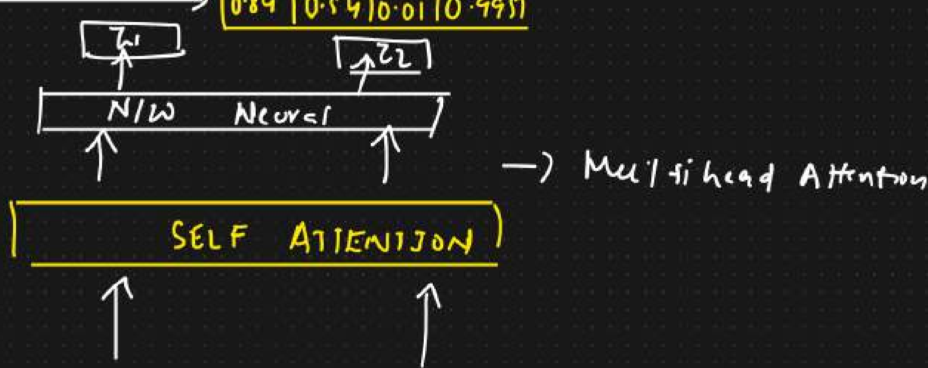
$$P.E(1,0) = \sin\left(\frac{1}{10000^{0/4}}\right) = \sin(1) = 0.8415$$

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

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

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

	Vector		Positional Encoding
The	0.1 | 0.2 | 0.3 | 0.4	→	0 | 1 | 0 | 1
CAT	0.5 | 0.6 | 0.7 | 0.8	→	0.84 | 0.54 | 0.01 | 0.995



Positional Encoding → | 0.84 | 0.54 | 0.01 | 0.99 |

\+

| 0.5 | 0.6 | 0.7 | 0.8 |

↑

CAT

| 0 | 1 | 0 | 1 | ← Positional Encoding

\+

| 0.1 | 0.2 | 0.3 | 0.4 |

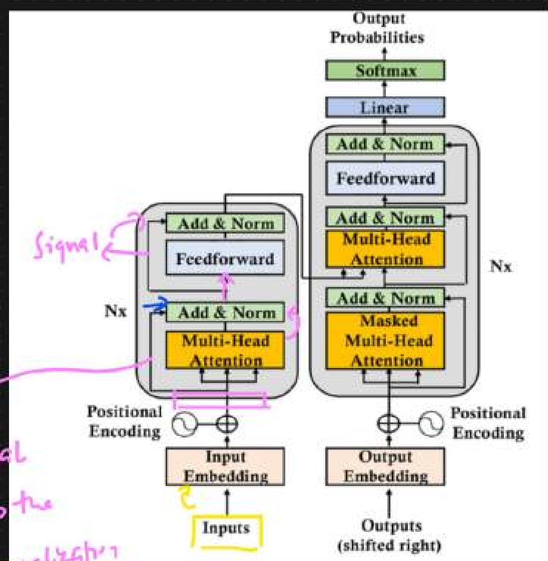
↑

The.

④ Layer Normalization In Transformers

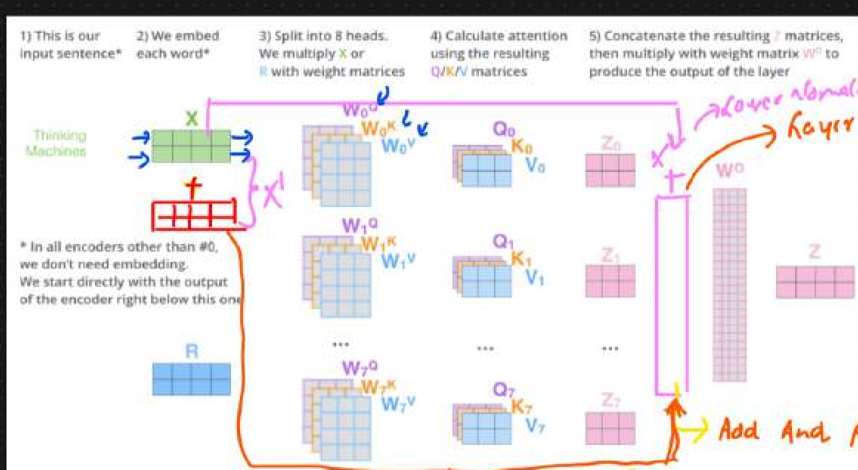
Transformers

Residuals

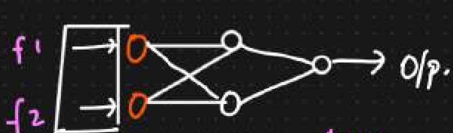


Residuals
↳ Additional signal to the Layer Normalization

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



Normalization
→ Batch Normalization
→ Layer Normalization



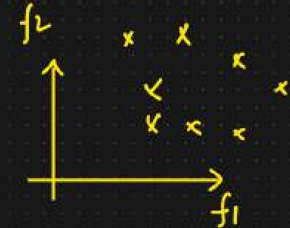
f1 House Size	f2 No.of Rooms	Price
1200	2	45
1500	3	70

$\mu, \sigma \leftarrow$ | 2000 | 3.5 80

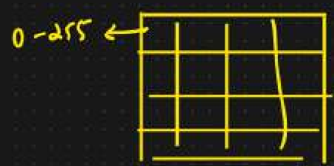
Normalization : Standard Scaling

$$Z_{score} = \frac{x_i - \mu}{\sigma} \Rightarrow \boxed{\mu=0,\ \sigma=1}$$

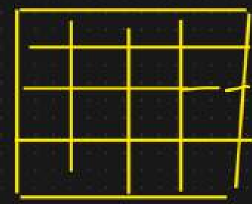
$f_1 \longrightarrow f_1' \Leftarrow \mu=0, \sigma=1$

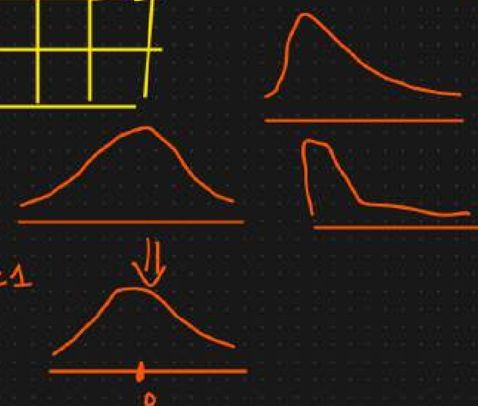


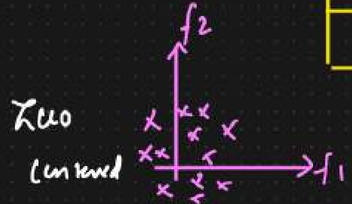
Deep learning : I/p Image $\Rightarrow$ Min Max Scaler



$\Rightarrow$ Min Max Scaler $\Rightarrow$







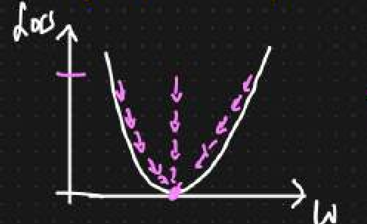
Advantages

① Improved Training Stability $\begin{Bmatrix} \mu=0 \\ \sigma=1 \end{Bmatrix}$ $\mu=0, \sigma=1$

$\Downarrow$

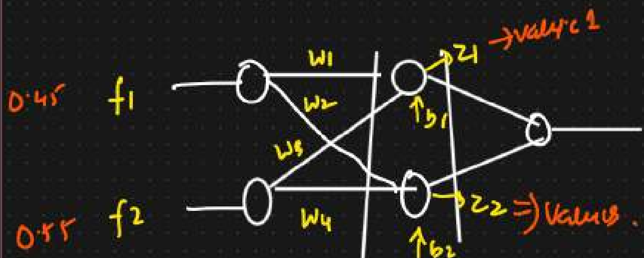
Vanishing and exploding gradient problem

② Faster Convergence



$\Rightarrow$ Back propogation $\Rightarrow$ Stable updates.

{Batch Normalization}



f1 House size	f2 Rooms	Price	Z_1 (Normalization)	Z_2 (Standardization)
0.45	0.55	45	—	—
0.60	0.20	—	—	—
—	—	—	—	—

$$Z_1 = \sigma\left[(0.45 * w_1 + 0.55 * w_3) + b_1\right] = \text{Value 1}$$

$$Z_2 = \sigma(\qquad + b_2) = \underline{\text{Value 2}}$$

} Does this distribution $\boxed{\mu=0, \sigma=1}$

μ_1, σ_1 μ_2, σ_2

Batch Normalization vs Layer Normalization

f_1	f_2	Z_1	Z_2	
—	—	—	—	$\rightarrow \mu_1, \sigma_1$
—	—	—	—	$\rightarrow \mu_2\ \sigma_2$
—	—	—	—	$\rightarrow \mu_3\ \sigma_3$

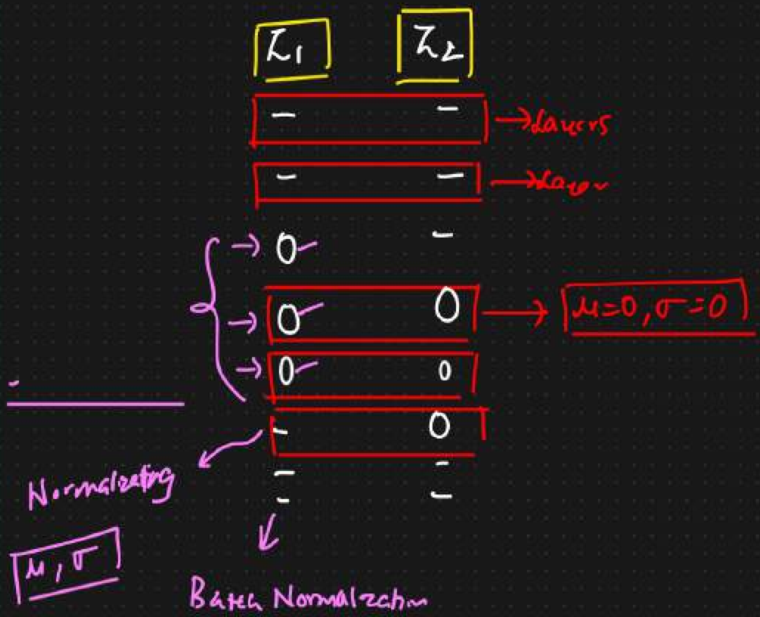
$Z_{score} = \frac{x_i - \mu_1}{\sigma_1}$

$Z_{score} = \frac{x_i - \mu_2}{\sigma_2}$

$Z_{score} = x_i - \mu_3 / \sigma_3$ ..

} Layer Normalization

$\gamma, \beta \rightarrow$ learnable parameters



Normalization

$\Downarrow$

γ, β

$$z_1 = \sigma\left[w_1^T x + b_1 \right]$$

learnable parameters

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

$\Downarrow$

Scale And Shift parameters

1) "CAT" = [2.0, 4.0, 6.0, 8.0] ← {Vectors} → 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

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

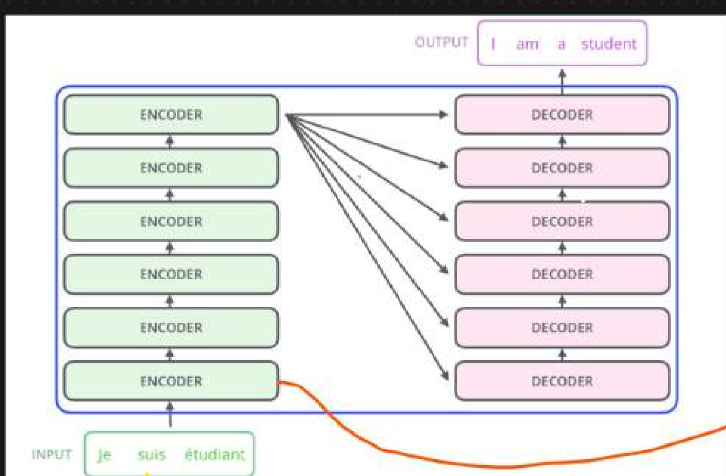
4) Scale And Shift

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

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

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

⑥ Encoder Architecture [Research Paper]



Sequence to Sequence ⇓ Complex

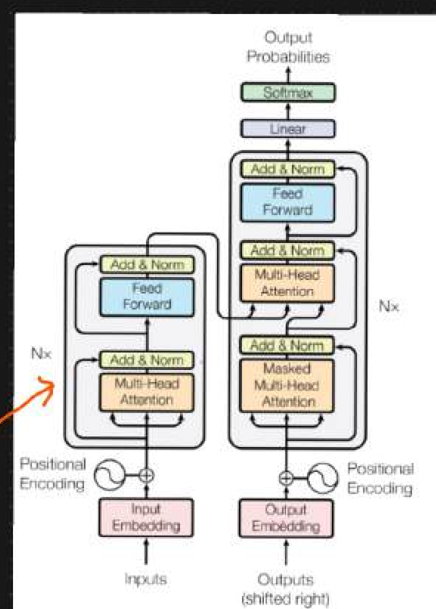


Figure 1: The Transformer - model architecture.

Feed Forward NN. → o o o o o o o o o o o o

↑

(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.

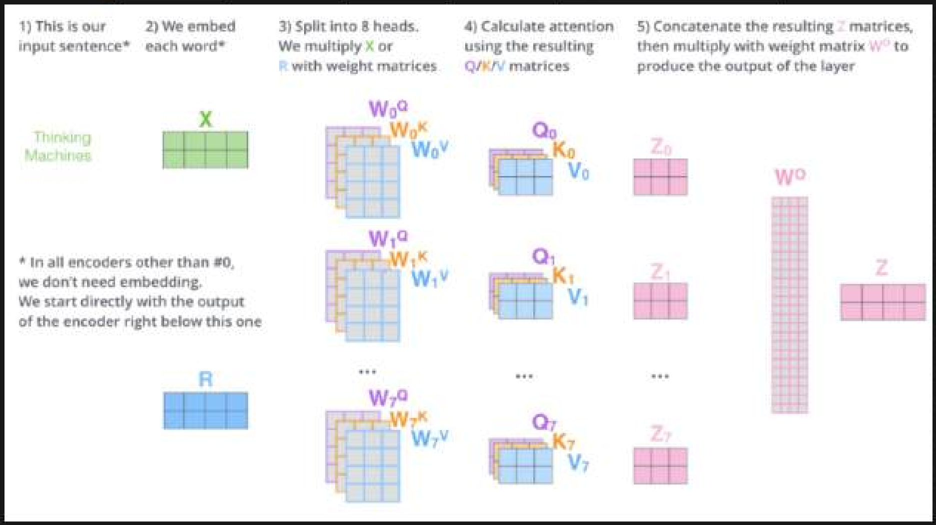
$Q_1 = 64$

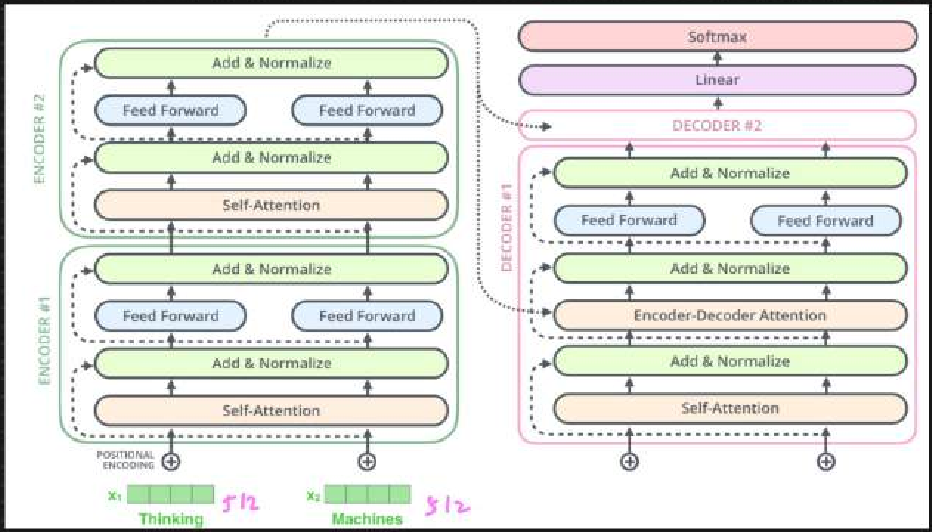
$K = 64$

$V = 64$

$\sqrt{64} = 8$ //.

Self Attention to Feed Forward Neural N/W





Embedding Vector: 512

Q, K, V = 64

Head Attention: 8

Ⓡ Residual connection : Skip connection NN.

1) Addressing the Vanishing Gradient Problem

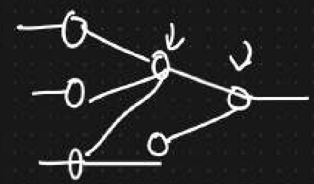
Residuals : Residual connection create a short paths for gradients to flow directly through the n/w. Gradient remains sufficiently large.

2) Improve Gradient flow

Convergence will be faster.

3) Enables Training of Deeper Networks.

④ Feed Forward NN



Linear function problem

{Non Linear function}.

① Adding Non linearity

② Processing Each position Independently.

Self Attention ⟶ Capture relationships

FFN ⟶ Each token representation Independently.

⇓

Transforming these representation furthers and allows the model to learn Richer Representation.

} ANN ⇒

③ FFN ⟶ Deeper ⇒ Adds Depth to the Model.

Depth ↑↑ ⇒ More learnings ⟶ DATA

④ 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.

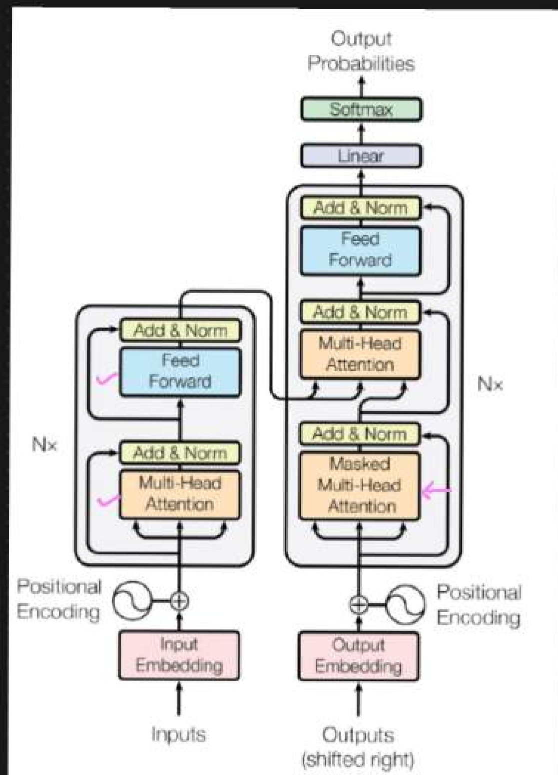
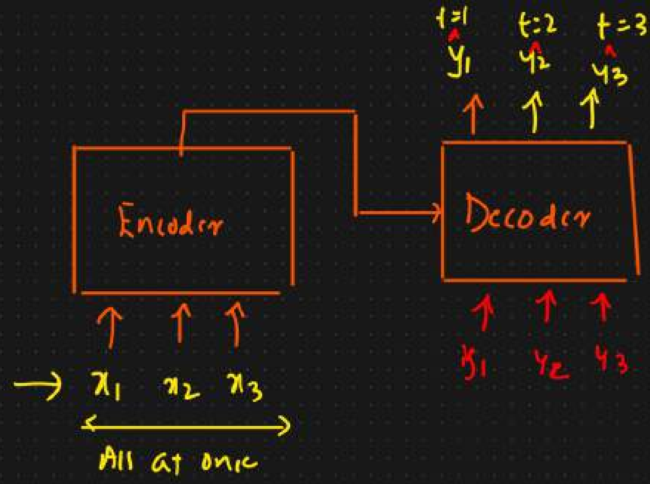


Figure 1: The Transformer - model architecture.



$\langle y_1, y_2, y_3 \rangle$

1. Training Mechanism
2. Inference Mechanism

(*) 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

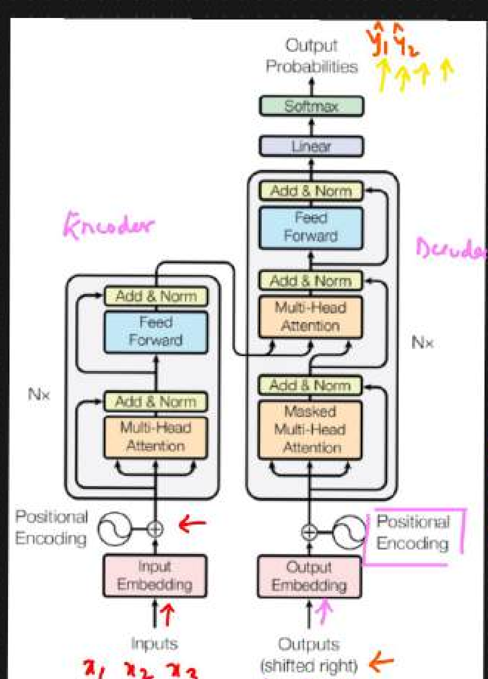


Figure 1: The Transformer - model architecture.

Dataset

Eng	Hindi
$\langle x_1 \cdot x_2 \ x_3 \rangle$	$\langle y_1 \ y_2 \rangle$ ←

⇓

$\langle y_1, y_2 \ 0 \rangle$

↓

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.

I/P $[4\ 5\ 6\ 7]$ O/P $[1\ 2\ 3]$

$[1\ 2\ 3\ 0]$

4 dimension vector

i) 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], + P.E ⇒ 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 = [[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]]

$$K^T: \begin{bmatrix}0.1\\0.2\\0.3\\0.4\end{bmatrix} \begin{bmatrix}0.5\\0.6\\0.7\\0.8\end{bmatrix} \begin{bmatrix}0.9\\1.0\\1.1\\1.2\end{bmatrix} \begin{bmatrix}0\\0\\0\\0\end{bmatrix}$$

③ Scaled Dot Product Attention Calculation

$$\text{Scores} = Q * K^T / \sqrt{d_K}$$
$$= Q * K^T / 2.$$

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

[ ]
[ ]
[ ]]

④ 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 diffuent 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, $\boxed{0}$] 0 is the padding token

→ Influence the Attention Mechanism

⇓

Lead to Incorrect or biased predictions.

→100.

O/P

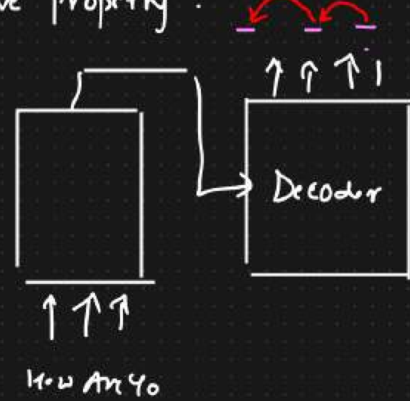
→ 4, 42 0 0000

A padding mask ⇒ The tokens Are ignored.

Masking → Padding Mask, Look Ahead Mask.

Padding mask [1 1 1]
[1 1 0]

(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

(2) Sequence → Language Modelling, Translation

Eg: $[4, 5, 0] \rightarrow [1, 1, 0]$ → 1 D Mask.

Convert 1D to 2D Mask

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

Token 1 attends to token 1, 2

1, 2

Attention Mechanism ⇐

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

(*) Look Ahead Mask → Decoder Output

$$[[1 \quad 0 \quad 0]\ [1 \quad 1 \quad 0]\ [1 \quad 1 \quad 1]]$$

(4) Combine Padding And Looking Ahead Mask

Element wise multiplication of 2 mask

Combine Mask = $[[1, 0, 0]\ [1, 1, 0]\ [0, 0, 0]]$

④ Mask

Scores :

$$\left[\begin{array}{l}[0.3,\ 0.7,\ 1.1,\ 0.0] \\ [0.7,\ 1.9,\ 3.1,\ 0.0] \\ [1.1\ \ 3.1\ \ 5.1\ \ 0.0] \\ [0.0\ \ 0.0\ \ 0.0\ \ 0.0]\end{array}\right]$$

Look Ahead Mask

$$\left[\begin{array}{l}[1\ \ 0\ \ 0\ \ 0] \\ [1\ \ 1\ \ 0\ \ 0] \\ [1\ \ 1\ \ 1\ \ 0] \\ [1\ \ 1\ \ 1\ \ 1]\end{array}\right]$$

Padding Masking [extended to 2 D Format]

$$\left[\begin{array}{l}[1,1,1,0] \\ [1,1,1,0] \\ [1,1,1,0] \\ [0,0,0,0]\end{array}\right]$$

Combined Mask : Look Ahead Mask * Padding Mask

$$\left[\begin{array}{l}[1*1,\ 0*1,\ 0*1,\ 0*0] \\ [1*1,\ 1*1,\ 0*1,\ 0*0] \\ [1*1,\ 1*1,\ 1*1,\ 0*0] \\ [1*1,\ 1*1,\ 1\times1,\ 1*0]\end{array}\right] = \left[\begin{array}{l}[1,0,0,0], \\ [1,1,0,0], \\ [1,1,1,0], \\ [1,1,1,0]\end{array}\right] \Rightarrow \left[\begin{array}{l}[1,-\infty,-\infty,-\infty]. \\ [1,1,-\infty,-\infty], \\ [1,1,1,-\infty] \\ [1,1,1,-\infty]\end{array}\right]$$

Masked Score

$$\left[\begin{array}{l}[0.3,\ -\infty,\ -\infty,\ -\infty] \\ [0.7,\ 1.9,\ -\infty,\ -\infty] \\ [1.1,\ 3.1,\ 5.1,\ -\infty] \\ [0.0,\ 0.0,\ 0.0,\ -\infty]\end{array}\right].$$

Zero out the influence when the Softmax is applied.

Attention weights.

(*) Softmax

Softmax Score = Softmax (Masked Scores)

$$= [[1.0, 0.0, 0.0, 0.0], [0.3, 0.7, 0.0, 0.0], [0.1, 0.3, 0.6, 0.0], [1.0, 0.0, 0.0, 0.0]]$$

(*) 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.

4 Encoder Decoder Multi Head Attention

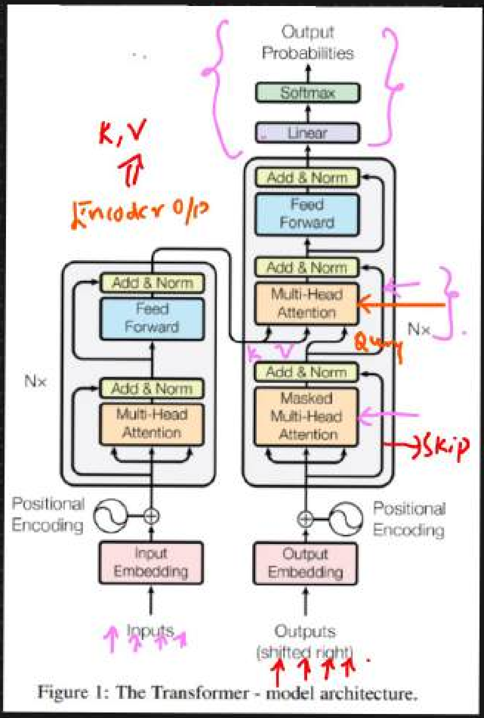


Figure 1: The Transformer - model architecture.

1. Encoder O/P → Set of Attention vector K & V
2. Masked Multihead → Attention vector Q {Query Vector}

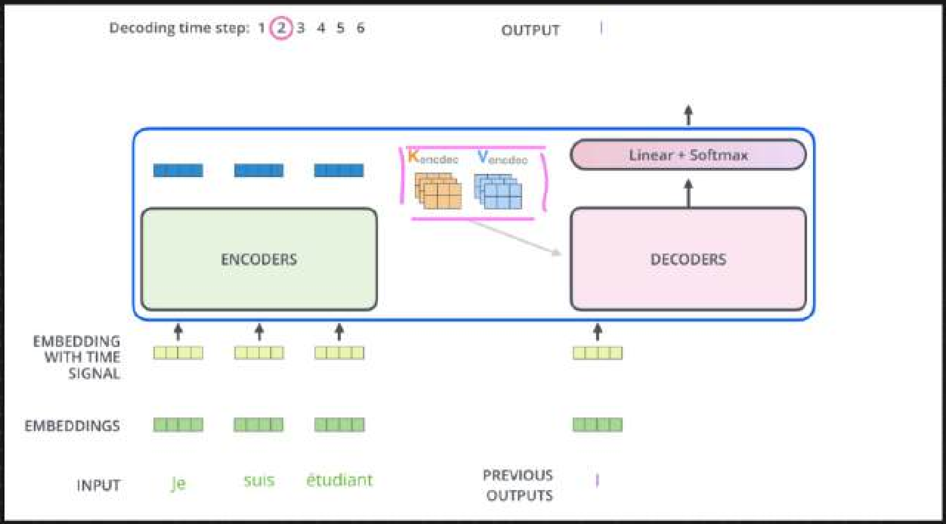
These are to be used by each decoder in its "encoder-decoder" attention layer

⇓

Helps the Decoder to focus on appropriate places in the i/p sequence

Dataset

i/p	o/p
<x₁ x₂ x₃>	<y₁ y₂ y₃>.



④ The Final Linear And Softmax Layer

{Vectors ⟶ o/p Word}

{BLOG} ⇒ Transformers

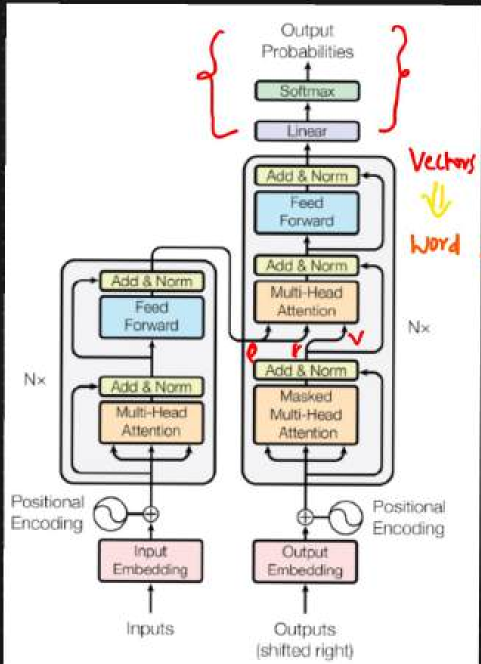
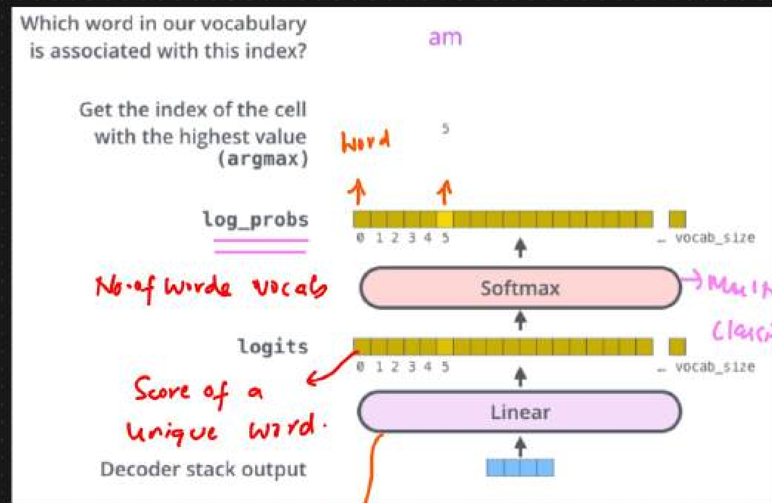


Figure 1: The Transformer - model architecture.



Linear ⇒ The linear layer is a simple fully connected neural n/w 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 upto 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

OHE ←

One-hot encoding of the word "am"

0.0	1.0	0.0	0.0	0.0	0.0

Merci → Thanks

I am a student <eos>.

Untrained Model Output

a	am	I	thanks	student	<eos>
0.2	0.2	0.1	0.2	0.2	0.1

Correct and desired output

a	am	I	thanks	student	<eos>
0.0	0.0	0.0	1.0	0.0	0.0

⇒ Loss function ↓↓.

Back Propogation

Target Model Outputs

Output Vocabulary:	a	am	I	thanks	student	<eos>
position #1	0.0	0.0	1.0	0.0	0.0	0.0
position #2	0.0	1.0	0.0	0.0	0.0	0.0
position #3	1.0	0.0	0.0	0.0	0.0	0.0
position #4	0.0	0.0	0.0	0.0	1.0	0.0
position #5	0.0	0.0	0.0	0.0	0.0	1.0

6 6

i/p — a, am, i, student

o/p — I am a student

↓ ↓ ↓ ↓

OHE OHE OHE OHE

Loss

Trained Model Outputs

Output Vocabulary:	a	am	I	thanks	student	<eos>
position #1	0.01	0.02	0.93	0.01	0.03	0.01
position #2	0.01	0.8	0.1	0.05	0.01	0.03
position #3	0.99	0.001	0.001	0.001	0.002	0.001
position #4	0.001	0.002	0.001	0.02	0.94	0.01
position #5	0.01	0.01	0.001	0.001	0.001	0.98