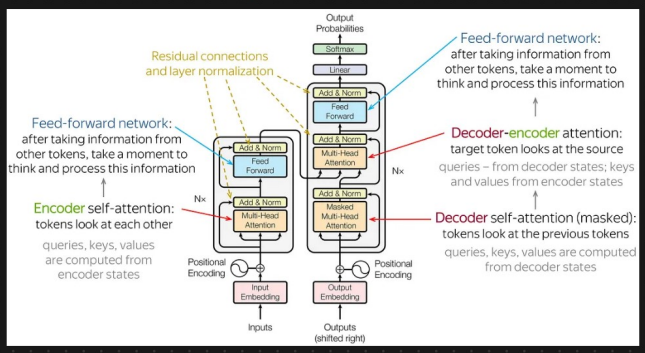
# Introduction To Transformers

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 → Chat GPT

↓

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

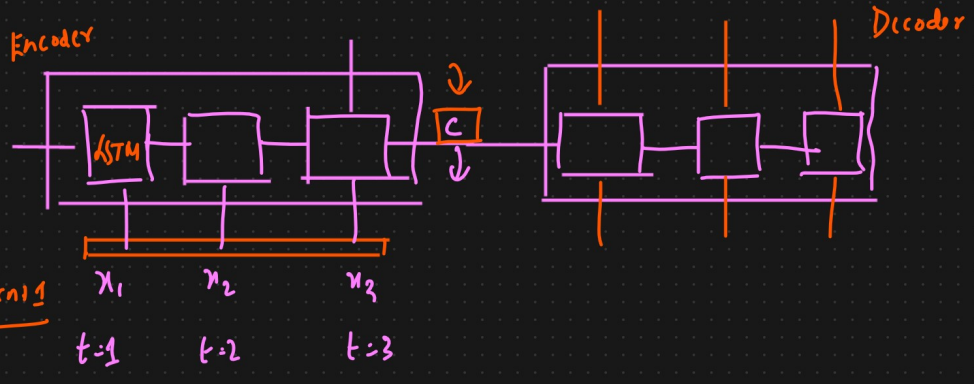
English → French

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

Encoder - Decoder

Sentence Length ↑↑ → Bleau Score ↓↓

Length Sentence ↑↑.



### 3.1 Decoder: General Description

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

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

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* $(h_1,\cdots,h_{T_x})$ 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 $h_i$:

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

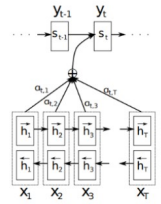
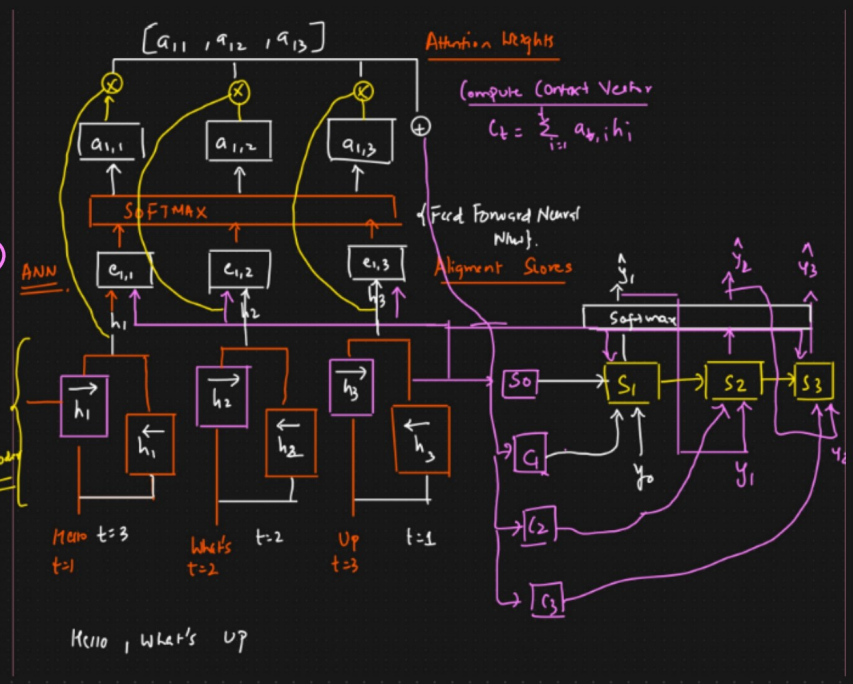


Figure 1: The graphical illustration of the proposed model trying to generate the $t$-th target word $y_t$ given a source sentence $(x_1, x_2, \dots, x_T)$.




Additional Context ⟶ Decoder

long Sentence. Accuracy ↑↑.

## Attention Mechanism ↓

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

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

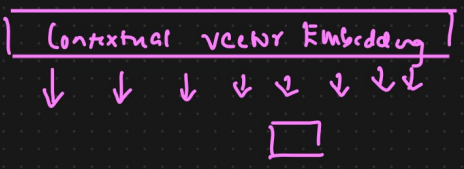
↑

Train Huge Data

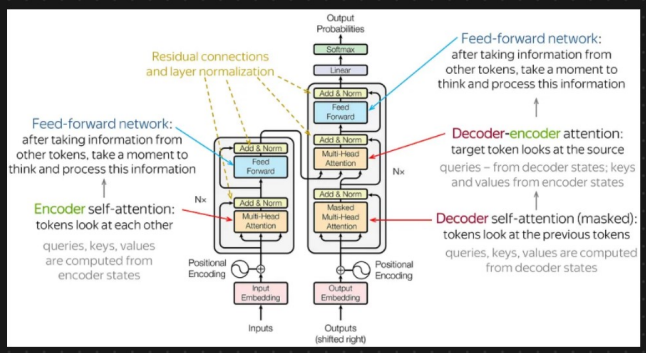
## ② Contextual Embedding → Self Attention — Contextual Vectors

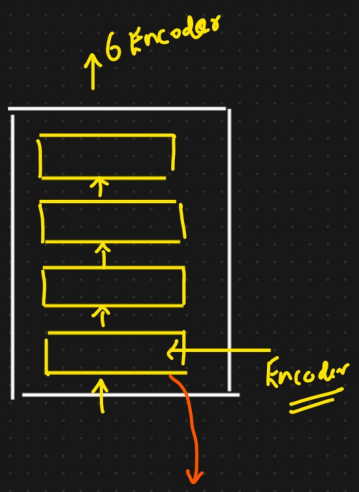
Eg: My name is KRISH And I play CRICKET

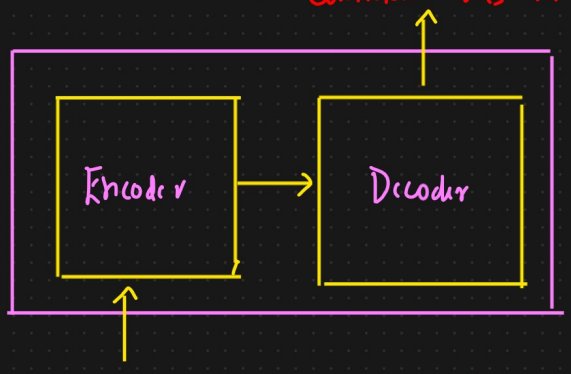
Word2Vec

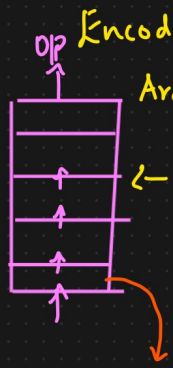




## ② Basic Transformer Architecture {Seq2Seq Task} → Language Translation {Eng → French}

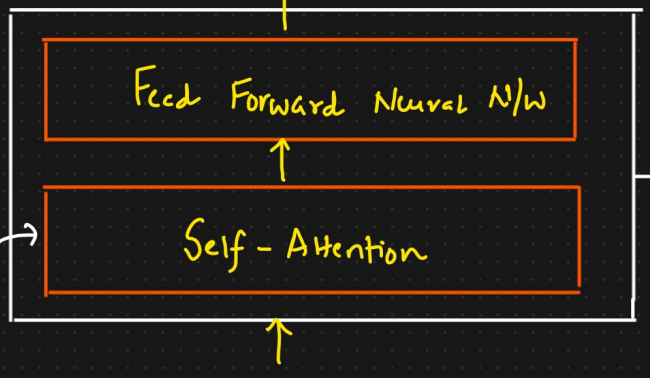



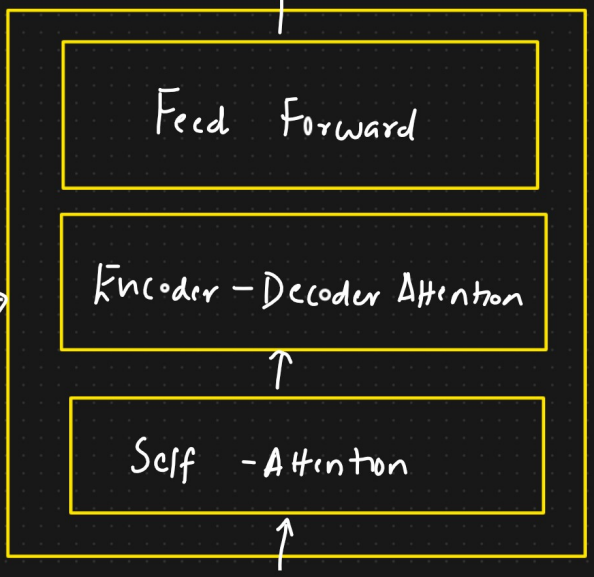



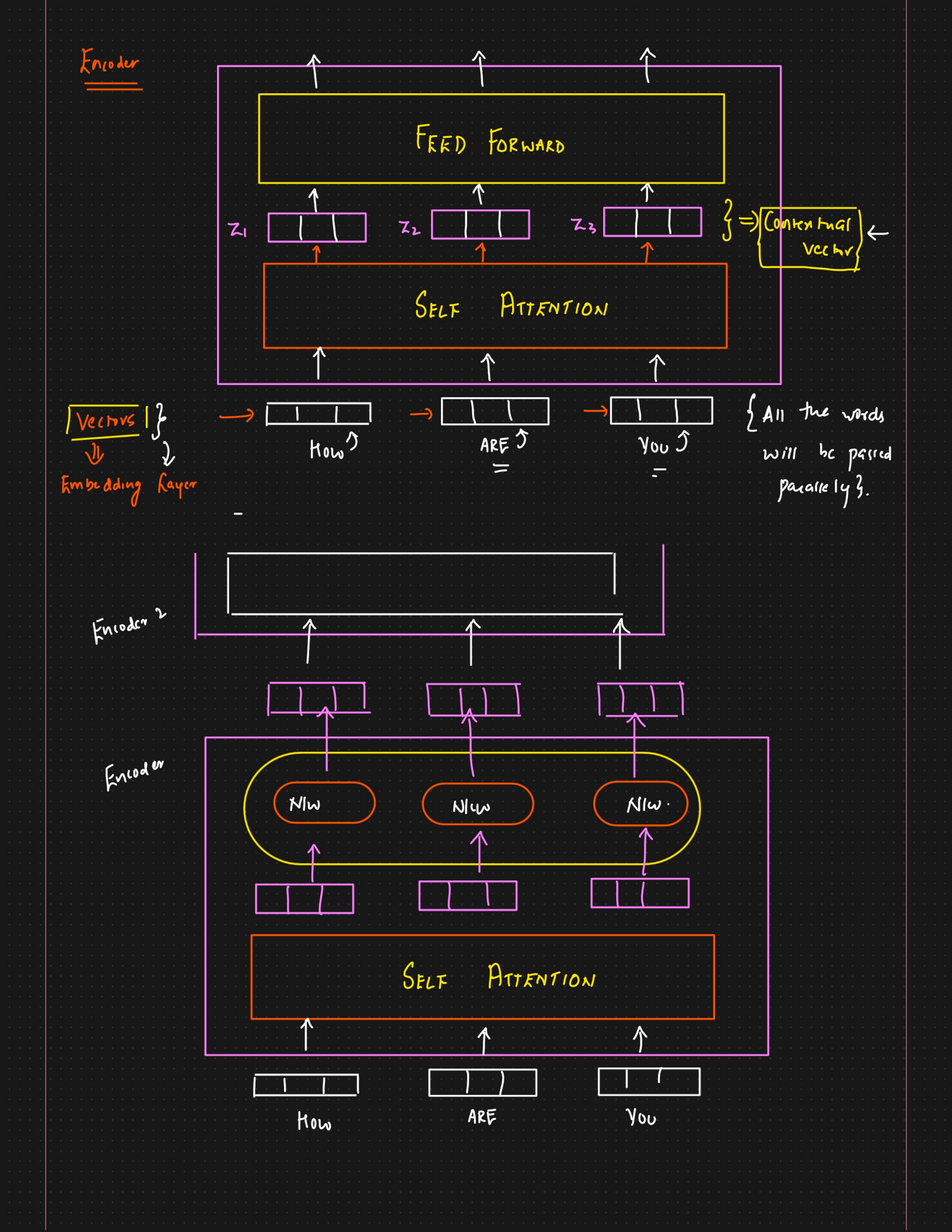

Encoder




{LSTM RNN

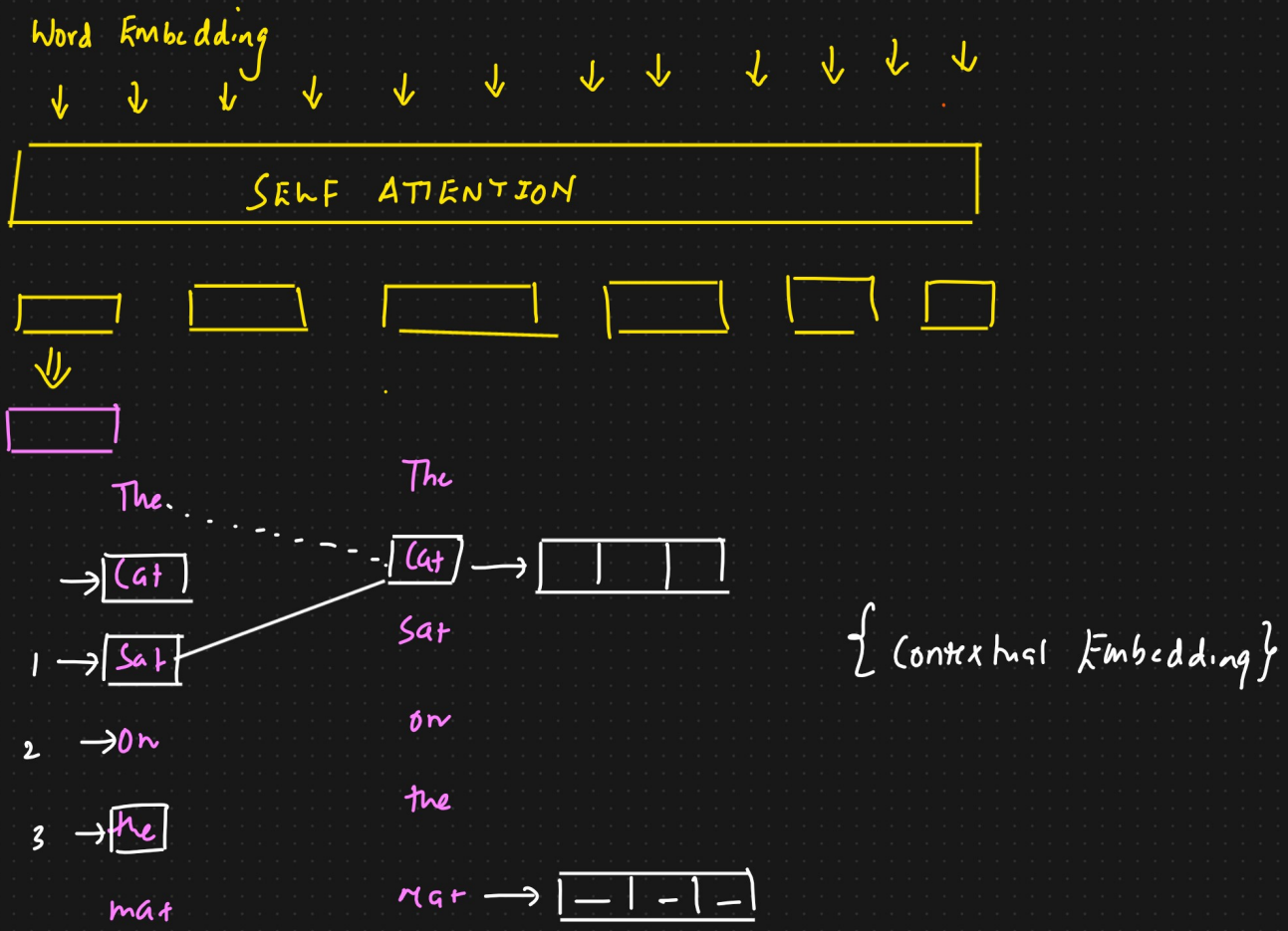
Decoder

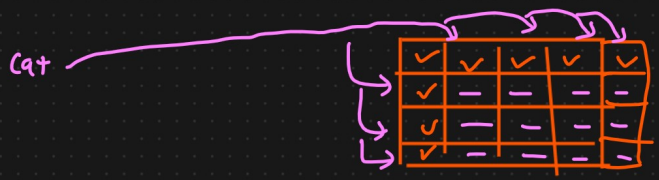



Encoder
FEED FORWARD
Z1
Z2
Z3
Contextual vector
SELF ATTENTION
Vectors
Embedding Layer
How
ARE
You
All the words will be passed parallely.
Encoder 2
Encoder
NN
NN
NN
SELF ATTENTION
How
ARE
You

# Self Attention At a Higher Level

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



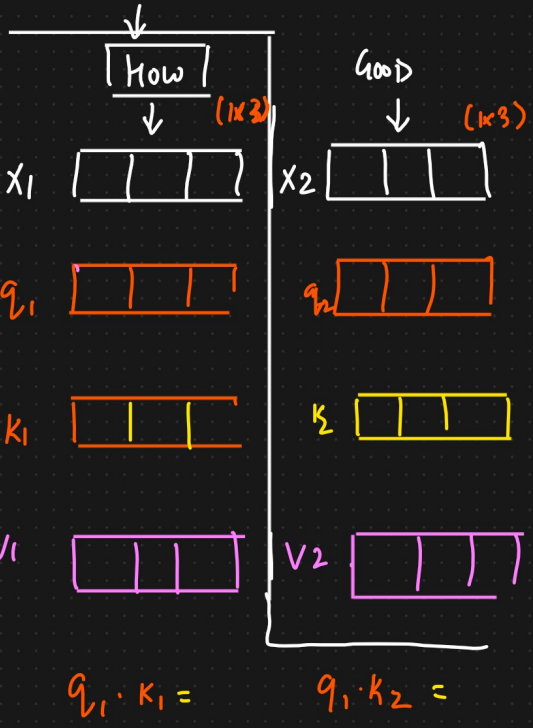

## Self Attention In Detail

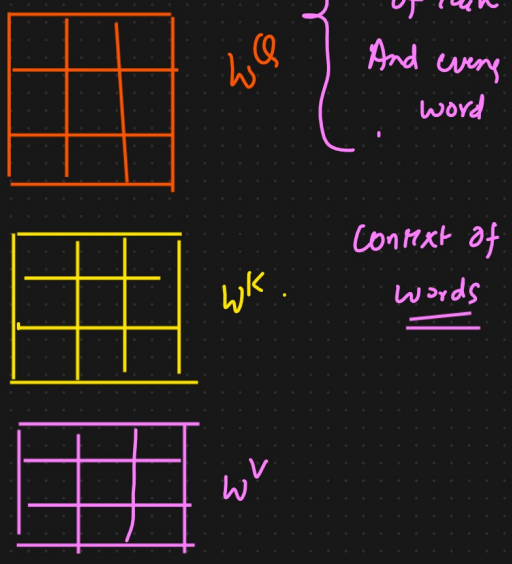
① To create 3 vectors from each of the encoder i/p. Query vector, Key vector, value vector. ⇒ Contextual Embedding

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 sentence.




(*) 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 ↓ Variance / $\sqrt{dk}$ ≈ Variance ↑↑↑ / $\sqrt{dk}$ ⇒

Dimension ↑↑ T12

O/p ↑↑↑

{More stable gradients}

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 $V_1$ [1 1 1] $V_2$ [1 0.6 1]

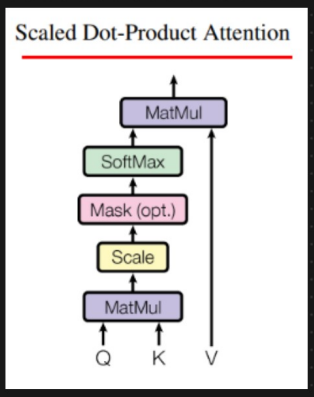
0.88 [1 1 1]

[0.88 0.88 0.88] + [0.12 0.6 0.12]

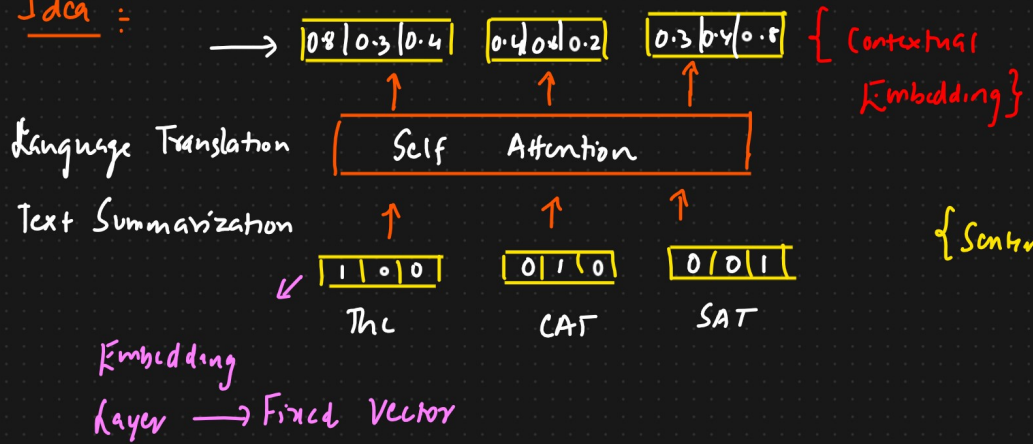
Contextual vector $Z_1$ → [ ] $Z_2$ → [ ]

# 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 :




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.**

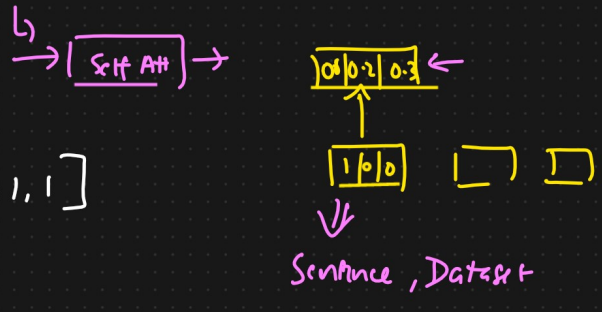
Importance:

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"]

Embedding Size = 4

Q, K, V ⇒ 4




① Token Embedding

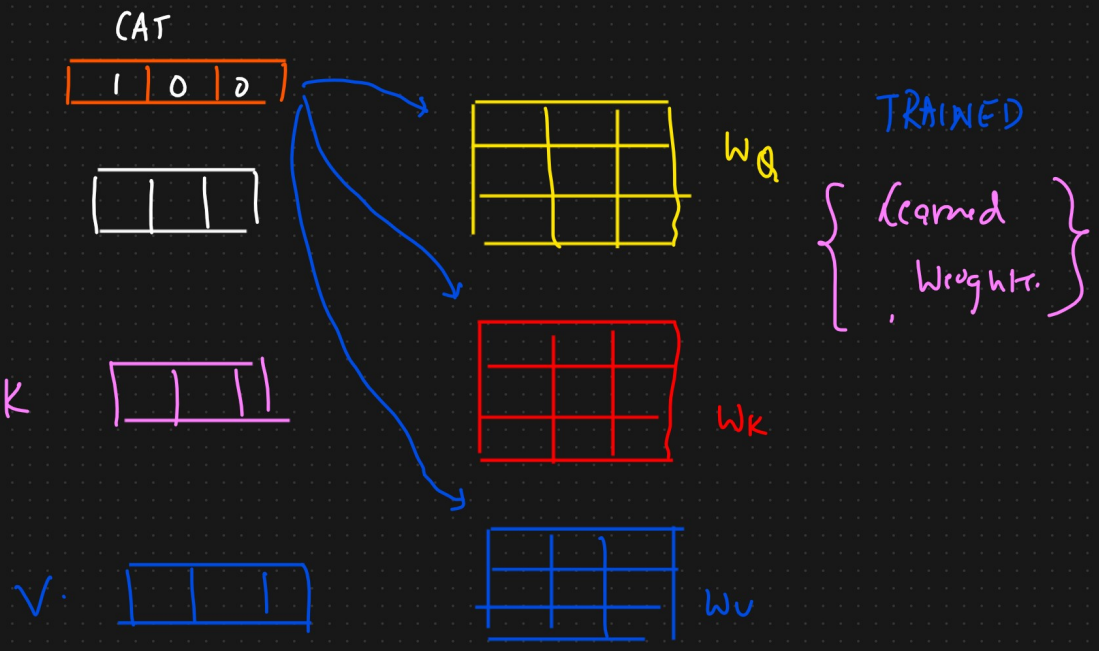
$E_{The} = [1\ 0\ 1\ 0]$

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

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

## 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

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

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

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

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

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

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

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

## ③ Compute Attention Scores

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

The

$$\text{Score}(Q_{The}, K_{The}) = [1010]\cdot[1010]^T = 2$$

$$\text{Score}(Q_{The}, K_{CAT}) = [1010]\cdot[0101]^T = 0$$

$$\text{Score}(Q_{The}, K_{SAT}) = [1010]\cdot[1,1,1,1]^T = 2$$

For the token CAT

$$\text{Score}(Q_{CAT}, K_{The}) = [0101]\cdot[1010]^T = 0$$

$$\text{Score}(Q_{CAT}, K_{CAT}) = [0101]\cdot[0101]^T = 2$$

$$\text{Score}(Q_{CAT}, K_{SAT}) = [0101]\cdot[1111]^T = 2.$$

For the Token SAT

$$\text{Score}(Q_{SAT}, K_{The}) = [1,1,1,1]\cdot[1010]^T = 2$$

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

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

## ④ Scaling ÷

We take up the scores and scale down by dividing the scores by the $\sqrt{d_k}$ $\Rightarrow$ $d_k = 4$ $\sqrt{d_k} = 2$.

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 $W_Q, W_K, W_V$

$([10, 1]) = [0.99, 0.01]$

Dot product = Large values

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

## With Scaling

① Compute Scaled Dot Product

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

dimension ↑ Variance ↑2

$\sqrt{d_k}$ ↑ same

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

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

$= [0.73, 0.27]$ ⇒ 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 =

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

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

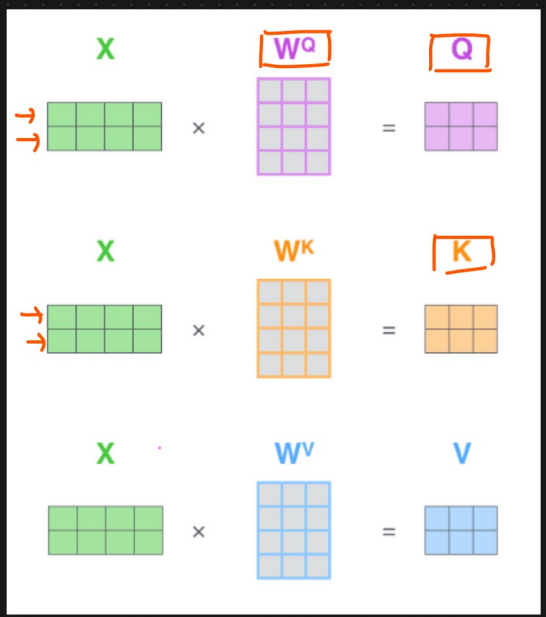
$$= [0.4223, 0, 0.4223, 0] + [0, 0.1554, 0, 0.1554] + [0.4223, 0.4223, 0.4223, 0.4223]$$

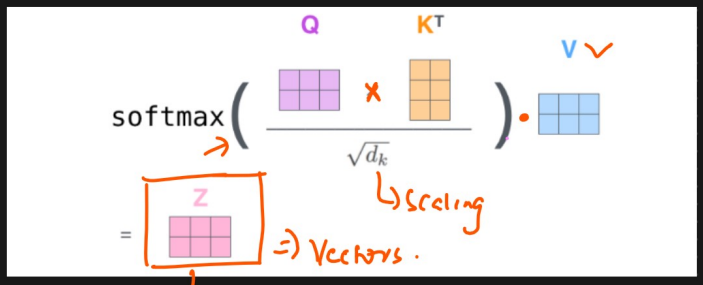
$$= [1.2669, 0.9999, 1.2669, 0.9999].$$

The $\boxed{1\,0\,1\,0}$ ⇒ Self Attention ⇒ [1.2669, 0.9999, 1.2669, 0.9999]. ⇓ Contextual vector

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

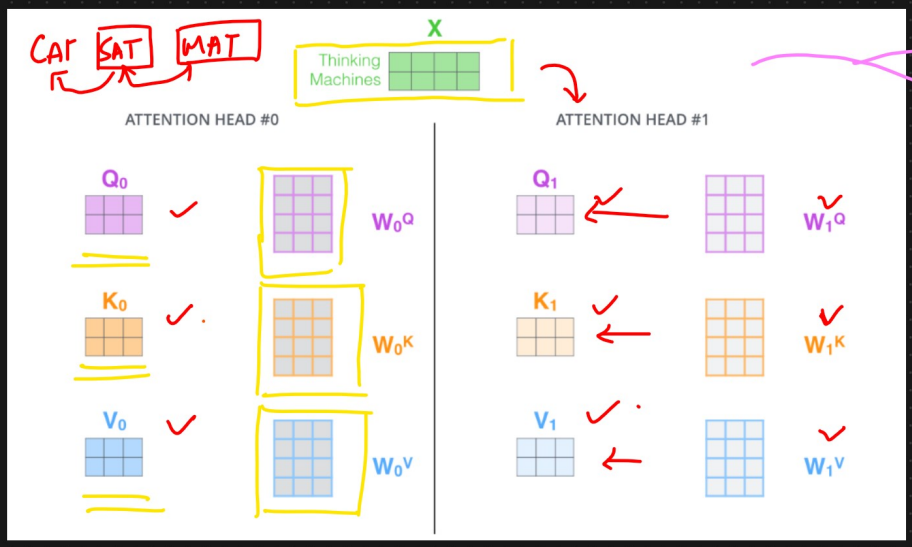
④ Multi Head Attention






Attention Head.

## → Self Attention with Multi Heads




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

Score, Softmax X V

[ _ _ _ ]

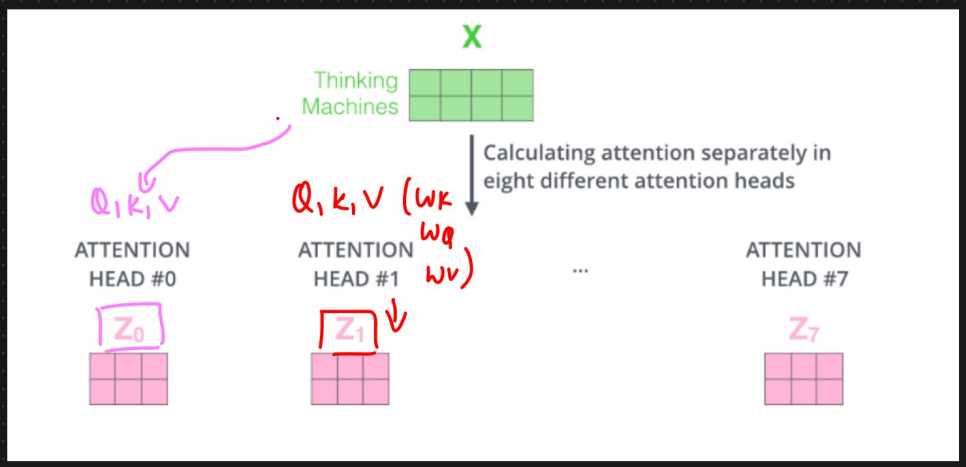
$Z_0$

Vectors

[ _ _ _ ]

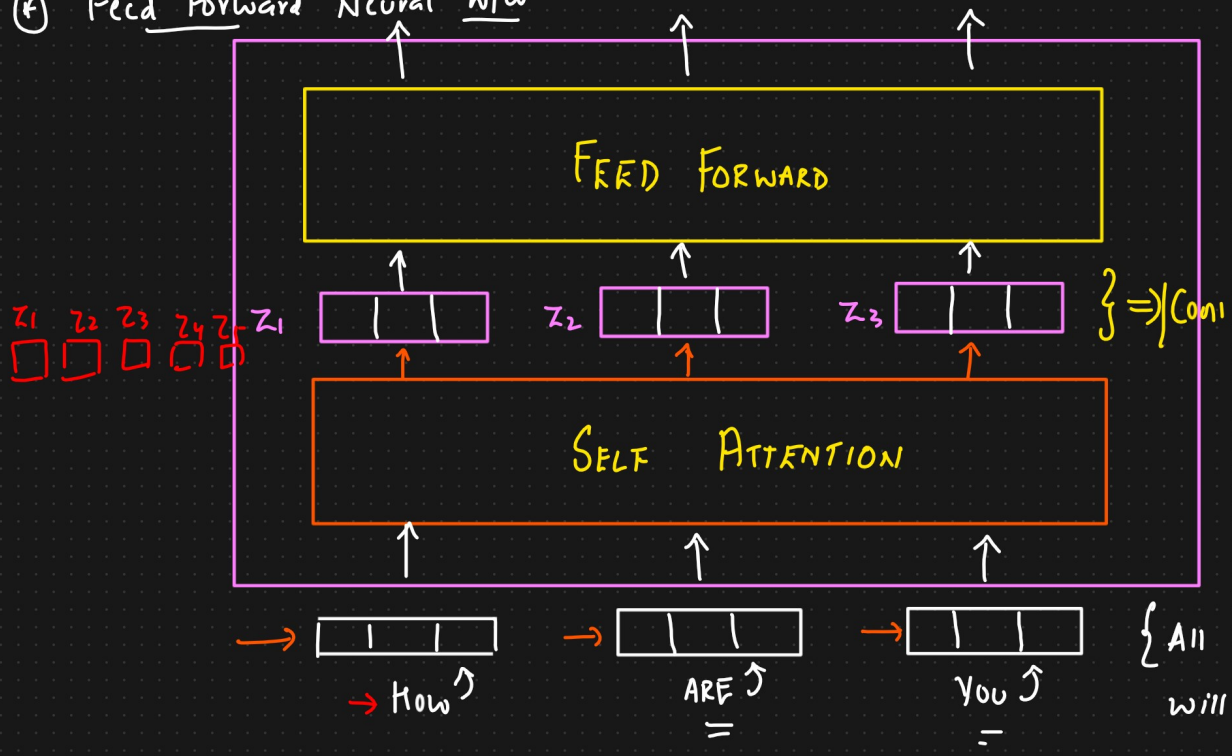
$Z_1$

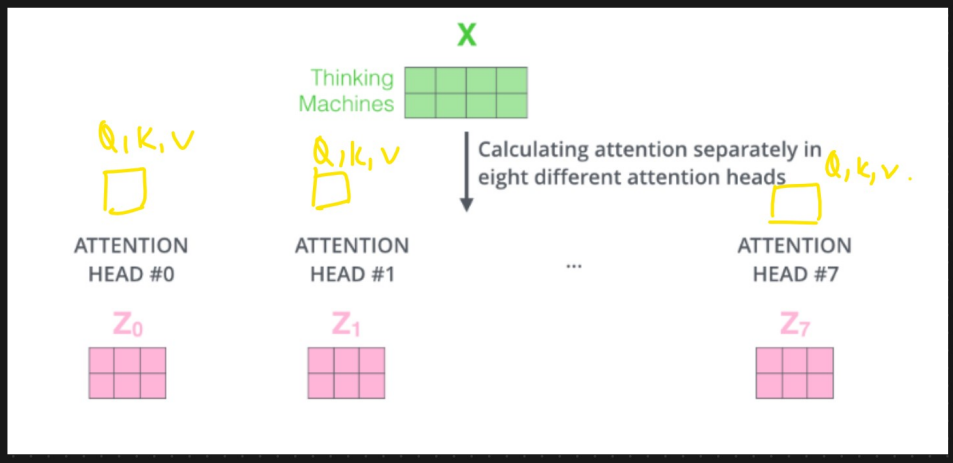
### Multi Head Attention

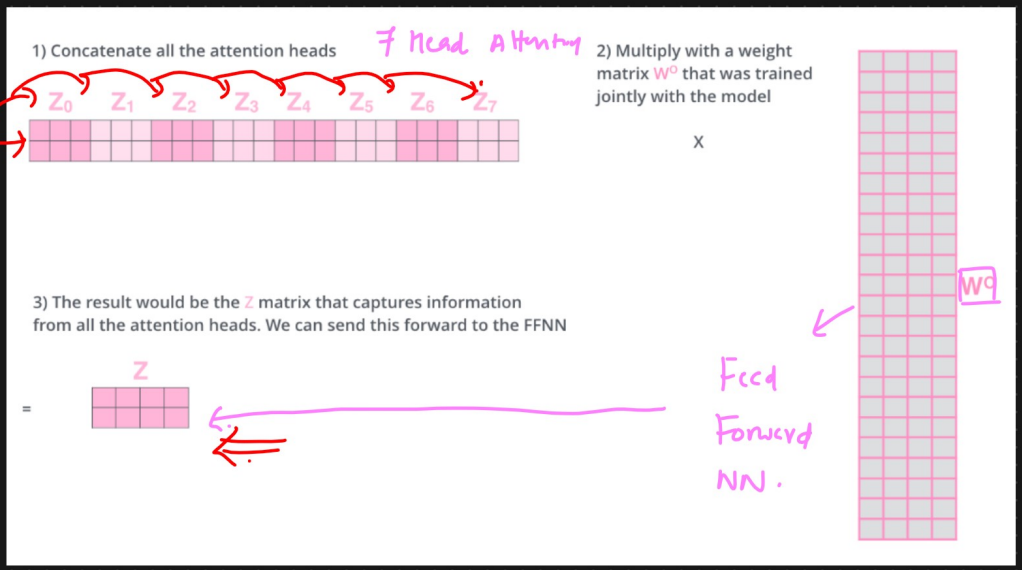


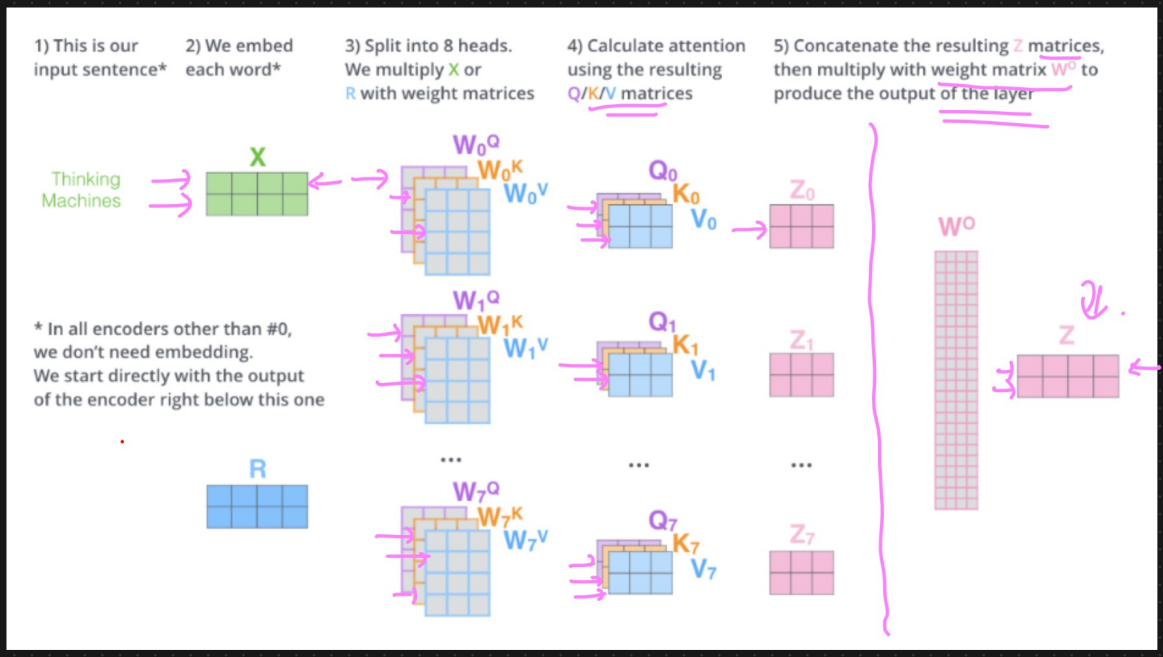

Z Z Z

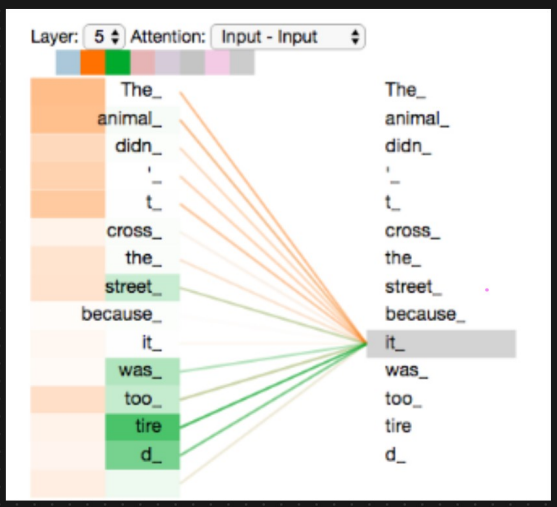
## (4) Feed Forward Neural N/W

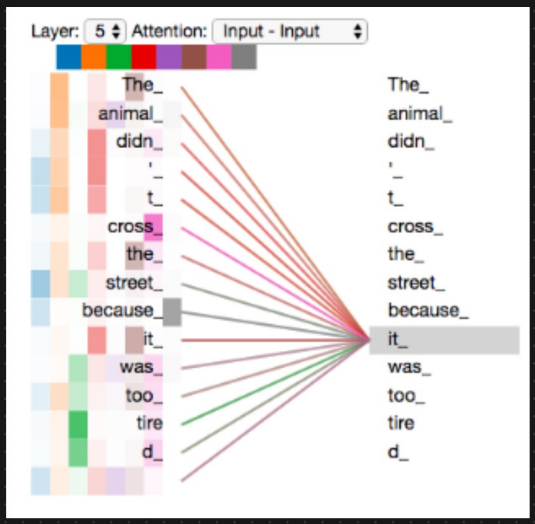




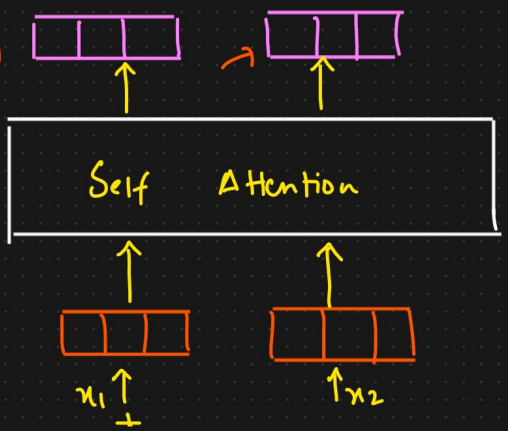




## ⑥ Positional Encoding – Representing Order of Sequence

① Lion KILLS TIGER
② TIGER KILLS LION

⇓

Missing




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 Vectors

Book, Journal, Novel
↳ 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 sine and cosine functions of different frequencies to create positional encodings

Formula :

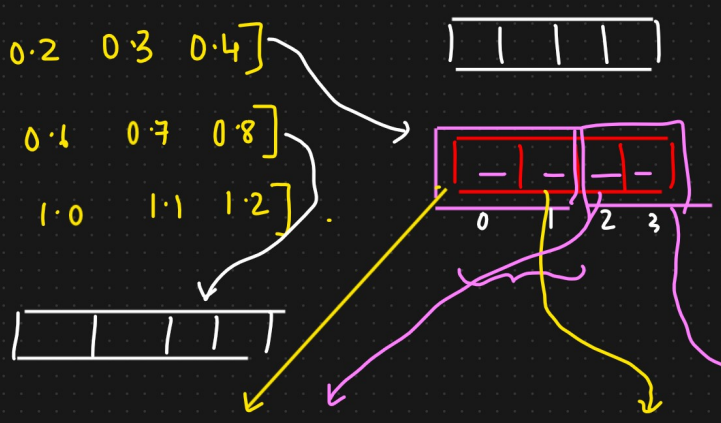
$$P.E(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

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

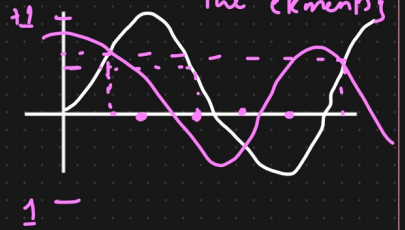
Where pos is the position
$i$ is the dimension
$d_{model}$ is the dimensionality of the embeddings.

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]




{ Miss the order of the elements }



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

For our example $d_{model} = 4$

For position $pos = 0$

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

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

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

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

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

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

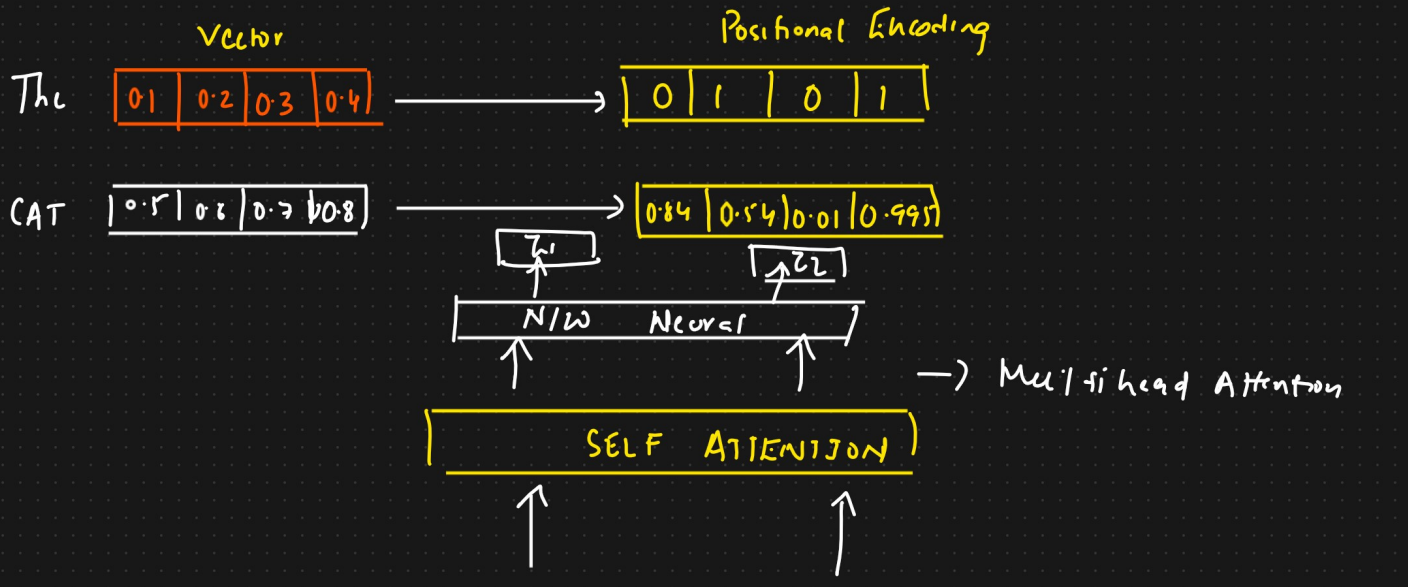
For $pos = 1$

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

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

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

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



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

+

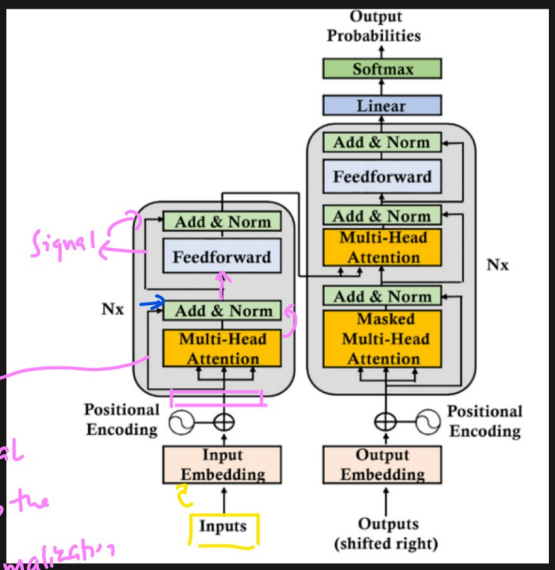
0.1 | 0.2 | 0.3 | 0.4

↑ The.

# ⑨ Layer Normalization In Transformers

## Transformers

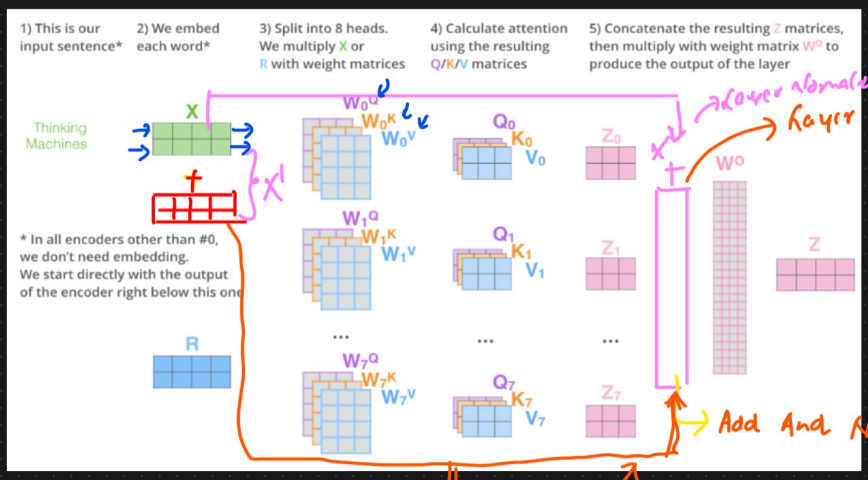
## Residuals




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 |

## Normalization : Standard Scaling

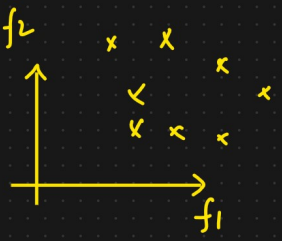
$\mu, \sigma$ ← | 2000 | 3.5 | 80

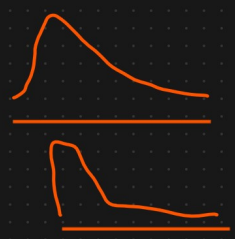
$$Z_{score} = \frac{x_i - \mu}{\sigma} \Rightarrow \boxed{\mu = 0, \sigma = 1}$$

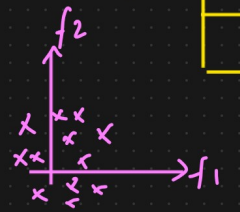
$f_1 \longrightarrow f_1' \Leftarrow \mu = 0, \sigma = 1$

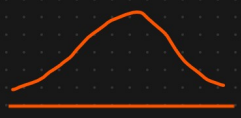
## Deep learning : I/p Image ⇒ Min Max Scaler


⇒ Min Max Scaler ⇒





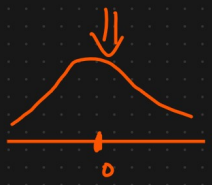





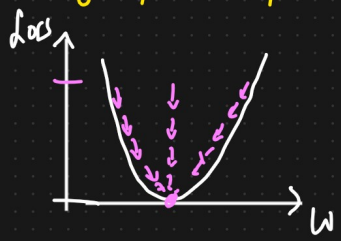
### Advantages

① Improved Training Stability ⇓ Vanishing and exploding Gradient problem

$\{\mu = 0, \sigma = 1\}$ $\mu = 0, \sigma = 1$



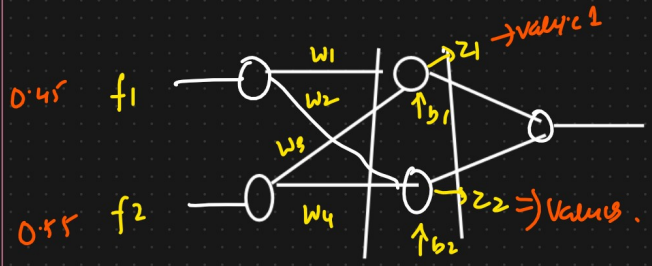

② Faster Convergence




⇒ Back propogation ⇒ Stable updates.

{Batch Normalization}

Normalize Normalized




| | $f_1$ House size | $f_2$ Rooms | Price | $Z_1$ = | $Z_2$ = |
|---|---|---|---|---|---|
| → | 0.45 | 0.55 | 45 | — | — |
| | 0.60 | 0.20 | — | — | — |
| | — | — | — | — | — |

$Z_1 = \sigma[(0.45 * W_1 + 0.55 * W_3) + b_1] = \text{Value 1}$ } Does this distribution

$Z_2 = \sigma( \quad + b_2) = \text{Value 2}$ }. $\boxed{\mu = 0, \sigma = 1}$ $\mu_1, \sigma_1$ $\mu_2, \sigma_2$

## Batch Normalization Vs Layer Normalization

| $f_1$ | $f_2$ | $Z_1$ | $Z_2$ | |
|---|---|---|---|---|
| — | — | — | — | → $\mu_1, \sigma_1$ |
| — | — | — | — | → $\mu_2 \sigma_2$ |
| — | — | — | — | → $\mu_3 \sigma_3$ |

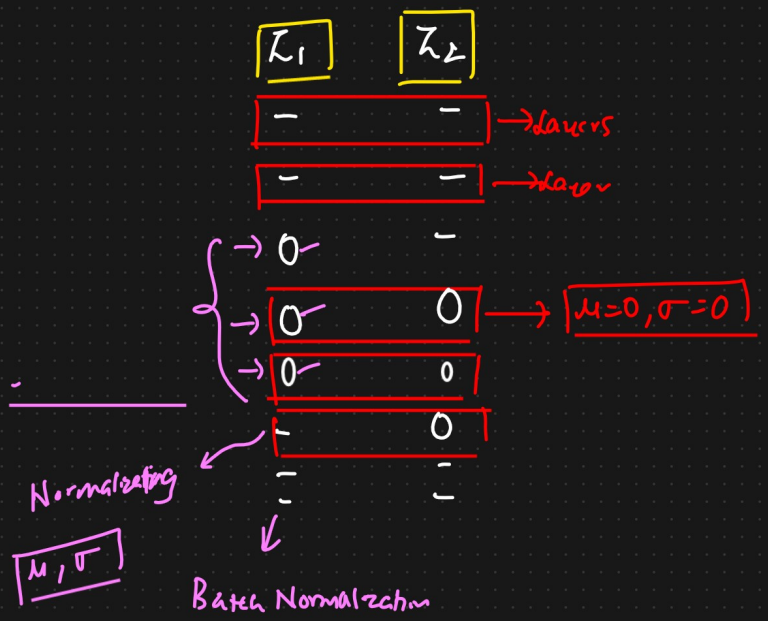
$Z_{score} = \frac{x_i - \mu_1}{\sigma_1}$

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

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

Layer Normalization

$\gamma, \beta \rightarrow$ learnable parameters




Normalization

$\gamma, \beta$

$Z_1 = \sigma[w_1^T x + b_1]$

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

learnable parameters

Scale And Shift parameters

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

2) Parameters = $\gamma$ = [1.0, 1.0, 1.0, 1.0] → Learned Scale

$\beta$ = [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 ($\sigma^2$)

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

iii) Normalize the i/p

$\hat{x_i} = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$ $\quad \epsilon = 1e^{-5}$ ⇒ 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

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

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

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