416. Plan of Action

1. Why Transformers
2. Architecture of Transformers
3. Self-Attention → Q K V
4. Positional Encoding
5. Multi-Head Attention
6. Combining the working.

417. What and Why To use Transformers!

Transfomers :- In NLP are type of deep learning model that use Self-Attention mechanisms to analyze and process natural lang. Data. They are Encoder and Decoder models that can be used for many applications, including machine Translation. → Specifically sequence to sequence Task

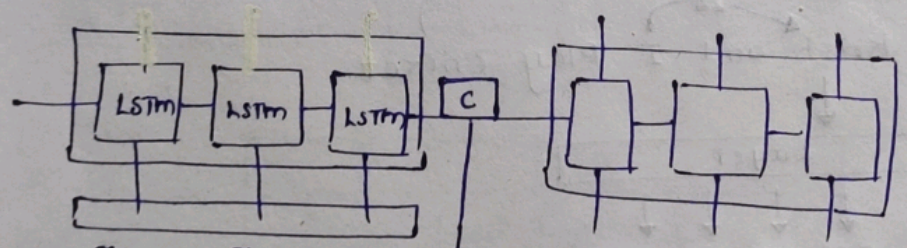
Eg :- Lang. Translation → Google Translation ✓

English → French

i/p many → o/p many

Length of sentence also meaningful Here! Better for Accuracy.

Encoder - Decoder!



Sent1 x_1 x_2 x_3

$t=1$ $t=2$ $t=3$

→ Represent the entire sentence! Not sufficient to Represent entire sent. (CV)

If sent. length increase

1. Sentence 1 passing through Embedding layer it convert into the vector
2. Then pass it through LSTM and after that we get/Generate context vector :- Based on this we provide to next decoder Layer!

∵ Bleu score is ↓ As Length of Sentence ↑

Problem With context vector we cannot send all at once so in this type of situation we actually take the help of attention mechanism

Problem with Attention mechanisms :-

① Parallely we cannot send all the words in the sentence

Dataset → Huge → Scable wr to Training

Transformers used → Self-Attention module → All the words will be parallely send! To encoder!

Scalable ✓

Scalable

As we increases ↑↑ the data in Transformers we will get amazing state of Art models (SOTA) → specifically wr to NLP

- With the help of Transfer Learning —used in→ multi models Task → NLP + image

- Because of Transformers → AI space change → SOTA model (many)

BERT ← Transformers → GPT —Help of→ Transfer Learning —Study→ SOTA model (many)

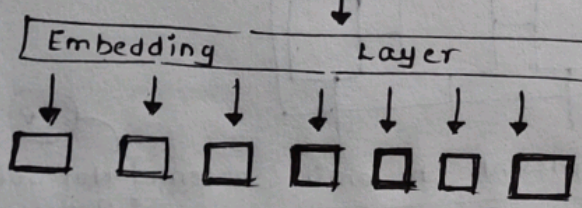
(Train with Huge Data)

② Contextual Embedding :-

Eg :- My name is Krish and I Play cricket.

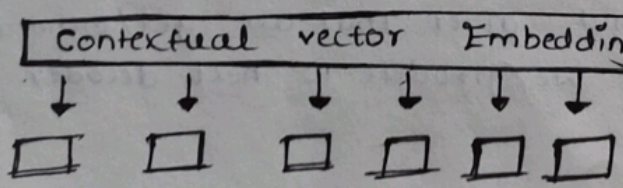
Related!

word2vec



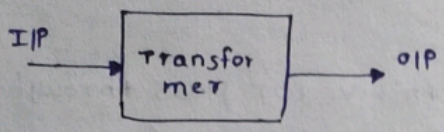
Here we will get a fix vector!

- Based on Relationship with (Krish) and (I)



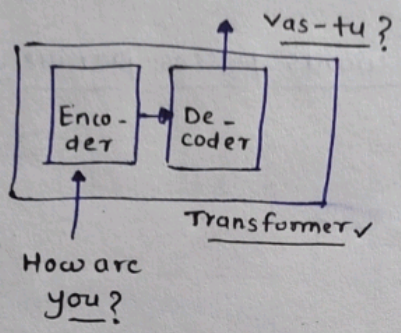
Should Have Relationship with other words!

418. Basic Transformer Architecture.

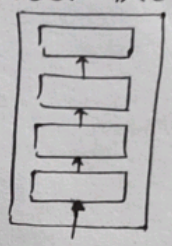


Transformer used to solve sequence to sequence Task!

Eg :- Language Translation (Eng To france)

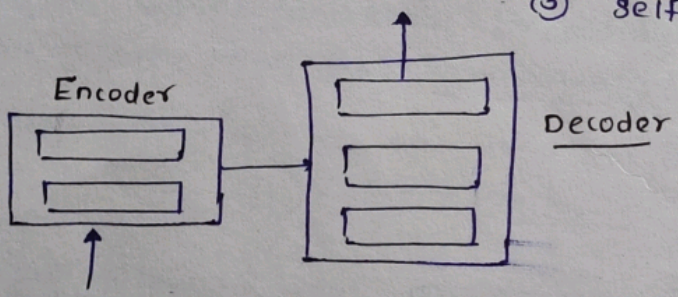


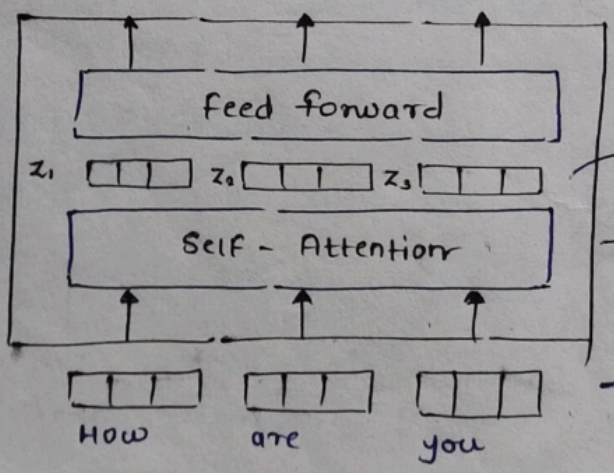
- follow Encoder - Decoder Architecture
- Inside En-Decoder we have multiple like this



mostly in Research paper Used 6 Encoder and Decoder!

- What is inside this Encoder ?
  - In Single Encoder :- ① Feed forward Neural NW ② Self - Attention
  - In single Decoder :- ① Feed forward NN ② Encoder - Decoder Attention ③ Self- Attention.

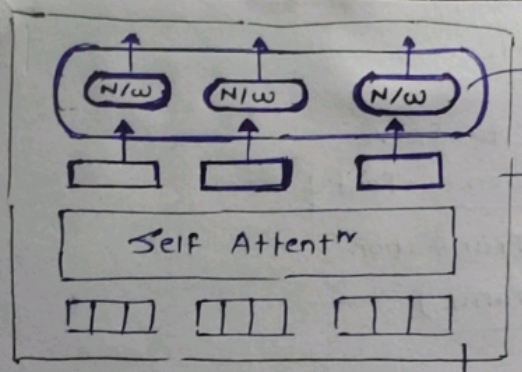




Contextual vector!

As pass this vector words into Self- Attention and it will convert this into Diff. vector! ($z_1 z_2 z_3$)

Converting all words into vector (Eg :- vector Emb.



This vector passes to another Encoder!

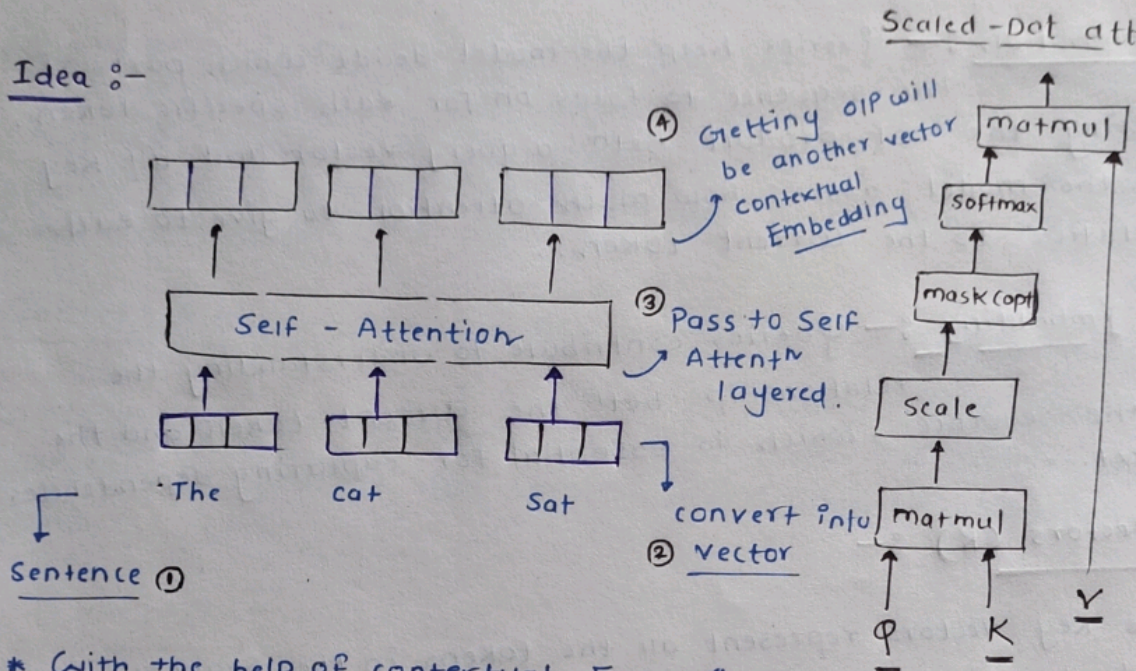
Each of this vector pass through this feed forward Neural Network

All words passes parallely!

419. Self Attention Layer Working

Self Attention, also known as Scaled-Dot product attention, is a crucial mechanism in the transformer architecture that allows the model to weight the importance of different tokens in the i/p sequence relative to each other.

Idea :-



* With the help of contextual Embedding will create that text Summarization, Lang. translation.

How Self-attention work on Scaled-Dot product attention.

Q K V ⟶ vector.

① Embedding Layer? What actually Happend :—

we get fixed vector for each and every word

④ Contextual vector? Based on context of other words

It actually Depend on entire sentence/Dataset we actually have.

How we will able to convert Embedding vector into contextual Embedding.

① Create 3 imp. vectors i/p !

ⓐ Queries ⓑ Keys ⓒ Values.

For every token if we convert it to Emb ⟶ contextual vector we specifically create a model that will compute this vectors i/p.

① Query vectors (Q) :-

→ Query vectors represent the tokens 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 Betn a query vector and all key vectors, the model assess how much attention to give to each token relative to the current token.

Contextual Embedding :- Queries contribute to understanding the relationship betn 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 calc. attentn scores.

Importance :-

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

Information Retrieval :- Keys are 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 Vector (V) :—

→ Value Vector hold the actual information that will be aggregated to form the O/P of the attentn mechanism.

Importance :—

Information Aggregation :- Values contain the data that will be weighted by the attention scores. The weighted sum of values form the O/P of the self-attentn 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 translatn, summarizatn and more.

Example :—

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

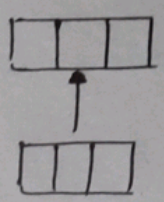
Embedding size = 4 [means for every words will convert it into 4 vector]

Q, K and V ⟶ consider as 4 Dimension.

Step 1 :— Token Embedding

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

So this the token that particularly convert for this words and this are my fixed vector.



contextual Emb.

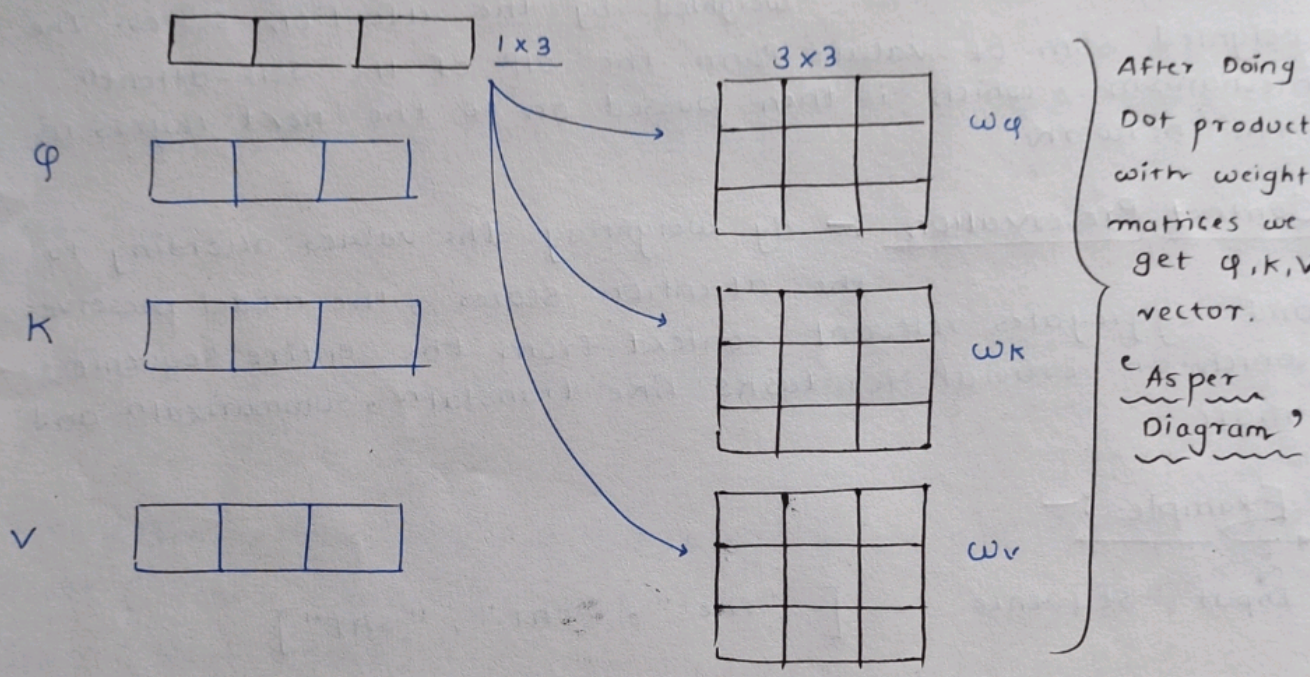
While converting sent. into cont. Embedding There's a Huge dependency for entire sentence.

For diff Dataset we have diff. sentence.

Why we used this q, K, V vector Because, The self attentn mechanism have some imfomatn of other words/sent. also ie why the reason we used this.

② Linear Transfomation :-

we will create Q K V vector by multiplying the embedding by learn weights matrices ω_Q, ω_K and ω_V.



$\omega_Q = \omega_K = \omega_V = I$ ⇒ Identity matrix ⇒ All Diagonal element 1 and other are zero.

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

When we do Dot operation with identity matrix, we definitely get Same value!

$Q_{The} = [1\ 0\ 1\ 0] \cdot I = [1\ 0\ 1\ 0]$ ✓

$K_{Cat} = [\quad] \cdot I = [\quad]$ ✓

$V_{Sat} = [\quad] \cdot I = [\quad]$ ✓

Lets compute :- $Q_{The} = K_{The} = V_{The} = [1\ 0\ 1\ 0]$

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

$Q_{sat} = K_{sat} = V_{sat} = [1\ 1\ 1\ 1]$

3 Compute Attention Scores :- Why ?

The model assess how much attentn to give to each token relative to the wrrent token.

∴ So for token the How much the important related to give cat and Sat.

Let's take the Dot operation.

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

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

Score (Q_{The}, K_{cat}) ⇒ $[1\ 0\ 1\ 0][0\ 1\ 0\ 1]^T$
⇒ 0

Score (Q_{The}, K_{sat}) ⇒ $[1\ 0\ 1\ 0]\ [1\ 1\ 1\ 1]^T$
⇒ 2

∵ This is how we compute score value!

For the Token **CAT**

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

Score (Q_{cat}, K_{cat}) = $[0\ 1\ 0\ 1] \cdot [0\ 1\ 0\ 1]$ ⇒ 0

Score (Q_{cat}, K_{sat}) = $[0\ 1\ 0\ 1] \cdot [1\ 1\ 1\ 1]$ ⇒ 2

Same for Token **SAT**

$[\ ][\ ]^T = 2$ ✓

$[\ ][\ ]^T = 2$ ✓

$[\ ][\ ]^T = 4$ ✓

4 Scaling :-

→ Means → whatever Score we are getting! why we performed the Scaling!

we takeup the score and Scale Down by Dividing the Score by the Sq. root of Key vector.

$\sqrt{d_K}$ · $d_K = 4$ → $\sqrt{d_K} = 2$

↓

Dimension of Key vector.

[Scaling in attention mechanisms is crucial to prevent the Dot product from growing too large

- In most Research paper → Ensure Stable gradient During Training

• If DK is Too large ⟶ Then ① Gradient Exploding
↳ Dimension of Key vector. facing ② Softmax Saturatⁿ.
2 prob.

For Example :— Consider 3 vector.

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

without Scaling Let's Do Dot product!

$q \cdot K_1^T = 2\times1 + 3\times0 + 4\times1 + 1\times0 = \boxed{6}$

$q \cdot K_2^T = 2\times0 + 3\times1 + 4\times0 + 1\times1 = 0+3+0+1 = \boxed{4}$

• Now after Scaling we have to Applied Softmax Actⁿ function. To get final contextual Embedding.

* Score [6, 4] ⇒ Scaling Not Applied.

Now softmax is applying for [6, 4] =

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

$$\Rightarrow \left[\frac{1}{(1+e^{-2})}, \frac{1}{(e^2+1)}\right] \Longrightarrow [0.88, 0.12] \ ★★$$

• Most of the attention weight is assigned to the Iˢᵗ Key vector very little to the Second vector,

One of the property of Softmax in Deep Learning.

⟶ Here the diff. Betⁿ score is very less But wr to OIP Here we see Huge diff

0.88 ⟶ 0.6

0.12 ⟶ 0.4

Similarly with [10, 1] ⟶ Here also we see Huge diff and value coming from Dot product of q and K.

In the Scenerio like [High, less] value we actually do not apply Scaling.

[10, 1] = [0.99, 0.01]

↓ when Dot product coming as a large value!

→ If small value comes into picture Here we are facing problem of softmax Saturation. / Also Known as Vanishing Gradient problem.

* Now by Scaling this value! [0.99, 0.01] with Scaling Dot problem.

① Compute scaled Dot product :-

Score [6, 4] ⇒ Scale ⇒ $\sqrt{d_k}$ (Divide by this) ⇒ with $\left[\frac{6}{2}, \frac{3}{2}\right] = [3, 2]$

⇓

$\sqrt{4} = 2$

Now applying softmax Activation :- $([3,2]) = \left[\frac{e^3}{e^3+e^2}, \frac{e^2}{e^3+e^2}\right]$

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

(Diff. is less/Normal)

⇒ [0.73, 0.27] ⟹ Also known as Attention Weights

- Here the attention weight are more balanced compared to the unscaled case :-

* Summary Importance :-

① Scaling prevent extremely large Dot products,
② Actually Help in stablizing the gradients during backpropagation
③ Making training process more Stable and efficient.

*→

① Prevent Saturation By Scaling Dot products, the softmax function produces more balanced attention weights, preventing the model from focusing too heavily on a single token and ignoring others.

② Improved Learning and 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.

Why we specifically Divide by $\sqrt{dK}$ ⇒ Variance (whenever Dot product Do!

(As we keep on increasing the Dimension the Dot product Dimension ↑ variance ↑)

$\sqrt{dK}$ (Variance for that particular Dot product is almost same)

* Scaling ensures the Dot product 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.

Now we have to Apply scaling :—

$\sqrt{dK} = \sqrt{4} = 2$

Scaled_Score (Q_{The}, K_{The}) = 2/2 = 1

Scaled_Score (Q_{The}, K_{CAT}) = 0/2 = 0

Scaled_Score (Q_{The}, K_{SAT}) = 2/2 = 1

Similarly Scaling will be Done for all other token.

⑤ Apply Softmax

Whenever we find out we Basically apply and create Attention weight.

Attention weights 'The' = Softmax([1,0,1]) = [0.423, 0.1554, 0.4223]

Attention weights 'CAT' = Softmax([0,2,2]) = [0.1554, 0.4222, 0.4223]

Attention weights 'SAT' = Softmax([2,2,4)] = [0.2119, 0.2119, 0.5762]

⑥ Weighted Sum of values :—

we multiply attention weights by corresponding values vectors for the token the =

$$Output_{(The)} = 0.4223 * V_{The} + 0.1554 * V_{cat} + 0.4223 * V_{sat}$$

Now we have to take the Token Embedding (3E)

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

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

$= [1.2669, 0.999, 1.2669, 0.9999]$

So the Embedding The Keyword $\boxed{1 \mid 0 \mid 1 \mid 0}$ → Self Attention

① Q K V . w^Q, w^K, w^V

② Attention Score

③ Scaled

④ Softmax

⑤ weighted Sum of value (softmax x v)

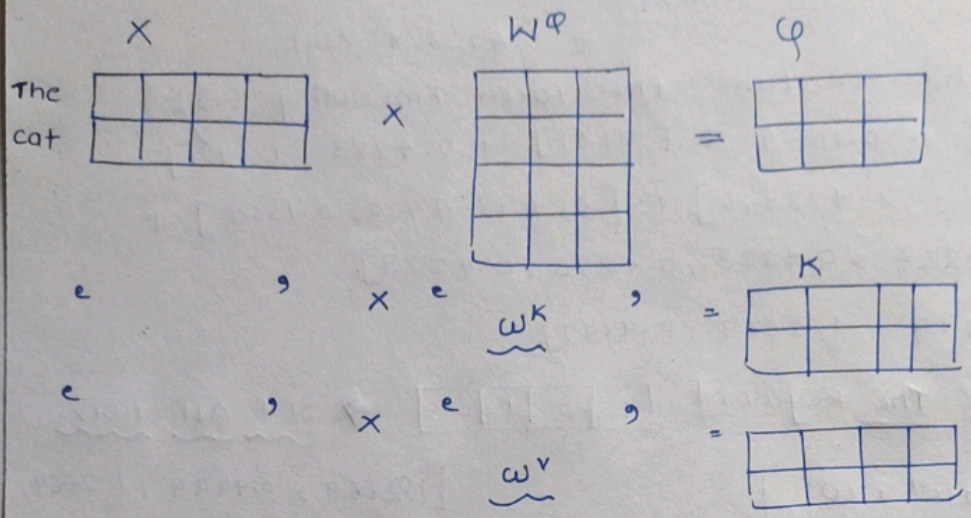
Summary

[1.2669, 0.9999, 1.2669, 0.9999]

Contextual vector

420. Multi-Head Attention

http://jalammar.github.io/illustrated-transformer/ → Visualize Transformer Understanding



Then Now Applying softmax. $\left(\frac{Q \times K^T}{\sqrt{d_K}}\right)$ V

Here we have to Do Dot operation

Here we do Scaling

= Z

Self Attention for 'The' value!

Now we have to Self-Attention with multi-head Attention.

Thinking machine X → Embedding vector!

Attention Head #0

Q_0

initialize Some other weight

W_0Q

K_0

W_0K

V_0

W_0V

Go to Score → Softmax AF $X \cdot V$ we got cont. vector.

Z_0

Attention Head #1

Q_1

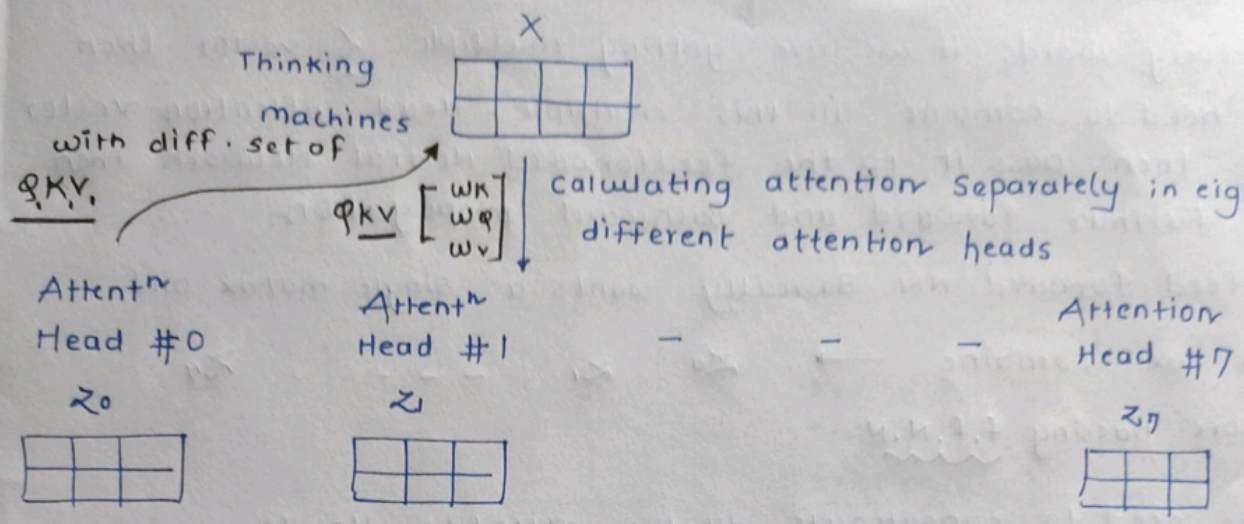
W_1Q

W_1K

W_1V

Here Some other context important is getting capture. (Z_1)

· It expand model Ability to focus on different position of tokens.

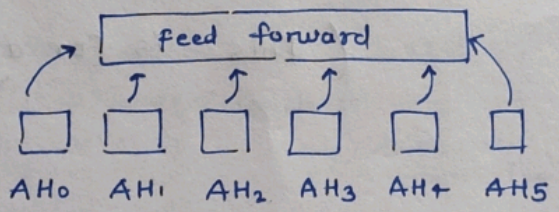


With a diff. set of QKV we get diff. Z0.

Here instead of one Attention Head ——— we calculate diff. diff multiple Head (Z0)

Here we focus on multiple Head attention

Basically



421. Feed forward Neural Net^n with MHA

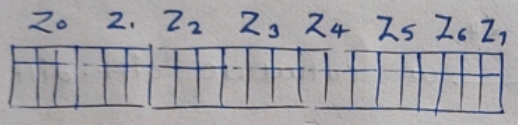
For every words if we are getting multiple Z vector then we need to combine all this multiple-Head attention vector and then pass it to the feedforward Neural Network then Do furthur forward and Backward propagation.

So, feed forward NN Basically wants a single matrix and

How we combine $\longrightarrow$ Z_0 Z_1 - - - - - Z_7

Before passing f.f.N.N.

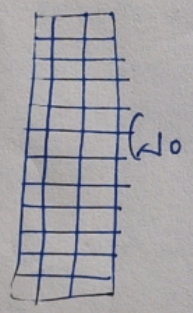
① we need to concatenate all the attention Heads



(7 multiple Head Attent^n)

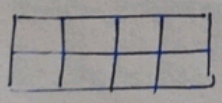
② Multiply with a weight matrix W^0 that was trained jointly with the model

X
(multiply)



(This is for a Single Words)

③ The result would be the Z matrix that captures information from all the attention Heads. We can ~~th~~ send this forward to the ffNN.



Z

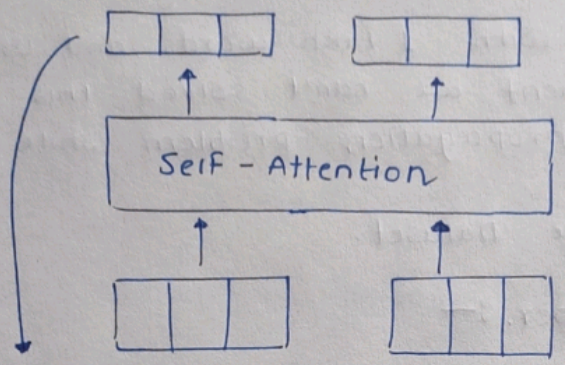
④ Then we have to calculate our 'Z'

So whatever 'Z' will get and will pass it to next enloder

- Highly co-related $\longrightarrow$ means Having High Dependency in multiple Head Attention.

422. Positional Encoding Indepth Intuition

* Representing – Order of sequence.



Advantage :–

① Words token process parallely

↓

Drawbacks (Becoz of this)

It lacks the sequential structure of the words (The orders Basically)

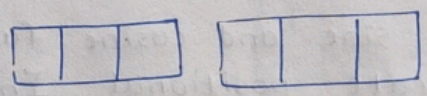
for Eg :–

① Lions Kills Tiger

② Tiger Kills Lions

Just Because of the order of Sentence is change the meaning of Sentence is change. All words are cons. Here,

suppose if we pass it to Self - Attentn then Both this words will return with this same vector



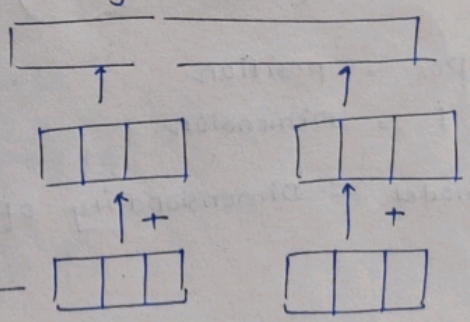
Actually positional Encoding vector.

Learnt ? Because of ordering of Sent. the entire sentence/meaning is change. So this information is missing in the Transformer that is why we use positional Enc to move forward in our problem Statement.

• Positional Encoded will be added to normal vector

↳ Responsible for actually telling which words are in which placed Basically.

* According To Research paper



Will be added to normal vector ⇐ Positional Encoding ⟵

There are multiple ways ① Embedding Ways.

Let's take 4 Dimension
(Actually Represent position)

| | | | 1 | | | | 2 | | | | 3 |

* Suppose if we have Book with 1 Lakh words and usually this type of problem statement we can't solved this Because it generally gives Back propagation problem while updating weight.
→ While considered Huge Dataset.

According To Research paper :—

Adding P.E.V which is some size of Embedding vector itself.

Type of Positional Encoding

① Sinusoidal P.E ② Learned P.E

It uses sine and cosine functions of different frequencies to create positional Encodings.
→ With the help of P.E will not have unlimited Becoz S.P.E all the values will be ranging Betn −1 to 1

How it works :—

All the values will be replace By this fomula :—

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

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

Where, pos = position
i = Dimension.
dmodel = Dimensionality of embedding

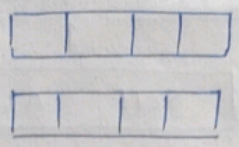
Eg :- The cat sat

converting into vector.

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]



4 Dimension

Embedding vector

$$PE_{(pos,2i)} = \sin\left(\frac{Pos}{10000^{2i/dmodel}}\right) \quad \Big| \quad PE_{(pos,2i+1)} = \cos\left(\frac{Pos}{10000^{2i/dmodel}}\right)$$

* Now we have to generate 4 positional Enc. vector!

For our Example dmodel = 4

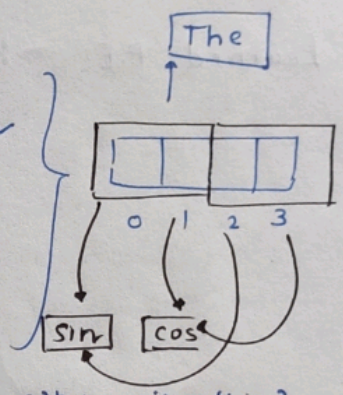
For position pos = 0

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

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

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

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



why like this?
Not getting similar like of Encoded value

if we get similar kind of Encoded value then will miss order of elements.

With Respect to first words the positional Encoding [0 1 0 1]

Now for P.E = 1 ⟶ CAT

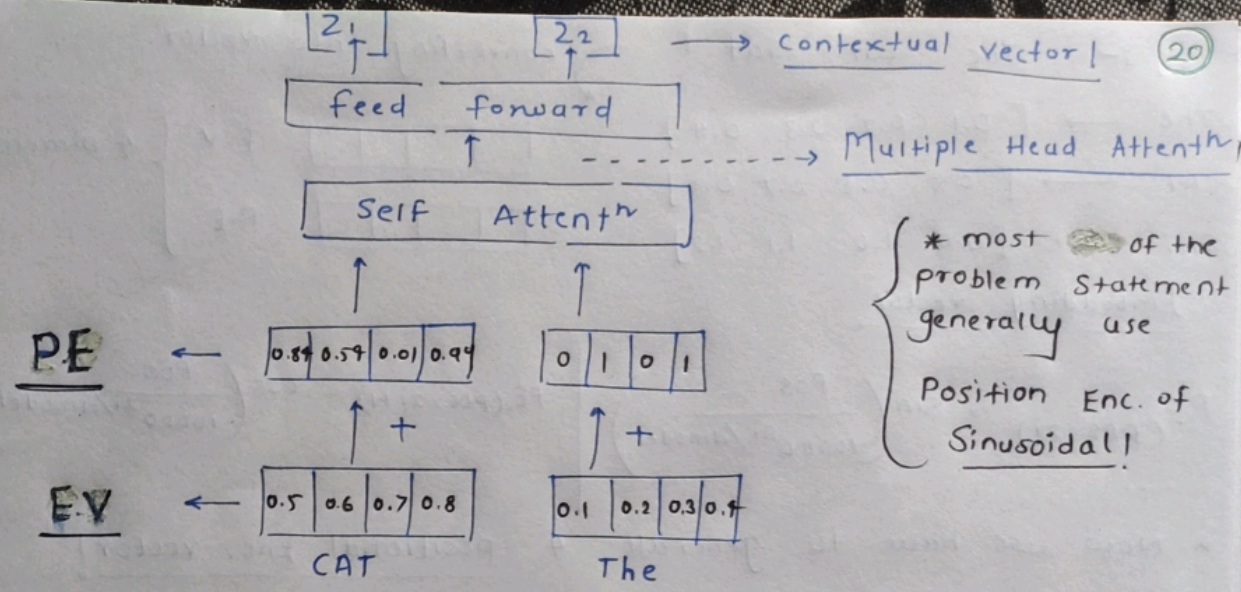
$$P.E(1,0) = \sin\left(\frac{1}{10000^{0/4}}\right) = \sin(1) = 0.8415$$

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

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

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

The	0.1	0.2	0.3	0.4	P.E for this vector →	0	1	0	1	P.E
CAT	0.5	0.6	0.7	0.8	→	0.84	0.54	0.01	0.99	



* most of the problem Statement generally use Position Enc. of Sinusoidal!

② Learned P.E :——— P.E are learned During training

423) Layer Normalization

* **Residual** :- Whatever vector we get before adding the positional encoding to the i/P embedding that is also passed to the add and normalized column ✓
  → Provide Some additional signal to the layer normalization!

* where this addition and normalization is added ?

* Talking about the steps ?

① This is our i/P sentence

② we embed each choirs → Here we add positional Encodding then

③ Split into 8 heads

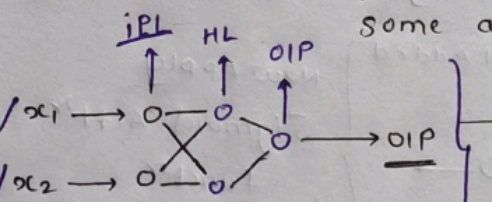
④ Calculate the attention using the resulting Q/K/V matrices

⑤ Concatenate the result Z matrices then multiply weight matrices W^o to produce the O/P of the layer

This is to add and Normalize then will do a Layer Normalisation.
→ we are providing signal to Entire layer and with some additional signal.

Normalization :-

→ Batch Normalize
→ Layer Normalize



Dataset :-

House Size	Area No.of Rooms	Price
F_1	F_2	
1200	2	45
1500	3	70
2000	3.5	80

• Before providing all i/P we do normalization
→ Standard Scaling → we take each of this Feature and Apply general Standard Scaling! formula

$$Z\text{score} = \frac{x_i - \mu}{\sigma}$$

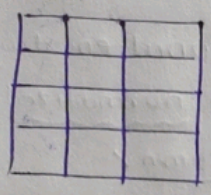
x_i → i/P
μ → mean
σ → Standard Deviatn

→ we apply such a way $\mu = 0$ and $\sigma = 1$

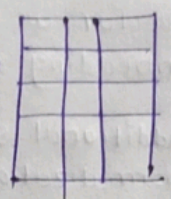
Talking F_1 and After Apply S.S will get → F_1'
$\mu = 0$ and $\sigma = 1$
∵ Basifically applied to entire column!

So we get μ, σ then we can apply Standard Deviation!

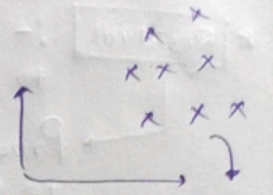
Deep Learning :- i/P Image ⟶ Min max Scaler!



After Applying the maxmin Scaler ⟶ we have all Dimension into 0-1



4x4

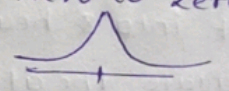


out of Range

Have Some images of 4x4 matrix value is betn 0 – 255.

(Here we have multiple Distributn

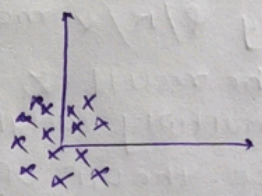
So we convert this distributn where all my informatn is centered! Nero to zero



- Advantage :- ① Improved Training Stability
- Suppose if Back Propagation is Happend! then will not get any type of vanishing gradient & Exploding Grad problem.

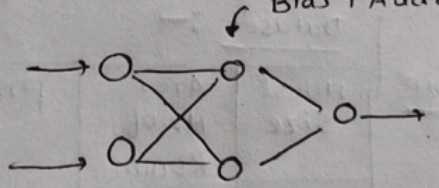
After Apply Scaling

② Due to this upper process ↳ will get fasten convergence!



Zero Centered!

! ⟶ In Back Propagation ⇒ It is stably update



Now Apply Sigmoid Acttn functn ⟶

House Size	Room	price
0.45	0.55	45
0.60	0.20	—
—	—	—

$z_1 = \sigma(0.45 * w_1 + 0.55 * w_3) + b_1]$

$= \text{value } 1$

$z_2 = \sigma[(\quad) + b_2]$

$= \text{value } 2$

Now the value 1 and value2 distribution is actually changed wrto f_1 and f_2

We dont need that change in distribution then we have to perform normalization in every o/P z_1 and z_2 ~ Batch Nomalizer

Apply Krna Pdenga formula $\mu_1 \sigma_1$, $\mu_2 \sigma_2$

* Batch Normalization (v/s) Layer Normalization (In Transformers ***)

(23)

① f_1 f_2 | z_1 | z_2

Taking comp z_1.

②

① f_1 f_2 z_1 z_2

We are taking Here layer by layer!

* Gamma and Beta! (γ β) ⟶ Learning Parameters

Suppose

z_1	z_2
-	-
-	-
-	-
0	-
0	0
0	0

∴ In ANN ✓ *** Batch Normalizt^n

After applying ~~Layer~~ normalizat^n Layer

Because every words is important

∵ Also need to normalize zero value!

In Batch normalizat^n There is a little change in no.

∵ Talking Layer by Layer

- Is this compulsory that we need to do normalizat^n?
  ↳ What if, final O/P and distribution is important for predicting O/P!

- If we do not want to normalization this!
  ↳ we using γ, β $\quad z_1 = \sigma[w_1^T x + b_1]$

[Assign Learnable parameter!] ↳ $y = \gamma\left[\frac{z_1 - H_1}{\sigma}\right] + \beta$ → Learnable parameters

∵ z_1 = all no.

γ → Learned Scale ✓

β → Shif

all called as 'Scale & Shift parameters'

① Emb! Each word! → X + PE

② ③ This will added to X'

④ X' [ ]

⑤ +

Here we specifically Apply Layer normalizat^n!

How this vector is normalized?

4 Dimension

Eg :- 'CAT' = [2.0, 4.0, 6.0, 8.0] ⟶ ①

Talking Example As Token

parameters = γ = [1.0, 1.0, 1.0, 1.0] ⟶ ②

β = [0.0, 0.0, 0.0, 0.0] ⟶

424 Layer Normalization Examples! :—

(I) Commute Mean → $Z_{score} = \frac{x_i - H}{\sigma}$

$$\left\{ H = \frac{1}{4}(2.0 + 4.0 + 6.0 + 8.0) = \frac{20.0}{4} = \underline{5.0} \right\}$$

(II) Compute the variance (σ^2)

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

$$= \underline{5.0}$$

(III) Normalize the I/P

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

(∵ $\epsilon = 1e^{-5}$ ⇒ Basically for Avoid Divide by zero!

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

$$\approx \sqrt{5.000} = \underline{2.236}$$

∵ Now normalizatn Each and Every w.r to Numbers!

$$\hat{x}_i = \frac{2.0 - 5.0}{2.236} \approx \underline{-1.34} \qquad \hat{x}_2 = \frac{4.0 - 5.0}{2.236} \approx \underline{-0.45}$$

$$\hat{x}_3 = \frac{6.0 - 5.0}{2.236} \approx \underline{0.45} \qquad \hat{x}_4 = \frac{8.0 - 5.0}{2.236} \approx \underline{1.34}$$

Basically Normalized vector ⇒ $\hat{x} = [1.34, -0.45, 0.45, 1.34]$

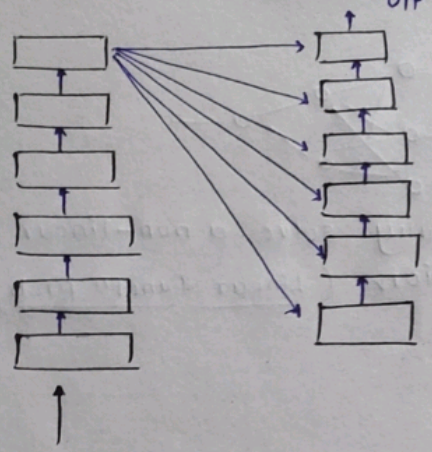
(IV) Scale and Shift :—

$$y_i = \gamma_i \hat{x}_i + \beta_i \qquad \gamma = [1.0, 1.0, 1.0, 1.0]$$

$$\beta = [0.0, 0.0, 0.0, 0.0]$$

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

425. Complete Encoder Transformer Architecture



O/P I am a Student

- Here we have 6 encoder!
- Giving i/P sequence → converting into vector (Text Emb. Technique) + Positional Encoding

No. of Dimension Taken is 512

Residual ✓

After we pass we get multiHead Attention

∵ Here Self Attn works also

Here we Have 8 Head Attentn

$Z_1\ Z_2\ Z_3\ Z_4\ Z_5\ Z_6\ Z_7\ Z_8$ → Add + Norma

↓

Feed forward NN

i/P Je Suis etudiant

∴ Every words convert into 512 of vector This is Basic info!

Inside Self-Attentn ✓

∴ What are the parameter that specifically use?

3/4 only!

. . . . . i/P Based on Dimension we are getting

. . . . . Hidden layer (12)

Q - 64 Dim
K - 64 Dim
V - 64 Dim

Divide by 64 $\sqrt{64} = 8$ } we have 8 no. of Attentn!

Why we used more encoder Because we get a good accuracy!
→ good Result!

① Why Residuals

②

③ Why Used feedforward neural network!

Residuals connection :— Also Known as 'Skip connection' why it Required?

① Addressing vanishing gradient problem. As no. of layers increases that Gradient Descent with resp to loss function is very small.

How Residuals Help :— Residual connection create a short paths for gradients to flow directly through the n/w Becoz of this gradient Remain sufficiently sharp. large.

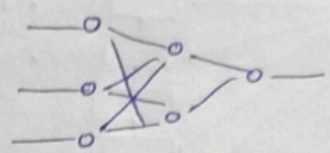
② It improved Gradient flows :- Convergence will be faster then Trainer is also Smoother!

③ This actually enables training of deeper networks.

Why feed forward NN :-

① Adding Non-Linearity

② processing Each position independently.



Actually solve a non-linear function. (Linear functn prob)

Self Attn ⟶ It capture relationship with tokens
(That each token is attend to every other tokens)

How does feed-forward neural network Help?

→ It processes each tokens representation independently, The feed-forward neural network applies the same 2 layer neural Network and this Help in Transforming these representatn furthur and allows the model to learn Richer Representatn.

③ FFN $\xrightarrow{\text{Help}}$ Neural Network to becomes much more deeper

Adds Depth to the model

more Depth

more Learning from Data!

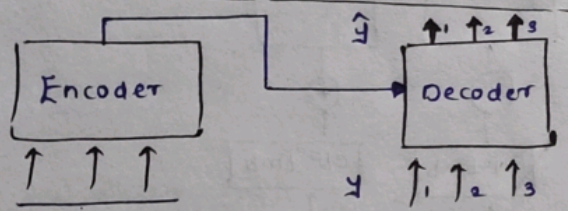
Also increases the parameters help us to generalize the training Data.

426. Decoder in Transformers

3 main components → ① masked multi-Head Attn
② multi-Head Attn (Enc-Decod. Attn)
③ Feed forward neural net.

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

- How Encoder and Decoder actually works.!



i/p passing all at once!

[∵ we do padding masking]

<Generated Based on Time stamp>

Understand Decoder Based on Two mechanisms.

① Training mechanism
② Inference mechanism

① Multi-masked-Head Attention.

① I/P Embedding and Positional Embedding → Here we zero padding to make our sequences length Equal!
② Linear projection for QKV
③ Scaled Dot projection Attentn
④ Mask Application. } → Try to understand the imp.
⑤ multi-Head Attention
⑥ Concatenation and final Linear program
⑦ Residual connection and Layer Normalization.

②
③
④
⑤

427. Decoder Transformer - Masked Multi-Head Attention Working

Dataset :—

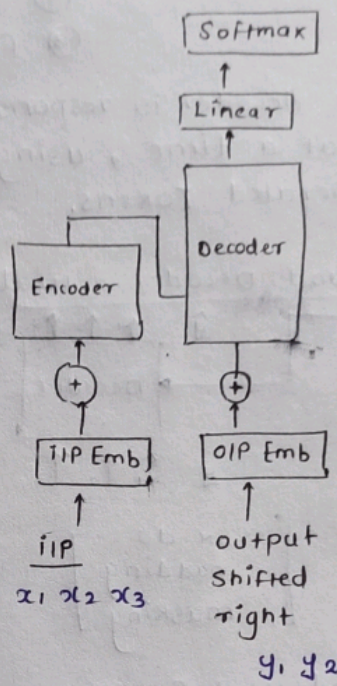
English ⟶ Hindi

$\langle x_1 x_2 x_3 \rangle$ (3 words) wr Eng.

$\langle y_1 y_2 \rangle$ (2 words) wr. Hindi. ← consider one IIP Data!

↓

Here we need to zero padding ie $\langle y_1 y_2 0 \rangle$



- What actually Happend in mask multi Attention ?

→ We have Data!

IIP	OIP
[4 5 6 7]	[1 2 3]

① First Step in Decoder is (i) IIP Embedding and positional Encoding OIP consider as padding and we use ↑ OIP Here! ⇝ 0 padding

Now output Embedding shifted to Right :—

$$\left.\begin{matrix} [0.1, 0.2, 0.3, 0.4] \\ [0.5, 0.6, 0.7, 0.8] \\ [0.9, 1.0, 1.1, 1.2] \\ [0.0, 0.0, 0.0, 0.0] \end{matrix}\right\} \oplus \boxed{PE = 0}$$

- Now the output Embedding should be going along with Specifically positional Encoding.

⟶ Now we furthur go to masked multi Head Attention.

There are few step :— ① Linear Projection Q K V

② Scaled Dot product – calc. Att^n Score

③ Masked Application → Looked Ahead, → Padding

Second Step :- Linear Projection for QKV

Creating query (Q), Key (K) and value (V) vectors.

$W_Q = W_K = W_V = I$ ⟶ considered as Identity Matrix

Q = output Embedding + W_Q = output Embedding

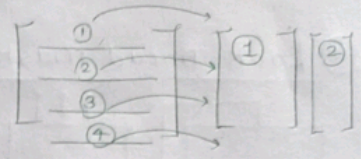
↳ Because of identity matrix!

K = e _____ , + W_K = OE

V = e _____ , + W_V = OE

we will get $Q = K = V$ [[0.1, 0.2 - -
- - - -
- - - -
0.0 0.0 0.0 0.0]]

$Q = K = V$ =



③ Scaled Dot Product Attention Calculation!

How do our

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

Now calculating score :-

What is Transpose of K^T (vertical matrix Basically)

After calculating my final score operator

[[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]]

[$1\times1 + 1\times2 + 1\times4 + 1\times5 + 2\times1 + 2\times2 + 3\times1 + 3\times2 + 4\times1 + 4\times2$

(ND Sir Rule)

④ Masked Application :-

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

purpose :-

① To Handle Sequences of different length in Batch.

② To ensure that padding tokens, which are added to makes sequences of uniform length, Do not affect the model prediction.

Eg :- Sequence 1 [1, 2, 3]

To encoder

Sequence 2 [4, 5, 0] ⟶ 0 is the padding Token

To Decoder

send ✓

↳ Influence the Attention mechanisms

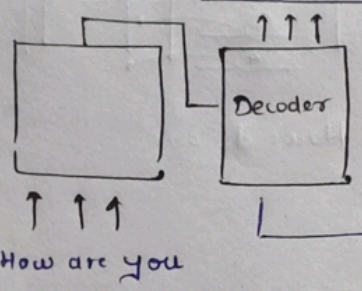
⬇

Lead to Incorrect or Biased Prediction

A padding mask → The Token are ignored

masking → (i) Padding mask
→ (ii) Look ahead mask → Combine of Both we used in Transformers!

(ii) Look Ahead mask :- Used To maintain Auto Regressive property



- Whenever we passed in any i/P we actually passed Decoder 1 by 1

Agar we want next words then we must some information of previous words/OP.

Q) why looked Ahed mask is specifically used?

Q) Why do we maintaing autoregressive property to ensure that each each position.

(1) To ensure that each position in Decoder O/P ~~system~~ Sequence can only attend to the previous position, but no future postn.

(2) For Diff. Seq-Seq. Task :- Language modeling, Translation.

Eg :- [4,5,0] → [1,1,0] → Convert 1D TO 3D.
→ zero padded Token.
→ we can also extended to 3 Dimension.

Every Token Represented by 3 Dimension.

[4, 5, 0] → $\begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ → In order to convert 1D to 2D, For each Token in the Sequences, the mask Should indicate which Token is attend to!

1D → 2D

(4) should probably Take the context of Token (5) Similarly with (5) (4)

Not Taking the context of zero!

→ Token 1 attends to token 1, 2
1, 2

Attention Mechanisms

(*) Now calculating look ahead mask :-

→ Depends on Decoder O/P

→ [[1 0 0]
[1 1 0]
[1 1 1]]

- Now we combine and looking Ahead mask
- How ?

→ Basically with help of two mask with element wise!

Combine Mask → [1 0 0]
[1 1 0]
[1 1 1]

wherever the value is value in combine mask!

- There we specify infinity

Compute our mask :—

Scores :— [[ ]
[ ]
[ ]
[ ]]

Look Ahead mask

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

- If i want to use padding masking [Extended to 2D format]

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

* If we want to probably find out combined mask →

Look Ahead * Padding Mask

$$\begin{bmatrix} 1\times1 & 0\times1 & 0\times1 & 0\times0 \\ 1\times1 & 1\times1 & 0\times1 & 0\times0 \\ 1\times1 & 1\times1 & 1\times1 & 0\times0 \\ 1\times1 & 1\times1 & 1\times1 & 1\times1 \end{bmatrix} \rightarrow \begin{bmatrix} [1,0,0,0] \\ [1,1,0,0] \\ [1,1,1,0] \\ [1,1,1,0] \end{bmatrix} \rightarrow \begin{bmatrix} [1,-\infty,-\infty,-\infty] \\ [1,1,-\infty,-\infty] \\ [1,1,1,-\infty] \\ [1,1,1,-\infty] \end{bmatrix}$$

* Compute masked Score → wherever we have zero

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

$-\infty$ is added to zero value!
It is Basically zero out / Removed the influence when the Softmax Act^n funct^n is applied!

* Softmax

Score = Softmax (masked Scores)

Softmax = [[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]]

* Weighted Sum of value

Attention O/P = Softmax Score * V

* **Masking** → masking in the transformer architecture is essential for several reasons. It helps manage the structure 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:

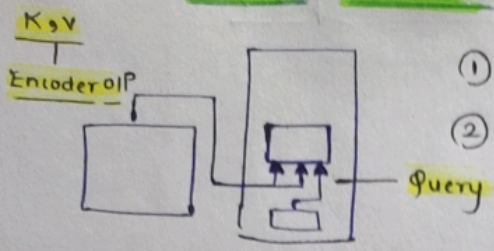
① Handling variable-length sequence with padding mask

Purpose → To Handle sequences of different length in a batch. To ensure that padding tokens, which are added to make sequences of uniform length, do not affect the model's predictn.

② Maintaining Autoregressive property with look-Ahead mask.

Purpose → To ensure that each position in the decoder O/P seq. 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.

428. Encoder Decoder Multi-Head Attention

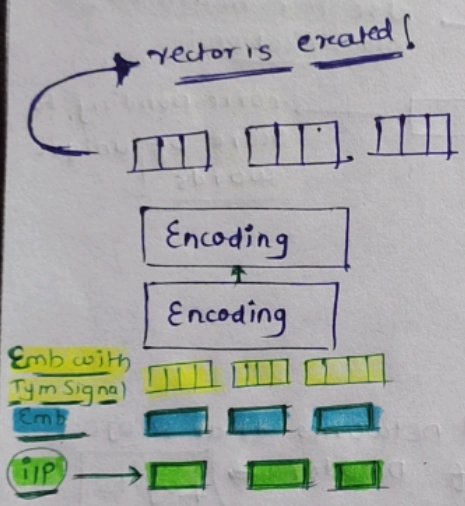


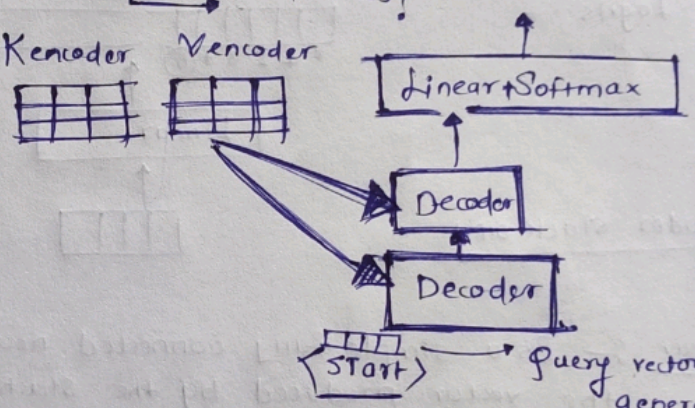
① Encoder O/P → Set of Attention vector K and V

② masked multi Head (O/P of) → Attention vector of {Query vector}

Why we used because this,

These are to be used by each decoder in its 'Encoder-Decoder' attention layer and why we specifically → Because it helps the Decoder to focus ton Appropriate place in the I/P sequences!





Query vector generated

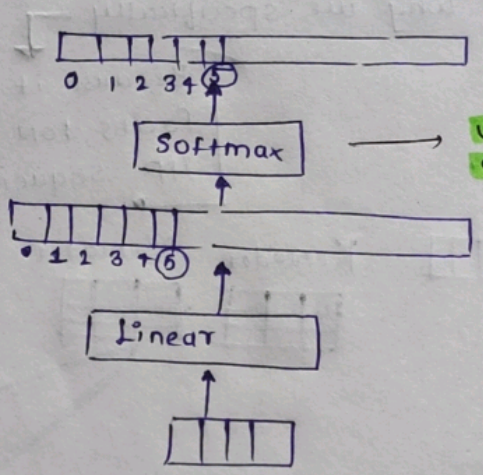
this will given to multiHead attention!

* Once I is comes (in previous O/P). What ever O/P is generated in t-1 it will pass again as a previous O/P and it generated the one by one O/P!

42g. Final Decoder Linear Softmax Layer!

which word in our vocab is associated with this index → am

Get the index of the cell with the Highest value (argmax) → 5



yaha jiski probability High Ho! wo o/p ayenga!

vocab_size

Use for multiclass classification!

corresponding to Score of unique words!

Linear :- Is a simple fully connected neural networks that projects the vector produced by the Stack of Decoder → Logits vector

Logits_vector :- model → 10000 vocabulary of unique words → Logits vector (10000 words) cells wide

(2) Softmax Layer Turns those score into probabilities (add all upto 1.0)
The cell with the Highest probability is chosen and the words associated with it is produced as the o/p. → Time Stamp!

Talking About the Recap of Training!

words	a	am	i	thanks	Student	<eos>
index	0	1	2	3	4	5

→ OHE

if i want to converted

Merci → Thanks

with an untrained model!

Now using Embedding Technique (OHE)

0	1.0	0	0	0	0

for the word 'am'

Untrained model o/p

0.2	0.2	0.1	0.2	0.2	0.1

Correct and desired o/p

0.0	0.0	0.0	1.0	0.0	0.0
a	am	i	thanks	Student	<eos>

we need this value as 1.0

Here we Basically we need to calculate the Loss function

↓

Imp for Back propagation will specifically Happend!

So my Target model something like this :—

o/p vocabulary :—

	a	am	i	thanks	student	<eos>
posit^n 1	0.0	0.0	1.0	0.0	0.0	0.0
posit^n 2	0.0	1.0	0.0	0.0	0.0	0.0
posit^n 3	1.0	0.0	0.0	0.0	0.0	0.0
posit^n 4	0.0	0.0	0.0	0.0	1.0	0.0
posit^n 5	0.0	0.0	0.0	0.0	0.0	1.0

(Before Training)

i/p: a am i student

o/p: I am a Student

↓

Each words convert into OHE!

* Trained model o/p

o/p vocab	a	am	i	thank	student	<eos>
post^{n}1	0.01	0.02	0.93	0.01	0.03	0.01
post^{n}2	0.01	0.8	0.1	0.05	0.01	0.03
post^{n}3	0.09	0.001	0.001	0.001	0.002	0.001
post^{n}4	0.001	0.002	0.001	0.02	0.94	0.01
post^{n}5	0.1	0.01	0.001	0.001	0.001	0.98

- This is because of Loss function and Epoochs

ie Keeps on updating