SELF - ATTENTION : ·

Self - Attention is a Mechanism that lets a model look at other words in the same sentence (or input sequence) to decide which ones are important when processing each word.

Why it Matters ?

Helps capture Context : Eg ; in "The bank near the river," self - attention helps understand the "bank" refers to a location, not finance.

Real-Time Example:

"The cat sat on the mat because it was tired"

Here the Tricky part:

⇒ The word "it" - refers to "cat", not "mat".

Self-attention is exactly designed to figure out these relationships.

Q, K, V in Self-Attention :-

Query (Q) :-

⇒ Represents the current word's question: "What am I looking for in other words?"

⇒ Example: "play" might query "Who is playing?" or "What is being played?"

Key (K) :-

⇒ Represents each word's identity / features: "What do I have that others may want to know?"

⇒ Example: "I" has the feature "subject / doer", "football" has "object / game".

Value (V) :-

⇒ The actual Information content carried by the word.

⇒ Once Attention is calculated (Q·K → importance score), We use it to pick up Values (V).

⇒ Example: if "play" attends to "football", it grabs "football's Meaning" through V.

Simple Analogy :

Query = What I want (my Question).

Key = What I can offer (my features).

Value = The actual info I give when someone atten[ds]
to me.

Self - Attention Formula :-

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V$$

Explanation :-

$QK^T \rightarrow$ Similarity Scores

=> Multiply each Query (Q) with all Keys (K).

=> This gives how much one word should atten[d]
to another.

=> Bigger Score = Stronger relationship.

=> Example : "play". "football" -> high Score,
"play". "the" -> low Score.

Softmax - Attention Weights

=> Converts Similarity Scores into pro[b]

=> Each word's attention to other[s]
0 and 1.

Why Softmax is chosen ?

=> Simple

=> Smooth & differentiable

⇒ Probabilistic

⇒ Balances focus vs. Distribution

That's why almost Every attention-based model (BERT, GPT, Transformers in general) uses Softmax.

Multiply With V - Weighted Values

⇒ Each weight is applied to the Value (V) of the word.

⇒ So the final Embedding of a word is the weighted Sum of Values it gathered from others.

Why Divide by $\sqrt{d_k}$ in Self-Attention?

Context

In Self-attention, we compute

$$Score = Q \cdot K^T$$

This dot product can get Very large when the Vector dimension d_k is high.

Problem:

⇒ Large dot products → large Values in Softmax.

⇒ Softmax becomes too Sharp → one token dominates.

Ex: [1.0, 0, 0]

⇒ Model becomes unstable during training.

Solution:

We Scale down the dot product by dividing by $\sqrt{d_k}$:

$$\text{Scaled Score} = \frac{Q.K^T}{\sqrt{d_K}}$$

This keeps the Values in a reasonable range, so softmax stays smooth and gradients don't Explo..

Intuition :

Think of $\sqrt{d_K}$ as a normalizer.

prevents attention from becoming overly Confident too early.

Example Calculation :-

I = [0.2 , 0.4 , 0.6]

play = [0.8 , 0.3 , 0.3]

football = [0.1 , 0.2 , 0.5]

Stacking them gives $X \in R^{3 \times 3}$

$$X = [[0.2, 0.4, 0.6], [0.8, 0.3, 0.3], [0.1, 0.2, 0.5]]$$

Weights of Q, K, V :-

Each map 3 -> 2 so $Q, K, V \in R^{3 \times 2}$

$$W\_Q = [[1.0, 0.0], [0.0, 1.0], [1.0, -1.0]]$$

$$W\_K = [[0.5, -0.5], [1.0, 0.0],$$

[0.0, 1.0]]

$W_V = [[1.0, 1.0],$
$[0.5, -0.5],$
$[1.0, 0.0]]$

compute Q, K, V ($Q = X \cdot W_Q$ = $X \cdot W_K$, $V = X \cdot W_V$)

$Q = [[0.8, -0.2],$
$[1.1, 0.80],$
$[0.6, -0.3]]$

$K = [[0.50, 0.50]$
$[0.70, -0.10],$
$[0.25, 0.45]]$

$V = [[1.00, 0.00],$
$[1.25, 0.65],$
$[0.70, 0.00]]$

i) Attention scores = QK^T

$$Q = \begin{bmatrix} 0.8 & -0.2 \\ 1.1 & 0.0 \\ 0.6 & -0.3 \end{bmatrix}, \quad K = \begin{bmatrix} 0.50 & 0.50 \\ 0.70 & -0.10 \\ 0.25 & 0.45 \end{bmatrix}$$

$$\text{Scores} = QK^T = \begin{bmatrix} 0.300 & 0.580 & 0.110 \\ 0.550 & 0.770 & 0.275 \\ 0.150 & 0.450 & 0.015 \end{bmatrix}$$

ii) Scale by $\sqrt{d_K}$

Here $d_k = 2 \Rightarrow \sqrt{2} \approx 1.4142$

$$\frac{Scores}{\sqrt{2}} = \begin{bmatrix} 0.2121 & 0.4101 & 0.0778 \\ 0.3889 & 0.5445 & 0.1945 \\ 0.1061 & 0.3182 & 0.0106 \end{bmatrix}$$

iii) Softmax (row-wise) → attention weights

$$Weights = \begin{bmatrix} 0.3233 & 0.3941 & 0.2826 \\ 0.3343 & 0.3905 & 0.2752 \\ 0.3179 & 0.3931 & 0.2890 \end{bmatrix}$$

Interpretation (each row sums to 1):

For I : attends ~ 32.3 % to I, 39.4 % to play, 28.3 % to football.

For Play : ~33.4 % I, 39.1 % play, 28.795 % football

For football : ~ 31.8 % I, 39.3 % Play, 28.9 % football

Weighted Sum with V :

$$V = \begin{bmatrix} 1.00 & 0.00 \\ 1.25 & 0.65 \\ 0.70 & 0.00 \end{bmatrix}$$

Final Context Vectors (one per token):

$$Output = Weight \cdot V = \begin{bmatrix} 1.0137 & 0.2561 \\ 1.0151 & 0.2538 \\ 1.0116 & 0.2555 \end{bmatrix}$$

Each row is the new Embedding for [I, play, football] after this attention head ie, a blend of the V Vectors weighted by the attention distibutions.

Multi-Head Attention :-

Multi-head attention allows the model to attend to different parts of the input simultaneously, using multiple sets of learned projections. Each head learns a different representation of the input Sequence.

Why Multi-Head Attention ?

In Normal Self - Attention, each token computes attention over all others → gives context.

Problem: a single attention head may only learn one kind of relationship (Eg: Word order, Syntactic role).

Solution : Use multiple attention heads in parallel. (Eg: each head learns different relationships grammer, semantics, long - Distance dependencies)

Single - Head (Vs) Multi-Head :-

Single - Head Attention (limitation) :-

⇒ In One head, queries (Q) look at Keys (K) to decide " where to pay attention"

⇒ This produces one set of weights per word → One "View" of dependencies.

⇒ Example :

In "The cat sat on the mat because it was tired; a single head might learn that "it" attends strongly to "cat."

But what if we also want :

"it" ⟷ "tired" (Semantic relation)

"sat" ⟷ "on the mat." (Syntactic relation.) ?

→ one head is too limited.

Multi-Head Attention (solution)

MHA solves this by projecting inputs into multiple subpspaces with different W^Q, W^K, W^V.

Each head :

⇒ sees the same sentence but from a different Angle.

⇒ Learns different attention patterns.

How multiple Heads Capture Different Relations :-

Each head has its own learned weight matrices (W^Q, W^K, W^V).

So Head 1 might align dimensions to capture Syntax (who relates to who structurally).

Head 2 might align dimensions to capture Semantics (meaning similarity)

Head 3 might look for long-range dependencies (eg: start ⟷ end).

Think of them like different Spotlights on the sentence, each lighting up a different pattern.

Steps Involved in Multi-Head Attention :-

1. Linear Transformation : The input X is projected into multiple smaller-dimensional subspaces using different weight matrices.

$$Q_i = XW_i^Q, \quad K_i = XW_i^K, \quad V_i = XW_i^V$$

Where i denotes the head index.

2. Independent Attention Computation : Each head independently computes its own self-attention using the scaled dot-product formula.

3. Concatenation : The outputs from all heads are concatenated.

4. Final Linear Transformation : A final weight matrix is applied to transform the concatenated output into the desired dimension. (W^O)

Purpose of Final Linear Transformation W^O :-

1. Dimensionality Restoration

=> After multi-head attention, the output shape is $h \cdot d_k$

=> But the model expects output in shape d_{model}

=> So $W^O \in R^{(h \cdot d_k) \times d_{model}}$ projects it back.

2. Learned Fusion of Head outputs

⇒ Each head captures different relationships (Syntax, Semantics, Position)

⇒ W^O learns how to combine these diverse perspectives.

⇒ It's not just resizing - it's a trainable mixer

3. Information Integration

⇒ Without W^O, head outputs remain isolated.

⇒ W^O blends them into a unified representation for the next layer.

⇒ Enables cross-head synergy and richer contextual understanding

Analogy :-

The Final Layer is like a team lead who reads all their reports and writes a cohesive summary It decides which insights are most relevant for the next layer.

Example Calculation :

Assume,

Input Sequence : "the", "cat", "sat"

Embedding Dimension (d_{model}) : $d_{model} = 4$

Number of heads (h): $h = 2$

Dimension of each head (d_k, d_v) : $d_k = d_v = d_{model} / h = 4/2 = 2$

Step 1 : Define Input Embeddings and Weight Matrices

$$X_{the} = [1, 0, 1, 0]$$

$$X_{cat} = [0, 1, 0, 1]$$

$$X_{sat} = [1, 1, 0, 0]$$

Next, We define the Weight matrices for the two heads. These matrices are of size $d_{model} \times d_K$ (ie, 4×2) for query, key and Value, and $d_{model} \times d_{model}$ for the final projection.

Head 1 weights :

$$W_1^{Q} = \begin{pmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \\ 0.5 & 0.6 \\ 0.7 & 0.8 \end{pmatrix}$$

$$W_1^{QK} = \begin{pmatrix} 0.8 & 0.7 \\ 0.6 & 0.5 \\ 0.4 & 0.3 \\ 0.2 & 0.1 \end{pmatrix}$$

$$W_1^{K} = \begin{pmatrix} 0.1 & 0.1 \\ 0.2 & 0.2 \\ 0.3 & 0.3 \\ 0.4 & 0.4 \end{pmatrix}$$

Head 2 weights :

$$W_2^{Q} = \begin{pmatrix} 0.9 & 0.8 \\ 0.7 & 0.6 \\ 0.5 & 0.4 \\ 0.3 & 0.2 \end{pmatrix}$$

$$W_{K2}^{K} = \begin{pmatrix} 0.2 & 0.3 \\ 0.4 & 0.5 \\ 0.6 & 0.7 \\ 0.8 & 0.9 \end{pmatrix}$$

$$W_2^{V} = \begin{pmatrix} 0.5 & 0.5 \\ 0.6 & 0.6 \\ 0.7 & 0.7 \\ 0.8 & 0.8 \end{pmatrix}$$

Final Projection Matrix :

Formula for W^O :

$$W^O \in R^{(h.d_K) \times d_{model}}$$

For Example :

$d_{model} = 512$

$h = 8$

So each head as $d_K = 64$ (since $512/8 = 64$

Then,

$$W^O \in R^{512 \times 512}$$

In our Ex calculation Example,

$d_{model} = 4$

$h = 2$

$d_K = 2$ (since $4/2 = 2$)

Then,

$$W^O \in R^{4 \times 4}$$

So our Example projection Matrix is,

$$W^O = \begin{pmatrix} 0.1 & 0.2 & 0.3 & 0.4 \\ 0.5 & 0.6 & 0.7 & 0.8 \\ 0.9 & 1.0 & 1.1 & 1.2 \\ 1.3 & 1.4 & 1.5 & 1.6 \end{pmatrix}$$

Step 2: Calculate Q, K and V Vectors for Each Head.

Head 1:

$$\Rightarrow Q_{the,1} = X_{the} \cdot W_1^Q = [1, 0, 1, 0] \begin{pmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \\ 0.5 & 0.6 \\ 0.7 & 0.8 \end{pmatrix}$$

$$= [0.1 + 0.5, 0.2 + 0.6]$$

$$= [0.6, 0.8]$$

$$\Rightarrow K_{the,1} = X_{the} \cdot W_1^K = [1, 0, 1, 0] \begin{pmatrix} 0.8 & 0.7 \\ 0.6 & 0.5 \\ 0.4 & 0.3 \\ 0.2 & 0.1 \end{pmatrix}$$

$$= [0.8 + 0.4, 0.7 + 0.3]$$

$$= [1.2, 1.0]$$

$$\Rightarrow V_{the,1} = X_{the} \cdot W_1^V = [1, 0, 1, 0] \begin{pmatrix} 0.1 & 0.1 \\ 0.2 & 0.2 \\ 0.3 & 0.3 \\ 0.4 & 0.4 \end{pmatrix}$$

$$= [0.1 + 0.3, 0.1 + 0.3]$$

$$= [0.4, 0.4]$$

$$\Rightarrow Q_{cat,1} = X_{cat} \cdot W_1^Q = [0, 1, 0, 1] (\ldots)$$

$= [0.3 + 0.7, 0.4 + 0.8] = [1.0, 1.2]$

$\Rightarrow K_{cat,1} = X_{cat} \cdot W_1^K = [0,1,0,1] (\ldots)$

$= [0.6 + 0.2, 0.5 + 0.1] = [0.8, 0.6]$

$\Rightarrow V_{cat,1} = X_{cat} \cdot W_1^K = [0,1,0,1] (\ldots) =$

$[0.2 + 0.4, 0.2 + 0.4] = [0.6, 0.6]$

$\Rightarrow Q_{sat,1} = X_{sat} \cdot W_1^Q = [1, 1, 0, 0] (\ldots) =$

$[0.1 + 0.3, 0.2 + 0.4] = [0.4, 0.6]$

$\Rightarrow K_{sat,1} = X_{sat} \cdot W_1^K = [1, 1, 0, 0] (\ldots) =$

$[0.8 + 0.6, 0.7 + 0.5] = [1.4, 1.2]$

$\Rightarrow V_{sat,1} = X_{sat} \cdot W_1^V = [1, 1, 0, 0] = [0.1 + 0.2,$

$0.1 + 0.2] = [0.3, 0.3]$

Head 2: (calculations are similar, just with Head 2 Weights)

$\Rightarrow Q_{the,2} = [1.4, 1.2]$,

$K_{the,2} = [0.8, 1.0]$

$V_{the,2} = [1.2, 1.2]$

$\Rightarrow Q_{cat,2} = [1.0, 0.8]$

$K_{cat,2} = [1.2, 1.4]$

$V_{cat,2} = [1.4, 1.4]$

$\Rightarrow Q_{sat,2} = [1.6, 1.4]$

$K_{sat,2} = [0.6, 0.8]$

$V_{sat,2} = [1.1, 1.1]$

Step 3: Calculate Attention Scores:

The attention scores are calculated as $score = \frac{Q.K^T}{\sqrt{d_K}}$

For $d_K = 2$, $\sqrt{d_K} = \sqrt{2} \approx 1.414$

Head 1:

$$\Rightarrow S_{the} = Q_{the,1} \cdot K_1^T = [0.6, 0.8] \cdot \begin{pmatrix} 1.2 & 0.8 & 1.4 \\ 1.0 & 0.6 & 1.2 \end{pmatrix}$$

$$= [0.6(1.2) + 0.8(1.0), 0.6(0.8) + 0.8(0.6), 0.6(1.4) + 0.8(1.2)]$$

$$\Rightarrow S_{the} = [1.52, 0.96, 1.8]$$

$$\Rightarrow S_{cat} = Q_{cat,1} \cdot K_1^T = [1.0, 1.2] \cdot \begin{pmatrix} 1.2 & 0.8 & 1.4 \\ 1.0 & 0.6 & 1.2 \end{pmatrix}$$

$$= [2.4, 1.52, 2.84]$$

$$\Rightarrow S_{sat} = Q_{sat,1} \cdot K_1^T = [0.4, 0.6] \cdot \begin{pmatrix} 1.2 & 0.8 & 1.4 \\ 1.0 & 0.6 & 1.2 \end{pmatrix}$$

$$= [1.08, 0.68, 1.28]$$

Now we divide each score by $\sqrt{2}$:

$\Rightarrow Scaled\_S_{the,1} = [1.07, 0.68, 1.27]$

$\Rightarrow Scaled\_S_{cat,1} = [1.70, 1.07, 2.01]$

$\Rightarrow Scaled\_S_{sat,1} = [0.76, 0.48, 0.91]$

Head 2 :

$\Rightarrow$ Scaled - $S_{the, 2}$ = [1.06 , 2.11 , 0.88]

$\Rightarrow$ Scaled - $S_{cat, 2}$ = [2.11 , 2.76 , 1.63]

$\Rightarrow$ Scaled - $S_{sat, 2}$ = [0.88 , 1.63 , 0.77]

Step 4 : Apply Softmax to Get Attention Weights

We apply the Softmax function to each row of the Scaled Score matrix. Softmax is $\frac{e^{x}}{\sum e^{x}}$

Head 1 :

$\Rightarrow W_{the, 1}$ = Softmax ([1.07 , 0.68 , 1.27])

= [0.43 , 0.29 , 0.28]

$\Rightarrow W_{cat, 1}$ = Softmax ([1.70 , 1.07 , 2.01])

= [0.38 , 0.20 , 0.42]

$\Rightarrow W_{sat, 1}$ = Softmax ([0.76 , 0.48 , 0.91])

= [0.39 , 0.29 , 0.32]

Head 2 :

$\Rightarrow W_{the, 2}$ = Softmax ([1.06 , 2.11 , 0.88])

= [0.24 , 0.66 , 0.10]

$W_{cat, 2}$ = Softmax ([2.11 , 2.76 , 1.63])

= [0.30 , 0.57 , 0.13]

$W_{sat, 2}$ = Softmax ([0.88 , 1.63 , 0.77])

= [0.27 , 0.56 , 0.17]

Step 5 : Calculate the Weighted Sum of Values

Now we multiply the attention weights by the Value Vectors of each head.

Head 1 Output (I_1) :

$\Rightarrow I_{the,1} = 0.43 \cdot V_{the,1} + 0.29 \cdot V_{cat,1} + 0.28 \cdot V_{sat,1} = 0.43\,[0.4, 0.4] + 0.29\,[0.6, 0.6] + 0.28\,[0.3, 0.3] = [0.172 + 0.174 + 0.084, 0.172 + 0.174 + 0.084] = [0.43, 0.43]$

$\Rightarrow I_{cat,1} = 0.38 \cdot V_{the,1} + 0.20 \cdot V_{cat,1} + 0.42 \cdot V_{sat,1}$

$= [0.38(0.4) + 0.2(0.6) + 0.42(0.3), \ldots]$

$= [0.34, 0.34]$

$\Rightarrow I_{sat,1} = 0.39 \cdot V_{the,1} + 0.29 \cdot V_{cat,1} + 0.32 \cdot V_{sat,1}$

$= [0.39(0.4) + 0.29(0.6) + 0.32(0.3), \ldots]$

$= [0.41, 0.41]$

Head 2 output (I_2) :

$\Rightarrow I_{the,2} = 0.24 \cdot V_{the,2} + 0.66 \cdot V_{cat,2} + 0.10 \cdot V_{sat,2} =$

$0.24\,[1.2, 1.2] + 0.66\,[1.4, 1.4] + 0.10\,[1.1, 1.1]$

$= [0.288 + 0.924 + 0.11, 0.288 + 0.924 + 0.11]$

$= [1.322, 1.322]$

$$\Rightarrow I_{cat,2} = 0.30 \cdot V_{the,2} + 0.57 \cdot V_{cat,2} + 0.13 \cdot V_{sat,2} = [0.3(1.2) + 0.57(1.4) + 0.13(1.1) \ldots] = [1.36, 1.36]$$

$$\Rightarrow I_{sat,2} = 0.27 \cdot V_{the,2} + 0.56 \cdot V_{cat \cdot 2} + 0.17 \cdot V_{sat,2} = [0.27(1.2) + 0.56(1.4) + 0.17(1.1) \ldots] = [1.30, 1.30]$$

Step 6 : Concatenate and project

We concatenate the outputs of the two heads for each word.

$$\Rightarrow I_{the} = [I_{the,1}, I_{the,2}] = [0.43, 0.43, 1.322, 1.322]$$

$$\Rightarrow I_{cat} = [I_{cat,1}, I_{cat,2}] = [0.34, 0.34, 1.36, 1.36]$$

$$\Rightarrow I_{sat} = [I_{sat,1}, I_{sat,2}] = [0.41, 0.41, 1.30, 1.30]$$

Finally we multiply the concatenated vectors by the final projection matrix W^0

$$\Rightarrow Output_{the} = I_{the} \cdot W^0 = [0.43, 0.43, 1.322, 1322] \begin{pmatrix} 0.1 & 0.2 & 0.3 & 0.4 \\ 0.5 & 0.6 & 0.7 & 0.8 \\ 0.9 & 1.0 & 1.1 & 1.2 \\ 1.3 & 1.4 & 1.5 & 1.6 \end{pmatrix}$$

$= [2.63, 2.92, 3.2, 3.5]$

$\Rightarrow Output_{cat} = I_{cat} \cdot k|^{0} = [0.34, 0.34, 1.36, 1.36]$.

$(\ldots)$

$= [2.67, 2.97, 3.27, 3.57]$

$Output_{sat} = I_{sat} \cdot k|^{0} = [0.41, 0.41, 1.30, 1.30]$.

$(\ldots)$

$= [2.58, 2.87, 3.16, 3.45]$

The Final output is a new 4-dimensional Vector for each word in the Sequence, which now Contains a richer representation that has been informed by all other words.

Note :-

\# In Original Transformer paper, they Experimented with different numbers of heads.

\# In the Base Model:

Number of heads = 8

d_model = 512 (Embedding size)

Each head dimension = 64 (Since $512/8 = 64$)

\# In the Large Model:

Number of heads = 16

d_model = 1024

Each head dimension = 64

$= [2.63, 2.92, 3.2, 3.5]$

$\Rightarrow Output_{cat} = I_{cat} \cdot W^{O} = [0.34, 0.34, 1.36, 1.36]$.

$(\ldots)$

$= [2.67, 2.97, 3.27, 3.57]$

$Output_{sat} = I_{sat} \cdot W^{O} = [0.41, 0.41, 1.30, 1.30]$.

$(\ldots)$

$= [2.58, 2.87, 3.16, 3.45]$

The final output is a new 4-dimensional vector for each word in the sequence, which now contains a richer representation that has been informed by all other words.

Note :-

In Original Transformer paper, they Experimented with different numbers of heads.

In the Base Model :

Number of heads = 8

d_model = 512 (Embedding size)

Each head dimension = 64 (Since ~~$5\!/\!8$~~ $512/8 = 64$)

In the Large Model :

Number of heads = 16

d_model = 1024

Each head dimension = 64