**SecA- batch1**

Q1. If a transformer layer has 8 attention heads, and each head has a weight matrix W of size 64x64, what is the total number of learnable parameters in the self-attention mechanism? How? (3)

A white paper with black text

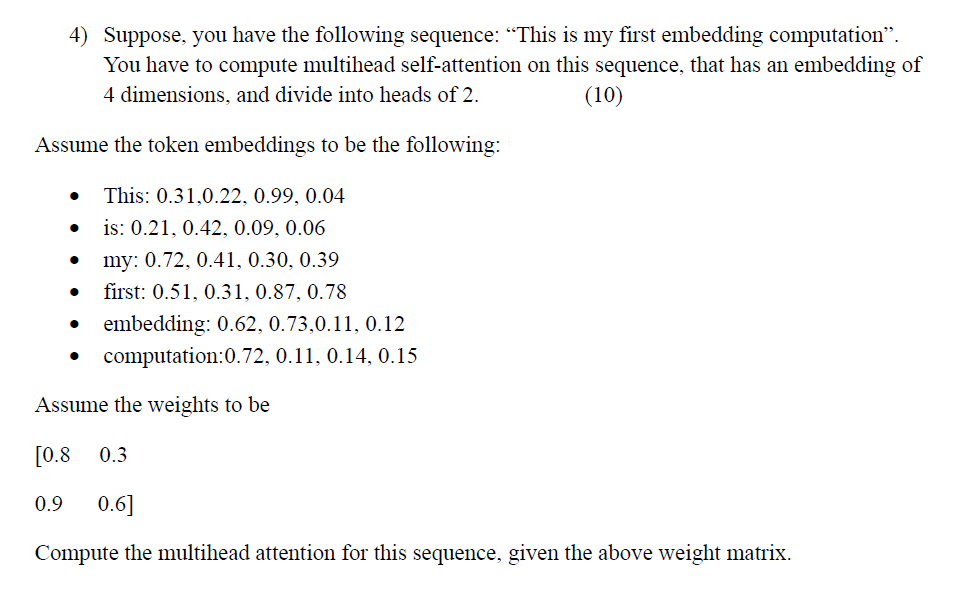
Description automatically generated

Q2. If a transformer model has 6 encoder layers and 6 decoder layers, with 8 attention heads each, how many total attention mechanisms are used in the entire model? How? (3)

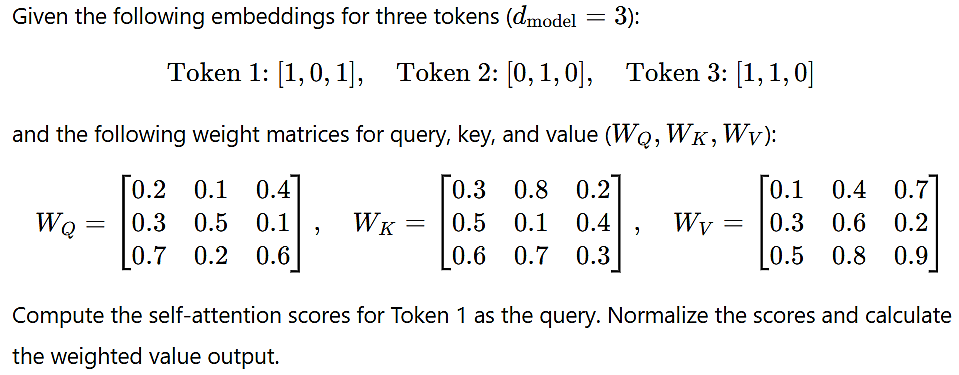
A white background with black text

Description automatically generated

**SecA- batch 2**



**SecB batch 1- 6 marks**



**1. Calculate Query, Key, and Value for each token:**

Since *d\_model* = 3, the embeddings and weight matrices have compatible dimensions for multiplication.

* **Token 1:**
  + Query (Q1): [1, 0, 1] \* W\_Q = [1, 0, 1] \* [[0.2, 0.1, 0.4], [0.3, 0.5, 0.1], [0.7, 0.2, 0.6]] = [0.9, 0.3, 1]
  + Key (K1): [1, 0, 1] \* W\_K = [1, 0, 1] \* [[0.3, 0.8, 0.2], [0.5, 0.1, 0.4], [0.6, 0.7, 0.3]] = [0.9, 1.5, 0.5]
  + Value (V1): [1, 0, 1] \* W\_V = [1, 0, 1] \* [[0.1, 0.4, 0.7], [0.3, 0.6, 0.2], [0.5, 0.8, 0.9]] = [0.6, 1.2, 1.6]
* **Token 2:**
  + Query (Q2): [0, 1, 0] \* W\_Q = [0.3, 0.5, 0.1]
  + Key (K2): [0, 1, 0] \* W\_K = [0.5, 0.1, 0.4]
  + Value (V2): [0, 1, 0] \* W\_V = [0.3, 0.6, 0.2]
* **Token 3:**
  + Query (Q3): [1, 1, 0] \* W\_Q = [0.5, 0.6, 0.5]
  + Key (K3): [1, 1, 0] \* W\_K = [0.8, 0.9, 0.6]
  + Value (V3): [1, 1, 0] \* W\_V = [0.4, 1, 0.9]

**2. Compute Self-Attention Scores for Token 1:**

We will calculate the attention scores for Token 1 by taking the dot product of its query (Q1) with the keys of all tokens (K1, K2, K3), and then dividing by the square root of *d\_model* (which is √3 ≈ 1.732).

* Score(Token 1, Token 1) = Q1 • K1 / √3 = (0.9 \* 0.9 + 0.3 \* 1.5 + 1 \* 0.5) / 1.732 = (0.81 + 0.45 + 0.5) / 1.732 ≈ 1.016
* Score(Token 1, Token 2) = Q1 • K2 / √3 = (0.9 \* 0.5 + 0.3 \* 0.1 + 1 \* 0.4) / 1.732 = (0.45 + 0.03 + 0.4) / 1.732 ≈ 0.508
* Score(Token 1, Token 3) = Q1 • K3 / √3 = (0.9 \* 0.8 + 0.3 \* 0.9 + 1 \* 0.6) / 1.732 = (0.72 + 0.27 + 0.6) / 1.732 ≈ 0.918

**3. Normalize the Scores (Softmax):**

* Softmax(1.016, 0.508, 0.918) ≈ (0.38, 0.19, 0.34) (These values don't add up to exactly 1 due to rounding, but they should be very close).

**4. Calculate the Weighted Value Output:**

The weighted value output for Token 1 is the weighted sum of the value vectors, using the normalized attention scores as weights:

* Output(Token 1) = 0.38 \* V1 + 0.19 \* V2 + 0.34 \* V3  
  ≈ 0.38 \* [0.6, 1.2, 1.6] + 0.19 \* [0.3, 0.6, 0.2] + 0.34 \* [0.4, 1, 0.9]  
  ≈ [0.228, 0.456, 0.608] + [0.057, 0.114, 0.038] + [0.136, 0.34, 0.306]  
  ≈ [0.421, 0.91, 0.952]

**SecB – Batch 2**

Q. A transformer model uses positional encodings to incorporate positional information into the word embeddings. The model has an embedding dimension *d\_model* = 4. Calculate the positional encodings for the first two positions (pos = 0 and pos = 1) using the standard sine and cosine formulas.

Show your calculations for each dimension of the positional encodings. What is the purpose of using both sine and cosine functions, and how does the 2i/d\_model term contribute to the effectiveness of positional encoding?

(3 + 3)

Here's how to calculate the positional encodings:

**Position 0 (pos = 0):**

* **i = 0:**
  + PE(0, 0) = sin(0 / 10000^(0)) = sin(0) = 0
  + PE(0, 1) = cos(0 / 10000^(0)) = cos(0) = 1
* **i = 1:**
  + PE(0, 2) = sin(0 / 10000^(2/4)) = sin(0) = 0
  + PE(0, 3) = cos(0 / 10000^(2/4)) = cos(0) = 1

Therefore, the positional encoding for position 0 is [0, 1, 0, 1].

**Position 1 (pos = 1):**

* **i = 0:**
  + PE(1, 0) = sin(1 / 10000^(0)) = sin(1) ≈ 0.8415
  + PE(1, 1) = cos(1 / 10000^(0)) = cos(1) ≈ 0.5403
* **i = 1:**
  + PE(1, 2) = sin(1 / 10000^(2/4)) = sin(1 / 100) ≈ 0.01
  + PE(1, 3) = cos(1 / 10000^(2/4)) = cos(1 / 100) ≈ 0.99995

Therefore, the positional encoding for position 1 is approximately [0.8415, 0.5403, 0.01, 0.99995].

**Purpose of Sine and Cosine:**

Using both sine and cosine functions allows the model to represent relative positional relationships. Because sine and cosine are related through a phase shift, the model can easily learn to attend to relative positions by learning simple linear transformations. This allows the model to generalize to longer sequences than those seen during training.

**Contribution of 2i/d\_model:**

The 2i/d\_model term creates different frequencies for different dimensions of the positional encoding. This allows the model to capture both broad and fine-grained positional information. Lower dimensions (smaller i) capture broader relationships (lower frequencies), while higher dimensions (larger i) capture finer-grained relationships (higher frequencies). This range of frequencies provides a richer representation of position and allows the model to learn more complex positional dependencies.