1. Self-Attention

Self-attention is the core operation in transformers. It helps determine how much focus each token in the sequence should pay to every other token, including itself.

For a token x_i:

1. Compute queries (Q), keys (K), and values (V):

- $Q = xW_Q$
- $K = xW_K$
- $V = xW_V$

where W_Q, W_K, and W_V are learned weight matrices.

2. Calculate attention scores:

Attention Score = QK^T

3. Scale the attention scores:

Scaled Scores = $QK^T/\text{sqrt}(d_k)$

where d_k is the dimension of the keys.
By dividing the scores by $\text{sqrt}(d_k)$, we reduce their magnitude, normalizing the range of inputs to the softmax function. This ensures: a) A smoother, more balanced attention distribution (not focussing on a subset of tokens too much). b) Stable gradients during backpropagation.

4. Apply softmax to obtain attention weights:

Attention Weights = $\text{softmax}(QK^T)/\text{sqrt}(d_k)$

5. Compute the weighted sum of values:

Output = (Attention Weights) dot (V)

The dot product with V aggregates information from all tokens based on their relevance (attention weights).

The weighted sum allows each token to “attend” to other tokens, capturing relationships and dependencies across the sequence.

The final output is a contextualized representation for each token, which can then be passed to subsequent layers in the transformer. (Use apple example)

Additionally, In the attention equation:

Attention (Q, K, V) = softmax((QK^T)/sqrt(d_k)) V

The term K^T represents the transpose of the matrix K (Keys). Here’s what it means in detail:

Keys (K)

- The keys matrix K has shape (seq_len, d_k), where:
- seq_len : The length of the input sequence.
- d_k : The dimensionality of each key vector.

Transpose (K^T)

- Transposing K changes its shape from (seq_len, d_k) to (d_k, seq_len).
- This transpose operation is essential for performing the dot product between Q (queries) and K (keys).

Purpose in Attention

1. Dot Product Operation (Q K^T):
   - Q (queries) has shape (seq_len, d_k).
   - K^T (transposed keys) has shape (d_k, seq_len).
   - Their dot product QK^T results in a score matrix of shape (seq_len,seq_len).
   - Each entry in this matrix represents the similarity (or relevance) between a query and a key for all tokens in the sequence.
2. Why Use Transpose?
   - The transpose ensures that the dot product aligns the query vectors (Q) with all key vectors (K) in the sequence.

- This is crucial for computing attention weights, as it allows comparing each token (query) to every other token (keys) in the sequence.

---

2. Attention Equation with Queries, Keys, and Values

The attention mechanism is described mathematically as:

Attention (Q, K, V) = softmax($(QK^T)/sqrt(d_k)$) V

- Q (Queries): Determines “what to look for.” (How relevant is a given token to other tokens in the sequence)
- K (Keys): Represents “what we have.” (Information in a given token)
- V (Values): Contains the “information to extract.” (Once a key matches a query, we extract the corresponding value associated with the key and pass on the information to the model)

The attention score is a measure of similarity between Q and K, and the values V are weighted according to this similarity.

---

3. Multi-Head Attention

Multi-head attention enhances the model’s ability to focus on different parts of the input sequence simultaneously.

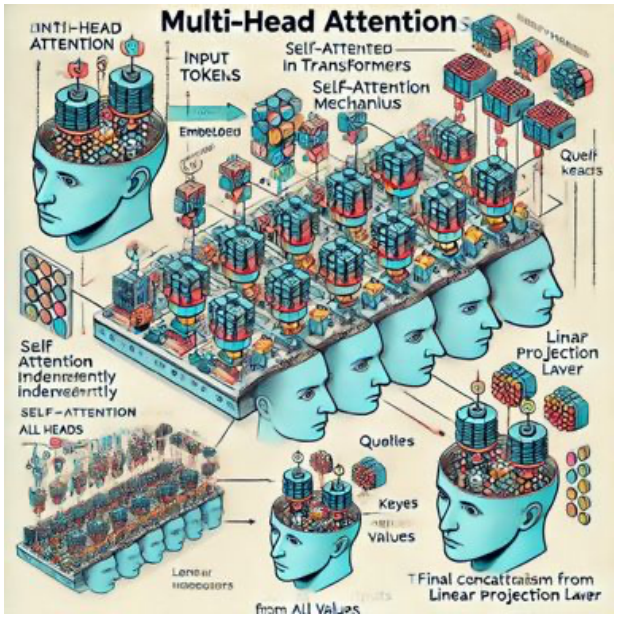
1. Split into Multiple Heads:
   Divide Q, K, and V into h heads of smaller dimensions (d_{model} / h). (h is a hyperparameter)
2. Apply Attention Mechanism for Each Head:
   Independently calculate attention outputs for each head:
   $head_i$ = Attention (Q_i, K_i, V_i)
3. Concatenate and Project:
   - Concatenate the outputs of all heads:
     MultiHeadOutput = Concat($head_1$, $head_2$, …, $head_h$)
   - Pass through a linear layer to combine the outputs.

This process allows the model to capture relationships from multiple perspectives.

Self-Attention vs. Multi-Head Attention

SA:
Concept: Focuses on relationships within the input sequence using a single attention mechanism.
Computation: A single attention mechanism with one set of Q, K, V.
Output: Produces one attention output for the entire sequence.
Purpose: Captures token relationships within a sequence.
Shape: Input/output size remains consistent with model dimensions.
Key Advantage: Simple, efficient for learning single relationships.

MHA:
Concept: Splits the attention mechanism into multiple heads to focus on different aspects of the sequence simultaneously.
Computation: Multiple parallel attention mechanisms with independent Q, K, V for each head.
Output: Combines the outputs from all attention heads into a single, aggregated output.
Purpose: Provides diverse representations by focusing on different relationships in the sequence.
Shape: Inputs/output dimensions split across multiple heads, then concatenated and projected back to the model dimension.
Key Advantage: More expressive; captures multiple relationships simultaneously.



---

4. Understanding the Shape

The shape of the tensors at each step is critical in implementing transformers:

- Input Sequence:
  Shape: (batch_size, seq_len)
  After embedding, it becomes: (batch_size, seq_len, d_{model})

- Queries, Keys, and Values:
  Each has a shape: (batch_size, seq_len, d_{head})
  where $d_{head} = d_{model} / h$

- Attention Weights:
  Shape after the scaled dot product: (batch_size, seq_len, seq_len)

- Attention Output:
  Shape after applying weights to values: (batch_size, seq_len, d_{head})

- Multi-Head Attention Output:
  After concatenation: (batch_size, seq_len, d_{model})

- Feed-Forward Layer:

Final output shape: (batch_size, seq_len, d_{model})

Multi-Head Attention

Purpose:

- Captures the relationships between tokens in the input sequence. It determines how much focus each token should pay to others in the sequence (contextual relationships).

Operation:

- Each attention head computes self-attention independently.
- Outputs from all heads are concatenated and linearly transformed to project back to the model's dimensionality

Key Takeaway:

- Multi-head attention focuses on token relationships by aggregating context from across the sequence.

Feed-Forward Layer

Purpose:

- Applies a non-linear transformation to the output of the multi-head attention mechanism, enabling the model to learn higher-level representations.
- Introduces depth and non-linearity to the transformer model.

Operation:

- The feed-forward layer is a simple position-wise fully connected neural network, applied independently to each token position in the sequence:

Key Takeaway:

- The feed-forward layer transforms each token's representation independently, enabling the model to capture complex patterns that multi-head attention cannot directly model.