The **Transformer** is a novel neural network architecture proposed for sequence transduction problems, such as language modeling and machine translation. Unlike the dominant models at the time, which were based on complex recurrent (like LSTMs or GRUs) or convolutional neural networks that included an encoder and decoder, the Transformer architecture **dispenses entirely with recurrence and convolutions**, relying solely on attention mechanisms.

The architecture follows a standard encoder-decoder structure. The encoder maps an input sequence of symbol representations to a sequence of continuous representations, and the decoder generates an output sequence one element at a time, being auto-regressive by consuming previously generated symbols as additional input. The Transformer implements this using **stacked self-attention and point-wise, fully connected layers** in both the encoder and decoder.

Key components of the Transformer include:

- **Encoder and Decoder Stacks:** Both consist of a stack of N=6 identical layers. Each encoder layer has two sub-layers: a multi-head self-attention mechanism and a position-wise fully connected feed-forward network. Each decoder layer has these two sub-layers plus a third sub-layer that performs multi-head attention over the encoder stack's output. **Residual connections** around each sub-layer, followed by **layer normalization**, are employed to facilitate training. The output dimension of these layers and the embedding layers is dmodel = 512 in the base model.
- **Attention:** The core mechanism. An attention function maps a query and a set of key-value pairs to an output, computed as a weighted sum of the values. The weight for each value is determined by a compatibility function of the query with the corresponding key.
    - **Scaled Dot-Product Attention:** This is the specific attention function used. It computes dot products of the query with all keys, divides by the square root of the key dimension (√dk), and applies a softmax function to get the weights. This scaling is used to counteract the effect of large dot products pushing the softmax into regions with small gradients for large key dimensions.
    - **Multi-Head Attention:** Instead of a single attention function, this performs the attention function in parallel on different linear projections of the queries, keys, and values. The results are concatenated and projected again. This allows the model to **jointly attend to information from different representation subspaces at different positions**, which is beneficial compared to a single attention head. The base model uses h=8 parallel attention layers (heads).

- **Applications of Attention in the Model:** Multi-head attention is used in three ways:
    - **Encoder-Decoder Attention:** Queries are from the previous decoder layer, keys and values from the encoder output. This allows decoder positions to attend over all input sequence positions.
    - **Encoder Self-Attention:** Keys, values, and queries all come from the output of the previous encoder layer. Each position attends to all positions in the previous encoder layer.
    - **Decoder Self-Attention:** Keys, values, and queries all come from the output of the previous decoder layer. Each position can attend to all positions in the decoder **up to and including that position**. Masking is used to prevent attending to subsequent positions, preserving the auto-regressive property.
- **Position-wise Feed-Forward Networks:** Applied to each position separately and identically within each layer, consisting of two linear transformations with a ReLU activation. The dimensionality is dmodel=512 for input/output and dff=2048 for the inner layer in the base model.
- **Embeddings and Softmax:** Learned embeddings convert input/output tokens to vectors of dimension dmodel. A learned linear transformation and softmax convert the decoder output to predicted next-token probabilities. The weight matrix is shared between embedding layers and the pre-softmax linear transformation.
- **Positional Encoding:** Since the model lacks recurrence or convolution, **positional encodings are added to the input embeddings** to inject information about the position of tokens. The paper uses sine and cosine functions of different frequencies, which allows the model to potentially extrapolate to longer sequences and learn relative positions. Learned positional embeddings were also found to produce nearly identical results.

The **motivation for using self-attention** centers on improvements in computational efficiency and the ability to learn long-range dependencies.

- A self-attention layer connects all positions with a **constant number of sequential operations (O(1))**, making it highly parallelizable, whereas a recurrent layer requires O(n) sequential operations. This parallelization is critical for training speed, especially with long sequences.
- Self-attention reduces the path length between any two positions in the input or output sequence to a constant number of operations (O(1)). Shorter paths are believed to make learning long-range dependencies easier. Convolutional networks require more layers (and thus longer paths) to connect distant positions, growing linearly or logarithmically with distance.
- Self-attention layers are computationally faster than recurrent layers when the sequence length (n) is smaller than the representation dimensionality (d), which is often the case in machine translation tasks.
- Additionally, self-attention can lead to **more interpretable models**, as attention distributions appear related to the syntactic and semantic structure of sentences.