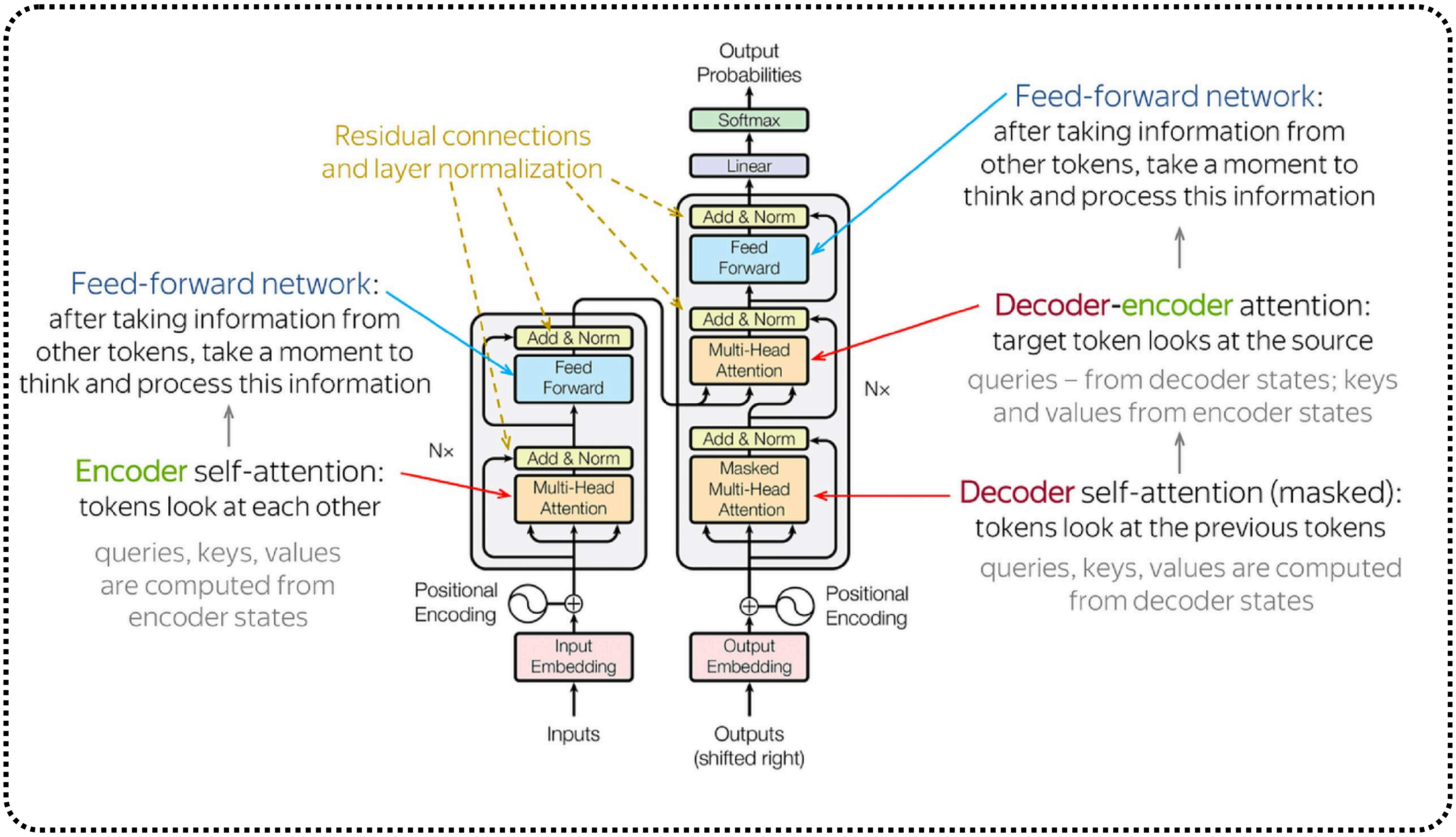
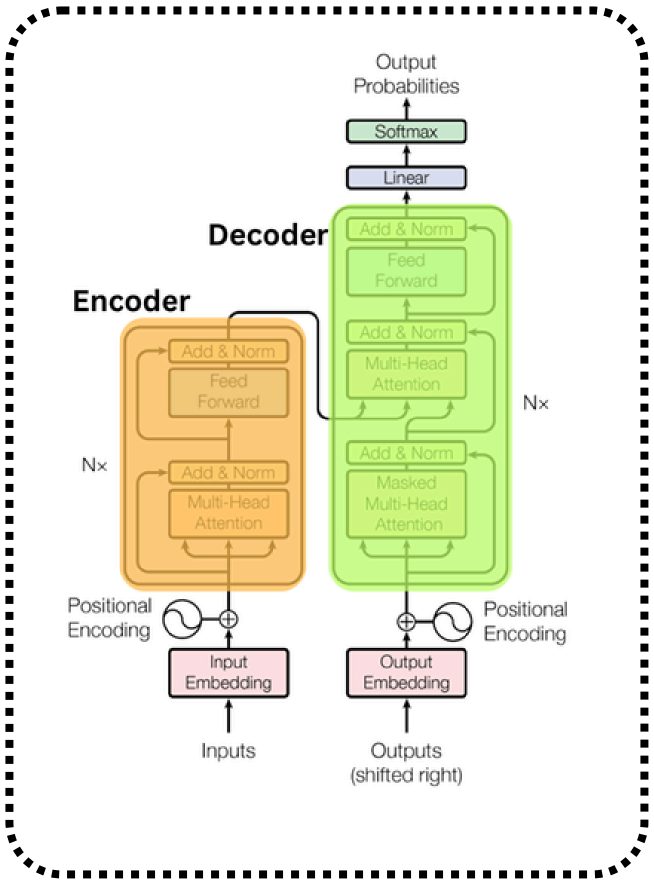
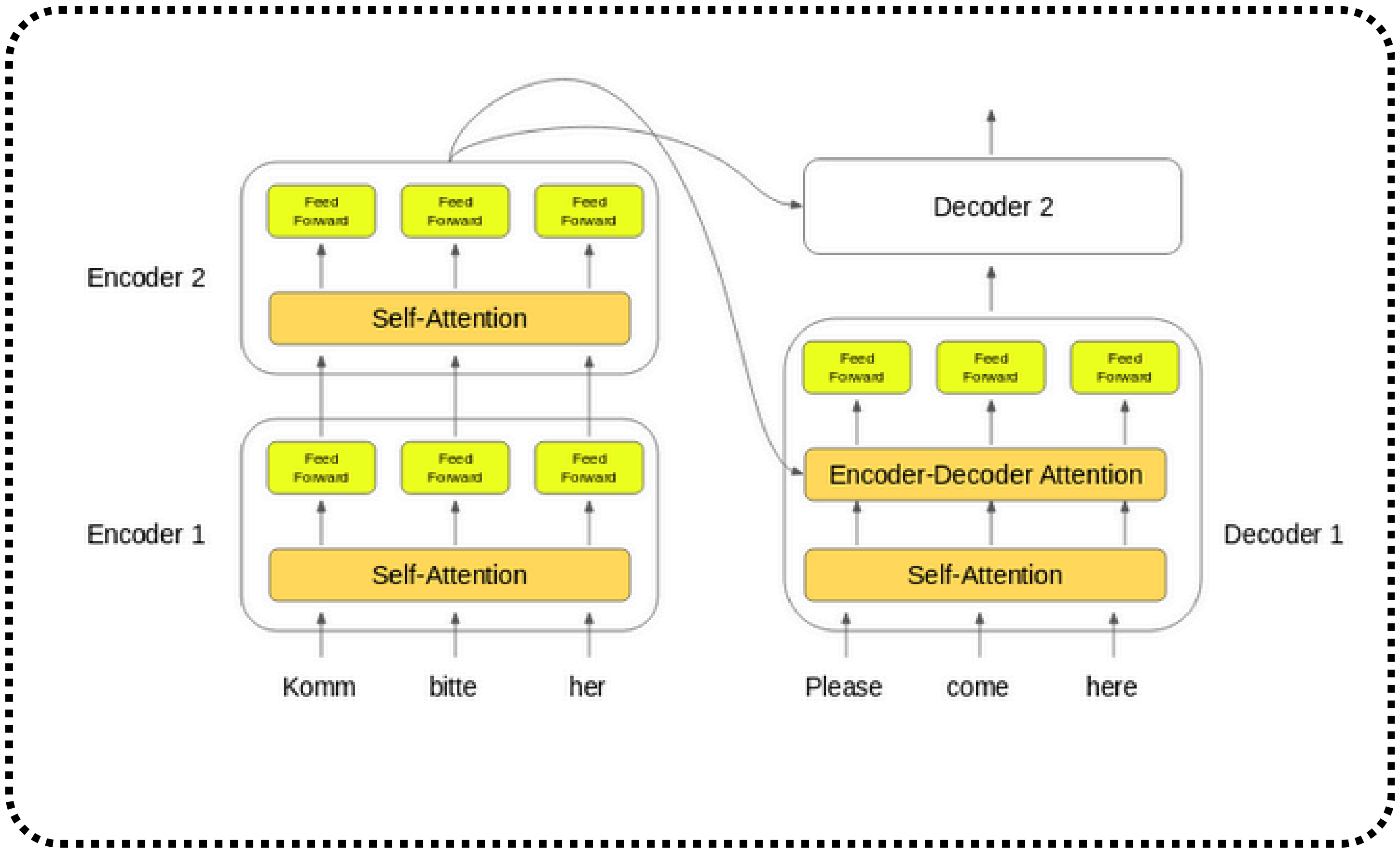
Analytics Vidhya

Mastering LLMs

Day 3: A Perfect Guide to Transformers


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Multi-Head Attention
Masked Multi-Head Attention
Input Embedding
Output Embedding
Positional Encoding
Nx
Inputs
Outputs (shifted right)
Residual connections and layer normalization
Feed-forward network: after taking information from other tokens, take a moment to think and process this information
Encoder self-attention: tokens look at each other
queries, keys, values are computed from encoder states
Decoder-encoder attention: target token looks at the source
queries – from decoder states; keys and values from encoder states
Decoder self-attention (masked): tokens look at the previous tokens
queries, keys, values are computed from decoder states



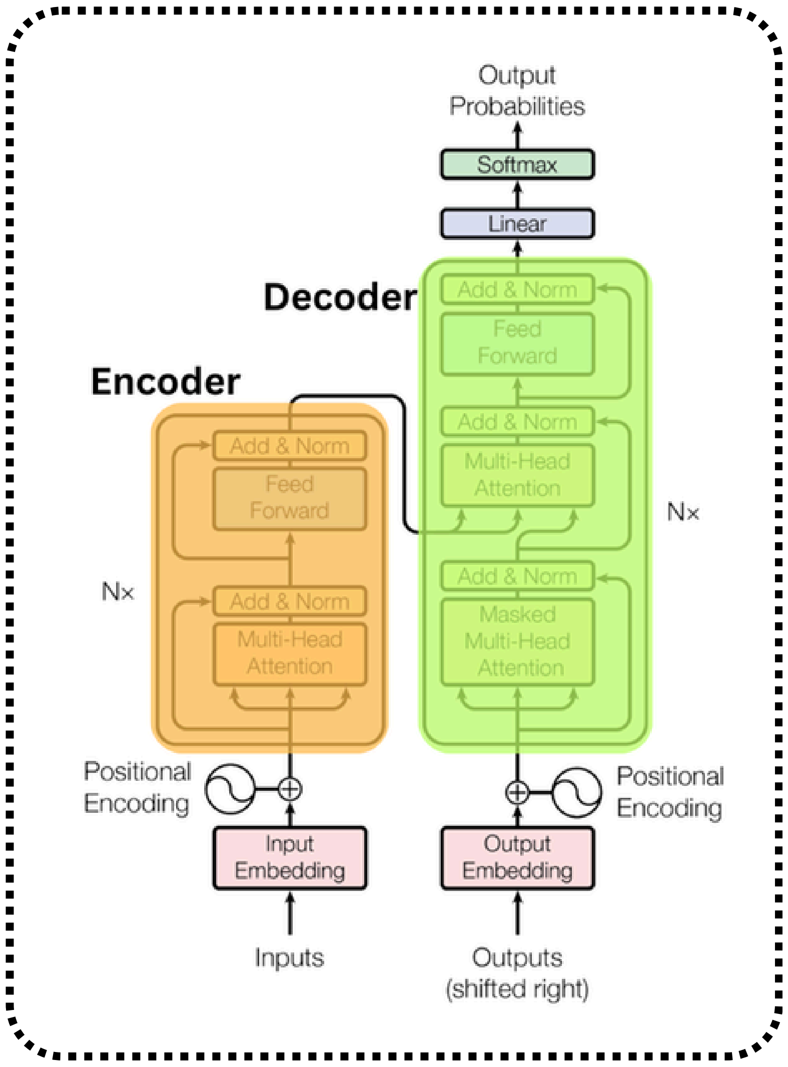


What is a Transformer?

The Transformer is a deep learning model architecture introduced in the paper "**Attention is All You Need**" by Vaswani et al. (2017). It replaced older architectures like **Recurrent Neural Networks** (RNNs) and **Long Short-Term Memory (LSTM) networks** because of its efficiency and parallelization capabilities.

Transformers are used in various tasks, including language translation, summarization, image captioning, and generative tasks such as text generation.

The Transformer model has two main components:



Encoder

The encoder processes the input sequence and generates a context-sensitive representation of the input.

Decoder

The decoder generates the output sequence from the encoded representation.

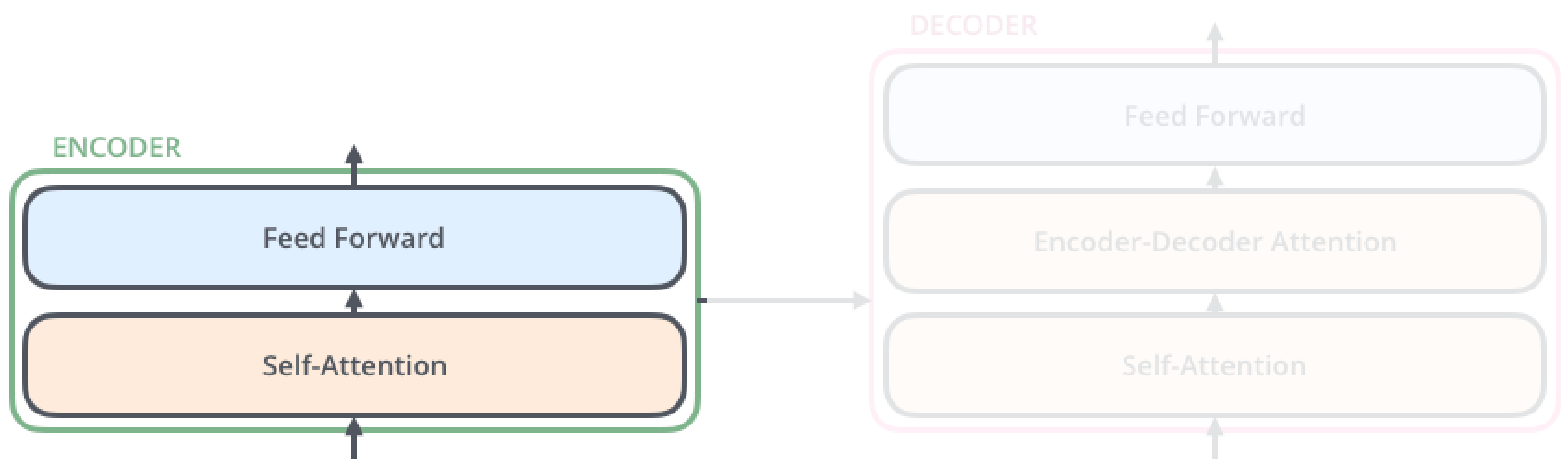
Both the encoder and the decoder consist of layers that work in parallel, unlike RNNs, which process sequential data step by step.

Encoder

The encoder processes the input sequence (e.g., a sentence in a translation task) and generates a context-sensitive representation of the input.

An encoder consists of two primary parts:

- **Self-Attention Layer**
- **Feedforward Neural Network**



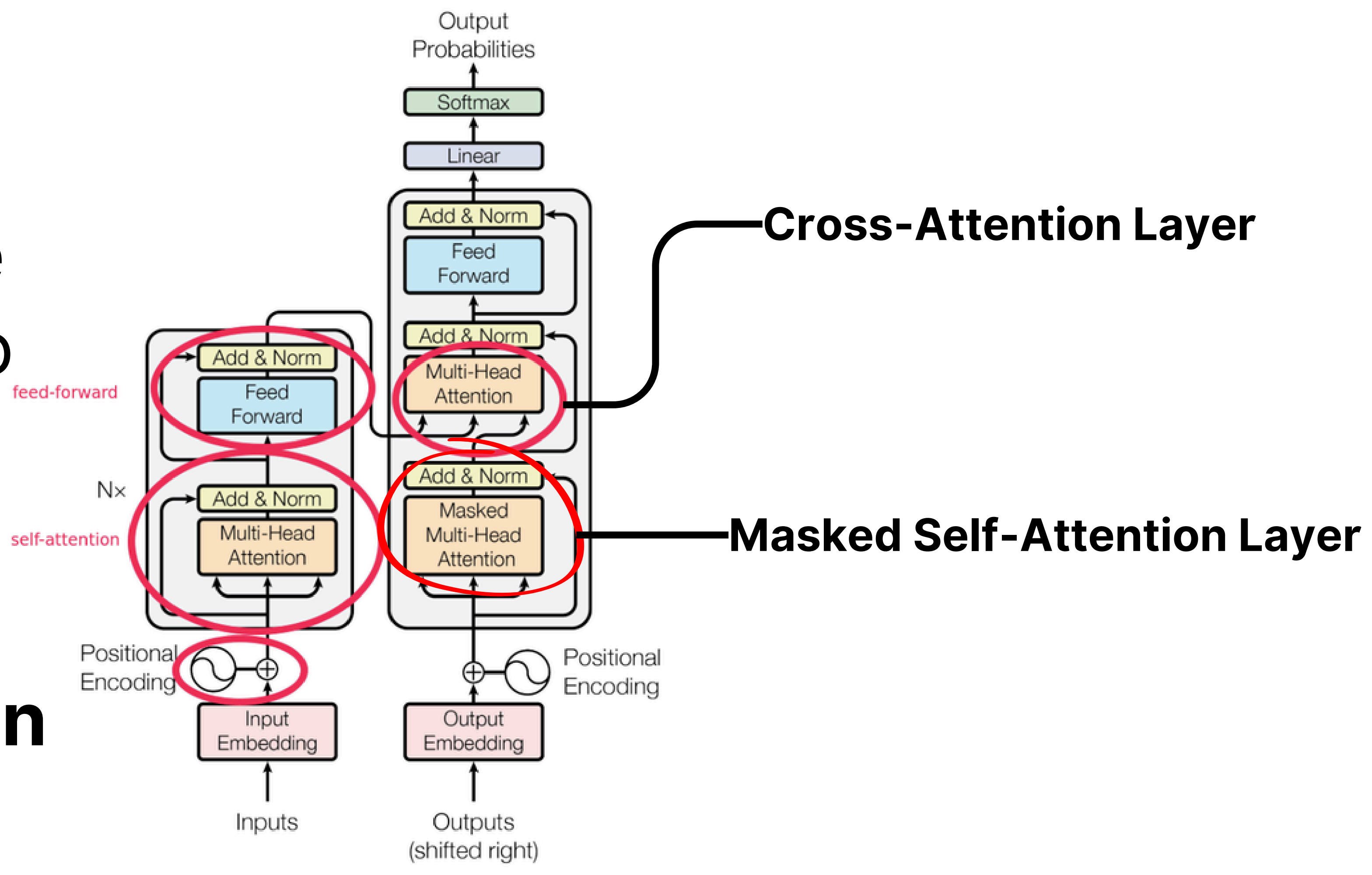
Each of these parts is followed by a normalization step and a residual connection to ensure better training and gradient flow.

Decoder

The decoder generates the output sequence (e.g., the translated sentence) from the encoded representation. Similar to the encoder, the decoder has two main components:

- **Masked Self-Attention Layer**: This layer ensures that predictions for the next word can only depend on previously generated words (important for autoregressive tasks like text generation).
- **Cross-Attention Layer**: This layer attends to the encoder's output, allowing the decoder to use the encoded input sequence’s information for generating the output.

Just like the **encoder**, the **decoder** also has a **feedforward layer** and **normalization** steps



Analytics Vidhya

Attention Mechanism

The most important concept in Transformer architecture is Attention. It helps the model focus on the most relevant parts of the input when producing an output. The attention mechanism is applied in multiple places within both the encoder and the decoder. Let’s break it down:

Scaled Dot-Product Attention

The basic attention mechanism calculates a weight (attention score) for each input word with respect to the others. The formula for the attention score involves three elements:

- **Queries** (Q): Represent the current token or word for which we want to find relevant information.
- **Keys** (K): Represent the tokens in the input sequence.
- **Values** (V): Contain the actual information for each token in the input sequence.

The formula for calculating the attention score is:

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

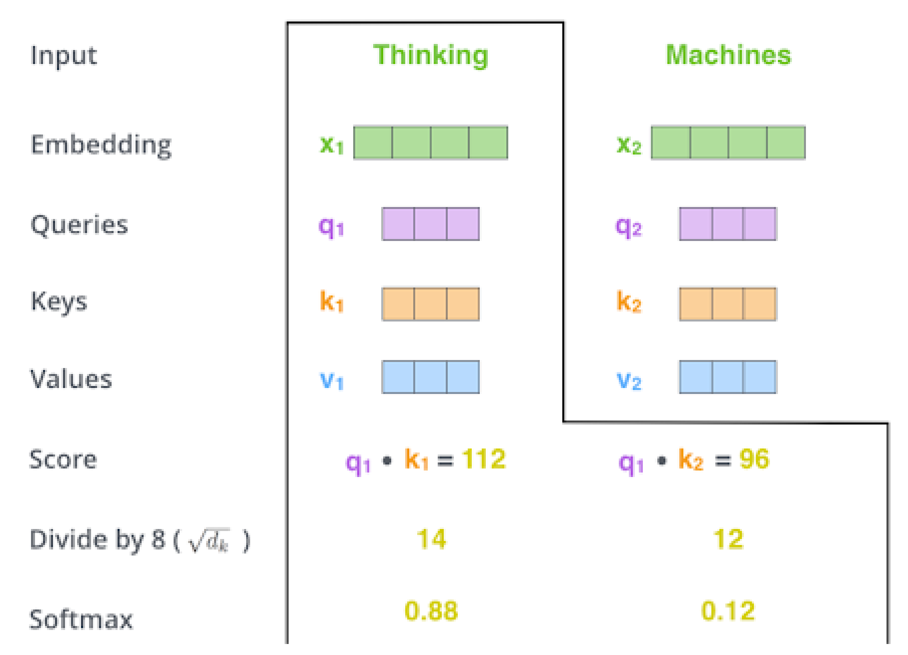


Image Source: The illustrated Transformer

Where:

- Q is the query vector.
- K is the key vector.
- V is the value vector.
- dk is the dimension of the key vectors.

Multi-Head Attention

- Instead of performing a single attention operation, the Transformer uses multi-head attention, where the input is split into multiple subspaces (or heads). Each head learns different aspects of the input sequence. The outputs of these heads are then concatenated and linearly transformed.

- This allows the model to capture various relationships between different words at different positions in the sequence.

Position Encoding

- Since the Transformer processes the entire input in parallel (as opposed to sequentially like RNNs), it needs a mechanism to understand the order of words in the sequence. Position encoding is added to the input embeddings to provide this positional information.

- Position encodings are vectors added to the input tokens' embeddings, allowing the model to recognize the order of tokens. These encodings are typically generated using sine and cosine functions at different frequencies.

- For each position i, the position encoding vector is:

$$\mathrm{PE}(i, 2k) = \sin\left(\frac{i}{10000^{2k/d}}\right)$$

$$\mathrm{PE}(i, 2k+1) = \cos\left(\frac{i}{10000^{2k/d}}\right)$$

Where:
- i is the position of the word.
- k is the dimension of the position encoding.
- d is the embedding size.

Analytics Vidhya

Layer Normalization & Residual Connections

Each sub-layer (like attention or feedforward) in both the encoder and the decoder is followed by:

- **Residual connections**: The input to each sub-layer is added to its output, helping to preserve the original information and prevent vanishing gradients.
- **Layer normalization**: Applied to normalize the outputs, helping with stability during training.

Feedforward Neural Network

Each encoder and decoder layer includes a position-wise feedforward network that operates independently on each position.

This consists of two fully connected layers with a ReLU activation function in between. The feedforward network is applied after the attention mechanism in both the encoder and the decoder.
The feedforward network typically has this structure:

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

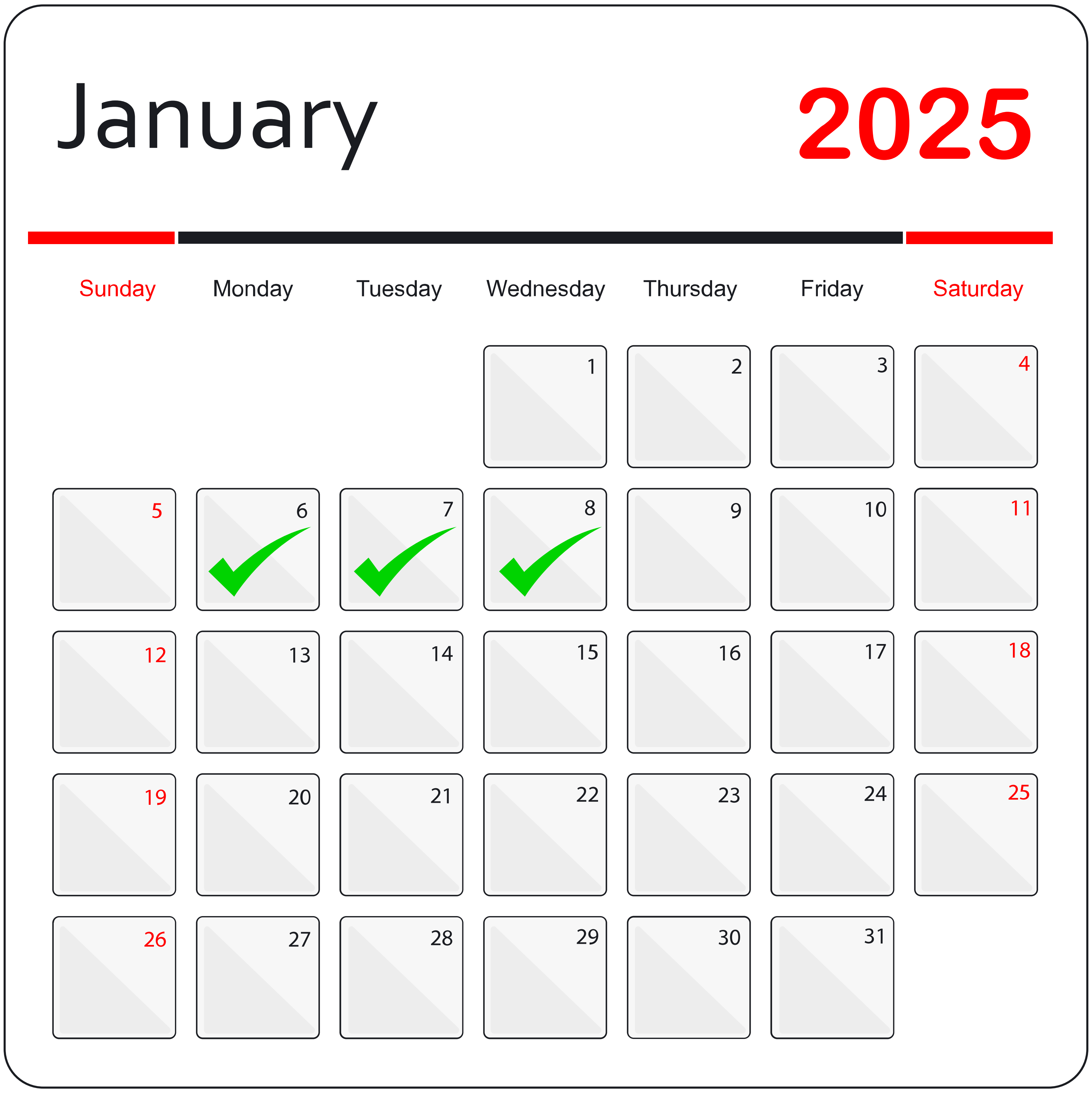
Where:

- x is the input to the layer.
- W_1, W_2 and b_1, b_2 are the learned parameters.
- The ReLU activation function is applied element-wise to $xW_1 + b_1$.

Final Output Layer

After processing the input through multiple layers of attention and feedforward networks, the final output of the decoder is passed through a linear layer followed by a softmax layer to produce the probabilities for each possible token in the vocabulary. This is how the model generates the next token in the sequence.

Analytics Vidhya



January 2025

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
			1	2	3	4
5	6 ✓	7 ✓	8 ✓	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

Stay Tuned for **Day 4** of

Mastering LLMs