**Transformer Architecture**

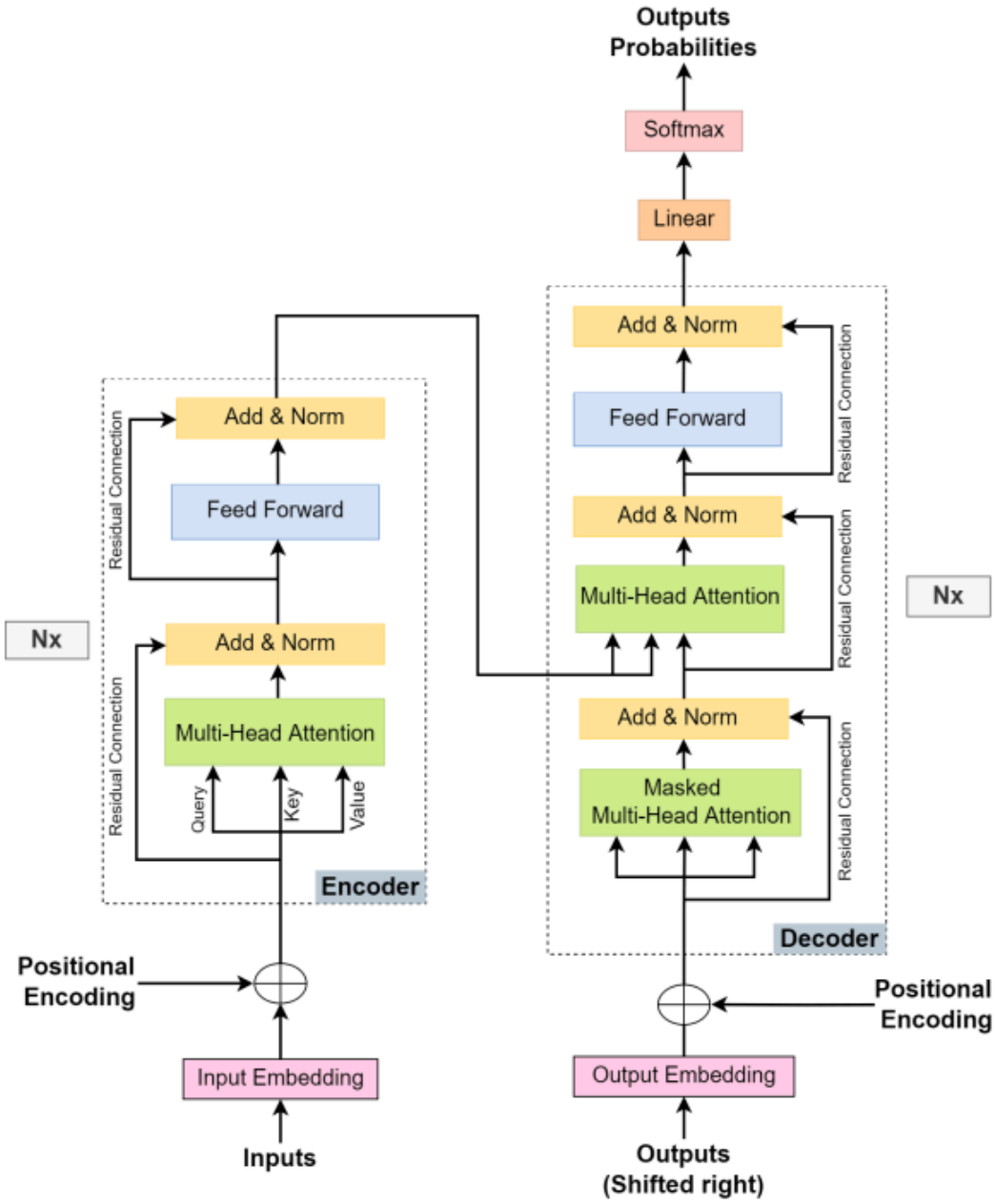
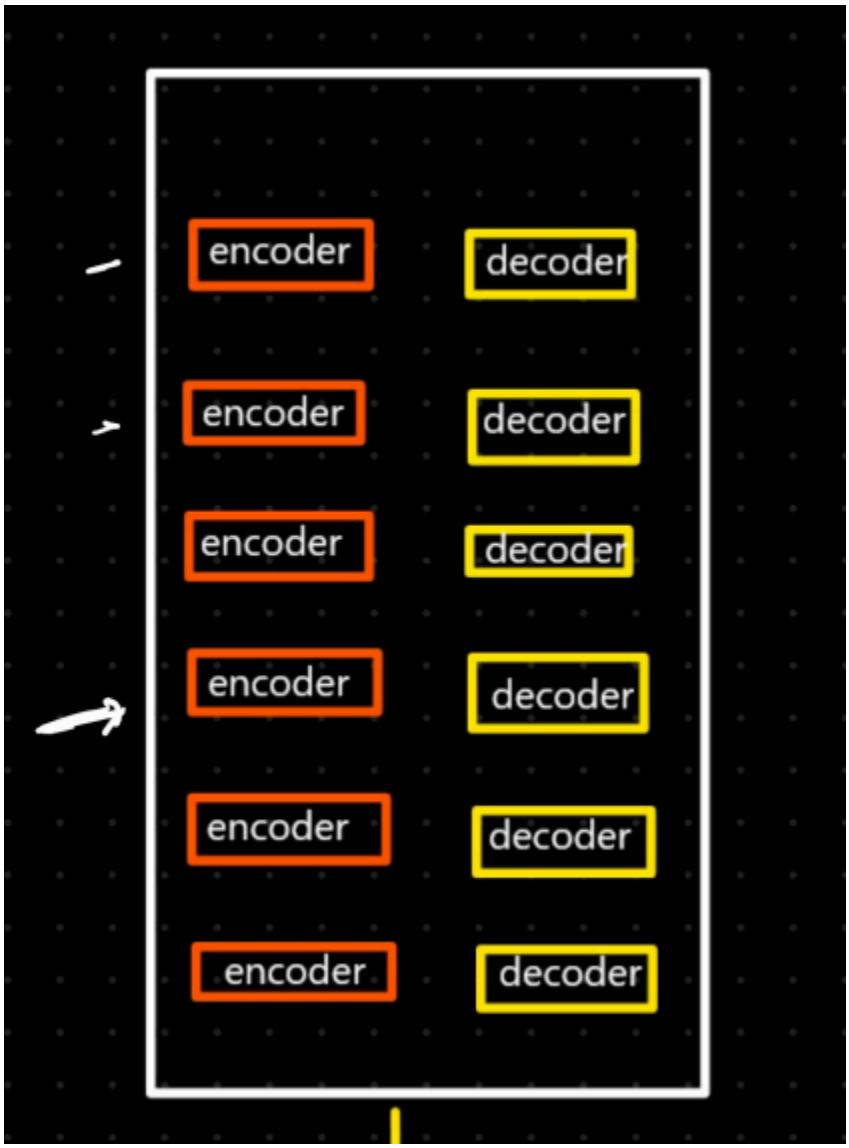



Figure 2: Transformer architecture (Vaswani et al., 2017)

In Original research paper there were set of 6 Encoders and Decoders.




Each of the Encoders has following steps involved in the **encoder side of the Transformer model**:

## 1. Input Sequence

- This is the raw data, typically a sequence of words or tokens. For example, in NLP, it could be a sentence like "I am learning Transformers."

## 2. Input Embedding

- Each token in the input sequence is converted into a fixed-size vector using an embedding layer(e.g Word2Vec or Glove). This vector represents the token in a high-dimensional space, capturing its semantic meaning.

## 3. Positional Encoding

- Since Transformers do not process the input sequentially (like RNNs), positional encoding is added to the embeddings to give the model a sense of the order of tokens in the sequence. It uses sine and cosine functions to create these encodings.

## 4. Self-Attention (Multi-Head Attention)

- **Self-attention** enables the model to focus on relevant parts of the input sequence when encoding a token. For example, in the sentence "She plays soccer," the word "She" relates to "plays."
- **Multi-head attention** allows the model to capture multiple types of relationships (e.g., syntax and semantics) simultaneously by having multiple attention heads.

## 5. Add and Layer Norm

- The output of the multi-head attention mechanism is added to the original input (residual connection) to prevent information loss.
- Then, **layer normalization** is applied to stabilize and normalize the data, improving training efficiency.

**why the residual connection is important** and how it works in the **Transformer model**.

### Why Add Residual Connections?

- **Prevent Information Loss**:
    - In deep networks (like Transformers), as data passes through multiple layers, there is a risk of losing the original information. This is because each layer transforms the input, and small errors can accumulate.
    - By adding the original input back to the output of the layer, we preserve the original information and allow the model to focus on learning only the "residual" (the difference between the input and the output).
- **Help with Gradient Flow**:
    - When training deep networks, gradients (used to update model weights) can become very small (vanishing gradients problem), making it harder for the model to learn.
    - Residual connections allow gradients to flow directly back to earlier layers, making it easier for the model to learn.

- **Stabilize Training**:
    - By combining the original input with the transformed output, residual connections help prevent extreme changes to the data, leading to more stable and faster training.

## How Does it Work in Multi-Head Attention?

1. **Input to the Multi-Head Attention Layer**:
    - Let's say we have an input matrix **X**. This could be the embeddings of words or tokens (after positional encoding).
2. **Multi-Head Attention Mechanism**:
    - The model processes the input **X** to compute attention weights and produce an output. Let's call this output **Attention_Output**.
    - The output captures relationships between words in the input (e.g., which words are important for understanding the current word).
3. **Add Residual Connection**:
    - The original input **X** is added to **Attention_Output**: \text{Residual_Output} = X + \text{Attention_Output}
    - This ensures the original information in **X** is preserved while incorporating new relationships learned by the attention mechanism.
4. **Layer Normalization**:
    - After adding the residual connection, layer normalization is applied to scale the data and stabilize learning. It ensures that the input to the next layer is well-behaved (has zero mean and unit variance).

## Why Is This Important for Transformers?

Transformers are deep networks with multiple encoder and decoder blocks (6 or more in the original paper). Without residual connections:

- The model might "forget" important parts of the original input as it passes through multiple layers.
- Training could become slower and harder because gradients would struggle to reach the earlier layers.

Residual connections solve both of these issues by **directly connecting earlier layers to later ones**.

## Think of an Analogy

Imagine you're writing a report and adding comments to each section to refine it. Instead of discarding the original draft, you keep it alongside your comments. This way:

- You don’t lose the original information (like the residual connection preserves the input).
- Your comments (like the output of the attention mechanism) refine the original draft without replacing it entirely.

Let’s dive into **layer normalization** and why it’s crucial for **stabilizing and normalizing data** in Transformer models, especially after operations like **residual connections**.

## What is Layer Normalization?

Layer Normalization (LayerNorm) is a technique used in deep learning to normalize the activations (output values) of a layer across all neurons in that layer. It ensures that the data fed into the next layer has a consistent scale, improving training stability and efficiency.

## Why is Layer Normalization Needed?

When data flows through a neural network, it is transformed at each layer. This transformation can cause:

1. **Exploding or Vanishing Activations**:
    - The values (activations) can become too large (explode) or too small (vanish), making it difficult for the network to learn effectively.
2. **Instability During Training**:
    - As weights are updated during backpropagation, small differences in activations can amplify over layers, leading to instability.
3. **Inconsistent Data Scale**:
    - If the input to a layer has varying scales, it can make optimization harder for the next layer, slowing down training.

Layer normalization solves these issues by normalizing the output at each layer.

### How Does Layer Normalization Work?

Layer normalization normalizes the data across all the neurons in a single layer. Here's how it's done step-by-step:

1. **Compute the Mean ($\mu$)**:
   - Calculate the mean of all activations in a layer for a given input.
   - Formula:
     $$\mu = \frac{1}{H}\sum_{i=1}^{H} x_i$$
     Here, $H$ is the number of neurons in the layer, and $x_i$ is the activation of the $i$-th neuron.
2. **Compute the Variance ($\sigma^2$)**:
   - Measure how much the activations deviate from the mean.
   - Formula:
     $$\sigma^2 = \frac{1}{H}\sum_{i=1}^{H}(x_i - \mu)^2$$
3. **Normalize**:
   - Subtract the mean and divide by the standard deviation ($\sqrt{\sigma^2 + \epsilon}$) for numerical stability ($\epsilon$ is a small constant).
   - Formula:
     $$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$
4. **Apply Learnable Parameters**:
   - After normalization, scale and shift the normalized values using learnable parameters $\gamma$ (scale) and $\beta$ (shift):
     $$y_i = \gamma \hat{x}_i + \beta$$
   - This allows the network to "undo" the normalization if needed and learn more flexible transformations.

## Why Does Layer Normalization Improve Training Efficiency?

1. **Stabilizes Activations**:
   - Prevents extreme changes in activations from destabilizing the model, especially in deep networks like Transformers.
2. **Faster Convergence**:
   - By ensuring consistent input scaling, the optimizer can make more effective updates, leading to faster convergence during training.

3. **Avoids Gradient Issues**:
    - Reduces the likelihood of vanishing or exploding gradients, improving backpropagation across multiple layers.
4. **Works Well with Variable-Length Sequences**:
    - Unlike batch normalization (which normalizes across a batch of data), layer normalization works on individual inputs, making it ideal for tasks like NLP, where inputs (sentences) can vary in length.

### How It Fits in the Transformer Architecture

After the **residual connection**, the data can have varying scales because it combines two components:

1. The **output of the layer** (e.g., multi-head attention or feed-forward network).
2. The **original input**.

To ensure that this combined data is in a suitable range for the next layer, layer normalization is applied.

---

### Think of an Analogy

Imagine you're working on a group project where everyone writes parts of a report. Each person's writing style is different (like activations with varying scales). Layer normalization acts as an editor, ensuring all sections of the report have a consistent tone and formatting, making the final report easier to read (or, in our case, easier for the next layer to process).

## 6. Feed Forward Network (ANN)

- A fully connected feedforward network processes the output of the self-attention layer. This network is applied independently to each position in the sequence.

  **Question:**

  **Reg. step " Feed Forward Network (ANN)" , how is "non linearity of the data" related to it ?**

  **Answer:**

  The mention of "non-linearity of the data" in the context of the Feed Forward Network (FFN) is essential because non-linearity is what makes neural networks powerful and capable of modeling complex relationships in data. Let’s break this down step by step to understand the connection:

## What is a Feed-Forward Network (FFN)?

In the Transformer architecture, the **Feed-Forward Network** is a fully connected layer applied independently to each position in the sequence. It consists of:

1. A linear transformation (dense layer). (**hidden layer**)
2. A non-linear activation function (like ReLU or GELU). (**part of hidden**)
3. Another linear transformation (to project back to the original dimensionality).(**output layer**)

This layer is responsible for transforming the data and introducing complexity into the model.

### 🚀. Why Do We Need Non-Linearity?

Without non-linearity, a feed-forward neural network would **only learn linear relationships**, regardless of how deep the network is. Let's see why:

- A **linear transformation** (like a dense layer without a non-linear activation) simply multiplies inputs by weights and adds a bias:

  $$y = Wx + b$$

  If you stack multiple such layers, the result is still **linear**, as linear transformations composed together remain linear:

  $$y = W_2(W_1x + b_1) + b_2 = (W_2W_1)x + (W_2b_1 + b_2)$$

This means the network cannot model complex, non-linear patterns in the data (e.g., relationships with curves or interactions between variables).

### 3. Non-Linearity Breaks the Linear Behavior

To overcome this, we introduce **non-linear activation functions** (like ReLU, GELU, or sigmoid). These functions break the linearity and enable the network to learn more complex patterns.

For example:

- **ReLU (Rectified Linear Unit)** introduces non-linearity by applying:

  $$f(x) = \max(0, x)$$

  It keeps positive values as they are and sets negative values to 0. This allows the network to approximate non-linear relationships.

- **GELU (Gaussian Error Linear Unit)**, which is often used in Transformers, provides a smoother non-linear transformation:

  $$f(x) = x \cdot \Phi(x)$$

  Here, $\Phi(x)$ is the cumulative distribution function of the standard normal distribution.

### How Is This Related to "Non-Linearity of the Data"?

Real-world data often contains **non-linear relationships**. For example:

- In text (NLP), the relationship between words is not always simple. For instance:
  - The meaning of "hot dog" is non-linearly related to the meanings of "hot" and "dog."
- In general data, interactions between features may follow curves or complex relationships (e.g., quadratic, exponential).

By introducing non-linearities into the FFN, the network can capture these **complex relationships**.

In Transformers:

- After the **self-attention mechanism** computes a weighted combination of the input sequence, the FFN uses non-linear activations to transform the data in a way that enables the model to capture more sophisticated patterns and dependencies.

## 5. Example to Illustrate Linear vs. Non-Linear Behavior

- **Linear Model**: Suppose a student's grade depends linearly on hours studied:

$$\text{Grade} = 5 \times \text{Hours Studied} + 20$$

  Here, doubling the hours studied always doubles the contribution to the grade. The relationship is simple and predictable.

- **Non-Linear Model**: If a student's grade depends on both hours studied and sleep quality (non-linearly), the relationship might look like:

$$\text{Grade} = 5 \times \text{Hours Studied}^2 + 3 \times \sqrt{\text{Sleep Quality}}$$

  Here, small changes in hours studied or sleep quality could lead to non-linear changes in the grade. Such patterns require a model with non-linear components to learn effectively.

## 6. Role of FFN in Transformers

In the **Transformer architecture**, the FFN is applied after the **multi-head self-attention mechanism**. While attention captures relationships between words (or tokens), the FFN introduces non-linearity, enabling the model to:

- Transform these relationships into complex features.
- Capture patterns that cannot be represented through simple linear combinations of inputs.

---

## 7. Summary

- The **Feed-Forward Network (FFN)** in Transformers consists of linear transformations combined with **non-linear activation functions**.
- **Non-linearity** is critical because real-world data is often non-linear, and non-linear activations allow the network to learn complex patterns and relationships.
- Without non-linearities, the model would behave like a simple linear regression, unable to handle intricate, multi-dimensional relationships.

## 7. Add (Residual Connection) and Layer Norm

- Similar to step 5, a residual connection is added, followed by layer normalization. This ensures smoother gradient flow and stabilizes training.

## 8. Nx (Multiple Encoder Blocks)

- The encoder consists of multiple identical blocks (as per the original paper, there were 6 encoder blocks).
- Each block repeats steps 4 to 7, refining the representations learned by the model.