Minor in AI

Revision

Transformer

April 30, 2025

1 Transformer Architecture

The Transformer architecture was introduced in the seminal paper "Attention is All You Need" (Vaswani et al., 2017). It marked a shift in natural language processing (NLP) by replacing recurrence with a fully attention-based mechanism, enabling greater parallelization and improved performance.

Why is it important?

Traditional RNNs (Recurrent Neural Networks) and LSTMs suffer from limitations in modeling long-range dependencies and are inherently sequential. Transformers allow for efficient processing of sequences as a whole, making training faster and capturing dependencies across tokens more effectively.

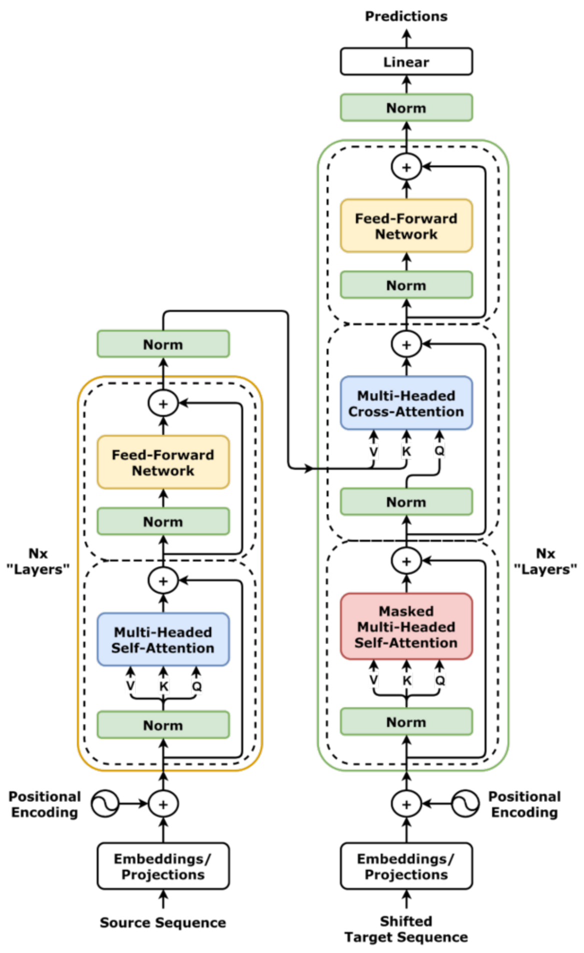

Figure 1: Transformer Architecture

Main Components:

- **Input Embedding + Positional Encoding**: Maps tokens to dense vectors and adds positional context.
- **Encoder**: Processes the input sequence and builds contextual representations.

- **Decoder**: Generates the output sequence using encoder outputs.
- **Multi-head Attention**: Captures different aspects of relationships in the sequence.
- **Feed-forward Network**: Adds non-linearity and depth.
- **Layer Normalization + Residual Connections**: Stabilizes training and allows gradient flow.

How it solves previous limitations:

By removing recurrence, the Transformer can:

- Process sequences in parallel
- Capture global dependencies through attention
- Scale to larger datasets and longer sequences

2 Positional Encoding

Transformers do not have inherent sequential order, unlike RNNs. To capture the position of each word in a sequence, we add positional encodings to the input embeddings.

What it is:

A deterministic signal (using sine and cosine functions of different frequencies) added to token embeddings to encode word positions.

Mathematical Formulation:

$$\begin{aligned} PE_{(pos,2i)} &= \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \\ PE_{(pos,2i+1)} &= \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \end{aligned}$$

Where:

- pos is the position of the word
- i is the dimension index
- d_{model} is the total embedding size

Why it is important:

Without positional encoding, the model treats the input as a bag of words. The encoding allows the model to distinguish, for example, ”cat sat mat” from ”mat sat cat”.

3 Attention Mechanism

What it is:

Attention allows the model to focus on different parts of the input sequence when generating or interpreting output. It assigns weights to tokens based on their relevance.

Why it's important:

Attention solves the bottleneck of fixed-length context in traditional models. It lets the model dynamically attend to all positions, enabling richer contextual understanding.

Scaled Dot-Product Attention:

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

Here:

- Q: Queries
- K: Keys
- V: Values
- d_k: Dimensionality of keys

How it solves problems:

Attention eliminates the need for fixed-size memory. Each token gets a dynamic representation influenced by all other tokens.

Example:

Let:

$$Q = [1, 0]$$
$$K_1 = [1, 0], \quad K_2 = [0, 1]$$
$$V_1 = [5, 5], \quad V_2 = [10, 10]$$

$$QK^T = [1, 0] \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = [1, 0]$$
$$\text{Softmax}(QK^T) = \text{softmax}([1, 0]) = [0.731, 0.269]$$
$$\text{Attention} = 0.731 \cdot V_1 + 0.269 \cdot V_2 = [6.35, 6.35]$$

4 Query, Key, Value (Q, K, V)

What they are:

- **Query** (Q): Represents what the model is searching for
- **Key** (K): Represents what the input offers
- **Value** (V): The information retrieved if the query matches the key

Why they matter:

Q, K, and V abstract the attention mechanism and allow efficient implementation of parallel dot-product attention across tokens.

Analogy:

In a search engine:

- Query = search term
- Key = indexed keywords
- Value = webpage content

5 Multi-head Attention

What it is:

Multiple attention heads running in parallel, each with different learned projections.

Why it's important:

Single-head attention is limited in representation capacity. Multiple heads allow the model to attend to different features or relationships simultaneously.

$$\begin{aligned}\text{MultiHead}(Q, K, V) &= \text{Concat}(head_1, ..., head_h)W^O \\ head_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)\end{aligned}$$

How it helps:

Each head focuses on a different part of the sentence: syntax, coreference, context, etc.

6 Layer Normalization (LayerNorm)

What is LayerNorm?

Layer Normalization is a technique used to stabilize and speed up the training of deep neural networks. Unlike Batch Normalization, which normalizes across the batch dimension, LayerNorm normalizes across the features of a single training example.

Given an input vector $x \in \mathbb{R}^d$, LayerNorm computes:

$$\mu = \frac{1}{d}\sum_{i=1}^{d} x_i, \quad \sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2}$$

Then the normalized output is:

$$\text{LayerNorm}(x) = \gamma \cdot \frac{x - \mu}{\sigma + \epsilon} + \beta$$

Where:

- γ and β are learnable parameters (scale and shift)
- ϵ is a small constant for numerical stability

Why is LayerNorm Important?

In the Transformer architecture, LayerNorm helps:

- Prevent internal covariate shift
- Maintain stable gradients during training
- Allow deeper networks to train effectively

Where is it Used in Transformers?

LayerNorm is applied before or after key sublayers in a Transformer block:

- Pre-norm: Normalize inputs before passing to self-attention or feed-forward layers
- Post-norm: Normalize outputs from those layers

Most modern Transformer variants use pre-norm due to better training stability.

Example

Let $x = [2, 4, 6]$, then:

$$\mu = \frac{2+4+6}{3} = 4$$

$$\sigma = \sqrt{\frac{(2-4)^2 + (4-4)^2 + (6-4)^2}{3}} = \sqrt{\frac{8}{3}} \approx 1.632$$

$$\text{Normalized } x = \left[\frac{2-4}{1.632}, \frac{4-4}{1.632}, \frac{6-4}{1.632}\right] \approx [-1.225, 0, 1.225]$$

With $\gamma = 1$, $\beta = 0$, the output is simply the normalized values.

7 Workflow with Example

Let's go through a simple Transformer encoder step-by-step.

Input:

"The cat sat"

Step 1: Embedding + Positional Encoding

- Tokens mapped to embeddings
- Positional encoding added

Step 2: Linear Projections

- Input embeddings projected into Q, K, V using learned weights

Step 3: Self-Attention

- Compute attention scores: $QK^T/\sqrt{d_k}$
- Apply softmax to get weights
- Use weights to compute weighted sum of V

Step 4: Multi-head Attention

- Repeat attention with multiple heads
- Concatenate all outputs and apply final linear projection

Step 5: Feed-forward Network

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

This brings non-linearity and allows deep processing of token representations.

Step 6: Add & Norm

- Residual connections: $x + \text{Sublayer}(x)$
- Followed by layer normalization to stabilize training

8 Reinforcement Learning from Human Feedback (RLHF)

What it is:

RLHF is a fine-tuning strategy to align language models with human preferences using reinforcement learning.

Why it matters:

Large language models trained on the internet can produce toxic or misaligned outputs. RLHF enables models to follow human intent more closely.

Steps:

1. **Supervised Fine-Tuning (SFT)**: Use human-annotated responses to fine-tune a base model.
2. **Reward Model (RM)**: Train a model to score generated responses based on human preference.
3. **Reinforcement Learning (RL)**: Use Proximal Policy Optimization (PPO) to reward high-quality outputs.

Where it's used:

- ChatGPT, Claude, Gemini, and other alignment-sensitive LLMs
- Models where safety, bias mitigation, and helpfulness are critical

Key Takeaways

1. **Transformers** removed the need for recurrence, enabling better parallelization and long-range dependency modeling.
2. **Positional Encoding** restores order information that is naturally lost in attention-only models.
3. **Attention Mechanism** allows dynamic weighting of tokens, making contextual representation powerful and flexible.
4. **Queries, Keys, and Values (QKV)** form the foundation of attention, representing intent, content, and information, respectively.
5. **Multi-Head Attention** lets the model learn from different perspectives and capture multiple relationships in parallel.
6. **Workflow** demonstrates that every token is encoded with rich, position-aware, and context-sensitive representations that feed into deeper layers.
7. **RLHF** aligns large language models with human values, using reinforcement learning to optimize responses preferred by humans.
8. **LayerNorm** provides normalization on the feature dimension, enabling training stability and helping gradients propagate in deep transformer stacks.
9. Together, Transformers and RLHF represent the state-of-the-art foundation and refinement mechanisms in modern language models.