The Transformer: Let's dive into math

Eng. Ahmed Métwalli - Eng. Alia Elhefny

Contents

1 Introduction

Transformers are the backbone of modern machine learning, excelling in Natural Language Processing (NLP), Vision, and Multimodal Learning. They process sequences in parallel using the **attention mechanism**, replacing sequential models like RNNs and LSTMs.

Key features:

- Handles **long-range dependencies**.
- Enables **parallel processing** of tokens.
- Scalable to large datasets and tasks.

This guide explores the Transformer's components, their inner workings, and practical implementation.

2 Core Components of the Transformer

The Transformer consists of an **Encoder-Decoder architecture**. Let us break it down.

2.1 Encoder

The encoder takes an input sequence and transforms it into a set of contextual representations. Each encoder block includes:

1. Multi-Head Self-Attention Layer.
2. Feed-Forward Neural Network (FFNN).
3. Residual Connections and Layer Normalization.

2.2 Decoder

The decoder generates the output sequence using the encoder's output and its own input. It includes:

1. Masked Multi-Head Self-Attention (prevents peeking at future tokens).
2. Encoder-Decoder Attention.
3. Feed-Forward Neural Network.

2.3 High-Level Workflow

1. Encode the input using the encoder stack.
2. Use the decoder stack to generate the output sequence, token by token.

3 Mathematics of Attention

The **Attention Mechanism** is the heart of the Transformer, allowing tokens to focus on relevant parts of the sequence.

3.1 Key, Query, and Value

For each input token, we compute:

- **Query** (Q): Encodes the focus of the current token.
- **Key** (K): Encodes how other tokens relate to the current token.
- **Value** (V): Encodes the information to pass along.

These are obtained using learned weight matrices:

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V$$

Where:

- X: Input embeddings of shape $(\text{seq\_len}, d_{\text{model}})$.
- W^Q, W^K, W^V: Weight matrices of shape (d_{model}, d_k).

3.2 Scaled Dot-Product Attention

The attention mechanism computes a weighted sum of V based on the similarity between Q and K:

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

Explanation:

1. Compute QK^T: Measures similarity between tokens.
2. Scale by $\frac{1}{\sqrt{d_k}}$: Prevents large dot-product values.
3. Apply softmax: Converts scores into probabilities.
4. Weight V by these probabilities.

3.3 Multi-Head Attention

Instead of computing a single attention, the Transformer uses **multiple attention heads**. Each head focuses on different parts of the sequence.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

Where each head is:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

3.4 Residual Connections and Normalization

Residual connections are added after attention and FFNN layers:

$$\text{Output} = \text{LayerNorm}(X + \text{Attention Output})$$

4 Positional Encoding

Since the Transformer lacks inherent sequence awareness, **Positional Encoding** is added to embeddings:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right), \quad PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right)$$

5 Numerical Example: Self-Attention

Let us compute self-attention for a toy example. [Goal of Attention: Identify relationships between tokens (queries and keys) and use them to produce weighted representations (values).]

5.1 Inputs

Given:

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \quad K = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \quad V = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

$$d_k = 2$$

5.2 Compute QK^T

$$QK^T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

5.3 Scale by $\frac{1}{\sqrt{d_k}}$

$$\text{Scaled Scores} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

5.4 Apply Softmax

Row-wise softmax normalizes the scaled scores:

$$\text{Attention Weights} = \text{softmax}\left(\frac{QK^T}{\sqrt{2}}\right)$$

5.5 Compute Output

$$\text{Output} = \text{Attention Weights} \cdot V$$

5.6 Python Code for Numerical Example

```python
import torch
import torch.nn.functional as F

Q = torch.tensor([[1.0, 0.0], [0.0, 1.0], [1.0, 1.0]])
K = torch.tensor([[1.0, 0.0], [0.0, 1.0], [1.0, 1.0]])
V = torch.tensor([[1.0, 2.0], [3.0, 4.0], [5.0, 6.0]])

d_k = Q.size(-1)
scores = torch.matmul(Q, K.T) / torch.sqrt(torch.tensor(d_k))
attention_weights = F.softmax(scores, dim=-1)
output = torch.matmul(attention_weights, V)

print("Attention Weights:\n", attention_weights)
print("Output:\n", output)
```

6 Applications of Transformers

- **NLP**: Machine Translation, Summarization, Question Answering.
- **Vision**: Vision Transformers (ViT) for Image Classification.
- **Multimodal Learning**: CLIP, DALL-E for text-to-image generation.