Introduction to Transformers in NLP

July 9, 2024

1 Introduction to Transformers

Transformers are a type of neural network architecture introduced in the paper **"Attention is All You Need"** by *Vaswani et al.* in 2017. They are designed to handle sequential data, making them particularly suited for tasks in natural language processing (NLP) such as translation, text generation, and more.

2 Transformer Architecture

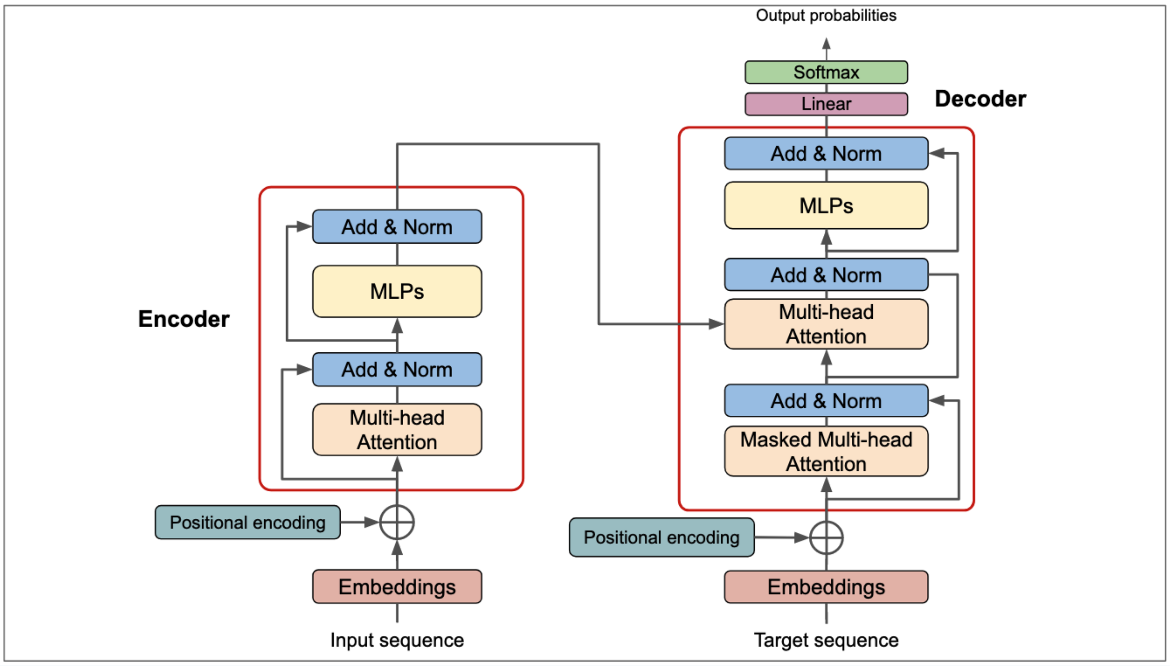

Figure 1: Transformer Architecture

2.1 Basic Components

A transformer model consists of two main parts: the **encoder** and the **decoder**.

- **Encoder**: Processes the input sequence and compresses its information into a context vector.
- **Decoder**: Takes this context vector and generates an output sequence.

In the case of GPT, only the decoder part is used.

3 Encoder in Transformers

The encoder in a transformer model processes the input sequence and transforms it into a sequence of encoded representations, often referred to as hidden states or feature vectors. Here's how it typically works:

- **Input Embedding**: Each token in the input sequence (words, characters, or subwords) is first embedded into a dense vector representation. This embedding captures semantic meaning and relationships between tokens.
- **Positional Encoding**: Since transformers do not inherently understand the order of tokens in a sequence (unlike recurrent neural networks), positional encoding is added to the input embeddings. This encoding informs the model about the position of each token in the sequence.
- **Multi-Head Self-Attention Mechanism**: The core of the encoder is the self-attention mechanism. This mechanism allows the encoder to weigh the importance of different tokens in the input sequence dynamically. It computes three vectors for each token:
  - **Query (Q)**: Represents the token used to calculate compatibility with other tokens.
  - **Key (K)**: Represents the token that determines how much attention to assign to other tokens.
  - **Value (V)**: Provides the information to be used for the output.

  These vectors are used to compute attention scores, which determine how much each token should focus on other tokens in the sequence.
- **Feed-Forward Neural Network**: After computing attention scores, the encoder applies a feed-forward neural network independently to each position (token) in the sequence. This network consists of fully connected layers and helps capture complex dependencies between tokens.
- **Normalization**: Layer normalization is applied after each sub-layer, which stabilizes and speeds up training by normalizing the outputs.
- **Residual Connections**: To mitigate the vanishing gradient problem and ease the training of very deep networks, residual connections are used. These connections allow the original input to be added to the output of each sub-layer before normalization.

4 Decoder in Transformers

The decoder in a transformer model takes the encoded representation from the encoder and generates an output sequence, step by step. Here's how the decoder works:

- **Input Embedding and Positional Encoding**: Similar to the encoder, the decoder first embeds the target sequence tokens and adds positional encoding to each embedding vector.
- **Masked Multi-Head Self-Attention Mechanism**: Unlike the encoder, the decoder uses a masked self-attention mechanism during training to prevent positions from attending to subsequent positions. This masking ensures that the model only attends to earlier positions in the output sequence, preserving the auto-regressive property during training.

- **Encoder-Decoder Attention**: In addition to self-attention, the decoder also employs an encoder-decoder attention mechanism. This mechanism allows the decoder to focus on relevant parts of the input sequence (output of the encoder) when generating each token of the output sequence.
- **Feed-Forward Neural Network, Normalization, and Residual Connections**: Similar to the encoder, the decoder applies a feed-forward neural network to each position, followed by layer normalization and residual connections.
- **Output Generation**: The final layer in the decoder outputs probabilities over the vocabulary for each token in the output sequence. During training, these probabilities are compared to the actual target tokens using cross-entropy loss to update the model parameters.

4.1 Detailed Breakdown

1. **Input Embedding**: Converts each token in the input sequence into a dense vector representation.
2. **Positional Encoding**: Adds information about the position of each token in the sequence, since transformers do not inherently handle order.
3. **Attention Mechanism**: Allows the model to focus on different parts of the input sequence when generating each part of the output.
4. **Feed-Forward Neural Network**: A fully connected layer applied to each position separately.
5. **Layer Normalization**: Stabilizes and speeds up training by normalizing the output of each layer.

5 Attention Mechanism

The core idea of transformers is the self-attention mechanism, which computes a representation of the input sequence by considering the entire sequence at each step. This allows the model to weigh the importance of different tokens dynamically.

Mathematically, self-attention can be described as follows:

- **Query (Q)**, **Key (K)**, **Value (V)**: For each token in the input sequence, these vectors are computed.
- **Attention Scores**: Calculated using the dot product of Q and K, scaled by the square root of the dimensionality of the key vectors, and then applying a softmax function.

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

Where d_k is the dimension of the key vectors.

6 Multi-Head Attention

To allow the model to focus on different parts of the sequence simultaneously, multiple sets of Q, K, and V vectors are computed. These are then concatenated and linearly transformed.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)W^O$$

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

7 Positional Encoding

Since transformers do not have a built-in notion of sequence order, positional encoding is added to the input embeddings. These encodings can be sine and cosine functions of different frequencies.

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

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

8 Encoder and Decoder Stacks

8.1 Encoder

- **Self-Attention Layer**: Computes self-attention for the input sequence.
- **Feed-Forward Layer**: Applies a feed-forward neural network to the output of the self-attention layer.
- **Add & Norm**: Residual connections followed by layer normalization.

$$\text{LayerNorm}(X + \text{SelfAttention}(X))$$

$$\text{LayerNorm}(X + \text{FeedForward}(X))$$

8.2 Decoder

- **Masked Self-Attention**: Prevents positions from attending to subsequent positions, ensuring the decoder can only attend to earlier positions in the output sequence.
- **Encoder-Decoder Attention**: Allows the decoder to attend to the encoder's output.
- **Feed-Forward Layer** and **Add & Norm** layers as in the encoder.

9 Applying Transformers in GPT

GPT (Generative Pre-trained Transformer) models use a stack of transformer decoders.

- **Training**: The model is trained on a large corpus of text data to predict the next token in a sequence. This is done by maximizing the likelihood of the sequence given the preceding tokens.
- **Generation**: At inference time, the model generates text by sampling from the distribution of possible next tokens iteratively.

9.1 Mathematical Formulation in GPT

Given an input sequence $x_1, x_2, \ldots, x_n$, the objective is to maximize the likelihood:

$$P(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} P(x_i | x_1, x_2, \ldots, x_{i-1})$$

Using transformers, this probability is modeled using self-attention mechanisms that allow each token to attend to the entire sequence before it.

10 Advanced Concepts

10.1 BERT (Bidirectional Encoder Representations from Transformers)

While GPT uses only the decoder part, BERT uses only the encoder part and is trained on a masked language modeling objective, allowing it to understand the context from both directions.

10.2 Transfer Learning

Both GPT and BERT demonstrate the power of transfer learning in NLP, where models pre-trained on large corpora are fine-tuned for specific tasks.

10.3 Differences between BERT and GPT

Aspect	**BERT (Bidirectional Encoder Representations from Transformers)**	**GPT (Generative Pre-trained Transformer)**
Architecture	• Encoder-only architecture • Uses transformer encoder blocks	• Decoder-only architecture • Uses transformer decoder blocks
Training Objective	• Masked Language Model (MLM) • Next Sentence Prediction (NSP)	• Autoregressive Language Model • Predicts next word in a sequence
Fine-tuning	• Fine-tuned for various NLP tasks • Task-specific layers added on top	• Fine-tuned for specific tasks • All layers updated during fine-tuning
Bidirectional Understanding	• Understands context bidirectionally • Captures dependencies from both directions	• Understands context unidirectionally • Processes text from left to right
Use Cases	• Question answering • Sentiment analysis • Language inference	• Text completion • Dialog generation • Machine translation

Table 1: Comparison between BERT and GPT