Reading: Beginner's Guide to Transformer Model Fundamentals

Estimated time needed: 5 mins

Overview

Generative AI Language Modeling with Transformers is the foundation of most modern AI applications you see today. In this module, you will learn the **building blocks** of Language Modeling with Transformers such as **embedding**, **positional encoding**, **self-attention**, and **optimization**. The building blocks—embedding, positional encoding, decoder, and linear layer—decide how GPT processes data. The optimization process decides how GPT learns from mistakes and improves.

A typical architecture of a custom GPT Model:

- Converts words into numbers (**embeddings**).
- Adds word positions so the model knows order (**positional encoding**).
- Uses the transformer **encoder** and **decoder** modules to read and understands the input, and then generate the output based on context. Some models only use encoders (like BERT for understanding text), while others only use decoders (like GPT for generating text).
- Maps predictions back to real words using a linear layer.

Understanding functional blocks

- **Word embeddings**
    - Converts words into numerical vectors that capture semantic meaning. Example: 'cat' → [0.21, -0.45, 0.87, …]
    - This makes it possible for the model to compare relationships, e.g., 'cat' is closer to 'dog' than 'car'. Similarly the word 'king' in English and 'rey' in Spanish need to be mapped to similar points in a 'meaning space' so the system knows they're related.

- **Positional encoding**
    - Since transformers process sequences in parallel, which means transformers look at all words at once, they need to evaluate the position information of the words.
    - Positional encoding mechanism uses signals like sine and cosine waves to capture word positions and helps the model keep track of this order. A typical formula for positional encoding for even and odd dimensions is given below:

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

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

pos: Word's position in the sequence (0,1,2,…)

i: Index inside the vector; for example, if the embedding is a four-dimensional vector, *i* can take on values 0, 1, and 2. For each slot *i*=0/1/2, you compute sin/cos with one frequency.

d_model: Total length of each word vector. Common values: 256, 512, 768, 1024, etc. If *d_model* = 512, then each word has a vector of 512 numbers.

- **Transformer decoder and encoder**
    - Transformers have two parts: encoder and decoder. In translation tasks, the encoder reads the input sentence (e.g., French) and the decoder produces the translated output (e.g., English). The decoder predicts the next word based on all previous words.
    - Attention mechanism is at the heart of both encoder and decoder modules that let the model focus on the most important words when making sense of a sentence.
    - Encoder uses **unmasked self-attention (bi-directional)**
    - Decoders use **masked multi-head self-attention** to avoid looking ahead into future tokens.
    - Attention scores determine how much one word should pay attention to another. The idea behind self-attention is to allow the model to weigh the importance of each input token when generating each output token using a mathematical operation called **scaled dot-product attention** as given below:

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

where
Q: Query vectors are derived from the source language
K: Key vectors represent the same source language and are used to compute attention scores
V: Value vectors represent the target language and are used for the final output after attention is applied
d: Dimension of the key vectors used for scaling

- **Optimization**
    - Training transformers (how the model learns/improves parameters) is an expensive and memory-intensive process.
    - To reduce the memory footprint, accelerate training, and several optimization techniques may be employed such as **gradient accumulation**, **mixed precision training**, **distributed training** and **efficient optimizers (AdamW, LAMB)**.
    - Distributed training breaks the workload across **multiple GPUs or multiple machines**, thus enabling training of **very large models** that wouldn't fit on a single device.

Summary

This reading covers the core components of language modeling: embeddings to represent words, positional encoding to track order, encoders/decoders to process and generate text, and linear layers to map outputs back to words. At its core lies the attention mechanism, enabling focus on relevant context. It also introduces key optimization techniques to improve efficiency and scalability in training.

Author(s)

Shilpa Giridhar

Skills Network