Transformer Theory

By: Yugal Singh

Key components of a Transformers:

The Transformer begins with an **embedding layer** that maps each input token to a continuous vector, generally scaled by $\sqrt{d\_model}$ and adds a positional encoding so the model knows word order. Its core is **self-attention**: for each token we compute a Query, Key, and Value vector; attention scores (SoftMax over scaled dot-products of Q and K) weight the Value vectors, obtaining a context-aware representations. In practice this is done in parallel across multiple "heads" and the whole this is known as **Mutli Head self-attention**, we can consider the Mutli Head self-attention as the multiple persons sitting in a room listing to the same thing but with **multiple perspectives**, so by using this mechanism Transformers can focus on different relationships simultaneously. After attention, each token embedding passes through a **position-wise feed-forward** network: typically, a two-layer MLP (with hidden size ≈4×d_model and ReLU/GELU activation) applied identically at each position. Each sub-layer (attention or feed-forward) is followed by a **residual connection** and **layer normalization**: the input to the sublayer is added to its output (bypassing it) and then normalized and we talk about full fledge Transformer architecture, there's also a thing called as **Cross Attention** which connects the **Encoder part of the Transformer to the it's Decoder part**. These shortcuts mitigate vanishing gradients in deep stacks and stabilize training by keeping each layer's outputs to a consistent scale.

Why self-attention scales better than RNNs/CNNs for long sequences?

As the Transformer's **self-attention mechanism** processes all the tokens at **once**, so that's why transformers scale much better to long sequences than **RNNs** or **CNNs**. RNNs must process tokens sequentially (**one timestep at a time**), while self-attention can "attend" to **any position in one layer**. And CNNs' works on the concept of **local receptive fields** means in CNNs a pixel or point only have information about its neighbouring pixels or points, but self-attention has no fixed locality – each token can immediately access any other token's information. This gives self-attention both **extreme parallelizability** and **a very short maximum path length** between any two positions, and it explains why Transformers **quickly capture long-range dependencies**.

One insight from the videos:

An insight from the video that I got is that, how each layer of attention essentially lets every token "see" the entire sentence at once. Viewing attention weight visualizations made it clear that even distant words can directly influence a token in one layer. This helped clarify why Transformers need no recurrence or convolution – they build context globally via stacked attention. For example, one explanation noted that a single attention layer gives an immediate highway of information flow across the sequence, reinforcing the idea that Transformer layers "enjoy parallel computation" and very short dependency paths.

One open question or limitation of Transformers I would like to explore:

One thing I want to explore about Transformers is **adapting Transformers to small-data regimes**. Training a Transformer from scratch on limited data tends to severely overfit, so practitioners rely on pretraining or fine-tuning. In fact, fine-tuning a pretrained model on a small labelled set "requires far less data” than training from scratch, but truly data-efficient architectures remain an active challenge. And there’s another thing I want to explore about the Transforms is that, the native attention computation scales quadratically with sequence length. I have checked in online articles that this can be addresses via sparse or linearized attention, but using sparse attention can cause problems like reduced model performance and model may miss the Important relationships between the tokens, so these are two things which I want to explore about the Transformers.