Decoder-Only Transformers

Garrett Goon

Abstract Notes on various aspects of Decoder-Only Transformers. Conventions are in the appendix.

Decoder-Only Transformers

1. Architecture

1.1. Decoder-Only Fundamentals

The Transformers architecture [1], which dominates Natural Language Processing (NLP) as of July 2023, is a relatively simple architecture. There are various flavors and variants of Tranformers, but focus here on the decoder-only versions which underlie the GPT models [2], [3], [4].

The full decoder-only architecture can be seen in Fig. Figure 1. The parameters which define the

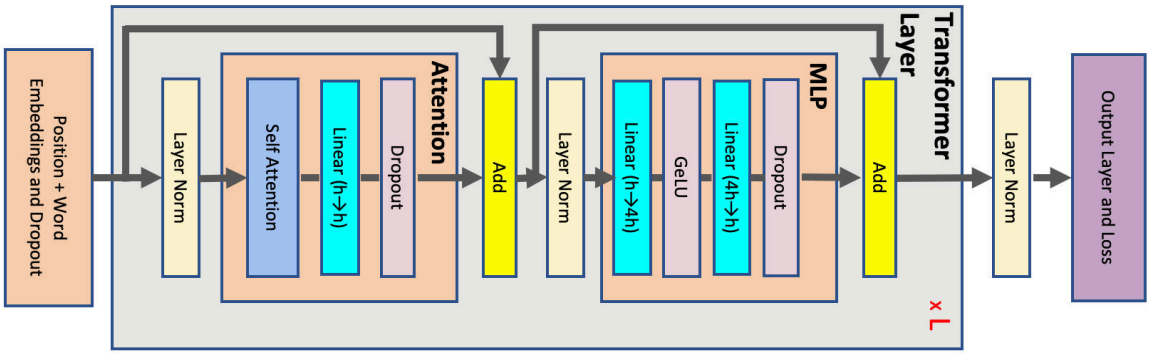

Figure 1: The full transformers architecture. Diagram taken from [5]

At a high level, decoder-only transformers take in an ordered series of word-like objects, called tokens, and are trained to predict the next token in the sequence. Given some initial text, transformers can be used to give a prediction for the likelihood of any possible continuation of that text. An outline of the mechanics[1]:

1. Raw text is **tokenized** and turned into a series of integers[2] whose values lie in , with V the vocabulary size.

[1] This describes the vanilla architecture; almost every component is modified in the available variants.

[2] There are about 1.3 tokens per word, on average.

2. The tokenized text is chunked and turned into -shaped (batch size and sequence length, respectively) integer tensors, x_{bs}.

3. The **embedding layer** converts the integer tensors into continuous representations of shape , z_{bsd}, with D the size of the hidden dimension. **Positional encodings** have also been added to the tensor at this stage to help the architecture understand the relative ordering of the text.

4. The z_{bsd} tensors pass through a series of transformer blocks, each of which has two primary components:

   1. In the **attention** sub-block, components of z_{bsd} at different positions (s-values) interact with each other, resulting in another -shaped tensor, z'_{bsd}.

   2. In the **MLP** block, each position in z'_{bsd} is processed independently and in parallel by a two-layer feed-forward network, resulting once more in a -shaped tensor.

   Importantly, there are **residual connections** around each of these[3] (the arrows in Fig. Figure 1), meaning that the output of each block is added back to its original input.

5. Finally, we convert the -shaped tensors to -shaped ones, y_{bsv}. This is the role of the **language model head** (which is often just the embedding layer used in an inverse manner.)

6. The y_{bsv} predict what the next token will be, i.e. x_{bs+1}, having seen the **context** of the first s tokens in the sequence. Specifically, removing the batch index for simplicity, a of y_{sv} gives the conditional probability $p_{sv} = P(t_{s+1} \mid t_s ... t_0)$ for the indicated series of tokens. Because of the chain rule of probability, these individual probabilities can be combined to form the probability that any sequence of tokens follows a given initial seed[4].

Each batch (the b-index) is processed independently. We omitted and layers above, as well as the causal mask; these will be covered below as we step through the architecture in more detail.

1.1.1. Embedding Layer and Positional Encodings

The **embedding** layer is just a simple look up table: each of the indices in the vocabulary is mapped to a D-dimensional vector via a large -shaped table/matrix. This layer maps $x_{bs} \to z_{bsd}$. In , this is an instance.

To each item in a batch, we add identical **positional encodings** to the vectors above with the goal of adding fixed, position-dependent correlations in the sequence dimension which will hopefully make it easier for the architecture to pick up on the relative positions of the inputs[5] This layer maps $z_{bsd} \leftarrow z_{bsd} + p_{sd}$, with p_{sd} the positional encoding tensor.

---

[3] This gives rise to the concept of the **residual stream** which each transformer block reads from and writes back to repeatedly.

[4] In more detail, these probabilities are created by products: $P(t_{s+n} ... t_{s+1} \mid t_s ... t_0) = P(t_{s+n} \mid t_{s+n-1} ... t_s ... t_0) \times ... \times P(t_{s+1} \mid t_s ... t_0)$.

[5] Positional encodings and the causal mask are the only components in the vanilla transformers architecture which carry weights with a dimension of size S; i.e. they are the only parts that have explicit sequence-length dependence. A related though experiment: you can convince yourself that if the inputs z_{bsd} were just random noise, the transformers architecture would not be able to predict the s-index of each such input in the absence of positional encodings.

The above components require $(V+S)D \approx VD$ parameters per model.

1.1.1.1. Layer Norm

The original transformers paper [1] put instances after the **attention** and **MLP** blocks, but now it is common [6] to put them before these blocks[6].

The operations acts over the hidden dimension (since this is the dimension the subsequent instances act on). Spelling it out, given the input tensor z_{bsd} whose mean and variance over the d-index are μ_{bs} and σ_{bs}, respectively, the output is

$$z_{bsd} \leftarrow \left(\frac{z_{bsd}-\mu_{bs}}{\sigma_{bs}}\right)\gamma_d + \beta_d \equiv \mathrm{LN}_d z_{bsd}$$

where γ_d, β_d are the trainable scale and bias parameters. In , this is a instance. Since there are two instances in each transformer block, these components require $2D$ parameters per layer.

We will continue discussing instances in what follows in order to adhere to the usual construction and to discuss methods like sequence-parallelism in their original form (see transformations due to γ_d, β_d are completely redundant when immediately followed by a layer, since both act linearly on their inputs and is already the most general data-independent linear transformation. Explicitly, the γ_d, β_d parameters can be absorbed into the parameters:

$$(x_{bsd}\gamma_d + \beta_d)W_{dd'} + b_{d'} = x_{bsd}W'_{dd'} + b'_{d'}\ , \quad W'_{dd'} \equiv \gamma_d W_{dd'}\ , \quad b'_{d'} \equiv b_{d'} + \beta_d W_{dd'}\ ,$$

for arbitrary x_{bsd}. That is, these transformations can be equivalently performed by the weight matrix and bias (if included) in layer[7].

1.1.2. Causal Attention

Causal attention is the most complex layer. It features A sets of weight matrices[8] $Q_{dea}, K_{dea}, V_{dea}$ where $a \in \{0, ..., A-1\}$ and $e \in \{0, ..., D/A\}$, where D is assumed perfectly divisible by A. From these, we form three different vectors:

$$q_{bsea} = z_{bsd}Q_{dea}\ , \quad k_{bsea} = z_{bsd}K_{dea}\ , \quad v_{bsea} = z_{bsd}V_{dea}$$

These are the **query, key, and value** tensors, respectively[9].

Using the above tensors, we will then build up an **attention map** $w_{bss'a}$ which corresponds to how much attention the token at position s pays to the token at position s'. Because we have the goal of predicting the next token in the sequence, we need these weights to be causal: the final prediction y_{bsv} should only have access to information propagated from positions $x_{bs'v}$ with $s' \leq$

[6] Which makes intuitive sense for the purposes of stabilizing the matrix multiplications in the blocks

[7] Note the importance of data-independence here: the data-dependent mean and standard deviation terms cannot be similarly absorbed. Also, because the usual training algorithms are not invariant under parameter redefinitions, the above unfortunately does not imply that removing the learnable parameters (in) will have no effect on training dynamics. γ_d, β_d can shoved into the layer's parameters as a small inference-time optimization, though.

[8] There are also bias terms, but we will often neglect to write them explicitly or account for their (negligible) parameter count.

[9] There are of course many variants of the architecture and one variant which is popular in Summer 2023 is multi-query attention vectors and only the query changes across heads, as this greatly reduces

s. This corresponds to the condition that $w_{bss'a} = 0$ if $s' > s$. The entire causal Transformers architecture as a whole obeys this condition: the outputs $z_{bsd} = \texttt{CausalTransformer}\ (x_{bs'd'})$ only depend on those inputs $x_{bs'd'}$ with $s' \leq s$.

These weights come from -ed attention scores, which are just a normalized dot-product over the hidden dimension:

$$w_{bss'da} = \mathrm{Sm}_{s'}\left(m_{ss'} + (q_{bse}k_{bs'ea})\left(\sqrt{\frac{D}{A}}\right)\right), \text{s.t.} \sum_{s'} w_{bdss'a} = 1$$

The tensor $m_{ss'}$ is the causal mask which zeroes out the relevant attention map components above

$$m\_\{ss'\} = \begin{cases} 0 & s \leq s' \\ -\inf = s > s' \end{cases}$$

forcing $w_{bss'da} = 0$ for $s > s'$. In other words, the causal mask ensures that a given tensor, say z_{bsd}, only has dependence on other tensors whose sequence index, say s', obeys $s' \leq s$. This is crucial for inference-time optimizations, in particular the use of the **kv-cache** in which key-value pairs do not need to be re-computed.

The $\sqrt{D/A}$ normalization is motivated by demanding that the variance of the argument be 1 at initialization, assuming that other components have been configured so that that the query and key components are i.i.d. from a Gaussian normal distribution[10].

The weights above are then passed through a dropout layer and used to re-weigh the **value** vectors and form the tensors $$\begin{aligned} y _{ bse a} & =\Dr (w _{ bdss' a}) v _{ bs'ea } \end{aligned}$$ and these -shaped tensors are then concatenated along the e-direction to re-form a -shaped tensor u_{bsd}

$$u_{bsd} = y_{bs(ea)}$$

in -like notation for concatenation. Finally, another weight matrix $O_{d'd}$ and dropout layer transform the output once again to get the final output $$\begin{aligned} z _{ bsd } & = \Dr (u _{ bsd' } O _{ d'd })\ . \end{aligned}$$

Bibliography

[1] A. Vaswani *et al.*, "Attention Is All You Need," 2017.

[2] A. Radford *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[3] T. B. Brown *et al.*, "Language Models are Few-Shot Learners," 2020.

[4] OpenAI, "GPT-4 Technical Report," 2023.

[10] However, in [7] it is instead argued that no square root should be taken in order to maximize the speed of learning via SGD.

[5] V. Korthikanti *et al.*, "Reducing Activation Recomputation in Large Transformer Models," 2022.

[6] R. Xiong *et al.*, "On Layer Normalization in the Transformer Architecture," 2020.

[7] G. Yang *et al.*, "Tensor Programs V: Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer," 2022.