LLM Interview Questions Ryan Tabrizi
(Please Do Not Share)

August 28, 2023

BERT related questions:

1. Briefly explain the structure (architectural components) of a single block. What is the purpose of using the multi-head structure?

   Solution: BERT consists only of transformer *encoder* blocks, i.e. no decoders or cross-attention (if we stick to the definition of cross-attention involving a query from a different source). The encoder block, highlighted in red below, consists mainly of skip connections, multi-head self-attention, layer normalization and a FFN.

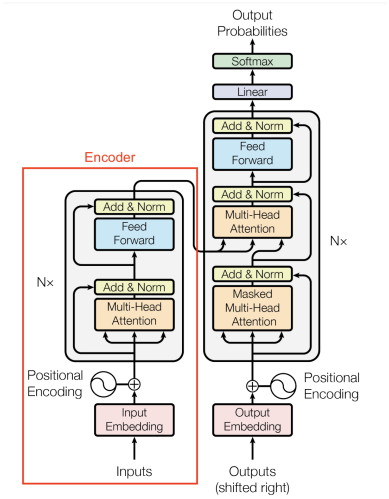


Figure 1: The Transformer - model architecture.

Figure 1: attention

The self-attention mechanism allows a given token to attend to all other tokens in the sequence, discovering inner-token relationships one would not see in an RNN or LSTM (hence the "self" attention).

A multi-head structure allows the model to attend to different components of the input sequence. The paper formally defines it as attending to "information from different representation subspaces at different positions." For instance, one head might attend to the syntax of the sequence whereas another might attend to the semantic relationships of the words. Figure 5 of the original paper demonstrates this elegantly.

2. BERT is trained in 2 separate stages: pre-training and fine-tuning. Briefly explain each of them, and what do you think is the strength of pre-training? What is the difference between fine-tuning and in-context learning (aka prompt tuning)?

   Solution: BERT's pre-training involves training the model on unlabeled data from a variety of tasks. This is done via two unsupervised tasks: masked language modeling and next sentence prediction. The inputs are sentence pairs that will be fundamental to a variety of downstream fine-tuning tasks. The bidirectional nature of BERT comes from the masking component in which 15% of the tokens are masked and BERT must learn to recover these tokens. Since not all downstream tasks involve masking, the masked words aren't always a mask token but may be a completely different word or remain the same. BERT also captures relationships between sentences through pre-training on a binarized next sentence prediction (NSP) task. Both of these tasks are relevant to the RAG as they involve establishing relationships between queries and potential answers, as well as generating missing tokens conditioned on some context.

   Once the model is pre-trained, BERT undergoes finetuning by training on labeled data from a desired downstream task. Both architectures are largely the same and all parameters are transferred over. The sequence pairs used in pre-training prove to be useful, as many downstream tasks take the form of sequence pairs like question answering, paraphrasing/summarization, and more.

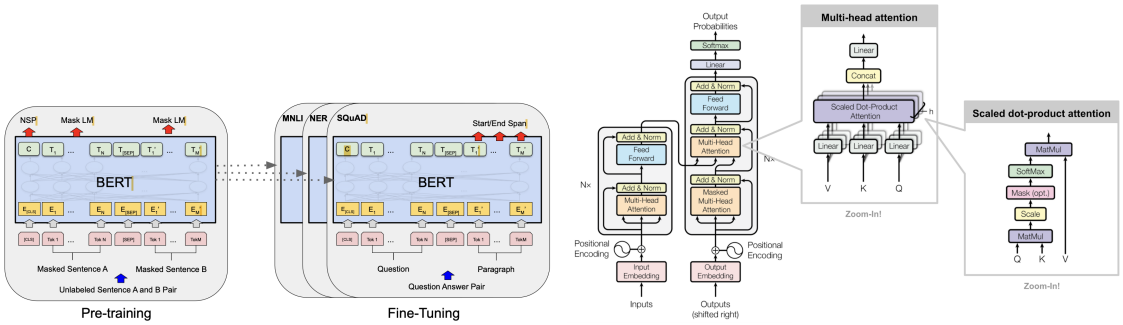
   The advantage of BERT's pre-training scheme is that it can take in large text corposuses of unlabeled data with no specific task in mind. There is plenty of accessible and unlabeled data that one can use to pre-train BERT. Not only that, but BERT operates in sentence pairs which was previously established to align well with downstream tasks.

   Fine-tuning is primarily concerned with optimizing an LLM's weights for a downstream task with supervised data. In-context learning, on the other hand, involves no parameter tuning but instead provides the LLM context through the inputted prompt itself. For example, we might feed an LLM a prompt of input-output ground truths and then have the LLM answer a final example conditioned on these ground truths. This is loosely related to BARTForConditionalGeneration in which we can prepend the retriever's fetched context to the original question and try to have the model answer the question conditioned on the context. I experiment with this in my RAG.

3. Compute the FLOPs for processing one BERT block for BERT-base with a batch size of 1. (Hint: Consider MatMul/Linear operations only.)

Solution:

Recall that BERT is effectively a stack of encoder-only transformer blocks, each of which consists of multi-head attention, skip connections, normalization layers, and a FFN (shown below).



(a) Overall BERT architecture. (b) Original Transformer architecture. image source.

Figure 2: BERT [1] and the original transformer [2]

I define my notation as follows:

Table 1: Self-Attention Parameters

Parameter	Description
$h \in \mathbb{N}$	The number of self-attention heads in $\text{BERT}_{\text{BASE}}$ ($h = 12$).
$n \in \mathbb{N}$	The context length.
$d_{\text{model}} \in \mathbb{N}$	The hidden size of $\text{BERT}_{\text{BASE}}$ ($d_{\text{model}} = 768$)
$d_{\text{FFN}} \in \mathbb{N}$	The hidden size of the FFN ($d_{\text{FFN}} = 4 * d_{\text{model}} = 3072$)
$d \in \mathbb{N}$	The hidden size for each head ($d = \frac{d_{\text{model}}}{h} = 64$)
$X \in \mathbb{R}^{n \times d_{\text{model}}}$	The input sequence of n tokens, each with size h
$W^Q, W^K, W^V \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$	The single-head attention projection matrices.
$W_j^Q, W_j^K, W_j^V \in \mathbb{R}^{d_{\text{model}} \times d}, \ j = 1 \dots h$	The multi-head attention projection matrices for head j.
$Q, K, V \in \mathbb{R}^{n \times d_{\text{model}}}$	The projected matrices for single-head attenetion.
$Q_j, K_j, V_j \in \mathbb{R}^{n \times d}, \ j = 1 \dots h$	The projected matrices for multi-head attention for head j.

single-head attention

One assumption that we make is that $d = d_q = d_k = d_v$. In the original paper, the key and value dimensions were kept the same.

From 2b, we begin with linear transformations to X with our projection matrices W^Q, W^K, W^V:

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

Multiplying $X \in \mathbb{R}^{n \times d_{\text{model}}}$ with $W^Q \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ yields n rows and d_{model} columns. Computing an arbitrary Q_{ij} involves multiplying two sets of d_{model} elements together and performing addition $d_{\text{model}} - 1$ times. Thus, for one projection, the total FLOPs is $nd_{\text{model}} \cdot (d_{\text{model}} + d_{\text{model}} - 1)$. In single-head attention, we perform this 3 times for each Q, K, V to arrive at $3nd_{\text{model}} \cdot (2d_{\text{model}} - 1) \approx 6nd^2_{\text{model}}$ FLOPs.

Upon projecting we must now perform scaled dot-product attention with our resultant matrices $Q, K, V \in \mathbb{R}^{n \times d_{\text{model}}}$ From the original paper, we define this as follows:

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

If we consider the attention map QK^T, our MatMul product has n^2 elements, each of which required 2 sets of d_{model} elements multiplied together, followed by $d_{\text{model}} - 1$ additions. Thus, this MatMul involved $n^2 \cdot (2d_{\text{model}} - 1) \approx 2n^2 d_{\text{model}}$ FLOPs.

We scale each of the n^2 elements by $\frac{1}{\sqrt{d}}$, which involves another n^2 FLOPs.

We ignore the softmax on the scaled attention map as this is not linear.

We then post-multiply our scaled softmax attention map $A := \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)$ with V to get $AV \in \mathbb{R}^{n \times d_{\text{model}}}$. This matrix has nd_{model} elements, each of which required n multiplications and $n - 1$ additions. This yields another $nd_{\text{model}} \cdot (2n - 1) \approx 2n^2 d_{\text{model}}$ FLOPs.

In single-head attention, there is no need to concatenate the results from different heads nor do we perform a final linear projection. Our work for single-head attention ends here with a final cost of $\approx 6nd^2_{\text{model}} + 2n^2 d_{\text{model}} + n^2 + 2n^2 d_{\text{model}} = 6nd^2_{\text{model}} + 4n^2 d_{\text{model}} + n^2$ FLOPs.

multi-head attention

From the attention paper:

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, \ldots, \text{head}_h) \cdot W^O \\ \text{where head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned}$$

Interestingly, the second equation is misleading and should be rewritten as

$$\text{where head}_i = \text{Attention}(XW_i^Q, XW_i^K, XW_i^V)$$

Our approach is largely the same for multi-head attention but there are slight changes w.r.t. the hidden dimensions of each head and how we process these heads in parallel. The process is outlined in figure 3.

In the original attention paper (and BERT by extension), we define the head size to be the model's hidden size divided by the number of heads: $d = \frac{d_{\text{model}}}{h} = \frac{768}{12} = 64$. This was done to achieve a cost comparable to single-head attention. Again, we assume that $d_q = d_k = d_v = d$.

We transform the input matrix $X \in \mathbb{R}^{n \times d_{\text{model}}}$ into $3h$ matrices: Q_j, K_j, and $V_j \in \mathbb{R}^{n \times d}$ for $j = 1$ to h. Each of these matrices are achieved via a projection matrix $W_j^i \in \mathbb{R}^{d_{\text{model}} \times d}$ for $i \in \{Q, K, V\}$ and $j = 1 \ldots h$.

$$Q_j = XW_j^Q, K_j = XW_j^K, V_j = XW_j^V$$

Let us compute the required FLOPs for just one head j. Consider the query matrix $Q_j = XW_j^Q$. Q_j has nd elements, each of which involved d_{model} multiplications and $d_{\text{model}} - 1$ additions which is approximately $2n \cdot d \cdot d_{\text{model}}$ FLOPs. Similarly, K_j and V_j both require an additional $2n \cdot d \cdot d_{\text{model}}$ FLOPs. Thus, the cost of computing 3 resultant projections Q_j, K_j, V_j is $6n \cdot d \cdot d_{\text{model}}$ FLOPs.

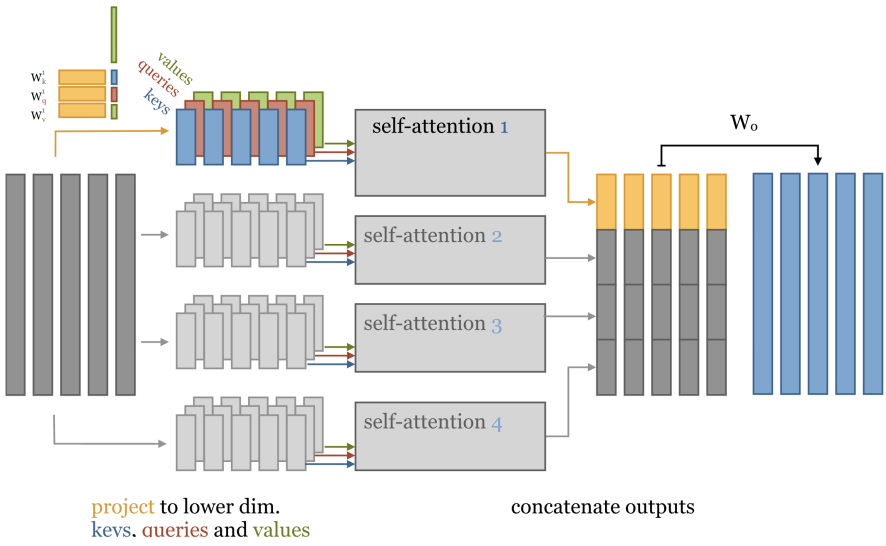


Figure 3: Multi-head attention as outlined here.

From single-head attention, we saw that the scaled dot-product attention costed $4n^2 d_{\text{model}} + n^2$ FLOPs for a head size of d_{model}. We are now working with a head size of d for a single head, yielding $4n^2 d + n^2$ FLOPs.

Thus, the FLOPs for one head is $6n \cdot d \cdot d_{\text{model}} + 4n^2 d + n^2$ FLOPs, and we must perform this h times. In multi-head attention, we concatenate these head outputs but ignore this cost for simplicity.

Across all h heads, we have $6hn \cdot d \cdot d_{\text{model}} + 4hn^2 d + hn^2$ FLOPs. Observe that $h \cdot d = d_{\text{model}}$ which suggests that our accumulated cost can be written as $6n \cdot d_{\text{model}}^2 + 4n^2 d_{\text{model}} + hn^2$.

Lastly, we must multiply our concatenated attention outputs $A \in \mathbb{R}^{n \times hd}$ by some parameter matrix $W^O \in \mathbb{R}^{hd \times d_{\text{model}}}$ where $hd = d_{\text{model}}$. The product has nd_{model} elements, each of which involves d_{model} multiplications and $d_{\text{model}} - 1$ additions yielding approximately $2nd_{\text{model}}^2$ FLOPs.

Combining all computations in multi-head attention, we compute $6n \cdot d_{\text{model}}^2 + 4n^2 d_{\text{model}} + hn^2$ FLOPs from the individual heads and an additional $2nd_{\text{model}}^2$ FLOPs from the final parameter matrix W^O, yielding $8n \cdot d_{\text{model}}^2 + 4n^2 d_{\text{model}} + hn^2$ FLOPs in total for multi-head attention.

Attention Type	FLOPs
Single-Head	$6nd_{\text{model}}^2 + 4n^2 d_{\text{model}} + n^2$
Multi-Head	$8nd_{\text{model}}^2 + 4n^2 d_{\text{model}} + hn^2$

Table 2: FLOPs associated with Single and Multi-Head Attention

If we revisit figure 1, we have yet to account for the two sets of skip connections and norms, as well as the FFN.

Residual connection and normalization

Our input $X \in \mathbb{R}^{n \times d_{\text{model}}}$ and multi-head attention output remain the same shape. The residual connection involves addition between two sets of nd_{model} elements, resulting in nd_{model} FLOPs.

As for our layer normalization, we must first compute the mean and variance for each token, followed by normalization and a final scale and shift.

Given an input $X \in \mathbb{R}^{n \times d_{\text{model}}}$, we first compute the mean and variance of each token i across all features:

$$\mu_i = \frac{1}{d_{\text{model}}} \sum_{j=1}^{d_{\text{model}}} x_j$$

$$\sigma_i^2 = \frac{1}{d_{\text{model}}} \sum_{j=1}^{d_{\text{model}}} (x_j - \mu_i)^2$$

Each of the n tokens requires $d_{\text{model}} - 1$ additions and 1 division, resulting in d_{model} FLOPs for the mean calculation of each token. To compute σ_i^2, we undergo $d_{\text{model}} - 1$ additions, where each summand requires 2 FLOPs (subtraction and squaring). We also perform 1 division, yielding $2(d_{\text{model}} - 1) + 1 = 2d_{\text{model}} - 1$ FLOPs.

Now we normalize each feature x_{ij}, the jth feature of token i:

$$\hat{x}_{ij} = \frac{x_{ij} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$$

Where ϵ is a small constant to avoid division by zero.

For each of our n tokens, we normalize d_{model} features each of which involves a subtraction, summation, exponentiation, and division (4 FLOPs total) This must be done for each of the j features of token i, resulting in $4d_{\text{model}}$ FLOPs. Note that square root (exponentiation) cost depends on implementation but we will assume 1 FLOP for simplicity.

Lastly, we must scale and shift our new features $\hat{x}_{ij}$ for $j = 1 \dots d_{\text{model}}$:

$$y_{ij} = \gamma \hat{x}_{ij} + \beta$$

This involves d_{model} multiplications and d_{model} summations for one token ($2d_{\text{model}}$ FLOPs).

Altogether for 1 token we undergo $d_{\text{model}} + (2d_{\text{model}} - 1) + 4d_{\text{model}} + 2d_{\text{model}} \approx 9d_{\text{model}}$ FLOPs for normalizing the features of token i. With n tokens in total, we require $9nd_{\text{model}}$ FLOPs.

Accounting for our residual connection, we undergo $nd_{\text{model}} + 9nd_{\text{model}} = 10nd_{\text{model}}$ FLOPs. We perform this before and after the FFN for a total of $20nd_{\text{model}}$ FLOPs.

feed-forward neural network

$$\text{FFN}(X) = \max(0, XW_1 + b_1)W_2 + b_2$$
$$\text{where } W_1 \in \mathbb{R}^{d_{\text{model}} \times d_{\text{FFN}}}, b_1 \in \mathbb{R}^{d_{\text{FFN}}}, W_2 \in \mathbb{R}^{d_{\text{FFN}} \times d_{\text{model}}}, b_2 \in \mathbb{R}^{d_{\text{model}}}$$

Define $A = XW_1 + b_1$. This MatMul costs $nd_{\text{FFN}}(2d_{\text{model}} - 1)$ FLOPs and broadcast addition with b_2 requires an additional nd_{FFN} additions, totaling $2nd_{\text{FFN}}d_{\text{model}}$ FLOPs. Matrix multiplying AW_2 is another $nd_{\text{model}} \cdot (2d_{\text{FFN}} - 1)$ FLOPs and b_2 addition involves nd_{model} additions. Thus, $\text{FFN}(X)$ requires $2nd_{\text{FFN}}d_{\text{model}} + 2nd_{\text{model}}d_{\text{FFN}} = 4nd_{\text{FFN}}d_{\text{model}}$ FLOPs.

In summary, the cost of each encoder component is outlined below, with BERT's encoder being the multi-head architecture. I avoid directly plugging in BERT-BASE's parameters as the main takeaway is our order of growth w.r.t. thet context length n for the next question.

Component	FLOPs
Single-Head Attention	$6nd_{\text{model}}^2 + 4n^2 d_{\text{model}} + n^2$
Multi-Head Attention	$8nd_{\text{model}}^2 + 4n^2 d_{\text{model}} + hn^2$
Residual Connection	nd_{model}
Layer Normalization	$9nd_{\text{model}}$
Feed Forward Network (FFN)	$4nd_{\text{FFN}}d_{\text{model}}$
Single-head Total	$6nd_{\text{model}}^2 + 4n^2 d_{\text{model}} + n^2 + 2(nd_{\text{model}} + 9nd_{\text{model}}) + 4nd_{\text{FFN}}d_{\text{model}}$
Multi-head Total (BERT)	$8nd_{\text{model}}^2 + 4n^2 d_{\text{model}} + hn^2 + 2(nd_{\text{model}} + 9nd_{\text{model}}) + 4nd_{\text{FFN}}d_{\text{model}}$

Table 3: FLOPs associated with each component of one BERT encoder block.

4. Suppose that you are using transformers for long documents. What are some possible issues that you might expect? (Hint: The previous question may help you find an answer to this question.)

   Solution: We see that our solution is quadratic in the context length n. Larger context windows are desired for longer documents, but we can see that this significantly impacts the cost to traverse through BERT. If we take GPT 3 with a context window of $2K$, we can imagine how expensive this becomes as we scale the context window even further for GPTN.

   In the case of a RAG, a larger context window will not only cause slowdowns but will also introduce noisy information into our context chunks that can then worsen our KNN search. I experienced this in my implementation as I tried to find the context window that kept all relevant information while not introducing too much verbosity.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.