Exercise Session 6

BERT, GPT and T5

Prepared by Xiuying Wei, Lukas Klein, Petr Grinberg

Overview

Task 1. Differences among BERT, T5, and GPT models **1**

Task 2. Memory Bottlenecks of Transformer models for Training and Decoding **2**

Task 3. Implementing training objectives and architectures for BERT, GPT and T5 models **3**

Additional Reading Materials. We recommend following three papers:
[1] **BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding**. Devlin, Jacob and Chang, Ming-Wei and Lee, Kenton and Toutanova, Kristina. NAACL 2019.
[2] **Language Models are Unsupervised Multitask Learners**. Radford, Alec and Wu, Jeffrey and Child, et al.
[3] **Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer**. Raffel, Colin and Shazeer, Noam and Roberts, et al. JMLR 2020

Task 1. Differences among BERT, T5, and GPT models.

Transformer-based architectures have been the foundation of many popular models, such as **BERT** (encoder-only), **T5** (encoder-decoder), and **GPT** (decoder-only). In this task, you will compare their main differences in terms of architectures and training objectives.

(a) **Encoder-only, decoder-only, and encoder–decoder architectures.**
Describe the key architectural differences among these three families, especially their different attention flows. **Hint:** Who supplies the queries Q, keys K, and values V in each case? How do the attention masks diff? You may include a small diagram to illustrate data flow.

(b) **Positional encodings.**
What positional encodings do BERT and GPT use? **Bonus:** Which does T5 use? Also mention at least one other positional encoding you know of. **Bonus:** Compare the advantages and limitations of these positional encodings.

(c) **Training pipelines and objectives.**
As discussed earlier, BERT uses masked language modeling and GPT uses an autoregressive objective. Elaborate on the training pipelines of the three models by describing how they construct the training inputs and labels. In particular, explain T5's span-corruption objective with sentinel tokens. Finally, write each model's objective function in mathematical form for a token sequence $x_{1:t}$.

Task 2. Memory Bottlenecks of Transformer models for Training and Decoding.

Transformer models are mainly composed of linear projections and scaled dot-product operations (the attention module has both and the feedforward network is mainly composed of linear projections). In the previous exercise session, we analyzed the FLOPs of each operation and identified computational bottlenecks for training and prefilling. In this exercise, we extend the analysis to memory bottlenecks in training/prefilling and decoding.

Background1. Training processes a batch of sequences in parallel: all tokens in each sequence are available at once. Prefilling is the first inference pass over an input prefix and the prefix tokens are processed in parallel. Compared with training, there is no backpropagation, but the key and value tensors in each layer's attention are cached for later decoding to save compute (this is known as the KV cache). Decoding then generates tokens autoregressively, one position at a time. At each step, the scaled dot-product operation reuses the KV cache from previous steps.

Background2. GPU memory is hierarchical. The first level is global memory (off-chip DRAM)—the "80 GB" you hear about on an A100. It offers gigabytes of capacity but is relatively slow. During training, model weights, activations, and optimizer states reside here. The second level is shared memory (on-chip SRAM), which is directly attached to the basic compute units. Therefore, when computing (e.g., matrix multiplication), the operation's inputs are loaded from global memory into shared memory, the computation is performed, and the results are written back to global memory. These data movements are memory accesses. When memory traffic dominates, latency becomes memory-bound even if FLOPs are modest. The rest of the hierarchy includes registers and the L2 cache, which we omit here for simplicity.

Memory bottlenecks appear at two levels: (1) *storage*—when global memory is not large enough to hold all model weights, optimizer states, activations, or the KV cache; and (2) *memory traffic*—when moving data from global to shared memory is slow and dominates runtime.

Notation. Let a = #heads, h = head dimension, d = model dimension ($d = ah$), t = sequence length, b = batch size, L = number of layers, and s = bytes per element (e.g., s=2 for bf16/fp16). Example: a 7B-style setting with h=128, a=32, d=4096, t=4096, and L=32. In all questions below, assume bfloat16 storage (s=2 bytes).

(a) **Memory traffic.** Analyze the memory traffic for training/prefilling and decoding modes. To keep it simple, consider from the operation level and only consider the linear projection and scaled dot-product operation (the naive implementation that you learned in the last exercise). Concretely, for each operator,

(i) compute the amount of data read and written in bytes by counting elements of inputs, weights, and outputs and multiplying by s;

(ii) compute the *arithmetic intensity* (AI) defined as FLOPs/bytes moved; and

(iii) discuss when AI of each operation is small (likely memory-bound) or large (likely compute-bound) for training and decoding.

(b) **Memory storage.** Analyze the storage problem for decoding and training.

(i) Decoding (KV cache). Consider decoding at position t+1 given a length-t prefix. Unlike training, we do not store optimizer states or most activations for backprop, but we must

store the KV cache of previous tokens. Derive the per-layer KV size and the total across L layers using the notation above. For the 7B example on an A100 80 GB GPU, and when generating a token at position 4096 (prefix length t=4095), roughly compute the maximum batch size $b_{\max}$ that can be cached.

(ii) Elaborate one storage problem in training.

Task 3. Implementing training objectives and architectures for BERT, GPT and T5 models.

See Task A in the Jupyter notebook.
