Aryan Bhardwaj

Coop: Memory is not a Commodity

The paper addresses inefficient memory usage during deep neural network (DNN) training, particularly in tensor rematerialization. Existing methods treat all free memory blocks as interchangeable, leading to fragmentation when non-contiguous tensors are evicted, causing inefficiencies and higher rematerialization costs. To address this, the authors propose "Coop," which co-optimizes tensor rematerialization and allocation. Coop evicts contiguous tensors using a sliding window and optimizes memory allocation, reducing fragmentation, computational overhead, and search latency while enabling large-scale DNN training with limited memory.

The work on optimizing tensor rematerialization and memory allocation in deep learning (DL) systems addresses a key bottleneck in training large neural networks with limited memory. Existing memory allocators in DL frameworks often cause significant fragmentation, leaving scattered, unusable memory chunks that hinder efficient allocation of new tensors. This inefficiency increases computational overhead and limits the scale of trainable models as they grow larger and more complex. By co-optimizing tensor rematerialization and memory allocation through methods like Coop, which reduces fragmentation and improves memory utilization, this work enables more efficient use of limited resources. This allows researchers to train larger models with lower computational costs, enhancing scalability and performance in real-world DL applications.

Coop manages the memory layout, which consists of the positions and sizes of tensors in the memory pool, by intercepting tensor allocation, eviction, and rematerialization operations. When memory runs out, Coop aims to evict a set of tensors that creates a contiguous memory block large enough for new tensor allocation while minimizing rematerialization costs. To achieve this, Coop uses a sliding window algorithm to identify an optimal set of contiguous tensors to evict, improving memory efficiency. It also introduces two key modules: (1) "cheap tensor partitioning" clusters computationally inexpensive tensors together to reduce the eviction cost, and (2) "recomputable in-place" ensures that memory layouts remain stable, preventing fragmentation. Together, these innovations help Coop reduce memory fragmentation and overhead while allowing the training of large-scale DNNs with lower memory budgets.

It introduces several components to optimize tensor rematerialization and allocation. The sliding window algorithm identifies which tensors to evict by selecting contiguous tensors that free up sufficient memory while minimizing rematerialization costs, preventing memory fragmentation and operating in linear time. Cheap tensor partitioning further optimizes memory layout by grouping tensors with similar eviction costs, making it easier to evict low-cost tensors while reducing the projected costs of eviction. To handle in-place operations, Coop introduces recomputable in-place, which reuses the memory of input tensors for output tensors, avoiding unnecessary memory allocations and reducing fragmentation. These strategies work together to minimize fragmentation, reduce rematerialization overhead, and enable efficient DNN training under limited memory constraints.

Coop demonstrates clear strengths by reducing compute overhead to just 11% when training BERT Large under a 50% memory ratio, compared to 29% with DTE and 41% with DTR. Additionally, Coop achieves a 60% reduction in memory usage for Inception V3, while DTR and DTE only save about half of that. Coop also maintains memory fragmentation rates below 5% across all DNNs, outperforming DTR and DTE, especially with BiLSTM and BERT Large models. However, Coop cannot be used alongside CUDA's built-in memory pool, limiting its compatibility with the stream-ordered memory allocator, which allows memory sharing between multiple programs. Additionally, as an online method, Coop provides efficient but potentially suboptimal solutions compared to offline methods like Checkmate, which can find better solutions but require significantly more time and computational resources.

Yes, there is room for improvement in Coop. One possible direction is enhancing compatibility with CUDA's built-in memory pool, allowing better memory sharing between programs without sacrificing efficiency. Another area for improvement is exploring hybrid approaches that combine the strengths of both online and offline methods. For example, integrating elements of offline optimization, such as pre-computed optimal eviction strategies, could help improve the solution accuracy without significantly increasing overhead. Additionally, addressing edge cases where Coop's performance may falter, such as fine-tuning its eviction strategies for specific deep learning models, could lead to even more consistent gains across different architectures.