DASlab Research Projects (Fast AI) HARVARD School of Engineering and Applied Sciences

Project 1: Sparsely Gated Mixture of Experts (MoE)

Introduction

Mixture of Experts (MoE) is an ensemble learning technique to boost the accuracy of deep learning models where instead of using one model or expert for inference, one deploys several experts and aggregates their individual predictions to output the final prediction. In case of Gated MoE's one usually trains a gate (multiplexor) that selects K experts among the available M experts, given an input. The gate is typically a light weight predictor trained along with the experts themselves.

Project Ideas

Idea 1: K is a hyperparameter and once initialized, stays the same for every input sample [1,3]. However, prior work has shown that different input samples have different complexity and can benefit from different amounts of computation [2]. The core intuition is that not all samples require the same amount of computation, simple samples might require less and complex examples might require more. We propose a new technique called delta-MoE, where the budget K is not fixed but is dynamically selected based on the input sample. We show how by doing this we can get better accuracy and throughput as compared to MoE.

Idea 2 (Bonus) : When using different ensemble inference strategies like a sparsely gated mixture of experts with dynamic per sample budgets [3], we often end up in scenarios where each expert receives a variable number of samples to infer. We propose a batching/queuing strategy to alleviate this problem. Incoming samples to every expert are placed in a queue of size B. Samples are processed only when the queue fills up. The hyperparameter B helps us to tradeoff between throughput and per sample latency. B =1 is best for latency and B=(whatever fully utilizes the machine) is best for throughput.

Project Milestones

1. **1.** Literature Review and Experiment planning (2 weeks)
   Read the papers in the references and have a clear understanding how Gated MoEs work. Select the types of models, sizes of models, number of models, datasets and GPUs to use for experiments. Get access to the FASRC cluster and complete the experimental setup.
2. Idea Validation Experiments (2.5 weeks)
   a. K vs. accuracy. Take existing MoE implementation and vary the budget k from 1 to K. For every value of k, capture the accuracy (as proportion of correctly classified samples). The hypothesis is that

if this accuracy number does not increase linearly to k, then different samples can benefit from a different value of k. Figure 2b in the paper [1] provides an idea of this relationship between budget and accuracy.

   b. K vs. expert load. For every k value, what is the distribution of load per expert i.e., the number of samples that every expert processes per unit time.

3. delta-MoE Implementation (2.5 weeks)
   Implement the delta-MoE idea by dynamically varying K per sample. There are multiple ways to implement this, choose which way would you like to do this and implement it.
4. Load Balancing (Bonus Task) (1 week)
   Implement Idea 2, and have queues per expert with length B. Vary the queue length and measure the throughput and latency per input sample.
5. Experimental Analysis and Write-up (1 week)
   Document your implementation and experimental analysis.

What is a successful project?

A successful project should aim to validate or invalidate the initial intuition through rigorous experimentation and analysis. The project should demonstrate the implementation of the proposed ideas upon validation of the intuition, and present comprehensive experimental results that show an increase or decrease in throughput and/or accuracy. It is important to note that a positive outcome is not necessary for the project to be considered successful, as long as the experimental analysis is thorough and well-documented.

Mid-way check-in deliverables

Milestone 1 and 2.

References

[1] https://arxiv.org/pdf/1701.06538.pdf
[2] https://arxiv.org/abs/2102.04906
[3] https://ai.googleblog.com/2022/01/scaling-vision-with-sparse-mixture-of.html

Project 2: Optimal topological ordering for reducing the peak memory usage of a DAG that represents a computational graph

Introduction

Most deep learning models today are written in one of the two popular AI frameworks PyTorch or Tensorflow. For this project we will consider PyTorch. In PyTorch, models are written as compositions of *nn.Modules*, which eventually map to a set of operators with data flowing across them. A computational graph can be obtained by tracing the execution of a single training iteration (forward + backward + optimizer), where the nodes represent the operators and the edges represent the data dependencies. Torch.compile [11] is a mechanism that enables the graph capture and allows developers to optimize the graphs using techniques like operator fusion and scheduling.

Techniques like *activation checkpointing* (AC), aim to reduce the GPU memory consumption by saving only certain activations from the forward pass and recomputing the rest in the backward pass (read this doc to understand what AC is and why we need it) [1]. To implement AC they modify the computational graph and replicate certain subgraphs from the forward graph in the backward graph. Such a graph can be executed by topologically sorting it. Given that a graph can have several topological orderings, each such ordering could result in a different amount of peak memory being consumed.

Obtaining the ordering becomes more challenging when we consider operator fusion. Operator fusion is a technique that combines the operations of two different operators into a single one such that the input and output of the new operator is the union of inputs and outputs of candidate operations respectively. Operator fusion is beneficial for reducing operation runtime but may increase the peak memory. In such cases deciding which operations to fuse becomes a compute-memory trade-off problem.

Project Ideas

Our goal is to find the optimal topological ordering of the computational graph that minimizes the peak memory consumption. We will use some techniques from the literature that allow us to do so.
Idea 1: Our goal is to identify if the graphs are series-parallel [12, 13, 14]. On verification of this property we can use efficient algorithms in [10] to come with an optimal topological order in polynomial time.
Idea 2: For graphs that do not satisfy this property, we write an ILP that will give us the optimal order. This will also be useful to establish a lower bound and a verification check for Idea 1.
Idea 3 (Bonus): Scoring operator fusion candidates and deciding which operations can be fused and determining the resulting increase in peak memory consumption if any.

Project Milestones

1. **Literature Review, System Setup and Understanding Torch.compile (2 weeks)**
   Review the papers and any related work. Get an account on the Harvard FASRC cluster and install PyTorch from source [3,4 5]. Understand the working of Torch.compile [11], its components dynamo [7] and inductor [8] and its building blocks like fx graphs [6].
2. **Benchmarking the increase in peak memory caused by torch.compile vs eager in case of activation checkpointing (1.5 weeks)**
   Measure the memory statistics of some models (discuss with TFs on which models to choose) with activation checkpointing in eager and torch.compile mode and calculate the increase in peak memory.
3. **Identification of series-parallel graphs (0.5 weeks)**
   Use ideas from [12, 13, 14] to check if the computational graphs are series-parallel and implement them in the scheduler component of the torch inductor.
4. **Implementing the polynomial time algorithm to get the topological order (1.5 weeks)**
   Implement the algorithm in [10] in the scheduler of the torch inductor.
5. **Implementing the ILP to get the optimal topological order (1.5 weeks)**
   Design an ILP that gives an optimal topological ordering that minimizes peak memory given an input DAG.
6. **Experimental analysis of Idea 1 and idea 2 and report (1 week)**
   Validate the ideas on different models, batch sizes and activation checkpointing configurations. Document your work and describe your learnings.
7. **Scoring fusion with respect to peak memory increase (Bonus) [To be added later]**

What is a successful project?

A successful project should aim to validate or invalidate the initial intuition through rigorous experimentation and analysis. The project should demonstrate the implementation of the proposed ideas upon validation of the intuition, and present comprehensive experimental results that show an increase or decrease in peak memory consumption. It is important to note that a positive outcome is not necessary for the project to be considered successful, as long as the experimental analysis is thorough and well-documented.

Mid-way check-in deliverables

Milestones 1,2 and 3.

References

[1] Activation Checkpointing
[2] GPU Fundamentals
[3] GPU Computing on the cluster

[4] FASRC Module Search
[5] > Running Jobs – FASRC DOCS
[6] Torch FX
[7] Torch Dynamo
[8] Torch Inductor
[9] Limiting the memory footprint when dynamically scheduling DAGs on shared-memory platforms
[10] Scheduling series-parallel task graphs to minimize peak memory
[11] What is Torch.compile?

What is a series-parallel graph and how to identify one?
[12] Wheatstone graph
[13] Decomposition Tree
[14] Theorem 6 (Valdes-Tarjan-Lawler)