EvoMoE: An Evolutional Mixture-of-Experts Training Framework via Dense-To-Sparse Gate

Main Idea: EvoMoE is a proposed end-to-end framework for training MoE models. The focus of EvoMoE is to deal with the issues of immature experts and unstable sparse gates (instabilities related to early stages of training, the same issue explored in StableMoE), which come from the traditional MoE framework and are harmful to convergence performance. This issue from traditional MoE is thought to come from training a sparse gate from scratch, with randomly initialized weights for both experts and router – impossible to not have router instabilities with this setup. To solve this, EvoMoE proposes starting training with a single expert, and gradually evolving that into a large and sparse MoE structure.

In sum:

- EvoMoE allows the model to warm-up before dividing it into experts.
- The gate starts as dense and gradually sparsifies, allowing it to better understand how to route inputs to experts before it reaches a high degree of sparsity.

Method

2 stages:

1. Expert-Diversify – can be seen as an improved initialization technique.

a. Start by training a single expert (so the early stages of training are the equivalent of training a dense Transformer architecture).
b. After T training steps, the single expert is replicated N times to initialize all experts. The initialization of experts from the initial expert can be done in multiple ways: adding random noise to each expert, randomly masking the initial expert's weights, etc.
    i. EvoMoE adopts the random masking strategy for initializing experts from an original warmed-up expert.
c. Once all experts are initialized, EvoMoE goes into a standard MoE period with a Dense-to-Sparse (DTS) Gate.
    i. The training of the DTS Gate is what the next stage is all about.

2. Gate-Sparsify – training the router.
    a. The router starts as a dense gate which routes the input to most experts. The idea is that at early stages of routing, the gate is not so good at its task, so would benefit from more dense routing so it can analyze the relevant experts more thoroughly, gaining more information about which experts work better from each input, instead of just using 1 or 2 experts at a time.
    b. As more training steps are done with the router, the better it becomes, so the sparser it can be. So DTS-Gate gradually becomes sparser.
    c. This stage uses an auxiliary load balancing loss.

Experiments

- Baselines
    - Switch - top-1 routing.
    - BASE – linear assignment routing.
    - Hash Layer – hashing-based routing.
    - DSelectK – differentiable routing achieved through smoothing techniques.
    - StableMoE – gate distillation and freezing for routing consistency during training.
- Evaluated on (all with 355M active parameters)
    - Machine translation - encoder-decoder setup.
    - Masked language modeling - encoder-only setup.
    - Language modeling - decoder-only setup.
- Every other FFN layer is replaced by an MoE layer (EvoMoE alternates between dense and sparse FFNs.
- EvoMoE beats other variants on all architectures (encoder, decoder, encoder-decoder) and provides training speedups.
- Both the expert-diversify and gate-sparsify stages are shown to be useful, per ablation studies.
- Compared to GPT-MoE, EvoMoE can provide a 2x training speedup (2x less training samples needed to achieve the same perplexity) as well as a 1.42x speedup in terms of FLOPs efficiency (1.42x less training FLOPs needed to achieve the same perplexity).

    - The sample efficiency and FLOPs efficiency speedups are different because EvoMoE's routing is dense during some of the gate-sparsify stage, which requires more FLOPs per training sample.
- With increasing number of experts per layer, EvoMoE shows consistent improvements.
- With increasing number of MoE layers (replacing denser FFNs by MoE layers than in the initial setup), EvoMoE shows better performance while maintaining inference FLOPs (although with a higher memory cost – more total parameters).

My takeaways:

- Research Idea – it might not be efficient to enforce load balancing due to some areas of the input space being more common than others (load balancing could cause undesired overlap in clusters at the token-level). Perhaps there could be some synergy between early-stage stability and MegaBlocks (for stable gating + no necessary load balancing at the batch level). Could also explore how custom compute depending on the complexity of the input could be implemented, and how this would perform.
- Overall, EvoMoE shows promising results. The challenge to this framework is the dense routing stage of training, which incurs high compute costs, but is a part of the trade-off for achieving better routing stability and sample efficiency.