Beyond Distillation: Task-level Mixture-of-Experts for Efficient Inference

Main idea: the goal of this work is to find an alternative method to distillation to store MoE models. It experimented with token-level, task-level and sentence-level routing. MoE solves the issue of training efficiency when compared to dense models (since only a subset of the network is activated at a time) (tradeoff of a few more communication costs due to experts' communication and routing but less parameters needing to be updated per forward pass compared to a dense model of the same size in terms of total parameters) but still leaves room for improvement in inference efficiency due to the requirement of storing the model across many devices, adding to communication costs and idle resources for calling small batches (since in small batches, most machines will not be used since the respective expert is not needed). This paper's main goal is to improve inference efficiency for sparse MoE models. Distillation is a possible solution but tends to lead to loss in quality. The task used for experiments was a multilingual machine translation task.

Approach

- Trained a routing strategy to leverage global task-level information to route all tokens corresponding to a particular task collectively to the same set of experts.
    - Decode different tasks separately and only load the subset of experts associated with the corresponding task during inference.
- Task-level routing strategy showed gains over a dense model trained from scratch and a distilled model (student) trained from learning through a token-level MoE teacher model.

- Comparable quality to token-MoE model (not distilled) while achieving significant inference gains (1.9x peak throughput and 6.3% of the decoder size).
- Top-2 routing mechanism used.

Routing Strategies Experimented With

- Token-level.
    - Traditional MoE where each token is routed independently.
- Sentence-level.
    - Route tokens by sentence, determined by the expert with the highest average token weight in the sentence.
    - First thought is that this won’t work well due to the average token weight per expert is used (this is proven to be correct by experiments done later in the paper). A better sentence-level approach could be to use sentence embeddings, which would also only need to call the router once per sentence.
- Task-level.
    - Route tokens based on a task. In the multilingual translation task, this can be determined by either the target language or the language pair.

Inference Implications

- The token-level and sentence-level approach makes inference costly. To help with the challenge of needing to have all experts ready and loaded to the server at inference, these approaches can have experts be dynamically loaded based on the routing decision or model parallelism can be employed (the server often needs to load all experts). Both incur high communication costs.
    - This needs to be done for every then, hence the high cost.
- Task-level routing only need to pre-load the top-k experts for the given input sequence. This is done by determining which task most resembles the input sequence and using the top-k experts for that task only for all tokens.
    - Loading experts only needs to be done once for each input sequence.

<u>Results</u>

- Sentence-level MoE did not perform well.
- The best encoder-decoder model used had a token-MoE in the encoder and a task-MoE in the decoder.
- The best decoder-only model was the task-MoE decoder.
- Statically determining the task through a deterministic approach did not work very well (experts are deterministically allocated to tasks).
- Task-level MoE has higher throughput (tokens/sec), uses less decoder parameters and has less communication overhead (or none) compared to token-level MoE.
- Task-level MoE performs better than models distilled from token-level MoE.

Additionally, analysis of the routing decisions shows that at a task level, the experts called in the encoder do not change much, but experts in the decoder seem to naturally specialize in tasks, giving a possible explanation why the decoder-only task MoE performed well.

<u>My takeaways</u>:

- In MoE, there is a tradeoff in training costs compared to dense models. MoE provides less communication costs overall:
    - There is a partial increase in communication costs due to the communication that needs to be done between activated experts and between these activated experts and the router.
    - Overall, however, MoE is more efficient at training due to only a subset of parameters needing to be updated per forward pass (on the MoE layers, where the bulk of parameters are located). This allows MoE to scale the total number of parameters in an easier way.
- The inspiration for the approach used comes from trying to decrease the cost of storing experts during inference.
    - This is a necessary step as all experts need to be ready to be called during inference, which leads to idle resources (no experts being used for some batches but needing to be stored and ready).
    - Distillation is (was) the most common approach for this, but distilling experts tends to lead to significant loss in quality.

    - Distillation consists of training a small dense model (student) from a large MoE expert (teacher).

- The gains obtained from inference efficiency do not come from calling less parameters at inference (number of active parameters), but from the number of experts being loaded (number of total parameters available).
  - The idea seems to be to predict the most relevant experts that will be needed on a task level, so only those need to be loaded and ready during inference.
  - The meat of this approach is to correctly predict the experts needed. If this prediction is correct, the model will have good quality, otherwise it won’t.
    - The approach seemed to work since the quality of the resulting model was comparable to token-MoE.
  - This ends up reducing the latency costs since the experts used only need to be loaded once per input sequence, and not for every token.
- The task-level approach seems to be useful in some scenarios but not possible in others. For example, if an out-of-domain task is shown at testing that is different than the training tasks, my intuition tells me that the router won’t be able to select the most relevant experts very well (and the experts won’t be prepared for this situation), thus leading to the model underperforming a token-level approach, which I believe would be more robust to these situations.
  - This approach sounds interesting in a scenario where there are predefined tasks that we want the model to perform well on, and it does not necessarily need to perform so well on out-of-domain tasks.

- This should be considered when choosing between the task-level MoE and a distilled student model (the student model, in theory, would perform better in terms of generalization – not as good in a few tasks, but good in everything -, while task-level MoE would probably perform better in specific tasks scenarios – especially good at a few tasks (depends on training)).