GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding

Main Idea:

GShard looks to make improvements on different challenges of training MoE models, particularly related to scaling. A 600B MoE model (2048E, 36L) is successfully trained, while the authors fail to reach a stable 1T parameter model (2048E, 60L) due to issues with training stability caused by reduced precision (bfloat32 to bfloat16). The improvements made were in the following topics:

- Computation costs when scaling.
- Ease of programming when scaling.
- Efficient scaling implementation on parallel devices.

GShard modifies the traditional Transformer architecture by alternating between a self-attention and a MoE layer with top-2 routing. To scale, the model is stretched vertically (increase in number of layers in each expert) and/or horizontally (increase in the number of experts per MoE layer).

For dealing with load balancing:

- A hyperparameter for a maximum threshold for the number of tokens to be sent to each expert per batch is set (expert capacity, set to N/E – N=# of tokens in the batch; E=# of experts).
    - Extra tokens (that couldn't 't make it due to the expert capacity being reached) are overflown/discarded.

- Training tokens are distributed evenly into G groups to take advantage of parallelism. Expert capacity is evaluated in a group basis – local group dispatching.
    - Experts are divided into groups that are optimized for communication (communication between experts in the same group is faster than between experts in other groups). Local communication (which is optimized) are used more between experts instead of global communication.
- Addition of a load balancing term to the loss function based on the mean number of token assignment to all experts compared to the token assignment for each expert (calculated at the group level).
- Random routing is employed to help with the expert capacity constraint. Top-2 routing requires a capacity factor of 2. To help with this, some tokens which have a low gating weight for the 2nd-best expert are not propagated through this expert (becoming top-1 routing). These 2nd-best experts are dropped randomly in proportion to the gating weight they were assigned (if assigned a score of 0.2, it would have a higher chance of being dropped than if it was assigned a score of 0.3).

Results:

- Scaling the number of layers (vertical scaling) leads to consistent gains.
- Increasing the number of experts used has diminishing returns.
- Increasing the number of experts helps with high-resource tasks (which have more data), while dense models adjust better to low-resource tasks (low amount of data).
- In terms of training efficiency:

    - Scaling with conditional computation is more practical and efficient than with dense models.
    - Deeper models are more sample efficient (converge faster with fewer examples). That is, increasing the number of layers in a model leads to an almost proportional speed up in terms of training steps to reach a certain loss (a 3x increase in number of layers would lead to ~3x speed up in training steps to reach a certain loss).
    - As mentioned previously, scaling the number of experts per-layer has diminishing returns.

My takeaways:

- GShard is the first attempt of massively scaling MoE in a Transformer architecture. It does so by optimizing the technical implementation of MoE for communication costs and parallelism.
- Highlights:
    - The techniques used for balancing dropping tokens by adjusting the expert capacity (needs to be higher for a higher k) as well as randomly dropping the 2nd-best expert are interesting.
    - The local group dispatching technique to minimize communication overhead costs also seems interesting and deserves a deeper look/understanding.
- Based on my analysis of this paper, I was left with a few questions/thoughts:
    - Are MoE layers robust to dropped tokens? As each expert is assigned to a specific input space, my first thought is that if the experts are of modest size and enough

experts are employed per layer (making the specialized input space for each expert smaller), the MoE architecture should be robust to dropping tokens.

- Are different experiments employed at future research works regarding MoE performing better at high-resource tasks and dense models performing better at low-resource tasks? This seems to mean that MoE's gains over dense models come at the expense of a bigger amount of data being needed.
- The fact that increasing the number of experts per-layer leads to diminishing returns makes sense –> as each expert specializes in a certain area of the input space, increasing the number of experts will decrease the size of that area, allowing the experts to specialize further. However, at some point the experts will become redundant (too many experts for a small input space area/cluster), leading to diminishing returns due to these redundant experts having the same specialization.
    - Based on this logic, it should be possible to balance the expert size with the number of experts in each layer. The logic is that decreasing the expert size would lead to each expert only being able to handle smaller input spaces/clusters of the data, so more experts would be needed.
- The fact that deeper models are more sample efficient hints that the scaling laws for MoE should be like those of dense models. This makes sense as adding more layers is adding computation to the model.