Mixture-of-Experts with Expert Choice Routing

Main Idea: the main goal of this paper is to tackle the limitations in the MoE architecture caused by imperfect load balancing, which leads to under-training some experts and over-training others, as well as dropping tokens, through a novel approach called Expert Choice (EC).

How Traditional MoE (Token-Choice) Works

- Pass inputs into a gating mechanism which selects the most relevant k expert(s), in a process that relies on each individual token selecting the most relevant expert (token-choice). This leads to training inefficiency as the tokens are unevenly distributed.
    - To help with this, an auxiliary loss is commonly used and added to the loss function, but this still leads to some imbalance.
- The issue of expert capacity is also prevalent, since for efficient computation, usually each expert has a fixed block size to work with in terms of token assignment. A capacity factor can be increased to minimize dropped tokens, but this leads to more memory inefficiency.
- In token-choice, each input is also assigned a fixed compute, regardless of its complexity and/or task.

Expert Choice (EC)

At a high-level, EC routing has the expert models picking the most relevant input tokens instead of the other way around.

$K = expert\ capacity = \frac{n*c}{e}$, where n is the number of tokens in a batch, c is the capacity factor hyperparameter and e is the number of experts. (c can also be thought of as the average number of experts assigned to each token). Expert capacity is the maximum number of tokens that can be assigned to each expert at a batch-level.

$\mathrm{S} = \mathrm{token - expert\ affinity} = \mathrm{SoftMax}(\mathrm{x}, \mathrm{W_g})$, where x is the input token representation and W_g is expert's g embedding.

G = matrix with weights given to each expert.

I = index matrix where I[I, j] specifies the j-th selected token of the i-th expert.

$$G, I = TopK(S_t, K)$$

$$P = OneHot(I)$$

The gating input to each expert is then determined by $X_{in} = P * X$

W1, W2 = parameters of the experts.

$$\mathrm{X_{e[i]}} = \mathrm{output\ of\ expert\ I} = \mathrm{GeLU}\left(\mathrm{X_{in[i]}} * \mathrm{W_{1[i]}}\right) * \mathrm{W_{2[i]t}}$$

OBS: EC routing has no constraints for the number of experts assigned to each token.

OBS2: The capacity factor in EC is the equivalent to top-k in token-choice -> it is the average experts assigned to each token.

Results

- EC with a capacity factor of 2 should be computationally equivalent to top-2 routing. EC-CF2 has training convergence 2x faster than GShard top-2 routing.
- Scaling the number of experts during pre-training, given the same expert size, leads to better results, as expected (more total parameters = more specialized model = better quality).
- EC-CF2 performs better than Switch top-1 and GShard top-2 in all settings, but given a fixed expert size of 100M, increasing the number of experts seems to lead to worse fine-tuning results (opposite to pre-training results).
- Capping the number of experts to be assigned to each token leads to worse fine-tuning results. This shows that allowing variable number of experts per token is indeed helpful.
- EC learns to allocate a variable number of experts per token.

My takeaways:

- Understanding the routing mechanism as an unsupervised clustering method
    - At the early stages of training a model with MoE layers, the routing mechanism (assuming it is a token-choice method and that it is learned) is random, that is, it does not have information regarding of the area it will specialize in on the embedding input space. Without load balancing, the risk is of a specific expert being disproportionately chosen at these early stages, and thus taking up a large area of the input space for itself.

    - In other words, as an expert is picked by the routing mechanism on inputs of a specific cluster that it performs well on in relation to other experts, it will gain abilities that can be generalized to other clusters that other experts still do not have, due to the lack of tokens being assigned to them. This will create a feedback loop that results in a single expert taking up more and more input space, due, again, to the generalization abilities that it picks up along the way, which will end up averaging a single dense model, since the tendency is for this over-generalized expert to take up the entire input space area for itself.
  - The addition of an auxiliary load balancing loss is added to prevent this. To visualize this, we can think of an expert trying to grow its input space area but quickly reverting to a smaller area because of penalization effects.
    - Although this is helpful, there is still the risk of non-perfect clusters being assigned to each expert, especially at a batch level, which leads to other issues like token dropping.
  - In Expert Choice, this auxiliary load balancing loss is not needed, as the experts themselves will pick the tokens that are more relevant to them at a batch level (and not the other way around). If an expert has already reached full capacity, the 2nd expert that wanted that token the most will be chosen, etc.
    - I can imagine this leading to other problems. For example, some batches will contain tons of tokens that are part of the cluster of a specific expert, but the expert won't be able to choose them because it has reached full

    capacity. In a token-choice scenario, this might lead to token dropping, which has the negative consequence of certain tokens not being used for inference (loss of information). In EC, this is not felt, but a new consequence may arise: tokens from the cluster of an expert will be given to another expert. Due to this impacting the next update, it can lead to nearby experts fighting for the input space of other nearby experts. Although this can be suboptimal at a batch level, training for many batches might neglect this effect (?).

    - Thought: Training an MoE model using token-choice and the strategy of MegaBlocks seems to be the ideal way to train a MoE model. This would get rid of the token dropping of token-choice, and not suffer from the negative consequence created by EC. The only assumption we'd have to make is that the load balancing loss and random noise penalties are a reliable way to find optimal token-expert assignments, given that token dropping is not an issue.
    - Future work idea – visualize the gating mechanism process and how it routes training inputs based on the clusters of each input embedding. Perhaps this can be done by using the checkpoints of the OpenMoE model (12 checkpoints available at HF I believe)?

- By not enforcing a constraint on the number of experts that can choose each token, EC creates a way for experts to determine how much compute will be used for each input. The idea is that the experts will learn complex and trivial inputs, maybe the intuition for this can be that complex inputs are in more complex/gray areas of the input space. With complex inputs, more experts will choose the token, leading to more computation being assigned to it. With trivial inputs that do not affect the output, no expert will choose the token, leading to no compute being applied to the token (token is dropped).

  - The difference between this token-dropping and token-choice’s token dropping is that this token dropping is learned, and not forced by lack of expert capacity.
  - This is shown to be helpful to fine-tuning performance.

- The result of increasing the number of experts helping in pre-training but hurting fine-tuning performance matches the findings of previous papers already discussed here.