MoE articles

The original MoE had 3 components:

- Experts, specialized models which are either regressors or classifiers.
- Manager (router), gate mechanism (like a softmax, for example) which decides in which area(s) of the input space each expert is trustworthy.
- Probabilistic model, which combines the expert and the manager. It joins the experts' Gaussian distributions (outputs) together based on the probability given by the manager. Y = summation of pi (probability given to expert I by the manager) * yi (output of the expert), for all experts.

This forms a fully differentiable dense ensemble of all experts with no inference speedup, as no expert computation is discarded.

Large dense neural networks are not efficient scaling in terms of training costs. Conditional computation models (sparse models) can provide advantages, but have their downfalls, such as the computational limitations of training such models (GPUs and TPUs are optimized for large matrix-matrix multiplication).

"Sparsely-Gated MoE Layer" tries to propose a solution to MoE's computation issues. When training an MoE model the deep learning way, the input is passed through the router the same way as the original MoE method, however, the router only sends the input signal through to the top-k selected experts (a discrete choice, not fully differentiable), and uses the scores given by

the router as weights of each expert's output on the final output. The final output is then a combination of the top-k experts' outputs weighted by their respective router score.

This deep learning approach has numerous potential problems:

- If one expert gets ahead and generalizes well fast, the router might send most of the data to this expert, overfitting and undertraining others while not specializing on anything. Therefore, training between experts needs to be somewhat uniform.
    - Common approaches to fix this are adding random noise to the router's probabilities (scores given to experts) in order to create some randomness in the selection of experts' process, especially in early stages of training (although we don't want this to be fully random, since it will prevent specialization) to ensure that worse performing experts are still randomly picked for updates; adding a penalty term for uneven router choice to the loss function so the router has motivation to distribute its picks in a more uniform manner. This means the loss would look like: loss = cross-entropy loss + auxiliary loss, where auxiliary loss represents the penalty term for uneven distribution.

This sparse approach is promising in some ways as it provides computational efficiency for inference (only the selected expert weights are a part of the computation). So given 8 experts of 100M parameters each and a dense model of 800M parameters, a forward pass on the MoE model using k=2 would only trigger 2*100M=200M parameters, while the dense model would always activate all 800M parameters (in reality, shared parameters should be accounted as well in MoE, but this is not mentioned here for simplicity). In theory, the quality of these 2 models

should be roughly the same since they both have the same number of total parameters available (800M).

On another hand, due to the need to balance loads through the router function, MoE can be a bit slower to train. That is, the random noise and auxiliary loss to help with router uniformity between experts can slow down training due to data being sent and updated on suboptimal places. Due to its parameter efficiency, MoE has the potential to provide significant speed ups on training steps, but due to challenges such as load balancing and communication costs incurred by MoE, the cost of each step tends to be larger, so each training step takes longer. Therefore, when comparing training speed-ups between sparse and dense models, it is important to consider both training steps and training time.