Branch-Train-Merge: Embarrassingly Parallel Training of Expert Language Models

Main Idea: This paper serves as a continuation to DEMix, focusing more on the aspect of employing techniques to train these modular models more efficiently. Due to the modularity of DEMix, Branch-Train-Merge (BTM) shows that it is possible to train these domain experts independently, saving on multi-node synchronization costs commonly required in the training of LLMs. BTM also explores scaling up the number of experts to 64 (DEMix only trained up to 16).

BTM trains an ELMForest (Expert Language Models for Efficient Sparse Training), which are embarrassingly parallel, that is, different parts of the model are independently trained on different subsets of the data, with no need for multi-node training or inference.

Each ELM is specialized in a different domain with no shared parameters (contrary to DEMix).

ELMs can be added or removed to the model at any time, or parameter-averaged to collapse back into a single LM.

Branch-Train-Merge Algorithm

- The BTM algorithm consists of repeatedly expanding the ELMForest (combination of experts) by adding experts in an embarrassingly parallel manner. There are two possible scenarios: when we are first building the forest (creating the first expert) and when we already have at least one expert created, which makes the process of initializing other experts easier.

- The addition of a new expert is done by:
    - Branch – initializing a new LM with an average of the parameters of the most relevant of the currently existing experts.
    - Train – train this recently initialized expert on new domain data.
    - Merge – merge the trained expert into the ELMForest.
- The first step (branch) needs to be done in a different manner when training the first expert since there are no experts to initialize this expert to. The training of the initial expert is done by training on heterogeneous (diverse) data.

This approach is shown to outperform dense and DEMix when used as an ensemble or when parameter-averaging the weights of the experts. This shows that there are inherent gains from training using the BTM approach.

Overall, BTM shows an efficient way of scaling LLMs without having to train extremely large models. Instead, an ensemble of domain experts, or even a parameter-average, outperforms the dense version. (the models were compared based on GPU training time; the parameter-averaged model is also compute-matched to dense).

In this work, the domains are defined by provenance (source). This is suboptimal and improved in later work.

Like DEMix, BTM has the advantages of fully adding and removing experts, if desired. Since each expert is trained on their own specific data split and there are no parameters shared, this means that removing an expert will lead to complete removal of that domain from the model. The only caveat is if other domain experts were initialized from an undesired domain. In this scenario, simply removing the undesired domain may not be sufficient.

<u>Ensembling and Averaging ELMs</u>

- Ensembling and averaging ELMs keeps the inference cost constant regardless of the number of experts added.
- Ensembling leads to higher inference costs (due to multiple expert results needed), however, results show that top-k routing should be possible.
    - The expert routing (for top-k) or score for parameter-averaging are done through the same domain posterior method from DEMix (with a cached prior, more specifically).

<u>BTM Approach (in more detail)</u>

- BTM can be done asynchronously, that is, multiple new ELMs can be trained in parallel. This can be thought of as having multiple BTM training rounds, each initializing its new experts based on the existing experts at the beginning of the training round.
- <u>Step 0</u>
    - The first ELM needs to be initialized differently, since there are no existing ELMs yet to obtain parameters to initialize an expert from.
    - For this, an initial ELM is trained on heterogeneous (diverse) data.

    - Once this initial ELM is trained, its parameters can be used to initialize the weights of the first batch of the ELMs.

- Branch
    - Refers to adding a new ELM (Expert Language Model).
    - Idea is to initialize the new ELM to be a parameter-average of the current ELMForest (all existing domain experts).
    - The best approach for initialization was to perform a weighted average of existing ELM parameters based on their domain posterior or the new domain data (finding the closest domains to the new domain and only use the parameters of the most relevant experts for this new domain).
- Train
    - After initializing the weights of the new ELM (branching), the ELM is trained independently on its domain data.
- Merge
    - Once the new ELM is fully trained on its domain data, it can be added to the ELMForest.
- It would make sense that the more ELMs exist, the less time new ELMs need to be trained for, since more ELMs means more specialized ELMs, and that the data distribution of the new domain will probably be closer to the distribution of existing domains (since there are more domains to pick from).

Initial Results

- Setup
  - ELMForest trained on 8 domains, one trained at step 0 and the remaining 7 were trained in parallel from the initial domain (only one BTM cycle done).
  - Models compared at a compute-matched basis at training.
  - 3 models used:
    - Dense Transformer - where the data from each domain is balanced.
    - DEMix – domain specialized layer (domain-level MoE).
    - ELMForest – full domain models (ELMs).
- ELMForest provides the best performance on all sizes (up to 1.3B dense), and these hold with scale.
  - However, a full ELMForest ensemble has an increased inference cost.
- ELMForest provides speedups during training (more updates per second).
  - This is justified by the reduction in cross-GPU communication for parameter synchronization (no alltoall operations needed).
- To match inference costs with dense, the ELMForest weights can be averaged. This is experimented through 3 strategies:
  - Uniform – each ELM is given the same weight.
  - Argmax – use only the ELM that is closer to the target data, equivalent to top-k with k=1.
  - Posterior – weighted average between all domains based on the domain posterior score.

    - Uniform performs worse than all other strategies, even dense.
    - Argmax performs better than dense in training domains, but worse in evaluation domains.
        - This is expected since evaluation domains (out-of-domain performance) benefit more from using shared knowledge/parameters.
    - Posterior performs better than all strategies (including dense) except for the smallest model (dense is the best in that scenario).
        - With enough training, Posterior top-k can outperform dense at the 125M scale.
- Even though Posterior parameter-averaging is promising due to improved performance over dense at the same training and inference cost, a full ensemble still provides the best results.
    - The significantly reduced inference cost from Posterior parameter-averaging makes this much more practical.

<u>Further Analysis</u>

- Ablations are made to compare the traditional BTM model with:
    - A random ensemble - same setup but each ELM is trained on a random data split, not on a specific domain. This results in an ensemble of general experts instead of an ensemble of specialized experts.

    - An ELMForest where all ELMs are randomly initialized. This should take away the effect of optimizing the initialization of new experts.
    - These 2 variations led to worse performances, so the ELMForest performance is not simply the result of ensembling parameters.
- Ablations were done to decide on how much compute should be given to the seed training (step 0) – these ablations explain and fix the underperformance of ELMForest compared to dense at the 125M scale:
    - In the initial setup used (8 training domains), the optimal amount of deed training, in relation to the total training budget, was from 40%-60%.
        - For the parameter-averaging approach, the ideal is 60%-70%, and randomly initialized ELMs (0% seed training) do not work well at all (they perform very poorly) in this setup.
    - Although not optimal, reducing seed training down to 10% of the total budget results in gains over dense and randomly initialized ELMs.
        - This shows that ELMForest performance is robust to a wide range of seed LM training compute allocations.
    - More seed training is especially useful for evaluation domains (out-of-domain performance).
- Further ablations were done using different datasets for seed training (using a 50% compute allocation to seed training).
    - The more heterogeneous (diverse) the seed data is, the better.
    - However, performance is robust to the choice of seed training corpus.

        - Even using only JavaScript code for seed training led to better performance than dense.
    - Removal of unwanted ELM domains is also robust to the seed training corpus.
        - Performance on removed domains degrades significantly when such domain is removed.

<u>Scaling the ELMForest to 64 Domains</u>

- 64 domains used for training and 16 for evaluation (80 total).
    - 4 BTM cycles are done, 16 training domains for each cycle/batch.
- The dense Transformer used for comparison:
    - 1.3B parameter model.
    - Trained for 6144 total GPU hours (using 128 GPUs).
- The 64-domain ELMForest:
    - Uses seed training of 75%.
    - 4 GPUs per ELM (4x16 = 64 GPUs used concurrently).
    - For BTM cycles/batches 2 and 3, 40 GPU hours were used for each domain, and for batch 4 20 GPU hours per domain were used.
    - The total training cost of training this 64-domain ELMForest was 2565 hours, significantly lower than training the dense model.

- Using only 40% of the dense Transformer's computational budget for training, the ELM full ensemble (not parameter-averaged) performs comparably to the Transformer (although at an increased inference cost).
    - ELMForest is especially better for training domains since the parameters for each training domain is not updated and thus not forgotten.
- Analysis shows that sparsity in the ELMForest posterior, suggesting a top-k approach can be taken to reduce inference costs:
    - Top-8 routing performs similarly to a full ensemble.
        - This means that the ensemble can be reduced to 8 experts chosen at inference without a loss in quality.
    - Top-1 routing still performs better than the dense Transformer if the Transformer was given the same amount of training.
        - Parameter-averaging performs significantly better than top-1, and almost as well as top-2 (top-2 has double the inference costs).
        - Anywhere from averaging the parameters to condense them into a single LM or using top-2 to top-8 routing seems optimal, depending on the compute available.

<u>My takeaways</u>:

- Future research can include:
    - How to improve the weights taken for parameter averaging of the ELMForest?

        - There is a hot area of research, so different techniques exist.
    - Best practices for scaling and coordinating the training of ELMForests.
    - Combining ELMForests with adapters to scale into smaller domains.
    - Unsupervised domain assignment.
        -
        - A small sampling of training data is required for calculating the domain posterior. Unsupervised assignment would get rid of this.
        - Topic of the next research paper that gives continuation from the research work by the University of Washington – “Scaling Expert Language Models with Unsupervised Domain Discovery”.
    - Recipes for leveraging ELMForests for user safety.