Scaling Expert Language Models with Unsupervised Domain Discovery

Main Idea: this research work picks up where BTM left off, adding a caveat to the framework – instead of classifying domains based on provenance (source), this is done in an unsupervised manner, assigning domain data based on clusters. The new framework is named c-BTM (cluster Branch-Train-Merge), and it still holds the embarrassingly parallel characteristic of the original BTM.

Pros of Unsupervised vs Provenance-based Domain Classification

- Not all datasets are able to be grouped based on provenance (like internet crawls).
- Groups created by provenance cannot be easily merged or divided, so one ELM is needed for each group. This is not flexible in terms of adjusting the size and number of ELMs.
- Instead of a domain posterior routing approach, which comes with many disadvantages, routing in c-BTM is done based on the distance of a document's vector to a cluster's center, a simpler and more effective approach for routing.

c-BTM vs MoE

- c-BTM routes sequences instead of tokens. This allows for different specialization in domains/tasks instead of specializing in semantics/syntax because of the routing being done at a sentence/document level, not at a token level.

- c-BTM uses offline balanced clustering (size of the clusters can be adjusted to achieve load balancing) compared to online load balancing.
- c-BTM has no shared parameters, which leads to savings in communication costs and results in a highly efficient framework for training domain experts.
- c-BTM has more interpretable experts.

OBS: c-BTM is directly compared to sparse upcycling, which mirrors how c-BTM initializes experts but instead of training ELMs, sparse upcycling substitutes the feedforward networks in the dense checkpoints by MoE layers.

<u>c-BTM Algorithm</u>

- OBS: this paper only focuses on using 1 iteration/cycle for c-BTM, meaning training all ELMs from the seed ELM (no cycles trained based on existing specialized ELMs.
- <u>Step 0: Cluster</u>
  - K-means clustering, with enforced balanced clusters (during training, not inference), is used during training.
  - Tf-idf is used since it was the best performing embedding approach.
- <u>Step 1: Branch (from seed LM)</u>
  - The seed LM is trained on a diverse corpus – experiments can be found at the BTM paper.
  - Done the same way as in BTM.
- <u>Step 2: Train</u>

    - Done the same way as in BTM.
- <u>Step 3: Merge</u>
    - Done the same way as in BTM.

<u>Inference</u>

- Sparse ensemble of outputs of existing ELMs - router chooses top-k ELMs and a weighted average of those ELMs is used.
- The router is fixed at inference and does not need to be updated after training.

<u>Experimental Setup</u>

- OPT is used as the seed LM (the dense checkpoint).
    - Both the 1.3B and the 6.7B versions of OPT were experimented with.
- K between 2 and 128 for k-means clustering was experimented with to evaluate the optimal number of ELMs.
- Dropout of 0.1 is used for all layers except the embedding layer.
- Using only the second half of a document from the embedding-based routing is shown to not result in a performance drop while leading to faster inference.
- Models are compared against each other in a compute basis. Using total training parameters is said to be misleading for sparse models, so it is not used.
    - Total GPU-time is used to evaluate training efficiency.

    - GPU-time needed for inference and latency for end-users are used to evaluate inference efficiency.

Results

- Controlling for total training tokens:
    - Using a single cluster (dense) is always the worst setup.
    - Increasing cluster count in c-BTM improves language modeling performance for a fixed compute budget (up to 16 clusters experimented with).
    - The advantage of c-BTM only increases with an increase in the number of total training tokens.
    - There is a range of optimum number of training clusters and this increases with an increase in total training tokens.
        - It is cheaper to train on more clusters in parallel, so there could be a tradeoff there.
        - This is consistent to scaling up the size of each ELM.
- Comparing training time:
    - Due to c-BTM models with higher cluster counts using fewer GPUs per expert under a fixed budget and having no communication costs between experts, c-BTM models with more clusters can be exposed to more data for the same amount of time as dense models.
        - The more clusters, the faster the training updates.

    - Training on more clusters, due to the small number of GPUs per ELM and the fact that no communication is needed between ELMs, results in a much more robust training setup to GPU failure.
    - Models trained with more clusters have faster updates as we increase the total compute.
        - Opposite is true for MoE due to communication costs between experts.
- Controlling for inference costs via parameter count:
    - The largest training budget used was 168B tokens.
    - c-BTM with top-1 routing (same inference cost as dense) outperforms dense.
        - The more clusters, the better the top-1 routing performance.
    - Top-2 to top-4 routing (of c-BTM) pretty much matches the performance of a full c-BTM ensemble (better parameter efficiency than regular c-BTM, which was top-2 to top-8).
        - Top-2 to top-4 routing sometimes even perform better than a full ensemble.
    - These conclusions hold true even when scaling the ELM count to large values (128).
- Comparing to a larger dense model:
    - 6.7B parameter dense model vs 1.3B parameter ELM c-BTM with 16 clusters and top-4 routing (5.2B inference cost) (1.3B latency cost – since the parameters of the ELMs can be run in embarrassingly parallel fashion).
    - c-BTM has a 3.5x speedup in training (using 168B training tokens).

Downstream Task Results (few-shot results)

- 6 classification tasks experimented with; 8-shot evaluations used. c-BTM trained on the C4 dataset.
- 16-cluster c-BTM always outperforms its 1-cluster version (1.3B ELM size).

Comparing to MoE (sparse upcycling aka MoE from a dense checkpoint)

- Sparse upcycling was shown to be unstable with a high number of experts (64 and 128). With 32 experts, it was shown to have stable training and perform better than the higher expert-count stable runs.
    - Based on this, the sparse upcycling model used here was a 32-expert MoE with top-2 routing.
- MoE has more expensive training due to top-2 routing.
    - Shared parameters need to communicate with each other, resulting in slower training.
- Sparse upcycling performs better at small compute budgets but performs much worse at larger budgets, even performing worse than dense models.
    - Authors speculate this could be due to distribution shifts after pretraining, which might increase the instability of sparse upcycling.

Final Analysis

- Clustering is important as random clustering underperforms it significantly.

- The load balancing constraint in k-means is shown to be useful.
    - This becomes more important with an increase in the number of clusters.
- Using the tf-idf approach, an analysis of important terms in clusters point to cluster specialization. Further analysis also shows that ELMs successfully specialize in their own cluster.
- It is shown that metadata may not correspond to the most optimal segmentation of the corpus (although it is more interpretable).
    - Since c-BTM performs better than regular BTM, with the only significant change being how the segmentation of data is done.
- Future research may investigate improving the technique to merge model weights (as this is a hot area of research).

<u>My takeaways</u>:

- Regarding the relatively low dropout used for the training of ELMs, I believe that these are more robust to overfitting than traditional MoEs due to ELMs being full networks, and thus having more parameters than a single expert FF.
    - The fact that k-means has a constraint to ensure the loads are balanced between ELMs at training time might also help with overfitting.
    - On a similar note, ELMs seem to benefit from larger batch sizes. This is also a sign that ELMs would be more robust to overfitting since the opposite is true for MoEs.

- Larger batch sizes = more accurate updates (less noise) = less regularization effect.