## Parameter-Efficient Sparsity Crafting from Dense to Mixture-of-Experts for Instruction Tuning on General Tasks

Main Idea: introduces parameter-efficient sparsity crafting (PESC), a technique consisting of transforming a dense LLM into an MoE architecture for added model capacity and making use of adapters to differentiate experts without altering their original weights. This work focuses specifically on instruction-tuning.

The motivation for this work comes mainly from leveraging the idea of sparse upcycling (converting dense into MoE) to improve LLMs' performance on instruction-tuning, since "Mixture-of-experts meets instruction-tuning: A winning combination for large language models" showed how the MoE architecture is highly effective for instruction-tuning tasks.

Method:

- Sparse upcycling:
    - From a dense pre-trained LLM, transform the FFN layers in each Transformer block into a mixture-of-experts by replicating (copying) the FFN layer n times (n being the number of experts per layer).
    - The other layers of the Transformer block (embedding, attention, normalization) remain the same.
    - Continue pre-training on this sparse architecture.
- PESC:

- Generally the same as sparse upcycling with a few caveats.
    - The dense to sparse transformation is similar, but instead of replicating the actual FFN layer, PESC initializes an adapter to represent each expert, while the FFN remains the same for each expert.
    - PESC does not continue normal pre-training as sparse upcycling, it only performs instruction-tuning.
    - PESC does not update all experts' parameters/weights, but only each expert's adapter instead for parameter-efficiency.
        - This means that we don't need n copies of the FFN parameters, but instead the equivalent of n copies of the adapter.
    - For constructing the adapter, PESC uses QLoRA.
    - Top-2 routing and auxiliary load balancing loss were used.

Parameter Efficiency Gains

While in sparse upcycling we are trying to optimize Fi (Theta(o)), where this represents the objective function in respect to all experts' parameters, in PESC we are optimizing expert adapters to approximate Fi(Theta(o)) through ~Fi(Theta(o), w(o)), where w(o) represents the adapters' weights.

This provides more efficiency in:

- Training costs, since w(o) is significantly smaller than Theta(o).
- Memory costs, since instead of replicating a full FFN layer for each expert, we are replicating an adapter for each expert, which is significantly smaller.

    - Original FFN weights are shared between experts, so only one copy per MoE layer is needed.

Experiments

- The largest PESC model trained was Camelidae-8x34B-pro (38B total parameters, ~34B activated parameters).
- Strong performance of Camelidae-8x34B-pro on benchmarks analyzed when compared to other SOTA chat models (Mixtral-8x7B-Instruct, GPT 3.5, Llama-2-70B-Chat).
    - Especially strong in knowledge and reasoning, math and coding.
    - Comparable overall performance to GPT 3.5.
- Dense vs sparse variations
    - Significant advantage of Camelidae-8x7B over Llama2-7B-Chat and Vicuna-7B, especially in more complex areas (coding and math).
    - Advantages are only amplified in the 10-20B range with Camelidae-8x13B.
    - Strong performance continues in the 30-50B range, with Cameliade-8x34B-pro outperforming the leading sparse model Mixtral-8x7B-Instruct (47B total parameters, 13B active parameters).
    - PESC effectively mitigates the knowledge forgetting issue observed in the instruction-tuning process of Camelidae's dense counterpart Camel.
    - Increasing the number of experts in the MoE layers significantly improves the model's performance.

        - Experimented with relatively low number of experts per MoE layer, from 4 to 16.
    - Increasing the number of experts in this approach seems way less costly than with a regular MoE, since we would need to add m more adapters and not m more FFNs.

My takeaways:

- The sparsity crafting idea seems to pretty much be parameter-efficient sparse upcycling applied to instruction-tuning.
- Sounds possible to practically apply this to TinyLlama-1B? Or another model in the 1-3B range.