Spring 2020

Internal Badger Workshop

Session 4: Targeting Scalability

Moderators: Joseph Davidson & Nicholas Guttenberg

- **Computational** scalability (more experts = smarter agents/wider search/etc.)
- **Problem** scalability (dimensions in guessing game)
- 'Symbiotic' experts - how to train things which improve architectures they're added to

Pre-discussion Comments & Resources

Problem scalability

Some current efforts in the guessing game have been training and testing architectures on exactly *n* experts. The guessing game is (what I would argue) an example of a scalable problem, in that we scale the problem to the number of experts. This ensures that each expert uses the input and contributes to the output. Problem scalability is a necessary prerequisite of computational scalability - where newly added experts may not have access to the I/O directly, but still communicate with pre-existing experts and contribute something to the overall computation.

Optimising directly for problem scalability has been approached presently via training with curricula, and performing gradient descent though minibatches of episodes/rollouts with randomised numbers of experts for each rollout.

Computational scalability

One idea for a benefit of going multi-agent is that computational resources might be available in a distributed fashion, which can't be used in a centralized way. E.g. if everyone runs Badger-at-home on their PC, maybe each person contributes 3-4 experts but there are hundreds of thousands running in parallel in total. So how do we write tasks which target this kind of scalability and learn to make use of it?

For scaling with computational resources, we need tasks which are computationally expensive but informationally simple. For example, calculating the 10^{80}th digit of pi doesn't require any more information than calculating the third digit, but it does require more computation. So what kinds of tasks literally just require longer run-times without requiring increased cleverness, better information flow, etc?

One thought would be to look at algorithms which people have to write fairly complex distributed code for. Could we make Badger handle the information flow, domain decomposition, ‘ghost cells’, etc of a distributed PDE solver? Could we make Badger do the same thing but for a (much harder) event-driven system or Monte-Carlo system where not only is it event-based, but those events occur in random order and position? What do those things look like specified as (SL, RL) tasks, given that there seem to be two parts - one is the accuracy of the solution, the other is the efficiency of the computation?

If we want to learn to manage a PDE solver for example, we’d probably want to insist that the actual output of the solver is the same in all cases and only the compute time varies. Otherwise the training process could focus on coming up with better approximations for the PDE solution, rather than how to better distribute the workload.

Discussion Notes

- *Scalability of life* - just like scaling in the number of experts, we need to be able to scale infinitely to the length of the inner loop, i.e. the lifetime of an agent
    - Could be achieved by inner loop locality as discussed in previous session?
- Potential benefit of the renormalization group and modelling scalability using Taylor expansion and secular terms close to 0
    - Possible link to generalization and invariance?
- Scalability might not be resolvable by a better loss function alone, but a better architecture might be necessary
- Databases and their approaches to distributedness and resiliency could provide some inspiration
- Relation of the concept of anti-fragility and how shocks can make a system stronger
    - “What doesn’t kill you makes you stronger”
    - Copper that is bent is harder to bend - it has memory
- Links to homeostasis and error correcting codes
- Holographic memory and relative lack of resilience of modularity in a system that can have holographic memory
- Mechanically targeting scalability in training
    - Hypernetworks
    - Outerloop training with a self-supervised objective