Spring 2020

Internal Badger Workshop

Session 1: Types of Inner Loop Learning

Moderators: Nicholas Guttenberg and Jaroslav Vitku

- Inner loop vs. outer loop bottleneck discussion - dig deeper into this idea, what exactly does it mean for a task to require inner loop learning?
- Supervised inner loops
- Reinforcement Learning/indirect supervision of inner loops (guessing game, other such ideas)
- 'Self-supervision': experts modifying loss functions?

Pre-discussion Comments & Resources

When we deal with two learning stages (for example an Outer Loop and an Inner Loop), there's a question of what gets learned in which loop? Oversimplifying, there's some total 'quantity' of learning that needs to take place, and some part will happen in the Outer Loop, and some part will happen in the Inner Loop. In most meta-learning setups, the Outer Loop usually learns the space of all solutions and a prior over those solutions, and the Inner Loop deals only with information which was not available during the Outer Loop training (such as 'specifically, what task is this out of the distribution?').

In Badger, we want to reverse that order. We want the Outer Loop to only learn some form of very vague strategy for learning in a smart way, and have the inner loop lead to the development of novel skills and strategies. So the difficulty is, if during the Outer Loop training we see the agent develop various skills (as that's what we want to happen in general), why not just memorize those skills in the outer loop so you don't have to learn them in the next inner loop?

The reason I call this a 'bottleneck' is not in the information bottleneck sense, but in the sense of 'where are we waiting for learning to complete?'. Right now, if we extend the Outer Loop, things improve, but if we extend the Inner Loop they don't. So the thing that determines the asymptotic performance of Badger is how much Outer Loop time we spend, not how much Inner Loop time we spend - we're bottlenecked by the Outer Loop.

Ideally we want to find algorithms for which the bottleneck lies in the Inner Loop though, so that we can train the Outer Loop on 10 or so steps of inner loop learning, but then just run the inner loop for hundreds of thousands of steps.

Some notes for potential discussion, not entirely organized

Main thing I'd want to focus on is the idea of whether the bottleneck is in the Outer Loop learning or Inner Loop learning. I think that's the biggest 'idea' of this session.

For most of the tasks/training protocols we've inspected, the bottleneck is **outer loop learning** - that is to say, we set up the task, iterate the outer loop, and see the result after the inner loop improve. However, the thing that limits how well Badger does is usually how many outer loop iterations we've performed - adding inner loop iterations does not make things better.

There is a conceptual issue here - how do we define a task that is explicitly not outer-loop limited, or maybe more positively: how do we make tasks that are explicitly inner-loop limited?

One method is hidden information - active inference tasks. For these, the information needed to solve the task can't be memorized from birth. However, this tends to look like inference where you already have strong priors, strong models of the world. It doesn't move us in the direction of general learning - and often, the improvement of the final score will have more to do with refinement of the prior during the outer loop than stuff which was picked up during the inner loop (can we measure this explicitly?)

Broadly the reason this seems hard is that, if there's something I could theoretically learn in either the inner loop or the outer loop (like 'how to do addition' or 'how to compute integrals' or 'how to program a sort'), why wouldn't I just store that information in the policy rather than making a policy that is capable of inventing those things some times, but takes time and effort to do so? Is it a model capacity issue (policy capacity has to be very small), or is there a better conceptual framework for thinking about this? In evolutionary terms, why do we boot up our visual system during childhood rather than shipping with a pre-trained visual system?

If it's a capacity thing, this may tell us something about the ratio of policy parameters to inner-loop-learnable parameters we need. Maybe we need orders of magnitude more inner loop capacity somehow (e.g. hidden states aren't nearly enough)?

Longwinded details - probably won't get to even half of all of this, but here for some potential notes/discussion if anything pops out

The 'types' here refer not to how the architecture as a whole is trained, but to the sort of meta-task which we're training on. Are we trying to get Badger to 'do' supervised learning, RL-like learning, intrinsic motivation, something else? Inner loop setup (inputs, outputs, updates) seems sensitive to these goals.

Supervised Inner Loops

Experts receive individualized gradient/error information. Compatible with MAML way of doing things. Local gradient descent logic seems to be important to getting the desired invariances - should be able to change input order, output order, # of inputs, things like that and still have the inner loop policy work.

What is insufficient about existing methods? - e.g. why Badger for this?

- Can't generalize to non-differentiable functions.
- Long rollouts/integrating large amounts of evidence is difficult for existing meta-learners (SGD is fine though).
- Lack of flexibility of input/output types.
- No good way to use explicit problem specifications (task metadata)
- Does MAML forget? Can MAML be used to make an agent that gains advantage from previous tasks?

Where is 'opportunistic AI' in this?

Big picture - why not just jump to RL?

One of the long-term views for Badger is that it should be taught the way a child would be taught. That means that supervisory signals can be rich and detailed, they don't just have to be hot/cold. May be a reason to look for alternate 'kinds' of supervision.

Potentially interesting tasks for supervised inner loops

- Routing task - we know it's hard in the RL framing, but is it still hard if supervised? We should figure this out.
- Skill transfer between experts can be investigated in this regime. Incompatible with MAML, somewhat compatible with task-representation types of approaches (e.g. teacher student experiments)
- Omniglot
- Using supervision signals 'explicitly' allows an agent to be meta-cognitive about the quality of supervision. May be relevant if the supervision is social rather than external --- e.g. you should learn to ignore advice from an idiot.
- Something something pathfinding? Can we do something like a combination of local supervision on one channel in the form of links you shouldn't touch, and global supervision on another channel in the form of where you should be going?
- Imitation learning with un-trustworthy sources?

Challenges

Learning should be invariant to transforms of input, output. How to merge this with e.g. attention Badger? Idea: Can Badger provide deltas to train input matrix, output matrix with gradient descent in a pseudo-Hebbian way? Should have the same symmetry.

RL inner loops

Main idea here is to learn better RL (learned exploration, learned integration to skills, etc), possibly in a supervised or RL manner. Common thread is that Badger gets an error signal and has to use that error signal to learn the task within lifetime.

So this focuses on kind of trial-and-error learning, not acting on provided information or making inferences about the nature of the task.

Tasks so far look like optimization problems of various forms (allow SL to be used to train the Outer Loop, but give an RL-like inner loop). What other options are there that we could use?

Self-supervised

Idea here is that experts supervise each-other, inner loop policy might even involve an explicit optimization algorithm like evolution or gradient descent, but using these self-provided losses. Relates to things like David Ha's world models, where the policy is optimized in a 'dream' using a world model that predicts rewards. Except here the reward structure is meta-learned - doesn't have to relate at all directly to the outer loop reward/loss function.

Relates to intrinsic motivation, opportunistic AI, 'economy of experts'/'competition between experts', etc.

Initial experiments on Omniglot suggest that after one set of proper gradients, self-supervision can take over and continue to improve performance. True for other tasks too?

One benefit of self-supervised, especially with also having an explicit optimizer in the policy, would be that letting Badger sit on its own for a while could produce improvements (e.g. as the optimizer better satisfies the internal loss/reward function) - so that might help get at the inner loop/outer loop bottleneck issue.

Discussion Notes

Computer Science analogy for outer/inner loops

We want outer loop to create a language, virtual machine and algorithms library that is exploited within the inner loop - a concrete program is learnt to find a solution of an arbitrary task - not a single task, but any task.

The information that passes from tasks through inner-loop to outer-loop helps to design the language/VM/libraries to fit the problem. E.g. a SGD can be learnt as a static module to be included in the library. Analogy: Matlab is more suitable for scientific computation, C++ for writing HW drivers.

The topology of experts and the communication strategy is forming the language and the VM, the algorithms are more like hard-wired adder on the CPU than a software library.

Further notes

- By asking the outer loop to 'construct' an agent/brain, rather than letting outer loop parameters directly make contact with inner-loop task information flows, we might get more generality (along the lines of MetaGenRL)
- Locality of inner loop steps - what if we limit the number of inner loop steps per expert but compose longer inner loops from shorter inner loops by different experts?

    - This is somewhat resembling the transfer of skills approach, where experts transfer a skill to another expert and that expert can continue in processing the task. In effect this whole topic could be called scalability in the inner loop (life of an agent)
- Possibility of interleaving the inner/outer loops with one another
- What are we really learning in the inner loop?
- A link between the inner loop and Bayesian Inference.
    - The fact that through meta-learning we are learning a suitable prior from which inference is then performed on the test data.
    - But this is heavily task specific as the prior learns task specific info.
    - If we move the same concept to the learning of learning algorithms, then are we learning a prior for the appropriate learning algorithm? Can we get some insight by posing Badger learning as Bayesian Inference?
- MetaGenRL - a bottleneck that allows for generalization. What kind of bottlenecks, besides the homogeneous policy do we have and want in Badger. Is there something else that can be constrained in order to aid generalization? Comms protocol? Hidden states? Others?
- We should have a broad idea of the mechanisms of inner loop learning and what this takes from an architecture point of view. Currently the inner loop is more like inference, as the hidden states are reset between rollouts in the guessing game. There's a fine line between 'learning how to solve an instance' and just performing the task trained for in the outer loop. We are likely doing the latter right now.
    - We specialise though our experiences, is just a hidden state enough to store these experiences?
- We want a general purpose learner, but meta-learning (in say RL^2) assumes that a distribution of tasks has some mutual information, and that mutual information is borne out of training on samples from that distribution.
    - How effective is this? Can we target the MI more directly, by e.g. devising tasks that have the agent directly learn the strategies?
        - Bad analogy: teaching the agents lots of math problems so that deduces the relationships on the number line - why not directly teach it the number line?
        - So instead of having the agent be able to solve tasks from the concrete distribution with common MI, we can apply the learned MI to task from any distribution which have that MI.
        - Pitfalls: The resultant solution might not be as general as a self discovered solution (how would we see this?) and there may be no freedom to explore and improve on solutions for MI (introspection into your own actions and consequences)
    - We want the experts to train on a task in the outer loop, but not actually fit to that task. Rather we want them to fit to the method for solving the task.

References mentioned during discussion

- Jaques et al., Social Influence as Intrinsic Motivation for Multi-Agent Deep Reinforcement Learning, ICML 2019
- Ha and Schmidhuber, "Recurrent World Models Facilitate Policy Evolution", 2018.