ALBA on GitHub

Paul Christiano Follow
Oct 19, 2016 · 5 min read

I've put up an implementation of ALBA on GitHub here.

The key method is ALBA(H, n), which hopefully implements an aligned AI. To actually use it, you would need to replace the stub TransparentHybrid with a competent algorithm for transparent semi-supervised imitation+RL.

For now you can mess around with a version of ALBA that uses a very simple learning algorithm—whenever it encounter a novel situation, ask the overseer what to do and memorize their answer for future reference.

I don't think that actually running the example is especially informative. I wrote the code in order to make the discussion of ALBA more precise, honest, and concrete, not because I thought that anyone would want to actually run it.

TODOs

In reality this implementation of ALBA(H, n) definitely *wouldn't* be an aligned AI; there are at least half a dozen obviously fatal problems. An immediate goal is to address the obvious problems: to write some code such that *if* we filled in the AI capabilities, then there would at least be *any hope* that the resulting system was aligned. I don't know how hard this goal is; I think there is a plausible roadmap and it might be relatively easy, or it might take a very long time.

(Realistically, even if there was *any* hope that the system would work, we would still need to improve all of the ingredients further before there was *much* hope that it would work.)

If that works out then we can start to search for non-obvious problems, and argue about more philosophical questions like whether Meta(HCH(·)) correctly implements capability amplification.

Here is the list of TODOs and FIXMEs in alba.py:

- TODO: Build powerful AI

- TODO: Implement semi-supervised RL
- TODO: Implement transparent RL
- FIXME: Prevent catastrophic failure on adversarial inputs. Adversarial training?
- FIXME: Ensure RL agent cannot outsmart overseer. Gradually scale up capacity as a function of n?
- FIXME: Prevent failure probability from growing with each iteration. Amplify reliability as well as capability?
- FIXME: Allow Amplify(A) to learn from training data, so it can keep up with the RL agent it is overseeing
- FIXME: Scores in [0, 1] are arbitrary, use comparisons between different actions instead
- FIXME: Use budgeted HCH so that errors can't result in hangs
- TODO: Figure out whether iterating A → Meta(HCH(A)) can really get us to arbitrarily powerful agents

How would this be used?

ALBA is designed to be an essentially drop-in replacement for imitation learning or RL. Of course you would only need to use it when you didn't have access to a suitable objective function.

A realistic AI system may use RL or imitation learning as components but will generally have other stuff going on as well. That's fine, you drop ALBA in for the learning part. For example, in AlphaGo you would drop in ALBA for the policy and value networks, but you'd still have to do MCTS.

Hopefully none of your other code is going to destroy the world, but that's not our responsibility.

ALBA(H, n) returns an agent A that is aligned with H but much smarter. You can use this agent in the same way that you'd use an episodic RL agent, except that you don't have to give it a reward. To start an episode you make a fresh copy of A; for example, each time a user visits google.com you might make a new copy of A and then interact with that agent in order to figure out how to process their queries, with calls like A.act("the user just typed `where should I eat dinner?`, what should happen now?"). In principle you could have

very long-running episodes (as with RL), but this would make training more complex.

Instead of specifying a reward directly we specify some overseer H. The system is designed to do what H wants, so picking H is very important. For example, if ALBA were being used in the UI for google.com, H might be implemented by professional trainers employed by Google. If Google wants to make a product that is useful to the users, then it tells the trainers that they should try to think about what the user wants and make decisions appropriately. But the trainers can also balance the user's interests with other desiderata, like Google's profit motive or legal constraints. (All of this is analogous to how Google generally balances the desire to be useful to the user against other desiderata.)

Before the first time that ALBA is invoked, it will perform some pre-training to ensure that its behavior is non-catastrophic, including meeting some minimal bar for quality. [This doesn't happen in the current implementation; it's the FIXME about avoiding catastrophes.] During operation it makes decisions immediately, but will continue collecting data and training asynchronously in the background. [Training occurs inside the implementation of the AI capabilities in capabilities.py.] During training, H will be invoked constantly.

If H is overseeing a single instance of ALBA that is performing a narrow task, then we can just explain the definition of the task to H. If H is overseeing a range of different agents pursuing a variety of tasks, then we can encode the task definition in the agent's observations. For example, if we want it to caption an image, rather than simply giving the agent the image and expecting a caption, we could also give the agent the natural language text "please describe what you see in this image," and perhaps a more detailed description of what kind of explanation we want, and so on. This information will ultimately be used by an aligned overseer to decide how to implement a reward function. As a result, the agent will learn to respond differently to different task explanations. (If the agent is very weak, this might be done simply by memorizing a few different possible task descriptions—we don't inherently require that the agent can learn to understand language, though the whole system becomes more flexible at that point.)

If we were using an RL agent we would implement a reward function in order to cause the agent to do what we want. We don't provide a

reward function for ALBA, but we can let the agent observe "rewards" to help it learn faster. This can be used as helpful unlabelled data, or it can be absorbed by the overseer to improve its understanding of the task. [This doesn't happen in the current implementation; the FIXME about having the overseer learn from training data.]

Conclusion

I think it's useful to be able to point to a completely concrete specification of ALBA. I also think that a concrete implementation is a nice setting in which to start fixing the obviously fatal problems, and I'll be especially excited if/when there is a concrete implementation that *isn't-obviously-doomed-to-fail*. For now, I hope that having access to the code will help people dig into any details of the scheme that they found ambiguous or unclear in the informal descriptions.