ALBA: An explicit proposal for aligned AI

Paul Christiano Follow
Feb 23, 2016 · 18 min read

In this post I propose an explicit procedure for aligning powerful learning systems with their users' interests. The goal is to introduce minimal overhead or additional complexity, yet to safely scale up to extremely powerful systems.

ALBA (algorithm learning by bootstrapped approval-maximization) is a method for providing rewards to reinforcement learners. It uses RL to train a reward function, rather than directly defining a simple reward function.

ALBA alternates two steps:

1. **Amplification**. We start with a fast and aligned agent. We allow this fast agent to think for a long time, and give it access to a large external memory and powerful computational aids. The resulting system is more powerful and much slower than the original agent.
2. **Distillation**. We use this strong, slow agent to define a reward function, and use this reward function to train another fast agent. (This is conceptually similar to knowledge distillation.)

Hopefully, each iteration increases the strength of the learner while maintaining alignment.

We start off the process with a weak learner obtained by applying step [2] with a human in place of the strong learner.

The rest of this post:

- Describes the necessary building blocks and offers candidate implementations.
- Makes this iterative process precise.
- Explains why we might expect ALBA to be aligned and efficient.

- Identifies key problems with this scheme and open questions about it.

(This research was supported as part of the Future of Life Institute FLI-RFP-AI1 program, grant #2015–143898.)

The problem

ALBA is designed to cope with the following problem.

Existing ML techniques are much better at optimizing objectives that we can measure. For example, if we want to use gradient descent to find a good policy, we need to be able to measure how good a policy is.

In general, we can't measure what we really care about directly. Instead, we use rough proxies to assess how well a learned policy gets us what we really want. For example, we may search for a traffic-routing algorithm that minimizes total time spent on the road, or we may search for a user interface that maximizes reported user satisfaction.

Because these proxies don't capture everything we care about, we aren't necessarily happy if they are creatively or ruthlessly maximized. Defining an appropriate objective becomes an arms race between the objective-setter and the objective-optimizer. Today the objective-setters are humans and the objective-optimizers are weak AI systems, so we don't have serious trouble. But that situation may change as AI improves.

ALBA addresses this problem by learning a sequence of increasingly complex objective functions, rather than working with a fixed objective specified by the user.

The building blocks

ALBA relies on two building blocks:

- A "semi-supervised" reinforcement learning algorithm (which does all of the heavy lifting).
- A bootstrapping scheme for turning a weak agent into a strong agent.

Reinforcement learning

We will apply reinforcement learners to a single-step problem:

- Given $x \in \{0, 1\}^n$
- Output $y \in \{0, 1\}^n$
- Receive a reward $r \in [0, 1]$
- Observe $z \in \{0, 1\}^n$

The goal of the learner is to maximize its total score. Our agent should treat different decisions as completely independent, ignoring the non-information effects of each decision on future episodes (as in the usual formulation of bandit problems). This makes the learning problem significantly easier, and it is also important for alignment. The independence assumption can be formalized by comparing two different learners in the context of a fixed sequence of episodes (even if in fact the learner's behavior in each episode influences the content of future episodes).

For concreteness, we can imagine training a deep neural network $f(x) = y$ by gradient descent, and learning a stochastic policy so that we can estimate the gradient of the expected reward as $\mathbb{E}[r(a)\partial p(a)/p(a)]$. (Realistically we will need to do a lot of work to cope with the challenging setup.)

We add one complication to the traditional reinforcement learning problem. Rather than assuming that rewards are always available, we mostly work with "unlabelled" data: our algorithm sees many inputs x, but rarely sees the actual reward. We allow the algorithm to request to see the reward, but only on a small fraction of instances. When it doesn't see the real reward, it can still observe the consequences of choosing a particular y, and use these consequences as side information to help build a model that can accelerate learning.

For concreteness sake, we can assume that we are using a traditional reinforcement learning algorithm which ignores the unlabelled data, requests reward information for a large number of initial examples, and thereafter requests reward information with probability ε. We can accelerate learning with more clever active and semi-supervised learning. Hopefully we can accelerate it a lot.

Finally, we assume that our reinforcement learner is "throttled"—we can control the capability of the system by some parameter c such as model capacity, training time, or computational resources. When c is small the system should be much weaker than its users, when c is

large the system should be state-of-the-art, and performance should scale up continuously with c.

We could also apply ALBA in this toy model of the AI control problem, with a human playing the role of the reinforcement learner.

Bootstrapping protocol

Our other key ingredient is a bootstrapping protocol. This protocol takes as input an agent A which is aligned, and produces a slightly more powerful agent B which remains aligned. We think of our agents as simple functions from $\{0,1\}^n \to \{0,1\}^n$. Critically, the agent B may take *much* longer to run than the agent A.

Intuitively, we'd like to just "give A more time to think." We need a non-trivial bootstrapping protocol because we do not trust A to automatically generalize from a training environment where it has little time to think, to a test environment where we give it a long time to think. (And we can't train A in cases where it has a long time to think, because we don't have any way to produce a suitable training signal.)

Roughly speaking, our procedure works as follows:

1. We set up a simple computational environment.
2. We provide A with a summary of the current environment, and it proposes an operation to perform next.
3. We execute that operation, updating the computational environment, and return to step 2.
4. Eventually, A decides to return a value.

We describe an explicit computational environment, annotated functional programming, here. B is the computation implemented by annotated functional programming, when A acts as the controller.

We could use a different framework for external memory / computational aids. Our only unusual requirement is that the framework be suitable for approval-directed training. That is, it must be easier to determine the correctness/suitability of a particular action than to actually run a whole computation. In particular, the correctness/suitability of an action cannot depend on the rest of the controller's policy, as it would if we were doing end-to-end training. Annotated functional programming is designed to meet this

requirement while remaining suitably expressive and as simple as possible.

Recurrent models

So far I have described an agent A who makes a single decision, and an agent B who evaluates a single decision. We could extend this idea to sequences of actions, allowing A to be e.g. a recurrent neural network, and having B evaluate a whole sequence of actions.

In this case, A still only cares about the payoffs within a single "episode," which now consists of a sequence of decisions, and ignores effects on future episodes (as in the usual RL setup). But within each episode, A may behave strategically.

From our perspective, the key difference between the external memory and the recurrent controller's internal state is that the controller's state is optimized to achieve high reward in future steps, while operations on the external memory are optimized to receive a high score.

Note that B can maintain only external state between each query posed to it, for fundamental technical reasons (the cost of training would increase exponentially if we try to preserve internal state, or else we would have to rely on transfer learning from short computations to long computations). That is, when evaluating A's first action, B can write some notes to itself. It can look at these notes when evaluating A's second action. But the different questions correspond to separate episodes, with no internal state shared between them.

Using a recurrent model is especially critical in the final distillation step, since training a recurrent model (rather than relying on external memory) might be necessary to achieve state-of-the-art performance.

I'm going to continue to talk about the single-action case. But it is straightforward to translate the discussion in this post to the sequential case.

Defining ALBA

We define two sequences of agents A^n and B^n, as follows. B^0 is a human operator. ALBA's external policy will be defined by the last agent A^N; our hope is that it is both aligned and efficient.

For $0 < n < N$, the agent A^n is a semi-supervised reinforcement learner which takes as input the current visible state of a computational environment, and outputs a computational action. We fix a sequence of constants c^n, and we throttle A^n with the parameter c^n.

The agent B^n takes as input a (state, action, auxiliary data) triple, and outputs a real number in [0, 1]. This number will be used to define the reward function for A^{n+1}.

The agent A^N acts directly in the world, in any context that a typical reinforcement learning system could act in the world, with sparse rewards defined by B^{N-1}. Note for example that A^N may operate arbitrary computational machinery—we can apply it wherever we would have applied a reinforcement learning subsystem.

Distillation [approval-maximization]

Agent A^n is given a state *s*, picks an action *a*, and receives reward B^{n-1}(*s*, *a*, auxiliary data). Recall that this reward is only actually computed when A^n asks for feedback—most of the time we don't actually run B^{n-1}.

As described here, we would like to augment this oversight by allowing B^{n-1} to view the internal state of A^n. I don't have any concrete implementation of this idea.

For now, I'll just assume that the raw computational history (e.g. the activations of each layer) of A^n is given to B^{n-1} as the auxiliary input. More sophisticated algorithms may replace *C* by a more effective approach to transparency.

(One reason this procedure is unsatisfactory is that some training procedures will result in *C* being optimized to make *a* look like a good action, or to obscure information that would allow the overseer to realize that *a* is a bad action. A working proposal will probably have to use an alternative to *C* that is actually optimized for being informative. Nevertheless, including *C* is better than nothing.)

Amplification [algorithm learning]

Agent B^n is implemented by using A^n as a controller for annotated functional programming. To evaluate a triple (*s*, *a*, *C*), we set up an initial computational state, with the goal/question:

- {Evaluate the quality of taking action {a}, after performing computation {C}, in a state summarized as {s}. Return the result as a real in [0, 1]},

where C is a summary of the computation carried out by A^n, and a, s, C are stored as terms in external memory.

We then consult A^n in order to produce an instruction, execute that instruction, and repeat (as described in the section **Bootstrapping protocol**). The output of B^n is the result eventually returned by this process.

The only free parameters are the constants c^n, and the free parameters in the reinforcement learning algorithm and bootstrapping scheme.

Note that this bootstrapping process may involve consulting the user—we can easily extend our computational model to include actions to elicit user input. This is not technically needed for our analysis below. However, the ability to elicit user input significantly increases the meaningfulness of having an "aligned" AI system, for our definition of alignment.

Discussion of alignment

We'll try to establish by induction that our systems are *aligned* with their users' interest. I'll postpone discussion of efficiency to the next section.

When I say that a system is aligned at capacity c, what I mean is that it serves the user's interests as well as an "optimally motivated" system with capacity c and access to the same information. I apologize for the fuzzy concepts.

Bootstrapping Lemma: If A^n is aligned at capacity c^n, then B^n = Bootstrap(A^n) is aligned at capacity significantly above c^{n+1}.

Oversight Lemma: If B^n is aligned at capacity significantly above c^{n+1}, then A^{n+1} = ApprovalMaximization(B^n) is aligned at capacity c^{n+1}.

Given these lemmas, and the starting condition that B^0 = [a human] is aligned at capacity significantly above c^1, we can establish by induction that A^n is aligned at capacity c^n. This is exactly what we want.

Note that we can choose the sequences of constants c^n so that they increase extremely slowly; this makes the bootstrapping lemma more likely, without affecting the oversight lemma.

Bootstrapping lemma

This lemma feels intuitively very solid to me, though I don't have any formal argument.

Intuitively, Bootstrap(A) will behave "like a copy of A that gets to think for longer." For example, if I imagine myself in the position of A, it seems clear that Bootstrap(me) is significantly smarter than I am, to roughly the same degree that thinking longer makes me smarter.

If you don't buy this, we can also make a slightly more conservative (though still informal) argument.

Bootstrap(A) can always just directly appeal to A. So as long as A is aligned at high enough capacity to implement this strategy, Bootstrap(A) will be aligned at the same capacity.

On top of that, Bootstrap(A) has access to many simple strategies that might let it outperform A. For example:

- It can use A to answer a large number of sub-questions and use the answers as features to help make a better decision.
- It can consult an ensemble of many slightly different copies of A, each using different strategies to think about the same problem.
- It can perform other expensive computations, and use them as features to help make a better decision.

In order for Bootstrap(A) to outperform A, we only need one of these techniques to help as much as significantly increasing in A's abilities.

Oversight lemma

The oversight lemma is discussed in my recent post adequate oversight.

The rough argument:

1. If B is aligned at capacity c', then B's estimates for the goodness of actions are the best possible for an agent at capacity c', where "goodness" is defined by the user's interpretation.

2. If A knows that B is aligned at capacity *c'*, and A is throttled by capacity *c* < *c'*, then A doesn't expect to be able to beat B at estimating how good an action is. That is, the best estimate for goodness is equivalent to the best estimate for B's estimate of goodness.

3. So at capacity *c*, optimizing B's estimate of goodness is equivalent to optimizing actual goodness.

4. Under a reflective consistency assumption, taking actions chosen to maximize actual goodness results in aligned behavior.

The key problem with this argument is that A can think thoughts that B hasn't considered. So B is not really strictly better informed than A. And as a result, A may be able to identify predictable "errors" made by B.

The step that fails is the move from [2] to [3]—for any particular action, A will (probably) want to defer to B, but if A *chooses the action for which B's estimate is highest*, this might systematically be an action for which A has noticed some considerations that B has overlooked.

There are two hopes for dealing with this problem:

- If B is much smarter than A, then maybe the resulting behavior is aligned anyway.

- Whatever approach we use for letting B inspect A's state is sufficient to allow B to rediscover everything that A thought of. In the concrete implementation in this post, B is simply allowed to directly read the internal state of A. Given the opacity of current machine learning methods, it isn't clear whether this would help at all.

Overall, I think that this aspect of the oversight lemma is the weakest step of the current analysis, and the most natural topic for future theoretical research.

Discussion of efficiency

I think that the additional overhead of running ALBA can probably be made very small, compared to a direct application of reinforcement learning. My argument is very similar to the one given in AI control on the cheap.

At runtime, the algorithm A^n is just as fast as a direct application of a reinforcement learning system—the only thing that differs is how it was trained.

So the remaining question is: how expensive is it to train this system?

Semi-supervised RL

One key ingredient is what I've been calling semi-supervised RL. That is, we need to be able to learn using mostly cheap signals from the environment, with only very rare direct supervision. A very simple example would be learning to play Atari games from pixel-data, where rewards are only provided in a very small number games. Once the learner has formed some very simple generalizations—and especially that the reward is the same as the score written on the screen—then it can in principle learn almost as efficiently as a traditional reinforcement learner.

If we can solve this semi-supervised problem well, then we may be able to get by with a very small amount of human supervision, with almost all learning occurring just by taking actions and observing their effects. As long as training is dominated by the time required for the agent to learn from observations of nature, rather than understanding "what the user wants," we are good.

Of course, semi-supervised RL is a hard problem. I've recently written about one simple idea for this problem, and we can apply a lot of traditional and ongoing work on active and semi-supervised learning, off-policy learning, and model-based RL. It's a big open question that will probably remain open for a while, but I think it is reasonable to expect significant progress in this area as RL improves.

Moreover, adequate semi-supervised RL seems to be a necessary capability of powerful AI systems, since it plays an important instrumental role. So overall I don't feel bad about having the efficiency of the scheme depend on this capability.

Bootstrapping for efficiency

Humans only need to directly oversee the simplest system A^1, which may be a relatively low-capacity model that is relatively quick to train.

The subsequent systems A^n are then overseen by the bootstrapped predecessors B^{n-1}. This oversight process might involve consulting humans (as part of the computation being run by B^{n-1}), but it could

potentially involve humans increasingly rarely as the learners become more sophisticated. Essentially, whatever protocol we might use to limit the training data required by an aligned agent, that protocol can also be learned by the agents B^{n-1} to minimize their reliance on human input. (For these agents, deciding to elicit input is an action which is itself trained using feedback from weaker agents.)

Of course, the algorithm B^{n-1} is also *significantly* more expensive than the algorithm A^n, and so this kind of bootstrapping can never reduce the cost of supervision too far. The fraction of rounds where we can afford to get supervision is roughly the ratio of [cost of running RL algorithm]/[cost of acquiring supervision], and so this approach would simply increase that fraction from "very small" to "small." It would still need to be combined with the kind of semi-supervised RL described in the previous section.

Major problems and open questions

Transparency and oversight

As discussed in Adequate oversight, this oversight procedure may not work when the overseen agent can find arguments that the overseer might miss.

Ideally, we would overcome this problem by training the overseen system to "show its work." I think this goal is especially plausible because the overseer is significantly more intelligent than the overseen system.

Or we may be able to find a totally different way to deal with the problem.

Or maybe this objection is a show-stopper. My main source of optimism is simply that it feels very tractable to have a strong agent oversee a weak one. If we could reduce the AI control problem to this strong-overseeing-weak subproblem, I would feel quite optimistic, and I wouldn't want to give up on that angle until we had explored many more candidate solutions.

Robust learning and catastrophic failures

Sometimes our learners will fail, for example when encountering an input that is different from anything they have seen so far. Such failures will eventually be integrated as training data and corrected,

but we need to ensure that they don't do serious damage in the meantime.

Solving this problem requires either:

- Building composite systems that are robust to a small number of serious failures by individual components. In this context, such robustness is a lot harder than it may seem. The key problem is that, although there won't be too many difficult context changes, a context change may cause simultaneous failures in many components (even in apparently unrelated systems).
- Designing learning systems that don't fail catastrophically on arbitrary inputs. For example, systems that "know that they don't know" and so can back down gracefully rather than going ahead with a potentially-catastrophic decision.

I've discussed synthesizing training data as one approach to building robust systems. But this is a large research area with many promising approaches. Many of those approaches don't look likely to scale to the particular kinds of failures I care most about, but there is still a lot to try.

Robust hierarchical active learning

ALBA is hierarchical in the following sense: the reward of A^n depends on the behavior of B^{n-1}, which in turn depends on the behavior of A^{n-1} in a whole bunch of subproblems. The reward of each A^{n-1} depends on the behavior of B^{n-2}, which in turn depends on the behavior of A^{n-2} in a whole bunch of subproblems…

Suppose that A^1 makes bad decisions on a very small fraction of possible inputs. This leads B^1 to make a bad decision on a larger, but still small fraction of inputs. In turn, this leads A^2 to make bad decisions on a small fraction of possible inputs. This leads B^2 to make a bad decision on a moderate fraction of inputs…

The result is that even if A^1 only makes bad decisions on a very small fraction of inputs, we can end up with very bad behavior for the overall system.

To avoid this problem, we'd like to make sure that our active learning process calls attention to the inputs that A^1 messes up on, even if those inputs are extremely sparse. To elicit such inputs, we need to identify the inputs on which A^2 messes up because of A^1's anticipated mistake, run B^1 on those inputs, and then query A^1 on the

problematic inputs that come up during that computation. And so on, up the hierarchical structure.

Even in the non-hierarchical case, I think that getting adequate guarantees may be difficult, and will have to rely on novel sources of information not normally used in active learning. The hierarchical case considerably amplifies the problem, exponentially increasing the potential number of errors if we use a naive learning strategy.

Semi-supervised RL

As described in the section on efficiency above, ALBA will only be acceptably efficient if we can do statistically efficient reinforcement learning from unlabelled data. Otherwise the cost of oversight will be prohibitive.

Theoretical understanding

The arguments above rely on a lot of fuzzy concepts, especially "aligned at capacity c" but more generally the entire notions of capacity and alignment. These notions are especially important when reasoning about what kind of oversight is adequate.

I expect intuitive arguments involving these concepts to be meaningful but potentially error-prone.

I suspect that there is a clean theoretical framework in which these arguments could be made formally. Developing such a framework would make it much easier to analyze and construct protocols like ALBA.

Does bootstrapping really help?

I've described one approach to bootstrapping. ALBA requires *any* mechanism for turning a weak agent into a (much slower) strong agent without compromising alignment.

I feel like this can probably be done, but I don't have strong reasons to be confident in my particular proposal. It would be great to understand these dynamics better and perhaps to have stronger candidates.

Malicious inputs

Some inputs may cause an agent to behave especially badly—e.g. imagine a buffer overflow in a perceptual subroutine which can corrupt the rest of the agent's processing.

Within the implicit formalism of this post, the existence of malicious inputs bounds the capacity of an agent, since they correspond to questions which that agent will predictably answer incorrectly.

So the capacity-enhancing requirement of our bootstrapping scheme may be stronger than it at first appears. For example, if the agent B begins its computation by feeding the input to agent A, then a malicious input may corrupt agent A. But the bootstrapping protocol needs to introduce robustness to an increasing range of malicious inputs.

Instead, agent B should process the input without ever passing the whole input to A (or at least by carefully preprocessing the input before giving it to agent A). That is, A only answers questions about "what to do next," only ever looking at small parts of the input or high-level conclusions about the input. I think that this is a better approach in general, but it does call into question the style of "more conservative" argument I made in the section **Bootstrapping lemma** above.

In general, I am not too concerned with the prospect of malicious inputs for now— I think that a scheme that works well in general will probably also handle malicious inputs, and so I am happy to focus on other issues. I think that malicious inputs are especially likely to be resolved by an adequate approach to **Transparency and oversight** above. But I do think malicious inputs illustrate some of the subtleties that have to be handled correctly, and they suggest that there are aspects of the problem we don't yet understand very well.

Conclusion

I've presented ALBA, a concrete proposal for aligning powerful learning systems with their users' interests. It can be applied with a generic RL algorithm rather than requiring significant architectural changes, and introduces minimal overhead at runtime. Given enough progress in RL, it might be possible to implement ALBA with very little increase in training costs.

I don't yet have any strong argument for ALBA's safety, but I think we are much closer than for any other existing scheme. The main missing piece is whether a strong agent can successfully oversee its weaker successor during the distillation step, as discussed in the section **Transparency and oversight.** That looks a lot easier than the usual AI control problem.

Aside from this central issue, we'd need to resolve many other theoretical questions to actually get this to work. Many of these theoretical questions are very closely related to traditional problems in machine learning, such as robustness, active learning, algorithm learning, and learning from unlabelled data.

On the experimental side, the algorithm learning problem in ALBA may be prohibitively difficult. That said, I think that experimenting with ALBA may be worthwhile in the same spirit as other (similarly difficult) work on algorithm learning. Bootstrapping may be useful for facilitating training, even if we don't care about scalability at all. And experimental work would shed light on the long-term feasibility of bootstrapping.