Optimization and goals

Paul Christiano Follow
Dec 28, 2014 · 7 min read

If we want to write a program that *doesn't* pursue a goal, we can have two kinds of trouble:

1. We might need to explicitly introduce goal-directed behavior into our program, because it's the easiest way to do what we want to do.
2. We might try to write a program that doesn't pursue a goal, and fail.

Issue [2] sounds pretty strange—it's not the kind of bug most software has. But when you are programming with gradient descent, strange things can happen.

In this post I illustrate issue [2] by considering a possible design for an approval-directed system, intended to take individual actions the user would approve of without considering their long-term consequences.

The architecture

Our agent takes as input a sequence of observations $x[t] \in \{0, 1\}$ and outputs a sequence of actions $a[t] \in \{0, 1\}$.

The main ingredient is a pair of functions $A : \Re^n \to \{0, 1\}$, $S: \Re^n \times \{0, 1\} \to \Re^n$. To run the agent:

- We initialize $s[0] = 0^n$.
- Define $a[t] = A(s[t])$
- Define $s[t+1] = S(s[t], x[t])$

For concreteness you might imagine that A and S are feed-forward neural networks, so that the entire system is a typical recurrent neural network. I'm going to imagine performance well beyond anything we can currently get out of a recurrent neural network.

Our training data consists of some sequence of observations $x[t]$ and reward functions $r[t] : \{0, 1\} \to \Re$. We train A and S to maximize $\Sigma\ r[t](a[t])$. We'll define $r[t](a)$ to be how much we approve of taking a in the context of step t.

To gather training data, we use the current version of A and S to control a robot, we define $x[t]$ as the robot's sensor reading at step t, and we manually provide the reward functions $r[t]$ as look-up tables.

We hope that if we use A and S to control a robot, the resulting system chooses each action to maximize our approval *of that action*. (This might seem like a crazy design, but for now let's just ask whether the resulting system will exhibit goal-directed behavior.)

The state s is important for most applications. We'd like our agent to make decisions based on everything it knows, not based only on what it currently observes. By optimizing S to store useful information, we allow the system to remember information across time steps and aggregate many observations to make informed judgments.

What happens?

This system looks like it shouldn't form any long-term plans; each action should be selected greedily, without concern for its consequences. The separation of A and S makes this perfectly explicit: A is essentially a classifier, which labels each state s with the most-approved action $a \in \{0, 1\}$. There isn't any room for A to consider long-term consequences.

How would this system work, if it worked well? What would the learned function S look like?

Here are some possible properties, which seem like the "right" way to optimize the reward function. That is, if you optimized well enough, over a large enough space of functions, I would expect you to observe these behaviors:

1. The state s encodes beliefs about the agent's environment. (Assuming that our approval, and hence the reward function r, is mostly determined by the environment.)

2. The function S updates the agent's current beliefs based on the observation x.

3. s selectively remembers facts that are likely to be useful, because it doesn't have enough space to remember everything in detail. It

also stores the results of cached computations—the function A doesn't have time to do much computation, so if complicated computations are useful, the agent must carry them out over many steps and remember the results. (For example, if the agent saw a puzzle written on a sheet of paper, the agent might compute and store its answer to use later.)

4. S uses non-trivial rules to decide what to remember or compute. All of these decisions are at least implicitly optimized to make efficient use of limited resources. These choices depend on what situations will arise in the future, and so S should use its full predictive ability to make them. For example, when deciding whether to forget X or forget Y, S might predict what reward it will receive in each case and choose the better option (this computation could use cached intermediate results and predictions, implicitly taking place over many steps). **In general, S updates *s* rationally to maximize its expected future reward.**

5. The agent is able to reason about the relationship between its own beliefs and the world. For example, it might realize that a particular fact will be repeated later, and so not bother remembering it. This clearly requires making a prediction about what it will observe in the future. But it's also based on a prediction about how S will update *s* in response to that observation, and about how A will respond to different values of *s*.

These all sound like good things. Each of them leads the agent to perform better when deployed. We could imagine accomplishing the same goals with simple heuristics, but as we get enough training data the heuristics should converge to rational behavior.

But these properties, and especially property 4, may have some counterintuitive consequences.

We can easily see the shadow of plans: when the agent chooses the state *s*, it won't greedily maximize the next round's performance (and we wouldn't want it to, since this would mean promptly forgetting everything that isn't immediately relevant). Instead, it will choose a state *s* that is useful for long-term performance. And to do that, it will think about long-term consequences.

What goes wrong?

When the agent anticipates the consequences of choosing different values of *s[t]*, it sees that different choices lead to different sequences of states, hence different sequences of predictions, hence different rewards. During training the agent *doesn't* expect different states to lead to different events in the world, because it was trained on the fixed actions taken by previous versions of A,S.

But when the system is actually deployed, changing *s* will also change what action is actually taken. If the system is a good general purpose learner, it will notice this relationship (along with many other patterns that happen to appear while the system is deployed but didn't appear in any training episode.)

For example, if the agent forgets where an item is and is unable to find it when asked, the agent is likely to be reminded where to look. In general, the agent should notice patterns like "If I forget X, I am more likely to be reminded of X in the future." Once the agent notices this pattern, the agent should act on it—after all, the agent is already acting on many similar patterns.

If the agent rationally chooses *s* to maximize future rewards, it might take more extreme measures. For example, it might reason "If I choose a state *s* which incorrectly `remembers' that the user asked me to write down X, then I will write down X. If I write down X, I will remember X in the future. This will lead to low reward in the next few rounds, since the user *didn't* ask me to write down X. But this is more than offset by the high future reward from remembering X on many different occasions."

In an even more extreme case, the agent might reason "If I choose a state *s* which incorrectly `remembers' that the user asked me to implement another AI whose goal is maximizing my future reward, I will achieve a high future reward. I will get an approval of 0 while I am implementing the new AI, but this will be more than offset by my high reward thereafter."

This example is of course facetious, but hopefully it illustrates the point. If we actually want the robot to maximize its total approval, then we should be happy with this result. But now we are right back in the goal-directed case.

Upshot

If you trained a sufficiently weak S, it would not display goals nor exhibit human-level reasoning. If you trained a sufficiently powerful

S, it might display goals and would definitely be superhuman. Somewhere in between you get human-level performance, and somewhere in between you probably get goal-directed behavior.

It's not clear whether goal-directed behavior emerges before or after human-level performance. But I am wary of systems that only behave as intended in a sweet spot between being "good enough" and "too good," or which depend on optimistic assumptions. It would be more satisfying to design systems that perform robustly regardless of how smart they are.

I suspect that similar problems will arise frequently if we train very intelligent but goal-free systems.

One response is to give up on designing intelligent systems without goals—to conclude that in order to build systems that work robustly, we should conservatively assume that they have goals. This is probably somewhat premature.

Another response is to find a better training criterion, which doesn't involve maximizing a sum of future rewards. But it's not clear how to do this while also training the system to build a useful representation, since the usefulness of a representation is determined by long-term performance. We could train F to create a state *s* which is useful on the very next time step (i.e. we could propagate gradients from s[*n+1*] to the parameters of S, but not back to *s[n])*, and hope that the resulting update rule happens to be appropriate over longer time scales. But this seems to rest on a pretty optimistic assumption.

A third response is to also update the state *s* in the most-approved-of way. But this forces the human engineers to figure out how the system should store its knowledge, which may be a serious problem (though it might be manageable).

A final possible response is to "cross that bridge when we come to it." I would like to understand AI safety further in advance, and understanding the inevitability of goal-directed behavior is a natural step. So although this option is tempting, I am hesitant to accept it. Our theoretical understanding is also so weak that I feel we can go somewhat further before we run into a hard requirement for empirical feedback. I'll find this option more tempting once we are going in circles rather than making steady headway.