Advisor games

Paul Christiano Follow

May 23, 2015 · 9 min read

Machine learning algorithms often learn models or policies that are inscrutable to humans. We believe that these systems work well because we have empirically validated them, but beyond that we may have little insight into why they work or what exactly they are doing.

This lack of understanding is an important part of most AI risk scenarios. If human users can understand why an AI system is making a decision, and can evaluate the underlying reasoning themselves, then it seems much harder for things to go terribly wrong. In addition to the intuitive appeal, I've found this kind of understanding to be a useful ingredient in concrete proposals for AI control—even in domains where it is impractical for a human to actually review an AI's individual decisions.

So we might ask: can we make the reasoning of machine learning systems "understandable," without making those systems much less powerful?

In this post I'll propose a precise version of this intuitive challenge. Unfortunately, the precise version of my proposal has a serious problem, and the proposed patches are much less precise.

One concrete instance of the proposed definition is: an algorithm for computer vision is "understandable" if it can convince a blind person that its output is correct. This post will make this intuitive definition more precise.

What does "understandable" mean?

Here is one interpretation: if you understand why a decision is good, then you can distinguish it from one which is bad. Moreover, you can distinguish good and bad decisions by looking at explanations of the decisions, rather than by trying each of them out.

I think this captures an important aspect of understandability. Moreover, I think it is the aspect that is most important for AI control.

“Explanations” need not be static. An explanation may be an interactive process, between a machine learning system and a user who wants to understand its decisions. It may also be an adversarial process, involving auxiliary machine learning systems trained to help find flaws in a given explanation. I expect that static explanations will usually be inadequate in realistic domains.

Operationalizing understanding: Advisor games

Consider an arbitrary machine learning task, for example playing a game or diagnosing a disease. How do we know if we’ve accomplished this task in an “understandable” way?

Following the last section, we introduce an associated *advisor game*, played by two agents: a “friendly” advisor and an “adversarial” advisor. In each round of the game, each advisor proposes a decision in the underlying task. A human judge can interact with the two advisors, and then must decide which of their decisions to accept. At the start of each round, the human doesn’t know which advisor is which.

The friendly advisor wants the human to score a high a payoff on the underlying task, while the adversarial advisor wants the human to score a low payoff. A strategy for an advisor game consists of an algorithm for the friendly advisor, and a protocol for the human judge. A successful strategy should allow the human to get a good payoff, regardless of how the adversarial advisor behaves.

(This formulation is only relevant in domains where the advisors are trying to achieve superhuman performance. This issue is discussed in the section *Human-level performance* below.)

For example, consider a medical diagnosis problem, where we are given a patient’s symptoms and diagnostic test results, and want to output a diagnosis. In the associated advisor game, a human interacts with two machine learning systems, each trying to convince the human of a different diagnosis. These systems might present justifications for their diagnoses, they might cite the historical examples on which their diagnoses are based, they might identify problems with each other’s diagnoses, and so on.

A defect

Advisor games have a glaring flaw—there is a simple strategy that works well but completely fails to address the spirit of the problem:

- Pick a correct algorithm A.
- Have the friendly advisor use algorithm A.
- If the two advisors disagree, have the judge figure out which of them used algorithm A, and trust that one. This can be done with only logarithmically many rounds of interaction and without any advanced technology—if the two advisors disagree about what algorithm A outputs, then they can find a subproblem on which they disagree, and they can continue subdividing until they disagree about an atomic step, using Merkle trees to commit to intermediate memory states.

As long as algorithm A works, there is nothing that the adversarial advisor can do to thwart this strategy.

The problem with this solution is that it doesn't require trusting *the advisor*, but it does require trusting *the algorithm*. The entire point of our procedure was to avoid trusting algorithms, not to avoid trusting implementations of those algorithms.

Advisor games are only potentially useful if we can patch them to rule out this kind of solution. I can think of a few possibilities, though none of them seem very satisfactory:

- Don't allow the advisors to record too much data from their training, and/or don't give them enough time to rerun significant parts of the training. This could prevent them from proving that they correctly followed a particular protocol for training, forcing them to provide a more direct justification for their judgment.
- Don't let the judge use a specialized algorithm, or have any particular knowledge about the algorithm being used by the friendly advisor. Perhaps the judge can receive a short briefing about how to interpret the explanations that the advisors will provide. But assume that the judge can't distinguish a "trustworthy" algorithm from an "untrustworthy" algorithm, and doesn't know anything about the empirical performance of different algorithms.
- Force the judge to use a simple procedure for judging and/or a procedure which could generalize to many different underlying

machine learning algorithms and many different tasks.

- Don't change the rules of the game, but require each solution to "be a good sport:" determine by inspection whether a proposed solution amounts to trusting the underlying algorithm, or whether it would continue to work even in contexts where the algorithm broke down.

All of these modifications seem plausible, and I don't know if it's worth trying to pick one until there are algorithms that could plausibly satisfy any of them. For now, advisor games are only compelling to the extent that we expect there to be some "fair" approaches to such games, and for it to be possible to eventually distinguish these from "unfair" solutions in a satisfactory way.

Elaborations and clarifications

Human-level performance

If a human can perform the underlying task themselves, then they can just ignore the advisors and make the decision directly.

Advisor games are only interesting for tasks that are too challenging for a human to accomplish unaided (similarly, the AI control problem seems to be much more challenging and important in this case).

Much AI research focuses on reproducing rather then exceeding human capabilities, and is not directly suitable for advisor games. But even in these cases, we might be able to construct useful analogies by restricting the abilities of the judge.

In a vision task we could force a judge to close their eyes. In the diagnosis example we could consult a judge a without medical training. And so on.

Of course, we can also work directly with domains where machines exceed human capabilities and avoid this problem.

Arbitration

In most domains, we don't actually want a human to evaluate every decision. In some domains, consulting a human for every decision would be so expensive that we can't even perform a single useful test (for example, imagine a human arbitrating between two decisions about how a robot should move one of its joints during a fraction of a second). In these domains, we can consider a slightly more challenging game.

In an *advisor game with arbitration*, the two advisors interact with an arbiter rather than with a human judge, and the arbiter must make each decision. This arbiter should be *very* simple and fast—too simple to do the underlying task itself, or to distinguish good from bad answers. Instead, the arbiter gives the two advisors an opportunity to resolve their disagreement by "talking it out." If they can't, then the human judge will be involved. This process is then repeated, and the identities of the two advisors are remembered between rounds (so if one advisor consistently gives bad advice, then the arbiter can learn to ignore them most of the time).

In this setting, a solution consists not only of a strategy for the human and the friendly advisor, but also for the arbiter. A good solution must not only get a high payoff, but must ensure that the human doesn't have to be involved too many times.

Where do you get an adversarial advisor?

A good solution should work for any adversarial advisor that the research community can come up with.

In order to validate a proposed solution, adversarial investigators can search for an adversarial advisor specially designed to challenge the proposed solution. If they fail to find any, we can provisionally conclude that the proposed solution is adequate.

Unfortunately, this process only really works if the adversarial advisor is able to integrate all of the information that was used to build the friendly advisor. For example, a research team capable of programming a more eloquent explainer could publish a "solution" in which the human simply believes whichever advisor is more eloquent. This solution wouldn't be satisfactory unless there was some fundamental reason that the right answer had a more eloquent explanation—and to test this, other researchers would need to have access to a similar level of expertise in building eloquent explainers. (Or else they could simply reject solutions which seem to rely on proprietary information as a "secret handshake.")

What can advisors do?

The advisors should be able to output text to the human judge or write files for the judge to interact with, at the judge's request. They may recommend that the judge look at particular information, especially pieces of the input data or of the training data. They may recommend that the judge run useful computations (e.g. to extract summary statistics from training data).

The advisors can also communicate with each other, potentially with much higher bandwidth. The judge is free to inspect parts of the resulting transcript, and the advisors can call the judge's attention to certain parts of the exchange. The advisors are free to generate cryptographic credentials or use any other tools that might be helpful for preventing cheating strategies by an adversarial advisor.

In some contexts, we may want the advisors to communicate in other ways, such as by displaying an image. To that end, we can simply let the judge use programs that can parse the advisors' messages and display them as images (or in whatever other form).

On regularization

For advisor games in a classification setting, one natural strategy is:

- Each advisor commits to a model, and uses that model to make a decision.
- The judge chooses some random validation data.
- The judge trusts whichever advisor's model performs better on the training data.

This strategy is likely to succeed if the distribution of test and validation data is identical and if the advisors don't have access to the validation data. It is much less compelling when the validation data may be from a different distribution (which is the usual setting of interest in AI risk scenarios) or outside of the classification setting.

In this harder case, there are adversarial models which work well on the validation data but deliberately fail on the test data.

A natural way to get around this problem is to use strong regularization. A model which deliberately fails on the test data seems likely to be more complicated. So if we score the expert's models not only by their performance but also by their complexity, then we may prevent this kind of cheating strategy.

This approach may work when the adversarial advisor is perfectly well-matched with the friendly advisor, but it is extremely brittle. If the adversarial advisor is able to find a model which is slightly simpler or which performs slightly better on the validation data, then the approach will break down again. An adequate solution to an advisor game should work for a very broad class of adversaries, rather than assuming that the friendly advisor is at least as capable as the adversarial advisor.

Conclusion

Building machine learning systems with “understandable” behavior would go a long way towards addressing concerns with AI risk. At the moment we don’t have any clear statement of what understandable behavior means, and a naive standard may be unattainable.

Advisor games give one operationalization of understandability. Unfortunately, the simple and precise statement of the game doesn’t really work, and so we would need to make do with a patched variant. Only time will tell whether any simple patch can lead to a reasonable problem.

For now advisor games only seem approachable for very simple domains. If they remain out of reach as machine learning systems become more sophisticated, that would be a (weak) warning sign about the difficulty of AI control. Hopefully it will be possible to make fast enough headway on this problem or other formalizations of “understandable” reasoning that they can catch up with unrestricted machine learning.