Worst-case guarantees

(Revisited)

Paul Christiano Follow

Jan 20 · 13 min read

Even if we are very careful about how we deploy ML, we may reach the point where a small number of correlated failures could quickly become catastrophic. Powerful models could actively undermine safeguards, resist attempts at correction, and manipulate their operators.

I think the long-term safety of ML systems requires being able to rule out this kind of behavior, which I'll call *unacceptable*, even for inputs which are extremely rare on the input distribution.

In this post I'll explain why I think this goal is reasonably likely to be achievable, by highlighting the three ingredients that seem most important to me: adversarial training, transparency, and relaxations.

Preliminaries

This is an updated version of this post. It mostly has the same content, but my thinking now feels clearer and there is a more detailed description of my hopes about relaxations.

Disclaimer

I'm trying to communicate intuitions about whether this problem is likely to be soluble someday, rather than making concrete progress or stake out new ground. I'm a theorist who doesn't work in this area.

Defining acceptable

I'll talk about "acceptable" behavior, but be deliberately vague about what "acceptable" means.

In different contexts, different behavior might be acceptable and it's up to the user of these techniques to decide. For example, a self-driving car trainer might specify: Crashing your car is tragic but acceptable. Deliberately covering up the fact that you crashed is unacceptable.

My key assumptions about "acceptability" are:

- As long as the model *always* behaves acceptably, and achieves a high reward *on average*, we can be happy. The model may still make mistakes, especially when it encounters novel situations, but if we are mindful about how we use AI we can avoid these mistakes escalating into irreversible catastrophes.
- Requiring a model to always behave acceptably wouldn't make a hard problem too much harder. It requires some knowledge about what is acceptable, but the complexity of that knowledge doesn't scale up with the difficulty of the task. Unacceptable behavior is a failure of commission rather than omission, and so avoiding it doesn't require intricate understanding of the domain. Moreover, we are OK if the model conservatively avoids behaviors that are anything close to unacceptable, allowing it to use simpler concepts.

For any given task there may or may not be a notion of acceptability that meets these properties. For a chess-playing AI, it seems pretty likely to be possible—if you find yourself murdering your opponent you've crossed a bright line, and understanding that line doesn't become harder as you get better at chess. For an AI that's *intended* to frequently overrule strenuous objections from humans, there may be no obvious bright lines to prevent catastrophe.

My view is that we can probably get everything we want out of life by applying AI systems to problems where there is a clear notion of acceptability, even in a ruthlessly competitive world where other actors don't care about safety. If you agree with me, then read "acceptable" as referring to whatever concept you think has those key properties. If you don't think there is any concept with these properties, then this post is less likely to be interesting to you.

This is the second guarantee from "Two guarantees," and is basically corrigibility.

I. Basic framework: adversarial training

In addition to training our agent to perform well on natural inputs, we can try to find adversarial inputs on which it behaves unacceptably and then penalize it for that behavior.

Adversarial training requires being able to reliably detect when the agent behaves unacceptably. I'm optimistic that we can achieve this by using debate or amplification to train an overseer to answer the question "is this behavior unacceptable?" Obviously the feasibility of this depends on the details of the task and what we consider unacceptable. I discuss the problem of training an overseer here, and I'll set it aside for the rest of this post.

Adversarial training also requires an adversary who is able to find inputs on which the agent behaves unacceptably. In current practice the adversary is most often gradient descent over the input space, but in the long run I think this technique can't find most possible failures. I'm optimistic it may be possible to train a strong enough adversary, by combining transparency techniques (section II), relaxations that make the adversary's job easier (section III), and amplification.

Even with both of those pieces, getting adversarial training to work well is an important research problem, and there could be other fundamental problems lurking in store. But I would be relatively optimistic that other challenges could be resolved. Overall, I think the most likely fundamental obstruction is the difficulty of implementing a good adversary.

II. Informed adversaries

The challenge

To see how hard the adversary's job could get, consider an agent that works as follows:

- Predict whether the current input could possibly appear during training.
- If the current input definitely couldn't appear during training, then behave unacceptably.

(This is an exotic failure mode, but could arise e.g. as part of a "treacherous turn," a situation where training produces a sophisticated enough consequentialist to understand that it needs to perform well during training in order to avoid a negative reward.)

In this context, the adversary is directly pitted against the agent. If there is *any* predictable difference between the adversary's challenges and the actual situations that appear at test time, then the agent can exploit that difference to fail only at test time.

Looking inside

This game seems almost impossible for the adversary, but we do have one big advantage: we have access to the internals of the agent, so in principle could give the adversary considerable information about what the agent is "thinking."

For example, we can examine internal features computed by the model and observe how those features depend on the input and each other. There is a growing field dedicated to interrogating neural networks to understand what they are doing. (This article describes some relevant techniques and makes an early attempt at combining them to get some insight into what a model is doing.)

On top of this kind of explicit transparency, we can have the adversary and agent share most of their computation, so that the adversary can more easily understand whatever the agent can understand (and can learn to manipulate the internal concepts used by the agent in the same way the agent does). This seems particularly important in combination with amplification, discussed briefly below.

Between these two techniques, I think it's realistic to hope for the adversary to understand "everything the agent understands," and moreover to understand how the agent is disposed towards unacceptable behavior. I don't think this would automatically resolve robustness problems, but I do think it could defuse the adversarial aspect of the problem and give us traction on the remaining problem.

For example, the adversary could start by generating a very broad range of situations in which it would be particularly easy for the agent to behave unacceptably. Once the agent learned to pass these tests, the adversary could try to understand *why* it was passing these tests. If the agent was using a misguided picture of acceptability, the adversary could choose an input that highlights the mistake. If the agent is actively trying to detect whether it's at training time, the adversary can look at the test that the agent uses and try to generate an input specifically to fool this test. (Though this problem may be too hard even if you understand exactly what the agent is doing—see section III.)

Approaches

RL. The adversary's job is to perform the task "find an input on which the agent will behave unacceptably." One approach is to train this adversary directly with an objective that rewards identifying unacceptable behavior.

We can think of such an adversary as an environment, whose observations are the agent's actions and whose actions are the agent's observations. We can then incorporate information about what the agent is thinking by allowing the adversary to interact with transparency tools before needing to interact with the environment.

We can also train this adversary jointly with the agent (being careful about how we take derivatives!), hoping that the intermediate state produced by the agent will be useful for an adversary trying to attack that agent.

Demonstrations. This RL problem is extremely difficult. It would be easier if the adversary could learn from demonstrations of successful attacks rather than having to solve an exploration problem from scratch.

For example, if our transparency tools were powerful enough, then we might hope that a human could reliably use them to find inputs where the model would behave unacceptably. If we only needed a few examples to train the agent to behave acceptably then we wouldn't need an automated adversary at all. If we needed many examples, we could use these human demonstrations to initialize a model that is able to implement an attack.

Amplification. Rather than using a human as an expert adversary, we could train the adversary with iterated amplification. This seems likely to be necessary in cases where the model is so complex that an individual human can't make sense of its behavior.

We can view this process as training a set of more advanced tools that can answer higher-level questions about "what the model is thinking," on top of some library of core building blocks for interpretability. The human constructs more and more powerful tools by using one generation of tools to produce training data for the next generation. The tools at the very highest level can be used to directly answer the question "where is the model likely to behave unacceptably?", and the human provides training data for these very hard question by using previous generations of tools to answer questions like "why did the model decide to take that action?" and "why is the model trying to achieve this goal?"

Relationship to ascription universality

Even with the aid of transparency tools, I'm only optimistic about an adversary being able to find an attack if they are able to understand

everything the model understands. This feels closely related to ascription universality.

Note that ascription universality isn't sufficient on its own: we ultimately care about what the model would have done under different conditions, which is not the kind of thing the model can be naturally said to know. We need some other ideas; in the description above the magic is done by interpretability techniques. The actual interface between interpretability and universality is blurry and I have a very foggy picture of this space.

I don't have any clear picture of what *is* enough; at this point I'm just feeling out intuitions. I'm planning to revisit this question sometime after getting more clarity on ascription universality.

III. Relaxation

The challenge

Unfortunately, no matter how completely the adversary understands the agent and how clever they are, we can't generally expect them to find all points on which the agent would behave unacceptably. The basic problem is that recognizing inputs with a certain property is easier than generating inputs with that property.

As a simple but silly example, suppose our model works as follows:

- Pick a cryptographic puzzle (e.g. "factor RSA-2048").
- When it sees a solution to that puzzle, it behaves badly.
- Otherwise, it behaves well.

Even the adversary understands perfectly what this model is doing, they can't find an input on which it will behave badly unless they can factor RSA-2048. But if deployed in the world, this model will eventually behave badly.

Possible approach

To deal with this problem, we need to penalize the agent for behaving badly on an input even if we can't actually find it. This will require being conservative, and sometimes penalizing our agent even when there isn't actually any input where it would behave badly, since merely telling if a puzzle is soluble can already be computationally infeasible.

(It might sound weird to penalize our agent if it wouldn't actually ever do anything wrong. For intuition, consider the puzzle case: learning that my model *would* behave badly if it saw a factorization of RSA-2048 already tells me that something has gone wrong, even if turns out that the puzzle is insoluble.)

So we can change the adversary's task to make it easier. Rather than asking them to actually produce an input, we are asking them to describe a situation in which the model would behave badly ("the input contains a solution to puzzle P"), even if they can't find a situation with that description or even tell whether one exists.

I'll call this easier problem the "relaxed" problem (as is typical in optimization), and call the hypothetical situations "pseudo-inputs" (inspired by Barak et al).

Illustration: SDP relaxation

This kind of relaxation has been explored in practice in the context of verification. For example, Raghunathan et al use an SDP to certify properties of a neural net (taking small liberties for simplicity):

- Rather than having an attacker specify an input, the attacker specifies some properties of a hypothetical distribution over inputs. In particular, they they specify each expectation of the form $\mathbb{E}[pq]$ where p and q are either input variables or activations in some layer of a neural network.
- We require the attacker's input to satisfy a bunch of simple properties that are always satisfied by real distributions. For example, if an input x is known to be in the interval [-1, 1], then $\mathbb{E}[x^2] \leq 1$. If $z = \text{ReLU}(x)$, then either $x = z$ or $z = 0$, so $\mathbb{E}[z^2] = \mathbb{E}[xz]$. If f is an arbitrary function then $\mathbb{E}[f^2] \geq 0$. These inequalities are all linear (in terms of the numbers $\mathbb{E}[pq]$), and so define a convex set over which it is easy to optimize.
- We read off the probability of bad behavior from $\mathbb{E}$. If we were testing a complex constraint computed by another neural network, then the attacker would also specify the expectations of all of the activations of *that* network, such that the final output is itself just one entry of $\mathbb{E}$.

Because the attacker's constraints don't perfectly characterize the set of achievable moments, it's not easy to tell whether there is any distribution over inputs that has those moments. (In fact it's NP-

hard.) But on the other hand, if the attacker can't find any set of moments, then the model really does always work.

The point of performing the relaxation is that it makes the attacker's job *much* easier. In this case we the relaxation is convex, which means that the attacker can be guaranteed to find an attack if one exists.

I think the relaxations we want to use in the long term won't be so easy to optimize over. For example, the attacker may still be able to specify properties of a hypothetical distribution, but won't be allowed to freely specify quite so many degrees of freedom, and in exchange will have to deal with a more complex set of constraints. But hopefully this can still make the adversary's problem easy enough that an adversary can solve them with the aid of the mechanisms described in section II.

The hope: stronger relaxations

We need to find a relaxation which has two properties:

- It's still "easy enough" for the agent to never behave unacceptably for any pseudo-input.
- It's easy for the adversary to find a pseudo-input on which the agent behaves unacceptably, if any exist.

For example, in the case of a model which fails whenever it encounters a factorization of RSA-2048, an attacker needs to be allowed to supply a pseudo-input like "a distribution over pairs of numbers that multiply together to 2048." Finding that input needs to be easy.

On the flip side, if the adversary has too much flexibility, then they might be able to specify a pseudo-input like "a distribution over inputs that would cause the agent to want to kill everyone." There need to be some limits on the adversary in order for the game to be possible.

This second requirement is tied up with one of the key properties of acceptability: somehow the agent was able to steer clear of unacceptable behaviors. We are now asking them to solve the harder task, of steering clear of unacceptable behavior not only for actual sequences of thoughts, but also for whatever "pseudo-thoughts" they might think in response to a pseudo-input. Intuitively, we need to force the adversary to respect enough of the agent's internal structure

that this "steering clear" procedure can still work, without forcing the agent to totally cripple itself.

Putting it together

I think the following general plan has a significant chance of producing robustly acceptable behavior:

- Specify a relaxed problem for an adversary, where they specify a distribution over inputs by specifying some of its properties rather than by actually producing examples. We don't have any clear sense of how this relaxation could work.
- Train our model with iterated amplification to answer the questions: "which action should we take in state s?", "is action a unacceptable in state s?", and "what's a state where you are particularly likely to recommend an unacceptable action?" The first two questions can potentially be answered directly with amplification, but the last would require some kind of progress in model interpretability (since it's a question about how the agent would behave on other inputs, rather than a question that the agent itself "knows" how to answer).
- Now we can use the model to generate a pseudo-input on which the agent is likely to behave badly, and use the model to verify that the agent in fact behaves badly on that agent (i.e., have the informed adversary described in section II play the relaxed game in section III instead of forcing it to generate an actual output)

I don't think this particular procedure is likely to work. But I do feel like we have a plausible angle of attack for getting sufficiently "easy" worst-case guarantees. I don't think we can see any fundamental obstruction to an approach based on adversarial training, and I feel pretty optimistic about either being able to make significant progress in this area or else identifying a clearer obstruction.

Postscripts

- To answer "is action a unacceptable?" we need to use the model to answer a bunch of sub-questions, like "what are the worst consequences of action a?" If the model sometimes chooses bad actions, it's reasonable to be concerned that it will also sometimes answer these subquestions badly. But we constantly train "is action a unacceptable?" on the pseudo-inputs most expected to yield unacceptable behavior, and so an average-case

guarantee from the question-answering system implies that it typically answers these subquestions well.

- If we trained the adversary using RL to elicit unacceptable behavior, then we could have a chicken-and-egg problem: the overseer fails to recognize some behavior as unacceptable, and so the adversary isn't motivated to elicit it, and so the overseer never learns to correctly recognize it. Demonstrations allow us to overcome this problem, if the demonstrator attempts to generate unacceptable behavior even when the overseer hasn't yet learned to recognize it.
- I find it helpful to think about sections II and III separately, since each of them seems to confront a distinct difficulty. It seems meaningful to think about how we can relax the adversary's problem, and independently to ask how we can give the adversary insight into where the model might behave badly. But at some point I expect them to blur together, and the whole procedure might not end up looking very much like adversarial training.