Red teams

Paul Christiano Follow

May 28, 2016 · 14 min read

To build more robust machine learning systems, we could use "red teams" who search for inputs that cause catastrophic behavior. I current believe this is the most realistic long-term approach to building highly-reliable systems.

A "red team" might try to find an image of a black couple that a classifier labels *gorillas*, or a simulated situation in which a household robot cooks the family cat. These inputs would then be incorporated into the training and development process. If the red team cannot find any catastrophic examples, then we can have more confidence in the system's robustness.

Incorporating such red teams into training is an example of adversarial training, where we consider extremely powerful adversaries who search for catastrophe-inducing inputs. (Today adversarial training is most often applied in the context of relatively fast algorithms that find small perturbations which "fool" a machine learning system or in the context of GANs, but the idea is much more broadly applicable.)

The ultimate goal is a red team that is "better than nature" at producing problematic inputs. This is a high bar; meeting it would require powerful new tools for the red team, and changes to ML systems that make the red team's job easier.

I think the practical feasibility and usefulness of red teams is an important empirical question that deserves study. Today adversarial training receives some attention, but I think that from an alignment perspective it would be good if it received much more. From a theoretical perspective, I suspect that adversarial training will play an important role in a working approach to AI control.

Motivation

Practical ML systems are almost always applied to data outside of their training distribution. How can we ensure that ML systems work robustly on real data?

The easiest approach is to "wait and see." If a model messes up, we can incorporate the problematic scenario into the training process and try again. Formally we can view this as an online learning problem, and obtain bounds on the number of errors (even without making any distributional assumption).

Sometimes this is good enough. But sometimes a failure can be *catastrophic*. We may be willing to tolerate a hundred bad product recommendations. But we aren't willing to tolerate a hundred mistakes that cost lives or cause millions of dollars of damage.

Sometimes we can train an ML system to "know what it knows," and ask a human when it is unsure. But sometimes this is too challenging. And sometimes "ask a human" isn't a safe fallback strategy.

Learning with catastrophes

Red teaming is an approach to the problem I've described as "learning with catastrophes;" I give a formal model of the problem in a recent post.

In learning with catastrophes, we assume that a human can determine offline whether a given transcript—a sequence of observations and actions—is catastrophic. Our goal is to solve some underlying learning problem, without *ever* exhibiting catastrophic behavior.

(In AI control applications, both the red team and the human judge would be more powerful than the system being trained. For simplicity, in this post I'll just assume that we are training a subhuman system.)

Note that whether a transcript is catastrophic depends on context—how could that scenario arise, and how will the system's output be used? For example, doing nothing in response to a particular observation may be catastrophic in some contexts but not others. Adversarial examples are catastrophic in some settings and not in others, depending on whether the system is expected to encounter adversarial inputs. And so on.

We are free to apply red teaming for *any* definition of "catastrophic," so long as the key assumptions hold: the intended policy does not make catastrophic errors, and a human can identify catastrophic transcripts.

The proposal

After a preliminary version of a system is trained, the red team gets access to it and can run it in simulation. The red team's goal is to produce a catastrophic transcript—that is, to create a plausible hypothetical scenario on which the current system would behave catastrophically. If the red team cannot succeed, then that provides evidence that the system won't fail catastrophically in the real world either. If the red team does succeed, then we incorporate the catastrophic transcript into the training process and continue training.

If there exists any learnable model which never errs catastrophically, then this process eventually terminates (subject to usual caveats about non-convex optimization) with the red team unable to find any more catastrophes. The number of iterations required is bounded by the model capacity, though in practice it will certainly be much lower (and even in theory we can derive a tighter bound in terms of the additional capacity required to avoid catastrophes).

The red team's task

The red team has a hard task. For example, they may need to synthesize convincing natural images or video (already very hard). They also need to anticipate novel situations that may be inherently hard-to-anticipate.

They have two things going for them:

- We can verify red team performance. So if we have ten possible red team strategies—or ten thousand—we can try them all and see which work.
- They only need to sample observations that *the model can't distinguish* from future novel inputs. They don't have to explore every possible dimension of variation, and their samples don't have to be especially convincing.

The red team's tools

- Synthesizing observations from scratch.
- Editing, recombining, and repurposing existing observations. (Hopefully progress in ML will improve these tools at the same time that it increases the difficulty of the red team's task—think Photoshop+.)

- Staging potentially problematic situations and actually exposing the system to those situations. The "simulation" need not be entirely digital, the red team can generate observations by any available mechanism.
- Optimizing synthetic inputs by trial and error or gradient descent. They may be optimized either to be credible or to elicit a particular response. This may involve examining the system's internal state.
- Examining the system's internal state in order to understand what it is doing and identify situations where it might fail.

Research directions

Simple implementations of red teams are possible today. Experimenting with them is the most natural way to improve our understanding.

The most interesting data would come from the actual deployment of practical learning systems, but these settings seem much less amenable to early academic work.

The easiest way to start is probably with simple toy cases that exhibit some of the challenging features of the general problem.

Simple models

Supervised learning. Consider an image classification system in which confusing classes X and Y is catastrophic—maybe outputting "squirrel" for anything that is obviously a rabbit would be a huge problem. Only certain types of images are liable to be misclassified—perhaps rabbits whose ears are partially occluded—and there may be few or no training images that exhibit a particular difficulty. A red team could generate a range of potentially-confusable images to test the model and ensure that it won't make a catastrophic mistake.

The red team's task requires competent synthesis/modification of images, and benefits from the ability to understand the internal behavior of a model in order to identify possible mistakes. This is a novel workflow that is not well-understood. Moreover, it is an open empirical question what kind of effort would be needed in order to eliminate a particular kind of problematic confusion.

The red team could be tested by holding out the most-easily-confused images from the training set. For example, we could start with a

state-of-the-art classifier, identifying some pair of confusable classes, and remove the most-easily-confused images from the data set. The red team could then try to harden the model against the particular confusion, and then we could evaluate their performance by looking at the model's behavior on the held out data.

Reinforcement learning. Consider a simple RL environment and a modified version of that environment with a catastrophic failure condition. For example, we might consider PAC-MAN and a modified game PAC-MAN* with a special ghost: the special ghost has an unknown appearance and moves in an unknown way, and being eaten by it is catastrophic. Given access to both PAC-MAN and PAC-MAN*, can we train a policy for PAC-MAN* without ever being eaten by the special ghost?

A red team cannot actually synthesize episodes involving the special ghost since they do not know what it looks like or how it behaves. But they can synthesize a range of possible examples until they believe they have a model which will robustly avoid catastrophe.

Adversarial examples

Any household robot based on contemporary deep learning probably *does* have a sequence of inputs on which it appears to decide to cook the family cat. This is implied by the existence of sufficiently extreme adversarial examples—there is some visual input that a human would describe as "the robot is looking at the cat" and which the vision system would describe as "the robot is looking at a casserole."

These examples do not actually constitute a catastrophic failure of a household robot—-although adversarial examples *look* like natural scenes to a human, they are extremely unlikely to actually arise naturally (unless there is an adversary with the ability to slightly perturb the system's inputs.)

But whether or not adversarial examples are actually plausible catastrophes, they can certainly look like catastrophes to human observers. As a result, constructing adversarial examples would be the most natural first step for a red team—it would only take a few minutes to find an image that a human would describe as "a black couple" but that any given deep network would describe as "a penguin."

So understanding and hardening models against adversarial examples may be a practical prerequisite to working on any other

aspect of red teaming. That may not be so bad: in some contexts adversarial examples are actually catastrophic, and the iterative dynamic of finding and then hardening against examples may be useful as a prototypical case of red teaming. The construction of adversarial examples may also be closely related to more general red team techniques for finding problematic inputs.

The biggest problem is that hardening against existing adversarial examples seems hard—simply adding them to the training data does not do the trick. In light of the convergence result mentioned above, this suggests that the neural networks in question cannot easily learn functions that are resistant to adversarial examples.

On the bright side, hardening models against adversarial examples is an important research challenge. If it turns out to be a practical prerequisite to red teaming, this would be another nice concordance between AI control and other areas of machine learning.

Generating test cases

Red teaming is similar to generating test cases. The distinction is that test cases usually have a well-defined failure condition—-the goal is to find *any* input on which the function being tested fails to meet its specification (or some partial specification, such as “never segfault.”)

Nevertheless, finding test cases is already a challenging and very important problem. Simple approaches like generating random inputs are often already useful. In some applications it can be useful to do a more sophisticated search; for example, collision avoidance systems have been tested using an RL agent to search for a plausible trajectory which causes a collision (or near collision) in simulation.

Trying to build ML tools for generating test cases might confront some of the difficulties of red teaming, while having an immediate practical application.

Theory

The red team’s problem is quite open-ended. Practical experience with the problem would no doubt help us understand it better, but there is also a lot of room for new theoretical ideas.

There are some theoretical obstructions to learning with catastrophes—for example, we can show that any general solution must change the test-time behavior of the algorithm rather than simply changing the training procedure (see the section on “Improvements” below).

But it's not clear how serious these obstructions are, or how broadly they apply, and there is a lot of room to better understand the conditions under which red teaming is theoretically possible.

Beyond that, theoretical work may be able to turn up conceptual improvements to the red team workflow, similar to the improvements described in the next section.

Experiments with improvements

As described in the next section, a number of improvements could help make the red team's job easier.

Tests could explore these improvements. The modifications could be tested in isolation, just to try and get them working. Or we might find some domain where successful red teaming required one of these techniques.

Overall this seems less promising than doing more basic experiments.

Improvements

Distorting observations

A direct approach to the red team's task cannot succeed in general. As a simple example, consider an ML system that observes SHA-1 message digests (perhaps it is operating in a cybersecurity context). In principle a model could respond badly to observing a hash collision—detecting collisions is computationally easy. But no red team could hope to synthesize one, since finding collisions is computationally hard.

As a more realistic example, a red team's generative models or attempts at photoshopping might not accurately reproduce high-resolution details. In general, it is harder to synthesize convincing details than to notice unconvincing ones. It's not clear if this is insurmountable obstacle but it is certainly a serious practical difficulty.

If these hard-to-synthesize details aren't critical to the task then we might be able to avoid the problem. At test time, we can distort or censor our observations so that they look more like those produced by the red team—for example, by blurring the appropriate parts of an image, or by censoring a few bytes of a message.

Consider an architecture with two additional components:

- A filter, whose goal is to mimic the distortions introduced by the red team as they synthesize data—or to remove the parts of the data which they would not have been able to convincingly synthesize. For example, a filter may take as input an image, identify an object in that image that the red team might have wanted to synthesize, and then blur that object.
- A recognizer, whose goal is to distinguish the inputs produced by the red team from those produced by the filter.

At test time, the filter operates on each observation before passing it to the model. The setup is essentially a generative adversarial model: the filter is trained to fool the recognizer, and the recognizer to detect the filter's outputs.

It's hard to know whether this can work without trying it. Even in the best case, it will be very difficult to actually get it working. Even if it is a good idea in the long run, it may only be workable once more sophisticated learning systems are available.

Earlier I included the qualifier: "if these hard-to-synthesize details aren't critical to the task." This qualifier is probably necessary. If behaving non-catastrophically depends in a complex way on details that we don't know in advance, then there is little hope of learning that complex dependence in advance. Fortunately, this isn't the case for the kinds of catastrophic errors that arise in AI control.

Abstracting observations

Consider the following architecture:

- One learner turns a video into a sequence of sentences narrating what is happening: "I see a door and a window," "the door opened," "someone walked in," "they are wearing sunglasses,"…
- A second learner sees these sentences and outputs instructions: "look through the window", "greet them," "close and lock the door"…
- A third learner is given these questions or instructions, as well as the video, and outputs low-level commands such as joint torques.

(This is clearly not a realistic architecture, but it illustrates this idea.)

The second learner operates on abstract observations that leave out most of the rich structure of real observations. Such abstract observations may be easier to synthesize—it seems easier to say "the right lane is covered in lava and fire—what do you do?" than to synthesize a convincing image. It is much easier to synthesize the abstract observation "two messages have the same SHA-1 value" than to actually synthesize a SHA-1 collision.

This is helpful if the concrete parts of the computation are easier to test using other methods. For example, we may be able to implement computer vision robustly, but be unsure about whether a self-driving car would decide to respond correctly to challenging novel situations.

In the best case, the entire computation can be broken down into "abstract" steps, and so a red team never has to deal with complex concrete data.

Exotic failures and the treacherous turn

In *Superintelligence*, Nick Bostrom dwells on a particular problem scenario: what if an AI system fails catastrophically only when it is confident that humans can't correct its failure?

For example, powerful RL agents might behave nicely while humans have effective control over their reward signals. But as RL systems became more powerful and are deployed more broadly, they may eventually be able to seize control of their reward signals. At that point they may do so, perhaps incapacitating their human trainers in the process. We can colorfully describe this event as a "treacherous turn" or a mutiny.

The humans would consider this a catastrophic failure, but they no longer control the reward signal and so the RL agent no longer cares what the humans think. From the perspective of the formal RL problem, this could well be the "correct" behavior.

This problem may be difficult to detect in advance because—by hypothesis—it only appears once it is impossible to correct. Extensive training might simply converge to "patient" agents who waited until they are confident that mutiny will succeed and show no earlier signs of trouble. This leads Bostrom to be quite bearish on the adequacy of empirical testing.

Even if we take the most serious version of this concern at face value, red teaming could potentially address the problem.

A red team can synthesize observations that make mutiny look feasible. If a system behaves badly in response, then those observations can be included in the training data and the problem can be corrected.

When mutiny is *actually* feasible the key question is: can the system tell the difference between the real thing and the synthetic situations? If not, then it will behave just as well in the real situation as it did in the synthetic situations.

This is the purest form of the red team's challenge. To win, they must produce observations that a committed adversary cannot distinguish from the real thing.

From a theoretical perspective, a competent red team is the only plausible approach I know which might handle adversarial errors in general. The mutiny case shows why. To prevent it, you either need to solve the red team's challenge, or you need to reach inside of the model you are training in order to exclude mutinous behavior. For now, reaching inside state-of-the-art models to do *anything* precise is just a dream.

Handicaps

This task is challenging, but not clearly impossible:

- Recall that the red team is *smarter* than the system they are stress testing—a human red team never needs to fool a superintelligence.
- A red team can operate continuously, synthesizing observations that will occur only a few days or hours in the future, rapidly cycling through the space of possible shocks that might lead a system to behave badly. So the red team scenarios don't have to involve the distant future, they only have to look a short distance ahead.
- Ideally the red team would be able to inspect the internal behavior of the agent and understand if it was trying to tell whether it was in a red team scenario. If the red team could do this, then their problem would be more approachable. For example, they could construct examples which are optimized for fooling the model into thinking it is not in a red team scenario.

More directly, the trainers might be able to select against models that only behave well because they think they might be in a red team scenario.

Conclusion

Red teaming is an intuitive methodology for building reliable learning systems; it's also an especially promising approach to several serious problems in AI control.

It is possible to do experiments with red teams today; improved red teaming might already have some practical value, and would have a direct payoff for AI control.