# Optimizing with comparisons

Paul Christiano Follow
Dec 1, 2015 · 5 min read

I could elicit a user's approval of an action $a$ by having them supply a rating $V(a) \in [0, 1]$. But putting myself in the shoes of the rater, administering such a rating *feels really hard*. My intuitive evaluation depends on what our system is capable of, and what the alternative actions were. I often lack an absolute sense of how good an option is in isolation.

It feels much easier to specify comparisons, e.g. via a function $C(a, a') \in [-1, 1]$ that compares two alternatives. When the function is positive the first argument is better (the more positive, the more better); when it is negative the second argument is better.

I've found myself feeling this way about many aspects of AI control proposals. So it seems worth exploring the idea more generally.

I'll assume that $C(a, a') = -C(a', a)$, but this can always be ensured by anti-symmetrizing. It might be easiest for the reviewer if C is ±1 valued, or perhaps {-1, 0, +1}-valued.

## How to optimize?

Comparisons can be used to evaluate proposed perturbations of a model, or to select amongst several candidate models. For most optimization approaches, this is all we need.

For gradient ascent, we can train by taking the partial derivative of $\mathbb{E}[C(a, a')]$ with respect to only the first argument of C, where $a$ and $a'$ are independent samples from the current model.

For evolutionary search you could evaluate the fitness of each individual $x$ by computing $\mathbb{E}[C(x, y)]$ for $y$ sampled from the current population.

## What is being optimized?

When optimizing a real-valued function V, we know where the optimization is going—towards inputs that are scored well under V. But what happens when we optimize for preferences defined by a

comparison C? If C is transitive, then it's clear what the optimum is. But if C isn't transitive then the situation is a bit more subtle.

In fact there is an easy answer. If we consider C as the payoff function of a zero-sum game, then the systems described above will converge to the minimax equilibrium of the game (if they converge). This seems to be a very natural generalization of maximizing a scalar function V, via the correspondence $C(a, a') = V(a) - V(a')$.

This suggests a general recipe for building agents to maximize preferences specified as (potentially intransitive) comparisons—we can apply the same techniques we use to play two player games with large state spaces.

### Is this OK?

If preferences are given as comparisons, then I think the minimax equilibrium is really the only possibly-sensible solution. But that doesn't necessarily mean its sensible.

For most applications I have in mind, I think this equilibrium is perfectly fine even if C is ±1 valued and throws out all information about the strength of comparisons.

The minimax equilibrium is supported entirely on solutions which can't be robustly improved upon. That is, for each action $x$ in the support of the minimax equilibrium, there is no option $y$ which "reliably outperforms" $x$—for every $y \neq x$ there is some plausible option $z$ such that $C(x, z) > C(y, z)$.

I think this is basically enough to make me happy with an AI's behavior, without even exploiting the further guarantee that $\mathbb{E}[C(x, z)] > \mathbb{E}[C(y, z)]$ for a random action $z$ selected by the system.

## Alternatives

If we don't use comparisons, what would we do? I'll talk about the case of assigning approval values to actions, but the discussion seems more general.

### Option 1: spread out over the scale

I could try to give actions scores ranging over the whole scale between 0 and 1, such that every pair of importantly different options have significantly different scores.

This seems quite difficult. It requires my judgments to be quite sensitive to slight variations in the quality of outcomes, since otherwise our system will not have any incentive to make small improvements. But it also requires different judgments to be extremely consistent, which is tension with sensitivity.

Even if we have estimates which are sensitive and consistent in expectation, if the estimates are at all noisy then they will introduce unnecessary noise that we could avoid by using direct comparisons.

Overall, I don't think this is a good option.

### Option 2: reason about internal state

In my unsupervised approval-directed proposal, the evaluating process has all of the time in the world. So it can literally enumerate different possible actions, and determine the capabilities of the agent by intensive inspection. I think this is a fine solution as far as it goes, but it's not going to fly for evaluation processes that actually exist.

### Option 3: compare proposals

In my supervised approval-directed proposals, each action is reviewed by a critic who can make a counterproposal. We can adjust our rating of the action based on the quality of the counterproposal.

The scheme in this post is a simpler and cleaner version of the previous scheme. It is cleanly separated from the other kinds of "argument" that might occur between AI systems and the other kinds of help that the human might get from AI assistants.

In particular, it seems important to notice that the same learning system can play both sides of the game—there is no need to have two different learners, or for one of the players to "move second."

It also seems worth noticing that this is a very simple modification to existing training procedures, and we could already implement it in practice if it seems necessary.

## Applications

### Human feedback

As discussed here, we could learn a function to predict which of two outputs a human would prefer, and then use this predictor to train an actor to produce good outputs. This sounds significantly more

promising than having the human score individual outputs and training a predictor to guess those scores, especially during the early phases of training when all of the outputs would be pretty bad.

(I think this is the most important application.)

## Imitation

Suppose that we are following this approach to imitation learning, in which a classifier learns to distinguish human and machine behavior, while a reinforcement learner tries to fool the classifier.

Rather than training the classifier to predict labels, we can train it to pick which of two trajectories is a human and which is machine. We can then use this comparison function to directly train the reinforcement learner to look more human-like.

I don't know whether this would be a more robust approach. For the SVM in Abbeel and Ng 2004 it wouldn't make any difference. For more complex classifiers it seems plausible that the comparison would work better then using the log probability, especially during early phases of training when the imitator is not doing a very good job.