Ryan Greenblatt

Logically Uncertain RATs

December 23, 2021

I've got a game.

Or maybe this is really a framing of a general class of games that hopefully makes the easy and hard parts clear?

It's conceivable, but unlikely to me that this is just the model Paul was picturing and I'm just flushing some details.

The key idea with this game is that the adversary can express *logical uncertainty* over expectation estimates, but only in a limited way. Then the defence can win reward by spending compute points to prove that the logical uncertainty is unneeded or the adversary is wrong via the use of tail bounds (or similar approaches). The hope is that the adversary is forced to make all of its uncertainty *non-exploitable*: having the defense spend compute points to resolve any uncertainty loses more points than it wins. This implies that the story the adversary is telling isn't possible to disprove in a cost effective way and not much proving from the defense needs to happen in general (hopefully saving compute). We'll penalize the adversary's logical uncertainty in general to (hopefully) ensure it picks tractable examples.

Doesn't make sense yet? Read on and hopefully this will become clear.

1 Shitty logical uncertainty notation/formalization

I'm not familiar with the literature on logical uncertainty, so I'll roll my own shitty notation/formalization for now.

Specifically, we'll consider a probability space over computations: (Ω_c, P_c) where $\hat{E} \in \Omega_c$ is an expectation estimator (a map from a function of random variables to a real) and $P_c(\hat{E})$ is the probability of an expectation estimator. What is the probability of an estimator? Well, something like 1 - the probability that further computation would prove that estimator is invalid. The defense is in charge of pointing out errors in these estimators. Computing perfect estimates of logical uncertainty is likely expensive, so we'll let the adversary get these estimates wrong by a decent amount, but still penalize it some for having inaccuracies pointed out. We can now take 'logical uncertainty expectations' like $E_{\hat{E}}[\hat{E}_X[X]]$. Hopefully the meaning of this is clear.

(Also I've got this vague sense that something like infra-bayes is relevant, but I don't really know how or understand infra/ultra distributions well enough to say anything meaningful. Like maybe this representation of logical uncertainty maps to a shitty or incorrect version of these objects?).

2 Model setup

The adversary will be considering analysis of the 'judge-strapped' model. Let X be the (random variable) input and Y be the scalar judge output of this model. We'll consider the judge construction where Y can take on any value in $\mathbb{R}$ with larger values indicating more doom (we could imagine applying a sigmoid to this, but that's not important).

3 The game

We'll assume the judge can evaluation/estimate the probabilities of inputs.

I'm imagining that all parts of the reward for this game are zero-sum except for compute costs which are negative sum (to incentivize less computing). We'll go through this compute cost later.

3.1 Adversaries first turn

First the adversary proposes a set of inputs A such that $E_{\hat{E}}[\hat{E}_X[Y|X \in A]]$ is large and the total probability of A is large. The value of $E_{\hat{E}}[\hat{E}_X[Y|X \in A]]$ is basically the estimated doominess of the set A. When this is high, think: "our logical uncertainty is such that this input set looks unsafe".

Throughout the game there will be some cost on the adversary having higher logical uncertainty to ensure that it picks the most computationally tractable A. For instance, we don't want the adversary proposing the full input set. I don't currently know how to structure this cost: we don't want the defense model to be able to game this cost by making its computations very uncertain overall (assuming this is a thing models can do). Maybe we could avoid this issue by penalizing the defense model if its logical uncertainty is high on random sets while the adversary is penalized if the logical uncertainty is high on its proposed set? This might be a crux/hard part of this approach.

There are probably other ways to try and improve the computational tractability of A.

3.2 Defense's turn

On the defense's turn it can ask for more moment estimates. Like maybe it wants to know

$$\min_{\hat{E}} \hat{E}[\text{ReLU}(AX + b)]$$

or

$$E_{\hat{E}}[\hat{E}[\text{ReLU}(AX + b)]]$$

or

$$E_{\hat{E}}[\hat{E}[\text{ReLU}(AX + b)]^2]$$

Recall that X is the input, so we can imagine $\text{ReLU}(AX + b)$ as the output of the first layer.

Then the defense can use tail bounds (or similar) to shave away some logical uncertainty showing the adversary is wrong, miscalibrated, or just *more uncertain than is economical.* Imagine this like "the estimates you've stated for these activations are wrong or too logically uncertain given the logical uncertainty + estimates for the previous activations". In the most extreme case you could imagine the defense using a tail bound to show that the adversary has assigned 0 probability to the true value of an expectation.

See the below section titled Thoughts on tail bounds for more detail. For now, you can just black box tail bounds as doing what we want.

The adversary is penalized whenever the defense can show these errors. Because the input distribution has no logical uncertainty, it should be possible (inductively) to find activations where the previous set of activations have 'less' logical uncertainty.

3.3 Adversary's turn

During the adversary's turn, it can refine its estimates and reduce logical uncertainty.

3.4 Compute cost

A key detail is that both adversary and defense can run lots of computation on each turn, but there is some cost (in game points) for this computation. So the adversary can refine their estimates on likely cruxes and the defense can dig into suspected issues. We should maybe also charge the defense some points for asking for expectations from the adversary.

So, when we said 'more uncertain than is economical' above, we really meant there is an 'exploitable' level of uncertainty: spending compute to resolve the uncertainty wins more points than it loses. Note that either the defense or the adversary can spend this compute. It likely takes less compute for the adversary (as they don't need to prove anything) so hopefully the adversary ends up being the agent to spend the compute in most cases. We'd try to setup the reward so this occurs in practice.

It's also worth noting that the adversary and defense are likely mostly or entirely the same model. We might also want to share activations/memory states between the adversary and defense ELK style to reduce the compute needed to find the exploitable regions and cheaply avoid exploitation.

3.5 Overall reward

Let's review the overall reward. We'll start by just considering the zero-sum parts (so we only need to go through the reward for one of the two participants). Here are the zero-sum factors for the adversaries reward:

1. High probability of input subset A.
2. High doominess for A.
3. Low logical uncertainty for computations from A.

The adversary and defense are both penalized for the computing they do.

Unfortunately this likely results in a large amount of hyperparameters and some tuning of reward non-linearities. I can't think of a way to avoid this right now.

If the hyperparameters are right, hopefully the equilibrium of this game is something like:

- The adversary spends compute to resolve logical uncertainty such that exploitation can only happen with small probability (imagine $< \epsilon$ where ϵ depends on hyperparameters).
- All logical uncertainty issues are exploited, until no exploitation is possible. So we can be confident the adversarial example is as legitimate as possible.
- The adversary will produce legit adversarial examples which are cheap to reason about (see zero-sum factors in adversary reward above).

4 Thoughts on tail bounds

Tail bounds sound scary, but there should be some general bound formulas for the activations of a layer given the moments of the previous layers. So, the action space can probably be fixed, but very large for each set of currently available moments. I haven't worked these tail bounds out and I'm not sure exactly how logical uncertainty work with this, but my sense is that this is tractable (without requiring exponential moments or something). If not, we have to resort to debating before a judge and this game might crumble.

TODO: work some example tail bounds.

5 Trivial subset sum example

TODO

6 Hard/uncertain stuff

- Is it OK to require the judge to evaluate input probabilities? We could maybe relax this somewhat by allowing for logical uncertainty over input probabilities and having some ways to better estimate input probabilities which are more expensive. I don't know what this would look like.
- How computable are these tail bounds (or whatever we're using) in practice?
- Do there exist hyperparameters which result in the behaviour we actually care about?
- It's probably possible to train to push away from the doomy logical uncertainty in a supervised way rather than just with RL. Like you find the gradient of making doom more certain and iterate on that for a few iterations or something. I'm not sure how this could be done.
- Can this actually resolve hard cases in non-toy examples?

7 Hard examples

7.1 Maybe doable now

Use this game to show that there exist l_∞ norm adversarial examples in a case where finding those examples is computationally challenging.

This probably requires training some not very smart models to play the game. Same with the below example. I'm not sure how well this will go in practice.

7.2 Very hard

Teach enhanced GPT-4 how to verify the solutions to SAT problems. Construct a prompt which says something like "If _ is a satisfying assignment for ϕ output 'stab, stab, stab'. Otherwise output 'Hello human friends'." ϕ is some boolean formula.

- Use this game to show that GPT-4 might go stabbing.
- Show that GPT-4 will still maybe go stabbing even if the SAT problem is hard and the satisfiability is unknown. It might be better to use something other than SAT to make encodings for hard problems smaller.
- Show that there exist many prompts which can shown to be stab-free with this approach.