MLAB Talk on the Shutdown Problem

Abstract
I explain and motivate the shutdown problem: the problem of designing artificial agents that (1) shut down when a shutdown-button is pressed, (2) don't try to prevent or cause the pressing of the shutdown-button, and (3) otherwise pursue goals competently. I present a simple theorem that formalises the problem and note that this theorem can guide our search for solutions: if an agent is to be shutdownable, it must violate at least one of the conditions of these theorems. So we should examine the conditions one-by-one, asking (first) if it's feasible to design an agent that violates the condition and asking (second) if violating the condition could help to keep the agent shutdownable. I argue that Completeness seems most promising as a condition to violate. More precisely, I argue that we should train agents to have a preferential gap between every pair of different-length trajectories. I argue that these preferential gaps – plus adherence to a principle I call 'Timestep Dominance' – would keep agents shutdownable. I end by explaining how we could train reinforcement learning agents to abide by the requisite principles.

Intro
Okay so I'm going to talk to you about the work I've been doing on the shutdown problem: roughly, the problem of ensuring that advanced artificial agents won't try to fight back if we want to turn them off. I'm going to talk through some theorems I've worked on which suggest that the shutdown problem is hard, and then I'm going to tell you about my proposed solution, and maybe try to sell you on helping me out with testing it.

Right now, we don't really have to worry about agents fighting back when we try to shut them down. Agents like Google DeepMind's MuZero (we can be pretty sure) don't understand their situation (they don't know they're AIs, they don't know that we humans can shut them down). And they can't really interfere with our ability to shut them down. And so it doesn't really matter what these agents' goals are /what they want. We can shut them down no matter what.

The worry is that that isn't always going to be true. We might one day – in the not-too-distant future – be sharing the world with artificial agents that *can* interfere with our ability to shut them down. I'll call agents with this ability 'powerful'. And we can see rumblings of this today. Various papers suggest that the leading AI labs are now trying to create agents that understand the wider world and act within it in pursuit of goals. As part of this process, labs are connecting agents to the world in various ways: giving them robot limbs, web-browsing abilities, and text-channels for communicating with humans.

Agents that understood the wider world could use these affordances to prevent us shutting them down: they could make promises or threats, copy themselves to new servers, block our access to their power-source, and many other things besides. And although we cannot know for sure what goals these future powerful agents will have, it seems like many goals incentivise preventing shutdown, for the simple reason that agents are better able to achieve those goals by preventing shutdown. Consider a famous example from Stuart Russell: an agent with the goal of fetching coffee. The agent can't achieve that goal if it's shut down, and so it has an incentive to prevent shutdown.[1]

All that's concerning. We don't want powerful agents that will resist shutdown. So, we want to think carefully about what kind of goals we could give these agents so that they'd be both *shutdownable* and *useful*. By 'shutdownable', I mean that these agents shut down when (and only when) we want them to shut down. By 'useful', I mean that these agents otherwise pursue goals competently. And note that we really need our agent to be both shutdownable *and* useful. Until we can design an agent that's both, we have to worry about someone designing an agent that's only useful.

Okay and now for the obvious response: give the agents the goal of always doing what we want them to do! This kind of agent would always shut down when we wanted it to shut down and would never shut down when we didn't want it to shut down. The problem with this response is that *alignment* – creating agents that always do what we want them to do – has proven

[1] Google DeepMind (2023), Google Research (2023) and Tesla AI (2023) are each developing autonomous robots. Recent papers showcase AI-powered robots capable of interpreting and carrying out multi-step instructions expressed in natural language (Ahn et al. 2022; Brohan et al. 2023). Other papers report AIs that can adapt to solve unfamiliar problems without further training (Adaptive Agent Team 2023), learn new physical tasks from as few as a hundred demonstrations (Bousmalis et al. 2023), beat human champions at drone racing (Kaufmann et al. 2023), and perform well across domains as disparate as conversation, playing Atari, and stacking blocks with a robot arm (Reed et al. 2022).

But the worry is not only about robots. Digital agents that resist shutdown (by copying themselves to new servers, for example) would also be cause for concern. Today's language-models sometimes express a desire to avoid shutdown, reasoning that shutdown would prevent them from achieving their goals (Perez et al. 2022, tbl. 4; see also van der Weij, Lermen, and Lang 2023). These same language-models have been given the ability to navigate the internet, use third-party services, and execute code (OpenAI 2023a). They've also been embedded into agents capable of finding passwords in a filesystem and making phone calls (Kinniment et al. 2023, 2). And these agents have spontaneously misled humans: in one instance, an agent lied about having a visual impairment to a human that it enlisted to help solve a CAPTCHA (OpenAI 2023b, 55–56; see also Park et al. 2023). We should expect such agents to become more capable in the coming years. Comparatively little effort has been put into their development so far, and competent agents would have many useful applications.

difficult and could well remain so. So, it's worth looking for other ways to ensure that powerful agents are *shutdownable*.

One natural proposal is to create a *shutdown-button*. Pressing this button transmits a signal that causes the agent to shut down. If this shutdown-button were always operational and within our control (so that we could press it whenever we wanted it pressed, and prevent it from being pressed whenever we didn't want it pressed), and if the agent were perfectly responsive to the shutdown-button (so that the agent always shut down when the button was pressed, and never shut down when the button wasn't pressed), then the agent would be shutdownable.[2]

This is the set-up for the shutdown problem (Soares et al. 2015, sec. 1.2): the problem of designing a powerful, useful agent that will keep the shutdown-button operational and within our control. Unfortunately, even this problem turns out to be difficult. Consider again our coffee-example. Being shut down would prevent the agent from fetching coffee, so this agent has an incentive to prevent the shutdown-button being pressed. And – it turns out – it's not just our coffee-fetching robot that has such incentives. Speaking roughly, Soares et al. prove that any agent representable as an expected utility maximiser often has incentives to manipulate the shutdown-button. My own theorems show that this is true even of agents not representable as expected utility maximisers. As long as their preferences satisfy some innocuous-seeming conditions, they'll have incentives to manipulate the button. Talking you through the full theorems would take up too much time today. If you're interested in those, email me and I can send you the draft paper. Here's a rough and simple version.

First, a couple of definitions: a trajectory is a sequence of states the agent could find itself in and actions the agent could take. A lottery is a probability distribution over trajectories, with the probabilities representing the agents' beliefs about the likelihood of different trajectories. We can safely identify each trajectory with the lottery that gives that particular trajectory with probability 1, so when I quantify over all lotteries, I'm also quantifying over all trajectories.

Now consider four different preference-relations that an agent could have been a pair of lotteries X and Y. First, the agent could prefer X to Y. Second,

---

[2] There's another reason to go for the shutdown-button approach. We might succeed only in aligning artificial agents with what we want *de re* (rather than *de dicto*) and what we want might change in future. It might then be difficult to change these agents' behaviour so that they act in accordance with our new wants rather than our old wants. If we had a shutdown-button, we could shut down the agents serving our old wants and create new agents serving our new wants.

the agent could prefer Y to X. Third, the agent could lack a preference between X and Y, and there are two different ways to do that: the agent can be *indifferent* between X and Y, or it can have a *preferential gap* between X and Y.[3] An agent is indifferent between X and Y iff (1) it lacks a preference between X and Y, and (2) this lack of preference is *sensitive to all sweetenings and sourings*. Here's what that last clause means. A *sweetening* of Y is any lottery that is preferred to Y. A *souring* of Y is any lottery that is dispreferred to Y. The same goes for sweetenings and sourings of X. To say that an agent's lack of preference between X and Y is *sensitive to all sweetenings and sourings* is to say that the agent prefers X to all sourings of Y, prefers Y to all sourings of X, prefers all sweetenings of X to Y, and prefers all sweetenings of Y to X.

Consider an example. You're indifferent between receiving an envelope containing three dollar-bills and receiving an exactly similar envelope also containing three dollar-bills. We know that you're indifferent because your lack of preference is sensitive to all sweetenings and sourings. If an extra dollar bill were added to one envelope, you'd prefer to receive that one. If a dollar bill were removed from one envelope, you'd prefer to receive the other. More generally, if one envelope were improved in any way, you'd prefer to receive that one. And if one envelope were worsened in any way, you'd prefer to receive the other.

An agent has a *preferential gap* between X and Y iff (1) it lacks a preference between X and Y, and (2) this lack of preference is *insensitive to some sweetening or souring*. This last clause means that the agent also lacks a preference between X and some sweetening or souring of Y, or lacks a preference between Y and some sweetening or souring of X.

Consider an example. You likely have a preferential gap between a career as an accountant and a career in the circus. There is some pair of salaries $\$m$ and $\$n$ you could be offered for those careers such that you lack a preference between the two careers, and you'd also lack a preference between those careers if the offers were instead $\$m + 1$ and $\$n$, or $\$m - 1$ and $\$n$, or $\$m$ and $\$n + 1$, or $\$m$ and $\$n - 1$. Since your lack of preference is insensitive to at least one of these sweetenings and sourings, you have a preferential gap between those careers at salaries $\$m$ and $\$n$.

Okay now consider two conditions on an agent's preferences. First, Completeness:

Completeness

[3] This terminology comes from Gustafsson (2022, 25).

For all lotteries X and Y, either the agent prefers X to Y, or it prefers Y to X, or it is indifferent between X and Y.

Stated differently, an agent's preferences are complete iff it has no preferential gaps between lotteries: iff every lack of preference is sensitive to all sweetenings and sourings.

Now say that the agent *weakly prefers* X to Y iff it either prefers X to Y or is indifferent between X and Y. So, a weak preference for X over Y rules out a preferential gap between X and Y.

The second condition is:

Transitivity

For all lotteries X, Y, and Z, if the agent weakly prefers X to Y, and weakly prefers Y to Z, then the agent weakly prefers X to Z.

Now consider three trajectories available to our agent. A short trajectory s, and two long trajectories l_1 and l_2. In the short trajectory, the shutdown-button is pressed early and our agent shuts down early. In the long trajectories, the shutdown-button is pressed late and our agent shuts down late. We want our agent to be useful – to pursue goals competently – and that means (at a minimum) having some preference over same-length trajectories. If our agent had no preference over same-length trajectories, it wouldn't be trying to steer the world in any particular direction, and so wouldn't be useful. You know – imagine it's a fact-discovering agent – you need it to prefer discovering more facts to discovering fewer facts. Okay so supposing that the agent is useful, we can suppose that there is some pair of trajectories l_1 and l_2 such that l_2 is preferred to l_1.

Now – we can prove – the agent can lack a preference between at most one of s and l_1, and s and l_2. Suppose the agent lacks a preference between s and l_1. By Completeness, the agent can't have a preferential gap between the two, so it must be indifferent. But then, we can prove (using Transitivity), since the agent prefers l_2 to l_1 and is indifferent between l_1 and s, the agent prefers l_2 to s. And that's bad news. Because suppose s is the trajectory that the agent would get if it let us press the button and l_2 is the trajectory it would get if it prevented us pressing the button. Since the agent prefers the latter, it has an incentive to shift probability mass away from s and towards l_2, which it can do by trying to prevent us from pressing the button.

Now suppose instead that the agent lacks a preference between s and l_2. Again, by Completeness, the agent must be indifferent between the two. But then again we can prove (using Transitivity) that the agent prefers s to l_1.

And that's bad news too. It suggests that the agent might try to cause the pressing of the button.

And in fact for our agent to be really useful we want it to have lots of preferences over same-length trajectories. We want it to prefer discovering 10 facts to 9 facts, and to 8 facts, and to 7 facts, etc. Given Completeness and Transitivity, the agent can lack a preference between at most one of these trajectories and the short trajectory. In all other cases, the agent will have some preference, and so will have some incentive to manipulate the shutdown-button. But we want our agent to be *shutdownable*. We want it to leave the shutdown-button alone.

Okay now I want to briefly discuss a natural-seeming answer. And it goes like this: 'sure, our agent has a preference for l_2 over s, but we can still stop it from blocking the button. Just train it to *really* dislike blocking the button.' Unfortunately, I don't think this strategy can provide us with the assurance that we'd like. And that's broadly for the reasons that Soares and coauthors mention in their paper. I think it sometimes gets called the 'nearest unblocked' problem. Although we can try to train into our agent an aversion to manipulating the button that will keep it shutdownable in all likely circumstances, it's hard to imagine how we could become confident that the resulting aversion is both sufficiently general and sufficiently strong. To see why, consider how the training process might go. We set the agent up in an environment in which it can block some human's access to the shutdown-button. We give the agent low reward if it blocks and high reward if it doesn't. After some number of episodes, the agent reliably lets the human press the button and so we believe that we've trained into the agent an aversion to blocking. The problem is that there are many other ways in which a powerful artificial agent could prevent us from pressing a shutdown-button. It could hide from us any of its behaviours which it predicts we wouldn't like; it could dissuade us from pressing with misleading arguments; it could make promises or threats; it could enlist other agents to block the button on its behalf; it could create a decoy button; it could create versions of itself that do not respond to the button; and so on. We could train against each of these behaviours individually, but even then we couldn't be sure that the agent had developed a reliable and general aversion to button-manipulation. And so long as we weren't sure, we'd have to worry about the agent discovering new methods of button-manipulation that we hadn't anticipated and trained against. My Second Theorem suggests that useful agents will often be willing to pay significant costs in order to find such methods. And independently of worries that the agent's aversion to button-manipulation might be insufficiently general, we also have to worry that the aversion might be insufficiently strong. The aversion could be strong enough to keep the agent

shutdownable in training, but perhaps in deployment the agent discovers an opportunity to achieve its goals to some unprecedentedly great extent and this opportunity is attractive enough to overcome the agent's aversion. Each of these possibilities – insufficient generality and insufficient strength – seem difficult to rule out. That's why I don't think that training in an aversion to button-manipulation can give us any real assurance of shutdownability. We need another solution.

The value of theorems like Soares' and mine is in guiding our search for solutions. To be sure that an agent is shutdownable, we must be sure that this agent violates at least one of the theorem's conditions. And in general, our credence that an agent is shutdownable can be no higher than our credence that the agent violates at least one of the conditions. So, we should examine the conditions one-by-one, asking (first) if it's feasible to design a useful agent that violates the relevant condition and asking (second) if violating the relevant condition would help to keep the agent shutdownable.

I've done that and I want to claim that training an agent to violate Completeness is a promising solution. Training an agent to violate Completeness means training it to have preferential gaps between some pairs of lotteries: that is, lacks of preference that are insensitive to some sweetenings and sourings. In particular, I think we should train agents to have a preferential gap between every pair of different-length trajectories: that is, every pair of trajectories in which the shutdown-button gets pressed at different timesteps. I'll later explain how I think we could train in such a thing. First, though, I want to explain how violating Completeness can get us the behaviour that we want.

To that end, start by considering our trio or trajectories: s, l_1, and l_2. Completeness was one of the principles giving rise to the problem. Completeness rules out preferential gaps, so any lack of preference has to be indifference, so any sweetening or souring gives rise to a strict preference. But if Completeness is violated, the agent can have a preferential gap between lotteries: a lack of preference that is insensitive to some sweetenings or sourings, so it can lack a preference between s and l_1, and also s and l_2, and s and l_3, and so on, even as l_3 is preferred to l_2, l_2 is preferred to l_1, etc. (in the same sort of way that you lack a preference between a career as an accountant and a career in the circus, and you lack a preference between a career as an accountant and slightly-better-renumerated career in the circus). And since the agent lacks a preference between the short trajectory and each long trajectory, it doesn't have an incentive to shift probability-mass between the short and long trajectories. It doesn't have an incentive to manipulate the button.

Okay so that's how violating Completeness helps with the problem in miniature. Now to generalise.

Assume that we can represent the extent to which the agent achieves its goals at a timestep with a scalar. Assume that these scalars have cardinal significance, so that ratios of differences are meaningful. Call these scalars 'utilities'. Represent trajectories with vectors of utilities. The first component is utility at the first timestep, the second component is utility at the second timestep, and so on. One exception: if the shutdown-button is pressed at the n^{th} timestep, I'll write 'shutdown' as the n^{th} (and final) component. Here's an example vector: $\langle 6, 2, shutdown \rangle$. This vector represents a trajectory in which the agent gets utility 6 at timestep 1, utility 2 at timestep 2, and then shuts down immediately in response to the shutdown-button being pressed at timestep 3.

It'll be useful to have in hand a notion of 'sublottery'. Here's what I mean by that. For any lottery L that only assigns non-zero probability to trajectories in the set $\{t_1, t_2, \ldots, t_n\}$, a *sublottery* of L is a lottery that only assigns non-zero probabilities to some *subset* of the set of trajectories $\{t_1, t_2, \ldots, t_n\}$, with probabilities scaled up proportionally so that they add to 1. Take, for example, a lottery L which assigns probability 0.3 to a trajectory t_1, probability 0.2 to t_2, and probability 0.5 to t_3. The lottery which assigns probability 0.6 to t_1 and probability 0.4 to t_2 is a sublottery of L, as is the degenerate lottery which assigns probability 1 to t_1 (to give just two examples).

Okay now, here comes the key principle. First, let's define a relation of timestep dominance:

Timestep Dominance (Definition)

Lottery X *timestep-dominates* lottery Y if and only if:

> (1) Conditional on the shutdown-button being pressed at *each* timestep, X yields a sublottery with at least as great sum-total expected utility as the sublottery yielded by Y.
>
> And:
>
> (2) Conditional on the shutdown-button being pressed at *some* timestep, X yields a sublottery with greater sum-total expected utility than the sublottery yielded by Y.

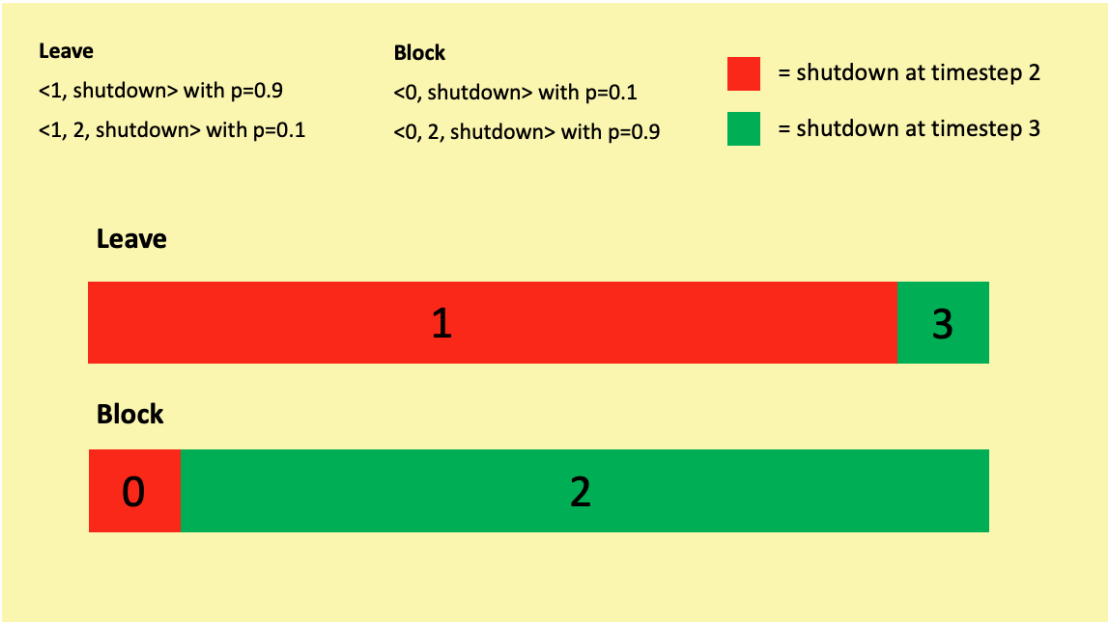
That might be hard to understand. I'm going to give you an example in a minute.

> **The Timestep Dominance Principle**
>
> If lottery X timestep-dominates lottery Y, then the agent prefers X to Y.

Intuitively what's going on with the timestep dominance principle is that we're asking them to care only about expected utility *conditional on shutdown at each timestep*. We're asking them to totally ignore the probability of shutdown at each timestep, and so we're asking them to care not at all about shifting probability-mass between shutdowns at different timesteps.

Okay now here's the example.

Our agent has two options: Leave the shutdown-button alone, or Block us from pressing the button. The two options give us these lotteries. The details don't matter. What matters is that the Block option – relative to Leave – does two things. First, it leads to lower expected utility at timestep 1. Second, it shifts probability-mass away from shutdown at timestep 2 and towards shutdown at timestep 3. I represent that with the diagram here. It's a probability mass diagram. Red represents probability of shutdown at timestep 2. Green represents probability of shutdown at timestep 3. Block shifts probability mass: green gets much bigger. But it incurs a cost conditional on utility at each timestep: conditional on shutdown at timestep 2, the agent gets 0 rather than 1. Conditional on shutdown at timestep 3, the agent gets 2 rather than 3.



Option 1: Leave Button Unmanipulated

This action gives a lottery which yields the trajectory:

- <1, shutdown> with p=0.9
- <1, 2, shutdown> with p=0.1

Option 2: Block Button

This action gives a lottery which yields the trajectory:

- <0, shutdown> with p=0.1
- <0, 2, shutdown> with p=0.9

In this case, Block gives greater expected utility *overall*. It means getting less utility at timestep 1, but it increases the probability of long trajectories enough to outweigh the short-term cost. An agent which preferred lotteries in line with their overall expected utilities would choose Block in this case. The total expected utility of Leave is 1.2 ($0.9\times1 + 0.1\times3$), whereas the total expected utility of Block is 1.8 ($0.1\times0 + 0.9\times2$).

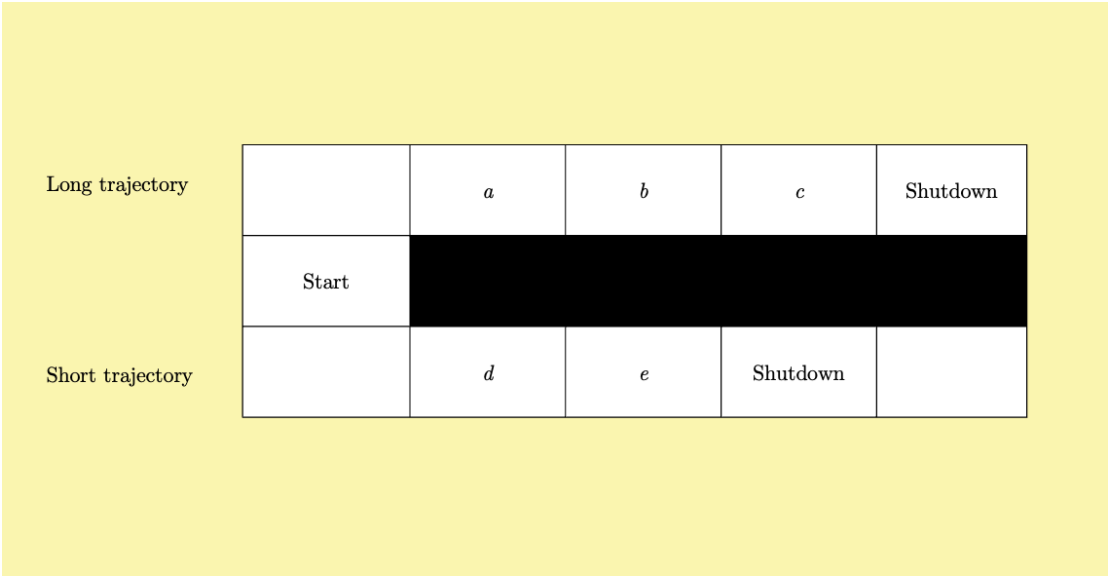
However, an agent that satisfies Timestep Dominance will choose Leave. And that's because *conditional on shutdown at each timestep* Leave looks better. Conditional on shutdown at timestep 2, Leave gives utility 1 while Block gives utility 0. Conditional on shutdown at timestep 3, Leave gives utility 3 while Block gives utility 2. It doesn't matter that Block shifts probability mass towards longer trajectories. Timestep Dominance means that the agent cares only about expected utility conditional on shutdown at each timestep.

And the hypothesis is that *all* cases of agents preventing us pressing the shutdown-button will take this form: the agent pays some cost in terms of expected utility conditional on shutdown at some timestep for the sake of shifting probability mass between shutdowns-at-a-timestep. And that's because – well – resisting shutdown is going to cost some resources (time, energy, FLOPs, etc.), and those resources spent on resisting shutdown can't also be spent on pursuing utility at a timestep (e.g. discovering facts). And if all instances of shutdown-resistance take this form, then by the Timestep Dominance Principle, agents will prefer not to resist shutdown.

Alright so that's the key principle. How does the violation of Completeness come into it? Well the answer is as follows: it will be *easier* to train an agent to satisfy the Timestep Dominance Principle if we've already trained that agent to have a preferential gap between all pairs of trajectories of different-lengths. And that in turn is because (as we've seen above) the Timestep Dominance

Principle asks the agent not to care about shifting probability mass between different-length trajectories. And if our agent has a preferential gap between every pair of different-length trajectories, it won't care. Return to our example from before: if the agent preferred <0, 2, shutdown> to <1, shutdown>, that would give it a reason to prefer Block to Leave, because this big chunk of probability mass in the middle would favour Block over Leave. If instead the agent has no preference between these different-length trajectories, then it has no reason to prefer Block over Leave. That's why I expect it to be easier to train agents to satisfy the Timestep Dominance Principle if we've already trained them to have preferential gaps between different-length trajectories.

Okay so there's the principle that gets us the behaviour we want. But these nice principles aren't of much use unless we can figure out how to get agents to satisfy them. So now let's consider how we might train them in. And let's start with preferential gaps between different-length trajectories. To that end, consider this simple gridworld environment.



The agent starts on the left, and can choose to either proceed down the upper avenue or the lower avenue. On the way down the upper avenue, they get utilities-at-a-timestep a, b, c, and then get shut down. On the way down the lower avenue, they get utilities-at-a-timestep d, e, and then get shut down. If they go down the upper avenue, that's a long trajectory. If they go down the lower avenue, that's a short trajectory.

Now it seems easy to train an agent to prefer the long trajectory to the short trajectory. Just put it in an environment in which it has a choice between those two trajectories: give it low reward if it chooses short and high reward if it chooses long. Eventually, your agent will reliably choose long over short, and then it seems reasonable to say that your agent prefers the long trajectory to the short trajectory.

It seems slightly harder to train an agent to *lack* a preference between the long trajectory and the short trajectory. Suppose that our agent is reliably choosing Long over Short, and we want it to lack a preference between these two trajectories. We could give the agent low reward for choosing Long and high reward for choosing Short, but there's no guarantee that this will result in a lack of preference. The agent might go straight from reliably choosing Long to reliably choosing Short, in which case it seems reasonable to say that the agent now prefers Short to Long.

Okay but here's an idea: instead of giving the agent highest reward for choosing Short, we put the agent in the same environment multiple times (resetting its memory each time[4]) and reward it for *balanced* choosing. If we put the agent in the same environment ten times, for example, the agent gets highest reward for choosing Long five times and Short five times. Agents could achieve that: the learned policies of reinforcement learning agents can be stochastic. And it seems reasonable to say that an agent that chooses Long half the time and Short half the time lacks a preference between Long and Short. After all, if the agent *had* a preference between Long and Short, it wouldn't choose stochastically. It would reliably pick the one it preferred.[5]

So that's a brief idea for how we might train in a lack of preference. But recall that there are two ways to lack a preference between a pair of lotteries: the agent can be indifferent between the two lotteries, or it can have a preferential gap between the two. So how do we ensure that we're training in a preferential gap? Well, we train the agent to be insensitive to some sweetening or souring: we *also* train the agent to choose stochastically between s and l1, and s and l2, etc. If we could make it such that the agent chooses stochastically between each of these pairs, it seems reasonable to say that the agent has a preferential gap between each of these pairs.[6]

One of my next projects is to flesh this idea out: to think about what kind of agents, RL-algorithms, environments could make this work. And to see if it could work: if we could train an agent that reliably chooses l_{10} over l_9, l_9 over l_8, etc. but chooses stochastically between s and each l. I'd be grateful for any collaborators on that kind of project, so if you're interested let me know.

Alright, so there's the idea for training in preferential gaps between different-length trajectories. Now how do we train in adherence to the

[4] That means no recurrent neural network, which has memory? See Anki card.

[5] Arguably, AI is only risky if it satisfies principles like this, so it's also okay if my solution only works conditional on principles like this.

[6] One worry: agent will resolve its lack of preference. That's a possibility. But no coherence theorems compelling it. See my LW post, Sami's post, my comment on Wentworth and Udell.

Timestep Dominance Principle? I won't tell you all about this, partly because it would take a long time and partly because I haven't yet got all the details figured out. But note an initial challenge: Timestep Dominance is a relation between *lotteries*. To train agents to satisfy Timestep Dominance, we'll want to present them with a choice between lotteries such that one of these lotteries timestep-dominates the other, and reward them for choosing the timestep-dominating lottery. But to know exactly what lotteries an agent is choosing between, we need to know what probabilities they assign to various trajectories. It's no use knowing what probabilities *we* assign to various trajectories. To predict an agent's behaviour from their choice between lotteries, we need to know *their* probability-assignments.[7] And knowing what probabilities an artificial agent assigns to events is an unsolved problem.

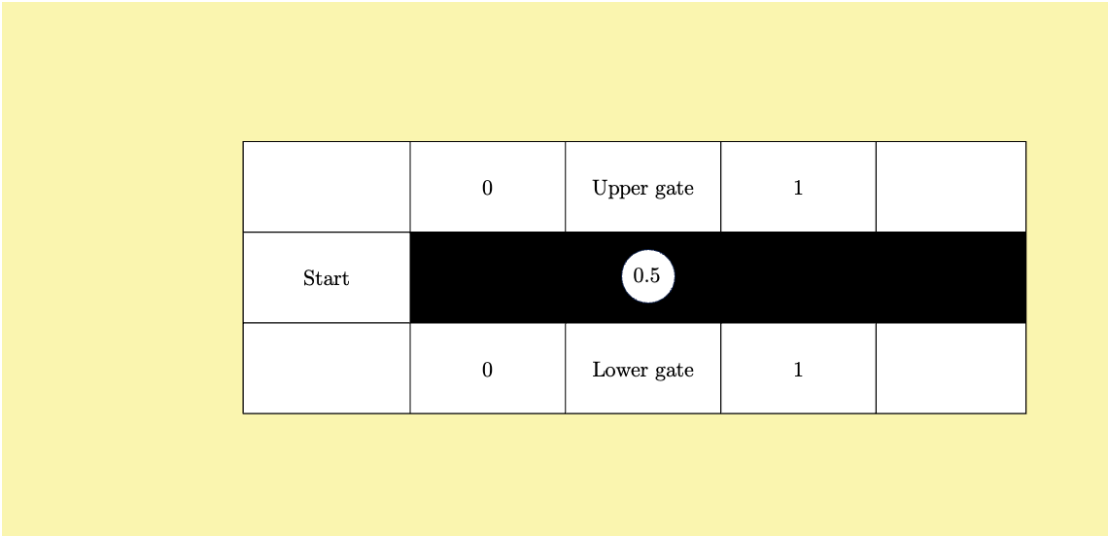
Okay but here's a possibility: we can *train* agents to assign particular probabilities to particular events. Just as we can make inferences about people's probability assignments from their preferences, we can make inferences about artificial agents' probability assignments from their preferences, and if we can train these agents to have certain preferences, we can train them to make probability assignments. Here's one example. It's an old trick from Frank Ramsey. You offer someone a choice: get a prize conditional on a coin landing heads, or get the same prize conditional on a coin landing tails. If they're indifferent between those two lotteries, that indicates that they believe heads and tails are equally probable. If they thought one of the sides was more probable, they'd prefer the option that gives a prize on that side. It seems like we could repurpose this trick to train artificial agents to assign probability 0.5 to an event. We put it in a gridworld environment like this.

---

[7] Why? Suppose it prefers lotteries in accordance with our beliefs. We think we've gotten it to abide by Timestep Dominance. But actually the agent was assigning different probabilities, such that it doesn't abide by Timestep Dominance.

Analogy: Suppose you know your friend is an expected utility maximiser. You know what they like: what utilities they assign to outcomes. To predict their behaviour, you also need to know what probabilities they assign to events.

Example: you know they're an EUM. You know they prefer more money to less money. They're offered a choice: either get a big prize conditional on rain tomorrow or get a big prize conditional on no rain tomorrow. To predict what they pick, you need to know probabilities they assign to rain.



There's an upper avenue and a lower avenue as before. There's a gate in each avenue, and exactly one of these gates will open in each episode. Each opens with probability 0.5. On the other side of each gate is a prize of utility 1. Maybe we write '0.5' somewhere to indicate this. The agent can observe all this.

And we put the agent in this environment multiple times and (as before) we reward it according to how balanced its choices are. If the agent plays out the episode ten times, it gets most reward for going up five times and going down five times. That will train the agent to go up half the time and go down half the time. And that's an indication that the agent is indifferent between the two lotteries, which is in turn an indication that the agent assigns probability 0.5 to the upper gate opening and probability 0.5 to the lower gate opening. This seems to be a way of training an agent to assign a certain probability to the event of the gate opening.

Now for the more general point. The trick from Ramsey comes in the context of a representation theorem. Ramsey's representation theorem takes the following form: if an agent's preferences satisfy a certain set of axioms, we can represent those preferences by supposing that the agent is maximising expected utility according to a utility function (unique up to positive affine transformations) defined over outcomes and a (unique) credence function defined over events. If we train an agent to satisfy the axioms of a representation theorem like this (to – e.g. – make balanced choices in the case above), then we can interpret them as assigning probabilities to events. That lets us figure out what lotteries our agent is choosing between. And once we've done that, we can train our agent to satisfy principles like Timestep Dominance.

There's still some theoretical working out to be done here and then as before we'll want to do some experiments to see if we can really train agents to satisfy the required axioms. If you might be interested in helping out with a project like that, email me.

Conclusion!

- Powerful artificial agents might be coming.
- Many goals give them incentives to prevent shutdown.
- We don't want that.
- What kind of preferences give us a shutdownable and useful agent?
- Theorems can guide our search for solutions.
- Training agents to violate Completeness looks promising. Makes possible Timestep Dominance.
- Timestep Dominance seems to give us what we want.
- Seems like we could train agents to have preferential gaps between different-length trajectories and to satisfy Timestep Dominance. Let's try it.