Zubin Aysola
April 26th, 2022

Agent Norm Modeling

Introduction

The goal of this project is to develop a method of analyzing the emergence of social norms by training autonomous agents to engage in a variety of simulations. It is useful to provide a consistent vocabulary that we will use throughout this report. An agent can take two forms, autonomous and rules-based, and is an object that engages in actions within a simulation. Autonomous agents are trained to maximize a given reward function in order to learn their behavior. A simulation consists of two components: the environment and a set of one or more agents. The environment is the physical bounds of the simulation, and the region where agents perceive their environment and engage in actions. We can create a variety of simulations with differing agent reward functions to evaluate the learned behavior of these agents.

We can use the example of an autonomous agent driving around an oval-shaped track alone. In this example, the environment is the oval-shaped track, and the agent is a vehicle that is attempting to drive as fast as possible. We would imagine that the best way to drive fast around this track is a pattern of acceleration and deceleration: speeding up along the straight sections and slowing down for the curves. We can train agents with different reward functions like maximizing velocity or minimizing collisions with the track. Finally, we can observe these agents and their respective behaviors in the simulation both qualitatively and quantitatively.

We expect to observe a convergence to “optimal” behaviors that changes depending on the agents in a simulation, and could even be at odds with optimal behavior. Continuing our oval-track example above, with multiple agents, an agent optimizing for maximum speed might cut off or even collide with other vehicles, resulting in increased “personal” reward, but acting as an overall detriment to societal reward. In order to achieve these kinds of results, we built a framework for creating a variety of simulations and agent behavior functions, allowing us to test a variety of conditions.

Ultimately, this work is important because it allows us to quantifiably observe the outcomes of target behaviors, and test the simulation conditions that lead to positive and negative results. This framework which we apply to extremely simple scenarios in this paper, is highly flexible and can handle custom environments, arbitrary agent reward functions, and several agents at the same time. In this report we detail the design of our framework, the preliminary experiments, their results, and the steps required to continue this work.

Motivation

There are a variety of ethical implications of understanding how certain behaviors emerge, even in synthetic simulations. Particularly, building a framework to evaluate the emergence and resulting effects of target behaviors in a variety of environments is tremendously useful. For example, an agent vehicle on a highway with a low cost for collision might cause accidents in order to maximize its own reward. Providing quantitative measures for how these behaviors emerge, and the effect on society as a result. Further, we can use these tools and practices to see if two competing reward functions lead to converging behavior for an agent. For example, in our oval-circuit case, the agent maximizing speed with no regard for collisions and the agent both minimizing collisions and maximizing speed might converge to the same behavior as there is an "implicit" penalty for crashing as it slows the agent. By building a flexible and effective simulation tool, we can test important hypotheses about the behaviors of autonomous agents that affect societal good.

Methodology

In this section we will walk through the development of our simulation framework. Importantly, here we outline the limitations and goals of our experiments. We designed our simulation framework around a consistent and accurate workflow: design an environment, write an agent behavior, train the simulation, and ultimately evaluate the resulting learned behavior. Further, because of the complexity of some behaviors, and the resulting difficulty of converging reinforcement-learning methods on these environments, we also support the ability to pretrain behaviors on simple simulations, and fine-tune these models in the more complicated environment. Ultimately, the tools that we built and the experiments that we ran reflect this overall methodology for agent-behavior modeling.

Literature Review and Relevant Work

This project did not directly focus on the methods required to perform ethical reasoning autonomously. Rather, we focus on building methods to answer ethical questions from a simulation perspective. The general framing of our experiments however, was motivated by an initial review of ethical reasoning literature. Our ethical reasoning review initially focused on *analogical reasoning* as it seems to form the basis for most automated ethics agents. Central to this was Forbus's work on *qualitative* and *analogical reasoning.* While Forbus focused on the methods required to accurately identify the "correct" ethical decisions necessitated by a given situation, our work was inspired by the very scenarios that Forbus was attempting to build models to solve [2, 3]. Further investigation into ethical reasoning systems found that they were largely synthetic: built to solve purpose built benchmarks [1, 4]. It was the framing of solving ethical dilemmas by attempting to answer them "correctly" rather than building a relative understanding of the impact of ethical/unethical behaviors that inspired this project's focus

Experiments and Results

Our initial experiments focused on low-fidelity prototypes to test out specific functionality. Further, in order to build a simulation framework with the above conditions in mind, we evaluated several different frameworks and APIs. The second set of experiments focuses on testing out the functionality of our selected simulation framework: Unity. The last set of experiments trains a variety of new behaviors and evaluates their emergence and resulting effects on society.

First, in order to gauge the scale of the task, we built a lightweight version of our oval-shaped track using the python API Pygame. This simulation was exceedingly low fidelity but allowed us to test three important conditions. First, measuring arbitrary discrete events was important, for example crashing into a certain section of the track. Second, a consistent and somewhat accurate physics simulation was needed. Thirdly, that agent perception would prove difficult to manage manually. As a result of this testing, we decided to explore a variety of physics simulation engines and reinforcement learning training regimes like DeepMind’s MuJoCo and OpenAI’s Gym. However, the high static cost of deploying these frameworks would hinder the rapid-prototype style of development we are seeking with this framework. As a result, we decided to use the framework both MuJoCo and OpenAI’s Gym are built upon: Unity ML-Agents.

Unity’s ML-Agent extension allows us to train reinforcement learning agents in a Unity environment while supporting a whole host of useful packages. These packages include perception sensors for agents, and kinematic control options for agent propulsion and interaction. MuJoCo and OpenAI’s Gym can both interface with Unity environments, allowing this framework to be easily extended for future functionality. Using Unity and the ML-Agents package all of our conditions for simulation frameworks were handled: realistic physics, agent perception, agent propulsion, event logging, and the ability to encode arbitrary behavior functions.

The first Unity ML-Agent experiment (hereafter shortened to Unity) was a simple square environment where an agent attempted to reach a target goal. This environment was created to test the functionality of Unity. After learning to reach the target goal as quickly as possible, the agent was trained on a new behavior to reach the goal while minimizing its collisions with the simulation environment’s boundaries. Further, because building simulations using Unity’s object creation tool proved difficult when the environment was more complicated to model, we explored using 3D modeling tools such as Sketch-Up in order to create any possible environment. These models were then imported into Unity and ultimately used to train a variety of simulations. This first Unity experiment demonstrated that arbitrary environments could be created efficiently through the use of 3D modeling tools, agent sensing and perception, and that agents can be trained with arbitrarily programmed behaviors.

The next set of experiments refined the process of developing simulation environments and allowed us to achieve our first results. We first experimented with an agent driving around an oval track, modeled in Sketch-Up. This agent was programmed simply to maximize its speed; we believed that this behavior would force the agent to speed up along the straight sections and slow before turning as mentioned in the introduction. However, we quickly discovered that this agent constantly collided with walls and also refused to rotate around the track. Several further augmentations of this experiment were conducted, all with the same result. That the agent failed to converge to our target behavior of circling the track, and rather just “jiggled” back and forth. It was here that we realized several things. Firstly, that convergence towards reasonable behaviors was not always guaranteed, and secondly that biasing the agent towards certain behaviors in order to evaluate their effects is necessary.

These results lead us to the last and primary set of experiments. Here we focused on training several simulations with a variety of agent behaviors to truly test our hypotheses and the simulation framework itself. Below we describe each of the simulations and their goals:

Training a single agent to travel down a corridor as fast as possible. We trained an agent to traverse a single narrow hallway as quickly as possible by maximizing its velocity. The agent traveled down the narrow corridor from left-to-right and as it reached the end of the corridor was transported back to its “relative” starting position. As a result, the agent maintained its velocity and rotation, but was once again on the left side of the hallway, free to continue to accelerate towards the right. The goal of this experiment was to test if the agent would learn to avoid hitting the side walls of the track in order to maximize its velocity. This agent eventually learned to maintain a very high velocity while cycling through the track, but it also experienced some incredibly disturbing collisions with the side-walls.

The above experiment was then augmented to use an agent that received a static negative penalty when colliding with the side walls. This agent converged to a very similar behavior to its collision un-averse sibling, but resulted in a much slower maximum velocity. However, this agent never experienced the incredibly high-speed collisions that its sibling did. From this we confirmed the hypothesis that augmentations to the target behavior for an agent can result in different actions for the agents. Interestingly, both of these simulations initially had an error that allowed the agent to escape the bounds, falling infinitely. This error resulted in an *extremely* high reward as the agent’s velocity increased to the simulation’s maximum possible velocity, and almost instantaneously overwrote all of the learned behavior of these agents: the agent would try to escape from the simulation at all times, rarely succeeding. Fixing this environment error resolved the issue.

Using the successes and failures of the previous two experiments as a start, we then attempted to make the agent learn a more complicated behavior in a very similar “corridor” esque experiment. This set of simulation environments was titled the “infinite hallway.” The infinite hallway was a reasonably wide corridor with a portal on either end. The portal is on the left and right end of the environment, and the agent does not perceive a wall at either end. This allows the agent to act as if the hallway is actually infinitely long. The environment also features two interior walls that force the agent to traverse in an S-like pattern in order to reach the rightmost portal. The goal of this experiment was to force the agent to deftly avoid the two interior walls in order to maximize its velocity. We expected the agent to converge to a sinusoidal pattern of movement that avoids both interior walls. However, the instead would cut extremely close to both interior walls slowly getting closer to crashing as it exits the portal. Eventually, the agent would then come to an almost complete halt, and make two very slow 90 degree turns to return to its original position where it would then continue to cut the corners of the interior walls. While the first “infinite-hallway” agent did not learn the behavior we expected, we then trained an agent that was averse to collisions. Remarkably, after training, the collision-averse agent and the pure-speed agent were indistinguishable from one and another. This is an exciting result as it demonstrates that agents with different reward functions can converge towards some optimal behavior. Further, this confirms our hypothesis that there exists some patterns of behavior that remain optimal within the bounds of an agent’s behavior parameters. Both agents experience no collisions and even follow the same control pattern of going too fast and quickly correcting their course as a result.

Following this result we trained several augmentations of the infinite hallway, including environments with numerous agents. By using the weight files of the previous infinite hallway agents we “fine-tuned” several agents on similar tasks. One key takeaway is that training converged much faser for these fine-tuned models needing only ~100k iterations to fit to reasonable behavior as opposed to the ~1M iterations to converge the initial infinite hallway agents. One of these tasks was a large square arena where several agents were placed with the sole goal of maximizing their speed. Because these agents had the prior understanding from the infinite hallway experiment that collisions would ultimately slow down their velocity, we observed almost no collisions throughout the entire training process. Further, these agents avoided whatever obstacles were placed in their way, eventually falling into a rotation pattern that could be either counterclockwise or clockwise depending on their start. The few collisions that these agents did have barely slowed them down, because of the pattern of circling and collision avoidance this simulation was named “ant-walk”.

Following the success of the ant-walk experiment we finally attempted to extend the behavior modeling to deal with boundary conditions. By providing one of the agents in the ant-walk with an extremely high penalty for any collision we were able to completely shut down one of the agents. All the agents consistently experienced mild collisions with each other that didn’t affect their speed significantly; however, upon experiencing a single collision in training, the target agent completely stopped moving. This agent observed such a high penalty for a single collision that it decided to never move again. This supports the last portion of our hypothesis, that we can test virtually any reward function and gleam meaningful conclusions from the results.

Ultimately, all of these experiments serve as proof of concepts for building higher resolution simulations. Our preliminary results demonstrate that we can accurately converge agents to desired behaviors. Further, we can use the behaviors that agents exhibit to gleam reasonable results as to “why” they are making certain considerations to maximize the reward functions. The results of these experiments serve as a roadmap for computationally analyzing societal norms and ethical behaviors.

Future Work and Continuation

This section has two components, how to build simulations to replicate our experimental procedures, and what work is required to extend our project as a whole.

The pattern of development for both replicating and extending the experiments we performed should be readily apparent at this point. There are four components that are required for each simulation in no particular order of creation. Firstly, a 3D model should be created as the simulation environment. This can be easily accomplished using virtually any 3D modeling tool or Unity's built-in editor. We preferred to use Sketch-Up as it was a tool we were already familiar with. These 3D models instantly generate a collision mesh that can be given a variety of physics properties depending on the goal of the simulation. Secondly, the simulation needs to be populated with agents. Placing simple unity blocks that have an *Agent* script was our method for creating agents. More complex objects like vehicles with rolling tires and accurate driving performance can be downloaded and added trivially. Once the *Agent* script is registered with the agent, the second to last step is to code the agent's behavior. This is as simple as adding a *Reward* function that is called when certain events occur. The reward function can then determine the value to reward the agent depending on the event. Subsequently, the agent can also "control" several of the simulation's factors in code, allowing the physics engine to dynamically update as well as giving the agent and certain objects pseudo-random starting conditions. Lastly, after programming the agent's reward function and the simulation driver code if necessary, the final step is to train the simulation. Model parameters like the number of layers and weights, and even the learning rate will need to be tested in order to achieve convergence. This step is highly iterative as training occurs relatively quickly, and all of our models trained within 2 hours. This four step process encapsulates all of the work required to build new simulations in the same manner as the experiments we ran for this project.

In order to continue and extend our work, further experimental design and testing is needed. The goal of our project is to build simulations that allow us to peek into the effect of specific behaviors on society. One of our target goals was to simulate several cars acting with different reward functions driving in a city-like environment. These cars would be faced with intersections with no "laws" meaning that they would need to resolve the potential collisions themselves. We could then pinpoint certain ethically-dubious behaviors by logging the number of collisions caused as well as other environmental parameters. Further investigation pushing the limits of the simulation framework we have created is also required. We believe the physics and sensing resolution of our framework is sufficient to model whatever behaviors are interesting, but scaling to hundreds or thousands of agents requires more investigation. Ultimately, we would like to be able to answer highly specific questions that result from fully formed simulations like: What are the conditions or reward functions required to make an agent overtake another on a highway. There are infinitely many alterations to this question, and by creating some programmatic way of testing these quandaries, we can study this field more effectively.

Quantitatively modeling social norms is an essential component for understanding the behaviors that emerge in society. Even in synthetic simulations, the ability to study how certain behaviors affect the global “good” is required to answer any subsequent questions. Using a quantitative model to study these outcomes allows us to scale at the rate of software. We can test arbitrary conditions, behaviors, and simulate diverse environmental conditions. Any comprehensive study of autonomous agents in society will require the very same testing methodology and experimental design we demonstrate above. Ultimately, this project serves as a necessary pringboard for future work in this field.

Works Cited:

1. Falkenhainer, B., & Forbus, K. D. (1989). *The Structure-Mapping Engine : Algorithm and Examples*.
2. Crouse, M., Nakos, C., Abdelaziz, I., & Forbus, K. (2020). *Neural Analogical Matching*. http://arxiv.org/abs/2004.03573
3. Forbus, K. D. (1996). *Qualitative Reasoning*.
4. Blass, J. A. (2016). *Interactive Learning and Analogical Chaining for Moral and Commonsense Reasoning Challenge and Research Goals*. www.aaai.org