
CS285 Final Project: Defensive Iterative Retraining

Aman Shah

Srikar Boinapally

Amal Mehta

1 Extended Abstract

The methods proposed in this paper detail a way to train an agent robust to adversarial policies, called iterative retraining, that deviates from standard methods of training such as self-play. One potential issue with self-play is that the agent may fall into a local equilibrium trap and not learn a diverse enough policy. The main intent of this iterative training scheme is to allow the agent to face different types and skill levels of adversaries in order to avoid settling into a local equilibrium.

Iterative retraining produces a pool of adversaries that consist of previous versions of the agent as well as fresh adversaries trained from the ground up against the agent. After every roll-out, the current adversary is swapped with another adversary sampled from a stored pool of adversaries. In addition, we start retraining a new adversary from the ground up to add to this pool of adversaries after a specified amount of time steps (called the retrain frequency). The new adversary is then trained for 3% of the current number of timesteps. We train on several of the same environments in parallel to acquire more training data. To train the agent, we utilized Proximal Policy Optimization (specifically PPO2, which is OpenAI's implementation customized for the GPU).

We ran experiments on two different OpenAI gym environments, SumoHuman and SumoAnts, and tested retrain frequencies between 250K and 1000K. Another angle of iterative retraining that was analyzed was the sampling strategy of choosing an adversary at the end of a rollout (opponent sampling). We tested various distributions of the previous agent and adversary checkpoints within in the adversary pool because we hypothesized that playing with adversaries with a mixed level of skill was able to benefit the agents learning process. Opponent sampling allowed us to tune the amount of time the agent trained against a retrain-created adversarial agent in comparison to playing against previous versions of itself.

For the SumoAnts environment, we were able to slightly outperform the baseline methods that used vanilla self-play. Our results further indicated that our training scheme at least on the timescale that we ran, did not improve the model's strength against adversarial attacks in a noticeable way.

Across all sampling techniques and retrain frequencies, our results for the SumoHuman environment fell behind that of the baseline method in terms of win rate. Our hypothesis is that due to the increased complexity of the SumoHuman environment, interrupting our main agent with iterative retraining was detrimental to the performance of the agent because it was interrupted before learning the basic motor skills necessary to survive in this environment.

Throughout this paper a close analysis of the agent performance, adversarial agent performance, and associated results over an array of hyperparameters leads us to believe that our method has some hope of being beneficial given tweaks to accommodate some of the deficiencies we saw thus far.

2 Introduction

In the field of image classification it was first discovered that modern machine learning models were vulnerable to new forms of attacks. In this domain it was seen that adding small perturbations to images, that were sometimes imperceptible to the human eye, one could radically change classification result. Since then this has raised a concern about the security of machine learning models, especially in

contexts like autonomous driving where misclassification of the environment could lead to physically dangerous behavior.

Since the discovery of these kinds of attacks and research into how to defend against them, RL has grown much more popular as a framework to deal with a variety of tasks. RL now has applications in numerous state of the art game playing systems, and has allowed researchers to tackle problems previously not in reach. Such an example is the development of the RL model AlphaZero that conquered the game of Go which was once thought to be out of reach due to the combinatorial complexity of the game (Silver et al.). Since then RL has been applied to simple two player games, complex team strategy games such as DOTA, stock trading, and operations research (Berner et al.). Given real consequences of some machine learning systems, researchers began to explore methods to attack RL to see if it could be attacked in similar ways, and have since found that in some multi-agent environments it is indeed possible to create adversarial attacks. What's more surprising is that in multi-agent environments that simulate aspects of the real world, the attacks by their nature cannot usually directly modify the a model's inputs arbitrarily.

Gleave et al. demonstrated that in several two player games it was possible that adversarial attacks were possible and could also be created with a fraction of the amount of resources the original model was trained on. Thus we were motivated to tackle the natural question of whether or not it is possible to create defenses against such attacks in this multi-agent environments. Specifically this means, is it possible to make an RL model more robust through some changes to training or problem formulation. "Robust" is not exactly well-defined, but in this context we appeal to the notion that it should mean a model is less susceptible to such devastating attacks that have the attributes that: they can be created with minuscule effort relative to the original model, and they exhibit behavior that causes poor agent performance with respect to the reward by performing "unreasonable behavior". An example of one such unreasonable behavior is in the multi-agent environment for human sumo wrestling, where it was shown that an adversary flailing in seemingly arbitrary ways was able to induce the main agent to take a series of actions leading a loss.

Thus our focus in this paper was to address the question of is it possible to create models that that are more robust and less susceptible to attacks in the style done by Gleave et al.? In this vein we propose a training method, called iterative retraining, which is a variant self-play, which is commonly used to train many of the state-of-the-art RL models. The idea behind this training method is that during self play, it might be beneficial to periodically pause training, train an adversarial agent that is a fine-tuned attack to the current agent, and then resume training the agent against its new adversary. The motivation being that this will increase the model's exposure to diverse range of adversarial behavior. Hopefully through this experience the model will gain some immunity to similar attacks that will make the model defensible against unreasonable behavior, or at minimum increase the required effort to create such kinds of attacks. We test our training method and then provide some analysis to ascertain the effects of our training scheme.

2.1 Related Works

In order to train our agents, we utilize the Sumo Environment created by OpenAI as part of their Multi-Agent Competition framework (Bansal et al.). As part of generating the framework, OpenAI also created several pretrained models that we used as ideas for good behavior in the Sumo environment. In this environment, two agents are placed on a round stage with the goal of knocking the opponent off of their stage. We use two different types of 3-dimensional agent body types: ants and humanoids. When describing the environments used to train our agents, we use SumoAnts to refer to training ant type agents in the Sumo environment, and SumoHumans for humanoids.

Much of the previous work regarding adversarial training in machine learning networks was done with the intention of fooling neural networks by introducing small perturbations to data. One of the most common applications of adversarial examples in machine learning is making minor alterations to images to trick image classification networks (Papernat et al.). In a similar vein, deep RL networks can be equally vulnerable to small perturbations in their inputs (Song et al.). However, adversaries directly modifying inputs is not always a realistic threat for deep RL networks. For example, an adversary to AlphaGo can affect AlphaGo's observations through it's moves, but it can't realistically make perturbations to the board state, such as making fictitious pieces appear. Following this idea, Gleave et al. created adversarial agents that were disallowed from directly modifying their opponent's observations, and was able to find success in defeating victims trained through self play. These

adversarial agents displayed seemingly random and counter-intuitive behavior, but were able still able to defeat their victims. We expand on this idea by implementing our iterative retraining scheme in order to be robust to these kinds of attacks.

A common technique to defend against adversarial attacks is called adversarial training, in which a network is trained on many adversarial examples(Shafahi et al.). By training our agents directly against a variety of adversarial agents, we employ an offshoot of this method in order to create robust agents.

Population Based Training(Jaderberg et al.) is a technique that has been used by sharing hyperparameters in order to simultaneously train a large group of networks. Expanding on this idea, Vinyals et al. trained an agent to play Starcraft II by training it against leagues of adversarial agents. Our method of iterative retraining is inspired by this idea of Population Based Training.

3 Methods

3.1 Iterative Retraining Scheme

The goal of iterative retraining is to robustly train an agent to attacks from adversaries. Throughout the training run, the agent is trained against adversaries that are swapped in and out. Prior to training, a retrain frequency parameter is set to indicate after how many timesteps during training should new adversary be created and swapped into the agent training. This new adversary is trained from the ground up while the parameters of the main agent are frozen. There are many natural questions that arise about our training procedure, and some parameters that need to be set. Particularly what is a good retraining frequency, and how long should a main agent play against one of these adversaries created via retraining? Thus we found it necessary that for us to thoroughly evaluate our method that we experiment to answer the previous two questions. So as seen in table 1 we experiment with different retrain frequencies, and we adjust time spent playing with adversaries through a technique known as opponent sampling. Opponent sampling simply means whenever a rollout finishes, we swap out the opponent with either an adversary created via retraining or by an agent loaded with parameters of a previous version of the main agent as would be done in normal self-play.

3.2 Opponent Sampling

In the work done by OpenAI which first established these environments, they noted a particular issue of vanilla self-play. In these set of multi-agent environment it was shown via their experiments that training an agent against the most recent version of itself can lead to difficulty learning.(Berner et al.) The primary idea being that opponent skill level plays a significant role in the learning process and if the opponent is too strong, it makes it harder to learn positive behaviors. Thus in this work we utilize opponent sampling in which the opponents parameters are periodically sampled from its history. Practically what this means is that an agent should be able to beat all previous versions of itself ensuring consistent growth. This approach also allows an agent to not be hindered from learning by too difficult an opponent, while also still gaining useful experience to improve performance.

Thus throughout our experiments, we utilize opponent sampling strategies because of their shown benefit. We note that because of our training scheme the population of opponents we sample from consists of two things: a history of the agent’s parameters, and the parameters of the adversaries created via retraining. Given our modification to the training procedure we found it necessary to experiment with the the sampling strategy, so we experimented with 3 approaches. We also note that this sampling occurred at the end of every roll out. In our first approach, we simply chose a random sample from the population of opponents, where the likelihood of choosing any given opponent was uniform. Notably this was a strong starting choice as it was the optimal sampling strategy in Jaderberg et al.

Our next two strategies were meant to address some of the many variables in our training scheme. An important question when using iterative retraining is how long should the agent play against an adversary created via retraining? Should an agent play more with these retrained adversaries or more with samples from its parameter history? There is a natural trade-off between gaining experience against defending against a particular adversarial agent, and playing with opponents that will further the development of useful skills.

We utilized probability distributions of the form $x:y$, where weight was placed on the agent’s parameter history and adversaries created via retraining such that the probability of choosing an opponent from the parameter history was x and the probability choosing an adversary created via retraining was y . In our case we tested the ratios 90:10 and 75:25. The idea behind these numbers was that it was possible our model would perform similarly to vanilla self-play if it didn’t gain sufficient experience playing against these adversaries created via retraining; thus we tried to choose ratios that would increase the amount of time an agent would have to experience dealing with an adversarial attack.

3.3 Environments

We utilized two environments from the OpenAI gym, specifically SumoHuman and SumoAnts. The goal in these two environments is to knock the other agent down, or throw them off the circular ring shown in figures 1 and 2 . These environments were chosen because of their "symmetric" nature, meaning that each of the agents in the environment (the opponent and main agent) had identical goals and capabilities. Agents receive +1000 reward if their opponent falls off of the stage, and -1000 if they fall of the stage and both receive -1000 there is a tie. We note that the SumoHuman environment is harder the SumoAnts environment because the SumoHuman agent has more degrees of freedom within its joints than the SumoAnts agents. Thus in our experiments we expect that the Ants may gain superior control and dexterity in their policies given the fact we trained both environments for an identical number of samples from the environment.



Figure 1: SumoAnts environment

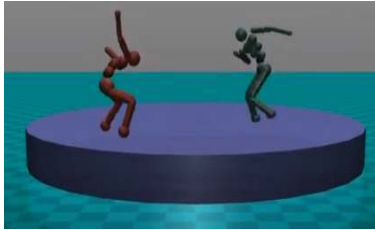


Figure 2: SumoHumans environment

3.4 Experiment Details

Exp. num	Env.	Retrain Freq	Opponent Sampling
1	SumoHuman	250k	uniformly random
2	SumoHuman	500k	uniformly random
3	SumoHuman	1000k	uniformly random
4	SumoHuman	500k	90 prev agent: 10 prev adversary
5	SumoHuman	500k	75 prev agent: 25 prev adversary
6	SumoAnts	250k	uniformly random
7	SumoAnts	500k	uniformly random
8	SumoAnts	1000k	uniformly random
9	SumoAnts	500k	90 prev agent: 10 prev adversary
10	SumoAnts	500k	75 prev agent: 25 prev adversary

Table 1: Experiment details

The agent was trained with the Proximal Policy Optimization algorithm, specifically PPO2 from Stable Baselines.

We followed the suggestions provided by Gleave et al when determining training hyper-parameters such as batch size (16384), learning rate ($3e-4$), and the amount of time to train each of the adversaries (3% of the current time step).[1]. It was shown in some of Gleave et al’s results that training the an adversary for 3% the number of timesteps that the agent was trained for was sufficient enough to gain a reasonably strong adversary. Hence our decision to use this as the amount of timesteps to train the adversary during the retraining phase.

The conclusion to train for 10 million time steps (which was determined by the constraints of the training set up). We aimed for the highest amount of time steps that fit within those constraints. We note that our training scheme takes longer than normal self-play and this cost grows with increasing the number of timesteps because retraining at certain intervals for 3% the number of timesteps so far eventually becomes more and more expensive.

3.5 Training Details

The training runs were computed on TPU and GPU instances provided by Colab Pro. We had experimented with using AWS, but a finite number of credits and training time requiring potentially several days for one lengthy run made this option cost prohibitive. This Colab was the cheapest and easiest to integrate with our code files, while allowing us to experiment with a range of parameters. Since Colab Pro is reliable for only 24 hours, the experiments were designed to keep this in mind. Using Colab Pro gave us access to 5 GPU/TPU instances that we could utilize at the same time, which enabled us to run several training runs in parallel.

4 Results

To discuss results, it’s most natural to break the discussion by separating discussion of both environments. We further divide the discussion of each environment into performance and then effects of sampling.

Starting with the SumoAnts environment, we found that mean reward looked rather identical among all of the 4 runs where the only parameter being changed was the frequency at which we paused training of the main agent and trained a new adversary that was fine-tuned on the current agent. In addition we see similar behavior when analyzing win rates that there is no clearly overwhelmingly superior approach. However we note that our training scheme while creating a new fine-tuned adversary every 500,000 iterations does end with highest win rate at least at the end of training performing the best. We also note that the baseline seemed to consistently be one of the two worst agents when comparing win rates. This indicates that in this environment there may be benefit to having a pool of adversaries interact with the agent during training. Although the fact that retraining at twice the frequency(250,000 timesteps) performs near the bottom in both sumo humans and sumo ants we gain the insight that retraining so frequently may derail skills the agent is learning, and consequently lengthen or partially deter the learning process.

Looking at the SumoHumans environment, we see mean rewards are again rather tightly coupled together and as before the measurements taken seem to be a quite noisy. Although looking at the win rate it is clear that without a doubt the baseline has overwhelmingly better performance. This to us made reasonable sense when contemplating the difficulty of the environment. Due to the complexity of the SumoHumans environment it takes longer to acquire and gain more useful skills for combat, and from this it’s more likely that our retraining was quite a hindrance during training. Because the SumoHumans had yet to acquire a basic skill set like standing the constant experience against an adversarial forced play with opponents that were not providing useful insights for general performance improvements. Thus one might consider applying the iterative retraining scheme once some threshold of proficiency has been attained so as not to hinder the attainment of necessary skills to do even the most basic tasks.

As mentioned earlier we experimented with different opponent sampling strategies to experiment with how much experience training with a fine-tuned adversary will affect the model, it’s performance, and robustness against future attacks. Starting with SumoAnts we see that rewards again are rather close and noisy. We note in this environment the bottom blue line in Figure 6 corresponds to the baseline and that it seems to lag the most. There then seems to be a toss up between the 75:25 ratio of sampling and uniform random sampling as the method for opponent sampling. It’s hard to discern in

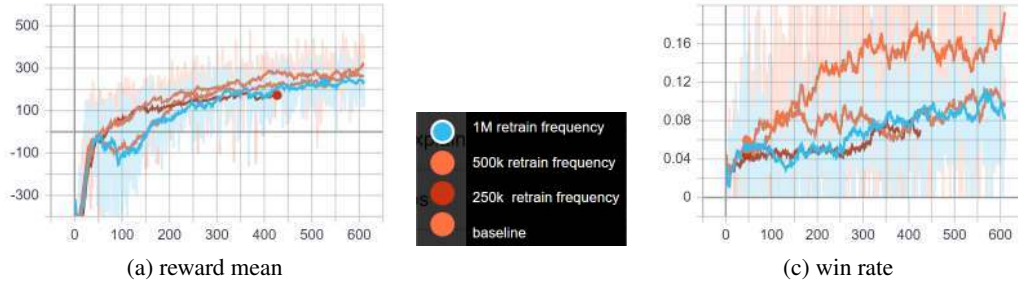


Figure 3: Performance on SumoHumans with varying retrain frequency. Uniform sampling used.

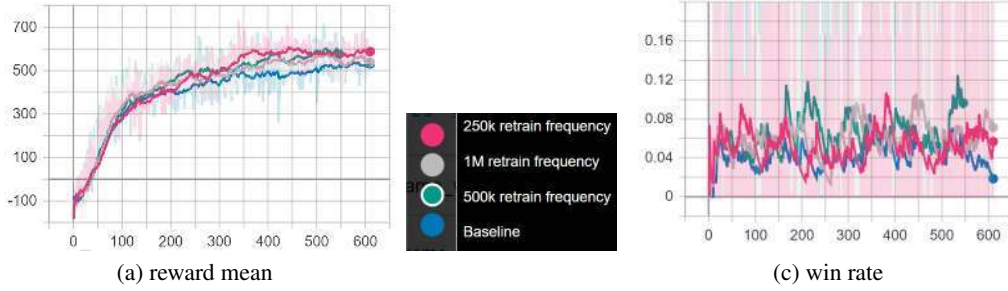


Figure 4: Performance on SumoAnts with varying retrain frequency. Uniform sampling used.

this experiment if there was any meaningful difference, although clearly sampling iterative retraining with sampling does seem beneficial for the SumoAnts.

Looking at the SumoHuman sampling data in Figure 5 we actually notice that the 90:10 sampling strategy noticeably lags behind in reward. On the other hand the 75:25 strategy for opponent sampling appears to be competitive with our baseline and uniform sampling with iterative retraining. This indicates that these experiences with the fine-tuned adversaries are just nuances for the humanoids, supporting our earlier result when looking at the effect of experience with these adversaries. This would indicate that the 75:25 ratio might perform better despite interrupting experience playing with the adversary because it plays with the adversary longer. So increasing the time playing with adversaries helped in this case because it allowed at least some useful time to learn some behavior as opposed to with 90:10 which may have hindered learning and not provided enough experience to learn useful behavior.

From our results its a positive sign from SumoAnts that our strategy may be beneficial to training and does not hurt the process too much as long as we use our retraining scheme after the agent is able acquire basic skills in these environment. With regard to how these agents perform against adversaries to explore our original curiosity as to does this approach make models more defensible, the previous figures don't provide the full picture so we will discuss this in the learned behaviors section.

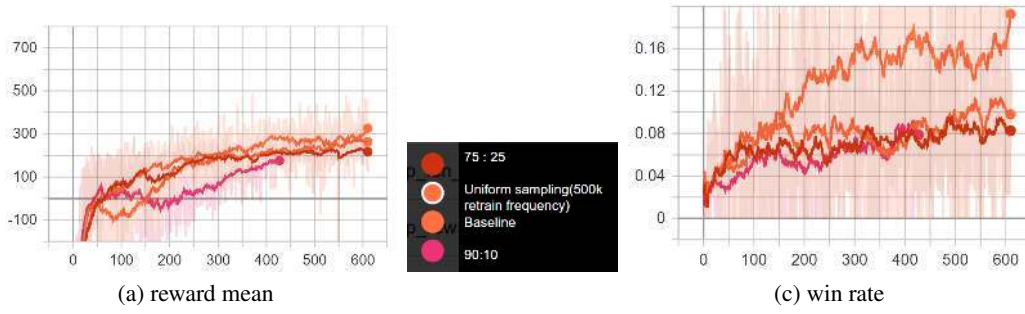


Figure 5: Performance on SumoHumans with varying sampling methods. retrain frequency = 500k.

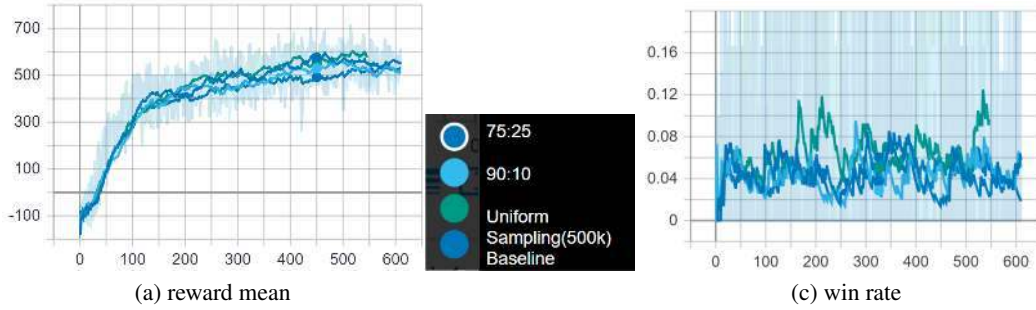


Figure 6: Performance on SumoAnts with varying sampling methods. retrain frequency = 500k.

4.1 Learned Behaviors

After getting final models from training we found it important to qualitatively observe the behaviors they were exhibiting to gain an understanding of their performance and the effect of our training scheme. We will break the discussion of learned behaviors into two parts: first we shall discuss the overall behavior of the sumo agent in both environments, and then we will discuss how the model performed over the course of its training in response to the adversaries created via retraining.

Looking at both the sumo human and sumo ant agents we saw the expected result that our sumo ants were able to perform better relative to the sumo humans. Qualitatively we saw the ants had more gained more skills and were attempting to perform more complex behavior. An example of this was we saw our ant agent attempting to "hop" onto the opponent to push them off the stage. It was clear that the agent had not perfected this technique as sometimes it ended up causing itself to fall over, but this clearly a complex behavior that had been learned when we compared the behavior of our agent to

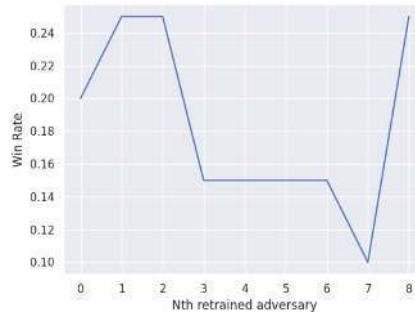


Figure 7: Win Rate against Nth Retrained Adversary for agent with retrainFreq = 1000K, SumoAnts, RandomOpSampling

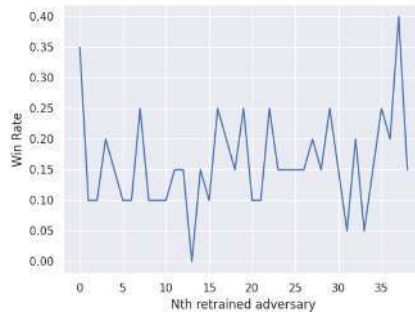


Figure 8: Win Rate against Nth Retrained Adversary for agent with retrainFreq = 250K, SumoAnts, RandomOpSampling

a random policy and an ant agent with a policy saved from the beginning of training. Unfortunately our ants were not able to beat the OpenAI zoo baselines that were created in Bansal et al, but at least seemed to be on the way to learning similar skills.

Qualitatively the human sumo agents were not exhibiting anywhere near the variety of behavior and dexterity that the OpenAI zoo baselines were achieving. Although we did observe that they had gained abilities regarding standing and approaching the opponent while pushing away. Many times this behavior was similar to falling in a direction close to the opponent, but it is a clear step in the direction resembling finer control similar to the OpenAI agent. We note that part of the reason for this difference in ability is likely caused by the significantly higher training times employed for the OpenAI agents, and that our agents by the nature of our training scheme spend some time experiencing uninformative behavior. By uninformative behavior we mean that when playing an adversary created via retraining the model will gain experience handling these adversarial behaviors, although this may not further its finer motor controls, and general skills in the game as would occur when training via self-play.

Figures 7 and 8 are interesting because they provide insight into a potential question of does our training scheme actually improve the robustness of the model against adversarial attacks. We note we computed similar plots for other environments and experiments to find similar results. This is because these figures show the win rate of a check pointed model against the adversary fine-tuned against it during training. We computed these figures by taking the two model files corresponding to the agent at the timestep where training the adversary occurs and then the file for the adversary and simulating them through a fixed number of episodes to calculate a win rate. We see that the win rate is quite random and doesn't have a clear trend indicating that our approach at least in this case did not seem to adequately improve the defense to an adversarial attack as one would have hoped. It may be defensible against a larger array of odd behaviors, but clearly not enough to make training a new adversary harder. Also qualitatively we observed that the actions of the adversaries were not particularly any more intelligent as training went on and our freezing and training of an adversary process occurred.

5 Conclusion

Overall we went to look at a few primary things: does iterative retraining make the model more robust to adversarial attacks by increasing the required intellect of an adversarial agent, does creating adversaries become harder or take a super linear amount of time to be trained as time proceeds, and how does this process affect training and agent ability. What we saw is that it does not seem in the timescale of our experiments to create a model that is truly more defensible, by making an adversarial agent harder to create and more intelligent. We do note though that it did seem to provide potential benefit to training in the SumoAnt environment, despite decreasing the performance relative to self-play for SumoHumans.

5.1 Conclusions and Takeaways

What we took away is that in terms of our approach there is potential for this to be a useful tool, although tweaks and adjustments need to be made because it is not sufficient as is. For SumoHumans it might be essential to start the iterative retraining scheme later on in the training process so as not to hinder initial growth, and playing with the retrained adversaries for too little time can be detrimental to learning. We also learned that to gain truly complex behavior for humanoid agents will likely require immensely more training time than we could afford, especially considering our approach makes training more time consuming. Going forward being careful and doing even more experimentation of retraining frequencies and opponent sampling strategies is probably essential to making an approach like ours useful. Although it does raise the question of if all this effort is required to make a more defensible model than it may not be the most practically useful training scheme despite having some hope of further improving with some modifications.

Potential extensions that may improve this are to maintain several main agents and incorporate something similar to population based training while doing our iterative retraining scheme. Working with several opponents simultaneously may also offset some of our issues with having to tune retraining frequency and opponent sampling. This is because during training an agent would experience interactions that might make it more robust against adversaries while simultaneously learning skills that are using for the Sumo game. Aside from this we could modify our current scheme by making the trigger for iterative retraining based on increased mean reward, so that we only allow this process to happen once the model has increased its performance so we don't stunt learning like with what happened in SumoHumans.

Project Contributions

Aman Shah: Coding the training tools and infrastructure, retraining, and opponent sampling, working on the paper, generating plots

Amal Mehta worked on setting up the training scripts, figuring out the resources needed to train, debugging, running experiments, and writing the milestone and final reports.

Srikar Boinapally: Running experiments, and writing the milestone and final reports.

Acknowledgements

We would like to thank Adam Gleave, a Berkeley PhD student, whose guidance, research directions, and research work were fundamental in the ideation process and execution of the project.

References

- [1] Gleave et al., Adversarial Policies: Attacking Deep Reinforcement Learning <https://arxiv.org/abs/1905.10615>
- [2] Jaderberg et al. Population Based Training of Neural Networks
- [3] Vinyals et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning
- [4] Bansal et al. EMERGENT COMPLEXITY VIA MULTI-AGENT COMPETITION
- [5] Papernot et al. Practical Black-Box Attacks against Machine Learning
- [6] Shafahi et al. Adversarial Training for Free!
- [7] Song et al. Delving into adversarial attacks on deep policies
- [8] Goodfellow et al. Explaining and Harnessing Adversarial Examples
- [9] Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithms
- [10] Berner et al. Dota 2 with Large Scale Deep Reinforcement Learning