Red Team

Kunal Lad
The University of Texas at Austin
kunal.lad@utexas.edu

ABSTRACT

Adversarial learning techniques like Graphical Adversarial Networks have recently gained momentum and have been applied successfully for a number of machine learning tasks. This work focuses on adversarial learning in the context of reinforcement learning domains. It introduces the concept of red team which searches for adversarial scenarios for main agent thereby helping it train in difficult scenarios, making it more robust.

Adversarial reinforcement learning should be particularly helpful in environments where adversarial scenarios are very rare and might not be encountered enough number of times for agent to train well. It can also be used for training agents to avoid catastrophic events. In this work we will demonstrate the effectiveness of this approach by applying it on two toy environments: Windy Grid World and Toy Self Driving Car

1. INTRODUCTION

"Reinforcement Learning is an area of machine learning inspired by behavioral psychology, concerned with how software agents ought to take actions in an environment so as to maximize some notion of cumulative reward. The problem, due to its generality, is studied in many other disciplines, such as game theory, control theory, operations research, information theory, simulation-based optimization, multi-agent systems, swarm intelligence, statistics, and genetic algorithms" [1]

In many real life scenarios like self driving cars or medical treatment agent's action can have catastrophic consequences like a crash or medicinal side effects. These can lead to loss of life and property. Since these scenarios might be rare, the agent might not encounter them enough during training. But it has to be robust to such catastrophic scenarios before it can be deployed. In this work we train an RL agent which avoids catastrophes by using the Red Team approach described in Learning with Catastrophes [2] and Red Teams [3]. Core idea of this approach is to use another agent called the Red Team which tries to create adversarial or catastrophic scenarios for our agent so that it is more robust to such scenarios after training. Main motivation for this work is the belief that if you know your enemy and understand how it functions, then you can train yourself to perform better against it.

The goal of the red team is to train to produce adversarial inputs by exploiting loopholes in agent's policy. We expect that red team will learn to produce adversarial scenarios during training more frequently than the natural environment. We apply this approach to two toy environments: Windy Grid World and Toy Self Driving Car and compare our results with agents trained without red team.

2. RELATED WORK

Adversarial learning techniques have been applied in number of scenarios like intrusion detection, spam filtering and terrorism detection. All of these involve an adversary that tries to avoid detection. In [8] authors proposed adversarial classifier reverse engineering (ACRE) learning problem (task of constructing adversarial attacks by learning the details of the classifier) and applied it to the task of spam filtering for linear classifier with continuous and boolean features. This work assumes that adversary has the knowledge about opponent's model (in this case Linear Classifier). In our work, both agent and the adversary (red team) learn about each other through environmental interactions in a reinforcement learning setting.

Recently adversarial techniques have been applied for training Deep Learning networks for computer vision tasks. Goodfellow et al. [5] proposed a framework for estimating generative models via adversarial nets. They jointly train two simultaneous models: a generative model G and a discriminative model D which tries to identify which images were generated by G. They also suggest ways to extend this approach for semi supervised learning and learning approximate inference. Denton et al. [4] modified the approach in [5] to use a cascade of convolutional networks within a Laplacian pyramid framework. Main idea behind this approach is to make refinements to generated image at successive layers within the pyramid using a convnet model. At each level of the pyramid, a separate generative convnet model based on Generative Adversarial Nets (GAN) approach. Our approach is similar to [5] and [4] in the sense that both the agent and adversary (red team) are trained jointly but through reinforcement learning algorithm like Q Learning [16]. However unlike [5] and [4] in our case red team is not a generative adversarial network. It is a reinforcement learning agent (just like the normal agent) which learns through its experiences with environment.

Goodfellow et al. [6] present simple and fast methods for generating adversarial examples for machine learning models like neural networks and show that adversarial training can help with regularization (even more than dropout). They claim that main reason for vulnerability of neural networks to adversarial perturbation is their linear nature and justify it through quantitative results. Our work is similar in the sense that the red team is learning to exploit loop holes in the agent's policy and both are learning their policies through a neural network.

A lot of work has been done in the area of Reinforcement Learning techniques in a multi-agent, adversarial environment. Littman [7] explored the Markov Games framework in the context of multi-agent reinforcement learning setting. They describe a Q Learning [16] like algorithm based on the principle of MiniMax which allows agent to converge to a fixed, safe strategy. Our agent also needs to learn such safe strategies assuming worst possible adversary. However, for Markov Games discussed in [7] safe strategies can be computed from the payoff which is known before hand, but in our scenario catastrophic events and their payoffs are not known in advance. Infact they are very rare and have to be learnt through experience by interacting with the environment.

In [14] and [15] authors propose two new algorithms for multi-agent, adversarial environments. They propose an extension to Prioritized Sweeping [11] that allows generalization of learnt knowledge over neighboring states and an extension of U Tree generalizing algorithm [9] that can handle continuous states. Their work compares the performance of the new algorithms with previous techniques developed for Adversarial Reinforcement Learning in Markov Games on grid soccer domain. More recently [10] have demonstrated the effectiveness of using Deep Neural Networks with experience replay as function approximators which work well even for continuous and high dimensional state spaces. We use a simpler version of their architecture and some of the tricks like experience replay which were used by them.

Perhaps most closesly related to our work are the competitive coevolution approaces inspired from the success of genetic algorithms. In [12] Rosin and Belew proposed 3 new coevolutoin techniques and applied them to the games of Nim and 3-D Tic Tac Toe. Although these ideas are similar in spirit to our approach, it is not obvious how they can be extended to complicated environments with continuous state spaces like ours. Leveraging some of the key ideas from these approaches is a good future direction to extend the red team approach.

3. MOTIVATING DOMAIN

Various Grid World environments are often used for evaluating effectiveness of new reinforcement learning algorithms. Grid World environments are chosen because they are simple, easy to understand and computationally inexpensive to solve due to small state space. This section describes a toy grid world domain on which we tried the red team approach.

3.1 Windy Grid World

We created a grid world environment as shown in Figure 1. Agent starts in the START cell on bottom left. Its objective is to reach GOAL cell on top right. However there are some cliffs along its paths as shown by cells marked CLIFF in Figure 1. Apart from this there is a strong wind at each cell whose direction keeps changing arbitrarily. At each time step agent can act to move in one of the standard directions (up, down, left, right). However due to the presence of strong winds, it might move in direction of the wind with probability of 0.1, irrespective of what action it chooses. It receives a reward of -1 for each time step, a penalty of -100 for falling into the cliff and a reward of 100 for reaching the goal. While training with red team, the reward for red team is negative of the agent reward. Since movement under influence of wind is probabilistic and there is a high negative penalty for falling into the cliff, it is considered to be a catastrophic event for the agent. Thus we want it to learn a safe policy which navigates to goal without getting very near the cliff.

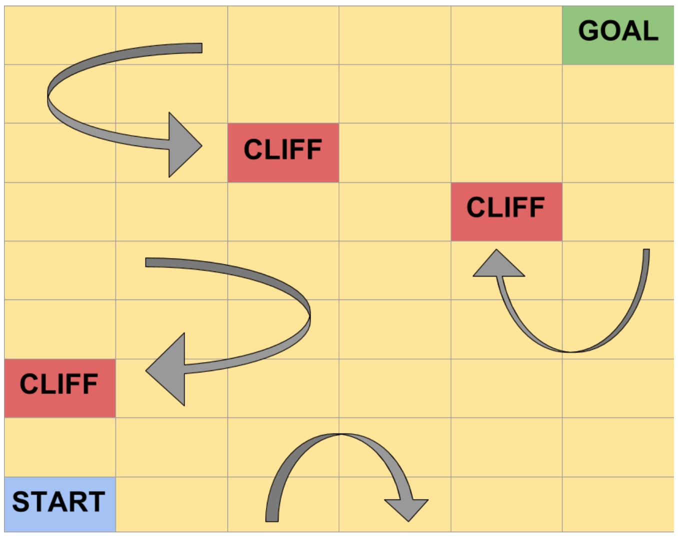

Figure 1: Windy Grid World Environment

3.2 Methodology

We applied Tabular TDLearning approaches SARSA and QLearning since they work well for small and discrete, grid world type environments. Our algorithm jointly trains the agent and the Red Team to learn the QValues for both at the same time as shown in Algorithm 1. We also implemented corresponding planning versions of TDLearning approaches which store the experience for simulating planning steps. Next section describes the results obtained by these approaches.

3.3 Results

In this section we compare the results of various TDLearning algorithms described above with the Red Team based algorithm. Figure 2 and 3 show the policies learnt by a QLearning agent with and without the red team after 100k iterations. We observe that the agent trained without red team (Figure 2) just learns to take the shortest path to goal which contains a cell next to the cliff. Although, this path is shortest, it is risky because the wind might cause the agent to fall in the cliff with a small probability. In contrast, the agent trained with red team(Figure 3) learns to go around several cliffs in its path. Although this is not the shortest path, this is a much safer policy since it never visits any cell which is directly adjacent to any of the cliffs. Figure 4 shows the policy learnt by Red Team. It shows that at most of the cells, Red Team learns to apply the wind in a direction that might push the agent towards the cliff or away from the goal. We believe that with more training, Red Team can learn even better policy, but the current policy serves the purpose of making agent learn to go around the cliff.

We also experimented with a SARSA agent. It also learns

Algorithm 1 Tabular TD-Learning with Red Team

```
1: procedure TDLearning(updateType)
2:     Initialize $Q(s,a)\ \forall s \in S,\ a \in A(s)$ arbitrarily and $Q(\text{Terminal-State}, .) \leftarrow 0$
3:     Initialize $RQ(s,a)\ \forall s \in S,\ a \in RA(s)$ arbitrarily and $RQ(\text{Terminal-State}, .) \leftarrow 0$
4:     for each episode do
5:         Initialize $S$
6:         for each step in the episode do
7:             $A \leftarrow \epsilon$-GreedyAction$(A(S))$
8:             $RA \leftarrow \epsilon$-GreedyAction$(RA(S))$
9:             $R, S' \leftarrow$ Act$(S, A, RA)$
10:            TDUpdate(Q,RQ,S,A,RA,R,S',updateType)
11:            $S = S'$
12:        end for
13:    end for
14: end procedure

15: function TDUpdate(Q, RQ, S, A, RA, R, S', updateType)
16:     if updateType = 'SARSA' then
17:         $A' \leftarrow \epsilon$-GreedyAction$(A(S'))$
18:         $RA' \leftarrow \epsilon$-GreedyAction$(RA(S'))$
19:         $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma\, Q(S',A') - Q(S,A)]$
20:         $RQ(S,RA) \leftarrow RQ(S,RA) + \alpha[-R + \gamma\, RQ(S',RA') - RQ(S,RA)]$
21:     else
22:         $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma\ max_a Q(S',a) - Q(S,A)]$
23:         $RQ(S,RA) \leftarrow RQ(S,RA) + \alpha[-R + \gamma\ max_{ra} RQ(S',ra) - RQ(S,RA)]$
24:     end if
25: end function
```

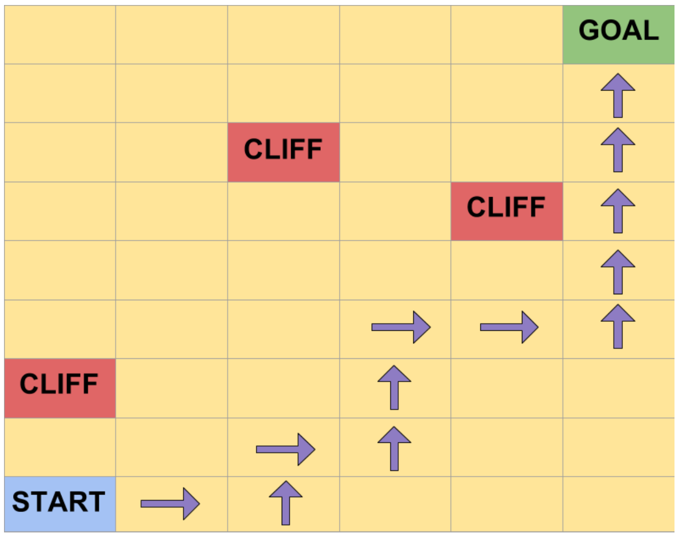

Figure 2: Agent Policy learnt with Tabular QLearning

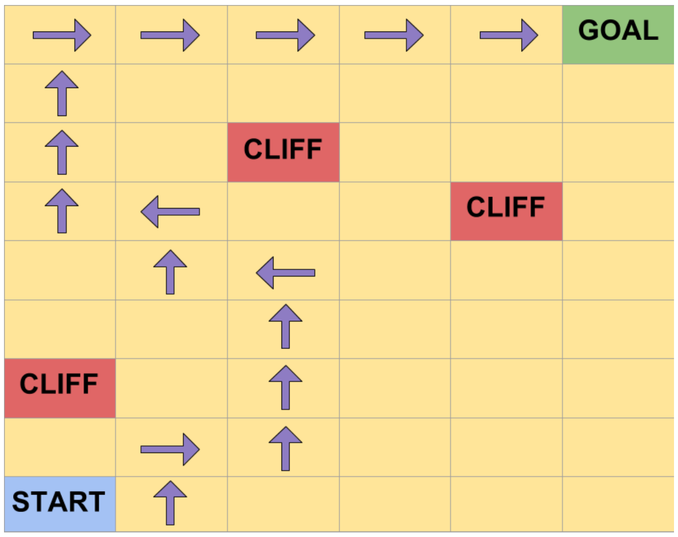

Figure 3: Agent policy learnt by Tabular QLearning with Red Team

to take the shortest path which passes by a cliff. Even the planning versions (10 planning steps) of QLearning and SARSA agents learn a similar policy which contains a cell adjoining a cliff. In the environment without the Red Team, direction of the wind is random and the probability of agent moving in the direction of the wind is 0.1 Thus the agent falls in cliff with 0.025 probability. Since the event of falling into the cliff is quite rare, even the simulated experience obtained through planning does not help much. These results demonstrate the effectiveness of Red Team in the toy domain of windy grid world.

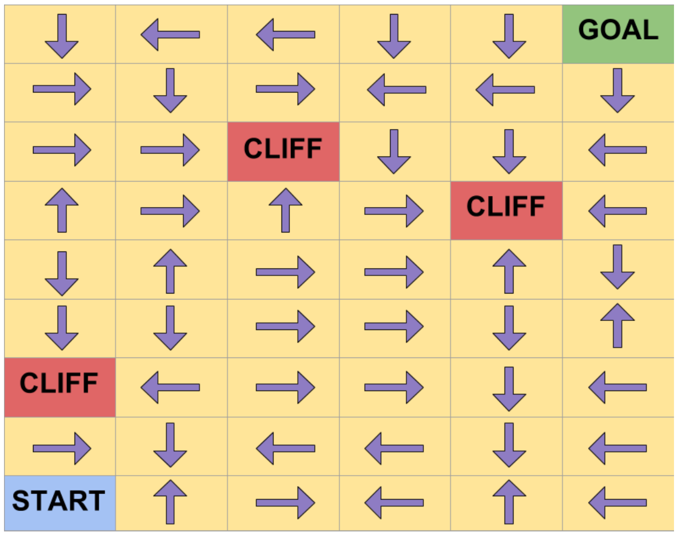

Figure 4: Policy learnt by Red Team for applying wind

4. SELF DRIVING CAR

The success of Red Team approach on windy grid world environment motivated us to apply it to a more complex environment with continuous state space. We decided to work with toy Self Driving Car environment. Motivation for choosing this environment was the fact that crashes are rare if the driving area is huge and has no obstacles. Thus ϵ epsilon greedy agents might not encounter many crashes during training. Thus it is a perfect setting for applying the Red Team approach. Rest of this section describes the problem, environment, approach and results.

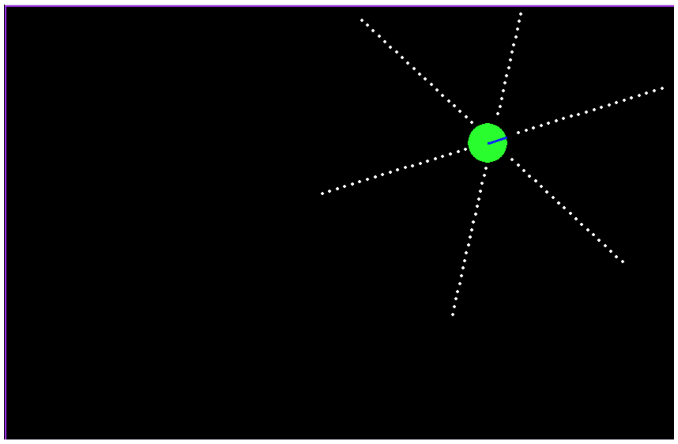

Figure 5: Training Environment

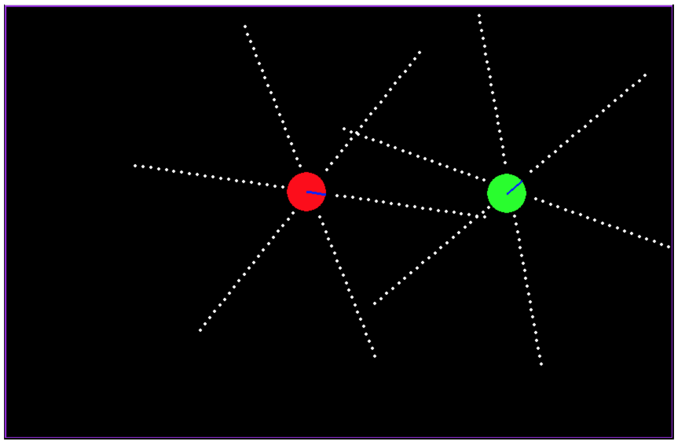

Figure 6: Training Environment with Red Team

4.1 Objective

Objective of our reinforcement learning agent is to navigate the green car in Figure 5) through the environment without colliding with any walls or obstacles. Objective of Red Team agent (red car in Figure 6) is to chase the toy car and collide with it.

4.2 Environment

Environment is built by modifying an existing implementation of the simulator in Pygame to adapt to our problem. We have different environments for training and test. Figures 5 and 6 show training environment for the agent with and without Red Team. Test environment consists of agent with 3 obstacles moving arbitrarily as shown in Figure 7. We describe the specific details of the environment below:

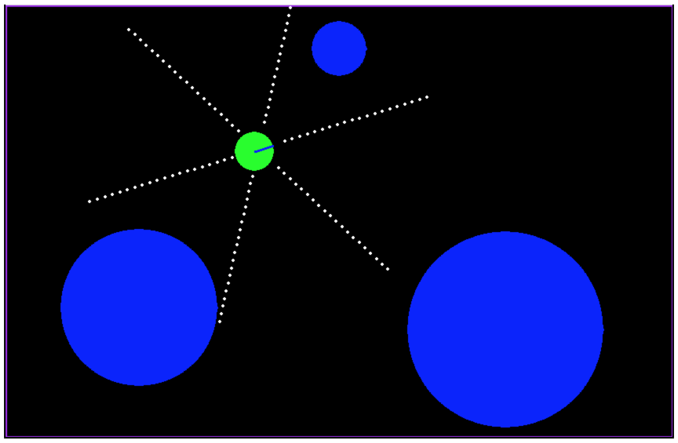

Figure 7: Test Environment

- **State Space (S):** Both the agent and the Red Team have 6 sensory arms along which they can measure the distance to obstacles. Thus it is a partially observable environment in the sense that both have limited field of view and can only measure distance, not the velocity of objects. Green agent's state consists of 6 distance readings along its sensory arms. Since Red Team needs to distinguish between obstacles like wall and the green agent, its state consists of 12 readings (6 for distances to agent and 6 for other obstacles) along 6 sensory arms.

- **Action Space:** Action space of both the agent and the Red Team consist of 3 discrete actions. At each time step they can either turn left or right by 0.2 radians or keep going straight.

- **Transition Function (T):** Transition function is a standard physics simulator (provided by Pygame) which considers all bodies as ideal and all collisions as elastic.

- **Reward (R):** Agent gets a reward of -50000 whenever it collides with a wall, obstacle or Red Team. Red Team gets a reward of $+50000$ for colliding with the agent and a small penalty of -500 for colliding with walls. Apart from reward/penalty on collision, both agents get a reward at each non-collision time step. We experiment with 2 different reward functions:

  Min Distance Reward In this setting agents get reward which encourages the agent to maximize the minimum distance along all its arms and the Red Team is encouraged to maximize the minimum distance to wall and minimize the maximum distance to agent. More concretely:

  $$\text{Agent Reward} = \text{Min(Sensor distances)}$$

  $$\begin{aligned}\text{Red Team Reward} = &\ \text{Min(Sensor distances to wall)}/50 \\ &- \text{Max(Sensor distances to agent)}\end{aligned}$$

  Sum Distance Reward This setting encourages the agent to maximize the sum of distances along all the sensor arms. Red Team is encouraged to maximize the sum of sensor distances to wall and minimize the sum of sensor distances to agent. More concretely:

$$\text{Agent Reward} = -5 + \text{Sum}(\text{Sensor distances})$$

$$\begin{aligned}\text{Red Team Reward} = {} & 0.01 * \text{Sum}(\text{Sensor distances to wall}) \\ & - 0.1 * \text{Sum}(\text{Sensor distances to agent}) \\ & - 5\end{aligned}$$

4.3 Methodology

Since our state space is continuous, we use a Deep Q Network for function approximation. Our architecture (Figure 8) consists of 2 DQN models (one for agent and one for the Red Team) which are trained jointly. Each model consists of 2 fully connected hidden layers. Input layer and 1st hidden layer are followed by Relu activations. Final layer gives the estimate of Q values of the three actions. For efficient learning we employ the trick of experience replay suggested in [10]. NOTE: Our architecture is very similar to the one described in this blog post.

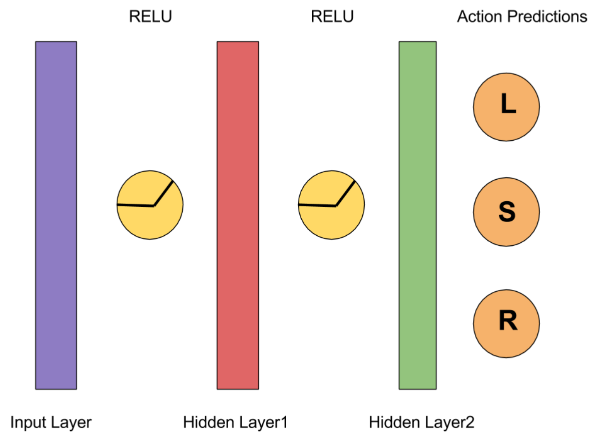

Figure 8: Neural Network Architecture for function approximation

4.4 Evaluation

4.4.1 *Baselines*

We compare our approach with following baselines:

- **Random Agent** Random agent is an agent which chooses random action at each time step.
- **QLearning Agent** This is an agent which uses DQN for function approximation using architecture described above, but is trained without Red Team.
- **QLearning Agent with Random Adversary** This agent uses DQN for function approximation and is trained with an adversarial agent which moves randomly.

4.4.2 *Metrics*

We compare the performance of different agents using the average number of steps per episode metric.

4.5 Experiments

This section presents various experiments which were performed to study the effectiveness of the proposed Red Team approach and discusses their results. We use the steps per episode metric defined in previous section for comparing the performance of different models. Demos of some of the agents described below can be found here

4.5.1 *Min Reward vs Sum Reward*

In this experiment we studied the effect of reward function. Min Reward encourages the agents to maximize the minimum distance along all sensor arms whereas Sum Reward encourages the agents to maximize the sum of distances along all sensor arms. Intuitively both of them are good heuristics for safe navigation. Figure shows the comparison of these reward functions for an agent trained with Red Team. Both the agents were evaluated on 100k steps. Agent trained with sum distance reward type performs slightly better than min distance (162 crashes vs 185 crashes). Hence we use sum distance reward for rest of the experiments.

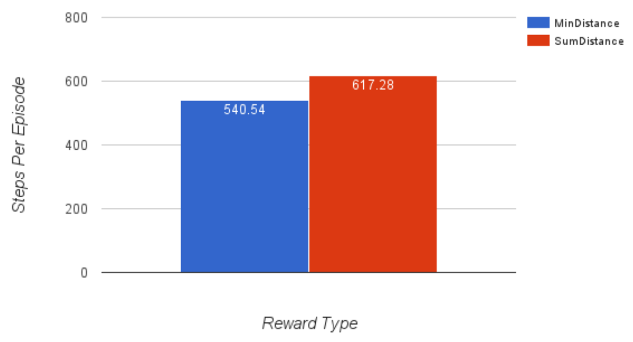

Figure 9: Comparison of Min Distance and Sum Distance reward types

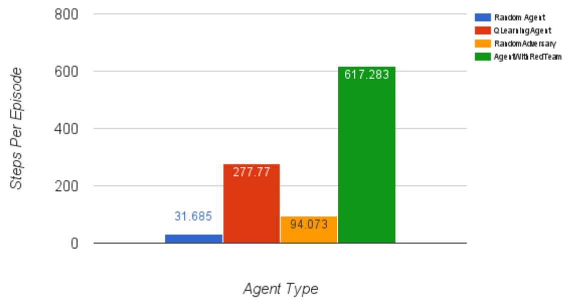

Figure 10: Comparison with baselines

4.5.2 *Comparison with baselines*

We compared the results of Red Team approach with the baseline approaches mentioned above. All models are trained by observing first 50k to build a experience replay buffer and then trained for 100k iterations. Best performing models are chosen for comparison. Figure 10 shows the results. As expected random agent (blue bar) performs the worst and agent trained with Red Team (green bar) performs the best. However, we observe that agent trained with an adversary which moves randomly (yellow bar) performs worse than agent trained without any adversary (red bar). A possible explanation for this could be that randomly moving adversary could be confusing the agent, making it learn a bad policy rather than helping it.

4.5.3 *Effect of screen size*

In this experiment we investigate the effect of reducing the screen size to 3/4th of original size while training. Motivation for this experiment was the fact that smaller screen area will increase the number of collisions and help both the Red Team and the agent learn faster. 11 shows the results. We observe that the agent trained in smaller screen performs barely better than a random agent. One possible explanation for this result could be that the agent does not see many of the states in small screen and hence is not able to generalize well to larger screen in test environment.

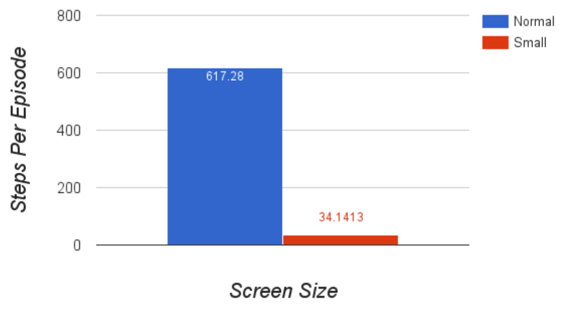

Figure 11: Performance of agents trained with different screen sizes

5. DISCUSSION & FUTURE WORK

In this section we discuss some of decisions about our network architecture, reward function, training details etc. First of all we decided to use sensory readings rather than using whole pixel image as raw input for our network. This reduces the dimensionality of the input and helps in efficient processing without much loss of information. We faced a number of issues while training models for self driving car environment. Designing a reward function for this task was challenging. If crash penalty is low, both the agent and red team just learn to keep moving in a circle. To solve this we increased the crash penalty for agent and the reward for Red Team. We also added a small penalty for Red Team if it collides with walls. Our network uses some of the tricks from DQN [10] paper but there are lots of other tricks which can be inculcated to improve the model. We can use frame stacking so that agent can sense velocity of obstacles and its own velocity. In our implementation we are using same network for prediction of Q values and updating the model weights. Like DQN architecture it would be better to have two separate identical networks for prediction and weight update, where updates are propagated after some delay. Currently agent can perceive only distance along 6 sensory arms. We can also experiment with more/less number of sensory arms. Another experiment would be to modify reward function to give higher weight for distance along forwards arms, since agent is moving in that direction. Another idea is to try training with multiple Red Team agents. This should help learn faster and better policy. Finally, we would want to make the task more realistic by requiring the agent to navigate source point to some predefined destination. In current setting, agents which keep moving in circle until obstacle approaches them perform better than agents which learn navigating around the environment.

6. CONCLUSION

We proposed an adversarial reinforcement learning approach called Red Team and applied it on 2 toy environments: windy grid world and self driving car. Our results demonstrate that Red Teams can be used for training robust RL agents for environments with sparse negative rewards and where agent mistakes can be catastrophic. Promising results of our work call for further exploration in this direction.

”If you know the enemy and know yourself, you need not fear the result of a hundred battles. If you know yourself but not the enemy, for every victory gained you will also suffer a defeat. If you know neither the enemy nor yourself, you will succumb in every battle.”
- Sun Tzu, The Art of War [13]

References

[1] Reinforcement Learning. `https://en.wikipedia.org/wiki/Reinforcement_learning`.

[2] Paul Christiano. Learning with Catastrophes. `https://medium.com/ai-control/learning-with-catastrophes-59387b55cc30#.kbjea8a57`, 2016.

[3] Paul Christiano. Red Teams. `https://medium.com/ai-control/red-teams-b5b6de33dc76#.qbxsb5ahk`, 2016.

[4] Emily L Denton, Soumith Chintala, Arthur Szlam, and Rob Fergus. Deep generative image models using a laplacian pyramid of adversarial networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1486–1494. Curran Associates, Inc., 2015. URL `http://papers.nips.cc/paper/5773-deep-generative-image-models-using-a-laplacian-pyrami` `pdf`.

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014. URL `http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf`.

[6] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[7] Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the eleventh international conference on machine learning*, volume 157, pages 157–163, 1994.

[8] Daniel Lowd and Christopher Meek. Adversarial learning. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 641–647. ACM, 2005.

[9] Andrew Kachites McCallum. *Reinforcement learning with selective perception and hidden state*. PhD thesis, University of Rochester, 1996.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[11] Andrew W Moore and Christopher G Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13(1):103–130, 1993.

[12] Christopher D Rosin and Richard K Belew. New methods for competitive coevolution. *Evolutionary Computation*, 5(1):1–29, 1997.

[13] Sun Tzu. 500 bc. *The art of war*, 1982.

[14] William Uther and Manuela Veloso. Adversarial reinforcement learning. Technical report, Technical report, Carnegie Mellon University, 1997. Unpublished, 1997.

[15] William TB Uther and Manuela M Veloso. Generalizing adversarial reinforcement learning. In *Proceedings of the AAAI Fall Symposium on Model Directed Autonomous Systems*, page 206. Citeseer, 1997.

[16] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.