Adversarial Reinforcement Learning Neural Networks

Robbie Sadre

December 11, 2017

Abstract

Deep Reinforcement Learning has become a hot research topic. Due to the innovations in hardware that have enabled the emergence of deep learning we have seen a boom in this industry [3]. Many experiments of applying Reinforcement Learning to games using DQNs [1] have been conducted in singular agent settings. What this work contributes is examples of methods of experimentally analyzing Reinforcement learning agents in Adversarial Settings, and also offers empirical evidence of the significance of conducting research in these settings through example of a popular Atari game: Pong. This research hopes to introduce a new perspective and inspire new research in adversarial Deep Reinforcement Learning Agents.

1 Introduction

Deep Learning Has become a very popular topic of research in the recent 5-10 years thanks to powerful innovations in parallelized processors for example GPUs. A common application for Deep Learning is in image classifiers. The inputs to the neural network are typically pixel values of the image, and the outputs are a probability distribution over all possible classes that an image can fall into. In supervised learning, images are paired up with labels in a set of training data that is already given and a process like stochastic gradient descent is used to minimize the loss between the output distributions and the targeted labels.

When Neural Networks are applied to Reinforcement Learning problems, the intuition for what the outputs mean is quite different. In a Deep Q Learning Network, the input to the neural network is the state of the environment, and the output is the action values of all possible actions that the agent can take. Additionally, the output of the neural network is not compared to some given label in a large training set. Rather, as the agent interacts with the environment, the states actions, rewards, and following states are recorded and used as training data to perform a loss minimization while the agent is acting in the environment [1] Deep Q Networks have become a very hot topic. A popular application amongst researchers is to have an agent learn an Atari game using the pixel data. Before DQNs Tabular Q learning was also used to learn simple gridworld games with a finite or small number of states so that a table of action values could be kept.

Another research field of reinforcement learning is the study of adversarial agents. This is essentially the studying of what occurs when two separate reinforcement learning agents are put in an environment together and have opposing goals. In [2] studies were done to analyze how different reinforcement learning agents compete with eachother in an adversarial setting in the form of a simple gridworld soccer game. In this experiment they pit different agents using different algorithms (Tabular) against eachother and analyzed which algorithms triumphed in adversarial settings.

Studies in Adversarial Agents in Reinforcement Learning have been primarily limited to the settings of a gridworld game that use tabular methods of creating action value functions. What this work contributes is a study on adversarial agents that utilize neural networks to approximate the action values of each state. This work focuses on the classic Atari game pong. This particular Atari game can be a two person game. The original deep mind paper [1] that uses this game for Deep Reinforcement Learning Research utilize a built in AI that makes decision based on the ball's location through a series of simple if statements and conditionals. However, what has yet to be assessed, is the results of training two separate reinforcement learning agents modeled by neural networks against eachother in this

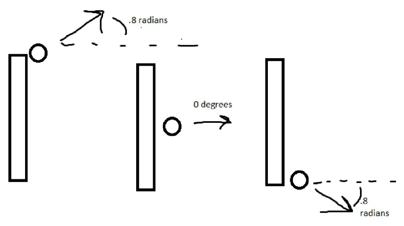

Figure 1: new direction of ball at three points of contact

game. This experiment is the novel contribution of this work. In particular, this work aims to identify and possibly quantify any benefits that result from training the reinforcement learning agent against another reinforcement learning agent rather than a predictable model that is based on if statements and simple conditionals.

2 System Model

The System model has various components including the environmental factors (game physics and parameters), Neural Network Architecture, Reinforcement Learning Algorithms, and Experimental set up.

2.1 Game Physics

The game was set up in python using the pygame API. The environment is a 320x320 pixel window. The rules follow the typical rules of the game pong. As far as rewards signals, these were awarded at three times: when an agent scores, a reward of +1 is signaled. When the opposing player scores on the agent, the agent receives a reward of -1. When the agent successfully hits/blocks the ball, a small reward of +.1 is awarded as well. The Ball is able to span across all parts of the window. When the ball surpasses one of the players (i.e when someone scores) , the ball's location is reset to the center of the screen, offset to the side of the player who was just scored on. Additionally, it's direction of motion at the beginning of the serve is determined randomly; however, its x direction is always opposite in magnitude from when it landed in the goal, simulating a "serve" from the loser of that round. Additionally, when the ball hits the top of bottom of the screen, it's y component of velocity reverses in magnitude, which mimics the way a ball bounces off of a wall. When a ball bounces off a paddle the new direction of the ball is calculated based on where it collides with the paddle. If it collides with the center, it will redirect to move at 0 degrees. At the ends of the paddle, a collision will cause the ball to redirect at an angle of .8 radians above or below the normal line to the paddle. the figure below demonstrates this reflection. anything in between these locations the direction is decided by linearly transforming the location of contact into an angle. This is demonstrated in fig 1 It is a significant part of the physics of the game because it gives the agent the ability to choose where to hit the ball based on the state of the game.

2.2 Neural Network Architecture

All agents trained in this experiment used the same neural network architecture. The inputs to the neural network are factors that define the state of the game, namely: the y position of the agent's paddle, the y position of the adversary's paddle, the x and y positions of the ball, and the x and y speeds of the ball. These are all that are necessary to learn a policy to interact with this environment, as they successfully capture and summarize the state of the game. They are the only things in the environment that change at any point in time. The neural network model is a fully connected neural network with four hidden layers. There are 3 outputs, one for every action available to the agent to move the paddle (up, down, don't move) and each output represents an action value for that corresponding action.

2.3 Reinforcement Learning Algorithm

The Training algorithm is very similar to that of [1], with the main difference being that pixels are not used as inputs to the neural network and therefore do not represent the state of the environment in this work. In traditional Q learning, a action value is adjusted using the bellman error. However, in DQN's it is quite different. The algorithm used to train an individual agent in this work follows the following logic:

- STATE = (BALL LOCATION (X,Y), BALL VELOCOTY (X,Y), PADDLE1 LOCATION, PADDLE2 LOCATION)
- Input STATE into Neural Network (NN)
- Get Action Values from NN
- 3/100 times take random action, else take action of highest value, thus alters the state.
- Rt = rewards that result
- NEWSTATE = (BALL LOCATION (X,Y), BALL VELOCOTY (X,Y), PADDLE1 LOCATION, PADDLE2 LOCATION)
- append [STATE,ACTION,REWARD, NEWSTATE] to agent's training data
- sample random 64 values from training data and perform stochastic gradient descent on loss function.

The loss function of a Q learning Neural network is more complicated than that of a regular supervised learning algorithm. for a given data point in the training data [STATE,ACTION,REWARD, NEWSTATE] The loss function is the difference between Q(STATE,ACTION) and $REWARD+ \gamma * ARGMAX_a[Q(NEWSTATE, A)]$. This requires using the Network to calculate the Q values for the NEWSTATE data point, getting the maximum value in the output, multiplying it by GAMMA and then adding R. How we have a target value for the loss function. This in a sense acts as a targeted value (much like a label in a supervised learning algorithm) for Q(STATE,ACTION), which is also the output of the neural network.

2.4 Experimental Set-up

The experiment set up is quite simple. We have two separate training sessions. The first session involves training a single agent (right side) against a simple model of an opponent controlled through if statements. The second session involves the training of two adversarial agents against each other each with no prior knowledge before hand. The final test comes down to competing the right side player of the first session (the agent) against the left side player of the second session (one of two agents in this session). Then it is to be observed which defeats the other in multiple trials of the game.

3 Results

. In session 1, a single reinforcement learning agent controlling the right side paddle plays and learns against a simple model of an opponent using the left paddle that is controlled through if statements and conditionals. Many tests were run to verify when the agent's policy stabilizes. This was identified by the lack of change in reward over time (See Numerical Results). The agent was trained for a total of 500000 time steps and the model was then saved.

In session 2 two reinforcement learning agents each controlling opposite paddles learn and play each other. Tests were also run on this scenario initially to identify when stability occurred. For this situation, assessing the policy stability on rewards through time would not have worked due to the fact that each agent in theory could be equal in skill at each point in time. Keeping track of rewards showed large oscillation through time which made it difficult to confirm policy stabilization. Rather than base it off reward, a somewhat odd method was concocted to identify when stability had occurred. In this game, a single play consists of all the time steps in between the time when the ball is served, to when it lands in the goal of one of the agents' paddles. This may also be defined as an episode in the simulation. One thing visually observable, is that the better the agents got at playing the game, the longer the episodes would last, because the players would be more likely to hit the ball rather than miss it as their policies became better. So in order to find when the policies reached their peak, the number of episodes every 5000 time steps was recorded and observed over time. As predicted, the number of episodes per 5000 time steps decreases over time and eventually stagnates, which should also note the peak performance of the agents. It was found that the policies stagnate well before 500000 time step, which is the number of time steps these two agents were trained through. The models were then saved.

The final part of the experiment is the easiest. The two models from the separate sessions compete in multiple games of 21 (first player to make it to 21 points wins). The results were very interesting. The Agent trained in the adversarial environment (session 2) had an enormous advantage over the other agent. It won all 50 games by a land slide each time.

3.1 Analysis of results

A big question is now posed: why did the agent trained in an adversarial environment have significantly better performance than the other? One speculation involves state coverage. The session 1 simple opponent modeled by if statements followed the same rules throughout the game. It essentially just followed the motion of the ball. Because the location of this paddle was part of the state input into the agent, the agent likely, in a sense, became used to the predictable motions of the paddle, and even relied on this predictable nature to function properly since its location is one of its state inputs. Due to the random nature of the second sessions two players,each of those agents, were likely to have seen many more states through their experiences, especially since their policies were not only somewhat random during training, but also that they changed over time. Another way of looking at it is that the sporadic nature of the session 2 agent looked like random noise to the input of the session 1 agent. This is a similar concept to adversarial examples in supervised image classifiers [3].

4 Numerical Results

The different agents all had very different rewards over time. Figure 2 shows the rewards over time for the single agent environmental model. As can be seen from figures 3 and 4 it is difficult to analyze the reward signals for the multi agent environment models. They tend to oscillate very largely as each agent adjusts its policy. Their reward signals are at odds with each other. Figure 5 demonstrates the stabilization of the two agents in the multi agent environments. In the beginning of the training session, the number of episodes is very high due to each ball consistently missing the paddle. As the agents improve their performances however, they are able to keep the ball in play for longer. Finally figure 6 is a diagram showing the final scores of each player in the final test. It is evident that the multi agent trained model is significantly better in terms of performance in an adversarial environment.

5 Conclusion

Adversarial Reinforcement Learning Agents may play a significant role in the future. As Artificial intelligence becomes more common place, it will be more likely that the ability for different agents to interact with each other becomes necessary. This study has demonstrated methods of testing and analyzing the performance of adversarial deep reinforcement learning agents. These experiments can be extended and applied to the other deep reinforcement learning algorithms as well. In the case of the game Pong, it is evident that an adversarial reinforcement learning agent can serve as a better opponent for the agent that a predictable and simple model of an opponent controlled by conditionals. While this is a specific case, further inspection and research may show that for other situations training one agent against another reinforcement learning agent could prove to be a very useful method of creating robust action value functions using neural networks.

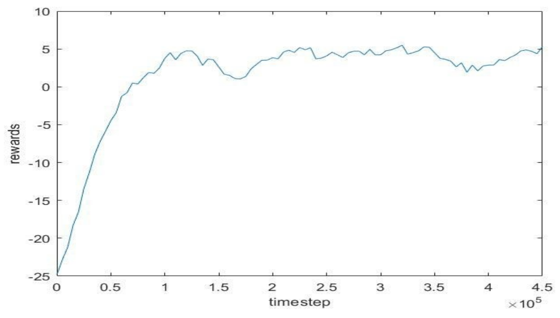

Figure 2

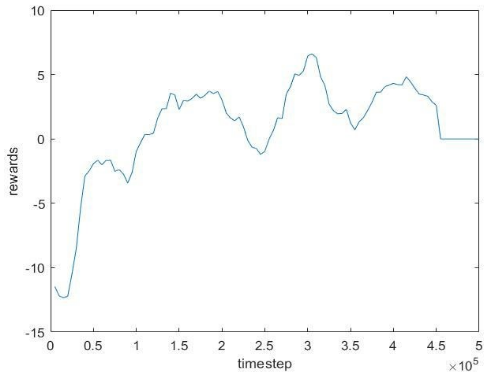

Figure 3

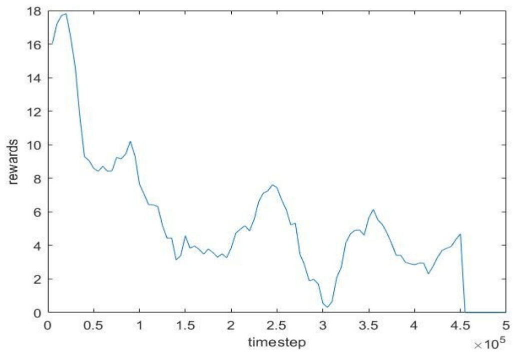

Figure 4

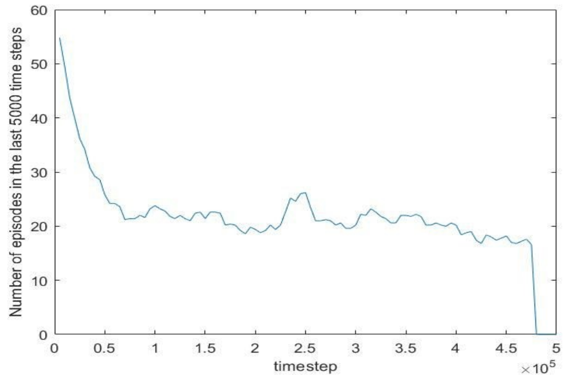

Figure 5

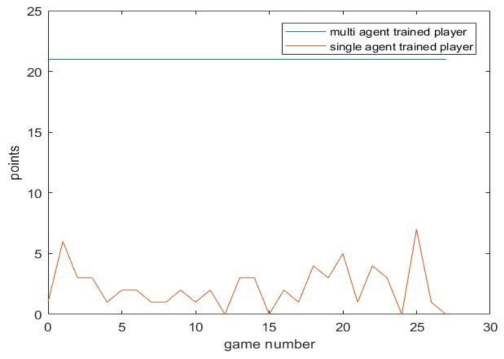

Figure 6

References

[1] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (Dec 2013). Playing Atari with deep reinforcement learning. Technical Report arXiv:1312.5602 [cs.LG], Deepmind Technologies.

[2] P. Stone, M. Veloso, Towards collaborative and adversarial learning: A case study in robotic soccer, Internat. J. Human-Computer Studies 48 (1) (1998) 83–104.

[3] J. Schmidhuber. Deep learning in neural networks: An overview. Neural Networks, 61: 85–117, 2015.

[4] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. Tygar. Adversarial machine learning. In Proceedings of the 4th ACM workshop on Security and artificial intelligence, pages 43–58. ACM, 2011.

[5]Bus¸oniu, L., Babuˇska, R., and De Schutter, B. (2008a). A comprehensive survey of multi-agent reinforcement learning. IEEE Transactions on Systems, Man, and Cybernetics. Part C: Applications and Reviews, 38(2):156–172.