Online Policy-Gradient Reinforcement Learning using OLGARB for SpaceWar

Jason Fager JFAGER@OU.EDU
University of Oklahoma, 660 Parrington Oval, Norman, OK 73019 USA

Abstract

The goal of this project is to explore the use of reinforcement learning techniques to address a difficult problem domain. The SpaceWar task is a competitive, continuous-valued, partially observable problem domain that provides a fun and interesting challenge for machine learning algorithms. This project focuses on the application of online policy-gradient reinforcement learning techniques to train a neural net controller for a ship. This paper shows that the OLGARB algorithm is effective for learning obstacle avoidance strategies for SpaceWar, and compares results across several values of the main tuning parameter for the algorithm.

1. Introduction

The goal of this project is to explore the use of reinforcement learning techniques to address a difficult problem domain.

1.1. The SpaceWar Problem Domain

The SpaceWar problem domain considered for this project takes its inspiration from the classic game of the same name written by Stephen Russell at MIT in the early 1960s[1]. The original game is simple: a single star exerts a gravitational force on two ships in a toroidal space. A player can rotate their ship to the right or left, fire their ship's thruster, or jump into hyperspace with the goal of avoiding any collisions, or shoot a bullet with the goal of hitting their opponent. The game continues until one of the ships is destroyed.

The version of SpaceWar used in this project is modified from the original version. Instead of a central sun exerting a gravitational force, the environment contains some number of large moving obstacles, and gravity is neglected. In addition, there can be more than two ships in a game. The amount of fuel and bullets available to each ship are limited, and regenerate at a rate that prevents their constant use.

SpaceWar is an interesting arena for machine learning algorithms for several reasons. It tests an agent's ability to cope with agent vs. environment scenarios as well as agent vs. agent scenarios. It requires the agent to deal with an infinitely variable state space over continuous-valued features with a limited and discrete control set (there are 13 reasonable combinations of commands). The most successful agents will be able to avoid collisions with obstacles and bullets as well as be able to shoot opponents, separate but related tasks that are often in conflict with each other. In other words, SpaceWar is hard, and it is similar to real-world learning tasks.

1.2. Policy-Gradient Reinforcement Learning

The SpaceWar domain can be described as a reinforcement learning domain: an agent takes actions in the space environment and is given punishment as its health is decremented for collisions and reward as its opponents are blasted into oblivion. An agent that is able to develop a policy that maximizes these rewards and minimizes these penalties will surely be able to win games.

Traditional reinforcement learning techniques typically attempt to model the underlying value function covering the state space, and allow the agent to map states to actions based on an estimation of this value function for each possible action. Q-learning and Sarsa are examples of reinforcement learning algorithms that work by estimating the underlying value function (Sutton & Barto, 1998)

(Sutton et al., 1999) point out some of the drawbacks of value function estimation. Most implementations lead to deterministic policies even when the optimal policy is stochastic, meaning that probabilistic action

[1]For a full history of the original SpaceWar, see http://www3.sympatico.ca/maury/games/space/spacewar.html

policies are ignored even when they would produce superior performance. Further, because these methods make distinctions between policy actions based on arbitrarily small value differences, tiny changes in the estimated value function can have disproportionately large effects on the policy.

An alternative method for reinforcement learning that bypasses these limitations is a policy-gradient approach. In this approach, instead of learning an approximation of the underlying value function and basing the policy on the expected reward indicated by that function, policy-gradient learning algorithms maximize the long-term expected reward by searching the policy space directly. In addition to being able to express stochastic optimal policies and being robust to small changes in the approximation, under certain conditions policy gradient algorithms are guaranteed to converge to an optimal solution. (Sutton et al., 1999), (Baxter & Bartlett, 2001).

2. The OLGARB Algorithm

The constraints of SpaceWar require an algorithm that can deal with a partially observable, continuous-valued state space and a discrete-valued action space. (Baxter & Bartlett, 2001) details the GPOMDP algorithm, an offline learner that computes a biased estimate $\nabla\eta_\beta(\theta)$ of the gradient of the average reward $\nabla\eta(\theta)$ for a policy (encoded in θ) in a partially observable Markov decision process. Baxter shows that under certain conditions,

$$\nabla\eta(\theta) = lim_{\beta\to 1}\nabla\eta_\beta(\theta)$$

where $\beta \in [0, 1]$ indicates a tradeoff between bias and variance. That is, lower values of β introduce higher bias to the estimate but higher values of β introduce higher variance. Experimental results show that this method can learn optimal control policies that elude standard value-function approximation techniques (Baxter et al., 2001).

OLPOMDP, given by (Baxter et al., 2001), is the online version of GPOMDP. While Baxter's experimental results with OLPOMDP are less encouraging than those with GPOMDP due to slow convergence times, (Bartlett & J.Baxter, 2000) show that the estimates $\nabla\eta_\beta(\theta)$ given by OLPOMDP will converge to $\nabla\eta(\theta)$ in a manner similar to GPOMDP.

2.1. The OLGARB Algorithm

The main drawback to both the GPOMDP and OLPOMDP algorithms is that they produce biased estimates of the policy gradient. This bias can be reduced by taking the β term arbitrarily close to 1; however, as β approaches 1, the variance of the estimation increases.

(Weaver & Tao, 2001) attempts to reduce the variance caused by increasing β values by introducing a baseline term that is the moving average of the reward. This baseline value directly affects the learning rate of the algorithm by scaling the contribution of the latest reward based on its relation to the average reward observed. If the new reward is dramatically different than the average reward, then the calculated gradient carries greater weight in the update of the policy parameters. On the other hand, when the observed reward is close to the average reward, the calculated gradient carries a smaller weight in the update of the policy parameters. Using this adaptive scaling should allow the increased variance typically associated with high β values to be somewhat mitigated, while preserving the benefit of decreased bias.

(Weaver & Tao, 2001) show that the use of this baseline term does in fact decrease variance for high β values in their GARB (GPOMDP with an Average Reward Baseline) algorithm, allowing the bias to be reduced without as significant a penalty in variance. They also show experimental results for OLGARB, their adaptation of OLPOMDP, that suggest a similar decrease in variance for high β values in the online algorithm. OLGARB is the algorithm selected for this project implementation:

At each time step:

1. Observe state X_t
2. Probabilistically select action A_t according to $\mu(\cdot|X_t, \theta_t)$
3. Apply A_t and observe reward R_t
4. Update variables:
   $B = B + \frac{R_t - B}{t}$
   $Z = \beta Z + \frac{\nabla_\theta \mu}{\mu}(A_t|X_t, \theta')$
   $\theta = \theta + \gamma(R_t - B)Z$

In this notation, μ refers to the agent policy, θ refers to the parameters that determine the policy, B is the baseline value corresponding to the average reward (introduced in OLGARB), Z is a vector describing the gradient direction of the parameter update, β is the bias-variance tradeoff value, and γ is the learning rate.

3. Experiments

The current work applies an implementation of OLGARB to a specific scenario in the general SpaceWar

task. The agent learns in an environment consisting of two ships and two obstacles in an 800 unit by 600 unit space, with a physics engine that models a frictionless environment. The engine updates the space every 1/30th of a second.

Each ship fires at random with a probability of 0.8, with a maximum of 5 bullets available to each ship at any given time. The ships are initialized with 15 units of health and 1000 units of fuel. Ships have a radius of 10 units and a relatively small mass.

Each obstacle is initialized with a random velocity. Obstacles have a radius of 50 units and a relatively huge mass.

Damage is dealt proportionately to the magnitude the difference in velocity before and after any collision between two ships or between a ship and an obstacle. If a ship is hit with a bullet, one damage unit is dealt.

3.1. Implementation Details

For the scenario described above, OLGARB is implemented for a neural network with 42 input nodes, 30 hidden nodes, and 6 output nodes. The input nodes use an egocentric coordinate system for the agent, and consist of the relative position and velocity vectors and current orientation towards each other object (opponents, bullets, and obstacles). When the opponent ship has not fired some of its alloted bullets, those bullets are assigned the same parameter values as the ship itself. The additional two input parameters give the agent's health and fuel.

The output nodes correspond to a subset of the reasonable actions an agent can take. A node corresponds to each of: 'do nothing', 'turn left', 'turn right', 'fire thruster', 'turn left and fire thruster', and 'turn right and fire thruster'. The agent is simplified by removing the options of jumping into hyperspace and firing a bullet.

When selecting the action to take, the output values are run through the softmax function

$$\frac{e^{o_i}}{\sum_j e^{o_j}}$$

and are interpreted as probabilities.

Updates to the network occur in a manner very similar to standard backpropagation. Since the objective target values are not know at any update step, the error term is calculated by making the assumption that the selected action should always be selected and that the unselected actions should never be selected. The reward given for the resulting state is used to influence whether and to what degree this assumption should be reinforced. The update procedure is almost identical to that given in (El-Fakdi et al., 2005), which implements OLPOMDP. Minor changes were made to incorporate the average baseline reward term used in the OLGARB algorithm.

3.2. Results

Performance is evaluated in two ways: first, the duration in timesteps is measured during online training sessions. A typical session between two random player will last approximately 500 timesteps, while a session between two players hardcoded to always 'do nothing' will last approximately 2500 timesteps (The 'do nothing' player is strong because the space is so sparsely populated). Second, the win percentage of the trained agent is evaluated against a 'do nothing' opponent. Random's win percentage against this heuristic is less than 5%.

Figure 1 illustrates the game duration for 20 agents trained from zero knowledge to 10,000 trials with a β of 0.8. Performance quickly climbs to around 1400 time steps, significantly better than random, and gradually improves, reaching an average of around 1900 timesteps at the end of 10,000 iterations. Figure 2 shows the performance of these agents against a do nothing opponent, with win percentage hovering around 30%, again significantly better than random. However, this performance seems to level off early, and little improvement is seen over time.

Figure 3 illustrates the game duration for 20 agents trained from zero knowledge to 10,000 trials with a β of 0.9999. Performance quickly climbs to around 1500 time steps, significantly better than random and the lower β value, and steadily improves, reaching an average of around 2800 timesteps at the end of 10,000 iterations. Figure 2 shows the performance of these agents against a do nothing opponent, with win percentage climbing to around 40%, again significantly better than random and the lower β value. Additionally, performance in both metrics seems to be increasing even at the end of the trial run.

Qualitatively, the observed behavior of some of the best agents from the training runs is impressive. The agent learns that a velocity with a high magnitude is dangerous, and that it can steer to avoid obstacles both when it is right next to the object and when it is approaching the object from a distance. This is compared to a random agent, which tends to reach high magnitude velocities and die in violent collisions, and 'do nothing' agents which survive for long periods of time until another object just happens to collide,

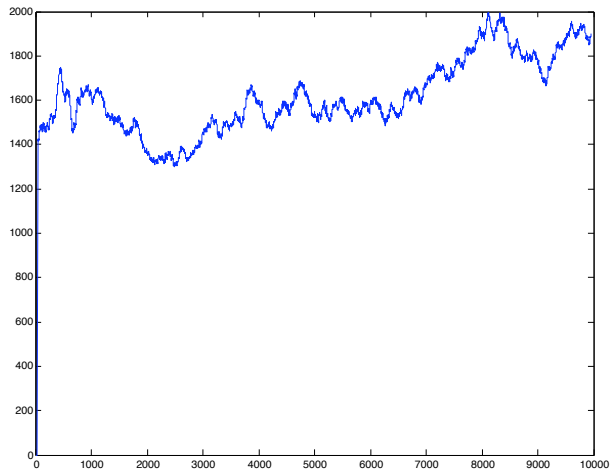

Figure 1. $\beta = 0.8$, Game duration (in number of timesteps) during training, plotted as the average performance of 20 separate agents trained over the course of 10,000 trials.

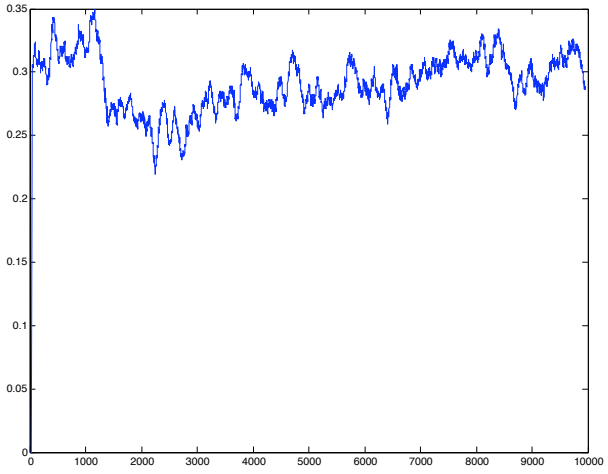

Figure 2. $\beta = 0.8$, Winning percentage against a strong heuristic opponent ('do nothing' opponent), plotted as the average performance of 20 separate agents trained over the course of 10,000 trials.

and then quickly dies as its motion is a liability. While the learning agents aren't prefect, they are obviously displaying desirable behavior.

4. Future Work

I would like to run the experiments again over a larger number of β and γ values in an effort to optimize agent performance as much as possible. I would also like to see how the agent responds to more detailed input information, as well as to different neural net architectures and lengthier trial runs. Finally, I would like to reintroduce the full range of possible actions to see if the agent is able to learn to attack the other agent.

Additionally, a collaborative effort is being planned to compare the performance of this OLGARB implementation with an implementation that uses a value-function estimator on an identical task. Another good point of comparison would be with a standard OLPOMDP implementation, to see the effects on performance that are attributable to the baseline function.

5. Conclusion

This paper introduced the SpaceWar problem domain and showed that the OLGARB algorithm is effective at producing good policies for obstacle avoidance in an environment with two ships and two obstacles. The effects of the varying β parameter were shown, with good performance achieved at a high β value. This paper should provide encouragement that further investigation of online policy-gradient reinforcement learning techniques for difficult problem domains would be valuable.

References

Bartlett, P. L., & J.Baxter (2000). Stochastic optimization of controlled partially observable markov decision processes. *Proceedings of the IEEE Conference on Decision and Control* (pp. 124–129).

Baxter, J., & Bartlett, P. L. (2001). Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, *15*, 319–350.

Baxter, J., Bartlett, P. L., & Weaver, L. (2001). Experiments with infinite-horizon, policy-gradient estimation. *Journal of Artificial Intelligence Research*, *15*, 351–381.

El-Fakdi, A., Carreras, M., & Ridao, P. (2005). Direct gradient-based reinforcement learning for robot behavior learning. *ICINCO* (pp. 225–231).

Sutton, R., McAllester, D., Singh, S., & Mansour, Y. (1999). *Policy gradient methods for reinforcement learning with function approximation* (Technical Report). ATT Labs.

Sutton, R. S., & Barto, A. G. (1998). Reinforcement learning: An introduction. MIT Press: Cambridge, MA.

Weaver, L., & Tao, N. (2001). The optimal reward baseline for gradient-based reinforcement learning. *Uncertainty in Artificial Intelligence: Proceedings of the Seventeenth Conference* (pp. 538–545).

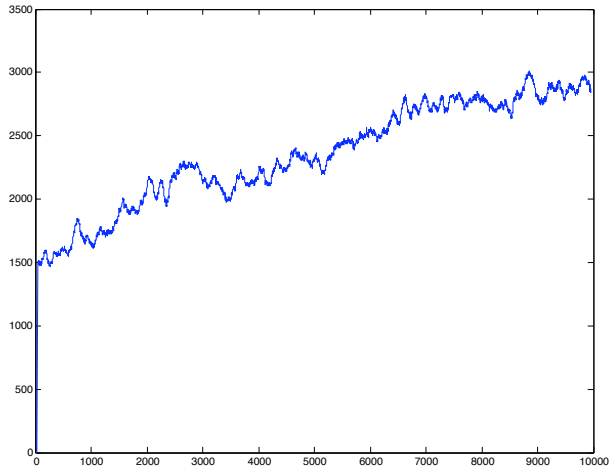

Figure 3. $\beta = 0.9999$, Game duration (in number of timesteps) during training, plotted as the average performance of 20 separate agents trained over the course of 10,000 trials.

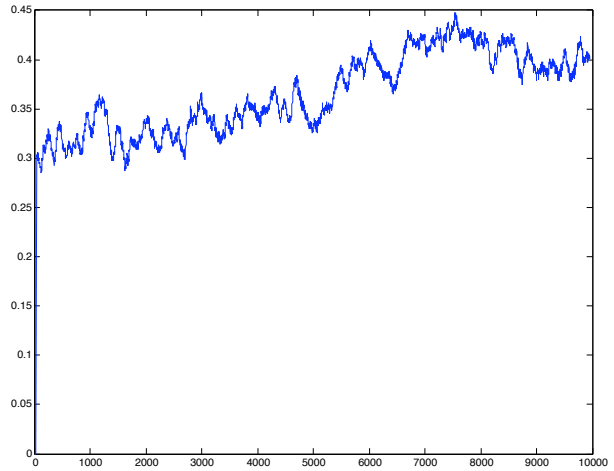

Figure 4. $\beta = 0.9999$, Winning percentage against a strong heuristic opponent ('do nothing' opponent), plotted as the average performance of 20 separate agents trained over the course of 10,000 trials.