Project 4: Reinforcement Learning

Sam Beckmann

April 28, 2017

1 Introduction

Project 4 involved building and testing reinforcement learners to play the game of tic-tac-toe. I programmed two different reinforcement learners: An off-policy Q-Learning agent using Watkin's algorithm, and an on-policy SARSA agent. For each of the learners, I implemented them in two unique ways: First as written, then as a λ version, using eligibility traces to back-propagate the rewards with fewer iterations. Note that the non-λ version of the learners is just a special case where $\lambda = 0$.

In order to validate and test the learning of the agents, three "benchmark" tic-tac-toe players were used:

1. A deterministic minimax player. Using the minimax algorithm, this player is optimal. In the event of multiple, equally-optimal moves, this agent would deterministically select its move.

2. A non-deterministic minimax player. Like the first agent, but would randomly choose between multiple equally-optimal moves.

3. A random player. This player always moves completely randomly.

In addition to varying the opponents of the learners, the probability with which an intermediate reward was given was also varied. In the special case of 0, all information to the learner came from the environment indicating to the learner the final state of the game. When intermediate rewards were used, a positive reward was given if the agent made the same move as the deterministic minimax player would have in the same board state. If any other move was made, a negative reward would be given.

2 Implementation

In order to provide a consistent framework and interface for learners to adhere to, I rewrote and re-factored the existing Java code. Although I believe the current iteration is much more understandable than the previously existing code, I acknowledge that some of the design decisions may not provide the simplest

experience for building reinforcement learners. I plan to publicly release the tic-tac-toe framework for the use of future students after the framework has undergone more extensive peer-review in the coming weeks. I hope the accurate documentation and interfaces can help future students more easily learn how reinforcement agents work.

All of the reinforcement learners used an ϵ-greedy approach with a decreasing ϵ. ϵ_0 was initialized to 1, and any ϵ_t used in iteration t was calculated according to the equation:

$$\epsilon_t = \epsilon_{t-1} \cdot u$$

Where u was the update value for the trial. In practice, u varied from 0.001 to 0.999 under different conditions. This update method ensured the theoretical guarantee of convergence, while reducing exploration in favor of exploitation as time went on.

3 Testing

In order to test the reinforcement learners, they were pitted against different players with varying probability of intermediate rewards. Specifically, each of the four reinforcement learners, Q, SARSA, Q(λ), and SARSA(λ), were tested in the following configurations:

- Vs random player, intermediate reward probability = 0
- Vs deterministic minimax player, intermediate reward probability = 0
- Vs deterministic minimax player, intermediate reward probability = 1
- Vs random minimax player, intermediate reward probability = 0.5
- Vs random minimax player, intermediate reward probability = 0.75

In addition, both the Q and SARSA learners were tested against each other, with intermediate reward probability of 0, to test the robustness of the learners when compared to each other.

For all trials, parameters of $\lambda = 0.9$ and $\gamma = 0.9$ were used. α_n was calculated as $\frac{1}{timesVisited(n)}$ where $timesVisited(n)$ represents the number of times the current state-action pair had been explored at time n.

All trials were run until convergence. Convergence is considered to be 100 consecutive iterations which all result in a tie between the two players, except in the case against the random player, in which convergence was considered to be when the random player did not win in 100 consecutive matches. The update value u was varied per trial, and will be reported in the results. The u value which resulted in fastest convergence was used. In the event of a tie in terms of convergence times, the smallest u value that resulted in that convergence speed was used.

	Random (prob = 0)		Minimax (prob = 0)		Minimax (prob = 1)		R. Minimax (prob = 0.5)		R. Minimax (prob = 0.75)	
	u	Iters.	u	Iters.	u	Iters.	u	Iters.	u	Iters.
Q	0.9999	20,000	0.001	4	0.001	4	0.001	4	0.001	4
SARSA	0.999	2,700	0.001	4	0.001	4	0.1	1,500	0.1	1,500
Q(λ)	0.9999	3,000	0.99	2,000	0.99	2,000	0.99	2,000	0.999	2,500
SARSA(λ)	0.999	3,200	0.99	600	0.99	600	0.999	600	0.999	600

Table 1: Results

4 Results

The results of the trials are tabulated in Table 1. For each trial, the u value used and the number of iterations required to reach convergence were reported. In general, the random opponent took much larger to develop an effective policy far. This is likely due to the increased size of the effective state space. Against optimal players, the learners do not need consider a policy for non-optimal moves. Against certain opponents, Q and SARSA found a policy that guaranteed a tie within 4 iterations, and could play against the optimal opponent from that point forward.

The self-trials, not reported on in the table, both converged in less than 1,000 iterations with update rules of 0.99 and 0.999 for Q and SARSA, respectively. These trials were interesting, because there was no optimal or random agent on the other side. Both players simultaneously learned how to play the game without intermediate feedback, and eventually could always tie the other player.

5 Discussion & Conclusions

I was surprised the λ versions of the algorithms converged slower than the respective non-λ versions. I theorize this is because the λ versions result in a more in-depth policy for the state-space, as opposed to only going down the one needed path to tie an optimal agent.

Although there are no statistics presented on it in the results section, I would also like to make a brief note on the speed of the various learners, gathered qualitatively from my experience during testing. In general, the non-λ learners were faster than the λ learners, and Q-Learners were faster than SARSA learners. All of them, however, were much faster than the minimax algorithm, showing one of the benefits have using a reinforcement learner when compared to exhaustive state-space exploration. The update rule for Q scales linearly with the number of possible actions, which makes it incredibly fast. Of course, the trade-off lies in the high memory requirement to keep track of information for each state-action pair in the world. In general, I found it interesting to see how the learners could learn how to play each other without any concept of the "game" they were learning to master.