Assignment 1: Stochastic Game Engine

Drew Wicke

March 4, 2013

Approach

Using Lisp, I created a stochastic normal form game engine for n-players using mixed strategies and with variable number of actions possible in each game. The engine requires the user to specify a config file which details a stochastic game which includes rewards and transition probabilities, number of iterations and each agent's properties. This makes it easy to create new games and configurations of players. To test my implementation I created game files for various games including: Chicken, Matching Pennies, Rock-Paper-Scissors, Stackelberg, Hill Climbing, Penalty, Power Grid and the Pollution Tax Model. To play the games I have implemented the Policy Hill Climbing (PHC), Win or Learn Fast (WoLF), Lenient Learner, Delayed Q-Learner (DQL) and a pure strategy player for Chicken[5, 21]. Each agent's policy history is recorded in a data file throughout the simulation so that offline analysis and graphing can be done with separate tools.

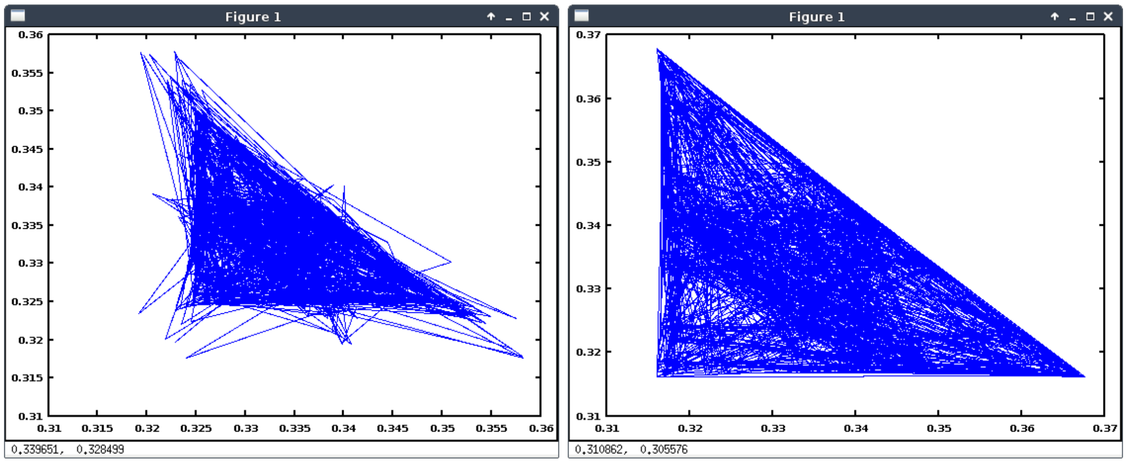
This paper first examines some of the results of repeated games I used to test PHC and WoLF. I then go on to describe how well PHC and WoLF do in stochastic games. Next, I examine the different extensions to the game engine I made, such as the Lenient Learner, DQL and n-player games. Finally, I discuss questions and discoveries I had during the assignment and end with conclusions and future work.

Hill Climbing

To begin the project, I attempted to reproduce parts of the WoLF paper. So, I first implemented PHC and the WoLF algorithms. They are both hill climbing algorithms. In my implementation I followed their outline of the algorithms and applied a Boltzmann distribution to the updated policy to ensure a valid probability distribution. I also added an epsilon exploration policy so that there is some chance that the agents will follow a random strategy. I then set up the two repeated normal form game experiments that the paper considered, Matching Pennies and Rock-Paper-Scissors. "Since PHC and WoLF-PHC are rational, we know that if they converge against themselves, then they must have converged to a Nash equilibrium"[5].

Matching pennies when WoLF is in self play gamma = 0.1, alpha= 0.1, deltaw = 0.05, deltal 0.1 and epsilon 0.001 with 1,000,000 iterations. Epsilon is the probability of picking a random action not based on my policy. PHC was set with gamma = 0.1, alpha= 0.1 and delta = 0.1. Compared to the original results my PHC results do a better job at converging. PHC converges to cycling between a policy of 0.45 to 0.55 for choosing heads. WoLF does a little better at converging to a policy of 0.5. It cycles around 0.475 to 0.525 for choosing heads. I am not sure why I didn't experience the extreme fluctuations like the original paper did for PHC.

I then tried rock-paper-scissors with WoLF and PHC over one million iterations. I was able to make graphs that looked similar to those in the paper. When using WoLF I set the parameters to gamma: 0.1, alpha: 0.1, deltaw: 0.05, deltal: 0.1 and epsilon: 0.001. When using PHC I set the parameters to gamma: 0.01, alpha: 0.1 and delta: 0.1.



(a) WoLF playing rock-paper-scissors in self play. (b) PHC playing rock-paper-scissors in self play.

Figure 1: X-axis is probability for choosing rock and y-axis is probability of choosing paper. The lines are most dense around (.3,.3) which is the mixed strategy equilibrium for rock-paper-scissors.

Next, I created two different stochastic games. The pollution tax model which has deterministic transitions and the power grid model with non-deterministic transitions. Both of which are competitive games which I used PHC and WoLF to play in.

The pollution tax model has WoLF (gamma: 0.1 alpha: 0.1 deltaw: 0.05 deltal: 0.1 epsilon: 0.001) and PHC (gamma: 0.1 alpha: 0.3 delta: 0.2) converging to the joint action of (1,1) in each of the games out of one million. The majority of the games (84%) were held in state two. This can be seen to be the case because in the first state the Nash equilibrium is at (1,1) and the transition leads to state two. The Nash equilibrium in state two is at (1,1) and the transition is back to state two. So, I tried lenient learners. They also converged rather quickly to a pure strategy of playing (1,1) in state two. The state that appears that it would

To try another stochastic game, I created the power grid game, which I made up myself. In this game, the producer, which has alternative and regular sources of energy, is the row player. The column player is a consumer who can buy electricity and sell it back. The nature of the game is competition to see who can make the most money. In each game the two players can choose from buy, buy a lot, sell, sell a lot. The stochasticity comes from weather which influences how much the producer can produce. So, on a sunny day the price may be lower because of solar panels. Since the transitions are based on joint actions, the players may transition to a state where there is a blackout, or due to the weather an increase in demand that causes prices to be high.

I applied WoLF and PHC players to the power grid game. To evaluate the players, I looked at the q-values in each of the states at the end of five million games. In the first, second and forth states for both the producer and consumer the highest q-value was to sell a lot. The third state which was the blackout state the highest q-value was a mix between buy, sell and sell a lot. For the producer the third state the mix was between buy and buy a lot. Based on the game matrix, these actions seem appropriate.

Elaboration

The next part of the project was to in some way extend the game engine. I chose to add an implementation of the Lenient Learner and capabilities for n-player games. I created my own algorithm called the Delayed Q-Learner (DQL). I also researched closed-form approaches to finding equilibria in stochastic games such as Minimax Q, Friend-or-Foe Q (FFQ) and QPACE [17, 18, 20].

Lenient Learner

The Lenient Learner algorithm game is made for cooperative repeated games. However, it performed well in the paper when playing a stochastic version of the Hill Climbing game. So, the goal was to extend the algorithm to be state aware. To test, I first implemented the version of the algorithm detailed in the paper. I ran the algorithm against the two types of repeated games used in the paper, Penalty and Hill Climbing. My results were similar to those in the paper.

I next made an attempt to adapt the algorithm to use state information in order to see if it performed any better in the stochastic version of Hill climbing. To do so I augmented the equation for updating U to:

$$U(s,a) = \lambda U(s,a) + (1-\lambda)(r + \gamma \underset{a'}{max} U(x',a'))$$

which is similar to the Q-table in Q-learning.

I then ran the augmented algorithm against the stochastic hill climb and the results were about the same as those obtained in the original paper. However, since in table 7 (the stochastic version of the hill climb) of the original paper I am unsure exactly what the numbers mean since the game was stochastic. What percent was using the first payout versus the second payout?

Delayed Q-Learner (DQL)

I then tried to create my own multi-agent learning algorithm. The idea is based on the idea of delaying the update of the q-value and the policy for x number of time steps. The algorithm would keep track of the history and create an average reward. The policy for each action would be initialized to be 1/|A| so that we can start with a large x. Then, we can update the q-value for each of the actions based on the average reward and the most frequently entered next state. The policy would be updated based on the current action and state, but using the updated Q-table. Finally, we can modify x based on a damped cosine wave as time goes on. I use the same selection method as PHC to choose an action. The idea of the DQL algorithm is somewhat similar to a lenient learner, but may also work against competitive players as well. DQL is not the same as having epsilon exploration, due to the fact that I average the rewards. The time where I average is sort of like a learning period to see what type of opponent I am playing with. However, once x becomes 1 the algorithm would decay into the PHC algorithm as the policy is updated every timestep. So, I believe that I will have at least the same guarantees as PHC. Say start with a max of 10,000 time steps before updating the policy. I would then decrease the max by the percentage d(t) as seen in the following equations.

$$d(t) = e^{\frac{-t}{4}} cos(2\pi t)$$

$$max_{t+1} = max_t * d(t+1)$$

$$\bar{r}_t^{(i)} = \frac{r_t^{(i)}}{max_t^{(i)}}$$

t starts at zero and is incremented by one once the player has played max games. $r_t^{(i)}$is the total reward accrued for action i at time t. Finally, $max_t^{(i)}$ is the number of times action i was played in time t. Once max equals one then I update q-values and policy as I did for PHC. One possible change would be to to use WoLF style updating instead.

N-player Games

One area I find extremely fascinating in multiagent learning is when there are a large number of agents. So, I added to the game engine the capability for simulating n-player games. Since the cells of game matrices must be in row-major order in the config file, I used a generalized formula for calculating the corresponding cell of the joint action.

$$n_d + N_d \cdot (n_{d-1} + N_{d-1} \cdot (n_{d-2} + N_{d-2} \cdot (\cdots + N_2 n_1) \cdots)) = \sum_{k=1}^{d} \left(\prod_{\ell=k+1}^{d} N_\ell \right) n_k$$

Figure 2: Given coordinates $(n_1, n_2, ..., n_d)$ and dimensions of size $N_1, N_2, ..., N_d$ the index can be calculated using the above formula. Image from http://en.wikipedia.org/wiki/Row-major_order.

Discoveries

While testing the algorithms I discovered various things. I learned that setting the learning parameters and number of iterations correctly is very important in order to achieve a good result. I also learned how difficult it is to create a stochastic game that resembles the real world. The project illustrated to me that the current state of the art for learning in stochastic games is solving linear programming problems. I also developed some questions while implementing the engine.

What if we have a stochastic game S and a set of agents A and in each game s we might have a different subset of A playing? One application seems to be a tournament type game. What type of game is that? Also, the agents that play in each s could be stochastic or fixed. Of course, we can always make it harder by making the number actions varied for the agents.

Another modification could be that at each state there is a set of games in which different subsets of agents play. The idea is that all players would be playing in each state. However, depending on the state they are in, who they are playing with may be different. What is a good way to learn in that situation and how would convergence to an equilibrium be determined? It may then be possible for games within a state to be parallelized. They may also be played serially, such that in the current state there are x number of games that need to be played before going to the next state. Like in stochastic games, how they are played determines the transition probabilities.

Another curiosity was, what if coalition games were played in a stochastic game setting? This seems to better model real life. USC agrees and in their summary they state they are using stochastic coalitional game theory as a technical approach for modeling human adversarial behavior. However, the topic seems underdeveloped.

Conclusion & Future Work

In this paper I showed the results of PHC and WoLF on various repeated and stochastic games. I detailed my attempt to augment the Lenient Learner algorithm and the creation of Delayed Q-Learner. Additionally, I described the details of my method for handling n-player games. Finally, I discussed the various discoveries and questions I developed throughout the assignment.

In the future I would like to improve the Lenient Learner algorithm by looking at changing the update rule to handle joint actions when calculating U. I would also like to study large multiagent learning systems in more detail. It would also be interesting to look at how well DQL works against other algorithms, as I only implemented and did not test the algorithm. I hope to also address some of the questions I developed. Specifically, stochastic games with variable number of agents. Future work on the game engine itself would be to add a game builder which would allow the user to create games within a GUI and have the game file automatically generated. The paper [16] lists many stochastic games that I would like to add to the game engine.

References

[1] Sherief Abdallah and V Lesser. A multiagent reinforcement learning algorithm with non-linear dynamics. *Journal of Artificial Intelligence Research*, 33:521–549, 2008.

[2] JM Black and DF Hougen. A benchmark for cooperative learning agents. *proceedings of the 21st national conference on ...*, pages 1855–1856, 2006.

[3] Bruno Bouzy and M Métivier. Multi-agent learning experiments on repeated matrix games. *... 27 th Intl. Conf. on Machine Learning*, 2010.

[4] Michael Bowling and M Veloso. An analysis of stochastic game theory for multiagent reinforcement learning. 2000.

[5] Michael Bowling and M Veloso. Rational and convergent learning in stochastic games. *International Joint Conference on Artificial ...*, 2001.

[6] Andriy Burkov and Brahim Chaib-draa. Multiagent learning in adaptive dynamic systems. *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems - AAMAS '07*, page 1, 2007.

[7] L Busoniu. A comprehensive survey of multiagent reinforcement learning. *Systems, Man, and ...*, 38(2):156–172, 2008.

[8] G Chalkiadakis. Multiagent reinforcement learning: stochastic games with multiple learning players. *Department of Computer Science, Univeristy of ...*, 2003.

[9] YH Chang. *Approaches to multi-agent learning.* PhD thesis, MIT, 2005.

[10] YH Chang and LP Kaelbling. Hedged learning: Regret-minimization with learning experts. *... international conference on Machine learning*, 2005.

[11] Liam Mac Dermed, CL Isbell, and Lora Weiss. Markov games of incomplete information for multi-agent reinforcement learning. *Workshops at the Twenty-Fifth AAAI ...*, pages 43–51, 2011.

[12] Sam Ganzfried and T Sandholm. Computing equilibria in multiplayer stochastic games of imperfect information. *Proceedings of the 21st International Joint ...*, 2009.

[13] GJ Gordon. Agendas for multi-agent learning. *Artificial Intelligence*, (December), 2007.

[14] P Hoen, K Tuyls, and L Panait. An overview of cooperative and competitive multiagent learning. *... Adaption in Multi-Agent ...*, 2006.

[15] Ian A Kash, Eric J Friedman, and Joseph Y Halpern. Multiagent Learning in Large Anonymous Games. (Aamas):10–15, 2009.

[16] Joshua Letchford, L MacDermed, and Vincent Conitzer. Computing optimal strategies to commit to in stochastic games. 2012.

[17] ML Littman. Markov games as a framework for multi-agent reinforcement learning. *...international conference on machine learning*, 1994.

[18] ML Littman. Friend-or-foe Q-learning in general-sum games. *MACHINE LEARNING-INTERNATIONAL ...*, 2001.

[19] L MacDermed and CL Isbell. Solving stochastic games. *Proc. NIPS*, pages 1–9, 2009.

[20] L MacDermed and KS Narayan. Quick polytope approximation of all correlated equilibria in stochastic games. *Twenty Fifth AAAI ...*, 2011.

[21] Liviu Panait, Keith Sullivan, and Sean Luke. Lenience towards teammates helps in cooperative multiagent learning. *...on Autonomous Agents and Multi Agent ...*, 2006.

[22] Y Shoham and K Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations.* 2008.