BotWar Simulation using Fuzzy SARSA(λ) Learning

By Francois Chaubard and Nikolaus West
fchaubar@stanford.edu nwest2@stanford.edu
Stanford University

Abstract

In this paper, we detail the analysis and results of a reinforcement learning experiment in the case of a Bot War Simulation. Using a reinforcement algorithm called Fuzzy SARSA(λ) coupled with a few 2-D pyramid member functions, we were able to achieve the performance of regular SARSA(λ) with 1.7 million times less states. We were also able to achieve a convergence rate that was 8 times faster.

1. Introduction

In simulated bot war multiple bots, or agents, compete against each other in teams, on a gridded playing field, with the objective of inflicting as much damage as possible to opponents. In our set up, several Bots, in several teams learn simultaneously, both how the game works and how to cooperate and compete in it. We developed a Bot War simulator to see how effective a reinforcement algorithm would be and what could be explored to better the results. The environment of the game is set as such:

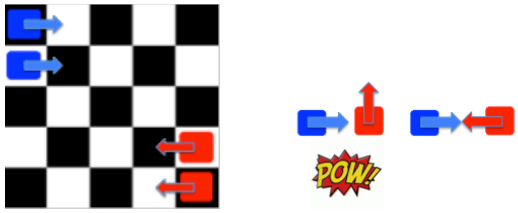

Figure 1: Layout of a Bot War

A hurt is logged only when a Bot is positioned to the side or behind another Bot. At each step of the game, every Bot must choose between four actions: stand still, turn left, turn right, and move forward.

Every game is run for 50 steps and the size of the environment is 5x5. A team is declared the winner when they have a combined higher score, which is number of hurts to the enemy less number of hurts to yourself and teammates.

2. Approach

During our implementation, we originally used SARSA(λ) learning. Due to memory restrictions and limited computer ability, we altered this approach to using an algorithm with state interpolation called Fuzzy SARSA(λ).

2.1. SARSA(λ)

For each Bot, the game is modeled as a Markov Decision Process (MDP) where the x and y distances and relative orientation of each teammate and enemy, as well as the Bot's own absolute position and orientation define the state, s. All actions, a, are modeled as possible actions in all states. Actions that are in fact impossible to take in a certain state will simply not change the agent's position.

The transition probabilities are not explicitly modeled/learned. Instead, the Bots learn the values of a state-action value function (Q-matrix). The Q-matrix is learned with the on-policy algorithm SARSA(λ), using replacing eligibility traces described in [1]. An $\varepsilon - greedy$ policy is used to select an action given the values of the Q-matrix corresponding to the current state. The algorithm, almost exactly as defined by Sutton & Barto, 1998, section 7.5, is as follows:

$$
\begin{aligned}
&Initialize\ Q(s,a)\\
&Repeat\ (for\ each\ game)\\
&\quad Initialize\ e(s,a) = 0\ for\ all\ s,a\\
&\quad Initialize\ s\\
&\quad Choose\ a\ from\ s\ using\ \varepsilon - greedy\ policy\ derived\ from\ Q\\
&\quad Repeat\ (for\ each\ step\ of\ game):\\
&\qquad Take\ action\ a, observe\ reward\ r, s'\\
&\qquad Choose\ a'\ from\ s'\ using\ policy\ derived\ from\ Q\\
&\qquad \delta \leftarrow r + \gamma Q(s',a') - Q(s,a)\\
&\qquad e(s,a) \leftarrow 1\\
&\qquad For\ each\ s,a:\\
&\qquad\quad Q(s,a) \leftarrow Q(s,a) + \alpha\delta e(s,a)\\
&\qquad\quad e(s,a) \leftarrow \gamma\lambda e(s,a)\\
&\qquad Choose\ a'\ from\ s'\ using\ \varepsilon - greedy\ policy\ derived\ from\ Q\\
&\qquad s \leftarrow s'; a \leftarrow a';\\
&\quad until\ game\ is\ done
\end{aligned}
\tag{1}
$$

Each time an action a is chosen in state s the eligibility trace $e(s,a)$ for that state-action pair is set to 1. Every round all eligibility traces are decayed by the trace decay parameter λ. The trace values are then used so that all the state-action values are updated dependent on how long ago it was the corresponding state-action pair was visited before the current reward was observed. α is the learning rate, δ is the temporal difference for the current time-step, and γ is the discount factor, which is used to discount the value of future rewards versus rewards in the current time-step.

2.2. Fuzzy SARSA(λ)

The main problem with SARSA(λ) applied to Bot War is that the expected value of each state-action pair is saved and the number of states is given by the huge number $P_n^z \cdot 4^n$ (2), in games with z grid positions, n Bots, and 4 orientations. To get around this problem a Fuzzy SARSA(λ) Learning algorithm [3] was used, where the states experienced by one Bot are expressed as all the other Bots' responses to a set of fuzzy membership functions. With m well-chosen membership functions, this results in that a state can instead be expressed as a sparse vector with $r = m \cdot (n-1)$ (3) elements.

Instead of using separate membership functions for the x- and y-directions, we used pyramid shaped membership functions laid out on the 2D playing field relative to the observing Bot. The response of a membership function to another Bot's position is given by the height of the pyramid at that Bot's relative position to the observing Bot. The height of each pyramid is 1, and the responses to each of the membership functions to the relative positions of all the other Bots are normalized to sum to 1 and concatenated to a vector $\Phi(s)$, where the i-th element is given by $\mu_i(s) \in [0,1]$.

The pyramids overlay each other in varying degrees to make use of the fuzzy formulation. We use five outer pyramids that describe positions not close to the observing Bot, and 16 (4 for each direction) orientation sensitive pyramids to describe positions and orientations close to the observing Bot. The layout of these pyramids can be seen in the figure below.

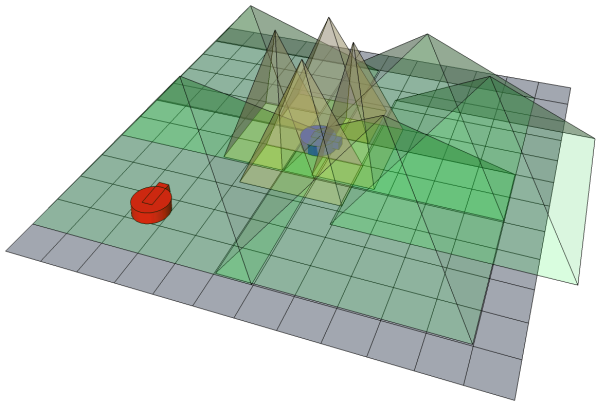

Figure 2: The layout of the 21 pyramid member functions. The green pyramids are angle insensitive. The yellow are angle sensitive meaning there are four overlaid pyramids for each.

The Q-matrix is a matrix with r rows and 4 columns, where the entry at (i,a), $q_a^{(i)}$, represents the expected value of taking action a if $\mu_i(s) = 1$. The $\varepsilon - greedy$ policy in state s therefore becomes:

$$\begin{aligned} &With\ probability\ \varepsilon, choose\ random\ a, else: \\ &a = argmax_a \max(\Phi(s)Q) \\ &= argmax_a \max\left(\sum_{i=1}^{r} \mu_i(s)[q_1^{(i)}, q_2^{(i)}, q_3^{(i)}, q_4^{(i)}]\right), \end{aligned} \quad (4)$$

We also use focused replacing eligibility traces described in [4] which changes the update for $\delta\ and\ e(s,a)$ given in (1) to:

$$\delta \leftarrow r + \gamma \sum_{i=1}^{r} q_{a'}^{(i)} \cdot \mu_i(s') - \sum_{i=1}^{r} q_a^{(i)} \cdot \mu_i(s), \quad (5)$$

$$e^{(i)}(s,a_j) \leftarrow \begin{Bmatrix} \mu_i(s), & if\ \mu_i(s) \neq 0\ and\ a_j = a \\ 0, & if\ \mu_i(s) \neq 0\ and\ a_j \neq a \\ \gamma\lambda e(s,a_j), & otherwise \end{Bmatrix}, \quad (6)$$

3. Results

We ran our "Fuzzy Smart" Bots (Bots using Fuzzy SARSA(λ) Learning) against three types of opponents: "Really Dumb" Bots (Not moving Bots), "Dumb" Bots (Bots moving in uniformly random fashion), and "Smart" Bots (Bots using SARSA(λ) Learning). Our results break down into three findings.

First, Fuzzy SARSA(λ) Learning is a very effective method for this type of learning situation. During the Really Dumb trials, the Fuzzy Smart Bots converged to a direct path to the enemy's vulnerable positions and sat

there for the remainder of the game collecting points. They found this optimal path in less than ten games. During the Dumb trials, the Fuzzy Smart Bots also saw similar success.

Second, our analysis shows that Fuzzy SARSA(λ) using 21 pyramid member functions reduces the state size by many orders of magnitude. In the 2v2 case, the number of states for SARSA(λ) amounts to 77,721,600 as calculated by Equation (2), which is ~1,233,676 times higher than the Fuzzy SARSA(λ) state size of 63. One might ask what the performance cost of such a dramatic reduction of state size is. Figure 3 shows the reduction in performance is very minimal. The Fuzzy SARSA(λ) does not begin to lose until nearly 8000 runs in the 1v1 Fuzzy Smart vs. Smart case, and even then the advantage is no more than a score of one point. Also, Figure 4 shows the learning rate has increased 8 fold.

Lastly, showing that the Fuzzy Smart Bots can defeat a Dumb bot in an even fight is good but our team was more interested with how the Fuzzy Smart Bot would fare in an uneven fight. As shown in Figure 5, one Fuzzy Smart Bot was able to defeat up to 7 Dumb Bots at once. Beyond this point, our Bot cannot handle additional enemies.

3.1. Fuzzy SARSA(λ) vs. Dumb / Really Dumb

We tested our learning algorithm in a number of ways. The first test was against an enemy that simply stands still. Below is the output of such a trial. The algorithm converges to the optimal solution in ~8 games.

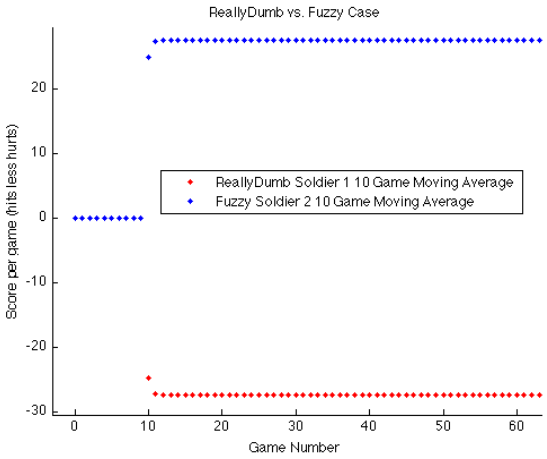

Figure 3: For the Really Dumb 1v1 trials, the Fuzzy Smart Bot learns the optimal policy in ~8 games compared to SARSA(λ) which took ~80 games.

We then wanted to ensure that the algorithm would still be successful when there are more than one Bot per team. As shown in Figure 4 below, the five Bots learn to each go after one specific Bot and not get in each other's way.

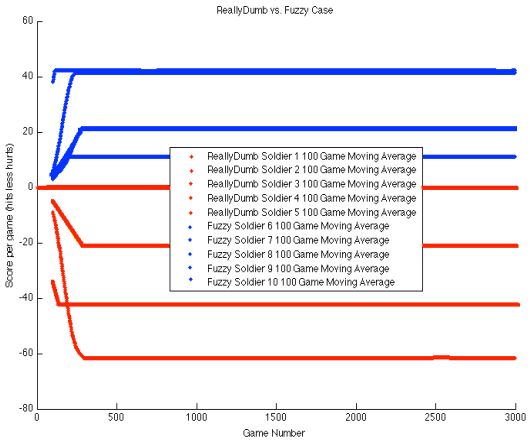

Figure 4: 5v5 Really Dumb case. Note that all Bots find the optimal policy and do not get in each other's way.

Now that we saw that the algorithm worked we wanted to see how it fared in more precarious situations. We tested the algorithm in a 5v5 battle against a uniformly random moving enemy. As show below, all of the Fuzzy Smart Bots worked together to defeat the opponent. We noticed that the strategy that began to form was for the Bots to clump together and only move to attack when there is an easy chance to strike.

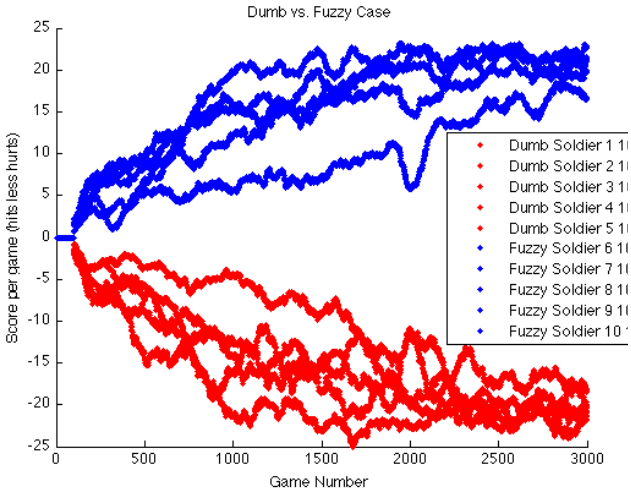

Figure 5: 5v5 Dumb trial. Notice that the Fuzzy Smart Bots are able to consistently defeat the meandering enemy Bots.

3.2. Fuzzy SARSA(λ) vs. SARSA(λ) Learning

We originally implemented SARSA(λ) Learning but quickly realized the memory intensiveness of such an algorithm would not allow us to scale a battle to have anymore than 2 Bots. In a 2v2 case, the Q-matrix fills at an alarming rate and actually occupies the 64 Megabytes available in the JVM causing the program to crawl and then eventually crash. Running the Fuzzy SARSA(λ) algorithm takes no where near that amount of memory, allowing the program to easily scale to higher number Bot trials. We successfully ran a 20v20 trial which would never be possible with SARSA(λ) learning. The drawback to such a dramatic reduction is of course correctness of action choice and overall ability of the Bot. In fact, we saw the opposite of this. The Fuzzy Smart Bot performance against the Really Dumb Bot converged in about 8 trials on average while the Smart Bot learned in about 80 trials as seen in the figure below.

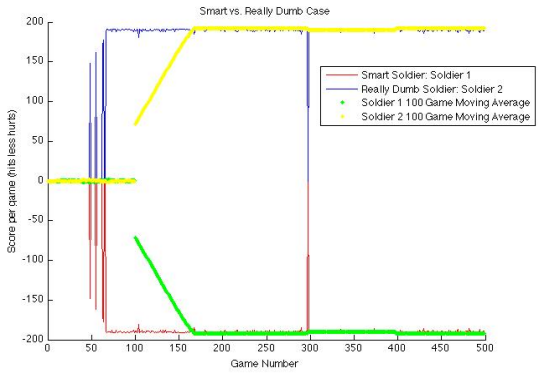

Figure 6: Using classic SARSA(λ) , we see convergence occurs at the 80th game. Fuzzy SARSA(λ) converges at around 8 games. Notice the big spike at game 300. This is the epsilon value taking effect.

We were also interested in seeing how the two learning algorithms fared in a head to head trial. Below we show a Fuzzy SARSA(λ) Bot vs. SARSA(λ) Bot.

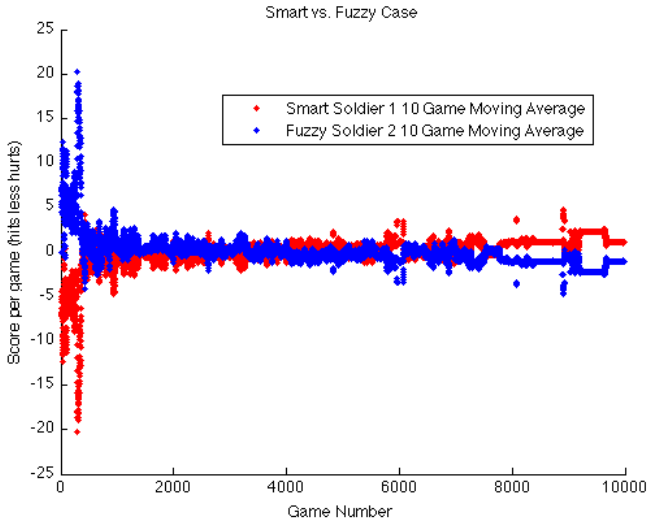

Figure 7: Fuzzy Smart vs. Smart trial. Notice that the Fuzzy Smart Bot easily wins the first 2000 games. The Smart Bot does not begin to win until the 8000th game.

Since the Fuzzy Smart Bot learns much faster, he wins the early games with ease and every time there is a change in the policy action of the other Bot, the Fuzzy Smart Bot quickly adapts to it. We noticed also that when the Smart Bot finally begins filling its Q-matrix, its policies are better than the Fuzzy Smart Bot. This is to be expected since an approximation of the states will never be as good as a full state representation.

3.3. The Hunger Games

Our final round of tests conducted we dubbed the "Hunger

Games". Fuzzy SARSA(λ) Learning is good but we wanted to see how good. Pinning one Fuzzy Smart Bot into a Battle against many Dumb Bots gave us some very interesting results. Figure 4 below shows an example of a 1v7 trial.

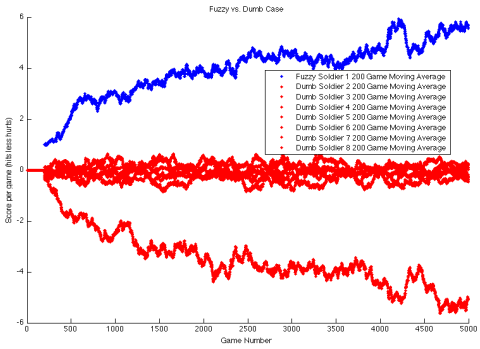

Figure 8: The one Fuzzy Smart Bot vs. seven Dumb Bot trial shows the robustness of the algorithm.

We found that the Bot could conquer up to 14 other Bots with ease. However, any additional soldiers added to the enemy team would give a different result. As shown in the figure below, the Fuzzy Smart Bot can not defeat 15 other dumb bots.

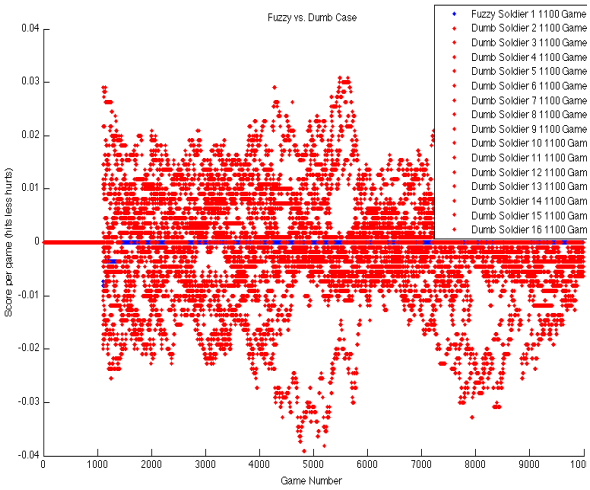

Figure 9: 1v15 overwhelms the Fuzzy Smart Bot. Notice the Bot does not lose by much, in fact the Bot usually breaks even.

4. Conclusion

The analysis shows how effective a good state interpolation technique can be. With the addition of just 21 membership functions per soldier, we were able to achieve a faster learning rate, no real loss of performance, and a memory reduction that is around 6 orders of magnitude. Future research must be done on selecting the optimal pyramid member functions, their positions, size, and number of pyramids. We believe this good improve results even more. Function approximation for Multiple Agent Learning problems is an active field of research within the Machine Learning community, which in itself makes it an interesting problem. Specifically, we are interesting in extending the game presented here to a Zynga-style online "War-bots manager" online computer game.

References

[1] Singh, S.P., Sutton, R.S.: Reinforcement Learning with Replacing Eligibility Traces. Machine Learning, 22, 123-158, Kluwer Academic Publishers, Boston, 1996

[2] Sutton, R.S., Barto, A.G.: Reinforcement Learning: an Introduction. The MIT Press, Cambridge, 1998

[3] Theodoris T., Hu, H.: The Fuzzy Sara'a'(λ) Learning Approach Applied To a Strategic Route Learning Robot Behavior. Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, Beijing, China, 2006.

[4] Kamdem, S., Ohki, H., Sueda, N.: Fuzzy Sarsa with Focused Eligibility Traces for Robust and Accurate Control, IEEJ Transactions on Electronics, Information and Systems, Volume 130, Issue 6, pp. 1023-1033, 2010.