Reinforcement Learning

Nuray Dindar, S. Ozgur Oguz

Abstract

Reinforcement Learning methods are receiving increasing attention not just in artificial intelligence but also from many other domains. Early studies in AI use games as a testbed, and recently, many computer games also incorporate reinforcement learning approaches within their interactive environment. We develop a testbed application where two agents fight with each other. We test three reinforcement learning algorithms, namely Q-learning, SARSA, and feature based function approximation, within that domain. Our results show that SARSA accumulates the highest reward, and the function approximation becomes the fastest learner.

Keywords: Reinforcement Learning, Q-learning, SARSA, Function Approxiamtion.

1. Introduction

Artificial Intelligence (AI) has always played a vital role for practically all commercially released video games. With the advancement of the available computer power for the last decade, commonly used script and finite-state based AI techniques have been replaced by novel AI techniques in games such as Creatures from CyberLife Technologies, Black & White from Electronic Arts, Forza Motorsports from Microsoft Game Studios, and F.E.A.R. from Monolith Productions among many others.

Video games are valuable testbeds for enhancing and evaluating AI algorithms. However, most of the current use of AI in games is related to either path-finding or action selection. There is clearly an open ground for exploring other techniques within video game environments. Those methods include reinforcement learning, supervised/unsupervised learning, and more sophisticated planning algorithms. The general criteria for evaluating an AI behavior within video games would be to assess its chances of winning, and/or the entertainment value it delivers. But, in our project we observe the performance of the agents, rather than the entertainment value.

In reinforcement learning, an agent learns how to act in a dynamic environment by trial and error interactions. Formally, the reinforcement learning models consist of

- a set of states, S;
- a set of agent actions, A;
- a set of scalar reinforcement learning rewards/punishments, R;
- a policy function π which defines which action to execute in a state.

As Sutton and Barto (1998) describe, reinforcement learning is about finding mappings from situations to actions in order to maximize the numerical reward signal, R. The agent does not have any knowledge about its environment, and it is not told which actions to take.

Instead, the agent must explore which actions produce the most reward by trying them. Most of the time, actions may affect not only the immediate reward but also the following situations, and hence the next rewards. Trial and error interaction together with the delayed reward concept constitutes the two fundamental aspects of reinforcement learning. In terms of how they approach the reinforcement learning problem and its fundamental aspects, there are several different approaches. Two of the main methods are policy search and value iteration methods.

Policy search methods aims to select the optimum policy that maximizes the expected reward collected. The main objective is to search over the space of all policies to find the best one. The agent can be controlled by a policy, and its value is determined as the agent acts in the environment. One common approach is using *evolutionary algorithms*, which start with an initial policy, then evaluate and improve that policy iteratively. However, there are possible drawbacks of the evolutionary algorithms. One is the size of the state space, which can result in intractable algorithms. Second, they do not use experiences efficiently in the sense that an action may be determined as good or bad very indirectly (Poole and Mackworth (2010)). Third, the representation of the policy can play a significant role on the performance of the evolutionary algorithms. This requires the tuning of the representations for each domain. Sutton and Barto (1998) claim that evolutionary algorithms can be effectively used if the space of policies is sufficiently small, or well-structured such that good policies are common and easy to find. Those algorithms are criticized as they do not use the fact that the policy they are searching for is a function from states to actions.

Second approach relies on iteratively improving the *value function* of the problem. As the *reward function* defines the goal in a reinforcement learning problem, a *value function* specifies what is good in the long run. Sutton and Barto (1998) consider rewards as the determinant of immediate desirability of the environmental state, and the values as the indicator of the long-term desirability of states after taking into account the states that are likely to follow. As Kaelbling et al. (1996) argues value iteration is very flexible in the sense that the V value can be assigned asynchronously.

2. Related Work

In the last two decades, Reinforcement Learning has emerged as a powerful tool for solving problems in decision making, control theory, information theory, genetic algorithms, and many other fields. There is a range of algorithms developed within those fields. The foundational paper by Sutton (1988) places the *temporal difference* methods as an important class of algorithms for RL. Another influential study by Watkins (1989) introduces Q-learning in which the q-values are iteratively improved. Rummery and Niranjan (1994) propose an online version of q-learning which eventually leads to SARSA. More sophisticated algorithms include policy gradient algorithms (Williams (1992), Sutton et al. (2000), Baxter and Bartlett (2001)); hierarchical reinforcement learning algorithms (Sutton et al. (1999), Dietterich (2000), Barto (2003)); and relational reinforcement learning methods (Deroski et al. (2001), Driessens (2006)) among many others.

In robotics, there has been a large amount of study on large-state-space applications such as RoboCup soccer. Uchibe (1999) uses reinforcement learning methods for a robot playing soccer to learn to shoot a ball into a goal while avoiding its opponents. Within same domain, Stone and Sutton (2001) apply a reinforcement learning algorithm with linear function approximation for a more sophisticated task called "keepaway", where a number of robots try to maintain the possession of the ball within a limited region. In control tasks, there are also many successful applications of RL algorithms. For example, Kohl and Stone (2004) present a RL algorithm to optimize a quadruped robot's locomotion. Fidelman and Stone (2004) further improve the previous approach to help the quadruped robot learn a more high-level, goal-oriented task, such as capturing a ball. Ng et al. (2004) successfully applies RL methods to autonomous helicopter flight.

As the games provide suitable test grounds, many studies focus on analyzing their methods within that domain. Actually, many early studies in RL has also chosen games as their domain. Temporal difference algorithm implemented by Tesauro (1995) is among the most famous applications of RL algorithms tested on a game domain. The backgammon playing agent can learn to play as successful as a professional human player. Schaeffer et al. (2001) also use temporal difference learning for the chess game, which result in a highly successful chess-player agent. Likewise, Yan et al. (2004), and Schaeffer et al. (2001) work on RL algorithms for agents to learn playing solitaire, and checkers, respectively. Recently, video games have also started to incorporate more intelligent characters. Hence, they rely on recent AI techniques, and especially reinforcement learning algorithms, to create interactive and intelligent characters. Microsoft (2009) introduced Drivatar technology for their car racing simulation game, Forza Motorsports. Human players can train their own drivers by driving several types of lanes. The agent then learns how to drive by using the previous data generated by the human player. Lionhead Studios released the video game Black & White in 2001 in which human players can manipulate an AI controlled creature (Lionhead (2001)). The creature's behavior changes over time based on reinforcement learning (Rabin (2002)).

As Russell and Norvig (2003) argue that reinforcement learning is a feasible way to train an agent to perform at high levels. For example, in game playing, it can be very hard for a human to evaluate large numbers of positions, actions accurately and consistently. On the other hand, the agent can be told when it has won or lost, and in turn, can use this feedback to learn an evaluation function to make reasonably accurate estimations about its next strategy. Before going into details of our application, we will briefly give a general information regarding reinforcement learning methodology in the next section.

3. The Game

In our study, we focus on reinforcement learning algorithms, and their possible applications on video games domain. As previously stated, reinforcement learning can be defined as learning policies from state-action-reward sequences, hence, video games domain provide a suitable workspace for RL algorithms. The tasks within a computer game could be control, value estimation, and policy learning, whereas the possible applications could include learning to drive, walk, and fight, etc.

As a testbed application, we implement RL algorithms in order to train fighting agents. We define different states and reward / punishment combinations and compare the resulting agent performances with each other. Learning algorithms that we have implemented include feature-based learning, Q-learning, and SARSA learning. For the fighting domain, game

	OFFENSIVE		DEFENSIVE	NEUTRAL			
Separation	Kick	Punch	Block	Walk Forward	Walk Backward	Run Forward	Run Backward
Close	7	15	3	0	5	0	5
Near	10	7	4	3	8	0	8
Far	0	0	0	4	0	4	0

Table 1: Rewards for different action and distance pairs

state features can be a combination of 'Separation (distance between opponents)', 'Last action', 'Mode (ground, knocked)'. On the other hand, possible actions can be 'kick', 'throw', 'stand', 'run', etc. In this application, we define a pre-configured character and train our fighting agent against it. We change reinforcement signals and test the corresponding trained character behavior.

For that testbed application, we train our agents and observe their behavior under different settings. Additionally, we also investigate how the performance of an agent gets affected with exploitation and exploration. Details of our application are explained in the following sections.

3.1 Reward & Punishment

The task of reinforcement learning is to use rewards to learn a function that makes the agent successful. This is difficult because the agent is never told what the right actions are, nor which rewards are due to which actions. Even though an agent plays a game flawlessly except for one mistake, at the end of the game it may get a single reinforcement that says "you lose", or vice-versa. The agent must somehow determine which move was the mistake or which move actually helped it to win. Hence, we designed our game to give a proper reinforcement signal to our agent as frequently as possible. In this sense, we use damage information as the main reinforcement signal. We classify each action as offensive, defensive, or neutral. The offensive actions have a damage value for each separation distance between the agents. On the other hand, the defensive action, i.e. blocking, has a value for decreasing the amount of damage taken. The neutral actions help the agent to either move closer to or run away from its opponent. Those actions and their corresponding damage values are given in Table 1.

3.2 Exploration vs. Exploitation

Reinforcement learning is different from other learning approaches, such as supervised learning, in terms of the trade-off between exploration and exploitation. To accumulate rewards, a reinforcement learning agent must prefer actions that it has tried in the past and found to be effective in producing reward. However, in order to discover such actions, it has to try actions that it has not selected before. In other words, the agent has to exploit what it already knows in order to obtain reward, but it also has to explore in order to discover other possibilities and select better actions in the future. Hence, instead of exclusively pursuing either exploration or exploitation, the agent must try a variety of actions and progressively favor those that appear to be the best. On a stochastic task, each action must be tried many times to gain a reliable estimate about its expected reward. We simply choose a ϵ-greedy strategy, where we set the greedy probability as 0.8 and 1 for the first and second phases respectively, in order to deal with the issue of balancing exploration and exploitation in our application. Hence, the agent selects the greedy action all but 1-ϵ of the time and selects a random action 1-ϵ of the time, where $0 \leq \epsilon \leq 1$.

3.3 Opponent Model

We implement a rule-based opponent, and we test all our three controllers against the same agent. Rule-based agent is characterized by a stochastic action selection mechanism. We tried to make that agent neither too intelligent nor too dull. It acts randomly for 20% of the time. For the remaining times, it balances its movements between offensive, defensive, and neutral actions, which are also decided stochastically.

4. Methods

For our testbed application, we implemented several RL algorithms, such as Q-learning, SARSA, and feature-based function approximation. Now, we will give brief information about those learning algorithms.

4.1 Q-Learning

In Q-learning an agent interacts with its environment and by using the history of those interactions, it tries to learn an optimal policy. A history of interactions consists of essentially a state-action-reward sequence:

$$s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, ... \tag{1}$$

In Q-learning, each s, a, r, s' tuple is considered as an *experience*, which means that the agent being in state s, made the action a, and received the reward r, in turn, the state changed into s'. The data gathered with those experiences is used by the agent to learn what to do. Formally, the agent updates its perception by:

$$Q[s,a] \leftarrow (1-\alpha_k)Q[s,a] + \alpha_k(r + \gamma max_{a'}Q[s',a']) \tag{2}$$

One simplification with Q-learning algorithm can be to assume a fixed α value. However, in our implementation we keep track of visits for each state-action pair as a different count (this is why use the notation as α_k).

4.2 SARSA

Q-learning is considered as an off-policy method, since it learns an optimal policy as long as it explores enough. On the other hand, SARSA is an on-policy method as it learns the policy that the agent is actually carrying out, and which can be improved iteratively. As a result, SARSA uses s, a, r, s', a' tuple as its experiences. Here, $Q(s', a')$ provides a new experience to update $Q(s, a)$. Those updates can be represented as,

$$Q[s,a] \leftarrow (1-\alpha_k)Q[s,a] + \alpha_k(r + \gamma Q[s',a']) \tag{3}$$

Poole and Mackworth (2010) point out that SARSA controller can learn a different policy than the Q-learning when the exploration may result in large penalties. Hence, SARSA can be useful in situations where we want to optimize the value of an agent that is exploring.

4.3 Feature Based Function Approximation

When there are many states to reason about, it might be reasonable to learn in terms of features. In order to implement a feature based learning algorithm, we use a linear function of features to approximate the Q-function in SARSA. Our features are a combination of states and actions, and they are used to represent the Q-function.

$$Q_w(s,a) = w_0 + w_1F_1 + ... + w_nF_n \tag{4}$$

for some tuple of weights, $w = w_0, w_1, ..., w_n$. Like in SARSA, an experience of the form s, a, r, s', a' gives a new estimation of $r + \gamma Q[s', a']$ to update $Q(s,a)$. Hence, we can update the weight w_i by:

$$w_i \leftarrow w_i + \eta\delta F_i, \ \ where \tag{5}$$

$$\delta = r + \gamma Q(s', a') - Q(s,a). \tag{6}$$

The agent selects a random action with probability 1-ϵ, and otherwise selects an action that maximizes $Q_w(s,a)$ same as it would do with SARSA. Here, it is non-trivial to choose which features to represent and update the Q-function.

In our implementation, for feature based functional approximation learning, the following features are used.

- $F_1(s,a)$ is the difference between the health changes of agents in state s.
- $F_2(s,a)$ is the distance between agents in state s.
- $F_3(s,a)$ is the damage agent gives to its opponent in state s.
- $F_4(s,a)$ is the damage agent takes from its opponent in state s.
- $F_5(s,a)$ is the separation between agents in state s, e.g. close, near, far.
- $F_6(s,a)$ is 1 if the agent hits the wall in state s and has value -1 otherwise

5. Experiment and Results

To compare performances of different reinforcement algorithms, we made each agent to fight against a rule based agent and compared their performances by using the average total accumulated reward over 10 trials. All algorithms used greedy exploit of 80% for the first 10.000 steps, and 100% for the next 10.000 steps. As can be seen on Figure 1 and statistics on Table 2, SARSA outperformed all other learning algorithms. The slope of accumulated rewards is steepest in SARSA both with 80% and 100% greedy exploit, which supports the algorithm is better than the other ones. However, zero-crossing value on Table 2 shows that

Algorithms	**Asymptotic Slope 1**	**Asymptotic Slope 2**	**Zero Crossing**	**Minimum Reward**
Q-Learning	4.3338	7.0533	57	-73.70
SARSA	4.8192	9.6111	114	-112.20
Function Approximation	1.2921	5.1609	0	0

Table 2: Statistics for different reinforcement learning algorithms. Asymptotic Slope 1 and 2 show the rate of change of the accumulated reward during the first, and second phase of the simulation, respectively. ϵ is 0.8 for the first phase, and 1.0 for the second phase.

it took longest for the SARSA to recoup its cost of learning. The graph also clearly shows that Q-learning is worked better than function approximation. There might be couple of reasons for these results. First, we believe that agents acting according to Q-learning and SARSA learned that they should avoid crashing into walls, while agent acting based on function approximation algorithm could not learn that. Hence, to improve the performance of this agent additional new features should be defined. Another interesting point is the zero-crossing value for the function approximation method. Even though, results seem to suggest that the agent does not get punished initially -i.e. the zero-crossing is zero- this is most probably due to the values we have assigned to rewards and punishments. In reality, all of the agents with different controllers get punished during their initial exploration.

For the functional approximation learning, to prevent abrupt increase of feature weights, we should find the correct normalization parameters for the feature values. In Figure 1-b it can easily be seen how the normalization parameter effect the performance of learning algorithm. With the wrong normalization parameters the agent could not learn anything.

6. Conclusion and Future Work

We implement a testbed application where we can experiment with three main reinforcement learning algorithms, namely Q-learning, SARSA, and feature based function approximation. Our results show that with our current setup, SARSA learning achieves the best results in terms of accumulated reward. Feature based FA comes last, but we speculate the reason for this could be due to the insufficient number of features.

To improve our work, new actions could be added, so that there would be more options for choosing the optimum action. Another enhancement could be to define new states based on the agents' position in y-axis, such as 'air', 'knocked', and 'stand'. These states could diversify the actions an agent can perform, in essence, they increase the state-action space of our application. Furthermore, new features could be determined for feature based function approximation and their performance would be measured.

We may try to challenge our different controllers against each other in order to observe whether we could get reasonable results. Moreover, greedy exploitation probability, ϵ, can be modified dynamically within the simulation.

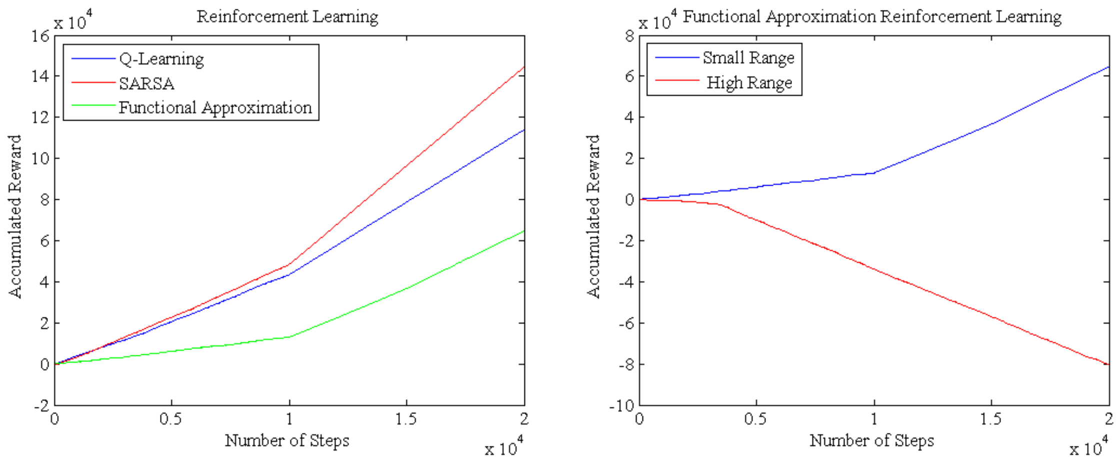

Figure 1: Cumulative reward as a function of the number of steps over 10 trials. On the left (a), the comparison of three methods; on the right (b), comparison of feature-based FA with different parameters are shown.

References

Andrew G. Barto. Recent advances in hierarchical reinforcement learning. 13:2003, 2003.

Jonathan Baxter and Peter L. Bartlett. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 2001.

Sao Deroski, Luc De Raedt, and Kurt Driessens. Relational reinforcement learning. *Machine Learning*, 43:7–52, 2001. ISSN 0885-6125. URL `http://dx.doi.org/10.1023/A:1007694015589`. 10.1023/A:1007694015589.

Thomas G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

Kurt Driessens. Relational reinforcement learning. In Michael Luck, Vladimr Mark, Olga tepnkov, and Robert Trappl, editors, *Multi-Agent Systems and Applications*, volume 2086 of *Lecture Notes in Computer Science*, pages 271–280. Springer Berlin / Heidelberg, 2006.

Peggy Fidelman and Peter Stone. Learning ball acquisition on a physical robot. In *In 2004 International Symposium on Robotics and Automation (ISRA*, 2004.

Leslie Pack Kaelbling, Michael Littman, and Andrew Moore. Reinforcement learning: A survey. *JAIR*, 4:237–285, 1996.

Nate Kohl and Peter Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *in Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2619–2624, 2004.

Lionhead. Technical report, Lionhead Studios, 2001. URL `http://lionhead.com/Games/BW/Default.aspx`.

Research Microsoft. Drivatar. Technical report, http://research.microsoft.com/en-us/projects/drivatar/default.aspx, 2009.

Andrew Y. Ng, H. Jin Kim, Michael I. Jordan, and Shankar Sastry. Inverted autonomous helicopter flight via reinforcement learning. In *In International Symposium on Experimental Robotics*. MIT Press, 2004.

David Poole and Alan Mackworth. *Artificial Intelligence: Foundations of Computational Agents*. 1 edition, 2010.

Steve Rabin. *AI Game Programming Wisdom*, chapter 11. Charles River Media, 2002.

G. A. Rummery and M. Niranjan. On-line q-learning using connectionist systems. Technical report, 1994.

Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003. ISBN 0137903952.

Jonathan Schaeffer, Markian Hlynka, and Vili Jussila. Temporal difference learning applied to a high-performance game-playing program. In *Proceedings of the 17th international joint conference on Artificial intelligence - Volume 1*, pages 529–534, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-812-5, 978-1-558-60812-2. URL `http://portal.acm.org/citation.cfm?id=1642090.1642163`.

Peter Stone and Richard S. Sutton. Scaling reinforcement learning toward robocup soccer. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 537–544, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-778-1. URL `http://portal.acm.org/citation.cfm?id=645530.655674`.

Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988. ISSN 0885-6125. URL `http://dx.doi.org/10.1007/BF00115009`. 10.1007/BF00115009.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: Introduction*. 1998.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112: 181–211, 1999.

Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *In Advances In Neural Information Processing Systems 12*, pages 1057–1063. MIT Press, 2000.

Gerald Tesauro. Temporal difference learning and td-gammon. *Commun. ACM*, 38:58–68, March 1995. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/203330.203343. URL `http://doi.acm.org/10.1145/203330.203343`.

Eiji Uchibe. Cooperative behavior acquisition by learning and evolution in a multi-agent environment for mobile robots. Technical report, 1999.

Christopher J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, May 1989.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8:229–256, May 1992. ISSN 0885-6125. doi: 10.1007/BF00992696. URL `http://portal.acm.org/citation.cfm?id=139611.139614`.

Xiang Yan, Persi Diaconis, Paat Rusmevichientong, and Benjamin Van Roy. Solitaire: Man versus machine. In *In Conference on Advances in Neural Information Processing*, 2004.