Reinforcement Learning in Games
Autonomous Learning Systems Seminar

Matthias Zöllner
Intelligent Autonomous Systems
TU-Darmstadt
zoellner@rbg.informatik.tu-darmstadt.de

Betreuer: Gerhard Neumann

Abstract

The field of reinforcement learning covers a considerably large variety of applications and possibilities of extensions. This report summarizes the basic concepts behind reinforcement learning and some of its applications in the field of game theory. The concept will be further discussed regarding the work of Tesauro with his TD-Gammon Program in 1995 [4] and the concept of Q-Learning in multi agent system as it was presented by Hu and Wellman in 2003 [1] where they combined the concept of Q-Learning with the Nash Equilibrium Points as an approach to reach better overall results in multiple agent environments instead of focussing on the individual per-agent reward. Some restrictions in what the Nash Q-Learning approach archieves will be discussed based on the examples which Hu and Wellman provided in their article.

1 Introduction

In the field of learning algorithms, the concept of reinforcement learning is particularely interesting because of its intuitive approach. Instead of hardcoding human knowledge about a problem, it depends on a learning progress with trial and error, where the process of decision making is improved by the result of the previously chosen actions. The TD-Gammon [4] learning software and the concept of Q-Learning with Nash Equilibrium Points [1] are presented as examples how reinforcement learning can be used as a basic concept in a variety of applications.

The basic concept of reinforcement learning is, to let an agent chose its action and provide feedback[1], which indicates how good or bad the reached state, which follows the agents decision, is. Based on this feedback the agent then updates its decision process accordingly to maximize its reward.

Many learning algorithms require a fully observable Markovian environment. Thus, the learning process can be modeled as a Markov Decision Process (MDP) which is defined as a Tuple $\langle S, A, R_{\cdot}(.,.), P_{\cdot}(.,.)\rangle$ where S is a set of all states, A is a set of all actions[2], $P_a(s, s')$ is the probability function which defines the probability to reach state s' when chosing action a in state s and $R_a(s, s')$ is the reward function for reaching state s' from state s with action a. $R_a(s)$ may be used to indicate a reward which does not vary depending on the reached state[3].

[1] Reward or penality are used as synonyms, depending on the context

[2] A_s is used to indicate a subset of A with the allowed actions in state s

[3] This applies to situations where the state transition is deterministic or where the state transition could be divided into multiple steps where a reward is granted before probabilistic effects occur. For example, in a game where the player first rolls a dice and then choses his action, the result of the action is deterministic, but there is a probabilistic element before he can chose the next action.

A strategy[4] π is a selection of possible actions for each state. Actions may be selected deterministic or with individual probabilities for each state. Here we only use *stationary strategies* where the chosen actions only depend on the current state and not on the history of events. π_s denotes the selected ruleset for a state s. For a deterministic rule we say $\pi_s = a\ ,\ a \in A_s$ and similary for a probability based selection of actions we say $\pi_s = \Delta(A_s)$.

In order to weight the importance of rewards, depending on the number of actions which are needed to reach them, a discount factor β is used, such that an expected reward which is t steps ahead is multiplied with β^t. Therefore the expected reward for a state s with a policy π can be expressed as its immediate reward and the sum of discounted expected rewards

$$v(s_0, \pi) = \sum_{t=0}^{\infty} \beta^t r_t$$

where $r_t \in R$ is the immediate reward for the state reached after t steps with the policy π starting at state $s_0 \in S$.

The value function can be transformed to a recursive aproach by combining the immediated reward with the expected values for the next state multiplied by the probability of reaching the specific next state.

$$v(s, \pi) = R_{\pi_s}(s) + \beta \sum_{s'} P_{\pi_s}(s, s') v(s', \pi)$$

An optimal policy π^* can be found by solving

$$v(s, \pi^*) = \max_a \left\{ R_a(s) + \beta \sum_{s'} P_a(s, s') v(s', \pi^*) \right\}$$

A possible learning goal could be to find an exact value or a good estimation for the $v(s, \pi)$ function or rather the $v(s, \pi^*)$ function, since the value of a state without an optimal strategy might not be interesting to the learning agent. Based on this value function, the agent can then extract a stategy π^* by always chosing actions which lead to a follow-up state with the highest estimated value.

2 Games and Learning methods

For research in the field of reinforcement learning, many concepts are evaluated by the means of games. The used games range from simple deterministic tasks which are often specially designed for the learning agent to sophisticated games that are actually played by humans. One advantage of games is, that there are quite a few games present, so there is no need to start developing an interesting concept from scratch. Instead it is possible to select a game which is already accepted to have reasonable rules and requires some kind of player strategy in order to archieve good results. It is also possible to add or remove some game features in order to fit the game with the learning agent if there are some restrictions on the model the agent relies on.

To comply with the Markov properties, sometimes additional information has to be coded into the game states, which is normally part of the game move history. For example this might apply to chess, where the possibilities of castling and passing strikes lead to situations where a board position refers to different game states depending on previous moves of the involved chess pieces.

The complexity of chosen games also depends on the learning system properties. While the practical restrictions of learning with lookup tables or similar techniques often require relatively simple games where the agent may learn some form of perfect playing strategy, the utilization of estimative approaches like non-linear function learning allows for bigger games where enumeration of the whole state space is not an option anymore.

[4] Also refered to as policy

Backgammon game

Since backgammon is used in one of the examples below, here is a short overview of the game. It is played by two players, identified as black and white player. The game consists of an one dimensional track with 24 fields. Each field may contain zero or more gamepieces, named checkers, from the same player but never checkers from both players at the same time. However, if a field contains only one enemy checker it is possible to land a hit and move it to the bar so that the enemy player has to bring it back into the game in order to finish. The track is divided into a home board per player and an outer board in the middle of the track. Each player starts with 15 checkers which are initially in a fixed starting position. The goal is to move all own checkers into the players home board and get them off the track afterwards. The player who is first at removing all his checkers from the track wins the game. If one player wins and the other one did not remove any of his checkers from the game this is called a gammon and the result is doubled. If one player wins a gammon and the other player still has checkers in his starting space or bar this is called a backgammon and is counted three times. Please refer to a game manual for a complete description of the rules.

Regarding a learning algorithm the state requires an input encoding which covers all the fields on the track with the number of checkers on them as well as information about the checkers which where hit and those which are already brought home. The output encoding must specify the winner and some additional information to distinguish between normal win, gammon and backgammon.

Temporal difference learning in TD-Gammon

The problem which frequently occurs in games as well as other real world applications is, that a reward is not given immediately after a decision is made but instead there is a sequence of decisions, which leads to one reward in the end. The challenge with this temporal difference between decisions and the related feedback is, to distribute the reward among all involved decisions to improve the accuracy of the value function. This concept is refered as temporal difference learning. An application for this concept is presented in TD-Gammon [4], which is a backgammon playing agent that was trained through reinforcement learning methods with temporal difference learning using the $TD(\lambda)$ algorithm which was introduced by Sutton 1988 [3]. It utilizes a neural network which is organized in a multilayer perception architecture as shown in Figure 1 to learn the value function from the game state inputs. The neural network is provided with an input pattern X, encoding the current game state and an output pattern Y encoding the estimated expected outcome of the game. The output pattern consists of four elements, encoding a normal win or gammon for eigher the white or black player. Backgammons are not separately considered in the learning progress since they only occur on rare occasions.

The actual learning process is implemented as an update to the weights w of the neural network edges based on current information, where Y is the output pattern, α is the learning rate, λ is the discount factor[5] and $\nabla_w Y_k$ is the gradient of the network output.

$$w_{t+1} - w_t = \alpha(Y_{t+1} - Y_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w Y_k$$

When the game ends, a final value signal which represents the actual result is used instead of Y_{t+1}.

Tesauro experimented with a variety of configurations with 40 and 80 hidden nodes where he utilized different input encodings. The most basic input encoding was a raw board encoding with information about the number of checkers on each position. Other configurations included a number of precoded concepts like blocking positions which Tesauro previously used in other backgammon learning programs on the base of supervised learning. Considering the training results, the TD-Gammon programs seems to scale very well with the number of hidden nodes as well as the encoding of additional information about the game state. Depending on the provided information and possibilities the learning speed and resulting accuracy changed accordingly.

The utilization of estimation strategies together with other techniques to maintain the learning progress may be a good entry point for learning agents in a real world environment. This assump-

[5] Refered to as β in other definitions

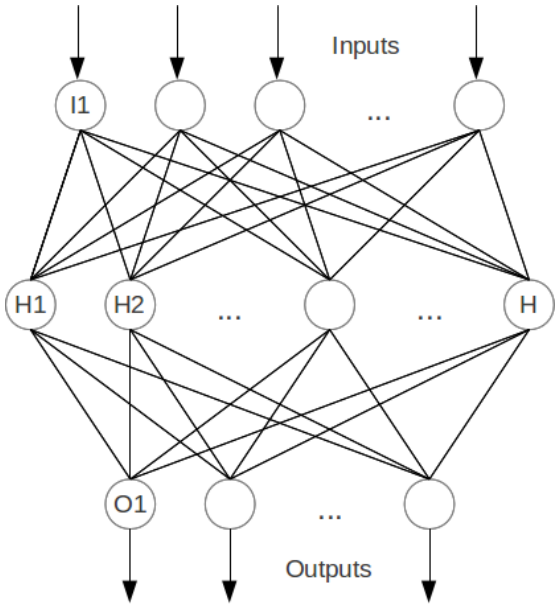

Figure 1: Illustration of a neural network as used in TD-Gammon and many other applications. The inner nodes H1..Hn are the hidden nodes and may be arranged in one or more layers. Figure reproduced after [4]

tion is supported by the positive experiences with TD-Gammon, which showed good results in the attempt to learn backgammon game patterns and move decisions without any prior knowledge other than the boundaries given by the rules of backgammon. The observation, that basic playing concepts settled in the underlying neural network of TD-Gammon after only a few games and where later refined with more sophisticated strategies suggests that the underlying model is capable of adapting to any environment where inputs are well defined and the possible feedback strenghtens good decisions.

One of the properties which indicates a high relevance towards real world applications is, that TD-learning requires only an implementation of rules which describe all possible actions in a specific state, a detailed representation of the state itself and some sort of feedback, to learn the consequences of recent decisions. The part that is left out here is any form of initial strategy or fixed knowledge about good decisions. Instead, all this strategic knowledge is learned through repeated trial and error of the learning agent itself. Due to this independent learning process the agent is not limited by the knowledge which is already known to its developers or other data sources but instead it may be capable to discover solutions which differ from expectations but prove to be reasonable once they are investigated further. This is also reflected in the fact, that TD-Gammon not only reached a level of backgammon play which approaches the worlds best human players, but also changed the common strategy of the worlds best players in some aspects, since the decisions of TD-Gammon proved to provide a higher success rate than the former stategies [6].

Q-Learning

Based on the basic concept of reinforcement learning, the Q-Learning algorithm is designed as an iterative solution to find the value function v. It was first introduced 1989 by Watkins [5]. It works by first learning a function Q where the perfectly learned function Q^* equals

[6]Evaluations of TD-Gammons strength include game series against world champion players as well as extensive computer analysis of difficult game positions. Computer analysis was done with rollouts, where the current board state is the starting point for a few thousand games with random rolls and a program deciding on the moves. This method is considered to provide a good insight on the strength of a position even if the program which plays the rollout is only on intermediate level

$$Q^*(s,a) = R_a(s) + \beta \sum_{s'} P_a(s,s') v(s', \pi^*).$$

The value function and therefore also the optimal strategy can then be computed from the Q^* function

$$v(s, \pi^*) = \max_a \{Q^*(s,a)\}.$$

The remaining problem is the computation of the Q^* function. This is done with an updating procedure which improves a given initial guess Q_0 for each state s and the learning rate sequence α:

$$Q_{i+1}(s,a) = (1-\alpha_i) Q_i(s,a) + \alpha_i \left[r_i + \beta \max_{a'} Q_i(s', a') \right]$$

where s' is the state which follows state s with action a.

Multi agent systems

Another extension to the basic concept of reinforcement learning is the utilization of multiple agent systems, where agents can improve the overall reward if they cooperate with other agents or, in case of a competitive game, can improve their individual reward by finding an optimal response to other agents strategies. The difficulty of this approach is, that an agent has to learn about other agents strategy in order to improve the overall result further than what the individual optimal strategy of each agent would archieve. To archieve this goal, the agent has to maintain its own strategy as well as its view on the other agents strategies.

In order to archieve some sort of cooperation in multi agent systems, Hu and Wellman [1] utilize the concepts of Q-Learning and well as the concept of Equilibrium Points which was first introduced by Nash, 1951 [2]. The idea behind this is, to find an optimal cooperative strategy for multiple agents by selecting an agents behavior not based on its own maximized reward but instead on the maximized overall reward for all involved agents.

In a n agent environment, the value function for agent i can be expressed as $v^i(s, \pi^1, \ldots, \pi^n)$, which the agent tries to maximize. The Nash Equilibrium is defined as a set of strategies, where each strategy is the best response to the other agents strategies, provided the otherstrategies do not change:

$$v^i(s, \pi^{*1}, \ldots, \pi^{*n}) \geq v^i(s, \pi^{*1}, \ldots, \pi^i \ldots, \pi^{*n})) \text{ for all } \pi^i \in \Pi^i$$

with Π^i as set of possible strategies for agent i.

While this definition leads to a locally optimal response of an agent to the strategy of other agents, it does not consider the posibilities of cooperative learning. So, if a strategy requires another agent to do (or refrain from) some actions, it would be interesting to motivate the desired behavior by influencing the other agents learning progress. However, since the Nash Equilibrium Point is a best response to other fixed strategies, results which require cooperative learning can not be obtained.

Grid-World Game examples with Nash Q-Learning

There are two games presented by Hu and Wellman which both appear on a 3x3 grid with two agents on them. The second of those games will be discussed in the following. The game starts in a state where the agents are located in the lower right and left corner of the field. The upper mid field contains a goal which both agents want to reach. An agent chooses one possible direction from $\{up, down, left, right\}$ where it wants to move to. When both agents try to move to the same field they get a penality of $r_t = -1$ and are retransfered to their previous position. The goal field however allows a concurrent reaching of the agents. When the first agent reaches the goal it is granted a reward of $r_t = 100$. If both agents reach the goal at the same time they are both rewarded, else the second agent will not get any reward. Additionally there is some probabilistic

element in this game. When trying to go up from the lower corner fields there is a probability of 50% to succeed. Otherwise the agent is transfered back to the previous position. The described game setting is shown in figure 2. Since there is an opportunity for both agents to receive their reward simultaneously and there is no penality for an agent when other agents receive an reward, this setting suggests a cooperative behavior to some extent. However, keep in mind that the slower agent will not receive a reward once one of the agents reaches the goal before the other one.

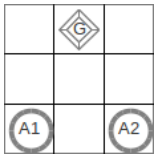

Figure 2: Illustration of the grid game by Hu and Wellman. Figure reproduced from [1]

Let's assume that each agent has already evaluated his pure strategies, without considering the other agents moves. The remaining challenge is now in finding an Equilibrium Point where the agents improve their performance compared to their pure strategies. For most game states there is a clear cut strategy how to reach the goal as fast as possible and without any collision between the agents. Only the agents starting position is different due to the probabilistic element when going up and the possible collision when going $right, left$. Possible action choices $(a1, a2)$ of the two agents are $(right, left)$, $(up, left)$, $(right, up)$ and (up, up). While the $(right, left)$ choice obviously leads to a dead end and the values for (up, up) are far from optimal, the other alternatives favor the agent which chooses the side step over going upwards. Since none of these choices are optimal, Hu and Wellman present another mixed strategy where actions are taken probabilistically. They define the probability for going up as 0.03 and the probability for going side way as 0.97. With this the $(right, left)$ situation is no longer an infinite hanging point. However, the problem remains that there are multiple Equilibrium Points which all involve the possibility of missing the reward for one of the players.

As mentioned earlier, the presented concept only allows the agents to learn how to react to other agents actions but it does not provide the possibility to have a cooperative learning part.

Sticking to the presented grid game, the optimal strategy for agent 1 would be to go $right$ in the first step if agent 2 choses the up action. However, this would lead to a situation where agent 2 does not get any reward in 50% of the games, so this is a very one sided strategy. With a discount factor near 1 the number of steps taken to reach the goal is not as important as avoiding collisions, so as long as the agent reaches the goal it would not be a problem to take some extra steps. Therefore it might be a good concept for agent 1 to increase agent 2's reward while making sure that the own strategy always reaches the goal first. This would improve the chances that agent 2 learns a strategy which allows agent 1 to have its way.

Starting at the initial positions with the moves $(right, up)$ there is a 50% possibility that both agents can reach the goal in an optimal number of steps. Otherwise agent 2 will still be in its starting position. In this case the next move should be $(up, *)$. In case agent 2 choses $left$ or succeeds in going up, the two agents are out of sync and only one will reach the goal unless they go back to their starting position to reenact the probabilistic element. However, if agent 2 choses up and is retransfered again, then there is a deterministic way to reach the goal at the same time. However, this requires agent 1 to leave its optimal strategy and take two extra steps. Finishing the game with $(left, left), (up, up), (right, up)$. As mentioned, for agent 1 this means a 100% win strategy but with a 25% probability of taking two extra steps. Agent 2 however gets a winning probability of 75% since it wins both on initial success (50%) and on failing the up action twice (25%).

While these values might not yet be an optimal strategy, they are an improvement over the variations which the Nash Q-Learning in the given environment archieve. It would be interesting to see, whether different settings produce better results even with probabilistic games.

3 Conclusion

Considering the intuitive form of learning concept behind reinforcement learning, specially the potential which is shown in TD-Gammon, suggests a lot of opportunities for further research and applications. However, to open this subject to more problem fields it might be necessary to find a solution to some of the restrictions which are associated with the presented techniques. Considering the nature of real world problems, one of the most important differences might be the observability of the environment. Therefore it would be interesting to see how TD-Learning with neural net may work without a complete model of the world but selective inputs which indicate a local environment.

The Nash Q-Learning games present an approach towards cooperative actions of multiple agents. However, since the concept comes with many restrictions on, what kind of environment they support and how Equilibrium Points are reached, there seems to be some way to go before the concept can actually improve over a hand-crafted cooperative strategy by humans.

References

[1] Junling Hu and Michael P. Wellman. Nash q-learning for general-sum stochastic games. *J. Mach. Learn. Res.*, 4:1039–1069, December 2003.

[2] J. Nash. Non-cooperative games. *Annals of mathematics*, 54(2):286–295, 1951.

[3] R.S. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

[4] G. Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[5] C.J.C.H. Watkins. *Learning from delayed rewards*. PhD thesis, PhD thesis, Kings College, 1989.