Reinforcement Learning - Agents Learn to Interact in Unknown Environments

Matthias Haselmaier[1], Alexander Schwarz[2], and Tim Hallyburton

[1] Kaiserslautern University of Applied Sciences
matthias.haselmaier@hs-kl.de
[2] Kaiserslautern University of Applied Sciences
alexander.schwarz@hs-kl.de

Abstract. Reinforcement Learning describes a machine learning paradigm which is applied to find an optimal sequence of actions to achieve a given goal. An agent receives an reward for performed actions and has to find a policy which maximizes the expected value of the sum of cumulative rewards. In this paper linear function approximation and SARSA Learning are explained and implemented. They are applied to the computer game Breakout. Experiments with different lengths of training episodes and different hyperparemeters were performed and the results presented. Because of the selected features very good results were achieved after only a short training periode.

Keywords: Machine Learning, Reinforcement Learning, SARSA Learning, Function Approximation, Temporal-Difference Learning

1 Introduction

Reinforcement Learning is a paradigm of machine learning. It is based on the idea to reward good actions and punish bad actions and tries to train an agent to perform sequential actions to achieve a given goal. One of its defining characteristics is the interaction between the agent and his surrounding environment. It is shown in figure ??.

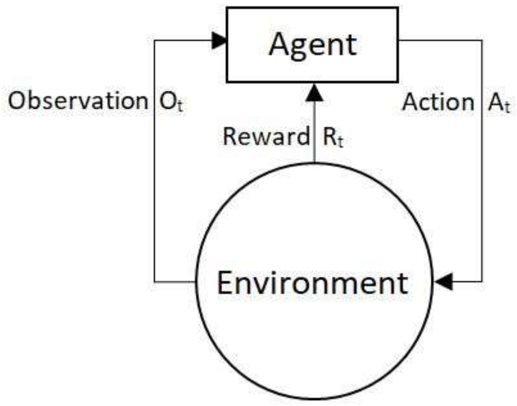

Fig. 1. The agent in his environment

With every action A_t the agent takes he is given a scalar *reward* R_t. He must use this *reward* to improve the *policy* he is using to choose his actions. Compared to other machine learning paradigms time is another defining charatistics of Reinforcement Learning, as the *reward* the agent receives for a specific action may possibly be received many timesteps after the agent performed the action. Furthermore the information or observation O_t the agent receives about its surroudings, which he has to make his decisions on, is not i.i.d. as it is influenced by his previous actions. If e.g. a small robot with a camera explores the right side of a room it may see drasticly different things as if he was to explore the left side of the room. To achieve his goal the agent has to find the *policy* that maximizes the expected value of the cumulative sum of *rewards*.

What happens in the environment surrounding the agent is based on the current state S_t. This state can be formulated as a function of the historie H_t, where H_t is the sequence of observations, rewards and actions until timestep t.

$$H_t = O_1,\ R_1,\ A_1,\ \ldots,\ A_{t-1},\ O_t,\ R_t \tag{1}$$

$$S_t = f(H_t) \tag{2}$$

This state S_t can be seperatet into the environment state and the agent state. The environment state may contain information that is unimportent or not accessible to the agent. The agent state includes everthing he needs to chose his next action. This also is the information that is used by the Reinforcement Learning algorithm. If the state S_t has the Markov property only the most recent state may be saved as there is no information lost.

Reinforcement Learning was proven to be successfull in a variaty of different problems, e.g. [?], [?], [?]. In this paper we apply Reinforcement Learning to the computer game Breakout and present a small software framework which can be used to implement a Reinforcement Learning algorithm. We will present the problem space in more detail first and explain the theoretical background of Reinforcement Learning. Afterwards we present our implementation and finally discuss the result of the applied Reinforcement Learning algorithm.

2 Breakout Application

The game Breakout was chosen as learning task. Breakout is a relatively simple game which was first published by Atari in 1976 as an arcade game. Basically Breakout is a grid of blocks at the top of the window, which have to be destroyed by a ball. The player tries to hit the ball with a paddle, which is located at the lower border of the window. The paddle can be moved left or right. The goal is to destroy all the blocks without losing your lives.
For this application a clone was implemented in Java. The grid consists of 90 blocks, which are arranged in 6 rows and 15 columns. If the ball hits the left, right or upper window border or one of the blocks, the exit angle corresponds to the entry angle. The player has the possibility to control the ball minimally. This happens when the paddle is in motion at the time of the collision with the ball. In this case the angle of the ball is deflected by 5 degrees in the respective direction. If the ball hits the resting paddle, the bounce behaves as with the walls and the blocks. At the beginning of each game, the player starts with 5 lives. If the ball is not returned upwards and hits the lower border of the window, the player loses one life. Each time a new life is started, both the ball and the paddle are placed in the middle of the window. At the beginning the ball always moves straight along the Y-axis to the bottom and in a random direction on the X axis. For each block hit by the ball, the player receives 5 points. Once all the blocks have been

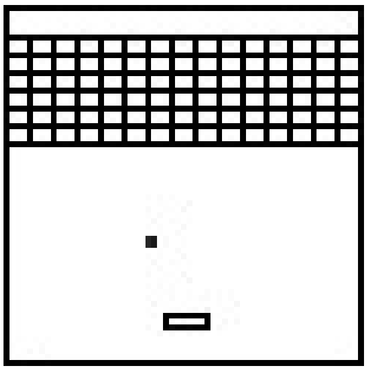

Fig. 2. Breakout-Application

destroyed, there is a bonus of 100 points and the game is reset like a life loss. At any time the player has the 3 possible actions stop, move left or move right.

3 Used Method and Algorithm

Since the state space of Breakout is too large for a tabular solution, the method of Value Function Approximation was used together with the SARSA Learning Algorithm. The SARSA algorithm belongs to the Temporal Difference Learning algorithms and uses the transitions from state-action pair to state-action. It is described by naming the tuple $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1})$. S is the current state, A the currently executed action, R the reward for this action, S_{t+1} the subsequent state and A_{t+1} the action in the subsequent state. The update rule used is the following:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\big] \tag{3}$$

The updates are performed after each transition from a non-terminal state. If S_{t+1} is a terminal state, $Q(S_{t+1}, A_{t+1})$ is set to zero.

Algorithm 1 SARSA-Algorithm

```
Initiazlize $Q(s,\ a), \forall s \in S,\ a \in A(s)$, arbitrarily, and $Q(\text{terminal-state},\ .) = 0$
Repeat (for each episode):
Initialize $S$
Choose $A$ from $S$ using the policy derived from $Q(e.g.,\ \epsilon$-greedy)
Repeat (for each step of episode):
Take action $A$, observe $R,\ S'$
Choose $A'$ from $S'$ using the policy derived from $Q(e.g.,\ \epsilon$-greedy)
$Q(S,\ A) \leftarrow Q(S,\ A) + a[R + \gamma Q(S',\ A') - Q(S,\ A)]$
$S \leftarrow S';\ A \leftarrow A';$
until $S$ is terminal
```

The Value Function Approximation approximates a state-action function q_π as a parameterizable function $\hat{q}$ with a feature vector w. The action-value function $\hat{q}$ is a linear combination of features:

$$\hat{q}(S, A, w) = x(S, A)^T w = \sum_{j=1}^{n} x_j(S, A) w_j \tag{4}$$

A feature is the description of a state action pair of the state space, e.g. how the distance from the paddle to the ball in the breakout example in state S changes due to action A. A state action pair is described by several features which are collected in a feature vector.

$$x(S,A) = \begin{pmatrix} x_1(S,A) \\ \vdots \\ x_n(S,A) \end{pmatrix} \tag{5}$$

The update function for the feature vector w, using the SARSA learning algorithm looks as follows.

$$\Delta w = \alpha(R_{t+1} + \gamma \hat{q}(S_{t+1}, A_{t+1}, w) - \hat{q}(S_t, A_t, w))\nabla_w \hat{q}(S_t, A_t, w) \tag{6}$$

The weight vector is now updated in each step.

4 Used Features

As a rather simple approach, the goal was to hit the ball with the paddle. The control over the direction of the ball is quite small, but if the ball is kept in play as long as possible, it is only a matter of time until all blocks are cleared. To achieve this goal, a feature has been implemented that maps the distance between the center of the ball and the paddle in the interval from 1 to 0. The closer the ball is to the paddle, the lower is the value of the feature. This caused the angle of the ball to become flatter over time, and since the ball is slightly faster than the paddle, the agent played himself in situations where he could no longer reach the ball. For this reason, an attempt was made to position the paddle in the middle of the playing field as soon as the ball was up to the blocks. For this a feature was implemented, which indicates the distance to the center of the playing field. The combination of these two features resulted in the agent being torn between the decision to minimize the distance to the ball or to the center, and usually just standing still. Several different combinations with other features were tested. The combination that performed best was formed from the first feature, the distance between the ball and the paddle, and a feature that maps the angle of the ball to an interval from 0 to 1, where 1 is a very steep and therefore preferred angle. The steep angle makes it easier for the agent to hit the ball again, as the paddle only needs to move minimally. Also, the distance feature has been adjusted to take a value of 0 if the ball will hit the paddle in its current trajectory if it doesn't move. Otherwise, the agent would try to further reduce the distance, making the angle of the ball flatter in many cases. The results from the next sections were obtained with this feature combination.

5 Results

The implemented framework served as a basis for several test series, which are now examined in more detail. In the first experiment the number of learning episodes was varied. With the resulting weights w_1, w_2 of the features f_1, f_2, 100 episodes were played. The average reward achieved during these episodes serves as a quality measure for the combination of w_1 and w_2. The remaining parameters were freely chosen and remained constant during the test series. They were $\alpha = 0,2$, $\gamma = 0,3$ and $\epsilon = 0,05$.

After just a few episodes, less than 10, good results were already achieved. This short training period was achieved because the chosen features were created with knowledge about the problem. Especially the distance from the ball to the paddle effected the fast learning process which resulted in the ball being hit almost every time. On average a reward of 1856 was achieved.

In further experiments, the parameters α, γ and ϵ were varied and the number of training episodes remained constant. Particularly interesting is the variation of ϵ in the range $\epsilon \in [0.02, 0.05]$. A higher ϵ-value can achieve better results if only a limited number of training episodes (here: 1000 episodes) are run through.

6 Conclusions

In summary, it can be said that the selected features can achieve very good results even after short training phases. However, the way these results are achieved can still be improved. Currently the agent avoids losing a ball and thus a "life" - but the remaining blocks are not considered when selecting the next action. The breaking of the blocks, and the associated reward, is therefore only a by-product of striving not to lose the ball. This causes the $\frac{Reward}{Time}$-ratio to deteriorate as the number of remaining blocks decreases. It would be more effective to destroy the blocks by targeting them. To achieve this a feature could be used that indicates whether the ball is currently under a block. The goal of such a feature would be to show the agent where a flat or steep angle in the field promises a better success.