In: Proc. BENELEARN-97, 7th Belgian-Dutch Conference on Machine Learning, pp 7-12,1997

A Short Introduction to Reinforcement Learning

Stephan ten Hagen* and Ben Kröse

Department of Mathematics, Computer Science, Physics and Astronomy
University of Amsterdam
Kruislaan 403, 1098 SJ Amsterdam
email: stephanh@wins.uva.nl

Abstract

This introduction is meant for readers with no knowledge about reinforcement learning. It presents the basic framework and introduce the basic terminology. We hope that this will make it easier to read other reinforcement learning literature. Pointers to more tutorial sources will be given at the end.

1 Introduction

In reinforcement learning an agent learns to optimize an interaction with a dynamic environment through trial and error. This seems to be a very natural approach to learning, because most of our skills we have learned in a similar way. So it got a lot of attention from many different research field, interested in learning mechanisms. This introduction will only focus on the computer science approach, where reinforcement learning is an optimization framework.

2 The Reinforcement Learning Framework

Besides the agent, the reinforcement learning framework consists of an environment and an optimization task.

2.1 The Environment

The agent is embedded in the environment and observes the environment's state s. This is the part of the environment that can change in time. The agent can influence the change of states by applying an action a to the environment. For this it receives an immediate reinforcement in the form of a scalar evaluation r. The assignment of reinforcements is regarded as part of the environment because it cannot be influenced directly by the agent.

Example 1 *A navigation task. Suppose the environment is a world, and the agent's position is the state. By taking an action the agent can change its position. Every time step the agent receives an immediate reinforcement $r = 1$, until some goal state is reached. Then the number of reinforcements received equals the amount of time spent to reach the goal. The task is to reach the goal as fast as possible.*

Example 2 *Playing a board game. Every possible position of the pieces on the board is a state. If it is the agents turn, it can make a move. By this it changes the state. It always receives a zero immediate reinforcement. Only at the end of the game it can receive either $r = 1$ if it wins, or $r = -1$ if it looses. The task of the agent is to try to win the game.*

*Supported by the Dutch technology foundation STW

In reinforcement learning the environment is usually expressed as a *Markov Decision Process* (MDP). The MDP defines a stochastic environment in which the changes are described by transition probabilities. If at the presents state s action a is applied to the environment, then $P^a_{ss'}$ is the probability that the next state is s'. So:

$$P^a_{ss'} = Pr\{s_{k+1} = s' | \, s_k = s, \, a_k = a\} \qquad \forall \, s, s' \in S, \, a \in A(s) \tag{1}$$

Where S is the set of all possible states and $A(s)$ is the set of all possible action when $s \in S$ is the present state.

In Example 1 the interaction comes to an end when the goal is reached and in Example 2 when the winner is known. So for some states the interaction terminates. These states, elements of $T \subset S$, are called absorbing states. Let N be the time step where the interaction terminates. If N is finite with probability one, then the MDP is called absorbing.

2.2 The Optimization Task

The task in reinforcement learning is to optimize the interaction with the environment. The only thing there is to optimize, is the action selection mechanism of the agent. By selecting the "right" action, the probability of a transition to the "right" next state can be increased. If s' is the "right" next state and if $P^a_{ss'} > P^{a'}_{ss'}$ then action a is a better choice than a'.

The action selection mechanism is called the *policy* π. Usually the policy is described as a function of the state. So for state s, $\pi(s)$ assigns selection probabilities to all elements of $A(s)$. The policy $\pi(s)$ can be improved by increasing the probability of selecting the best action. Finding the best action is not a trivial task. Future consequence of an action are not incorporated in the immediate reinforcement, but they can have a large influence on the overall performance of the interaction. This is referred to as a *temporal credit assignment problem*, or as a problem with a *delayed reward*.

This problem can be seen in Example 1 and Example 2. For all non-absorbing states the received immediate reinforcements are identical. Still it is possible to prefer one action over an other because of the consequences it has on future reinforcements. This is because the overall performance can be expressed as the sum over all received reinforcements. In Example 1 this corresponds with the number of time steps to reach the goal, in Example 2 this corresponds to winning or loosing[1]. So the optimization task is to minimize the sum for Example 1, or to maximize it for Example 2.

In order to optimize the interaction, the future consequences of a policy should be known. Given a policy π it is possible to define a *value function* V^π. It is defined as the expected sum of (discounted) future evaluations. For state s the value $V^\pi(s)$ is defined as:

$$V^\pi(s) = E_\pi\{\sum_{i=k}^{N-1} \gamma^{i-k} r_{i+1} \, | \, s_k = s\} \tag{2}$$

The discount factor $\gamma \in [0, 1]$ is used to weight future reinforcements. If $\gamma = 0$ this corresponds to the immediate reinforcement. If $\gamma = 1$ this corresponds to the expected sum of future reinforcements. So $\gamma = 1$ is a good choice for Example 1 and Example 2.

The solution to the optimization task is the optimal policy. This can now be defined using the value function. The optimal policy is the policy that maximizes (or minimizes) the value

[1]It might look silly to express this as a sum, but in this way it fits in the framework.

function. For Example 2 the optimal value function for state s can be defined as:

$$V^*(s) = V^{\pi^*}(s) = \max_{\pi} V^{\pi}(s) \tag{3}$$

The optimal value function exist and is unique if the description of the MDP does not change and if the s represents the complete state. Then the optimal policy π^* is deterministic, so it will select one action $a \in A(s)$. This means that in the optimization task, only deterministic policies have to be considered. So the notation $a = \pi(s)$ can be used to indicate that, action a is selected in state s according to policy π.

3 Solution Methods

The value function can be used to solve the optimization task. This section presents two dynamic programming methods and one statistical method.

3.1 Policy Iteration

One method to determine the value function is called policy evaluation. This is an iterative method to approximate V^{π} for a given policy π. The value function V^{π} can be used to improve the policy. The key idea behind *policy iteration* is to start with an arbitrary policy π_0. Then the policy evaluation and policy improvement steps are alternated until the optimal policy is found.

The idea of policy evaluation is to look only one step ahead in (2). Suppose that the value $V^{\pi}(s')$ of the next state s' is known. Then the value $V^{\pi}(s)$ of the present state s is the expected sum of the immediate reinforcement and $\gamma V^{\pi}(s')$. Policy evaluation starts by assigning values V_0 to every state. These values are used as the "known" values of the next state, to calculated the values of the present state. This results in an iterative procedure based on $V_{l+1}(s) = E_{\pi}\{r_{k+1} + \gamma V_l(s_{k+1}) | s_k = s\}$, where l is the iteration step.

The expectancy in the iteration can be calculated using the known transition probabilities $P_{ss'}^{\pi(s)}$ and their expected reinforcements $R_{ss'}^{\pi(s)}$. So value evaluation uses:

$$V_{l+1}(s) = \sum_{s'} P_{ss'}^{\pi(s)}(R_{ss'}^{\pi(s)} + \gamma V_l(s')) \quad \forall s \in S \tag{4}$$

It can be proven[2] that $\lim_{l\to\infty} V_l = V^{\pi}$. To be useful in policy iteration the policy evaluation has to be truncated after some iterations. For this a stopping condition is used.

The $\pi(s)$ in (4) indicates the action prescribed by policy π. This policy is improved by finding the action that maximizes the right hand side of (4).

$$\pi'(s) = \operatorname*{argmax}_a \left(\sum_{s'} P_{ss'}^a (R_{ss'}^a + \gamma V_m(s')) \right) \quad \forall s \in S \tag{5}$$

where m is the number of iterations used in policy evaluation.

If $\pi'(s) = \pi(s)$ for all states, then the policy is stable and optimal. If it is not stable, the policy iteration continuous.

[2] Assume that a lookup table is used to map the value function. Every state has its own entry

3.2 Value Iteration

In value iteration, the policy evaluation and policy improvement steps are combined. This means that in the update (4) the policy is used that maximizes the result. So the value function and policy are improved together:

$$\left.\begin{array}{l} V_{l+1}(s) = \max_a \left(\sum_{s'} P^a_{ss'}(R^a_{ss'} + \gamma V_l(s')) \right) \\ \pi_{l+1}(s) = \mathrm{argmax}_a \left(\sum_{s'} P^a_{ss'}(R^a_{ss'} + \gamma V_l(s')) \right) \end{array}\right\} \quad \forall s \in S \tag{6}$$

It can be proven that $\lim_{l\to\infty} V_l = V^*$ and $\lim_{l\to\infty} \pi_l = \pi^*$. This method is called value iteration, because it directly searches for the optimal value function.

3.3 Monte Carlo

In policy iteration and value iteration, the value function was calculated using the known probabilities $P^a_{ss'}$ and $R^a_{ss'}$. If those probabilities are not known the value function can be learned. Given a fixed policy π, several complete iterations are performed. After termination of an interaction, all the reinforcements are known. The value $V^\pi(s)$ can be approximated by taken the average over the known future reinforcements. Because these methods are trial-based, they are sometimes referred to as "Monte Carlo" methods.

Example 3 *Take a game, like in Example 2, and use a fixed policy. Play the game several times. Suppose that in four games $s_k = s$ for some time step k. Three of those games were lost, one game was won. Then $\frac{1-3}{4} = -\frac{1}{2}$ is the approximated value for $V^\pi(s)$. Now this state is considered to have a higher probability of loosing the game. So this state has to be avoided, which can be achieved by improving the policy.*

The trial-based method "knows" only the states that are visited during the interactions. So the value function is not calculated for the complete state space, as done by the dynamic programming methods. This can be an advantage, for very large state spaces. Large state spaces are very common in games, because of the combinatoric explosion in possible states. This is often referred to as the *curse of dimensionality.*

But searching through the complete state space also has an advantage. The optimal solution will not be "overlooked". In a trial-based method this will depend on the policy and the probabilities of the environment. So the policy has two functions. The first is to extract as much information as possible from the environment. The second is to optimize the interaction. This is called the *Exploration/Exploitation dilemma.*

4 Reinforcement Learning

In the reinforcement learning community, the solutions presented in the previous section, are also considered to be part of reinforcement learning. In spite of them having their origin in other research fields. Reinforcement learning addresses the same kind of problems, and some novel solution methods were introduced.

4.1 Temporal Difference Learning

Temporal difference learning is a method to approximate the value function. It combines aspects of the policy evaluation and the Monte Carlo method. Recall the idea of policy evaluation. The value of the present state can be expressed using the next immediate reinforcement and the value of the next state. Only if they do not agree with each other the value has to be updated.

The standard temporal difference update looks like:

$$V_{l+1}(s_t) = V_l(s_t) + \alpha(r_{t+1} + \gamma V_l(s_{t+1}) - V_l(s_t)) \tag{7}$$

where $\alpha > 0$ is the learning rate. It is clear that this can only be calculated as soon as s_{k+1} and r_{k+1} are known.

Temporal difference learning has a computational advantage over the Monte Carlo method. This is because during the interaction, part of the update already can be calculated. In the Monte Carlo method the calculation can only take place after the interaction. Another advantage is that information from previous interactions can be used during the interaction. In (7) the value function V_l is the one from the previous interaction. The Monte Carlo method does not use any gained information.

Example 4 *Suppose a task consists of a sequence of subtask (the states). The value $V^\pi(s)$ of the states indicate the time to finish the complete task (assume $\gamma = 1$). If a "hick-up" happens during the first task, then it affects the total time to complete the task. This hick-up will effect all the training data if the Monte Carlo method was used. Using temporal difference learning, the hick-up only has effect on the approximated value of the first state.*

In Example 4 the hick-up appeared in the first state. But if it would appear in a later state, it should have effect on the approximated values of previous states. For this reason the temporal difference update (7) can be changed so that the update has effect on all previous states. Usually not all previous states are updated. A discount factor $\lambda \in [0, 1]$ is used. The longer ago the state was visited, the less it will be effected by the present update. Using this method is called $TD(\lambda)$ learning.

4.2 Q-Learning

Using the value function to solve the optimization problem results in a dual optimization task. Two mappings have to be optimized. One mapping for the policy, this is called the actor because it determines which actions to take. The other mapping for the value function, this is called the critic. Therefore all these solution methods are referred to as *actor/critic* methods.

It is possible to combine the actor and critic into one mapping. This is called the Q-function. It can be defined as:

$$Q^\pi(s, a) = E_\pi\{\sum_{i=k}^{N-1} \gamma^{i-k} r_{i+1} | s_k = s, a_k = a\} \tag{8}$$

Or it can be defined using the value function:

$$Q^\pi(s, a) = \sum_{s'} P^a_{ss'}(R^a_{ss'} + \gamma V^\pi(s)) \tag{9}$$

The advantage of Q-learning is that it only requires one mapping ($S \times A \to \mathrm{I\!R}$). The actor critic mapping requires two mapping: one $S \to \mathrm{I\!R}$ and the other $S \to A$. It is clear that the Q-function has an advantage if the state space is large. This method is the most popular in combination with temporal difference learning.

4.3 Applications

Using a MDP to describe the environment, seems to limit the possible applications of reinforcement learning. This is not true, because the MDP is only used to get a theoretic understanding of the different algorithms. In most cases it is enough to assume that the environment is a MDP. If the state space is too large to use dynamic programming, or if the environment is not completely known, then reinforcement learning provides methods to solve the optimization problem.

Reinforcement learning has been applied to solve many different discrete optimization problems. Solutions were found for resource allocation problems, scheduling problems, process control and adaptive games. One of the most impressive application was, learning to play backgammon.

Example 5 *TD-gammon is a backgammon playing program. It was trained using "self-play". So two agents optimized their policy by playing against each other. The board position was represented by a feature vector. This was the input for the actor and the critic, which were implemented as multi layer perceptrons. The TD(λ) learning method was used. Only at the end of the game the agents received a non-zero reinforcement (-1 or 1). After more than a million training games, the program played at world championship level!*

In Example 5 the solution was found for a problem with a very large state space (all board position). It also shows that it is possible to store the mappings in compact representations like artificial neural networks. For such representations there is no guarantee that it will find the optimal policy. Still, the resulting program performed extremely well. However, nobody understands exactly how this program was able to play at such a high level.

5 Conclusion

Reinforcement learning provides solution methods for dynamic optimization problems. It bridges the gap between dynamic programming and Monte Carlo methods. By incorporating advantages of both of them, for more problems it is feasible to find a solution.

References

[1] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-dynamic Programming*. Athena Scientific, 1996.

[2] Mance E. Harmon. Reinforcement learning: A tutorial. http://eureka1.aa.wpafb.af.mil/rltutorial/.

[3] Leslie P. Kaebling, Michael L. Littmann, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4, 1996.

[4] S. Sathya Keerthi and B. Ravindran. A tutorial survey of reinforcement learning. *Sadhana*, 19(6):851–889, 1994.

[5] Satinder Singh, Peter Norvig, and David Cohn. How to make software agents do the right thing: An introduction to RL. http://envy.cs.umass.edu/People/singh/RLMasses/RL.html.

[6] Richard S. Sutton and Andrew G. Barto. Reinforcement learning: Course notes. ftp://ftp.cs.umass.edu/pub/anw/pub/sutton/IntroRL/{RL1.ps,RL2.ps,RL3.ps}.