University of Illinois Fall 2018

ECE 544NA: Pattern Recognition
Lecture 23: November 13

Lecturer: Alexander Schwing Scribe: Shiyi Yang

1 Overview

Previous lectures have introduced the discriminative and generative learning whose main underlying theorem is the maximum likelihood. The parameters in those models will be optimized as to maximize the probability of the collected sample data. This lecture illustrates another machine learning paradigm which, instead of studying the maximum likelihood estimation, deals with how an agent seeks actions to maximize the future expected reward in an environment.

The goals of this lecture:

1. Getting to know reinforcement learning
2. Getting to understand Markove Decision Process

This lecture will describe the reinforcement learning (RL) and discuss the Markov Decision Process (MDP) by going through graphical representation of MDP examples. The analysis of exhaustive research, policy iteration and value iteration will then be introduced, and the difference between policy evaluation and policy iteration will be emphasized. At the end of this lecture, the features of reinforcement learning and how to use the MDPs should be grasped.

2 Task Formulation

Reinforcement learning consists of parameters including agent, actions, rewards and environment. The agent is the training target. The actions refer to all possible moves the agent could make at given state. Rewards illustrate the feedback of the agent's actions, and the environment is the medium which the agent interacts with.

In a classic reinforcement learning process as shown in Figure 1, an agent interacts with the environment and update its information in a discrete time step. At each time step t, the agent will receive observation signal O_t which indicates the state information S_t. Then, the agent will take actions based upon the given state S_t and receive a reward r_t from the environment. Subsequently, the agent will end up with a new state S_{t+1}, and receive an updated observation signal O_{t+1}.

Figure 2 demonstrates an example of the RL task. In a 3×4 grid map, the robot finds itself in state $(1, 3)$. The robot could choose to move left, move up or move right with a 0 reward in this state. When the robot ends up with state $(2, 4)$, it will receive a -1 reward, and when the robot ends up with state $(3, 4)$, it will receive a $+1$ reward as feedback.

2.1 Task Settings

The actions taken by the agent are stochastic in the environment. According to Figure 3, instead of moving upward in the top figure, the robot could turn left, move upward or turn right with

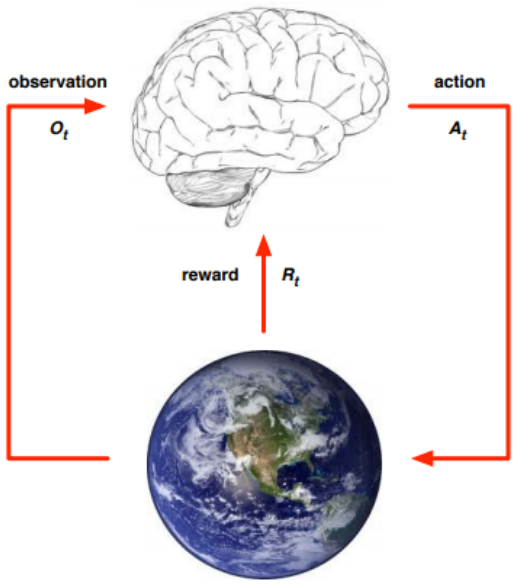

Figure 1: Task formulation of reinforcement learning.

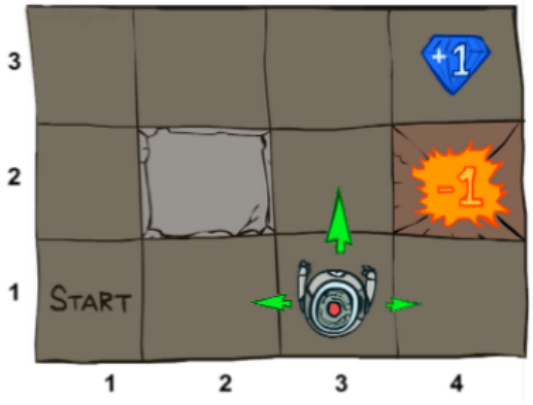

Figure 2: An example of reinforcement learning task.

certain probability. The transition probability in Section 3 will describe the stochastic setting in the reinforcement learning.

2.2 Reinforcement learning examples

1. Fly stunt manoeuvres in a helicopter
2. Play Atari games
3. Defeat the world champions at Go
4. Manage investment portfolio
5. Control a power station
6. Make a humanoid robot walk

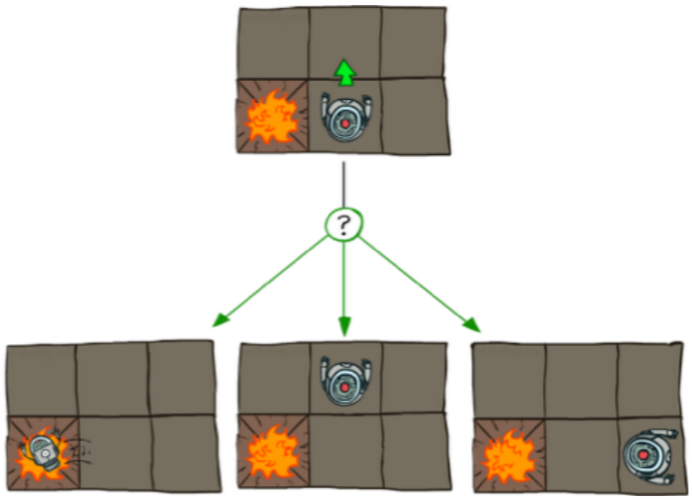

Figure 3: Stochastic setting of the robot task.

3 Markov Decision Process

The framework of the reinforcement learning is the Markov Decision Process (MDP). The MDP describes a discrete time stochastic process and it contains the following parameters:

1. A set of states s $\in S$
2. A set of actions based upon given state a $\in A_s$
3. A transition probability $P(s'|s,a)$
4. A reward function $R(s,a,s')$
5. A start and a terminal state

The Markov Decision Process is a markov random process in which the future is independent of the past given the current state. Mathematically:

$$\begin{aligned} P(S_{t+1} = s'|S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1} = ..., S_0 = s_0) \\ = P(S_{t+1} = s'|S_t = s_t, A_t = a_t) \end{aligned} \tag{1}$$

3.1 Graphical Representation of MDP

The finite state machine could describe the pictorial representation of MDP. In Figure 4, S_0, S_1 and S_2 represent three states of the agent, and a_0 and a_1 shows the possible actions at each state. The values following actions refer to the transition probability that the current state can transit to. The -1 and +5 illustrate the scalar reward for the actions the agent takes in particular state.

Suppose the current state is S_1, the two possible actions are a_0 and a_1 respectively. When action a_1 is chosen, there is 0.95 probability that $S_{t+1} = S_1$, and 0.05 chance that $S_{t+1} = S_2$. When the agent takes action a_0, the probability of $S_{t+1} = S_0$ is 0.7, the probability of $S_{t+1} = S_1$ is 0.1, and the probability of $S_{t+1} = S_2$ is 0.2. Note that when the agent is ending up with state S_0 from state S_1 by taking action a_0, it will receive a reward $R(S_1, a_0, S_0) = 5$, and when the agent is ending up with state S_0 from state S_2 by taking action a_1, it will receive a reward $R(S_2, S_0, a_1) = -1$.

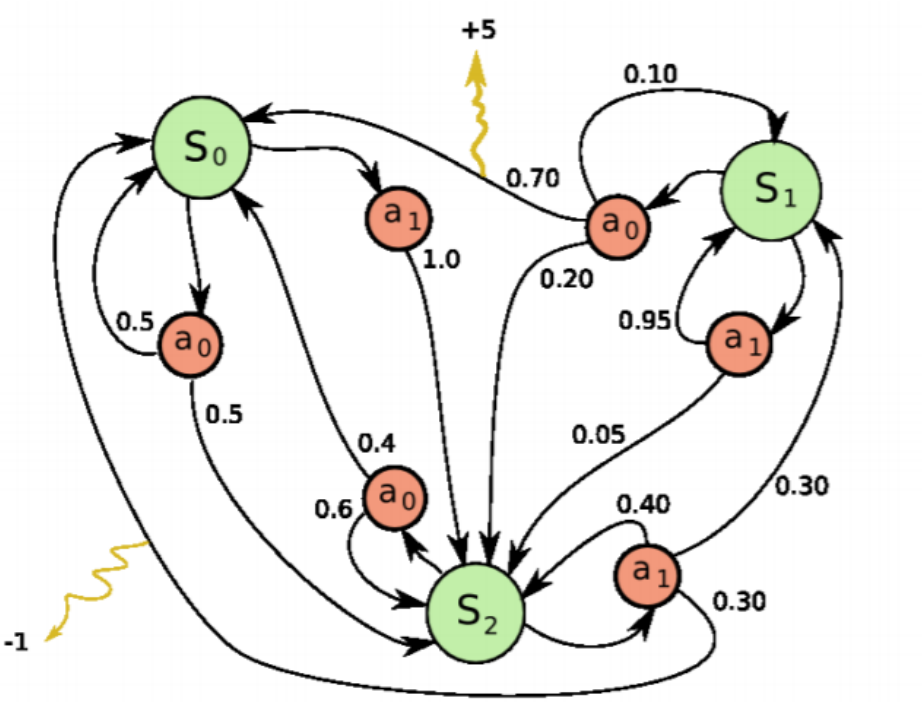

Figure 4: Graphical representation of Markov Decision Process.

4 Policy Optimization

The policy $\pi(s)$ determines the actions of the agent at a given state.

$$\pi(s) : S \to A_s \tag{2}$$

The solution to the MDP is the optimal policy $\pi^*(s)$ which maximizes the expected future reward at a particular state. The policy takes the state as input, and determines the actions the agent performs as output such that the highest reward could be collected in current state. There are three algorithms to investigate to illustrate how to obtain the optimal policy $\pi^*(s)$ in the system.

4.1 Exhaustive Search

The exhaustive search inspects all possible policies, and compute the expected future reward $V^{\pi}(s)$ for each of the policy. The best policy π^* is the one which maximizes the expected future reward $V^{\pi^*}(s)$ among all policies investigated. For a given MDP, the total policies are:

$$\prod_{s \in S} |A_s|. \tag{3}$$

For each policy, the expected future reward $V^{\pi}(s)$ will be computed by policy evaluation. Figure 5 describes an example of how to use exhaustive search to derive the optimal policy in a MDP problem. In this example, all possible policies will be examined, and the expected future reward for each policy will be generated by policy evaluation.

- Let the policy be $\pi(s_0) = a_1, \pi(s_1) = a_1$.

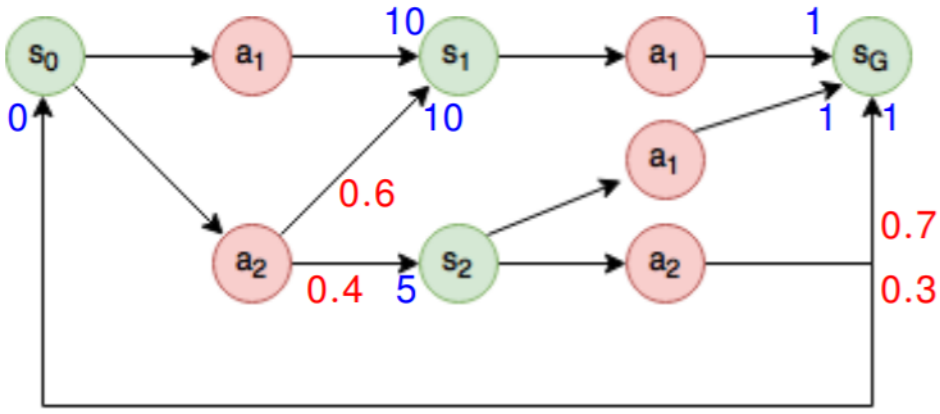

Figure 5: A MDP to illustrate policy optimization by using exhaustive search.

The action the agent could perform is deterministic in state s_1, when action a_1 is chosen in state s_0, the policy graph is shown in Figure 6. The future expected value is derived as:

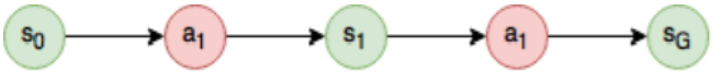

Figure 6: Policy graph when $\pi(s_0) = a_1, \pi(s_1) = a_1$

$$V^{\pi}(s_1) = 1$$
$$V^{\pi}(s_0) = 11$$

- When the policy is $\pi(s_0) = a_2, \pi(s_2) = a_1$,

  if the agent takes action a_2 in state s_0, the new state could either be s_1 or s_2. The transitional probabilities of $P(s_1|s_0, a_1)$ and $P(s_2|s_0, a_1)$ are 0.6 and 0.4 respectively. Then, the policy graph is depicted in Figure 7, and the future expected value is calculated through back-propagation from state s_1 and s_2 to s_0:

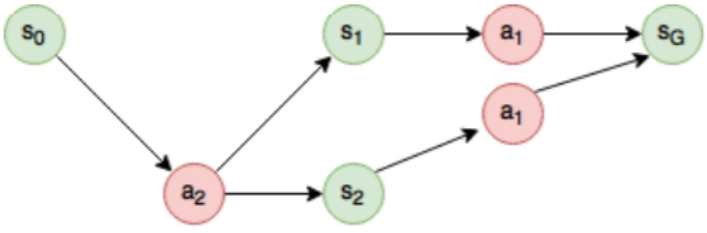

Figure 7: Policy graph when $\pi(s_0) = a_2, \pi(s_2) = a_1$

$$V^{\pi}(s_1) = 1$$
$$V^{\pi}(s_2) = 1$$
$$V^{\pi}(s_0) = 0.6 \times (10 + 1) + 0.4 \times (5 + 1) = 9$$

- As the policy is $\pi(s_0) = a_2, \pi(s_2) = a_2$,

  The expected reward at state s_2 is the sum of product of transitional probability $P(s_G|s_2, a_2)$ with the reward value $R(s_2, a_2, s_G)$ and the transitional probability $P(s_0|s_2, a_2)$ with the expected value $V^{\pi}(s_0)$. Similarly, the expected reward value at state s_0 could be expressed as a function of expected reward of $V^{\pi}(s_1)$ and $V^{\pi}(s_2)$. Then, the policy graph is described in Figure 8, and the expected future reward is given by:

Figure 8: Policy graph when $\pi(s_0) = a_2, \pi(s_2) = a_2$

$$V^{\pi}(s_1) = 1$$
$$V^{\pi}(s_2) = 0.7 + 0.3 \times V^{\pi}(s_0)$$
$$V^{\pi}(s_0) = 0.4 \times (5 + V^{\pi}(s_2)) + 0.6 \times (10 + V^{\pi}(s_1))$$

This could be expressed as a linear system of equations:

$$\begin{bmatrix} 1 & -0.4 & -0.6 \\ 0 & 1 & 0 \\ -0.3 & 0 & 1 \end{bmatrix} \begin{bmatrix} V^{\pi}(s_0) \\ V^{\pi}(s_1) \\ V^{\pi}(s_2) \end{bmatrix} = \begin{bmatrix} 8 \\ 1 \\ 0.7 \end{bmatrix} \tag{4}$$

The solution to the linear system of equations is:

$$\begin{bmatrix} V^{\pi}(s_0) \\ V^{\pi}(s_1) \\ V^{\pi}(s_2) \end{bmatrix} = \begin{bmatrix} 10.09 \\ 1 \\ 3.73 \end{bmatrix} \tag{5}$$

According to the policy evaluation, by comparing the reward at each state, the optimal policy at state s_0 is $\pi(s_0) = a_1, \pi(s_1) = a_1$, since the total expected reward value is 11. The optimal policy at state s_2 is $\pi(s_0) = a_2, \pi(s_2) = a_2$ with a total expected reward value 3.73. All the policies could be the best policy at state s_1 because the expected values are the same in these three policies.

For the exhaustive search, the policy evaluation will require to solve linear systems of equations in each policy. Equation 7 indicates the general case of the expected reward value for a particular

state, which is the summation over the product of the probability that the current state could transit to with the sum of the corresponding reward and the expected reward value in other states.

$$V^{\pi}(s) = 0 \qquad \text{if } s \in G \tag{6}$$

$$V^{\pi}(s) = \sum_{s' \in S} P(s'|s, \pi(s))[R(s, \pi(s), s') + V^{\pi}(s')] \tag{7}$$

4.2 Policy Iteration

The policy iteration requires policy evaluation and policy improvement. Instead of searching all possible policies, a policy π is initialized in the beginning of iteration, and will be refined in each step. When the policy π is initialized, the value of policy for all states will be computed by solving linear systems of equations. Then the policy is updated to π' through policy improvement to get new policy value $V^{\pi'}(s)$. Until the policy iteration converges, the optimal policy is derived. The policy iteration algorithm is summarized as follows:

- Initialize policy π
- Repeat until policy π does not change
  - Solve the systems of equations

$$V^{\pi}(s) = \sum_{s' \in S} P(s'|s, \pi(s))[R(s, \pi(s), s') + V^{\pi}(s')] \tag{8}$$

  - Extract new policy π

$$\pi(s) = \underset{a \in A_s}{\operatorname{argmax}} \sum_{s' \in S} P(s'|s, a)[R(s, a, s') + V^{\pi}(s')] \tag{9}$$

According to Howard [2], the policy iteration model is illustrated in Figure 9. Note that each policy is guaranteed to be a strict improvement over the previous one (unless it is already optimal), and because a finite MDP has only a finite number of policies, the policy iteration must converge to an optimal policy and optimal value. The proof is shown in [4].

4.3 Value Iteration

Value iteration algorithm concentrates on seeking the optimal value function instead of computing the policy. The general setup for value iteration is as following:

- Search the expected value instead of policy
- Compute the resulting policy after obtaining V^*

The optimal value function is approached by Bellman update:

$$V^*_{i+1}(s) \leftarrow \max_{a \in A_s} \sum_{s' \in S} P(s'|s, a)[R(s, a, s') + V^*_i(s')], \tag{10}$$

for ith step. When the value iteration converges, the optimal value function is characterized by the Bellman optimality principle [1]:

$$V^*(s) = \max_{a \in A_s} \underbrace{\sum_{s' \in S} P(s'|s, a)[R(s, a, s') + V^*(s')]}_{Q^*(s,a)}, \tag{11}$$

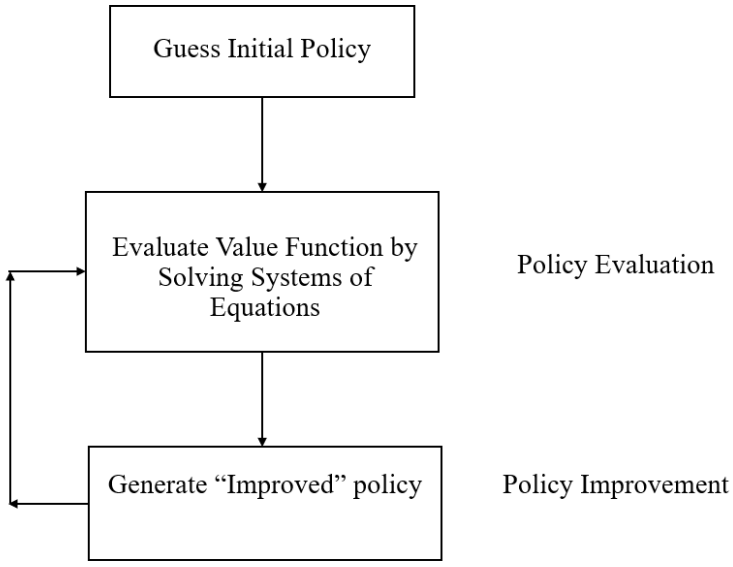

Figure 9: Block diagram of policy iteration

which could be rewritten as:

$$Q^*(s,a) = \sum_{s' \in S} P(s'|s,a)[R(s,a,s') + \max_{a \in A_{s'}} Q^*(s',a')], \tag{12}$$

where Q is the action value.

To extract the policy from the optimal value function and Q function, implement:

$$\pi(s) = \underset{a \in A_s}{\operatorname{argmax}} \sum_{s' \in S} P(s'|s,a)[R(s,a,s') + V^*(s')] \tag{13}$$

$$\pi(s) = \underset{a \in A_s}{\operatorname{argmax}}\, Q^*(s,a) \tag{14}$$

The advantage of the Q function while decoding the policy is that the calculation of Q function is slightly less complicated than that of the value function V^*, which is the reason why the Q function is preferred in seeking optimal policy in value iteration.

Note that the value iteration will always converge to the optimal value, and the proof of convergence is shown below [3]:

Theorem 1 *Value iteration converges to optimal value:* $\hat{V} \to V^*$.

Proof: For any estimate of the value function $\hat{V}$, the Bellman backup operator is defined as: $\Re^{|S|} \to \Re^{|S|}$,

$$B\hat{V}(s) = R(s) + \gamma \max_{a \in A_s} \sum_{s' \in S} P(s'|s,a)\hat{V}(s'), \tag{15}$$

where γ is the discount factor.
It will be shown that the Bellman operator is a contraction, that for any value estimate function V_1, V_2,

$$\max_{s \in S} |BV_1(s) - BV_2(s)| \leq \gamma \max_{s \in S} |V_1(s) - V_2(s)| \tag{16}$$

Since $BV^* = V^*$,

$$\max_{s \in S} |BV_1(s) - BV_2(s)| \leq \gamma \max_{s \in S} |V_1(s) - V_2(s)| \Rightarrow \hat{V} \rightarrow V^* \tag{17}$$

Proof of contraction property:

$$|BV_1(s) - BV_2(s)| = \gamma |\max_{a \in A_s} \sum_{s' \in S} P(s'|s,a) V_1(s') - \max_{a \in A_s} \sum_{s' \in S} P(s'|s,a) V_2(s')| \tag{18}$$

$$\leq \max_{a \in A_s} |\sum_{s' \in S} P(s'|s,a) V_1(s') - \sum_{s' \in S} P(s'|s,a) V_2(s')| \tag{19}$$

$$= \max_{a \in A_s} \sum_{s' \in S} P(s'|s,a) |V_1(s') - V_2(s')| \tag{20}$$

$$\leq \gamma \max_{s \in S} |V_1(s) - V_2(s)| \tag{21}$$

5 Quiz

1. What differentiates RL from supervised learning?

   - The reinforcement learning doesn't require supervisor. RL doesn't have labeled data and it only has reward signal.
   - The RL has a delayed feedback. The agent is likely to perform some actions which don't have reward signals. As a result, RL will not have a immediate feedback.
   - The actions the agent chooses will affect the received data. The interaction with the environment enables the agent to change the reward signal based upon different actions.

2. What is a MDP?

   The MDP is a markov random process in which the future is independent of past given the current state. The MDP consists of following parameters. A set of states $s \in S$, a set of actions based upon the state $a \in A_s$, a transition probability $P(s'|s,a)$, a reward function $R(s,a,s')$ and a start and a possible terminal state.

3. What differentiates policy iteration from policy evaluation?

   The policy iteration is an algorithm to search the optimal policy as to maximize the expected future reward at a given state. The policy evaluation is to calculate the expected future reward for a fixed policy $\pi(s)$ in a particular state.

6 Conclusion

In this lecture, the background of reinforcement learning and MDP has been introduced. Three algorithms have been discussed to seek the optimal policy in the MDP problems. For a particular MDP, the optimal policy could always be computed by exhaustive search. By implementing policy evaluation for all possible policies, the optimal policy could be derived by searching the one with the highest expected future reward. However, the exhaustive research is computational expensive. Then, the value iteration and policy iteration have been come up with to reduce the computational complexity. For policy iteration, the optimal policy could be approached by policy evaluation and policy improvements step by step. The policy evaluation will be implemented and the policy will be updated until the iteration converges. As for the value iteration, the optimal value function V^* will be derived when the iteration converges according to the Bellman's optimality equation, and the optimal policy could be then decoded from the corresponding value function or Q function.

References

[1] R. Bellman. A markovian decision process. *Indiana Univ. Math. J.*, 6:679–684, 1957.

[2] D. P. Bertsekas. Dynamic programming and optimal control 3rd edition, volume ii. *Belmont, MA: Athena Scientific*, 2011.

[3] J. Z. Kolter. Markov decision process, Feb 2016.

[4] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.