Neural Networks for Computing

Reinforcement Learning

Hikmat Farhat

August 20, 2018

Introduction

- Reinforcement learning usually involves a "learner" that needs to make a sequence of decisions to reach some goal
- The learner interacts with an environment by performing actions and observing their effects
- Actions by the learner have consequences such as the new state of the environment and rewards from the environment
- Reinforcement learning is learning to maximize the reward for a sequence of actions
- By observing the consequences of its actions the learner tries to reach a given goal
- Note that the learner is not supervised, learns from experience
- Reinforcement learning is based on Markov decision processes (MDP).

Agent and Environment

- Reinforcement learning is the problem of learning from interactions, to achieve a goal
- The decision maker (the learner) is called the *agent*
- Everything outside the *agent*, the things the agent *interacts with*, is called the *Environment*
- The *agent* continuously interacts with the *environment*
- The *agent* chooses an *action* and the environment responds to these actions by
  1. Changing its state
  2. Presents the agent with a *reward*
- The goal of the agent is to maximize the reward

Agent-Environment Interface

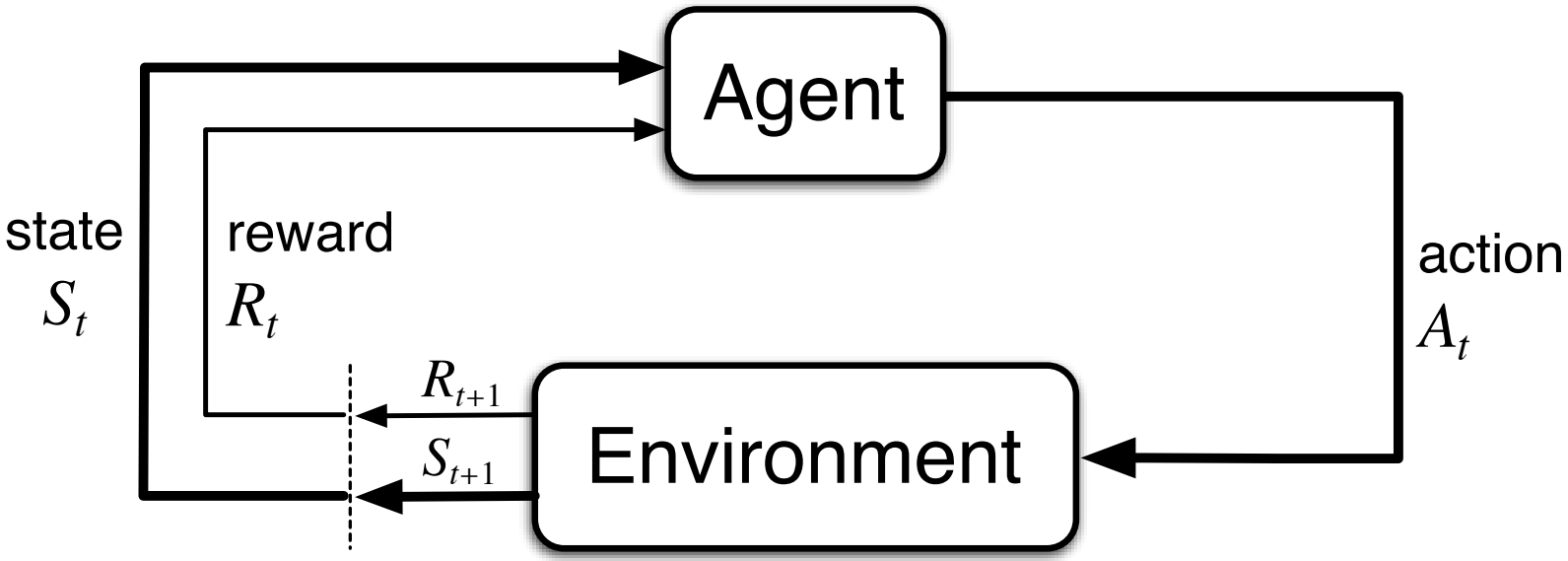


Figure: From Sutton et. al.

Agent-Environment Interface

- The agent-environment interaction is summarized by the figure above.
- The agent and environment interact in a *sequence* of discrete time steps: $t = 0, 1, 2, 3, \ldots$.
- At each time step t the agent receives information about the environment state $S_t \in \mathcal{S}$
- The agent chooses an action $A_t \in \mathcal{A}(S_t)$ form the set of allowed actions in state S_t.
- As a response the environment will move to state S_{t+1} and presents with agent with reward R_{t+1}

Agent Policy

- At each time step t the agent has to choose an action A_t.
- This choice depends on the state of the environment S_t and the time step t.
- In general the action of the agent is not deterministic: e.g. when in state S_t choose action a 50% of the time and action b 50% of the time
- The choice of the action is called the *policy* of the agent.
- $\pi_t(a \mid s)$ is the probability that action a is chosen when in state s at time t.
- The agent changes its policy as a result of experience
- Typically to maximize reward over the long run.

Example (Sutton et. al.)

- Consider a recycling robot that collects cans in an office and place them in onboard bin.
- The robot runs on rechargeable battery.
- The robot can do three actions: search for cans, put can in bin, recharge battery.
- Some states of the robot and actions along with reward:
  1. battery ok, no can $\rightarrow$ search: reward 0
  2. battery ok,found can $\rightarrow$ put in bin: reward 100
  3. battery low,_ $\rightarrow$ recharge : reward -100
- It is important to note that the battery level is part of the *environment*

Return and Rewards

- In reinforcement learning, the goal of the agent is to maximize some function of the rewards it receives from the environment for the sequence of actions
- At step t the agent receives a reward $R_t \in \mathbb{R}$
- The goal of the agent is to maximize the cumulative *reward in the long run*
- This goal can be stated as

Goal

The goal of the agent is to maximize the expected value of the cumulative sum of the received awards

- For example in the case of the recycling robot we might give the robot a reward of 0 for all steps except when it finds a can it gets a reward of $+1$
- It is important to note that the rewards should be setup in way such that maximizing the rewards will allow the agent to reach the goal
- Note that the reward should not reflect *how* the agent will achieve the goal
- The reward should tell the agent *what* to accomplish *not how* to accomplish it
- The reward is given *by the environment* to the agent
- In the case of animals(including humans) for example, when the animal is in a state of hunger it is rewarded for finding food.
- Note that the state of hunger and reward for food are part of the *environment*
- In this case the agent-environment interface is drawn such that the sensors for hunger and finding food are environment not part of the agent.

Returns

- We have mentioned that the goal is not to maximize the immediate reward R_t.
- But the *cumulative reward in the long run*.
- Given a sequence of rewards $R_t, R_{t+1}, \ldots$ we seek to maximize the *expected return*
- In other words the expected value of the return G_t which is some function of the reward sequence.
- In the simplest case we seek to maximize the expected value of

$$G_t = R_{t+1} + R_{t+2} + \ldots + R_T$$

- Where T is the final time step.
- In the above we assume the existence of a *terminal state*, as in the case of game playing, steps through a maze, etc...
- These sequences are called *episodes*.
- Each episode starts independently of how the previous one ended.

Discounting

- Sometimes the agent-environment interaction is continuous and does not end.
- In the example of the recycling robot there is no ending state unlike the case of game playing where the game terminates.
- In that case $T = \infty$ and the return becomes infinite.
- A *discounting* parameters, $0 \leq \gamma \leq 1$, is introduced into the return and the return becomes

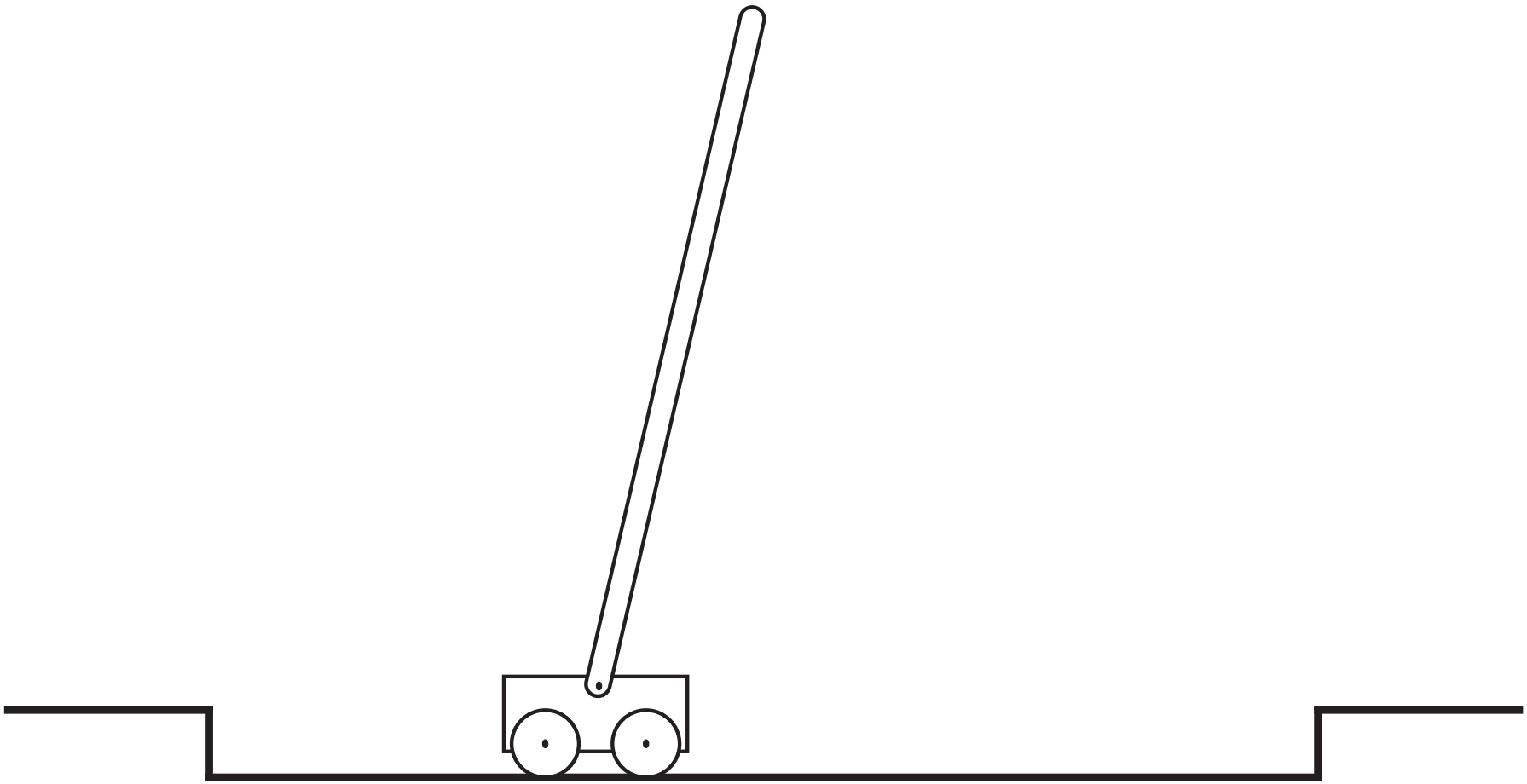
$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

Discounting

- The discount rate γ tells the agent the present value of future rewards
- A reward received k steps in the future is worth γ^{k-1} its immediate value.
- If $\gamma < 1$ then the return is finite even if there is an infinite sequence (as long as the rewards are bounded).
- If $\gamma = 0$ then the agent is "myopic", it will try to maximize the immediate reward.
- But in many situations maximizing the immediate reward will reduce the return (consider most drivers in heavy traffic in Lebanon)
- As $\gamma \to 1$ the agent becomes far sighted and takes into account future rewards more strongly.

Example (Sutton)

- Consider the example of an agent driving a cart with a pole hinged to it as shown in the figure below



Example cont.

- The objective is to keep the pole from falling (or its angle less then some angle α)
- A failure occurs when the angle of the pole is more than α. And the pole is reset to vertical after a failure.
- We can consider the task as episodic where each episode ends when a failure occurs.
- The agent is rewarded $+1$ for each step in which a failure did not occur. The return would be the total number of steps before failure.
- Alternatively, we can treat the problem as non-episodic by using a discounting factor γ.
- The reward is set to -1 after each failure and 0 otherwise. Suppose that the pole fails after K_1 steps and again after $K_2 > K_1$ steps.
- The return would be $-\gamma^{K_1} - \gamma^{K_2}$. The return is maximized by making K_1 and K_2 as much as possible
- In both models the return is maximized by keeping the pole from falling as long as possible

Combining Episodic and non-episodic

- In all cases we can write the return as

$$G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}$$

- and we can include the cases of $\gamma = 1$ or $T = \infty$ but not both at the same time.

Markov Property

- Suppose that at a certain time step t, the environment is in state S_t and responds to action A_t by the agent.
- The response is modeled by giving to the agent information about the probability of the next state.
- This can be written as

$$Pr\{S_{t+1} = s', R_{t+1} = r \mid S_0, A_0, R_1, \ldots, S_{t-1}, A_{t-1}, R_t, S_t, A_t\}$$

- This is interpreted as the probability of the environment moving to state s' and offering reward r given the past states,actions and rewards.

Markov Property

- If the system has the *Markov property* then the environment response at $t+1$ depends only on the states S_t and the action A_t.
- This response is given as

$$p(s', r \mid s, a) = Pr\{S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a\}$$

- Note that even though the response depends only on the current state and action but the current state S_t *contains* some information on the sequence of states and actions prior to reaching that state.

Markov Decision Process

- A reinforcement learning tasks that satisfies the Markov property is called a *Markov Decision Process* (MDP)
- We restrict our study to *finite* MDP where the number of states and number of actions are finite.
- An MDP is defined by the one-step dynamics of the environment

$$p(s', r \mid s, a) = Pr\{S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a\}$$

- Given the above dynamics one can compute many quantities such as the expected(average) reward for a given state-action pair

$$r(s, a) = \sum_r \sum_{s'} rp(s', r \mid s, a)$$

- One can compute the state transition probability

$$p(s' \mid s, a) = \sum_{r} p(s', r \mid s, a)$$

Example: recycling robot

- The recycling robot introduced earlier can be models using MDP.
- The actions that the robot can perform are
  1. Actively search for cans (search)
  2. Wait until someone puts a can in (wait)
  3. Go to base station to recharge (recharge)
  4. So $A = \{search, wait, recharge\}$
- The possible states of the robot reflect the status of the battery: high and low so $S = \{high, low\}$
- Also we assume that when the battery is high the robot does not recharge.

Example cont.

- The transitions are modeled as
- When the robot state is high and the action is search the next state can be high with probability α and low with probability $1 - \alpha$.
- When the state is low and search is performed it could stay low with probability β or becomes depleted with probability $1 - \beta$. In this case the robot needs to be rescued and recharged.
- Finally when the robot is waiting we assume the battery is not used.
- We also assign rewards to every action as shown in the table below with reward for search higher than for wait to give an incentive to the robot otherwise it will not move

Example cont.

s	s'	a	$p(s' \mid s, a)$	$r(s, a, s')$
high	high	search	α	r_s
high	low	search	$1 - \alpha$	r_s
high	high	wait	1	r_w
high	low	wait	0	r_w
low	low	search	β	r_s
low	high	search	$1 - \beta$	-5
low	low	wait	1	r_w
low	high	wait	0	r_w
low	high	recharge	1	0
low	high	recharge	0	0

Table: MDP model for recycling robot

Value Functions

- Almost all reinforcement algorithms involve the concept of *value functions* that estimate "how good" for the agent to be in a given state
- Another value function estimates "how good" it is for the agent to perform an action in a given state
- Value function is defined as
- The concept of "how good" refers to the expected return:
  1. What is the expected return when the state is s?
  2. What is the expected return when the action is a in state s?
- Since what is computed involves actions and states in the future then those values depend on the *policy* of the agent

State value function

- First recall that a policy $\pi(a \mid s)$ maps a state s and action a to the probability of taking action a when in state s

$v_\pi(s)$=the value of state s under π

Is the expected return when starting in state s and following policy π

- Formally

$$v_\pi(s) = E_\pi \left[G_t \mid S_t = s \right] = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \right]$$

- Where E is the expected value (average)

State-action value function

- Similarly we define the value of taking action a in state s under policy π, $q_\pi(s, a)$

$q_\pi(s, a)$=the value of taking action a in state s under π

Is the expected return when starting in state s and taking the action a and thereafter following policy π

- Formally,

$$q_\pi = E_\pi \left[G_t \mid S_t = s, A_t = a \right] = E_\pi \left[\sum_{k=1}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right]$$

Bellman equation for $v_\pi(s)$

- We can obtain a recursive relation, called the Bellman equation, between $v_\pi(s)$ and its successors as follows

$$
\begin{aligned}
v_\pi(s) &= E_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1} \mid S_t = s\right] \\
&= E_\pi\left[R_{t+1} \mid S_t = s\right] + E_\pi\left[\gamma\sum_{k=0}^{\infty}\gamma^k R_{t+k+2} \mid S_t = s\right] \\
&= \sum_a \pi(a \mid s)\sum_{s',r} r\ p(s', r \mid s, a) \\
&+ \gamma\sum_a \pi(a \mid s)\sum_{s',r} p(s', r \mid s, a)E_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+2} \mid S_{t+1} = s'\right] \\
v_\pi(s) &= \sum_a\sum_{s',r} \pi(a \mid s)p(s', r \mid s, a)\left[r + \gamma v_\pi(s')\right]
\end{aligned}
$$

Detailed derivation

$$v_\pi = \sum_{a_t} \pi(a_t \mid S_t = s) \sum_{S_{t+1}, R_{t+1}} p(S_{t+1}, R_{t+1} \mid a_t, S_t = s)$$
$$\times \sum_{a_{t+1}} \pi(a_{t+1} \mid S_{t+1}) \sum_{S_{t+2}, R_{t+2}} p(S_{t+2}, R_{t+2} \mid a_{t+1}, S_{t+1})$$
$$\ldots \times \sum_{k}^{\infty} \gamma^k R_{t+k+1}$$

- We divide the expression $\sum_k^\infty \gamma^k R_{t+k+1}$ into two terms

$$\sum_{k}^{\infty} \gamma^k R_{t+k+1} = R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2}$$

- Expanding, the first term yields

$$\sum_{a_t} \pi(a_t \mid S_t = s) \sum_{S_{t+1}, R_{t+1}} p(S_{t+1}, R_{t+1} \mid a_t, S_t = s) R_{t+1}$$
$$\times \sum_{a_{t+1}} \pi(a_{t+1} \mid S_{t+1}) \sum_{S_{t+2}, R_{t+2}} p(S_{t+2}, R_{t+2} \mid a_{t+1}, S_{t+1})$$
$$\times \ldots$$

- Since the probabilities are normalized, the terms after the first line above sum to one so we get

$$\sum_{a_t} \pi(a_t \mid S_t = s) \sum_{S_{t+1}, R_{t+1}} p(S_{t+1}, R_{t+1} \mid a_t, S_t = s) R_{t+1}$$

- Expanding the second term yields

$$\sum_{a_t} \pi(a_t \mid S_t = s) \sum_{S_{t+1}, R_{t+1}} p(S_{t+1}, R_{t+1} \mid a_t, S_t = s)$$
$$\times \sum_{a_{t+1}} \pi(a_{t+1} \mid S_{t+1}) \sum_{S_{t+2}, R_{t+2}} p(S_{t+2}, R_{t+2} \mid a_{t+1}, S_{t+1})$$
$$\ldots \times \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2}$$

- The last two lines above are just the definition of $\gamma v_\pi(S_{t+1})$. Thus the second term is

$$\sum_{a_t} \pi(a_t \mid S_t = s) \sum_{S_{t+1}, R_{t+1}} p(S_{t+1}, R_{t+1} \mid a_t, S_t = s) \gamma v_\pi(S_{t+1})$$

Example

- As an illustration consider an agent that can move in a grid: up,down,left,right in a grid.
- The cells of the grid are the states of the MDP
- So the actions of the agent are L,U,R and D.
- The response of the environment is deterministic: if the agent takes an L action then the next state is to the left of the current one (unless it is on the boundary then it remains in the same state)
- Therefore the probability $p(s', r \mid s, a) = 1$ if s' is the "logical" next state for action a and zero otherwise

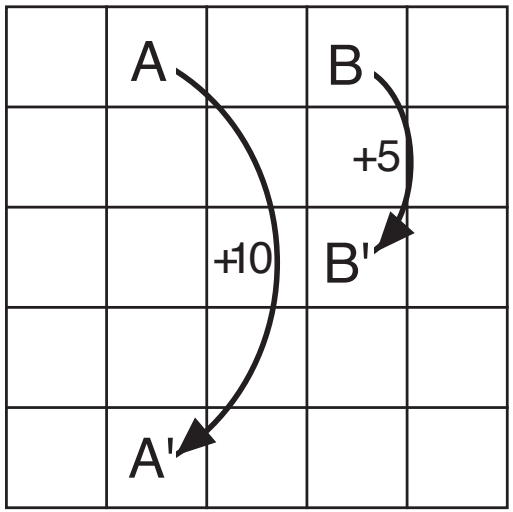
- A move that takes the agent off the grid leaves the state as it is but gets a reward of -1
- All other moves get a reward of 0 except moves from A and B they get a reward of $+10$ and $+5$ respectively.
- Furthermore, all moves out of A end in A' and all moves out of B end in B'
- At each step the agent selects the move with equal probability
- The MDP is shown in the grid below
- As an example consider the cell between A and B and denote it by s and the cell below s denote it by s'.
  - For action U, $p(s, -1 \mid s, U) = 1$ because moving up takes the agent off the grid which gets a reward of -1 and the agent remains in s. All others are zero.
  - For action L, $p(A, 0 \mid s, L) = 1$ all others are zero.
  - For action R, $p(B, 0 \mid s, R) = 1$ all others are zero.
  - For action D, $p(s', 0 \mid s, D) = 1$ all others are zero.

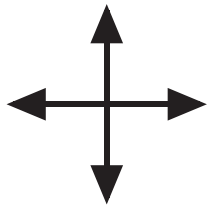
- According to the above we get

$$
\begin{aligned}
v(s) &= \pi(U \mid s) \times p(s, -1 \mid s, U)\,[-1 + v(s)] \\
&\quad + \pi(L \mid s) \times p(A, 0 \mid s, L)\,[0 + v(A)] \\
&\quad + \pi(R \mid s) \times p(B, 0 \mid s, R)\,[0 + v(B)] \\
&\quad + \pi(D \mid s) \times p(s', 0 \mid s, D)\,\left[0 + v(s')\right] \\
v(s) &= 0.25 \times [-1 + v(s)] \\
&\quad + 0.25 \times [v(A)] \\
&\quad + 0.25 \times [v(B)] \\
&\quad + 0.25 \times \left[v(s')\right]
\end{aligned}
$$

Example

- As an illustration consider an agent that can move: up,down,left,right in the grid below





Actions

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

Figure: Gridworld (Sutton)

Optimal value functions

- A policy π is said to be better than policy π' if $v_\pi(s) \geq v_{\pi'}(s)$ for all s.
- The optimal policy that is better than all other policies is denoted by π_*
- The optimal value functions are

$$v_*(s) = \max_\pi v_\pi(s)$$
$$q_*(s,a) = \max_\pi q_\pi(s,a)$$

- But

$$q_*(s,a) = \max_\pi E_\pi \left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \mid S_t = s, A_t = a \right]$$

- Let π^* be the optimal policy so

$$v_{\pi^*} = \sum_a \pi^*(a \mid s) q_{\pi^*}(s, a)$$

- Let π^+ be the policy where the best immediate action is chosen and thereafter the optimal policy is followed:

$$\pi^+(b \mid s) = \begin{cases} 1 \text{ if } b = \text{argmax}_a \, q_{\pi^*}(s, a) \\ 0 \text{ otherwise} \end{cases}$$

- Then

$$v_{\pi^+}(s) = q_{\pi^*}(s, b)$$

- Where, by definition, $q_{\pi^*}(s, b) \geq q_{\pi^*}(s, a)$ for all a.
- Since we are assuming that the number of actions is finite, say k, then we can expand $v_{\pi^*}(s) = \sum_a \pi^*(a \mid s) q_{\pi^*}(s, a)$ as

$$
\begin{aligned}
v_{\pi^*}(s) &= \pi^*(a^1 \mid s) q_{\pi^*}(s, a^1) + \ldots + \pi^*(a^k \mid s) q_{\pi^*}(s, a^k) \\
&\leq \pi^*(a^1 \mid s) q_{\pi^*}(s, b) + \ldots + \pi^*(a^k \mid s) q_{\pi^*}(s, b) \\
v_{\pi^*}(s) &\leq \left(\underbrace{\pi^*(a^1 \mid s) + \ldots + \pi^*(a^k \mid s)}_{=1} \right) q_{\pi^*}(s, b) \\
&\leq q_{\pi^*}(s, b) = v_{\pi^+}(s)
\end{aligned}
$$

- But $v_{\pi^*}(s)$ is assumed to be the optimal policy therefore

$$
v_{\pi^*}(s) = \max_a q_{\pi^*}(s, a)
$$

- Since no matter what π is $q(s, a)$ is the expected return **given** that $S_t = s$ and $A_t = a$ then the first term is fixed for a given s and a. Then

$$q_*(s, a) = \max_{\pi} E_{\pi} \left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \mid S_t = s, A_t = a \right]$$
$$= \sum_{s', r} p(s', r \mid s, a) \left[r + \gamma v_*(s') \right]$$

- Using the value of $q_*(s, a)$ we get

$$v_*(s) = \max_a \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_*(s') \right] \tag{1}$$

- Using the above value of v_* in the equation for $q_*(s, a)$ we obtain

$$q_*(s, a) = \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma \max_{a'} q_*(s', a') \right] \tag{2}$$

Evaluation of the optimal value functions

- If the number of states and actions are finite and if the dynamics of the system are know i.e. $p(s', r \mid s, a)$ are known
- Then the equations for v_* above two equations are just n equations in n unknown (n is the number of states)
- In principle they can be solved but in practice the number of states could be very big.
- Next we develop methods to obtain v_*.

Iterative evaluation of the optimal value functions

- Our basic strategy is to obtain better and better policies starting from an arbitrary policy
- To be able to compare policies we need to obtain the value function for a given policy
- This step is called *policy evaluation*: given a policy π (not necessarily optimal) we would like to compute $v_\pi(s)$ for every s.
- In principle, given a policy and the known dynamics of the system we can use the Bellman equation to compute the value function using

$$v_\pi(s) = \sum_a \sum_{s',r} \pi(a \mid s) p(s', r \mid s, a) \left[r + \gamma v_\pi(s') \right]$$

- If we have n states the above gives us n equations in n unknowns $(v_\pi(s))$

Iterative evaluation

- Instead we devise an iterative solution to compute v_π for a given policy.
- Consider a set of approximate values for v: $v_1, v_2, \ldots$ where each value is obtained from the previous one using the Bellman equation as an update rule

$$v_{k+1}(s) = \sum_a \sum_{s',r} \pi(a \mid s) p(s', r \mid s, a) \left[r + \gamma v_k(s') \right]$$

- Note that v_k keeps changing until $v_k = v_\pi$ in this case we reach a fix point.

Implementation

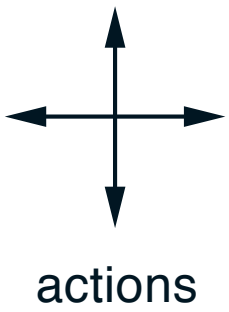
- To implement the above algorithm we maintain two arrays one for $v_k(s)$ for every s and one for $v_{k+1}(s)$.
- Having $v_k(s)$ we can compute $v_{k+1}(s)$ using the above equation.

EVAL(π)

foreach $s \in S$ **do**
  $V(s) \leftarrow 0$
end
while $\Delta > \theta$ **do**
  $\Delta \leftarrow 0$
  foreach $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) [r + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, \mid v - V(s))$
  end
end
return V

Example (Sutton)

- Consider an agent that needs to move in a grid as shown below
- The agent has 4 difference actions $\{up, down, left, right\}$.
- Each action causes the state to change accordingly except at the edges.
- The reward is -1 for each move until the terminal state is reached.
- The agent uses an equiprobable policy.



	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

Figure: Grid example

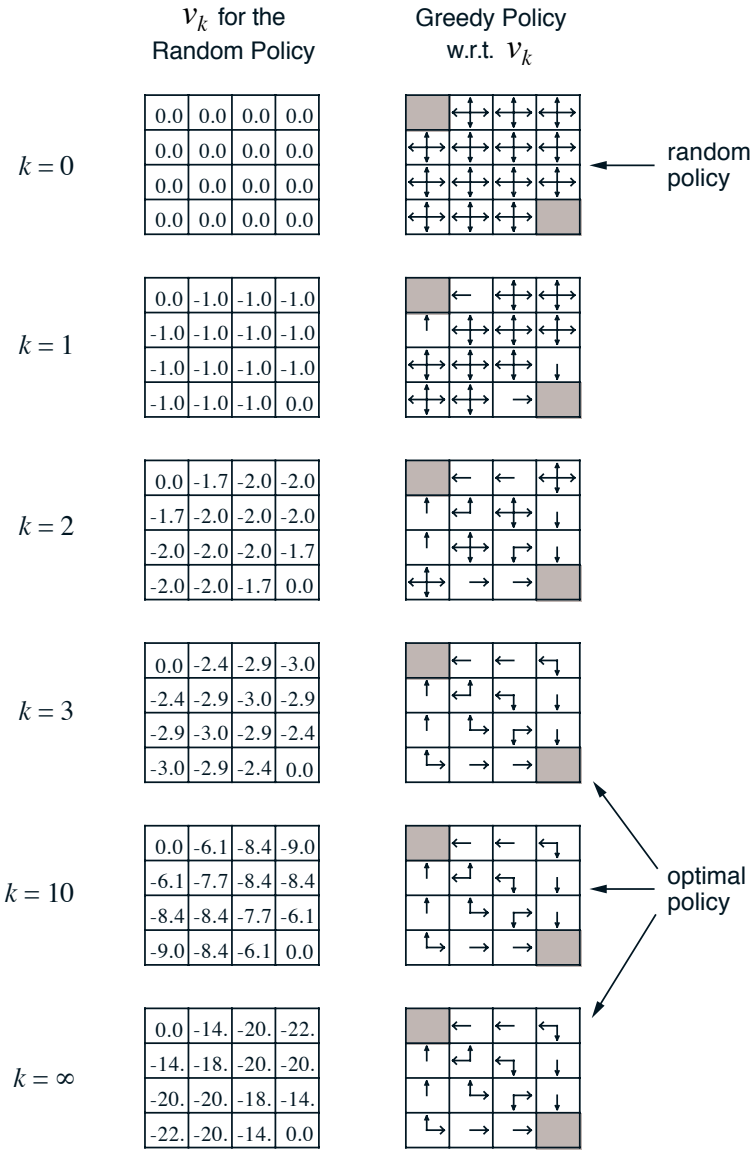


Figure: Resulting policy

Deterministic Policy Improvement

- We will describe an algorithm to find the optimal policy for a given MDP
- We will restrict ourselves to deterministic policies.
- In the case of deterministic policy and for a given state the choice of the action is deterministic (i.e. one action is selected)
- If the set of actions is $A_t(s)$ then

$$\pi(a \mid s) = 1$$
$$\pi(b \mid s) = 0 \text{ for all } b \neq a$$

- In this case we just write $\pi(s) = a$.

- Given a deterministic policy π consider the policy π' everywhere the same as π except at s.

$$\begin{aligned}\pi'(s) &= \operatorname*{argmax}_a q_\pi(s,a) \\ &= \operatorname*{argmax}_a \sum_{s',r} p(s',r \mid s,a)\left[r + \gamma v_\pi(s')\right]\end{aligned}$$

- Then

$$v_{\pi'}(s) \geq v_\pi(s) \text{ for all } s$$

Policy Iteration

- Now we have iterative procedures to evaluate and improve policies. We can combine them to obtain an approximate optimal policy as shown below

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \dots$$

Value Iteration

- One drawback to policy iteration is that for each of the iteration we need to do iterations for policy evaluation
- As we saw in the grid example this might not be necessary.
- Value iteration is similar to policy iteration where the policy evaluation is iterated only once
- In this case we combine the evaluation and the improvement steps.

$$v_{k+1}(s) = \max_a \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_k(s') \right]$$

Monte Carlo

- The previous methods for evaluating policies and obtaining optimal policies assume **complete knowledge of the environment**
- In other words we need $p(s', r \mid s, a)$ for each s', r, s, a.
- Sometimes these values are unknown or simply difficult to compute
- In this section we develop methods that *learn* or estimate these values from experience.
- Monte Carlo methods require *experience* (sample episodes)
- These methods do not require knowledge of the dynamics of the environment

Monte Carlo

- The basic idea is to average sample returns to obtain the state value and state-action values
- These methods are defined for episodic task: each run terminates in a finite number of steps
- Once an episode is completed we estimate the values.
- We look at two algorithms
  1. Policy evaluation for a given policy
  2. Obtaining the optimal policy
- Both of these algorithms will be implemented for a Black Jack game

Evaluation

- We will develop an algorithm to compute the state value function for a given policy
- Recall that $v(s)$ is the expected return starting from state s
- A simple way to compute the expected value is to run many episodes and average the values over them
- Given a policy π and state s we would like to compute $v_\pi(s)$.
- Each occurrence of s when following policy π is called a *visit*
- State s can be visited multiple times during the same episode.
- In this lecture we restrict ourselves to *first visit* averages

Algorithm for policy evaluation

EVAL(π)

foreach $s \in S$ **do**
  $V(s) \leftarrow 0$
end
for $i = 1$ **to** *nepisodes* **do**
  generate episode using π
  foreach s *in episode* **do**
    $G \leftarrow$ return of the first occurrence if s in episode
    $v[s] \leftarrow v[s] + G$
    $n[s] \leftarrow n[s] + 1$
    $v[s] \leftarrow v[s]/n[s]$
  end
end
return V

Evaluation of state-action value

- Evaluating the state-action value $q_\pi(s, a)$ is done similarly to the evaluation of $v_\pi(s)$.

$$\pi_0 \xrightarrow{E} q_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} q_{\pi_1} \xrightarrow{I} \dots$$

- As done in the value iteration algorithm the evaluation is mixed with the improvement step.
- Unlike the case of the state value function if we choose a deterministic policy it cannot be improved
- This is because not all actions for a given state are generated, only the action according to the policy
- One way to overcome this obstacle is the start each episode with different state-action with a given probability
- We hope that this leads to the sampling of all the state-action pair
- This strategy is called *Exploring Starts*
- Instead we use an $\epsilon-$greedy strategy.

$\epsilon-$greedy policy

- The main problem in computing that optimal $q(s, a)$ is computing the maximum over all possible actions
- To do that we need to have the values of all possible actions in order to select the maximum value
- The $\epsilon-$greedy policy choose the maximum value *most* of the time
- But with probability ϵ it might choose a random policy
- If $\mid A(s) \mid$ is the number of actions in state s
- Then non-greedy actions are selected with probability $\frac{\epsilon}{|A(s)|}$
- The greedy action is selected with probability $1 - \epsilon + \frac{\epsilon}{|As|}$.

ϵ-greedy policy

for $i = 1$ **to** *nepisodes* **do**
  generate episode using π
  foreach s, a *in episode* **do**
    $G \leftarrow$ return of the first occurrence if s, a in episode
    $n[s, a] \leftarrow n[s, a] + 1$
    $q[s, a] \leftarrow (q[s, a] + G)/n[s, a]$
  end
  foreach s *in episode* **do**
    $A^* \leftarrow \text{argmax}_a\, q[s, a]$
    foreach $a \in A(s)$ **do**
      if $a = A^*$ **then**
        $\pi(a \mid s) = 1 - \epsilon + \frac{\epsilon}{|A(s)|}$
      end
      else
        $\pi(a \mid s) = \frac{\epsilon}{|A(s)|}$
      end
    end
  end
end