IEOR 8100: Reinforcement learning

Lecture 1: Introduction

By Shipra Agrawal

1 Introduction to reinforcement learning

What is reinforcement learning?

Reinforcement learning is characterized by an agent continuously interacting and learning from a stochastic environment. Imagine a robot moving around in the world, and wants to go from point A to B. To do so, it tries different ways of moving its legs, learns from its successesful motion as well as from its falls and finally finds the most effective way to walk. Reinforcement learning is a branch of artificial intelligence that formalizes this trial-and-error method of learning.

It is essentially the science of making sequential decisions. How should the robot move its limbs so that it can eventually learn to walk and reach point B quickly? More generally, "how" should the agent interact with the environment, what actions should it take now, so that it is able to learn more about the environment and is more successful in future.

Reinforcement learning sits at the intersection of many disciplines of science, namely:

- Optimal control (Engineering)
- Dynamic Programming (Operations Research)
- Reward systems (Neuro-science)
- Classical/Operant Conditioning (Psychology)

In all these different fields there is a branch that is trying to study the same problem as reinforcement learning essentially the problem of how to make optimal sequential decisions. In engineering it is the problem of finding optimal control, in Operations research it is studied under Dynamic programming. The algorithmic principles behind reinforcement learning are motivated from the natural phenomena behind human decision making in simple words that "rewards provide a positive reinforcement for an action": this phenomena is studied in psychology as conditioning and in neuroscience as reward systems.

Key characteristics

Now, lets look at a few things that make reinforcement learning different from other paradigms of optimization, and other machine learning methods like supervised learning.

- **Lack of a "supervisor":** One of the main characteristic of RL is that there is no supervisor no labels telling us the best action to take, only reward signals to enforce some actions more than others. For example, for a robot trying to walk, there is no supervisor telling the robot if the actions it took were a good way to walk, but it does get some signals if form of the effect of its actions - moving forward or falling down which it can use to guide its behavior.
- **Delayed feedback:** Other major distinction is that the feedback is often delayed: the effect of an action may not be entirely visible instantaneously, but it may severely effect the reward signal many steps later. In the robot example, an aggressive leg movement may look good right now as it may seem to make the robot go quickly towards the target, but a sequence of limb movements later you may realize that aggressive movement made the robot fall. This makes it difficult to attribute credit and reinforce a good move whose effect may be seen only many steps and many moves later. This is also referred to as the "credit assignment problem".

- **Sequential decisions:** Time really matters in RL, the "sequence" in which you make your decisions (moves) will decide the path you take and hence the final outcome.
- **Actions effect observations:** And, finally you can see from these examples that the observations or feedback that an agent makes during the course of learning are not really independent, in fact they are a function of the agents' own actions, which the agent may decide based on its past observations. This is very unlike other paradigms like supervised learning, where the training examples are often assumed to be independent of each other, or at least independent of learning agents actions.

These are some key characteristics of reinforcement learning which differentiate it from the other branches of learning, and also make it a powerful model of learning for a variety of application domains.

Examples

Lets concretize this discussion with some examples, we already discussed the example of a robot trying to walk. Here are a few others:

- **Automated vehicle control:** Imagine trying to fly an unmanned helicopter, learning to perform stunts with it. Again, no one will tell you if a particular way of moving the helicopter was good or not, you may get signals in form of how the helicopter is looking in the air, that it is not crashing, and in the end you might get a reward for example a high reward for a good stunt, negative reward for crashing. Using these signals and reward, a reinforcement learning agent would learn a sequence of maneuvers just by trial and error.
- **Learning to play games:** Some of the most famous successes of reinforcement learning have been in playing games. You might have heard about Gerald Tesauro's reinforcement learning agent defeating world Backgammon Champion, or Deepmind's Alpha Go defeating the world's best Go player Lee Sedol, using reinforcement learning. A team at Google Deepmind built an RL system that can learn to play suite of Atari games from scratch by just by playing the game again and again, trying out different strategies and learning from their own mistakes and successes. This RL agent uses just the pixels of game screen as state and score increase as reward, and is not even aware of the rules of the game to begin with!
- **Medical treatment planning:** A slightly different but important application is in medical treatment planning, here the problem is to learn a sequence of treatments for a patient based on the reactions to the past treatments, and current state of the patient. Again, while you observe reward signals in the form of the immediate effect of a treatment on patients condition, the final reward is whether the patient could be cured or note, and that can be only observed in the end. The trials are very expensive in this case, and need to be carefully performed to achieve most efficient learning possible.
- **Chatbots:** Another popular application is chatbots: you might have heard of Microsoft's chatbots Tay and Zo, or intelligent personal assistants like Siri, Google Now, Cortana, and Alexa. All these agents try to make a conversation with a human user. What are conversations? They are essentially a sequence of sentences exchanged between two people. Again, a bot trying to make a conversation may receive encouraging signals periodically if it is making a good conversation or negative signals some times in the form of human user leaving the conversation or getting annoyed. A reinforcement learning agent can use these feedback signals to learn how to make good conversations just by trial and error, and after many many conversations, you may have a chatbot which has learned the right thing to say at the right moment!

2 Introduction to MDP: the optimization/decision model behind RL

Markov decision processes or MDPs are the stochastic decision making model underlying the reinforcement learning problem. Reinforcement learning is essentially the problem when this underlying model is either unknown or too difficult (large) to solve in order to find an optimal strategy in advance. Therefore, instead the reinforcement learning agent learns and optimizes the model through execution and simulation, continuously using feedback from the past decisions to learn the underlying model and reinforce good strategies. But, more on that later, first lets understand what is a Markov decision process.

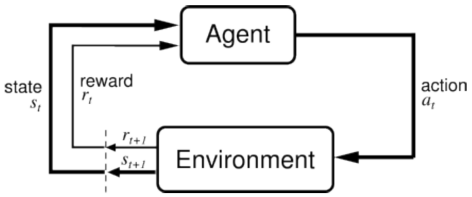


Figure 1: Figure taken from Sutton and Barto 1998

In this sequential decision making model, the agent needs to make decisions in discrete rounds $t = 1, 2, \ldots,$. In every round, all the relevant information from the past, is captured in the state of the process. The definition of 'state' depends on the problem, and is part of the modeling process. Let's consider again the example of a robot who is trying to move from point A to B. The current state of the robot in this example could be a combination of the location of the robot, the stance of the robot: whether it is standing, sitting, or moving, and its current velocity if it is moving. The decision maker, which in our example the robotic controller, observes the current state and takes an action, for example whether to lift a leg, or to move a limb of the robot forward. The agent then observes the reward signal and the transition to the next state, which depend both on the action taken and the state in which it was taken. For example, aggressively moving a leg may be a good action when the robot is walking or running, it will produce a positive reward signal and next state will also be a desirable state the robot would have moved closer to the target. However, when the robot is still say in a standing position, the same aggressive action may make the robot fall down.

The defining property of MDPs is the Markov property which says that the future is independent of the past given the current state. This essentially means that the state in this model captures all the information from the past that is relevant in determining the future states and rewards.

Formal definition. A Markov Decision Process (MDP) is specified by a tuple (S, s_1, A, P, R, H), where S is the set of states, s_1 is the starting state, A is the set of actions. The process proceeds in discrete rounds $t = 1, 2, \ldots, H$, starting in the initial state s_1. In every round, t the agent observes the current state $s_t \in S$, takes an action $a_t \in A$, and observes a feedback in form of a reward signal $r_{t+1} \in \mathbb{R}$. The agent then observes transition to the next state $s_{t+1} \in S$.

The Markov property mentioned earlier is formally stated as the following two properties: firstly, the probability of transitioning to a particular state depends only on current state and action, and not on any other aspect of the history. The matrix $P \in [0, 1]^{S \times A \times S}$ specificies these probabilities. That is,

$$\Pr(s_{t+1} = s' | \text{ history till time } t) = \Pr(s_{t+1} = s' | s_t = s, a_t = a) = P(s, a, s')$$

And secondly, the reward distribution depends only on the current state and action. So, that the expected reward at time t is a function of current state and action. A matrix R specifies these rewards.

$$\mathbb{E}[r_{t+1} | \text{ history till time } t] = \mathbb{E}[r_{t+1} | s_t = s, a_t = a] = R(s, a)$$

In some problems, a different reward r_{t+1} may be specified for every triple s_t, a_t, s_{t+1}. This is equivalent to the above model. Let $R(s, a, s')$ be the expected (or deterministic) reward when action a is taken in state s and transition to state s' is observed. Then, we can obtain the same model as above by defining

$$R(s, a) = \mathbb{E}[r_{t+1} | s_t = s, a_t = a] = \mathbb{E}_{s' \sim P(s,a)}[R(s, a, s')]$$

Policy A policy specifies what action to take at any time step. A history dependent policy at time t is a mapping from history till time t to an action. A Markovian policy is a mapping from state space to action action $\pi : S \to A$. Due to Markovian property of the MDP, it suffices to consider Markovian policies (in the sense that for any history

dependent policy same performance can be achieved by a Markovian policy). Therefore, in this text, policy refers to a Markovian policy.

A deterministic policy $\pi : S \to A$ is mapping from any given state to an action. A randomized policy $\pi : S \to \Delta^A$ is a mapping from any given state to a distribution over actions. Following a policy π_t at time t means that if the current state $s_t = s$, the agent takes action $a_t = \pi_t(s)$ (or $a_t \sim \pi(s)$ for randomized policy). Following a stationary policy π means that $\pi_t = \pi$ for all rounds $t = 1, 2, ...$

Any stationary policy π defines a Markov chain, or rather a 'Markov reward process' (MRP), that is, a Markov chain with reward associated with every transition. The transition probability vector and reward for this MRP in state s is given by $\Pr(s'|s) = P^\pi_s, \mathbb{E}[r_t|s] = r^\pi_s$, where P^π is an $S \times S$ matrix, and r^π is an S-dimensional vector defined as:

$$P^\pi_{s,s'} = \mathbb{E}_{a\sim\pi(s)}[P(s, a, s')], \forall s, s' \in S$$
$$r^\pi_s = \mathbb{E}_{a\in\pi(s)}[R(s, a)]$$

The stationary distribution (if exists) of this Markov chain when starting from state s_1 is also referred to as the stationary distribution of the policy π, denoted by d^π:

$$d^\pi(s) = \lim_{t\to\infty} \Pr(s_t = s|s_1, \pi)$$

Goals. The tradeoffs between immediate reward vs. future rewards of the sequential decisions and the need for planning ahead is captured by the goal of the Markov Decision Process. At a high level, the goal is to maximize some form of cumulative reward. Some popular forms are total reward, average reward, or discounted sum of rewards.

- **finite horizon MDP**: Here, actions are taken for $t = 1, \ldots, H$ where H is a finite horizon. The total (discounted) reward criterion is simply to maximize the expected total (discounted) rewards in an episode of length H. (In reinforcement learning context, when this goal is used, the MDP is often referred to as an episodic MDP.) For discount $0 \le \gamma \le 1$, the goal is to maximize

$$\mathbb{E}[\sum_{t=1}^{H} \gamma^{t-1} r_t|s_1]$$

- **Infinite horizon MDP:**
  - Expected total discounted reward criteria: The most popular form of cumulative reward is expected discounted sum of rewards. This is an asymptotic weighted sum of rewards, where with time the weights decrease by a factor of $\gamma < 1$. This essentially means that the immediate returns more valuable than those far in the future.

$$\lim_{T\to\infty} \mathbb{E}[\sum_{t=1}^{T} \gamma^{t-1} r_t|s_1]$$

  - Expected total reward criteria: Here, the goal is to maximize

$$\lim_{T\to\infty} \mathbb{E}[\sum_{t=1}^{T} r_t|s_1]$$

    The limit may not always exist or be bounded. We are only interested in cases where above exists and is finite. This requires restrictions on reward and/or transition models. Interesting cases include the case where there is an undesirable state, the reward after reaching that state is 0. For example, end of of a computer game. The goal would be to maximize the time to reach this state. (A minimization version of this model is where there is a cost associated with each state and the game is to minimize the time to reach winning state, called the shortest path problem).

– Expected average reward criteria: Maximize

$$\lim_{T\to\infty} \mathbb{E}[\frac{1}{T}\sum_{t=1}^{T} r_t|s_1]$$

Intuitively, the performance in a few initial rounds does not matter here, what we are looking for is a good asymptotic performance. This limit may not always exist. Assuming bounded rewards and finite state spaces, it exists under some further conditions on policy used.

Discounted sum of rewards is one of the most popular forms of goal in MDP for many reasons: it is mathematically convenient as it is always finite and avoids the complications due to infinite returns. Practically, depending on the application, immediate rewards may indeed be more valuable. Further, often uncertainty about far future are not well understood, so you may not want to give as much weight to what you think you might earn far ahead in future. The discounted reward criteria can also be seen as a soft version of finite horizon, as the contribution of reward many time steps later is very small. As you will see later, discounted reward MDP has many desirable properties for iterative algorithm design and learning. Due to these reasons, often the practical approaches which actually execute the MDP for finite horizon, use policies, algorithms and insights from infinite horizon discounted reward setting.

Gain of the MDP. Gain (roughly the 'expected value objective' or formal goal) of an MDP when starting in state s_1 is defined as (when supremum exists):

- episodic MDP:

$$\rho(s_1) = \sup_{\{\pi_t\}} \mathbb{E}[\sum_{t=1}^{H} \gamma^{t-1} r_t|s_1]$$

- Infinite horizon expected total reward.

$$\rho(s_1) = \sup_{\{\pi_t\}} \lim_{T\to\infty} \mathbb{E}[\sum_{t=1}^{T} r_t|s_1]$$

- Infinite horizon discounted sum of rewards.

$$\rho(s_1) = \sup_{\{\pi_t\}} \lim_{T\to\infty} \mathbb{E}[\sum_{t=1}^{T} \gamma^{t-1} r_t|s_1]$$

- infinite horizon average reward:

$$\rho(s_1) = \sup_{\{\pi_t\}} \lim_{T\to\infty} \mathbb{E}[\frac{1}{T}\sum_{t=1}^{T} r_t|s_1]$$

Here, expectation is taken with respect to state transition and reward distribution, supremum is taken over all possible sequence of policies for the given MDP. It is also useful to define gain ρ^π of a stationary policy π, which is the expected (total/total discounted/average) reward when policy π is used in all time steps. For example, for infinite average reward:

$$\rho^\pi(s_1) = \lim_{T\to\infty} \mathbb{E}[\frac{1}{T}\sum_{t=1}^{T} r_t|s_1]$$

where $a_t = \pi(s_t), t = 1, \ldots, T$.

Optimal policy. Optimal policy is defined as the one that maximizes the gain of the MDP. Due to the structure of MDP it is not difficult to show that it is sufficient to consider Markovian policies. Henceforth, we consider only Markovian policies.

For infinite horizon MDP with average/discounted reward criteria, a further observation that comes in handy is that such a MDP always has a stationary optimal policy, whenever optimal policy exists. That is, there always exists a fixed policy so that taking actions specified by that policy at all time steps maximizes average/discounted/total reward. The agent does not need to change policies with time. This insight reduces the question of finding the best sequential decision making strategy to the question of finding the best stationary policy.

The results below assume finite state, action space and bounded rewards.

Theorem 1 (Puterman [1994], Theorem 6.2.7)**.** *For any infinite horizon discounted MDP, there always exists a deterministic stationary policy π that is optimal.*

Theorem 2 (Puterman [1994], Theorem 7.1.9)**.** *For any infinite horizon expected total reward MDP, there always exists a deterministic stationary policy π that is optimal.*

Theorem 3 (Puterman [1994], Theorem 8.1.2)**.** *For infinite horizon average reward MDP, there always exist a stationary (possibly randomized) policy which is an optimal policy .*

Therefore, for infinite horizon MDPs, optimal gain:

$$\rho^*(s) = \max_{\pi:\text{Markovian stationary}} \rho^\pi(s)$$

(limit exists for stationary policies [Puterman Proposition 8.1.1])

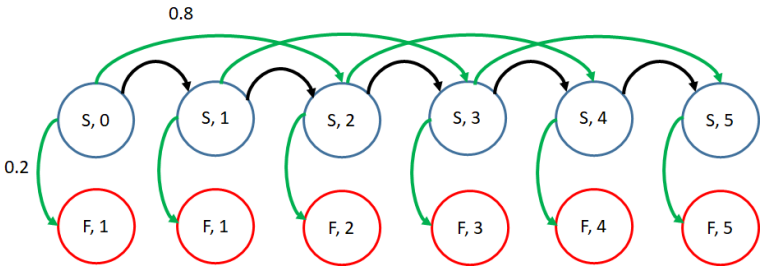
These results imply that the optimal solution space is simpler for infinite horizon case, and make infinite horizon an attractive model even when the actual problem is finite horizon but the horizon is long. Even when such a result on optimality of stationary policy is not available, 'finding the best stationary policy' is often used as an alternate convenient and more tractable objective, instead of finding the optimal policy which may not exist or may not be stationary in general.

Solving an MDP, finding optimal policy. Solving or optimizing an MDP means finding a strategy for the agent to choose actions in such a way so as to maximize the stated form of cumulative reward. Note that an action not only determines the current reward, but also future states and therefore future rewards. So, the agent needs to choose these actions or policy strategically in order to optimize the overall cumulative reward. The rest of the lecture will develop formal constructs necessary to design algorithmic solutions for solving this optimization problem. But, let's first look at some examples.

3 Examples

Example 1. Lets formulate a simple MDP for a robot moving on a line. Let's say there are only three actions available to this robot: walk or run or stay. Walking involves a slow limb movement, which allows the robot to move one step without falling. Running involves an aggressive limb movement which may allow the robot to move two steps forward, but there is a 20% chance to fall. Once the robot falls, it cannot get up. The goal is to move forward quickly and as much as possible without falling.

We can model this as MDP. We define state of the robot as a combination of its Stance: whether it is standing upright or has fallen down, denoted here as S or F, and its location on the line, represented here as $0, 1, 2, 3, 4, 5, \ldots$. So, this state $(S, 1)$ for example means that the robot is upright at location 1, where as this state (F,2) means that the robot has fallen down at location 2. The robot starts in a standing state at the beginning of the line, that is at state (S,0). Action space consists of three actions: walk , run, and stay. The state transition depends on current state and action. Walking in a standing state always transfers the robot to a standing state at a location one step ahead(this transition on taking walk action is represented here by these black arrows). So, by walking the robot can always move up by one step. On the other hand, by taking the second action of running or an aggressive limb movement, a robot in a standing state may move by 2 steps at a time (shown here by these green arrows), but there is also a 20% chance of falling and transitioning to a Fallen state. In fallen state, there is no effect of any action, the robot is stuck. Stay action keeps the robot in the current state.



As is often the case in applications of MDPs or more generally, reinforcement learning, the rewards and goals are not exogeneously given but are also an important part of the application modeling process. Different settings will lead to different interpretations of the problem. Lets say the reward is the number of steps the agent moves as a result of an action in the current state. Now, If the goal is set to be the total reward (infinite horizon), then the agent should just walk, because the aim is to move as many steps as possible, so moving quickly is not important, and the robot should not take the risk of falling by running. The total reward is infinite. But if the goal is set as discounted reward, then it is also important to move more steps initially and gather more reward quickly before the discount term becomes very small, so it may be useful to run (depending on how small the discount factor γ is). One can also set reward to be 0 for all the states except the final destination, say (S,5), where the reward is 1. In that case, the discounted sum of rewards would be simply γ^τ if (S,5) is reached at time τ, so the agent will want to minimize τ, the time to reach the end without falling, and therefore may want to move aggressively at times.

Another important point to note from this example, is that in an MDPs an action has long term consequences. For instance, it may seem locally beneficial to run on state (S,0) here because, even with some chance of falling, the 2 steps gain at 80% chance means that expected immediate reward is .8*2 = 1.6, which still more thatn the expected immediate reward of 1 step that can be earned by walking, but that greedy approach ignores all the reward you can make in future if you don't fall. Finding an optimal sequential decision making strategy under this model therefore involves careful tradeoff of immediate and future rewards.

Here are some examples of possible policies. Suppose you decide that whenever the robot is in a standing position, you will make the robot walk and not run - this is a stationary Markovian (deterministic) policy. A more complex policy could be that whenever the robot is standing 2 or more steps away from the target location (5), in those states you will make it walk, otherwise you make it run. This is another Markovian stationary deterministic policy which is conservative in states farther from target and aggressive in states closer to the target. You can get a randomized policy by making it walk, run or stay with some probability.In general you can change policies over time, it doesn't need to be stationary. May be initially you decided that you will always walk in standing state, but later on after realizing that you are moving very slowly, you changed your mind and started running in all states. In this case the agent is using different policies at different time steps, i.e., a nonstationary policy.

Example 2. Inventory control problem. Each month the manager of a warehouse determines current inventory (stock on hand) of a single product. Based on this information, she decides whether or not to order additional stock from a supplier. In doing so, she is faced with a trade-off between holding costs and the lost sales or penalties associated with being unable to satisfy customer demand for the product. The objective is to maximize some measure of profit over decision-making horizon. Demand is a random viable with a probability distribution known to the manager. Let s_t denote the inventory on hand at the beginning of the tth time period, a_t the number of units ordered by the inventory manager period and D_t the random demand during this time period. We assume that the demand has a known time-homogeneous probability distribution $p_j = \Pr(D_t = j)$, $j = 0, 1, \ldots$. The inventory at decision epoch $t + 1$ referred to as s_{t+1}, is related to the inventory at decision epoch t, s_t, through the system equation

$$s_{t+1} = \max\{s_t + a_t - D_t, 0\} \equiv [s_t + a_t - D_t]^+.$$

That backlogging is not allowed implies the non-negativity of the inventory level. Denote by $O(u)$ the cost of ordering u units in any time period. Assuming a fixed cost K for placing orders and a variable cost $c(u)$ that

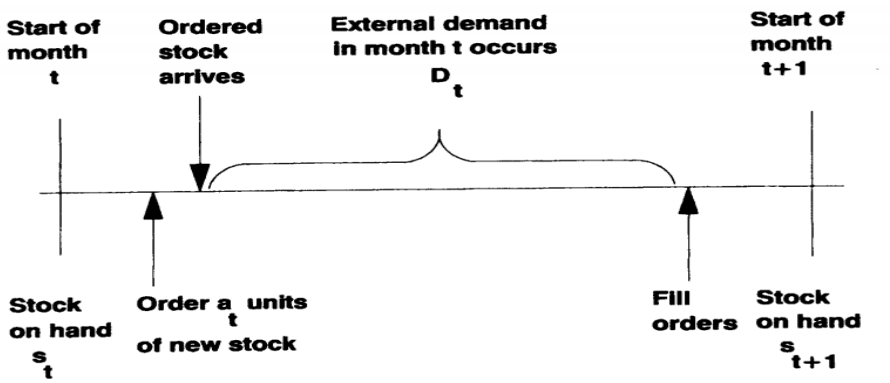


Figure 2: Timing of events in an inventory model

increases with quantity ordered, we have

$$O(u) = [K + c(u)]1_{\{u>0\}}.$$

The cost of maintaining an inventory of u units for a time period is represented by a nondecreasing function $h(u)$. Finally, if the demand is j units and sufficient inventory is available to meet demand, the manager receives revenue with present value $f(j)$. In this model, the reward depends on the state of the system at the subsequent decision epoch, that is

$$r_t(s_t, a_t, s_{t+1}) = -O(a_t) - h(s_t + a_t) + f(s_t + a_t - s_{t+1}).$$

The goal of a inventory policy could be to maximize expected total reward in a finite horizon, or discounted reward if the firm cares more about near future.

Running Example 3. Here is another simple example of MDP model for a robot trying to walk. We eliminate the location and target location for the robot. The robot now just want to make as much progress as possible without falling. The robot can be in three states : Fallen state, Standing state, or Moving state. There are two

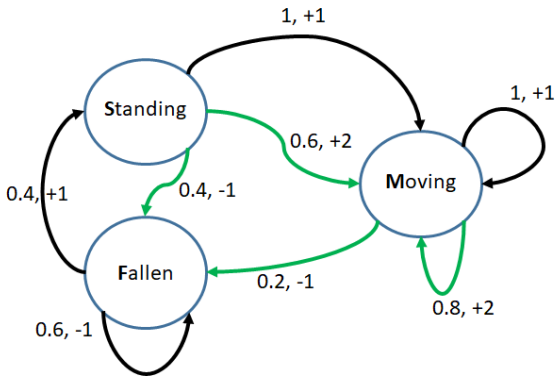


Figure 3: A simple MDP for the robot toy example

possible actions: moving the robot legs slowly and moving the robot legs aggressively. Black arrows show what happens with slow action, and green arrows show what happens with the aggressive action. By slow action in Fallen state, the robot may be able to stand up only with 0.4 prob to receive reward +1, but with 0.6 probability it may fall back and receive reward −1. The fast or aggressive action is not available in this state. In Standing state, slow action is very reliable and puts the robot in moving state with prob 1, also earning a reward 1. Fast action in a standing state can earn more reward when it is successful in transferring the robot to the moving state,

but with 0.4 probability, it may make the robot fall, which means transfer to the Fallen state and a reward of -1. In moving state, again, the slow action is reliable, but fast action can earn more reward, with a risk of falling that is smaller than the risk in standing state.

Here, state space $S = \{'F', 'S', 'M'\}$, $A = \{'slow', 'fast'\}$. R is an $S \times A$ matrix and P is $S \times A \times S$ matrix.

$$R = \begin{bmatrix} (0.6 \times -1 + 0.4 \times 1) & 0 \\ 1 & (0.6 \times 2 + 0.4 \times -1) \\ 1 & (0.8 \times 2 + 0.2 \times -1) \end{bmatrix} = \begin{bmatrix} -0.2 & 0 \\ 1 & 0.8 \\ 1 & 1.4 \end{bmatrix}$$

$$P(s, \text{slow}, s') = \begin{bmatrix} 0.6 & 0.4 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix} P(s, \text{fast}, s') = \begin{bmatrix} 1 & 0 & 0 \\ 0.4 & 0 & 0.6 \\ 0.2 & 0 & 0.8 \end{bmatrix}$$

4 Solving an MDP (Bellman equations)

4.1 Finite horizon

Bellman equations named after their discoverer Richard Bellman, provide a recursive formula for gain of an MDP. For finite horizon MDP this is simply dynamic programming.

Consider the toy example of robot trying to walk in Figure 3. Let the starting state is 'Standing'. Try to compute the optimal policy for horizon $H = 1, 2, 3, \ldots, 10$ for total reward criteria ($\gamma = 1$) *by enumeration*. For $H = 1$, the optimal policy simply maximizes immediate reward $\arg\max_a R(s, a)$ for $s =' Standing'$. And, therefore optimal policy is to take the slow action (black). For $H = 2$, the optimal policy involves deciding a two-stage decision. Deciding the first action (in 'Standing' state) involves enumerating the tree of all possible trajectories of state-action sequences starting from this state and every action. That is, $A^H(S)^{H-1}$ possibilities. A central idea in solving MDPs is that the Markovian structure can be used to make this computation tractable, using the simple idea of memoization (dynamic programming).

Bellman optimality equations. Let $V_k^*(s)$ be defined as the maximum total (discounted) reward achievable over a k length horizon starting in state s. Then,

$$V_k^*(s) = \max_\pi \mathbb{E}[\sum_{t=1}^{k} \gamma^{t-1} r_t | s_1 = s]$$

where maximum is taken over all (non-stationary) policies $\pi = (\pi_1, \ldots, \pi_k)$, $a_t = \pi_t(s_t), \mathbb{E}[r_t|s_t] = R(s_t, a_t), \Pr(s_{t+1} = s'|s_t, a_t) = P(s_t, a_t, s')$.

Then, we have optimal substructure property:

$$\begin{aligned} V_k^*(s) &= \max_\pi \left\{ \mathbb{E}[r_1|s_1 = s] + \mathbb{E}[\mathbb{E}[\sum_{t=2}^{k} \gamma^{t-1} r_t | s_1 = s, s_2 = s']] \right\} \\ &= \max_a R(s, a) + \max_{\pi_2, \ldots, \pi_k} \sum_{s'} P(s, a, s') \mathbb{E}[\sum_{t=2}^{k} \gamma^{t-1} r_t | s_2 = s'] \\ &= \max_a R(s, a) + \gamma \sum_{s'} P(s, a, s') \left\{ \max_{\pi_1, \ldots, \pi_{k-1}} \mathbb{E}[\sum_{t=1}^{k-1} \gamma^{t-1} r_t | s_1 = s'] \right\} \\ &= \max_a R(s, a) + \gamma \sum_{s'} P(s, a, s') V_{k-1}^*(s'), k = 1, \ldots, H \end{aligned}$$

And, by definition

$$\rho^*(s_1) = V_H^*(s_1)$$

This can be used to solve a finite horizon MDP by dynamic programming, by building a table of $H \times S$ values, starting from the last time step.

Example. Let's compute below for the toy example of robot MDP. Further examples are available in Section 4.6 of Puterman [1994].

Let's optimize for horizon $H = 4$. Now, $V_1^*(\cdot)$ is simply immediate reward maximization,

$$\begin{aligned} V_1^*(F) &= 0(\text{fast action/do nothing}) \\ V_1^*(S) &= 1(\text{slow action}) \\ V_1^*(M) &= 1.4(\text{fast action}) \end{aligned}$$

This suggest that if time horizon is 1, the robot should not try to get up from fallen state.

$$\begin{aligned} V_2^*(F) &= \max\{-0.2 + 0.4 \times 1, 0 + 0\} = 0.2(\text{slow action}) \\ V_2^*(S) &= \max\{1 + 1.4, 0.8 + 0.6 \times 1.4 + 0.4 \times 0 = 2.4(\text{slow action}) \\ V_2^*(M) &= \max\{1 + 1.4, 1.4 + 0.8 \times 1.4 + 0.2 \times 0 = 2.56(\text{fast action}) \end{aligned}$$

$$\begin{aligned} V_3^*(F) &= \max\{-0.2 + 0.4 \times 2.4, 0 + 0\} = 0.76(\text{slow action}) \\ V_3^*(S) &= \max\{1 + 2.56, 0.8 + 0.6 \times 2.56 + 0.4 \times 0 = 3.56(\text{slow action}) \\ V_3^*(M) &= \max\{1 + 2.56, 1.4 + 0.8 \times 2.56 + 0.2 \times 0 = \max\{3.56, 3.448\} = 3.56(\text{slow action}) \end{aligned}$$

(If you use $\gamma < 1$, it might take more time steps for the action in state M to become slow action, depending on how small γ is. Intuitively, if horizon is short or future is either discounted heavily you might want to be more aggressive).

In the next iteration, the policy is the same:

$$\begin{aligned} V_4^*(F) &= \max\{-0.2 + 0.4 \times 3.56, 0 + 0\}(\text{slow action}) \\ V_4^*(S) &= \max\{1 + 3.56, 0.8 + 0.6 \times 3.56 + 0.4 \times 0 = \max\{4.56, 2.936\}(\text{slow action}) \\ V_4^*(M) &= \max\{1 + 3.56, 1.4 + 0.8 \times 3.56 + 0.2 \times 0 = \max\{4.56, 4.248\} = 4.56(\text{slow action}) \end{aligned}$$

4.2 Infinite horizon discounted reward

Henceforth we will assume finite and discrete state space S, finite and discrete action space A, bounded rewards $R(s, a)$ and discount $\gamma < 1$. In this case, there exists an optimal stationary policy. We abuse notation, to denote a stationary policy $(\pi, \pi, \pi, \ldots)$, as π. Therefore, we are effectively looking for a stationary policy $\pi^* \in \arg\max \rho^\pi(s_1)$.

Value of a policy π in a given state s at time t is the gain when starting from state s.

$$V^\pi(s) = \lim_{T\to\infty} \mathbb{E}[\sum_{t=1}^{T} \gamma^{t-1} r_t | s_1 = s], \forall s.$$

Note that gain of a policy is simply $\rho^\pi(s_1) = V^\pi(s_1)$, i.e., the value from the starting state. (Value is also referred to as 'cost-to-go' when cost-based version of MDP is considered. In that version, instead of reward, you observe a cost, and the goal is to minimize total/average/discounted cost).

Bellman equations for value of a policy. In infinite horizon case, the value of policy only depends on the state and not the time, and satisfies the following recursive relation.

$$V^\pi(s) = \mathbb{E}_{a\sim\pi(s), s'\sim P(s,a)} \left[R(s, a, s') + V^\pi(s') \right], \text{ or,}$$

$$V^\pi = \mathbf{R}^\pi + P^\pi V^\pi$$

Proof:

$$\begin{aligned} V^\pi(s) &= \mathbb{E}[r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \ldots | s_1 = s] \\ &= E[r_1 | s_1 = s] + \gamma \mathbb{E}[\mathbb{E}[r_2 + \gamma r_3 + \gamma^2 r_4 + \ldots | s_2] | s_1 = s] \end{aligned}$$

The first term here is simply the expected reward in state s when action is given by $\pi(s)$. The second term is γ times the value function at $s_2 \sim P(s, \pi(s), \cdot)$

$$\begin{aligned} V^\pi(s) &= \mathbb{E}[R(s,\pi(s),s_1) + \gamma V^\pi(s_2)|s_1 = s] \\ &= R(s,\pi(s)) + \gamma \sum_{s_2 \in S} P(s,\pi(s),s_2) V^\pi(s_2) \\ &= R^\pi(s) + \gamma [P^\pi V^\pi](s) \end{aligned}$$

Bellman optimality equations. Let $V^*(s) = \max_\pi V^\pi(s)$.

$$V^*(s) = \max_a R(s,a) + \gamma \sum_{s'} P(s,a,s') V^*(s')$$

And, by definition

$$\rho^*(s) = V^*(s)$$

Proof: for all s, from the theorem ensuring stationary optimal policy:

$$\begin{aligned} V^*(s) = \max_\pi V^\pi(s) &= \max_\pi \mathbb{E}_{a\sim\pi(s), s'\sim P(s,a)} \left[R(s,a,s') + \gamma V^\pi(s')\right] \\ &\leq \max_a R(s,a) + \gamma \sum_{s'} P(s,a,s') \max_\pi V^\pi(s') \\ &= \max_a R(s,a) + \gamma \sum_{s'} P(s,a,s') V^*(s') \end{aligned}$$

Now, if the above inequality is strict then the value of state s can be improved by using a (possibly non-stationary) policy that uses action $\arg\max_a R(s,a)$ in the first step. This is a contradiction to the definition $V^*(s)$. Therefore,

$$V^*(s) = \max_a R(s,a) + \gamma \sum_{s'} P(s,a,s') V^*(s')$$

Technically, above only shows that V^* satisfies the Bellman equations. Theorem 6.2.2 (c) in Puterman [1994] shows that V^* is in fact unique solution of above equations. Therefore, satisfying these equations is sufficient to guarantee optimality, so that it is not difficult to see that the deterministic (stationary) policy

$$\pi^*(s) = \arg\max_a R(s,a) + \gamma \sum_{s'} P(s,a,s') V^*(s')$$

is optimal (see Puterman [1994] Theorem 6.2.7 for formal proof).

Linear programming. The fixed point for above Bellman optimality equations can be found by formulating a linear program. It amounts to :

$$\begin{aligned} \min_{\mathbf{v}\in\mathbb{R}^S} \quad & \textstyle\sum_s w_s \nu_s \\ \text{subject to} \quad & v_s \geq R(s,a) + \gamma P(s,a)^\top \mathbf{v} \quad \forall a, s \end{aligned}$$

for any positive weights $w_1, \ldots, w_S$.

Proof. V^* clearly satisfies the constraints of the above LP. Next, we show that $\mathbf{v} = V^*$ minimizes the objective.

$v_s \geq R(s,a) + P(s,a)^\top \mathbf{v}, \forall s, a$ implies that

$$v_s \geq R(s,\pi^*(s)) + \gamma P(s,\pi^*(s))^\top \mathbf{v}, \forall s$$

(Above is written assuming π^* is deterministic, which is infact true in the infinite horizon discounted reward case.) Or,

$$(I - \gamma P^{\pi^*})\mathbf{v} \geq R^{\pi^*}$$

Because $\gamma < 1$, $(I - \gamma P^\pi)^{-1}$ exists for all π, and for any $u \geq 0$

$$(I - \gamma P^\pi)^{-1} u = (I + \gamma P^\pi + \gamma^2 (P^\pi)^2 + \cdots) u \geq u$$

Therefore, from above

$$(I - \gamma P^{\pi^*})^{-1}((I - \gamma P^{\pi^*})\mathbf{v} - R^{\pi^*}) \geq 0$$

Or,

$$\mathbf{v} \geq (I - \gamma P^{\pi^*})^{-1} R^{\pi^*} = V^*$$

Therefore, $\mathbf{w}^\top \mathbf{v}$ for $\mathbf{w} > 0$ is minimized by $\mathbf{v} = V^*$.

4.3 Infinite horizon average reward

Communicating MDPs. For average reward MDP, we will make an additional 'communicating' assumption.

Definition 4. *An MDP is called* **communicating** *if for any two states s, s', there exists a policy such that the expected number of steps to reach s' from s is finite.*

A convenient fact is that optimal gain does not depend on the starting state under mild conditions on the communicating properties of the MDP:

Theorem 5 (Puterman [1994], Theorem 8.3.2 in Section 8.3.3)**.** *For communicating MDP, for optimal gain policy $\rho^*(s) = \rho^*$, i.e., optimal average infinite horizon reward does not depend on starting state.*

Proof. (Sketch) Suppose that that there exists $s_1 \neq s_2$ such that $\rho^*(s_1) > \rho^*(s_2)$. Since the MDP is communicating there exists a policy π_0 using which we can go from s_2 to s_1 in time τ with finite expected value, say $\mathbb{E}[\tau] \leq D$. Then we can construct a (possibly non-stationary) policy, which first goes from s_2 to s_1 using π_0 in at most D steps in expectation, and then uses the optimal policy (say π_1) for s_1. Such a policy will have infinite horizon average reward $\rho^*(s_1)$ which is strictly greater than $\rho^*(s_2)$, thus violating the optimality of $\rho^*(s_2)$. □

Value/bias of a policy. For average reward case, it is conventional to define value of a policy for a given state as the deviation of total reward starting from from the asymptotic average reward. For this reason, in some texts, this is also referred to as the **'bias'** instead of value. (Note that if it was defined as total expected reward, it may not be finite).

$$V^\pi(s) = \lim_{T \to \infty} \mathbb{E}[\sum_{t=1}^{T} r_t - T\rho^\pi | s_1 = s], \forall s.$$

where $\rho^\pi = \lim_{T\to\infty} \mathbb{E}[\frac{1}{T}\sum_{t=1}^{T} r_t | s_1]$, the gain of policy π, is independent of the starting state s_1 under some mild conditions (in particular if the Markov chain defined by policy π is such that all the states communicate, i.e., for any two states i and j there is a positive probability of reaching i from j).

The limit in above is Cesaro limit, which exists, more details in Section 8.2 of Puterman [1994].

Bellman equations for value(actually, bias) of a policy.

$$V^\pi(s) + \rho^\pi = \mathbb{E}_{a\sim\pi(s), s'\sim P(s,a)}\left[R(s,a,s') + V^\pi(s')\right]$$

$$V^\pi + \rho^\pi \mathbf{e} = \mathbf{R}_\pi + P^\pi V^\pi$$

Proof: (assumes finite or countable state space and policy space, and deterministic policy π for simplicity. Similar derivation can be done for randomized policy with notational changes.)

$$\begin{aligned}
V^\pi(s) &= \lim_{T\to\infty} \sum_{t=1}^{T} \mathbb{E}[r_t - \rho^\pi | s_1 = s] \\
&= R(s,\pi(s)) - \rho^\pi + \lim_{T\to\infty} \sum_{s'} \sum_{t=2}^{T} \mathbb{E}[r_t - \rho^\pi | s_2 = s'] P(s,\pi(s),s') \\
V^\pi(s) + \rho^\pi &= R(s,\pi(s)) + \sum_{s'} P(s,\pi(s),s') V^\pi(s')
\end{aligned}$$

Bellman optimality equations. Recall stationary (possibly non-deterministic) optimal policy π^* exists for this setting. And, $\rho^{\pi^*}(s_1) = \rho^*$ is indepedent of the starting state. Let $V^*(s) := V^{\pi^*}(s)$, where $\pi^* = \arg\max_\pi \rho^\pi$, and $\rho^* = \rho^{\pi^*}$. Then,

$$V^*(s) + \rho^* = \max_a R(s,a) + \sum_{s'} P(s,a,s')V^*(s')$$

Proof. Assume finite or countable state space. By definition of bias:

$$V^\pi(s) \quad = \quad \lim_{T\to\infty} \sum_{t=1}^{T} \mathbb{E}[r_t - \rho^\pi | s_1 = s]$$

Let $\pi^* = \arg\max_\pi \rho^\pi$. Then,

$$\begin{aligned}
V^{\pi^*}(s) &= \lim_{T\to\infty} \sum_{t=1}^{T} \mathbb{E}[r_t - \rho^{\pi^*} | s_1 = s] \\
&= R(s,\pi^*(s)) - \rho^{\pi^*} + \lim_{T\to\infty} \sum_{s'} \sum_{t=2}^{T} \mathbb{E}[r_t - \rho^{\pi^*} | s_1 = s'] P(s,\pi^*(s),s') \\
V^{\pi^*}(s) + \rho^{\pi^*} &= R(s,\pi^*(s)) + \sum_{s'} V^{\pi^*}(s') P(s,\pi^*(s),s') \\
\rho^{\pi^*} &= R(s,\pi^*(s)) + \sum_{s'} V^{\pi^*}(s') P(s,\pi^*(s),s') - V^{\pi^*}(s) \\
&\leq \max_a R(s,a) + \sum_{s'} V^{\pi^*}(s') P(s,a,s') - V^{\pi^*}(s)
\end{aligned}$$

Again, by optimality of stationary policy π^*, above must be equality. Therefore,

$$\rho^* = \max_a R(s,a) + \sum_{s'} V^*(s') P(s,a,s') - V^*(s)$$

Example. For MDP in Figure 3, the optimal average reward policy (say π^*) is to take the slow action in all states, with gain of $\rho = 1$. The bias of this policy is $V(F) = (-0.2 - 1) \times (1/0.4) = -3, V(S) = 0, V(M) = 0$. (For calculating $V(F)$, note that the expected number of steps spent in Fallen state when taking slow actions is $1/0.4$, after that the reward of the given policy is 1).

Check that the bias and gain of the optimal policy satisfy the Bellman equations stated above.

$$\underbrace{\begin{bmatrix} -3 \\ 0 \\ 0 \end{bmatrix}}_{V^*} + \underbrace{\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}}_{\rho^* \mathbf{e}} = \underbrace{\begin{bmatrix} -0.2 \\ 1 \\ 1 \end{bmatrix}}_{R^{\pi^*}} + \underbrace{\begin{bmatrix} 0.6 & 0.4 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}}_{P^{\pi^*}} \underbrace{\begin{bmatrix} -3 \\ 0 \\ 0 \end{bmatrix}}_{V^{\pi^*}}$$

Linear programming. For infinite horizon average reward case, the fixed point for Bellman optimality equations can be found by formulating a linear program.

$$\begin{aligned}
\min_{\rho \in R, \mathbf{v} \in \mathbb{R}^S} \quad & \rho \\
\text{subject to} \quad & \rho \geq R(s,a) + P(s,a)^\top \mathbf{v} - v_s \quad \forall a, s
\end{aligned}$$

However, solving LP is slow, and therefore, faster iterative methods are used.

5 Iterative algorithms (discounted reward case)

Using dynamic programming directly may not be very efficient especially for large/infinite horizon case. Below, we discuss some popular iterative methods that are more efficient than linear programming. For succinctness, we limit our discussion primarily to the discounted reward case. Similar algorithms and convergence results are available for the average reward case. For the average reward case, more conditions on the transition matrix are required for convergence. See Chapter 8 of Puterman [1994] for more details.

5.1 Value Iteration.

Indirect method that finds optimal value function (value vector $\mathbf{v}$ in above), not explicit policy.

Pseudocode

1. Start with an arbitrary initialization $\mathbf{v}^0$. Specify $\epsilon > 0$
2. **Repeat** for $k = 1, 2, \ldots$ **until** $\|\mathbf{v}^k(s) - \mathbf{v}^{k-1}(s)\|_\infty \leq \epsilon \frac{(1-\gamma)}{2\gamma}$:
   - for every $s \in S$, improve the value vector as:
$$\mathbf{v}^k(s) = \max_{a \in A} R(s,a) + \gamma \sum_{s'} P(s,a,s') v^{k-1}(s'), \tag{1}$$
3. Compute optimal policy as
$$\pi(s) \in \arg\max_a R(s,a) + \gamma P(s,a)^\top \mathbf{v}^k \tag{2}$$

Bellman operator It is useful to represent the iterative step (1) using operator $L : \mathbb{R}^S \to \mathbb{R}^S$.

$$\begin{aligned} LV(s) &:= \max_{a \in A} R(s,a) + \gamma \sum_{s'} P(s,a,s') V(s') \\ L^\pi V(s) &:= \mathbb{E}_{a \in \pi(s)}[R(s,a) + \gamma \sum_{s'} P(s,a,s') V(s')] \end{aligned} \tag{3}$$

Then, (1) is same as
$$\mathbf{v}^k = L\mathbf{v}^{k-1} \tag{4}$$
Also, for any policy π, if V^π denotes its value function, then, by Bellman equations:
$$V^* = LV^*, V^\pi = L^\pi V^\pi \tag{5}$$

Below is a useful 'contraction' property of this operator, which underlies the convergence properties of all DP based iterative algorithms.

Lemma 6. *The operator $L(\cdot)$ and $L^\pi(\cdot)$ defined by* (3) *are contraction mappings, i.e.,*
$$\|Lv - Lu\|_\infty \leq \gamma \|v - u\|_\infty.$$
$$\|L^\pi v - L^\pi u\|_\infty \leq \gamma \|v - u\|_\infty.$$

Proof. First assume $Lv(s) \geq Lu(s)$. Let $a_s^* = \arg\max_{a \in A} R(s,a) + \gamma \sum_{s'} P(s,a,s') v(s')$

$$\begin{aligned} 0 &\leq Lv(s) - Lu(s) \\ &\leq R(s,a_s^*) + \gamma \sum_{s'} P(s,a_s^*,s') v(s') - R(s,a_s^*) - \gamma \sum_{s'} P(s,a_s^*,s') u(s') \\ &= \gamma P(s,a_s^*)^\top (v - u) \\ &\leq \gamma \|v - u\|_\infty \end{aligned}$$

Repeating a symmetric argument for the case $Lu(s) \geq Lv(s)$ gives the lemma statement. Similar proof holds for L^π. □

Convergence

Theorem 7 (Theorem 6.3.3, Section 6.3.2 in Puterman [1994])**.** *The convergence rate of the above algorithm is linear at rate γ. Specifically,*

$$\|\mathbf{v}^k - V^*\|_\infty \leq \frac{\gamma^n}{1-\gamma}\|v^1 - v^0\|_\infty$$

Further, let π^k be the policy given by (2) using v^k. Then,

$$\|V^{\pi^k} - V^*\| - \infty \leq \frac{2\gamma^k}{1-\gamma}\|v^1 - v^0\|_\infty$$

Proof. By Bellman equations $V^* = LV^*$.

$$\begin{aligned}
\|V^* - v^k\|_\infty &= \|LV^* - v^k\|_\infty \\
&\leq \|LV^* - Lv^k\|_\infty + \|Lv^k - v^k\|_\infty \\
&= \|LV^* - Lv^k\|_\infty + \|Lv^k - Lv^{k-1}\|_\infty \\
&\leq \gamma\|V^* - v^k\| + \gamma\|v^k - v^{k-1}\| \\
&\leq \gamma\|V^* - v^k\| + \gamma^k\|v^1 - v^0\| \\
\|V^* - v^k\|_\infty &\leq \frac{\gamma^k}{1-\gamma}\|v^1 - v^0\|
\end{aligned}$$

Let $\pi = \pi^k$ be the policy at the end of k iterations. Then, $V^\pi = L^\pi V^\pi$ by Bellman equations. Further, by definition of $\pi = \pi^k$,

$$L^\pi v^k(s) = \max_a R(s,a) + \gamma \sum_{s'} P(s,a,s')v^k(s') = Lv^k(s).$$

Therefore,

$$\begin{aligned}
\|V^\pi - v^k\|_\infty &= \|L^\pi V^\pi - v^k\|_\infty \\
&\leq \|L^\pi V^\pi - L^\pi v^k\|_\infty + \|L^\pi v^k - v^k\|_\infty \\
&= \|L^\pi V^\pi - L^\pi v^k\|_\infty + \|Lv^k - Lv^{k-1}\|_\infty \\
&\leq \gamma\|V^\pi - v^k\| + \gamma\|v^k - v^{k-1}\| \\
\|V^\pi - v^k\|_\infty &\leq \frac{\gamma}{1-\gamma}\|v^k - v^{k-1}\| \\
&\leq \frac{\gamma^k}{1-\gamma}\|v^1 - v^0\|
\end{aligned}$$

Adding the two results above:

$$\|V^\pi - V^*\|_\infty \leq \frac{2\gamma^k}{1-\gamma}\|v^1 - v^0\|_\infty$$

□

In average reward case, the algorithm is similar, but the Bellman operator used to update the values is now $LV(s) = \max_a r_{s,a} + P(s,a)^\top V$. Also, here $\mathbf{v}^k$ will converge to $\mathbf{v}^* + c\mathbf{e}$ for some constant c. Therefore, the stopping condition used is instead $sp(v^k - v^{k-1}) \leq \epsilon$ where $\text{sp}(v) := \max_s v_s - \min_s v_s$. That is, span is used instead of L_∞ norm. Further since there is no discount ($\gamma = 1$), a condition on the transition matrix is required to prove convergence. Let

$$\gamma := \max_{s,s',a,a'} 1 - \sum_{j \in S} \min\{P(s,a,j), P(s',a',j)\}$$

Then, linear convergence with rate γ is guaranteed if $\gamma < 1$. This condition ensures that the Bellman operator in this case: is still a contraction. For more details, refer to Section 8.5.2 in Puterman [1994].

5.2 Q-value iteration

$Q^*(s,a)$: expected utility on taking action a in state s, and thereafter acting optimally. Then, $V^*(s) = \max_a Q^*(s,a)$. Therefore, Bellman equations can be written as,

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s,a,s') \left(\max_{a'} Q^*(s',a') \right)$$

Based on above a Q-value-iteration algorithm can be derived:

Pseudocode

1. Start with an arbitrary initialization $\mathbf{Q}^0 \in \mathbb{R}^{S \times A}$.
2. In every iteration k, improve the Q-value vector as:

$$\mathbf{Q}^k(s,a) = R(s,a) + \gamma \mathbb{E}_{s'}[\max_{a'} Q^{k-1}(s',a')|s,a], \forall s,a$$

3. Stop if $\|Q^k - Q^{k-1}\|_\infty$ is small.

5.3 Policy iteration.

Direct method that finds optimal policy.

Pseudocode

1. Start with an arbitrary initialization of policy π^0, and initial value vector $\mathbf{v}^0$ as the value of this policy.
2. In every iteration k, improve the policy as:

$$\pi^k(s) = \arg\max_a R(s,a) + \gamma \mathbb{E}_{s'}[v^{k-1}(s')|s,a], \forall s$$

   And, set $\mathbf{v}^k$ as value of policy π^k.
3. Stop if $\pi^k = \pi^{k-1}$.

For computing value of policy π^k, one can use a similar procedure as value iteration.

5.4 Exercise

Use policy iteration and value iteration to compute optimal policy for the MDP in Figure 3 by hand.

6 Reinforcement learning algorithms

Reinforcement learning is essentially the sequential decision problem when the underlying MDP model (state transition probabilities and reward function) is either unknown or too difficult (large) to solve. We have seen some algorithms (value iteration, policy iteration, linear programming) for solving MDPs. There are two main challenges in using those for reinforcement learning problems:

- The model: $R(s,a), P(s,a,s')$ is not available in reinforcement learning. The model however may be accessible as a blackbox to generate samples. The challenge for RL algorithms is to (implicitly) learn this model from samples, while computing optimal policy. It is therefore important to consider both sample complexity and computation complexity when designing these algorithms.
- Number of states in most RL problems is too large for *tabular* methods like those discussed before to be scalable.

'Modern reinforcement learning' broadly refers to the algorithmic approaches to tackle these challenges, in order to solve a large/unknown MDP by learning and approximation. The focus is largely on scaling up reinforcement learning to make it possible to find complex effective policies for realistic tasks. The algorithmic approaches can be categorized as follows.

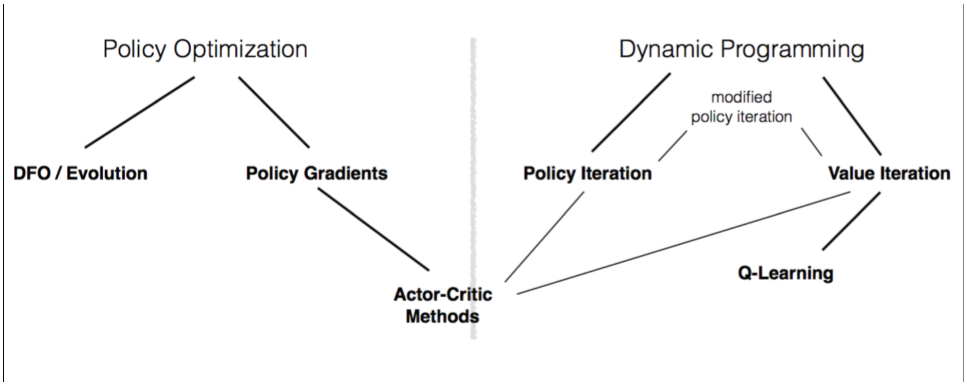


Figure 4: Direct Learning algorithms for RL (Drawing from Pieter Abbeel's slides)

- Direct Learning: Here, the optimal control policy is learned without first learning an explicit model. Such schemes include:
  - Approximate dynamic programming based approaches (Q-learning, TD learning)
  - Direct Policy search (policy gradient, genetic algorithms)
  - and their combinations (Actor-critic)
- Indirect Learning or Model based RL: Estimate an explicit model of the environment, and compute an optimal policy for the estimated model (e.g., Certainty Equivalence and R-MAX). This means learning estimates ($\hat{P}$ and $\hat{R}$) when the model is small but unknown. When the model is large, approximate compact representations of the model are learned. This approach provides a good framework for incorporating prior knowledge about the model into RL algorithms.

References

Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994. ISBN 0471619779.