DYNAMIC PROGRAMMING

MAURICIO GARCIA TEC
ITAM - PRIMAVERA 2017

We shall review some examples and intuition behind the two main dynamic programming algorithms in reinforcement learning, namely:

(1) Policy Iteration
(2) Value Iteration

Both algorithms are based on recursive equations yielded by the Markov Property. In fact, the technique of finding an unknown function defined on the states of a closed system having the Markov Property is what the literature calls *Dynamic Programming*.

The key idea behind the Markov or "lack of memory" property is that

> *The future dynamics of a system having the Markov Property is independent of its past given its current state.*

By "dynamics" we mean the probabilities of the system being in a specific state. The Markov Property extends the traditional modelling of physical system using systems of differential equations, incorporating a probabilistic approach.

1. Reinforcement Learning Tasks and Markov Decision Processes

Reinforcement Learning tasks

A *reinforcement learning task* is the ultimate mathematical abstraction of "learning by experimentation". It uses the language of stochastic processes. A *stochastic process* is a sequence (X_t) of random variables X_t indexed by a time parameter t.

A *reinforcement learning task* starts with an *agent* who takes decisions as he tries to learn from an *environment* outside his control. At each point t in time, the agent finds himself in a situation or state of the world S_t. He then *chooses* to take an action A_t. He may select A_t deterministically or randomly, depending on the current state of the world and his preferred rule of choice. Once the agent has acted, he receives a reward R_{t+1}. The agent's goal is to maximize the total reward he accumulates over time.

Simultaneous to receiving his reward, the agent's situation will change again to a new state of the world S_{t+1}. The agent's decisions may influence the new observed state, but it may also contain an external random component. Now, the agent must act again. This process yields a cycle of decisions, rewards and changes in the state of the world, which compose the elements of a reinforcement learning task.

In summary, a reinforcement learning task is a triple of stochastic processes $(S_t), (A_t), (R_t)$ such that

- (S_t) represents the process of *states* of the world dictated by the environment outside the learning agent's control. We shall denote the set of all states or *state-space* as $\mathcal{S}$.

- (A_t) represents a sequence of *decisions* taken by the agent. The decisions available to the agent depend on the current state of the world. The set of available actions at time t when in state S_t will be denoted $\mathcal{A}(S_t)$, and the set of all actions as $\mathcal{A}(\mathcal{S}) = \bigcup_s \{\mathcal{A}(s) | s \in \mathcal{S}\}$.
- (R_{t+1}) represents the successive *reward*s that the learning agent receives. After taking decision A_t, the agent receives a reward R_{t+1} and the state of the world becomes S_{t+1}

.At a first glance, the definition of a reinforcement learning task is difficult because it contains many elements, but hopefully the reader will find every ingredient a natural generalization of common real life experience.

An important aspect of reinforcement learning tasks is the sequence in which decisions and responses occur.

Given state S_t → Agent acts A_t → Agent receives R_{t+1} and state changes to S_{t+1} → Agent acts A_{t+1} → ⋯

The Markov Property

The Markov property is an additional assumption that greatly simplifies the dynamics of a reinforcement learning task[1]. It formally states

$$\mathbb{P}(S_{t+1} = s', R_{t+1} = r | S_0 = s_0, A_0 = a_0, R_0 = r_0, \dots, S_t = s_t, A_t = a_t, R_t = r_t) \\ = \mathbb{P}(S_{t+1} = s', R_{t+1} = r | S_t = s_t, A_t = a_t)$$

The property says that the future state of a system and the reward received after observing the full history of the learning task, depends solely on the current state and decision taken and not of the past.

The consequence of the Markov Property is that the full dynamics of the reinforcement learning task are determined by two ingredients

(1) The transition probabilities determined by the environment
$$\mathbb{P}(S_{t+1} = s', R_{t+1} = r | S_t = s, A_t = a)$$
which we shall write in a slightly abusive way as $p(s', r | s, a)$.

(2) The *agent's policy*, which is how the agent chooses an action A_t given the current state S_t. We will model the agent's decision as a random choice (it includes the deterministic case). The agent's policy π is a function
$$\pi(a|s) \coloneqq \mathbb{P}(A_t = a | S_t = s).$$

Note: Here I am assuming $p(s', r | a, r)$ and $\pi(s|a)$ do not depend on t. This is what the literature calls a *stationary task*. Many interesting problems are in fact non-stationary, but a general theory is much harder to develop. Nonetheless, some of the algorithms designed for stationary problems also work with non-stationary problems. The difficulty is guaranteeing with a general theorem that the algorithm will lead to a solution.

---

[1] It is an adaptation for the Markov Property of stochastic processes which reads
$$\mathbb{P}(X_{t+1} = x' | X_0 = 0, \dots, X_t = x_t) = \mathbb{P}(X_{t+1} = x' | X_t = x_t)$$
for a stochastic process (X_t).

Knowing the transition probabilities, we can know other interesting quantities. We only need to use the *law of total probability and total expectation and marginalization,* tools that we will use frequently[2]

(1) Transition probabilities between states given an action taken in a specific state

$$p(s'|s,a) = \sum_r p(s',r|s,a)$$

(2) Transition probabilities between states (given the underlying policy, considers all actions)

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

(3) Probability of getting a certain reward given an action taken in a specific state

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

(4) Expected rewards given an action taken in a specific state

$$r(s,a) = \mathbb{E}(R_{t+1}|S_t = s, A_t = a) = \sum_{s',r} rp(s',r|s,a)$$

(5) Expected rewards given a state only (and the underlying policy, considers all actions)

$$r(s) = \mathbb{E}_\pi(R_{t+1}|S_t = t) = \sum_a \pi(a|s)r(s,a) = \sum_{a,s',r} r\pi(a|s)p(s',r|s,a)$$

Markov Decision Processes

A *Markov Decision Process* (MDP) is reinforcement learning task $(S_t), (A_t), (R_t)$ endowed with the Markov Property, together with a policy $\pi(a|s)$ and transition probabilities $p(s',r|s,a)$ so that the full dynamics of the process are specified.

In this course, all our reinforcement learning tasks will be MDPs. The goal of reinforcement learning is to find an optimal policy, so that the total future reward is maximized. We discuss this further in the next section.

Total Future Reward

The concept of *total future reward* encompasses all the reward that the agent will accumulate in the future given its current policy π. Formally, we shall denote the total future reward at time t as a random variable U_t (the letter U stands for utility) given by

$$U_t = R_{t+1} + R_{t+2} + \cdots + R_T$$

[2]Marginalization $\mathbb{P}(X = x) = \sum_y \mathbb{P}(X = x, Y = y)$;
Law of Total Probability $\mathbb{P}(X = x) = \sum_y \mathbb{P}(X = x|Y = y)\mathbb{P}(Y = y)$;
Law of Total Expectation states $\mathbb{E}(X) = \sum_y \mathbb{E}(X|Y = y)\mathbb{P}(Y = y)$.

Where T denotes the (possibly random) end of the task. Some tasks may have $T = \infty$, in this case an adaptation is needed or otherwise U_t would almost always be infinite. A common solution is to include a time-discount factor $0 < \gamma < 1$ and define

$$U_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{n=1}^{\infty} \gamma^{n-1} R_{t+n}$$

Tasks that have $T = \infty$ are called *continuous tasks,* in contrast, tasks with $T < \infty$ are *episodic tasks,* since there is a clear notion of end, and the tasks is repeated many times.

For example, learning how to walk or investing in finance is continuous task, whereas learning to play chess is an episodic task, with an episode being a full match.

Value functions

To search for an optimal policy, it will be important to evaluate how good or bad is a current state in terms of the expected total future reward starting from that state. Another relevant and useful quantity is the expected total future return given the current state and assuming a specific action is taken. The latter quantity is useful when comparing the current policies with policies that suggest actions other than those of the current policy; it is a key ingredient for policy improvement, as it will be clear in the next sections.

These concepts are encompassed in what we call *value functions*:

(1) The *state-value function*

$$v_\pi(s) = \mathbb{E}_\pi(U_t | S_t = s)$$

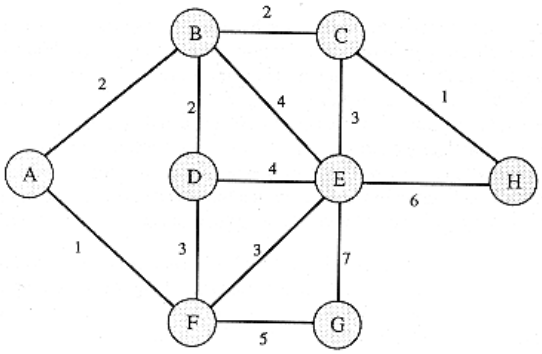
gives the value of the state s under the policy π given the current state s. It is simply the total future reward expected when starting from the current state.

(2) The *action-state value function*

$$q_\pi(s, a) = \mathbb{E}_\pi(U_t | S_t = s, A_t = a)$$

which is the expected total return given state s and assuming the agent takes action a

Example 1 (Finding the shortest path). Suppose you want to find the shortest path to a destination H, starting from any node of a graph, where the available routes are represented by the edge weights of the graph



$v_\pi(s) = -$"path length from s to H according to moving policy π"

For this learning task, we set the rewards R_t after each move to be minus the weight of the edge joining the nodes. The minus is used because rewards are supposed to be good. A long path is a punishment so it is a negative reward. For example, a transition A→B→D→F→E→H would generate rewards $-2, -2, -3, -3, -6$ for a total of -16, which is minus the length of the path. Suppose that an agent will move until it reaches state H.

This is an example of an episodic problem since there is a final step T when the agent reaches H

Examples of possible policies π that the agent could follow are

- Move to the closest node not previously visited (choose alphabetically for tie breakers)
- Move to the furthest node not previous visited (choose alphabetically for tie breakers)
- Move randomly to any adjacent node. This is the first example of a random policy
- Choose the node that leads to the shortest path to H.

Exercise. Try to compute $v(D)$ for the first two policies of the previous example. Now try the third and fourth one. What are their complication in each case?

Exercise. In each of the examples above, what is $q(D, B)$? For the first two cases argue that

$$q(D, B) = 2 + v(B)$$

Find a formula for the other cases.

Optimal Policies

The goal of Reinforcement Learning is to find the optimal policy π_* such that its state-value function v_* satisfies

$$v_*(s) = \max_{\pi} v_\pi(s) \text{ for every state } s \in \mathcal{S}.$$

What the last equation means is that the optimal policy maximizes the value of every state.

Exercise. Make a pause and do not continue until the last equation is obvious to you.

Example. Consider again Example 1 but now choose π_* to be the policy that always moves in the direction that will minimize the length of the path (maximize the total reward). By definition, $v_*(s)$ is the shortest path from s to H. Any other policy π will have longer path to H, that is, will have

$$v_\pi(s) = -\text{path from } s \text{ to } H \text{ under policy } \pi \leq -\text{shortest path from } s \text{ to } H = v_*(s).$$

2. Dynamic Programming

2.1. Evaluating Policies using the State-Value Functions

The Recursive Property of State-Value Functions

The Markov property yields a recursive equation for the state-value function v_π of any policy. This reclusiveness translates into a system of linear equations that gives a solution method to find $v_\pi(s)$ for very state $s \in \mathcal{S}$

$$\begin{aligned} v(s) &= \mathbb{E}_\pi(U_t|S_t = s) \\ &= \mathbb{E}_\pi(R_{t+1} + (\gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots)|S_t = s) \\ &= \mathbb{E}_\pi(R_{t+1}|S_t = s) + \gamma \mathbb{E}_\pi(U_{t+1}|S_t = s) \\ &= \mathbb{E}_\pi(R_{t+1}|S_t = s) + \gamma \sum_{s'} p(s'|s)\mathbb{E}_\pi(U_{t+1}|S_{t+1} = s', S_t = s) \\ &= \mathbb{E}_\pi(R_{t+1}|S_t = s\ ) + \gamma \sum_{s'} p(s'|s)\mathbb{E}_\pi(U_{t+1}|S_{t+1} = s') \\ &= r(s) + \gamma \sum_{s'} p(s'|s)v(s'). \end{aligned}$$

Here we used the Law of Total Expectation for the fourth equality and the Markov Property for the fifth. The quantities $r(s)$ and $p(s'|s)$ where computed in earlier sections from the transition probabilities $p(s', r|s, a)$. Substituting their values, we get

$$v(s) = \sum_{a,r} r\pi(a|s)p(s', r|s, a) + \sum_{s',r,a} \pi(a|s)p(s', r|s, a)v(s')$$

which is the full long form of the recursive equation. Sometimes it is easier to state the equation in terms of $r(s)$ and $p(s'|s)$ only; it is clearer and sometimes directly obtainable from the formulation of the problem.

Observe that the previous equation holds for every $s \in \mathcal{S}$, so we have a full system of linear equations. These are the magical recursive equations that will allow us to find $v(s)$ for each $s \in \mathcal{S}$. These equations are sometimes referred to as *the Bellman Equations*, although the name is also used for the Bellman optimality condition that we will introduce later.

Matrix Notation for State Transitions Probabilities and Expected State Rewards

A matrix or operator notation is helpful to simplify many of the otherwise complicated results of this section. First, we define a state transition probability matrix P_π with each entry (s', s) representing $p(s'|s) = \sum_{a,r} \pi(a|s)p(s', r|s, a)$. Of course, the exact form of the matrix P_π depends on the order of the states, but if we keep the same order always there will be no problem. The matrix P_π will look something like

$$P_\pi = \begin{bmatrix} p(s_1|s_1) & p(s_2|s_1) & \cdots \\ p(s_1|s_2) & p(s_2|s_2) & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$$

We also define a vector R_π with entries $r(s) := \mathbb{E}(R_t|S_t = s) = \sum_{a,r} r\pi(a|s)p(s', r|s, a)$, that is

$$R_\pi = \begin{bmatrix} r(s_1) \\ r(s_2) \\ \vdots \end{bmatrix}$$

Note that in many real applications we know or can estimate the quantities $r(s)$ and $p(s'|s)$ directly without needing $\pi(s|a)$.

The Recursive Property of State-Value Functions (Matrix Form)

This fundamental recursive equation can be written in matrix form as

$$v_\pi = R_\pi + \gamma P_\pi v_\pi$$

Where evidently $v_\pi = [v_\pi(s_1) \cdots v_\pi(s_k) \cdots]^\top$.

Assuming the system has a unique solution (see the warning below), then v_π is given by

$$v_\pi = (I - \gamma P_\pi)^{-1} R_\pi,$$

where I is the identity matrix. Coming from the jargon of discrete-time Markov processes, the matrix $(I - \gamma P)^{-1}$ is known as the fundamental matrix.

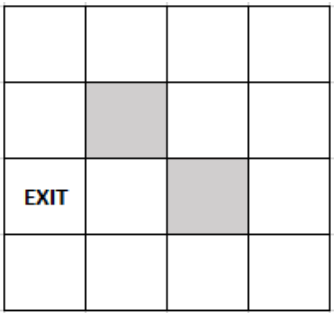
Technical but important warning! Episodic tasks are usually so because they have a final state that will end the game. In the jargon of Markov Processes, they have *absorbent states*. If s' is an absorbent state, then $p(s'|s)$ will be equal to 1 if $s' = s$ and 0 otherwise. What is the problem? Well, in episodic tasks we often set $\gamma = 1$, this will imply that the matrix $I - \gamma P_\pi$ will not be invertible as rows corresponding to absorbent states will be all 0's. But if we think about it... What should $v(s')$ be for an absorbent state s'? The answer is $v(s') = 0$. Recall that the definition of $v(s')$ is the total *future* reward to be received when starting from s', but if s' marks the end of the game, then no reward will be received. The conclusion is that we need not estimate $v(s')$ as we already know it is zero. Hence, for episodic tasks we *must* remove the rows and columns in P_π and R_π corresponding to absorbent states. For the numerical algorithm that we will see in the next section it is not necessary, although we must still recall that the value-function of absorbent states should be set to zero from start.

Estimating v_π is the most fundamental step in reinforcement learning. Policies are judged based on their value functions. If $v_\pi \leq v_{\pi'}$ then π' improves π. This establishes a partial order in the space of policies. The supreme champion optimal policy π_* has $v_* \geq v_\pi$ for all policies π.

Exercise 1 (Finding the shortest path continued). In Example 1 (Finding the shortest path), use any computer program to find v_π solving directly the system of linear equations $v_\pi = (I - \gamma P_\pi)^{-1} R_\pi$ for the case when π is the random policy that takes a step to any of the adjacent

nodes with equal probability. Remember to remove the row and column corresponding to the node H from P_π and R_π (see the technical warning above). What do you conclude?

Exercise 2 (Grid-world). Here is another example equivalent to the graph previous graph problem. Suppose we want to escape from a labyrinth, or more precisely a grid. There is one exit and two obstacles:



4x4 grid example, the grey squares are obstacles

The agent's objective for this task is to reach the exit block. **(a)** Suppose that the agent's policy is to move with equal probability to any of the adjacent blocks (states) that are not obstacles in the directions (actions) *up, right, down, left.* He receives a reward of -1 for any move until he reaches the exit. Compute the value of every state. Give an interpretation of the exact number obtained. Can you imagine an optimal path using the outcome? **(b)** Do the same as before but suppose now that it is possible to move in diagonals as well. What do you conclude from the differences in state values? ***Hint:*** You don't need to work twice. If you convert the problem into a graph with edge weights equal to one you can use the solution method of Exercise 1.

Iterative Policy Evaluation: The Numerical Algorithm to find v_π

There is a practical numerical way to solve the equation $v_\pi = R_\pi + \gamma P_\pi v_\pi$ in an iterative way. This is achieved using the *fixed-point iteration method*. The allure of this method is that sometimes we only need approximate solutions as we want to quickly improve the current policy and not spend too much time evaluating it.

A *fixed-point* of a continuous function B is a point such that $B(x_*) = x_*$. We are obviously interested in finding fixed-points since the value function v_π of a policy π is a fixed point of the operator

$$B_\pi(x) \coloneqq R_\pi + \gamma P_\pi x.$$

The fixed-point iteration method is a technique to find this point. To derive it, we first note that if the limit

$$x_* = \lim_{n\to\infty} B^n(x_0)$$

exists, with x_0 *any* point, then it most also be true that

$$x_* = \lim_{n\to\infty} B^n(x_0) = \lim_{n\to\infty} B\big(B^{n-1}(x_0)\big) = B\left(\lim_{n\to\infty} B^{n-1}(x_0)\right) = B(x_*),$$

so x_* would be a fixed point. For the operator $B_\pi(x) \coloneqq R_\pi + \gamma P_\pi x$ the condition that guarantees that with any starting point $v_\pi^{(0)}$ we will achieve a limit

$$v_\pi = \lim_{k\to\infty} B_\pi^k\left(v_\pi^{(0)}\right),$$

is the *contraction property* of the mapping B_π and Banach's contraction mapping theorem. We will not deal with the details of this theorem. A contraction is mapping B such that

$$\|B(x) - B(y)\| < \eta\|x - y\|,$$

for some $0 \leq \eta < 1$ and any choice of norm $\|\cdot\|$. These guarantees that with each iteration

$$v_\pi^{(k+1)} = B_\pi\left(v_\pi^{(k)}\right) = R_\pi + \gamma P_\pi v_\pi^{(k)}$$

the difference between the value of $v_\pi^{(k+1)}$ and $v_\pi^{(k)}$ goes to zero, guaranteeing convergence.

In conclusion, the Iterative Policy Evaluation Algorithm, which is a widely-used form of dynamic programming for solving the Bellman recursive equations for the value-state functions goes as follows:

Iterative Policy Evaluation Algorithm (Non-matrix form)

Inputs: The policy π to be evaluated and the transition probabilities $p(s', r|s, a)$

Output: The value $v_\pi(s)$ of every state $s \in \mathcal{S}$ with respect to policy π

1. **Initialize** $v_\pi^{(0)} = 0$ for each $s \in \mathcal{S}$.
2. **For** $k = 1,2,\ldots$ and **for each** $s \in \mathcal{S}$ until convergence, compute
$$v_\pi^{(k)}(s) = \sum_{a,r} r\pi(a|s)p(s', r|s, a) + \gamma \sum_{s',r,a} \pi(a|s)p(s', r|s, a)v_\pi^{(k-1)}(s')$$

Iterative Policy Evaluation Algorithm (Matrix form)

Inputs: The policy π to be evaluated and the transition probabilities $p(s', r|s, a)$

Output: The vector v_π of state values with respect to policy π

1. **Initialize** $v_\pi^{(0)} = 0$ for each $s \in \mathcal{S}$.
2. **Compute** P_π and R_π with the formulas
$$r(s) \coloneqq \sum_{a,r} r\pi(a|s)p(s', r|s, a) \quad \text{and} \quad p(s'|s) \coloneqq \sum_{s',r,a} \pi(a|s)p(s', r|s, a)$$
3. **For** $k = 1,2,\ldots$ until convergence, compute
$$v_\pi^{(k)} = R_\pi + \gamma P_\pi v_\pi^{(k-1)}$$