Dynamic Programming

Diego Ascarza-Mendoza

Escuela de Gobierno y Transformación Pública

Outline

Introduction

- Dynamic Programming (DP) refers to a set of algorithms that can be used to compute optimal policies in a perfect model of the environment as a MDP.
- DP is super useful, especially in economics. In rocket science applications that are unique to RL they are more limited:
  1. Assumption of perfect model.
  2. Great computational expense.
- Why do we study it then?
  1. They provide foundations for understanding the rest of methods that we study.
  2. Provide important theoretical predictions.
- The key idea is that we will assume that $\mathcal{S}$, $\mathcal{A}$ and R are finite and that $p(s', r|s, a)$ is well defined $\forall s \in \mathcal{S}$, $a \in \mathcal{A}$, $r \in \mathcal{R}$ and $' \in \mathcal{S}^+$.

- The key idea of DP and in RL generally is the use of value functions to organize and structure the search for good policies.
- In this lecture, we learn how DP can be used to compute the value functions defined in our previous chapter.
- Remember that one can obtain optimal policies once we find v_* or q_*, which satisfy the optimal Bellman equations:

$$\begin{aligned} v_*(s) &= \max_a \mathbb{E}\left[R_{t+1} + \gamma v_*(S_{t+1}|S_t = s_t, A_t = a\right] \\ &= \max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma v_*(s')\right], \quad or \end{aligned} \tag{1}$$

$$\begin{aligned} q_*(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a')|S_t = s, A_t = a\right] \\ &= \sum_{s',r} p(s', r|s, a)\left[r + \gamma \max_{a'} q_*(s', a')\right], \end{aligned} \tag{2}$$

- In DP, basically, we will turn Bellman equations into assignments (update rules for improving approximation of desired VF).

Outline

Policy Evaluation (Prediction)

- The first step in DP is computing the state-value function v_π for an arbitrary policy π.
- In the literature, this is called *policy evaluation* or to the *prediction problem*. Recall:

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \mathbb{E}_\pi\left[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s\right] \qquad (3) \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')], \qquad (4) \end{aligned}$$

- It can be shown that v_π is unique as long as either $\gamma < 1$ or eventual termination is guaranteed from all states under π.
- If the environment's dynamics are completely known, then to find v_π we have to solve a system of $|\mathcal{S}|$ simultaneous linear equations in $|\mathcal{S}|$ unknowns. That is too costly.
- For our purposes, iterative solution methods are most suitable.

- Consider a sequence of approximate VFs $v_0, v_1, \ldots$, each mapping $\mathcal{S}^+$ to $\mathbb{R}$. The initial approximation v_0 is chosen arbitrarily (except the terminal state, which must be given value 0).
- Also, the successive approximation is obtained using Equation 4 as an update rule:

$$\begin{aligned} v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s'r} p(s', r|s, a)[r + \gamma v_k(s')], \quad \forall s' \end{aligned} \tag{5}$$

- Notice that $v_k = v_\pi$ would be a fixed point for this update rule, because the Bellman equation for v_π assures us of equality in this case.
- **It can be shown** that $\{v_k\}$ in general converges to v_π as $k \to \infty$ under the same conditions that guarantee existence of v_π. This algorithm is called *iterative policy evaluation*.

Iterative Policy Evaluation

- To obtain v_{k+1} from v_k, iterive policy evaluation applies the same operation to each state s: replace old value of s with a new that is obtained from the old values of successors of s and current rewards.
- This operation is called *expected update*.
- In DP algorithms, all the updates are called expected updates because they are based on expectations over all possible states (not a sample).
- How do we put this in the computer? you need two arrays:
  1. One for the old values $v_k(s)$.
  2. One for the new values $v_{k+1}(s)$
- Compute the new values one by one from old values. We think of the updates as being done in a *sweep* through the state space.
- We will always have a 'in-place' way of doing updates in DP: use one array and update the values "in place": with each new value immediately overwrite the old one.

Iterative Policy Evaluation (algorithm)

Input: π to be evaluated, and parameter θ that is a threshold to determine the accuracy of estimation.

- **Initialize:** For each $s \in \mathcal{S}$, initialize $V(s)$, arbitrarily except for terminal state.
- Loop:
  - $\Delta \leftarrow 0$ Loop for each $s \in \mathcal{S}$:
  - v$\leftarrow V(s)$.
  - $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$.
  - $\Delta \leftarrow max(\Delta, |v - V(s)|)$

  until $\Delta < \theta$

Example

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

There are 14 nonterminal states $\mathcal{S} = \{1, 2, ..., 14\}$ and four possible actions in each state $\mathcal{A} = \{up, down, right, left\}$ except for those that take the agent out of the grid. The reward is -1 until the terminal state is reached. What is v_π for the equiprobably random policy in each iteration?

Outline

Policy Improvement

- We compute the value of functions for policies to find better policies!
- Suppose we have found v_π for some arbitrary policy.
- For some state s we would like to know whether we should change the policy to instead pick an action $a \neq \pi(s)$.
- We know how good is to follow π but would it be better or worse to change to the new policy?

- A way to answer the previous question is to consider selecting a in s and thereafter following the existing policy, π. The value of this way of behaving is:

$$\begin{aligned} q_\pi(s,a) &\doteq \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t, A_t = a] \\ &= \sum_{s',r} p(s',r|s,a)[r + \gamma v_\pi(s')]. \end{aligned} \tag{6}$$

- The key criterion is whether this is greater than or less than $v_\pi(s)$.
- If it is greater, then one would expect it to be better still to select a every time s is encountered, and that the new policy would in fact be a better one overall.

- This is the special case of a general result called *policy improvement theorem*.
- Intuitively, it says that if selecting a every time s is encountered and then follow π the rest of the time, then that new policy in fact would be a better one overall.
- Formally, let π and π' be any pair of deterministic policies such that, $\forall s \in \mathcal{S}$,

$$q_\pi(s, \pi'(s)) \geq v_\pi(s). \tag{7}$$

Then the policy π' must be as good as or better than π. This is, it must satisfy:

$$v_{\pi'}(s) \geq v_\pi(s). \tag{8}$$

If equation 7 holds at any state, then there must be strictly 8. It can be seen that 7 holds for any other state. Therefore, the changed policy is indeed better than π

Sketch of proof (policy improvement theory)

We know:

$$
\begin{aligned}
v_\pi(s) \leq q_\pi(s, \pi'(s)) &= \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = \pi'(s)\right], \\
&= \mathbb{E}\left[R_{t+1} + \gamma v_\pi(S_{t+1} | S_t = s, A_t = \pi'(s)\right], \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s\right], \\
&\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) | S_t = s], \\
= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma v_\pi(S_{t+2}) | S_{t+1}, A_{t+1} = \pi'(S_{t+1})] | S_t = s\right], \\
&= \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) | S_t = s\right] \\
&\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) | S_t = s\right]
\end{aligned}
$$

If we keep doing this operation:

$$
\leq \mathbb{E}_{\pi'}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... | S_t = s\right] = v_{\pi'}(s).
$$

Policy improvement

- We have seen how given a policy and its value function, we can easily evaluate a change at a single state.
- We can then consider changes at all states and to all possible actions, selecting at each state the action that appears best according to $q_\pi(s,a)$. In other words, consider the new *greedy policy* π', given by:

$$\begin{aligned}\pi'(s) &\doteq \arg\max_a q_\pi(s,a) \\ &= \arg\max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a] \quad (9) \\ &= \arg\max_a \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_\pi(s')\right], \quad (10)\end{aligned}$$

- This policy takes the action that looks best after one step of lookahead according to v_π.
- The policy meets the condition of the policy improvement theory (by construction), so it is as good or better than the original policy.
- The process of making a new policy that improves on the original policy, by making it greedy with respect to the value function of the original policy, is called *policy improvement*.

- Now, suppose the new greedy policy π' is as good but not better than π. Then: $v_\pi = v_{\pi'}$, and from equation 9 it follows that:

$$v_{\pi'}(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a]$$

$$= \max_a \sum_{s',r} p(s'r|s,a)[r + \gamma v_{\pi'}(s')].$$

- This is the same as the Bellman optimality equation!!! Therefore, $v_{\pi'} = v_*$ and both π and π' must be optimal policies.
- Therefore, policy improvement must give us a strictly better policy except when the original policy is already optimal.

Stochastic Policies

- So far, we have considered the special case of deterministic policies.
- In the general case, a policy π specifies probabilities $\pi(a|s)$, for each action, in each state, s.
- It can be shown that all the ideas of this section extend to the stochastic case: the policy improvement theorem works for the stochastic case.
- Furthermore, if there are ties in policy improvement steps, then in the stochastic case, we need not select a single action from among them (mixed strategy).

Outline

Policy Iteration

- Once a policy π has been improved using v_π to yield a better policy π', we can compute $v_{\pi'}$ and improve until we obtain a sequence of improving policies:

$$\pi_0 \xrightarrow{\mathsf{E}} v_{\pi 0} \xrightarrow{\mathsf{I}} \pi_1 \xrightarrow{\mathsf{E}} v_{\pi 1} \xrightarrow{\mathsf{I}} \pi_2 ... \xrightarrow{\mathsf{I}} \pi_* \xrightarrow{\mathsf{E}} v_*,$$

- Because MDP has only a finite number of policies, this process must converge to an optimal policy and optimal value function.
- This way of finding optimal policy is called *policy iteration*.

Policy Iteration - Algorithm

1. Initialization: set up initial policies and value functions $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)\ \forall s \in \mathcal{S}$.
2. Policy Evaluation Loop:

   $$\Delta \leftarrow 0$$

   Loop for each $s \in \mathcal{S}$:

   $$v \leftarrow V(s)$$
   $$V(s) \leftarrow \sum_{s',r} p(s'r|s, \pi(s))[r + \gamma V(s')]$$
   $$\Delta \leftarrow max(\Delta, |v - V(s)|)$$

   until $\Delta < tol$.
3. Policy Improvement: start setting *policy-stable*$\leftarrow$*true*. For each $s \in S$:

   $$\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')].$$

   If *old-action*$\neq \pi(s)$, then *policy-stable*$\leftarrow$false. If true, stop and return $V \approx v_*$ and $\pi \approx \pi_*$. Else, go to 2.

Car Rentals

- Jack manages two locations for a car rental company.
- Every day, some customers arrive at each location to rent cars.
- If a car is available, he rents it out and is credited $10. If he is out of cars at that location, business is lost.
- Cars become available the day after they are returned.
- Moving one car from one location to the other costs 2$ per car.
- Cars requested and returned at each location are Poisson random variables: $\frac{\lambda^n}{n!}e^{-\lambda}$. $\lambda = 3$ and 4 for rental requests for first and second location, and $\lambda = 3$ and 2 for returns.
- For simplicity, assume there can be no more than 20 cars at each location, and you cannot move more than five cars from one location to another in one night. Assume $\gamma = 0.9$.

Car Rentals

- Time steps are days.
- What are the state variables?
- What are the actions?
- How do we solve this problem using Policy-iteration?

We are going to see that policy iteration often converges in surprisingly few iterations.

Outline

Value Iteration

- There is a problem with policy iteration: it involves policy evaluation in each of its iterations.
- If policy evaluation is done iteratively, convergence to exactly v_π happens only in the limit.
- Should we wait until convergence, or maybe we can truncate it?
- It turns out that the policy evaluation step can be truncated without losing the guarantee of convergence of policy iteration. In particular, after just one sweep.
- This case is called *value iteration*.
- This can be written as a simple update operation that combines the policy improvement and truncated policy evaluation steps:

$$\begin{aligned} v_{k+1}(s) &\doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) | S_t = s, A_t = s] \\ &= \max_a \sum_{s',r} p(s', r|s, a) \left[r + \gamma v(k') \right]. \end{aligned} \tag{11}$$

Value Iteration

- For an arbitrary v_0, it can be shown that a sequence $\{v_k\}$ converges to v_* under the same conditions that guarantee the existence of v_*.
- One way of seeing value iteration is by noticing that it uses the Bellman optimality equation (2) as an update rule.
- It can also be seen that the value iteration update is identical to the policy evaluation update 4, except that it requires the maximum to be taken over all actions.
- Like policy evaluation, value iteration formally requires an infinite number of iterations to converge exactly to v_*.
- In practice, we stop once the value function changes by a small amount in a sweep. Let's go with the algorithm.

Value Iteration - Algorithm

set up a small threshold $\theta > 0$ determining accuracy of estimation.

1. Initialization: set up $V(s)\ \forall s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$.
2. Loop:

   $$\Delta \leftarrow 0$$

   Loop for each $s \in \mathcal{S}$:

   $$v \leftarrow V(s)$$
   $$V(s) \leftarrow \max_a \sum_{s',r} p(s', r|a, s)[r + \gamma V(s')]$$
   $$\Delta \leftarrow max(\Delta, |v - V(s)|)$$

   until $\Delta < tol$.
3. Output a deterministic policy, $\pi \approx \pi_*$, such that:

   $$\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')].$$

Value Iteration

- Value iteration combines, in each of its sweeps, one sweep of policy evaluation and one sweep of policy improvement.
- Faster convergence is often achieved by interposing multiple policy evaluation sweeps between each policy improvement sweep.
- In general, the entire class of truncated policy iteration algorithms can be thought of as sequences of sweeps, some of which use policy evaluation updates and some of which use value iteration updates.
- Using the max operation is the only difference between these updates. All of these updates converge to an optimal policy for discounted finite MDPs.

Outline

Asynchronous Dynamic Programming

- There is a big issue with the DP methods we just discussed: they involve operations over the entire state set of the MDP.
- If the state set is very large, then even a single sweep can be prohibitively expensive. For instance, in the game of backgammon, there are 10^{20} states.
- Even if we could perform the value iteration on a million states per second, it would take over a thousand years to complete a single sweep.
- *Asynchronous* DP algorithms are in-place iterative DP algorithms that are not organized in terms of systematic sweeps of the state set.
- These algorithms update the values of states in any order whatsoever, using whatever values of the other states happen to be available.
- To converge correctly, these algorithms must continue to update the values of all states. It can't ignore any state after some point in the computation.
- These algorithms offer great flexibility in selecting states to update.

Asynchronous Dynamic Programming

- A version of these algorithms updates the value, in place, of only one state, s_k, on each step k, using the value iteration update as in 11.
- If $0 \leq \gamma \leq 1$, convergence is guaranteed given only that all states occur in the sequence $\{s_k\}$ an infinite number of times
- Similarly, it is possible to intermix policy evaluation and value iteration uipdates to produce a kind of asynchronous truncated policy iteration.
- Obviously, avoiding sweeps does not necessarily mean that we can get away with less computation. It just means that an algorithm does not need to get locked into any hopelessly long sweep before making progress to improve policy.
- Asynchronous algorthms also make it easier to intermix computation with real-time interaction: we can run an iterative DP algorithm *at the same time that an agent is actually experiencing the MDP*.

Generalized Policy Iteration

- Policy iteration consists of two simultaneous, interacting processes: policy evaluation and policy improvement.
- In policy iteration, these two processes alternate, each completing before the other begins, but *this is not really necessary*.
- In value iteration, for instance, only a single iteration of policy evaluation is performed between each policy improvement.
- In asynchronous DP methods, evaluation and improvement processes are interleaved at an even finer grain (sometimes you even only update a single state before returning to the other).
- As long as both processes continue to update all states, the ultimate result is typically the same (convergence to optimal value and policy).
- We use the term *generalized policy iteration (GPI)* to refer to the general idea of letting policy-evaluation and policy-improvement processes interact, independent of the granularity and other details of the two processes.

Generalized Policy Iteration

- Almost all RL methods are well described as GPI.
- This is because all have identifiable policies and value functions, with the policy always being improved with respect to the value function and the value function always being driven toward the value function for the policy.
- The value function stabilizes only when it is consistent with the current policy, and the policy stabilizes only when it is greedy with respect to the current value function.
- This is equivalent to saying that the Bellman optimality equation 2 holds, and thus the value and policy function are optimal.
- Evaluation and improvement in GPI are both competing and cooperating:
  1. They compete in the sense that they pull in opposing directions (being greedy makes the value function incorrect for the change policy), and making the value function consistent with the policy typically causes that policy no longer to be greedy.
  2. In the long run, however, these two processes interact to find a single joint solution: optimal value function and an optimal policy.

Efficiency of Dynamic Programming

- DP may not be practical for very large problems, but compared with other methods for solving MDPS, DP methods are actually quite efficient.
- In the worst case, the time that DP methods take to find an optimal policy is polynomial in the number of states and actions.
- This is exponentially better than any direct search in policy space could be, because direct search would have to examine each policy to provide the same guarantee exhaustively.
- Linear programming methods can be sometimes faster than DP but they become impractical at a much smaller number of states than do DP methods (by a factor of 100).
- The big issue with DP in terms of applicability is the *curse of dimensionality*. The number of states grows exponentially with the number of state variables.
- This is a complication of the problem itself, not of DP. Actually, DP is comparatively better suited to handling large state spaces compared to direct search and linear programming. With our computers now we can use it for a large variety of problems.

Summary

- Policy evaluation refers to the iterative computation of the value function for a given policy.
- Policy improvement refers to the computation of an improved policy given the value function for that policy.
- Putting these two computations together, we obtain *policy iteration* and *value iteration*, the two most popular DP methods.
- Classic DP methods operate in sweeps through the state set, performing an *expected update* operation in each state. When those sweeps do not generate more change, we say that we converged to values that satisfy the corresponding Bellman equation.
- Generalized Policy Iteration is the general idea of two interacting processes revolving around an approximate policy and an approximate value function.
- One performs some policy evaluation to get a closer approximation of the value function. The other takes the value function as given and performs some form of policy improvement, changing the policy to make it better.
- all DP methods update estimates of the values of states based on estimates of the values of successor states (this general idea is called *bootstrapping*).