

CS497 Planning and Decision Making Week 5 Notes

(2/18, 2/20)

Scribe: Shiau Hong Lim

1 Cycles and Termination

Assuming that termination actions are to be applied when the goal state is reached, but the number of stages, K is unknown. We need the following conditions to ensure that the dynamic programming algorithm will terminate:

- Non-deterministic
 1. No negative (in a minimax sense) cycles
 2. Must be able to escape positive cycles
- Probabilistic
 - None of the above, with probability 1

Consider the following example:

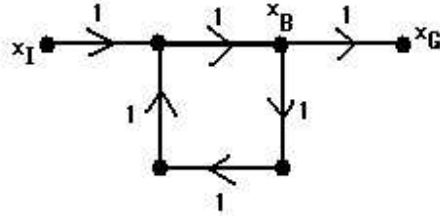


Figure 1: Cycle example

The loss for every state transition is 1. At state x_B , the probability of going to either of the next two states is $1/2$. The expected loss is therefore,

$$E[L] = \frac{1}{2}(3) + \frac{1}{4}(7) + \cdots = 3 + \sum_{i=0}^{\infty} \frac{1}{2^{i+1}}(4i) < \infty$$

which is finite. Problem arises when we perform dynamic programming iteration that stops only when L^* values stabilize. In the above scenario, there will be a decreasing, non-zero difference between subsequent L_k^* due to the cycle. To ensure termination of the algorithm, we introduce an error-term ϵ such that the algorithm stops when

$$\max_{x \in X} |L_{k+1}^*(x) - L_k^*(x)| < \epsilon$$

2 Infinite Horizon Markov Decision Process

In an infinite-horizon Markov Decision Process (MDP), the number of stages is unbounded. It can be viewed as a game that never ends — you have to play forever. Many real-life problems can be modeled as infinite-horizon MDPs. One consequence is that there are no specific goal states. Our objective is therefore to minimize the cost function

$$L = \lim_{n \rightarrow \infty} \sum_{i=1}^n l(x_i, u_i)$$

The problem is that L may be unbounded. To fix this problem, we have the following options:

1. *Average cost-per-stage*

where instead of L as described above, we try to minimize

$$\lim_{K \rightarrow \infty} \frac{1}{K} E \left\{ \sum_{i=1}^{K-1} l(x_i, \gamma(x_i), \theta) \right\}$$

For many problems, the average cost-per-stage is well-defined and finite.

2. *Discounted loss*

Let $\alpha \in (0, 1)$ denote a discount factor. We define the loss function as

$$L = \lim_{K \rightarrow \infty} \left\{ \sum_{i=1}^{K-1} \alpha^i l(x_i, u_i, \theta_i) \right\}$$

where $u_i = \gamma(x_i)$, and γ is the strategy/policy. Since α is less than one, L is finite. The problem is then to choose γ that optimizes:

$$\lim_{K \rightarrow \infty} E \left\{ \sum_{i=1}^{K-1} \alpha^i l(x_i, u_i, \theta_i) \right\}$$

The discount factor, α determines the importance of future loss. A large α will result in slow convergence, but give more weight to the future.

From now on, we shall focus on the second option.

2.1 Forward Dynamic Programming

We would like to optimize:

$$\lim_{K \rightarrow \infty} E_{\theta_i} \left\{ \sum_{i=1}^{K-1} \alpha^i l(x_i, u_i, \theta_i) \right\}$$

If we assume a finite K for now, and let $l_i = l(x_i, u_i, \theta_i)$. As K increases, we get the following sequence of L_K^* ,

$$\begin{aligned} K=1, & \quad L_1^* = 0 \\ K=2, & \quad L_2^* = l_1 \\ K=3, & \quad L_3^* = l_1 + \alpha l_2 \\ K=4, & \quad L_4^* = l_1 + \alpha L_2 + \alpha^2 l_3 = l_1 + \alpha(l_2 + \alpha l_3) \\ & \quad \vdots \end{aligned}$$

We can visualize this sequence as in Figure 2. For every new stage, the forward dynamic programming adds a level to the top.

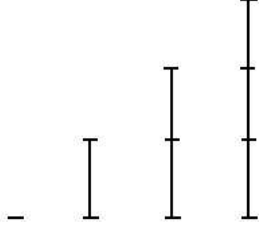


Figure 2: Forward growth

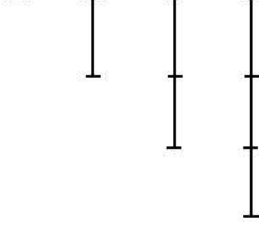


Figure 3: Backward growth

2.2 Backward Dynamic Programming

What we really want is a backward dynamic programming algorithm, which is usually more natural to implement. This is visualized in Figure 3. For every new stage, the backward dynamic programming adds a level to the bottom. Consequently, we need to multiply the previous values by α (pushing them to the future) in order to make use of the previous stage's result. Assuming that K is fixed, we have the following update equations:

$$L_K^*(x) = 0 \quad \forall x \in X$$

$$L_k^*(x) = \min_{u_k \in U(x)} E_{\theta_k} \left\{ \alpha^k l(x_k, u_k, \theta_k) + L_{k+1}^*(f(x_k, u_k, \theta_k)) \right\}$$

We define

$$J_{K-k}^*(x_k) = \alpha^{-k} L_k^*(x_k)$$

Substitute J for L , we get

$$\alpha^k J_{K-k}^*(x_k) = \min_{u_k \in U(x_k)} E_{\theta_k} \left\{ \alpha^k l(x_k, u_k, \theta_k) + \alpha^{k+1} J_{K-k-1}^*(f(x_k, u_k, \theta_k)) \right\}$$

Divide both sides by α^k , and let $i = K - k$,

$$J_i^*(x_k) = \min_{u_k \in U(x_k)} E_{\theta_k} \left\{ l(x_k, u_k, \theta_k) + \alpha J_{i-1}^*(f(x_k, u_k, \theta_k)) \right\}$$

From the above equation, J_i^* can be interpreted as the expected loss for an i -stage optimal strategy. It can be shown that for finite X , U and Θ , $J_i^*(x) \rightarrow J^*(x)$ as $i \rightarrow \infty$, where $J^*(x)$ is the optimal value function for an infinite-horizon MDP. So we have

$$J^*(x) = \min_{u \in U(x)} E_{\theta} \left\{ l(x, u, \theta) + \alpha J^*(f(x, u, \theta)) \right\}$$

How do we find J^* ? Two common ways: *value iteration* and *policy iteration*.

2.3 Value Iteration

Also known as *cost-to-go iteration* or *cost-to-come iteration*. It basically performs a greedy policy update until the J values converge:

1. Initialize $J_0^*(x) = 0 \quad \forall x \in X$

2. Calculate $J_1^*(x), J_2^*(x), \dots$

3. Until

$$\max_{x \in X} |J_{i+1}^*(x) - J_i^*(x)| < \epsilon$$

2.4 Policy Iteration

Given a fixed strategy γ , we evaluate the strategy by

$$J_\gamma(x) = E_\theta \left\{ l(x, u, \theta) + \alpha J_\gamma(x') \right\}$$

Suppose nature does not directly affect loss, i.e. $l(x, u, \theta) = l(x, u) \forall x, u, \theta$, then

$$J^*(x) = \min_{u \in U(x)} \left\{ l(x, u) + \alpha \sum_{x'} P(x'|x, u) J^*(x') \right\} \quad (1)$$

and

$$J_\gamma(x) = l(x, u) + \alpha \sum_{x'} P(x'|x, u) J_\gamma(x') \quad (2)$$

We perform policy iteration as follows,

1. Guess an initial strategy γ
2. Evaluate γ using Equation (2)
3. Use Equation (1) to find an improved γ (greedily)

Example

Consider the following 2-state MDP.

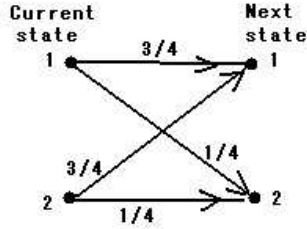


Figure 4: $u = a$

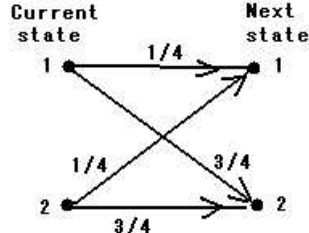


Figure 5: $u = b$

We have $X = \{1, 2\}$, $U = \{a, b\}$ and we assume $l(x, u, \theta) = l(x, u)$. We pick $\alpha = 9/10$. Let the expected loss be $l(1, a) = 2$, $l(1, b) = 1/2$, $l(2, a) = 1$ and $l(2, b) = 3$. The first step of policy iteration is to guess a γ , so let $\gamma(1) = a$ and $\gamma(2) = b$. To evaluate γ , we calculate

$$J_\gamma(1) = 2 + \frac{9}{10} \left[\frac{3}{4} J_\gamma(1) + \frac{1}{4} J_\gamma(2) \right]$$

$$J_\gamma(2) = 3 + \frac{9}{10} \left[\frac{1}{4} J_\gamma(1) + \frac{3}{4} J_\gamma(2) \right]$$

Solving the linear system, we obtain $J_\gamma(1) \approx 24.12$ and $J_\gamma(2) \approx 25.96$. For step three, we find a new γ' using

$$J'(x) = \min_{u \in U(x)} \left\{ l(x, u) + \alpha \sum_{x'} P(x'|x, u) J_\alpha(x') \right\}$$

and we get $J'(1) = 23.45$, $J'(2) = 23.12$ with the corresponding $\gamma'(1) = b$ and $\gamma'(2) = a$. We then repeat the process again (with γ') until convergence (within ϵ).

3 Reinforcement Learning

So far we have been assuming that $P(x'|x, u)$ is known, i.e. the transition function / model / nature is known. What if it is not known? We learn it. The traditional view of the entire optimization process involves three phases of operation:

1. Learning phase — get $P(x'|x, u)$
2. Planning phase — get γ
3. Execution phase — use γ

It turns out that we can actually combine all these together in one well-defined operation. This is *reinforcement learning*. It is essentially a learning-by-doing algorithm, where we interact with the world for a large number of times and based on the experience, obtain an optimal loss function as well as γ . The world can usually be simulated using a Monte Carlo Simulator, where it provides feedback to the algorithm based on the chosen action as shown below.

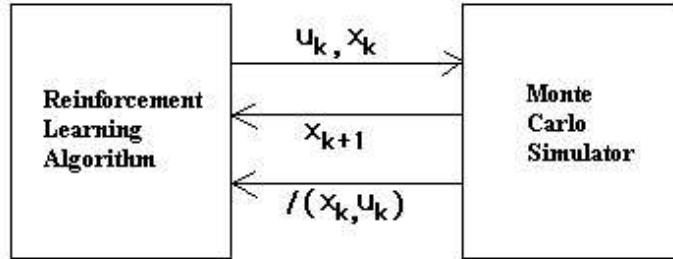


Figure 6: Reinforcement learning

3.1 Evaluating a Strategy

The evaluation function is again given as:

$$J_\gamma(x) = l(x, u) + \alpha \sum_{x'} P(x'|x, u) J_\gamma(x')$$

But this time, we do not know $P(x'|x, u)$. It turns out that we can estimate $J_\gamma(x)$ through repetitive observation and update during the simulation. As the number of run increases, our estimation of $J_\gamma(x)$ will approach the correct values. This is an outcome of the *stochastic iterative algorithm*. It

basically says that given a fix point $y = h(y)$, we can estimate y by observing a “noisy” version of h and using the following update equation:

$$y \leftarrow (1 - \rho)y + \rho(h(y)), \quad \rho \in (0, 1)$$

In our case, $y = \hat{J}_\gamma(x)$. So the update equation is

$$\hat{J}_\gamma(x) \leftarrow (1 - \rho)\hat{J}_\gamma(x) + \rho\left(l(x, \gamma(x)) + \alpha \hat{J}_\gamma(x')\right)$$

While we can improve γ greedily from $\hat{J}_\gamma(x)$ that we obtain, the process is tedious. We shall look at a more straight-forward way of obtaining optimal γ using *Q-learning* next time.