CS 497 SML: Week 5 Notes (Feb. 18, 20)

Xiaolei Li (xli10@uiuc.edu)

1 Probabilistic Cycles

Recall from last week that in order to compute the cost-to-go function for deriving γ (assuming K is unknown), we needed several conditions. For the *non-deterministic* model, we required (1) no negative cycles (2) ability to escape or avoid positive cycles. For the *probabilistic* model, we had the same requirements with probability 1.

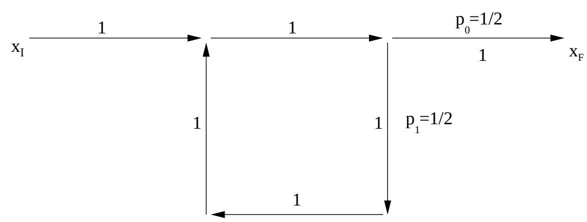

Figure 1: **Probabilistic Cycle**

Example Consider more carefully what exactly happens in a probabilistic cycle. Suppose we have the situation shown in Figure 1. If the positive cycle cannot be avoided (i.e., $p_1 = 1$), L^* would be ∞. In our model, $p_0 = \frac{1}{2}$ and thus we can avoid the cycle. It's easy to see that the final L^* calculations would be a geometric series:

$$E[L] = \frac{1}{2}(3) + \frac{1}{2^2}(7) + \frac{1}{2^3}(11) + \ldots = 3 + \sum_{i=0}^{\infty} \frac{1}{2^{i+1}}(4i) < \infty$$

In other words, the probability that one continues in the positive cycle decreases as one keeps going around. Eventually, the probability will be zero (i.e. $\lim_{i\to\infty} \frac{1}{2^{i+1}} = 0$). Therefore, the expected value of L will be smaller than ∞. ■

1.1 Termination

With a geometric series such as the one above, how does one know when to stop computing? In the series $L_F^*, L_K^*, L_{K-1}^*, \ldots$, each term will always be a little bit different from the previous term. Convergence will not occur in the sense that two consecutive terms equal each other. So, instead, we will terminate when:

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

where ϵ is a predefined *accuracy* threshold. In other words, we will stop when consecutive L_k^* terms get sufficiently close to each other. This convergence condition is guaranteed to occur if all positive cycles are of finite length.

2 Infinite-Horizon Markov Decision Processes (MDP)

In an infinite-horizon MDP, K (number of stages) is infinite and there are no termination actions. In this situation, the accumulated loss ($\sum_{i=1}^{\infty} l(x_i, u_i)$) will be ∞. This means that we will end up with an ∞-loss plan, which would be quite useless. There are two solutions to the problem and they are described below. The first one is to average the loss-per-stage and derive the limit. The second is to discount losses in the future. We will look at both of them but focus in depth on the latter.

2.1 Average Loss-Per-Stage

The intuition behind this idea is to basically limit the horizon. In this manner, we could figure out the average loss per stage and calculate the limit as $K \to \infty$. The exact equation is shown below.

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

2.2 Discount Loss

An alternative to the average loss-per-stage scheme is the concept of *discounted loss*. The intuition is that losses in the far future do not count too much. So the discounted loss scheme will gradually reduce the losses in the future to zero. This will force $\sum_{i=1}^{\infty} l(x_i, u_i, \theta_i)$ to converge. The exact definition of the discounted loss functional shown below. The α is known as the *discount factor*. A larger α gives more weight to the future.

$$L = \lim_{K\to\infty} E\left\{ \sum_{i=1}^{K-1} \alpha^{i-1} \times l(x_i, u_i, \theta_i) \right\} \qquad 0 < \alpha < 1 \tag{2}$$

With the above definition, it is clear that $\lim_{i\to\infty} \alpha^{i-1} = 0$. Thus as i approaches ∞, the term inside the summation will be 0. Therefore, the entire equation will converge. ■

3 Optimization in the Discounted Loss Model

Using the discounted loss model described in the previous section, it is now possible to optimize infinite-horizon MDPs using dynamic programming (DP). We need to find the best policy (γ) such that L is minimized (i.e., optimize Equation (2)). Before we look at dynamic programming, let us examine how L accumulates as K increases. When $K = 1$, there are no losses. As K increments, additional loss terms are attached on as shown below.

Stage	L_K^*
$K = 1$	0
$K = 2$	l_1
$K = 3$	$l_1 + \alpha l_2$
$K = 4$	$l_1 + \alpha l_2 + \alpha^2 l_3$
$\vdots$	

Figure 2: **Discounted Loss Growth**

3.1 Forward Dynamic Programming

From Figure (2), we can easily envision how forward dynamic programming can solve for L. We can set L_1^* to 0 and at each iteration after, find the best next step. In other words, search through all possible γ's and find the one that gives the least l_{i+1} where i is the current stage. As i increases, each $|L_{i+1}^* - L_i^*|$ will get smaller and smaller because $\lim_{i\to\infty} \alpha^{i-1} = 0$. And we can use a condition similar to Equation (1) that will allow the DP to stop after so many stages. This process sounds fairly easy on paper but turns out to be rather difficult in practice. Therefore, we will instead use backwards dynamic programming.

3.2 Backwards Dynamic Programming

Similar to forward dynamic programming, the backwards method will work in an iterative fashion. The main difference is that it will start at the end. What is the end for our problem? It's stage K. But in the infinite-horizon MDP, K is equal to ∞. This presents a problem in that we cannot annotate stage ∞; we will use a notational trick to get around this problem.

Recall in Figure 2 that each dynamic programming step added a term to L. In the forward DP method, we can envision this process as shown in Figure 3. In the backward DP method, we can envision the growth pattern in Figure 3 as being flipped upside down in Figure 4.

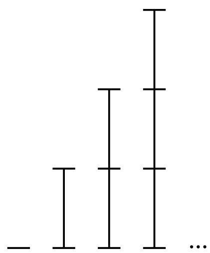

Figure 3: **FDP Growth**

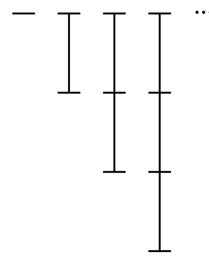

Figure 4: **BDP Growth**

An observation we could make about Figure 4 is that the bottom of the *stage list* is growing into the past. In other words, the stages in the previous step of the DP is being slid into the future. Due to discounted loss, we will need to multiple them by α because they're now further in the future. To make this process natural in terms of notation, we will define a new term J^* as below.

$$J^*_{K-k}(x_k) = \alpha^{-k} L^*_k(x_k)$$

For example, if K was equal to 5, L^*_5 will be equal to J^*_0 and L^*_1 will be equal to J^*_4. Intuitively, J^*_i is the expected loss for an i-stage optimal strategy. Recall that the original dynamic programming had the solution of:

$$\begin{aligned} L^*_K(x) &= 0 \qquad \forall x \in X \\ L^*_k(x) &= \min_{u_k \in U(x_k)} E_{\theta_k} \left\{ \alpha^k l(x_k, u_k, \theta_k) + L^*_{k+1}(f(x_k, u_k, \theta_k)) \right\} \end{aligned} \tag{3}$$

Equipped with the new J notation, we will re-write Equation (3) as the following by replacing all L's with J's.

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

We will then divide out α^k from the equation and also re-write $(K-k)$ as i. This will leave us with the following.

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

And more generally,

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

Note that now it is possible to enumerate through the backwards DP by starting at J^*_0. It would be just like solving the original BDP by starting at L^*_K. Furthermore, if we removed the *min* term in front of the equation, it will also allow us to evaluate a particular strategy:

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

It is also possible that our loss function could be independent of nature. That is $l(u, x, \theta) = l(x, u)$. We can then further simplify the last pair of equations to the following. For simplicity, we will rewrite $f(x, u, \theta)$ as x'.

$$J^*(x) = \min_{u \in U(x)} \left\{ l(x, u) + \alpha \sum_{x'} P(x'|x, u) J^*(x') \right\} \tag{4}$$

$$J_\gamma(x) = l(x, u) + \alpha \sum_{x'} P(x'|x, u) J_\gamma(x') \tag{5}$$

Notice that the loss function no longer has θ as a parameter. This allows us to remove the expectation of nature from the equation. However, since x' still depends on nature, we simply wrote out the definition of expectation as a weighted sum of all x''s. This hides θ amongst the probabilities. For a given fixed strategy, it is now possible to find J^* by iteratively evaluating Equation (5) until a condition such as Equation (1) is satisfied. This is known as *value iteration*.

3.3 Policy Iteration

The method of finding an optimal strategy, γ^*, using Equation (4) is known as *policy iteration*. The process can be summarized below.

1. Guess a strategy γ.
2. Evaluate γ using Equation (5).
3. Use Equation (4) to find an improved γ'.
4. Go back to Step 2 and repeat until no improvements occur in Step 3.

Example We shall illustrate the above algorithm through a simple example. Suppose we have $X = \{1, 2\}$ and $U = \{a, b\}$. Let Figures 5 and 6 be the probabilities of actions a and b. In addition, let the discount factor α equal $\frac{9}{10}$.

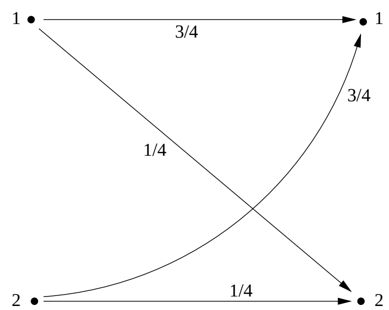

Figure 5: **Action a**

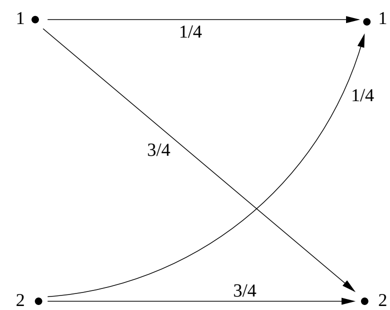

Figure 6: **Action b**

Assuming that $l(x, u, \theta) = l(x, u)$, we have the following loss values.

$$\begin{array}{ll} l(1,a) = 2 & l(1,b) = \frac{1}{2} \\ l(2,a) = 1 & l(2,b) = 3 \end{array}$$

Now, let us follow the algorithm described earlier. **Step 1** is to choose an initial γ. We will randomly choose one that is $\gamma(1) = a$ and $\gamma(2) = b$. In other words, choose action a when in state 1 and choose action b when in state 2.

Step 2 is to evaluate γ using Equation (5). This results in the following pair of equations. With them, we see that there are two unknowns with two equations and thus can be easily solved.

$$J_\gamma(1) = l(1,a) + \frac{9}{10}\left(\frac{3}{4}J_\gamma(1) + \frac{1}{4}J_\gamma(2)\right)$$

$$J_\gamma(2) = l(2,b) + \frac{9}{10}\left(\frac{1}{4}J_\gamma(1) + \frac{3}{4}J_\gamma(2)\right)$$

$$J_\gamma(1) = 24.12 \qquad J_\gamma(2) = 25.96$$

Step 3 is to minimize J_γ. With the answers above, we can now evaluate Equation (4) by putting them in place of $J^*(x')$. This will let us find a new γ which we can repeat in Step 2 (which will turn out to be $\gamma(1) = b$ and $\gamma(2) = a$). This process is relatively simple and is guaranteed to find a global minimum. However, when the number of states are large and the number of actions are large, the system of equations can become impossible to solve practically.

4 Reinforcement Learning

We can now extend the infinite-horizon MDP problem by assuming that $P(x'|x, u)$ in Equations (4) and (5) is unknown. This is essentially saying that we have no idea what the distributions of nature are. Traditionally, this hurdle is handled by the following steps.

1. Learning phase (Travel through the states in X, try various actions, and gather statistics.)
2. Planning phase (Use value iteration or policy iteration to computer J^* and γ^*.)
3. Execution phase.

In the learning phase, if the number of trials is sufficiently large, $P(x'|x, u)$ can be estimated relatively well. Also during the learning phase, we can observe the losses associated with states and actions. If we combine the three steps above and run the world through a Monte Carlo simulator, we get *reinforcement learning.* Figure 7 shows an outline of the architecture.

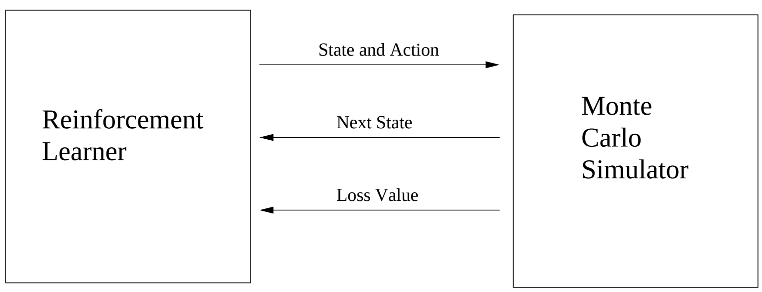

Figure 7: **Reinforcement Learning Architecture**

A major issue of reinforcement learning is the problem of *exploration vs. exploitation.* The goal of exploration is to try to gather more information about $P(x'|x, u)$, but it might end up choosing actions that yield high losses. The goal of exploitation is to make good decisions based on knowledge of $P(x'|x, u)$, but it might fail to learn a better solution. Pure exploitation is vulnerable to getting stuck to a bad solution while pure exploration requires lots of resources and might never be used.

Recall that the original evaluation of a particular strategy was:

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

But the problem now is that $P(x'|x, u)$ is unknown. Instead, we use what is called a *stochastic iterative algorithm.* $J_\gamma(x)$ will be updated with the following equation. ρ is the learning rate.

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

In this equation, x' is now observed instead of calculated from $f(x, u, \theta)$. A question a keen reader might ask is where have the probabilities gone? They're conspicuously missing in the above equation. The answer is that they're really embedded in the observations of x' from nature. In the Monte Carlo simulation, states that have high probability will occur more often and thus will make a bigger influence to $\hat{J}_\gamma$. In this manner, over time the probabilities distribution of x' will be stored in $\hat{J}_\gamma$.

Next time, we will talk about Q-Learning which will associate the J's with X *and* U and rename them Q's.