Lecture Notes: Markov Decision Processes

Marc Toussaint
Machine Learning & Robotics group, TU Berlin
Franklinstr. 28/29, FR 6-9, 10587 Berlin, Germany

April 13, 2009

1 Markov Decision Processes

1.1 Definition

A Markov Decision Process is a stochastic process on the random variables of state x_t, action a_t, and reward r_t, as given by the Dynamic Bayesian network in Figure 1. The process is defined by the conditional probabilities

$$P(x_{t+1} \,|\, a_t, x_t) \qquad \text{transition probability ,} \tag{1}$$

$$P(r_t \,|\, a_t, x_t) \qquad \text{reward probability ,} \tag{2}$$

$$P(a_t \,|\, x_t) = \pi(a_t \,|\, x_t) \qquad \text{policy .} \tag{3}$$

In the following we will assume the process to be stationary, i.e., none of these conditional probabilities explicitly depends on time.

For a given policy π and a start state x we may forward simulate the process and compute the expectation of future rewards. The *value function* $V^\pi(x)$ is such a measure, more precisely it measures the expected discounted return,

$$V^\pi(x) = \mathrm{E}\{r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \,|\, x_0 = x; \pi\} \tag{4}$$

$$= \mathrm{E}\{\sum_{t=0}^{\infty} \gamma^t r_t \,|\, x_0 = x; \pi\} \,. \tag{5}$$

Choosing the discount factor $\gamma \in [0, 1]$ smaller than 1 ensures convergence of the sum. For every start state x we may investigate what the best policy π is and what its value would be. Let us define the optimal value function as

$$V^*(x) = \max_\pi V^\pi(x) \,. \tag{6}$$

We say that a policy π^* is optimal if it maximizes the value for every start state, i.e.,

$$\pi^* \text{ optimal} \iff \forall_x : \; V^{\pi^*}(x) = V^*(x) \,. \tag{7}$$

This seems like a rather strong property of a policy, but it turns out that for every MDP there always exists at least one optimal policy. There may exist multiple optimal policies, which each gain the optimal value for every start state. Further, there also exists at least one deterministic optimal policy (there may also exist an optimal randomized policies). This will become obvious from the Bellman optimality equation that we introduce shortly.

The problem of optimal control in a given MDP is to compute an optimal policy π^* when the model $P(x' \,|\, a, x)$, the rewards $P(r \,|\, a, x)$, and the discount factor $\gamma \in \mathbb{R}$ are known.

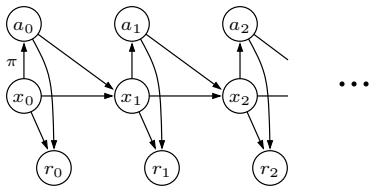

Figure 1: Markov Decision Process

1.2 Recursive properties of the value – the Bellman optimality equation

For simplicity, let us assume the policy π is deterministic, i.e., a mapping $x \mapsto a$. All of the following derivations can analogously be made for a stochastic policy by considering expectations over a.

The value function $V^\pi(x)$ is an expectation of a series of rewards which, by definition, satisfies a simple recursive property,

$$V^\pi(x) = E\{r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \mid x_0 = x; \pi\} = E\{r_0 \mid x_0 = x; \pi\} + \gamma\, E\{r_1 + \gamma r_2 + \cdots \mid x_0 = x; \pi\} \tag{8}$$

$$= R(\pi(x), x) + \gamma \sum_{x'} P(x' \mid \pi(x), x)\; E\{r_1 + \gamma r_2 + \cdots \mid x_1 = x'; \pi\} \tag{9}$$

$$= R(\pi(x), x) + \gamma \sum_{x'} P(x' \mid \pi(x), x)\; V^\pi(x') \tag{10}$$

Clearly, this recursive property also holds for the optimal value function V^*. But the optimal value function has an additional property, the *Bellman optimality equation*,

$$V^*(x) = \max_a \Big[R(a, x) + \gamma \sum_{x'} P(x' \mid a, x)\; V^*(x') \Big] \tag{11}$$

$$\pi^*(x) = \operatorname{argmax}_a \Big[R(a, x) + \gamma \sum_{x'} P(x' \mid a, x)\; V^*(x') \Big] \tag{12}$$

This is easily proved via negation: If a policy π selected an action that does not maximize the bracket term, then the policy π' which is equal to π everywhere except at state x – where it chooses the action maximizing the bracket term – would have a higher value. Thus such a π cannot be optimal. Inversely, every optimal policy must choose actions to maximize the bracket term.

The Bellman optimality equation is central throughout the theory of MDPs and reflects the *principle of optimality*. If the system is deterministic, this principle can be formulated as follows: "Form any point on an optimal trajectory, the remaining trajectory is optimal for the corresponding problem initiated at the point." This is another way of saying that an optimal policy achieves optimal value for every start state, or, whenever the process reaches a state x, the optimal policy will select the same action as an optimal policy if x was the start state of the process.

1.3 The Q-function

As alternative to the value function V one may define the Q-function,

$$Q^\pi(a, x) := \mathrm{E}\{r_0 + \gamma r_1 + \gamma^2 r_2 + \cdots \mid x_0 = x, a_0 = a; \pi\} \tag{13}$$

which is related to the value function as follows,

$$Q^\pi(a, x) = R(a, x) + \gamma \sum_{x'} P(x' \mid a, x)\; V^\pi(x')\;, \tag{14}$$

$$V^\pi(x) = Q^\pi(\pi(x), x)\;. \tag{15}$$

The Q-function holds the following recursive property and the corresponding Bellman optimality equation,

$$Q^\pi(a, x) = R(a, x) + \gamma \sum_{x'} P(x' \mid a, x)\; Q^\pi(\pi(x'), x')\;, \tag{16}$$

$$Q^*(a, x) = R(a, x) + \gamma \sum_{x'} P(x' \mid a, x) \max_{a'} Q^*(a', x')\;, \tag{17}$$

$$\pi^*(x) = \operatorname*{argmax}_a Q^*(a, x)\;. \tag{18}$$

1.4 Computing value functions

1.4.1 Value Iteration

Many algorithms which compute the optimal policy π^* are based on first computing the value function V^π or Q^π for a suboptimal policy π, or the optimal value function V^* or Q^*. Iterative algorithms to compute these functions can directly be derived from their recursive properties and the Bellman optimality equations. Let us summarize all of

them,

$$V^\pi(x) = R(\pi(x), x) + \gamma \sum_{x'} P(x' \,|\, \pi(x), x)\; V^\pi(x') \;, \tag{19}$$

$$V^*(x) = \max_a \Big[R(a, x) + \gamma \sum_{x'} P(x' \,|\, a, x)\; V^*(x') \Big] \;, \tag{20}$$

$$Q^\pi(a, x) = R(a, x) + \gamma \sum_{x'} P(x' \,|\, a, x)\; Q^\pi(\pi(x'), x') \;, \tag{21}$$

$$Q^*(a, x) = R(a, x) + \gamma \sum_{x'} P(x' \,|\, a, x) \max_{a'} Q^*(a', x') \;, \tag{22}$$

and for each of these four equations consider the respective update equations

$$\forall_x :\; V_{k+1}(x) = R(\pi(x), x) + \gamma \sum_{x'} P(x'|\pi(x), x)\; V_k(x') \tag{23}$$

$$\forall_x :\; V_{k+1}(x) = \max_a \Big[R(a, x) + \gamma \sum_{x'} P(x'|a, x)\; V_k(x') \Big] \tag{24}$$

$$\forall_{a,x} :\; Q_{k+1}(a, x) = R(a, x) + \gamma \sum_{x'} P(x'|a, x)\; Q_k(\pi(x'), x') \tag{25}$$

$$\forall_{a,x} :\; Q_{k+1}(a, x) = R(a, x) + \gamma \sum_{x'} P(x'|a, x)\; \max_{a'} Q_k(a', x') \tag{26}$$

Each of these equations describes an iterative algorithm which starts with an initial value function $V_0(x)$ and $Q_0(a, x)$ respectively (e.g., they could be initialized to constant zero), and then updates these functions while iterating $k = 1, 2, ...$ Note that the equations (19-22) are fixed point equations of the updates (23-26), respectively. Hence it seems rather intuitive that these iterative updates will converge to the desired value functions. We can prove this convergence for the case of computing Q^* via (26): Let $\Delta_k = ||Q^* - Q_k||_\infty = \max_{a,x} |Q^*(a, x) - Q_k(a, x)|$ be an error measure for the Q-function in iteration k. We have

$$\begin{aligned}
Q_{k+1}(a, x) &= R(a, x) + \gamma \sum_{x'} P(x'|a, x)\; \max_{a'} Q_k(a', x') && (27)\\
&\le R(a, x) + \gamma \sum_{x'} P(x'|a, x)\; \max_{a'} \Big[Q^*(a', x') + \Delta_k \Big] && (28)\\
&= \Big[R(a, x) + \gamma \sum_{x'} P(x'|a, x)\; \max_{a'} Q^*(a', x') \Big] + \gamma\Delta_k && (29)\\
&= Q^*(a, x) + \gamma\Delta_k && (30)
\end{aligned}$$

which describes an upper bound for the deviation of Q_{k+1} from Q^*. Similarly we can prove a lower bound of the deviation, $Q_k \ge Q^* - \Delta_k \;\Rightarrow\; Q_{k+1} \ge Q^* - \gamma\Delta_k$. Hence, the error measure $\Delta_{k+1} \le \gamma\Delta_k$ converges to zero for $\gamma \in [0, 1)$. A stopping criterion for value iteration is $\max_x |V_{k+1}(x) - V_k(x)| \le \epsilon$ and $\max_{a,x} |Q_{k+1}(a, x) - Q_k(a, x)| \le \epsilon$, respectively.

1.4.2 Direct matrix inversion

In the case of policy evaluation, where we are interested in computing the value V^π of the current policy (rather than directly V^*), there is an alternative to the iterative algorithm. If the state space is discrete and not too large we can think of $V^\pi(x)$ as a large vector $\boldsymbol{V}^\pi$ with entries $V^\pi_x \equiv V^\pi(x)$ indexed by x. We also define a large transition matrix $\boldsymbol{T}^\pi$ with entries $T^\pi_{xx'} = P(x' \,|\, \pi(x), x)$ and the reward vector $\boldsymbol{R}^\pi$ with entries $R^\pi_x = R(\pi(x), x)$. Then the recursion equation (19) can be written in vector notation,

$$\begin{aligned}
\boldsymbol{V}^\pi &= \boldsymbol{R}^\pi + \gamma \boldsymbol{T}^\pi \boldsymbol{V}^\pi && (31)\\
(\boldsymbol{I} - \gamma \boldsymbol{T}^\pi) \boldsymbol{V}^\pi &= \boldsymbol{R}^\pi && (32)\\
\boldsymbol{V}^\pi &= (\boldsymbol{I} - \gamma \boldsymbol{T}^\pi)^{-1} \boldsymbol{R}^\pi \;. && (33)
\end{aligned}$$

This is an exact solution which requires the inversion of an $n \times n$ matrix (complexity $O(n^3)$) when n is the cardinality of the state space.

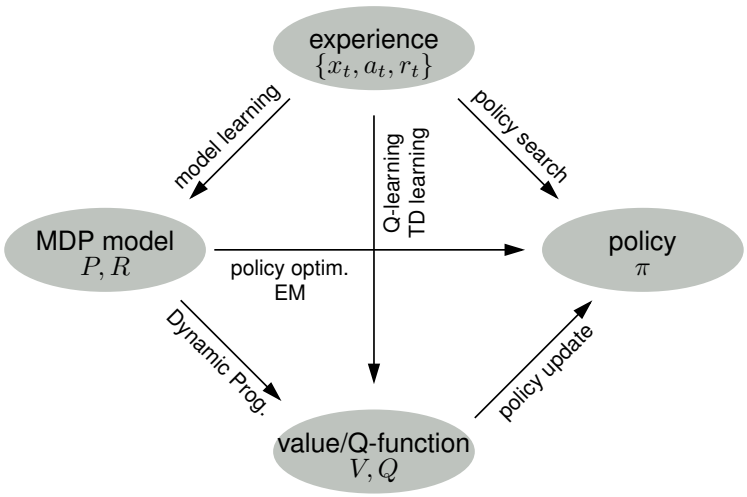

Figure 2: Model-based & model-free RL, direct policy search

1.5 Policy Iteration

When we have computed the optimal value function $V^*(x)$ or $Q^*(a,x)$ the optimal policy can directly be derived using equations (12) or (18).

If we have only computed the value function $V^\pi(x)$ or $Q^\pi(a,x)$ for the current policy (policy evaluation), we can apply the *policy iteration* algorithm which alternates between evaluating and improving the policy:

1. *initialize:* $k=0$ and π_0 randomly
2. *policy evaluation:* compute V^{π_k} or Q^{π_k}
3. *policy update:* $\pi_{k+1}(x) = \text{argmax}_a Q^{\pi_k}(a,x)$
4. $k \leftarrow k+1$ and iterate from 2.

1.6 Learning from experience

While control theory usually assumes knowledge of the model $P(x' \,|\, a, x)$ and focuses on the problem of finding an optimal control policy π, the field of Reinforcement Learning typically focuses on the problem of learning from experience when the model $P(x' \,|\, a, x)$ is not a priori known. There are three common approaches to learn from experience (see also figure 2),

1. model-based RL: use the experience to learn a model $P(x' \,|\, a, x)$ and $P(r \,|\, a, x)$, and then use this model to derive a policy,
2. model-free RL: use the experience to directly learn a value function (via temporal difference, Q-learning, see below),
3. policy search: use the experience to evaluate different policies and directly search in the space of policies; when it is possible to compute a gradient one can also perform a gradient ascent in the space of policies.

Model-based learning. When the model is represented in the direct representation, i.e., $P(x' \,|\, a, x)$ is represented as full look-up table, learning the model is trivial. We simply count experienced transitions and use these counts as an estimator $\hat{P}$ for the transition probabilities,

$$\hat{P}(x'|a,x) \propto \#(x' \leftarrow x|a)$$

where $\#(x' \leftarrow x|a)$ means how often have we experienced a transition from x to x' when executing action a in state x. Using the estimator $\hat{P}$ we can use standard value iteration to derive an optimal policy w.r.t. the estimated model.

Q-learning. Recall the Bellman optimality equation and value iteration for the Q-function,

$$Q^*(a,x) = R(a,x) + \gamma \sum_{x'} P(x' \,|\, a,x) \max_{a'} Q^*(a',x') \;, \tag{34}$$

$$\forall_{a,x}:\; Q_{k+1}(a,x) = R(a,x) + \gamma \sum_{x'} P(x'|a,x) \;\max_{a'} Q_k(a',x') \;. \tag{35}$$

When we try to learn Q^* directly from experience we cannot apply the second equation since P is unknown. But we can use the experience to approximate the second equation. Experience is given in terms of a state-action-reward sequence $x_0 a_0 r_0 \; x_1 a_1 r_1 \; x_2 a_2 r_2 \; x_3 a_3 r_3 \; \cdots$. Q-learning (Watkins, 1988) does the following update of the Q-function in every step,

$$Q_{\text{new}}(x_t,a_t) = (1-\alpha) Q_{\text{old}}(x_t,a_t) + \alpha[r_t + \gamma \max_a Q_{\text{old}}(x_{t+1},a)] \;. \tag{36}$$

Note that the bracket term is a sample of the RHS of equation (35). That is, we only update the state-action pairs that are actually visited and use a learning rate $\alpha \in [0,1]$. In effect, for pairs with positive feedback Q is increased, while for pairs with negative feedback Q is decreased. This refers to the classical paradigm of *reinforcement*.

Since Q-learning is a stochastic variant of the exact value iteration (35) Q-learning converges with probability 1. More precisely, when writing equation (35) as a deterministic process $Q_{k+1} = T(Q_k)$, then Q-learning is a stochastic variant $Q_{k+1} = (1-\alpha)Q_k + \alpha[T(Q_k) + \eta_k]$ with a random variable η_k. One can show that η_k has zero mean. Thus, Q-learning was the first provably convergent model-free adaptive optimal control algorithm.

Temporal Difference (TD) learning is the analogous algorithm based on updating the V-function instead of the Q-function,

$$V_{\text{new}}(x_t) = V_{\text{old}}(x_t) + \alpha[r_t + \gamma V_{\text{old}}(x_{t+1}) - V_{\text{old}}(x_t)] \;. \tag{37}$$

Eligibility traces. Consider a state-reward sequence $x_0 r_0 \; x_1 r_1 \; x_2 r_2 \; x_3 r_3 \; x_4 r_4 \; x_5 r_5 \cdots$. At time t, the TD equation (37) takes the received reward r_t and updates the value function at location x_t only. For instance, if the reward r_4 was remarkably high, i.e., much higher than predicted by the temporal difference $\gamma V(x_5) - V(x_4)$, then the value of $V(x_4)$ will be significantly increased by the update (37). But perhaps one should also increase the value of $V(x_3)$ because in the previous time step $V(x_3)$ was updated by the difference $[r_3 + \gamma V(x_4) - V(x_3)]$ and we now know that $V(x_4)$ was underestimated. Arguing recursively, all the values of the previously visited states should also be increased because $V(x_4)$ has significantly been underestimated. Eligibility traces do exactly this in a proper way: Ideally, when we have observed a sequence of rewards $r_{t\text{-}2} r_{t\text{-}1} r_t$ we should update the value of $V(x_{t\text{-}2})$ according to

$$V_{\text{new}}(x_{t\text{-}2}) = V_{\text{old}}(x_{t\text{-}2}) + \alpha[r_{t\text{-}2} + \gamma r_{t\text{-}1} + \gamma^2 r_t + \gamma^3 V_{\text{old}}(x_{t+1}) - V_{\text{old}}(x_{t\text{-}2})] \;. \tag{38}$$

Such updates should be made for all states that have been visited in the history, accounting for the whole history of rewards. This can be done in an online implementation by keeping track of which states have been visited in terms of *eligibility traces*: Whenever we visit a state x_t we increase its eligibility by 1,

$$e(x_t) \leftarrow e(x_t) + 1 \;. \tag{39}$$

When we observe the reward r_t we update all eligible states

$$\forall_x:\; V_{\text{new}}(x) = V_{\text{old}}(x) + \alpha e(x)[r_t + \gamma V_{\text{old}}(x_{t+1}) - V_{\text{old}}(x_t)] \;, \tag{40}$$

then we decay all eligibilities according to the discounting γ and an additional forgetting rate $\lambda \in [0,1]$,

$$\forall_x:\; e(x) \leftarrow \gamma\lambda e(x) \;. \tag{41}$$

This algorithm is called TD(λ). Eligibilities speed up convergence significantly.