Reinforcement Learning Basics - Notes

John Martin - jmarti3@stevens.edu

May 20, 2016

Markov Reward Process

Markov Reward Processes describe the open-loop dynamics of a stochastic process

Definition 1. A Markov Reward Process is a tuple $\langle \mathcal{X}, \mathcal{P}, \mathcal{R}, \gamma \rangle$.

- $\mathcal{X}$: finite set of states
- $\mathcal{P}$: transition probability matrix $\mathcal{P}_{xx'} = \mathbb{P}\big[x_{k+1} = x' \big| x_k = x\big]$
- $\mathcal{R}$: reward function $\mathcal{R}_x = \mathbb{E}\big[g_k \big| x_k = x\big]$
- γ: discount factor $\gamma \in [0, 1)$

The total discounted reward, or cost-to-go, allows us to compare trajectories with respect to an objective function.

Definition 2. The total discounted reward is the value of an entire sequence

$$J_k = g_{k+1} + \gamma g_{k+2} + \cdots = \sum_{j=0}^{\infty} \gamma^j g_{j+k+1}$$

The value function is the central quantity in Reinforcement Learning.

Definition 3. The state value function is the expected return starting in state x:

$$v(x) = \mathbb{E}\big[J_k \big| x_k = x\big]$$

This quantity summarizes how good it is to be in a state.

Bellman's equation shows that value functions obey a recursive decomposition:

$$\begin{aligned} v(x) &= \mathbb{E}\big[J_k \big| x_k = x\big], \\ &= \mathbb{E}\big[g_{k+1} + \gamma g_{k+2} + \gamma^2 g_{k+3} + \cdots \big| x_k = x\big], \\ &= \mathbb{E}\big[g_{k+1} + \gamma(g_{k+2} + \gamma g_{k+3} + \cdots) \big| x_k = x\big], \\ &= \mathbb{E}\big[g_{k+1} + \gamma J_{k+1} \big| x_k = x\big], \\ &= \mathbb{E}\big[g_{k+1} + \gamma v(x_{k+1}) \big| x_k = x\big], \end{aligned}$$

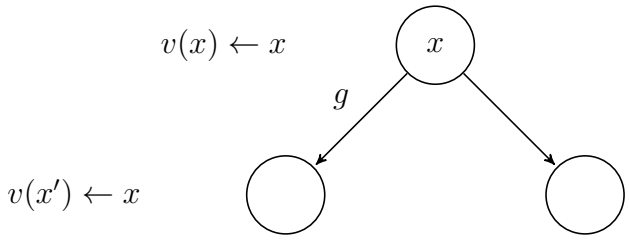

Figure 1: One-step lookahead tree.

We can think of the Bellman equation as a one-step lookahead tree.
Mark

Markov Decision Process

Markov Decision Processes describe the stochastic dynamics of a system under deliberate action:

Definition 4. A Markov Decision Process is a tuple $\langle \mathcal{X}, \mathcal{U}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where

- $\mathcal{X}$: finite set of states
- $\mathcal{U}$: finite set of actions, or control inputs
- $\mathcal{P}$: transition probability matrix $\mathcal{P}^u_{xx'} = \mathbb{P}\big[x_{k+1} = x' \big| x_k = x, u_k = u\big]$
- $\mathcal{R}$: reward function $\mathcal{R}^u_x = \mathbb{E}\big[g_k \big| x_k = x, u_k = u\big]$
- γ: discount factor $\gamma \in [0, 1)$

Policies are mappings from actions to states

Definition 5. A policy is a distribution over states

$$\pi(u|x) = \mathbb{P}\big[u_k = u \big| x_k = x\big]$$

With these definitions we can adjust the ideas of value functions

Definition 6. The state value function under a Markov Decision Process is the expected return starting from state x and following policy π thereafter

$$v_\pi(x) = \mathbb{E}_\pi\big[J_k \big| x_k = x\big].$$

Definition 7. The action value function under a Markov Decision Process is the expected return when starting in state x, applying the input u and following the policy π thereafter

$$q_\pi(x, u) = \mathbb{E}_\pi\big[J_k \big| x_k = x, u_k = u\big].$$

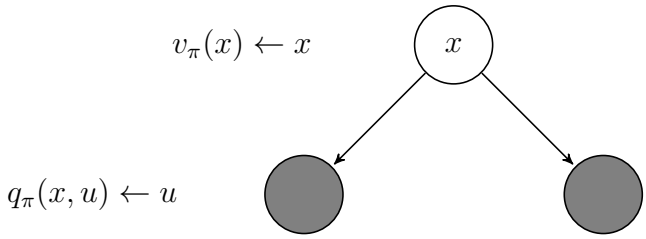

Figure 2: Relating v_π to q_π.

How does the state value functions relate to the action value function?

$$v_\pi(x) = \sum_{u \in \mathcal{U}} \pi(u|x) q_\pi(x,u).$$

How does the action value function relate to the state value function?

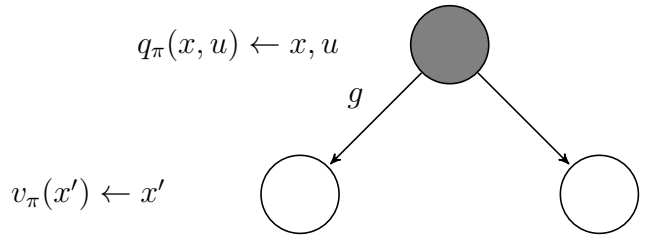

Figure 3: Relating v_π to q_π.

$$q_\pi(x,u) = g(x,u) + \gamma \sum_{x \in \mathcal{X}} \mathcal{P}^u_{xx'} v_\pi(x).$$

If we stitch these results together we find

$$v_\pi(x) = \sum_{u \in \mathcal{U}} \pi(u|x) \left(g(x,u) + \gamma \sum_{x \in \mathcal{X}} \mathcal{P}^u_{xx'} v_\pi(x) \right).$$

This, in essence, defines the Bellman expectation equation. This is useful. However, we care about the optimal value. The optimal value functions are defined as

$$v^*_\pi(x) = \max_\pi v_\pi(x),$$
$$q^*_\pi(x,u) = \max_\pi q_\pi(x,u).$$

Using the same steps as before we arrive at the Bellman optimality equation. In any particular state we have

$$v^*_\pi(x) = \max_u q_\pi(x,u),$$
$$q^*_\pi(x,u) = g(x,u) + \gamma \sum_{x \in \mathcal{X}} \mathcal{P}^u_{xx'} v^*_\pi(x')$$

Use one in the other to get

$$q^*_\pi(x,u) = g(x,u) + \gamma \sum_{x \in \mathcal{X}} \mathcal{P}^u_{xx'} \max_u q_\pi(x,u)$$

Once we have the optimal action value function then we have solved the MDP. We know how to behave optimally for any state we may be in.

Policy Evaluation

Dynamic Programming: Use the Bellman equation to backup every state according to the dynamics and reward structure. Starting with arbitrary values for initial value function apply the update

$$v(x_k) \leftarrow \max_{u \in \mathcal{U}} g(x_k, u_k) + \gamma \sum_{x' \in \mathcal{X}} \mathcal{P}^u_{xx'} v^*(x').$$

Monte-Carlo Evaluation: Execute some trajectory and average the return received at the end. This gives us our estimate $v(x_k)$. If we execute the trajectory multiple times then can update our estimate of $v(x_k)$ as follows

$$v(x_k) \leftarrow v(x_k) + \alpha(J_k - v(x_k)).$$

TD Evaluation: Instead of waiting until the end of a trajectory to update our value function, we will observe one return then estimate the remaining value we expect to receive from then onward. This concept is referred to as bootstrapping. Updates in this setting are given by

$$v(x_k) \leftarrow v(x_k) + \alpha(g(x_k, u_k) + \gamma v(x_{k+1}) - v(x_k)).$$

The quantities of interest are

- TD target: $g(x_k, u_k) + \gamma v(x_{k+1})$. This is the estimated value of starting in x_k knowing that we receive the reward $g(x_k, u_k)$.
- TD error: $g(x_k, u_k) + \gamma v(x_{k+1}) - v(x_k)$. The difference between the estimate before and after receiving the reward.

Policy Iteration

This framework alternates between two steps: policy evaluation and policy improvement. Using the state value function policies are improved by acting greedily with respect to the value function.

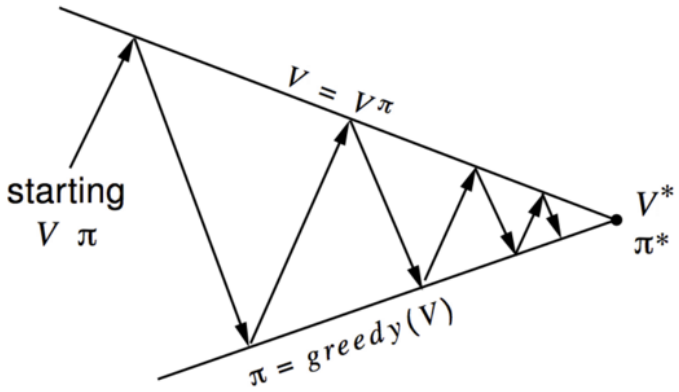

Figure 4: Policy iteration cycle. (Figure from David Silver's lectures)

Actor-critic Algorithms

FAQs

What makes RL different from supervised learning?
The difference is in the data. Supervised learning algorithms use a labeled data set to which their hypothesis is fit. Reinforcement learning algorithms use unlabeled data that is sampled online. Most Supervised learning algorithms also assume that samples derive independently and identically from some distribution. Whereas the online execution of reinforcement learning algorithms usually cannot make this guarantee.

What makes RL different from classical dynamic programming?
Classical DP algorithms perform full backups; RL algorithms perform partial backups. Allow me to elaborate.

Classical dynamic programming solutions are said to solve all overlapping subproblems. Meaning that if we know the optimal path 10m away from the goal, then it can be used to find the optimal path 11m away from the goal. Starting at 11m away, we simply find the optimal 1m path that takes us to the start of the optimal 10m path. Finding the optimal 1m path is, in essence, a backup operation. In the classical case, backups are applied to all states and for all possible actions of the problem. In contrast to RL, only one backup per state is performed. And this backup is typically chosen by sampling.

If RL is a solution method for MDPs then how can it be applied to continuous spaces?
Classical reinforcement learning extends to continuous spaces through function approximation. The value function and the policy can both be expanded as a with a set of basis vectors: think polynomial, gaussian functions, neural nets, etc. When new

information arrives, updates are applied to the parameters that characterize the approximations.

How can RL algorithms find solutions without models?
They use the action value function as a basis for policy improvement. Choosing an action which maximizes the value in a given state turns out to be the same as finding a successor state that maximizes the value. The difference is that one approach only requires knowledge of the available actions, and the other requires a model to predict the successor state.

References