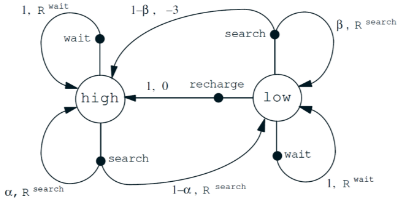


ECE 414/517 Reinforcement Learning

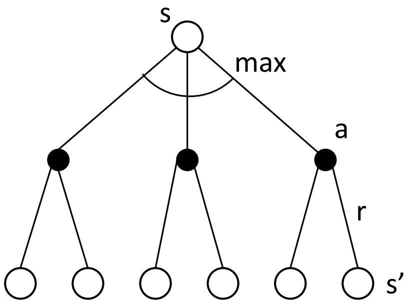
LECTURE 6: BELLMAN OPTIMALITY EQUATIONS AND DYNAMIC PROGRAMMING

SEP. 15 2022

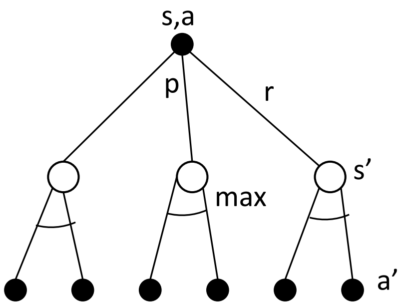
(ADAPTED FROM SLIDES BY DR. ITAMAR AREL)

Bellman Optimality Equation

1. Since $V^*(s)$ is the value function for a policy, it must satisfy the Bellman Equation
2. When using the optimal policy we call the equation the **Bellman Optimality Equation**.
3. Intuitively, the Bellman Optimality Equation expresses the fact that the value of a state under an optimal policy must equal the expected return for the best action from that state:



$$V^*(s) = \max_a E[r_{t+1} + \gamma V^*(s_{t+1}) | S_t = s, A_t = a]$$
$$= \max_a \sum_{s',r} p(s', r | s, a)[r + \gamma V^*(s')]$$



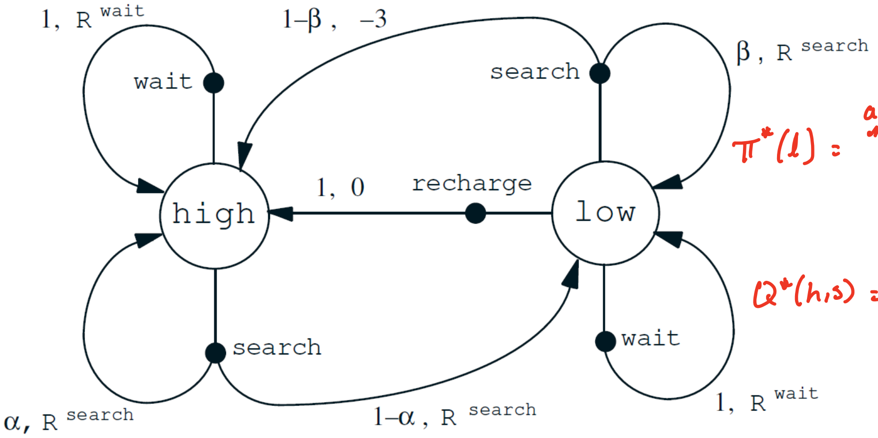
$$Q^*(s,a) = E[r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') | S_t = s, A_t = a]$$
$$= \sum_{s',r} p(s', r | s, a)\left[r + \gamma \max_{a'} Q^*(s', a')\right]$$

Bellman Optimality Equations Example

$$V^*(s) = \max_a E[r_{t+1} + \gamma V^*(s_{t+1}) | S_t = s, A_t = a]$$

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

h: high
l: low
s: search
w: wait
re: recharge



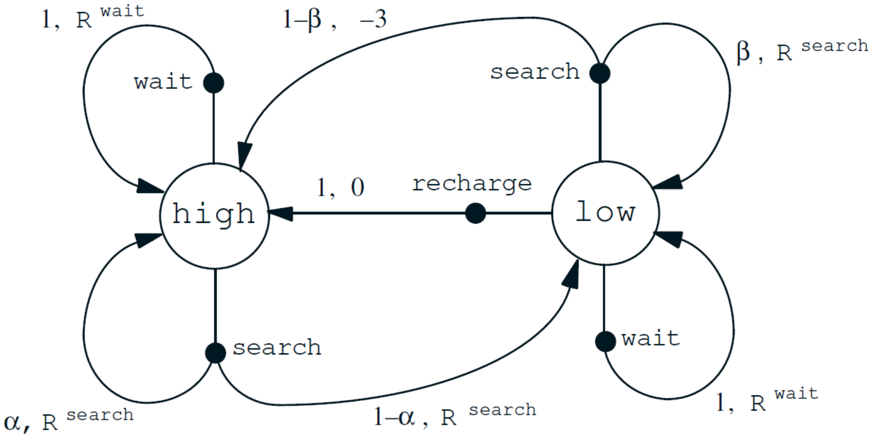
$$\pi^*(l) = \underset{a}{\arg\max} \begin{bmatrix} \beta(R^{search} + \gamma V^*(l)) + (1-\beta)(-3 + \gamma V^*(h)) \\ R^{wait} + \gamma V^*(l) \\ 0 + \gamma V^*(h) \end{bmatrix}$$

$$Q^*(h,s) = \alpha\left(R^{search} + \gamma \max_a \begin{bmatrix} Q^*(h,s) \\ Q^*(h,l) \end{bmatrix}\right) + (1-\alpha)\left(R^{search} + \gamma \max_a \begin{bmatrix} Q^*(l,s) \\ Q^*(l,w) \\ Q^*(l,R) \end{bmatrix}\right)$$

Bellman Optimality Equations Example

$$Q^*(s,a) = E[r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1},a') | S_t = s, A_t = a]$$
$$= \sum_{s',r} p(s',r|s,a)\left[r + \gamma \max_{a'} Q^*(s',a')\right]$$

h: high
l: low
s: search
w: wait
re: recharge



Solving the Bellman Optimality Equation

1. Finding an optimal policy by solving the Bellman Optimality Equation (for example with dynamic programming) requires the following:
   - Accurate knowledge of the environment dynamics (environment model)
   - Enough space and time to do the computation
   - The Markov property
2. How much space and time do we need:
   - Polynomial in the number of states (will discuss in next chapter)
   - However, number of states can be extremely large (e.g. backgammon has about 10^{20} states)
3. We usually have to settle for approximations
4. Many RL methods can be understood as approximately solving the Bellman Optimality Equation.
   - Advantages of approximation?
     - learn more effectively
     - feature extraction can reduce noise
     - can address large scale problems

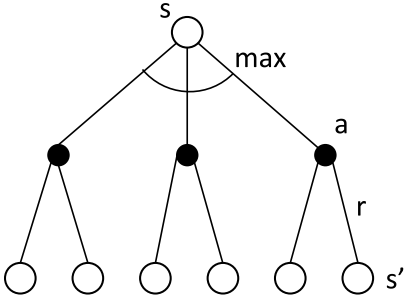
Solving the Bellman Optimality Equation

1. During the next few weeks we'll talk about techniques for solving the RL problem:
   - Dynamic Programming – well developed mathematically, solves the Bellman equations, but requires an accurate model of the environment.
   - Monte Carlo Methods – Do not require an exact model, but work only on episodic tasks since returns only calculated at the end.
   - Temporal Difference Learning – Do not require an exact model and can work on non episodic tasks.
     - More complex to analyze
     - Launched the revisiting of RL as a pragmatic framework (1988)
2. All methods differ in efficiency and speed of convergence to the optimal solution.

Review

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

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} \sum_{s',r} p(s',r|s,a)[r + \gamma V^*(s')]$$



1. Multi Armed Bandits ($\gamma = 0.9$)
   a) States? 1 State
   b) Actions? pull lever - 4 actions [0,1,2,3]
   c) Rewards? Rewards from machine




Bernoulli 1.4 3.6 0.1 2.7

what should V^* be?

$$V^*(s) = 3.6 + 0.9 \cdot 3.6 + 0.9^2 \cdot 3.6 + \dots$$

$$= \frac{3.6}{1-0.9} = 36$$

$$V^*(s) = \max_a \begin{bmatrix} 1.4 + (0.9) \cdot V^*(s) \\ 3.6 + (0.9) \cdot V^*(s) \\ 6.1 + (0.9) \cdot V^*(s) \\ 2.7 + (0.9) \cdot V^*(s) \end{bmatrix}$$

$$V^*(s) = 3.6 + 0.9 \cdot V^*(s)$$

$$0.1 V^*(s) = 3.6$$

$$V^*(s) = 36$$

$$\pi^*(s) = \arg\max_a \begin{bmatrix} a & 0 \\ 36 & 1 \\ b & 2 \\ c & 3 \end{bmatrix}$$

Summary

1. Agent environment interaction
   - states
   - actions
   - rewards
2. Policy:
   - $\pi(a|s)$ good for stochastic policy

     $\pi(a|s) = \pi(a|s) = p$ 0SP61

     $\pi(s) = a^* \; (\pi(s, a^*) = 1)$
   - probability of selecting action in state
3. Return: (weighted) sum of future rewards
   - episodic vs. continuing tasks
4. Markov property
5. Markov Decision Process:
   - Transition probability
   - Expected reward
6. Value Functions
   - State value function for a policy
   - Action value function for a policy
7. Bellman Equations
8. Optimal value functions
9. Optimal Policies
10. The need for approximation

Dynamic Programming

Dynamic programming is the collection of algorithms that can be used to compute optimal policies give a perfect model of the environment.

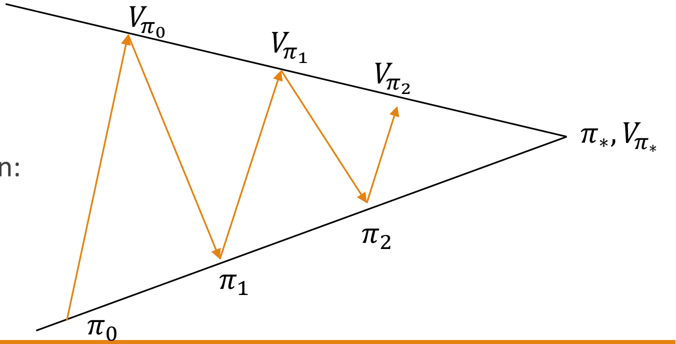
1. DP constitutes a theoretically optimal methodology
2. In reality it is often limited since DP is computationally expensive.
3. As related to other methods:
   - Hope to do “just as well” as DP
   - Hopefully can do it with less computation
   - Hopefully can do it with less memory
   - Hopefully without model
4. Key Idea (In general): Use value function to derive optimal policy
5. Need to find a way to calculate optimal state value function

Policy Iteration

1. Technique for obtaining the optimal policy
2. Comprises of two complementing steps:
   - Policy Evaluation: updating the value function in view of current policy
   - Policy Improvement: updating the policy given the current value function (which can be sub-optimal)

$$\pi_0 \xrightarrow{E} V_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi_* \xrightarrow{E} V_{\pi_*}$$

3. The process converges by bouncing between these two steps
   - Finding a greedy policy for V_{π_i} makes the state value function incorrect
   - Finding a value function for a given policy makes the policy not greedy anymore
   - Yet together they drive each other to the optimal solution: π_*, V_{π_*}



Calculating Value Function

We will simply convert the Bellman Equations into update rules for improving our approximation:

Jacobi Method
$x = ay + bz$
$y = cx + dz$
$z = ex + fy$

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

$$\Downarrow$$

$$V^{\pi}_{k+1}(s) = \sum_{a} \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma V^{\pi}_{k}(s')]$$

Iterative Policy Evaluation

1. Choose an arbitrary V_0 (except terminal states)
2. Perform a series of sweeps where at each step you use the Bellman Equation:

$$V_{k+1}^{\pi}(s) = \sum_{a} \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma V_k^{\pi}(s')]$$

3. Sweeps:

$$V_0 \to V_1 \to V_2 \to \cdots \to V_k \to V_{k+1} \to \cdots \to V^{\pi}$$

4. Can be proven to converge to V^{π} as $k \to \infty$
5. In each iteration all states are updated
   - scheme can be computationally heavy

$$V_1^{\pi_0}(0,0) = \frac{1}{4}(-1 + 0.9 \cdot 0) + \frac{1}{4}(-1 + 0.9 \cdot 0)$$

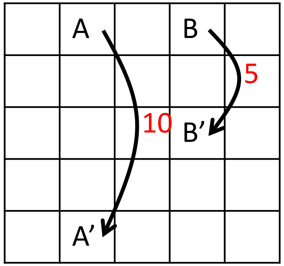
$$+ \frac{1}{4}(0 + 0.9 \cdot 0) + \frac{1}{4}(0 + 0.9 \cdot 0)$$

$$V_1^{\pi_0}(0,1) = 10 + 0.9 \cdot 0 = 10$$

$$V_2^{\pi_0}(0,0) = \frac{1}{4}(-1 + 0.9 \cdot -0.5) + \frac{1}{4}(-1 + 0.9 \cdot -0.5)$$

$$\frac{1}{4}(0 + 0.9 \cdot 10) + \frac{1}{4}(0 + 0.9 \cdot -.75)$$

Example: Grid World



$$\pi(s) = \{0.25, 0.25, 0.25, 0.25\} \ \forall s \in S$$
$$\gamma = 0.9$$

$$V_{k+1}^{\pi}(s) = \sum_{a} \pi(a|s) \sum_{s',r} p(s', r | s, a)[r + \gamma V_k^{\pi}(s')]$$

3.31	8.79	4.43	5.32	1.49
1.52	2.99	2.25	1.91	0.55
0.05	0.74	0.67	0.36	-0.40
-0.97	-0.44	-0.35	-0.59	-1.18
-1.86	-1.34	-1.23	-1.42	-1.97

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

V_0

-0.50	10.0	-0.25	5.00	-0.5
-0.25	0.00	0.00	0.00	-0.25
-0.25	0.00	0.00	0.00	-0.25
-0.25	0.00	0.00	0.00	-0.25
-0.50	-0.25	-0.25	-0.25	-0.5

V_1

1.47	9.78	3.07	5.00	0.34
-0.48	2.19	-0.06	1.07	-0.48
-0.42	-0.06	0.00	-0.06	-0.42
-0.48	-0.11	-0.06	-0.11	-0.48
-0.84	-0.48	-0.42	-0.48	-0.84

V_2

2.25	9.57	3.75	4.95	0.67
0.37	2.07	1.42	0.99	-0.13
-0.57	0.37	-0.05	0.12	-0.57
-0.66	-0.24	-0.14	-0.24	-0.66
-1.09	-0.66	-0.57	-0.66	-1.09

V_3

Iterative Policy Evaluation

1. Choose an arbitrary V_0

2. Perform a series of sweeps where at each step you use the Bellman Equation:

$$V_{k+1}^{\pi}(s) = \sum_{a} \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma V_k^{\pi}(s')]$$

3. Sweeps:

$$V_0 \to V_1 \to V_2 \to \cdots \to V_k \to V_{k+1} \to \cdots \to V^{\pi}$$

4. Can be proven to converge to V^{π} as $k \to \infty$

5. In each iteration all states are updated
   - scheme can be computationally heavy

6. When to stop?
   - Since we cannot do infinite iterations, do until largest update is smaller then some threshold

$$\max_{s \in S} |V_{k+1}(s) - V_k(s)|$$

In place schemes

1. Update all values in single array, instead of in new one
2. Usually converges quicker since it using newer values earlier
3. Can depend on the order in which the update happens.

Out Place Converges in 112

V_0

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

V_1

-0.50	10.0	-0.25	5.00	-0.5
-0.25	0.00	0.00	0.00	-0.25
-0.25	0.00	0.00	0.00	-0.25
-0.25	0.00	0.00	0.00	-0.25
-0.50	-0.25	-0.25	-0.25	-0.5

V_2

1.47	9.78	3.07	5.00	0.34
-0.48	2.19	-0.06	1.07	-0.48
-0.42	-0.06	0.00	-0.06	-0.42
-0.48	-0.11	-0.06	-0.11	-0.48
-0.84	-0.48	-0.42	-0.48	-0.84

V_3

2.25	9.57	3.75	4.95	0.67
0.37	2.07	1.42	0.99	-0.13
-0.57	0.37	-0.05	0.12	-0.57
-0.66	-0.24	-0.14	-0.24	-0.66
-1.09	-0.66	-0.57	-0.66	-1.09

In Place Converges in 81

V_0

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

V_1

-0.50	10.0	2.00	5.00	0.63
-0.36	2.17	0.94	1.34	0.19
-0.33	0.41	0.30	0.37	-0.12
-0.32	0.02	0.07	0.10	-0.26
-0.57	-0.37	-0.32	-0.30	-0.62

V_2

1.44	9.66	3.71	5.33	1.02
0.41	2.57	1.78	1.73	0.38
-0.21	0.60	0.64	0.53	-0.13
-0.50	-0.04	0.08	0.01	-0.47
-0.95	-0.63	-0.51	-0.57	-1.02

V_3

2.42	9.43	4.31	5.47	1.28
0.92	2.86	2.15	1.92	0.53
-0.07	0.76	0.79	0.58	-0.14
-0.60	-0.09	0.05	-0.09	-0.64
-1.21	-0.80	-0.66	-0.78	-1.28

Policy Improvement

1. Policy evaluation deals with finding the value function under a given policy
2. However, how do we find the optimal policy? (and therefore the optimal value function)
   - Policy Improvement
3. Suppose that for some arbitrary policy π we've computed the value function (using policy evaluation)
4. Let policy π' be a deterministic policy defined such that in each state s it selects action a that maximizes the first-step value, i.e.:

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

   - Note the difference between $\pi(s,a)$ and $\pi(s)$
5. It can be shown that $\pi'$is at least as good as π, and if they are equal they are both the optimal policy.

Policy Improvement

1. What is the value if we first choose action a which is not necessarily $\pi(s)$?
   - The value of behaving this way:

$$Q^{\pi}(s,a) = \sum_{s',r} p(s',r|s,a)[r + \alpha V^{\pi}(s')]$$

2. If this is higher than $V^{\pi}(s)$, then it should be better to select action a and then follow π
3. Shouldn't that mean that it is always better to select action a at s?
   - Should we change the policy?
4. Yes – Policy improvement theorem.
   - Given two deterministic policies π and π':

$$\begin{aligned}
&if: \\
&Q^{\pi}(s,\pi'(s)) \geq \mathrm{V}^{\pi}(s) \quad \forall \;\; s \in S \\
&Then: \\
&\mathrm{V}^{\pi'}(s) \geq \mathrm{V}^{\pi}(s)
\end{aligned}$$

Policy Improvement

$$
\begin{aligned}
v_\pi(s) &\le q_\pi(s, \pi'(s)) \\
&= \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = \pi'(s)] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\
&\le \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1})) \mid S_t = s] \\
&= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}_{\pi'}[R_{t+2} + \gamma v_\pi(S_{t+2}) | S_{t+1}] \mid S_t = s] \\
&= \mathbb{E}_{\pi'}\big[R_{t+1} + \gamma R_{t+2} + \gamma^2 v_\pi(S_{t+2}) \bigm| S_t = s\big] \\
&\le \mathbb{E}_{\pi'}\big[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 v_\pi(S_{t+3}) \bigm| S_t = s\big] \\
&\ \ \vdots \\
&\le \mathbb{E}_{\pi'}\big[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \bigm| S_t = s\big] \\
&= v_{\pi'}(s).
\end{aligned}
$$

Policy Iteration

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
       $\Delta \leftarrow 0$
       Loop for each $s \in \mathcal{S}$:
           $v \leftarrow V(s)$
           $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s))\big[r + \gamma V(s')\big]$
           $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
       old-action $\leftarrow \pi(s)$
       $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
       If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2