Artificial Intelligence
DSE 3252

Reinforcement Learning

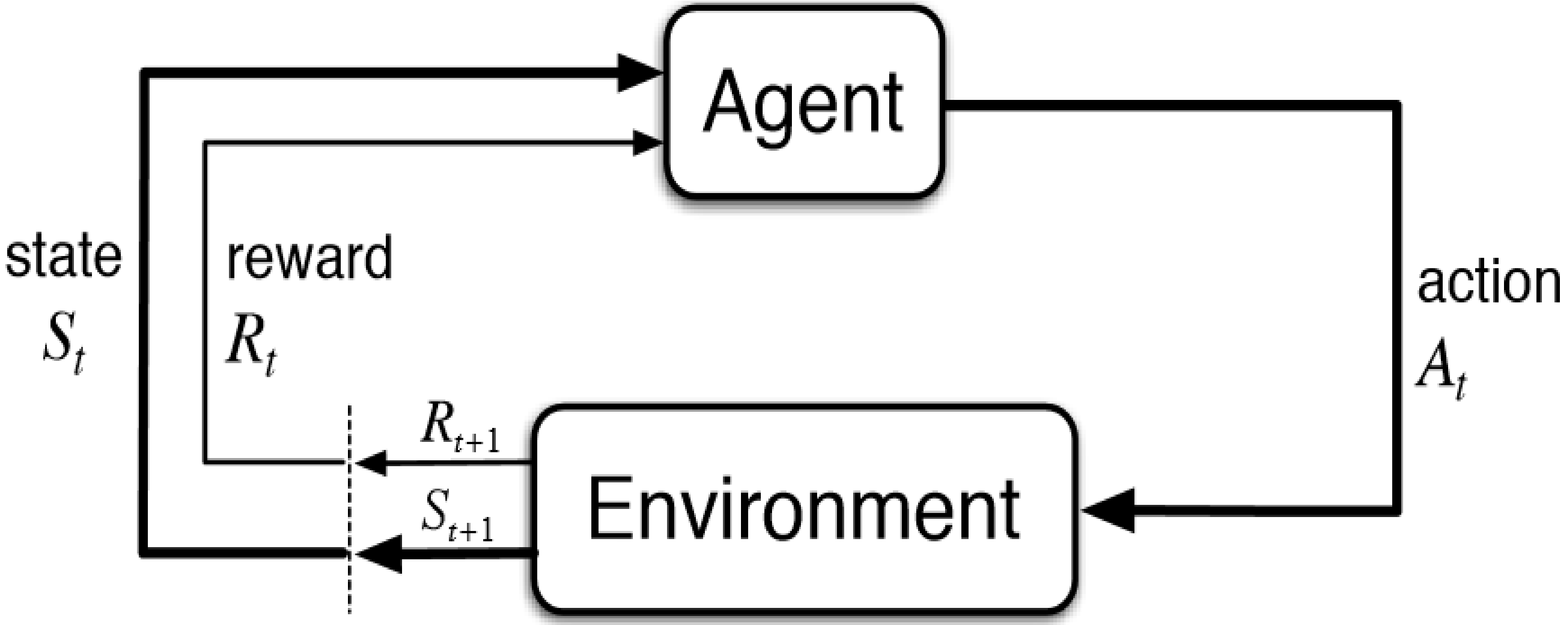
DR.ROHINI R RAO & DR.RASHMI LAXMIKANT MALGHAN
DEPT OF DATA SCIENCE & COMPUTER APPLICATIONS

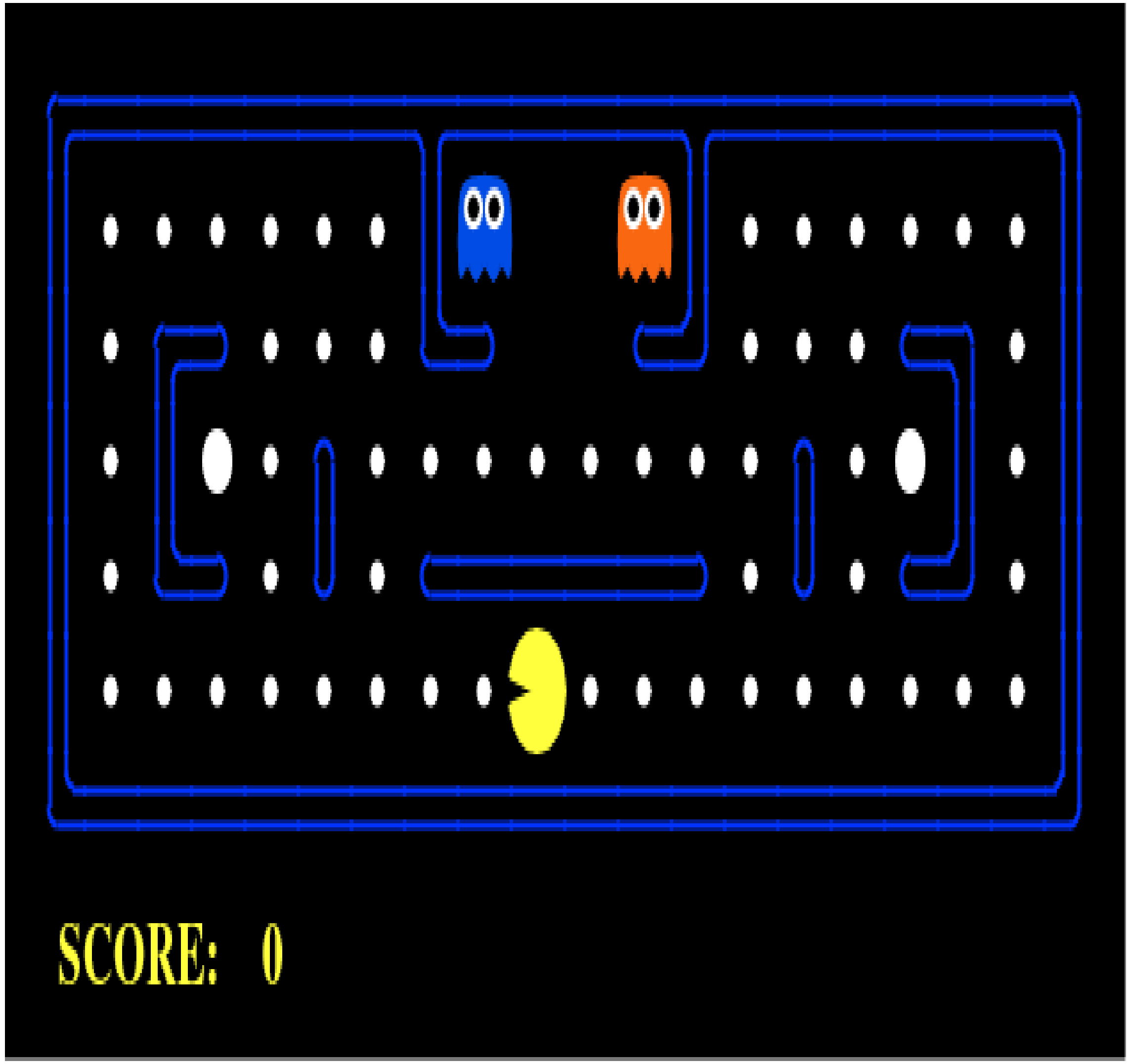
JANUARY 2024

What is Reinforcement Learning

- Reinforcement Learning is a
  - feedback-based Machine learning technique in which an agent learns to behave in an environment by performing the actions and seeing the results of actions.
  - For each good action, the agent gets positive feedback
  - for each bad action, the agent gets negative feedback or penalty.
- ***"Reinforcement learning is a type of machine learning method where an intelligent agent (computer program) interacts with the environment and learns to act within that."***

Reinforcement Learning





Problem Formulation in RL

Environment

Physical world in which the agent operates

State

- Current situation of the agent

Reward

- Feedback from the environment

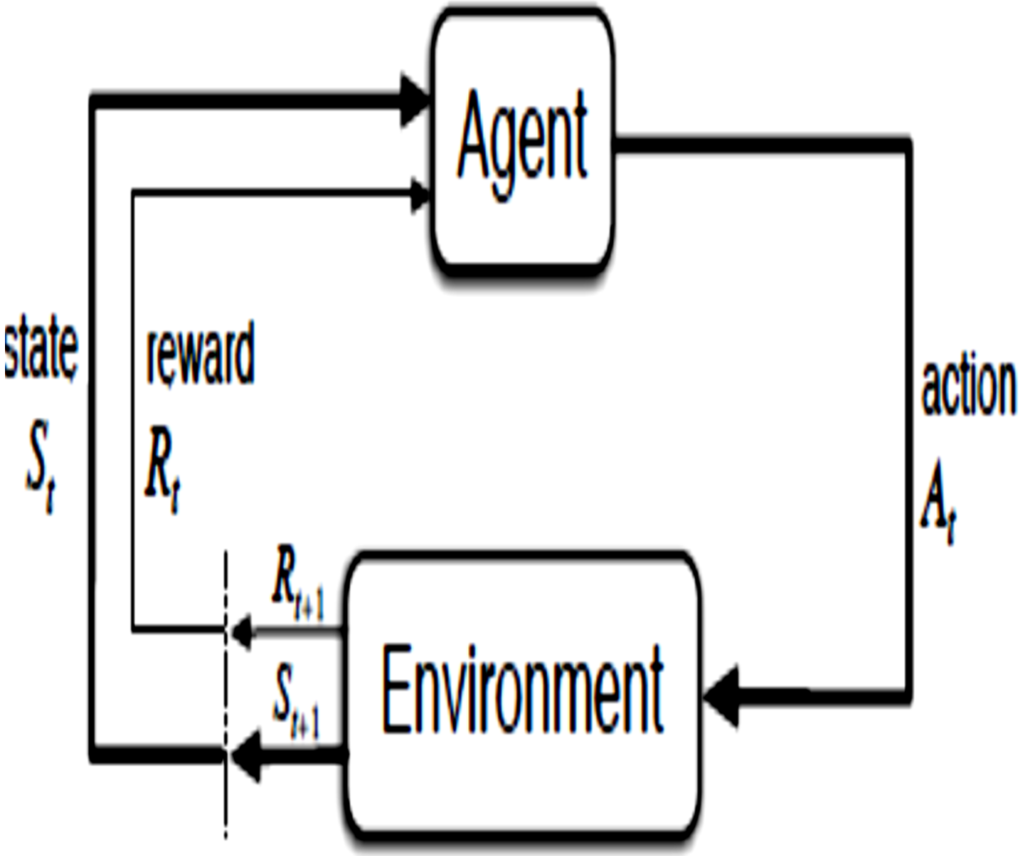
Policy

- Method to map agent's state to actions

Value

- Future reward that an agent would receive by taking an action in a particular state

Markov Decision Process



MDP and Agent give rise to a trajectory

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

Dynamics of MDP

$$p(s', r \mid s, a) \doteq \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\},$$

for all $s', s \in \mathcal{S}$, $r \in \mathcal{R}$, and $a \in \mathcal{A}(s)$

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r \mid s, a) = 1, \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}(s).$$

MDP as state transition probabilities

$$p(s'|s,a) \doteq \Pr\{S_t = s' \mid S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in \mathcal{R}} p(s', r|s,a). \tag{3.4}$$

We can also compute the expected rewards for state–action pairs as a two-argument function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$:

$$r(s,a) \doteq \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r|s,a), \tag{3.5}$$

and the expected rewards for state–action–next-state triples as a three-argument function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$,

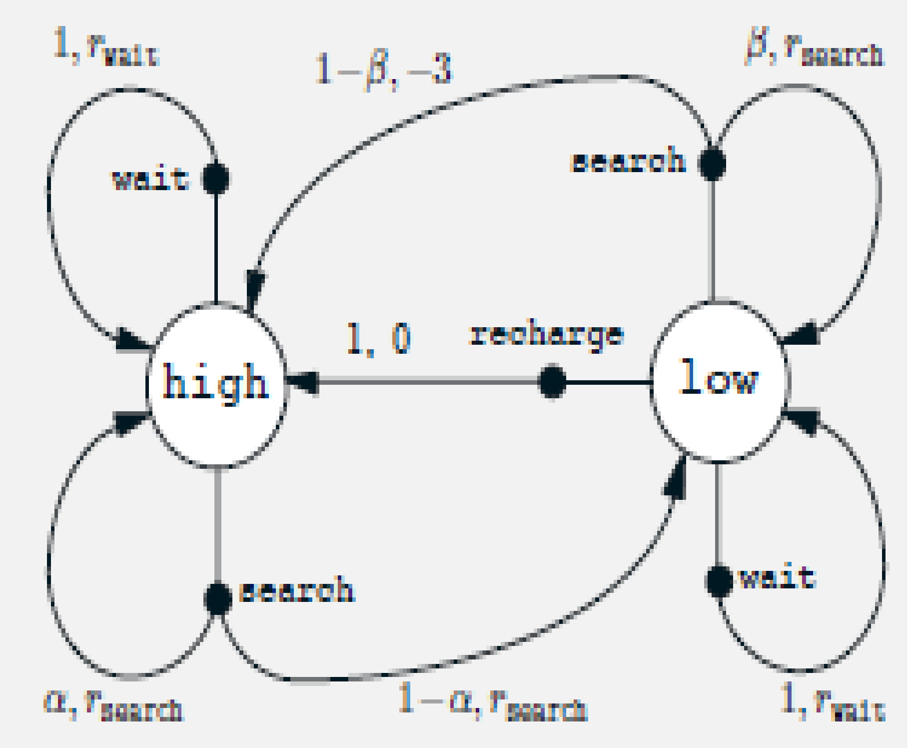
$$r(s,a,s') \doteq \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r \in \mathcal{R}} r \frac{p(s', r|s,a)}{p(s'|s,a)}. \tag{3.6}$$

Recycling Robot

A mobile robot has the job of collecting empty soda cans in an office environment. It has sensors for detecting cans, and an arm and gripper that can pick them up and place them in an onboard bin; it runs on a rechargeable battery. The robot's control system has components for interpreting sensory information, for navigating, and for controlling the arm and gripper. High-level decisions about how to search for cans are made by a reinforcement learning agent based on the current charge level of the battery. To make a simple example, we assume that only two charge levels can be distinguished, comprising a small state set $\mathcal{S} = \{\texttt{high}, \texttt{low}\}$. In each state, the agent can decide whether to (1) actively `search` for a can for a certain period of time, (2) remain stationary and `wait` for someone to bring it a can, or (3) head back to its home base to `recharge` its battery. When the energy level is `high`, recharging would always be foolish, so we do not include it in the action set for this state. The action sets are then $\mathcal{A}(\texttt{high}) = \{\texttt{search}, \texttt{wait}\}$ and $\mathcal{A}(\texttt{low}) = \{\texttt{search}, \texttt{wait}, \texttt{recharge}\}$.

Example – recycling robot

s	a	s'	$p(s' \mid s, a)$	$r(s, a, s')$
high	search	high	α	r_{search}
high	search	low	$1-\alpha$	r_{search}
low	search	high	$1-\beta$	-3
low	search	low	β	r_{search}
high	wait	high	1	r_{wait}
high	wait	low	0	-
low	wait	high	0	-
low	wait	low	1	r_{wait}
low	recharge	high	1	0
low	recharge	low	0	-



Discounted Return

Agent chooses A_t to maximize the expected discounted return:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

where γ is a parameter, $0 \leq \gamma \leq 1$, called the *discount rate*.

Discount rate determines the present value of future rewards

- a reward received k time steps in the future is worth only γ^{k-1} times what it would be worth if it were received immediately
- If $\gamma < 1$, the infinite sum in has a finite value as long as the reward sequence R_k is bounded
- If $\gamma = 0$, the agent is "myopic" , maximizing immediate rewards
- As γ approaches 1, the return objective takes future rewards into account more strongly; the agent becomes more farsighted

Returns at successive time steps

$$
\begin{aligned}
G_t &\doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\
&= R_{t+1} + \gamma\left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots\right) \\
&= R_{t+1} + \gamma G_{t+1}
\end{aligned}
$$

If reward is constant + 1

$$G_t = \sum_{k=0}^{\infty} \gamma^k = \frac{1}{1-\gamma}.$$

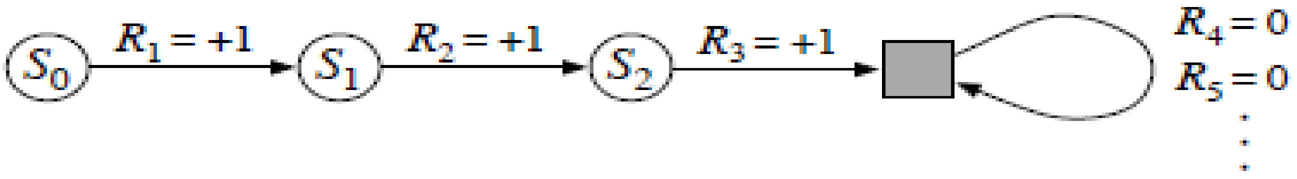
Computation of Expected Return

Consider an MDP process, and compute the expected return G_0, G_1 ….till G_5. Let $\gamma = 0.9$, the following sequence of rewards is received :

R1 = −3, R2 = 4, R3 = 2, R4 = 1, and R5 = -3, with T = 5.

Unified notation for Episodic & Continuous Tasks

State Transition Diagram



$$G_t \doteq \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k,$$

including the possibility that $T = \infty$ or $\gamma = 1$ (but not both).

Policies and Value Functions

Policy

- is a mapping from states to probabilities of selecting each possible action.
- If the agent follows policy π at time t, then π(a|s) is the probability that $A_t = a$ if $S_t = s$.

Value Function of state s under policy π (state-value function for policy π)

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right], \text{ for all } s \in \mathcal{S},$$

where $\mathbb{E}_\pi[\cdot]$ denotes the expected value of a random variable given that the agent follows policy π, and t is any time step. Note that the value of the terminal state, if any, is always zero. We call the function v_π the *state-value function for policy* π.

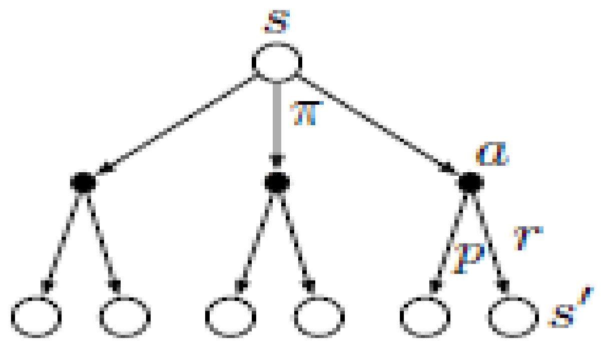
Expected return starting from s, taking the action a, and policy π: (action-value function for policy π)

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right].$$

Bellman Equation v_π

Expresses a relationship between the value of a state and the values of its successor states

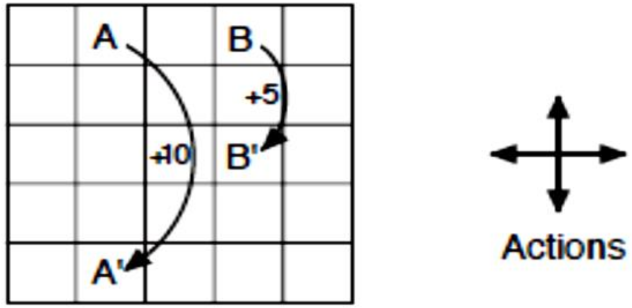
$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r \mid s, a) \Big[r + \gamma \mathbb{E}_\pi[G_{t+1} | S_{t+1} = s'] \Big] \\
&= \sum_a \pi(a|s) \sum_{s', r} p(s', r \mid s, a) \Big[r + \gamma v_\pi(s') \Big], \quad \text{for all } s \in \mathcal{S},
\end{aligned}
$$



Backup diagram for v_π

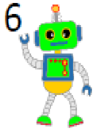
Gridworld

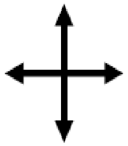
of a simple finite MDP. The cells of the grid correspond to the states of the environment. At each cell, four actions are possible: **north**, **south**, **east**, and **west**, which deterministically cause the agent to move one cell in the respective direction on the grid. Actions that would take the agent off the grid leave its location unchanged, but also result in a reward of -1. Other actions result in a reward of 0, except those that move the agent out of the special states A and B. From state A, all four actions yield a reward of $+10$ and take the agent to A′. From state B, all actions yield a reward of $+5$ and take the agent to B′.



3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

Figure 3.2: Gridworld example: exceptional reward dynamics (left) and state-value function for the equiprobable random policy (right).

1	2	3	4
5	6 	7	8
9	10	11	12
13	14	15	16

actions 

Reward is -1 for all transition

$$
\begin{aligned}
v_1(6) &= \sum_{a\in\{u,d,l,r\}} \pi(a|6) \sum_{s',r} p(s',r|6,a)[r + \gamma v_0(s')] \\
&= \sum_{a\in\{u,d,l,r\}} \underbrace{\pi(a|6)}_{=0.25\ \forall a} \sum_{s'} p(s'|6,a)[\underbrace{r}_{=-1} + \underbrace{\gamma v_0(s')}_{=0\ \forall s'}] \\
&= 0.25 * \{-p(2|6,u) - p(10|6,d) - p(5|6,l) - p(7|6,r)\} \\
&= 0.25 * \{-1 - 1 - 1 - 1\} \\
&= -1
\end{aligned}
$$

$$\Rightarrow v_1(6) = -1$$

Optimal Policies &Optimal Value functions

Optimal State Value Function

$$v_*(s) \doteq \max_{\pi} v_\pi(s),$$

for all $s \in \mathcal{S}$.

Optimal Action Value Function

$$q_*(s,a) \doteq \max_{\pi} q_\pi(s,a),$$

for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$.

Thus, we can write q_* in terms of v_* as follows:

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]$$

Bellman Optimality

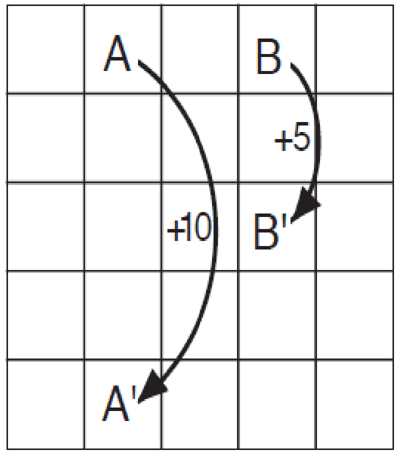
Bellman optimality equation says that the value of each state under an optimal policy must be the return the agent gets when it follows the best action as given by the optimal policy. For optimal policy π*, the optimal value function is given by:

$$
\begin{aligned}
v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s, a) \\
&= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]
\end{aligned}
$$

Given a value function q*, we can recover an optimum policy as follows:

$$
\begin{aligned}
\pi'(s) &\doteq \arg\max_a q_\pi(s, a) \\
&= \arg\max_a \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a]
\end{aligned}
$$

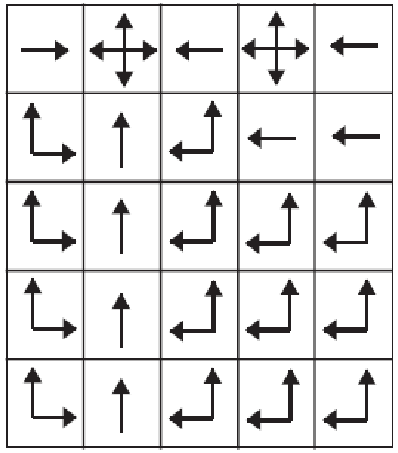
Solving Gridworld



Gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

v_*



π_*

Figure 3.5: Optimal solutions to the gridworld example.

Bellman Optimality Equations for recycling robot

example. To make things more compact, we abbreviate the states **high** and **low**, and the actions **search**, **wait**, and **recharge** respectively by **h**, **l**, **s**, **w**, and **re**. Because there are only two states, the Bellman optimality equation consists of two equations. The equation for $v_*(\mathtt{h})$ can be written as follows:

$$\begin{aligned} v_*(\mathtt{h}) &= \max \left\{ \begin{array}{l} p(\mathtt{h}|\mathtt{h},\mathtt{s})[r(\mathtt{h},\mathtt{s},\mathtt{h}) + \gamma v_*(\mathtt{h})] + p(\mathtt{l}|\mathtt{h},\mathtt{s})[r(\mathtt{h},\mathtt{s},\mathtt{l}) + \gamma v_*(\mathtt{l})], \\ p(\mathtt{h}|\mathtt{h},\mathtt{w})[r(\mathtt{h},\mathtt{w},\mathtt{h}) + \gamma v_*(\mathtt{h})] + p(\mathtt{l}|\mathtt{h},\mathtt{w})[r(\mathtt{h},\mathtt{w},\mathtt{l}) + \gamma v_*(\mathtt{l})] \end{array} \right\} \\ &= \max \left\{ \begin{array}{l} \alpha[r_{\mathtt{s}} + \gamma v_*(\mathtt{h})] + (1-\alpha)[r_{\mathtt{s}} + \gamma v_*(\mathtt{l})], \\ 1[r_{\mathtt{w}} + \gamma v_*(\mathtt{h})] + 0[r_{\mathtt{w}} + \gamma v_*(\mathtt{l})] \end{array} \right\} \\ &= \max \left\{ \begin{array}{l} r_{\mathtt{s}} + \gamma[\alpha v_*(\mathtt{h}) + (1-\alpha) v_*(\mathtt{l})], \\ r_{\mathtt{w}} + \gamma v_*(\mathtt{h}) \end{array} \right\}. \end{aligned}$$

Following the same procedure for $v_*(\mathtt{l})$ yields the equation

$$v_*(\mathtt{l}) = \max \left\{ \begin{array}{l} \beta r_{\mathtt{s}} - 3(1-\beta) + \gamma[(1-\beta)v_*(\mathtt{h}) + \beta v_*(\mathtt{l})], \\ r_{\mathtt{w}} + \gamma v_*(\mathtt{l}), \\ \gamma v_*(\mathtt{h}) \end{array} \right\}.$$

For any choice of $r_{\mathtt{s}}$, $r_{\mathtt{w}}$, α, β, and γ, with $0 \le \gamma < 1$, $0 \le \alpha, \beta \le 1$, there is exactly one pair of numbers, $v_*(\mathtt{h})$ and $v_*(\mathtt{l})$, that simultaneously satisfy these two nonlinear equations. ■

Dynamic Programming

In Reinforcement Learning

- **Policy evaluation** refers to determining the value function of a specific policy
- **Control** refers to the task of finding a policy that maximizes reward.

Control is the ultimate goal of reinforcement learning and policy evaluation is usually a necessary step to get there.

Pros & Cons

- Mathematically exact, expressible, and analyzable
- If the problem is relatively small (few states and few actions), DP methods might be the best
- May not be easy to use in continuous actions and states
- To calculate updates environment model is required
- Can get samples from this distribution by having an agent interacting with the environment and collecting experience

Dynamic Programming

To solve a given MDP, the solution must have the components to:

1. Find out how good an arbitrary policy is
2. Find out the optimal policy for the given MDP

Policy Evaluation (Prediction)

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big], \end{aligned}$$

$$\begin{aligned} v_{k+1}(s) &\doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_k(s')\Big], \end{aligned}$$

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π, the policy to be evaluated

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation

Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:

$\quad \Delta \leftarrow 0$

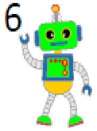
$\quad$ Loop for each $s \in \mathcal{S}$:

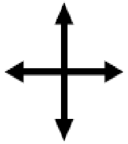
$\qquad v \leftarrow V(s)$

$\qquad V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\left[r + \gamma V(s')\right]$

$\qquad \Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$

1	2	3	4
5	6 	7	8
9	10	11	12
13	14	15	16

actions 

Reward is -1 for all transition

$$v_1(6) = \sum_{a\in\{u,d,l,r\}} \pi(a|6) \sum_{s',r} p(s',r|6,a)[r + \gamma v_0(s')]$$

$$= \sum_{a\in\{u,d,l,r\}} \underbrace{\pi(a|6)}_{= 0.25\ \forall a} \sum_{s'} p(s'|6,a)[\underbrace{r}_{=-1} + \gamma \underbrace{v_0(s')}_{= 0\ \forall s'}]$$

$$= 0.25 * \{-p(2|6,u) - p(10|6,d) - p(5|6,l) - p(7|6,r)\}$$

$$= 0.25 * \{-1 - 1 - 1 - 1\}$$

$$= -1$$

$$\Rightarrow v_1(6) = -1$$

$k = 0$

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0

$k = 3$

0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0

$\cdots$

$k = 10$

0.0	-6.1	-8.4	-9.0
-6.1	-7.7	-8.4	-8.4
-8.4	-8.4	-7.7	-6.1
-9.0	-8.4	-6.1	0.0

$\cdots$

$k = \infty$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

$\leftarrow v_\pi$

Policy Improvement

for some state s, we want to understand what is the impact of taking an action *a* that does not pertain to policy π.
Let's say we select *a* in *s*, and after that we follow the original policy π.

$$
\begin{aligned}
q_\pi(s,a) &\doteq \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big].
\end{aligned}
$$

If this happens to be greater than the value function $v_\pi(s)$, it implies that the new policy π' would be better to take.
We do this iteratively for all states to find the best policy.

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 $s \in \mathcal{S}$

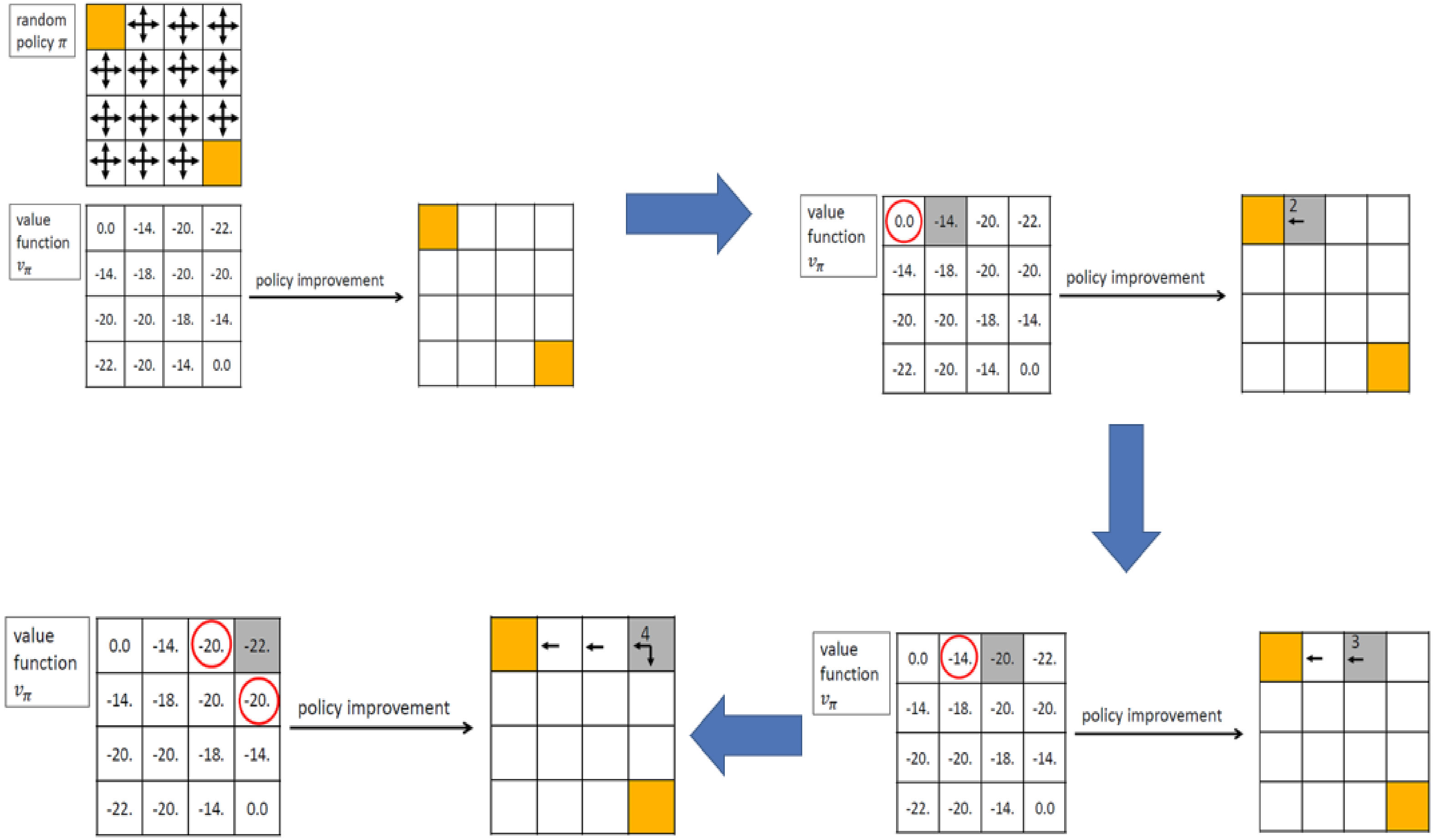
2. Policy Evaluation
   Loop:
   - $\Delta \leftarrow 0$
   - Loop for each $s \in \mathcal{S}$:
     - $v \leftarrow V(s)$
     - $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s)) \big[r + \gamma V(s') \big]$
     - $\Delta \leftarrow \max(\Delta, |v - V(s)|)$

   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*

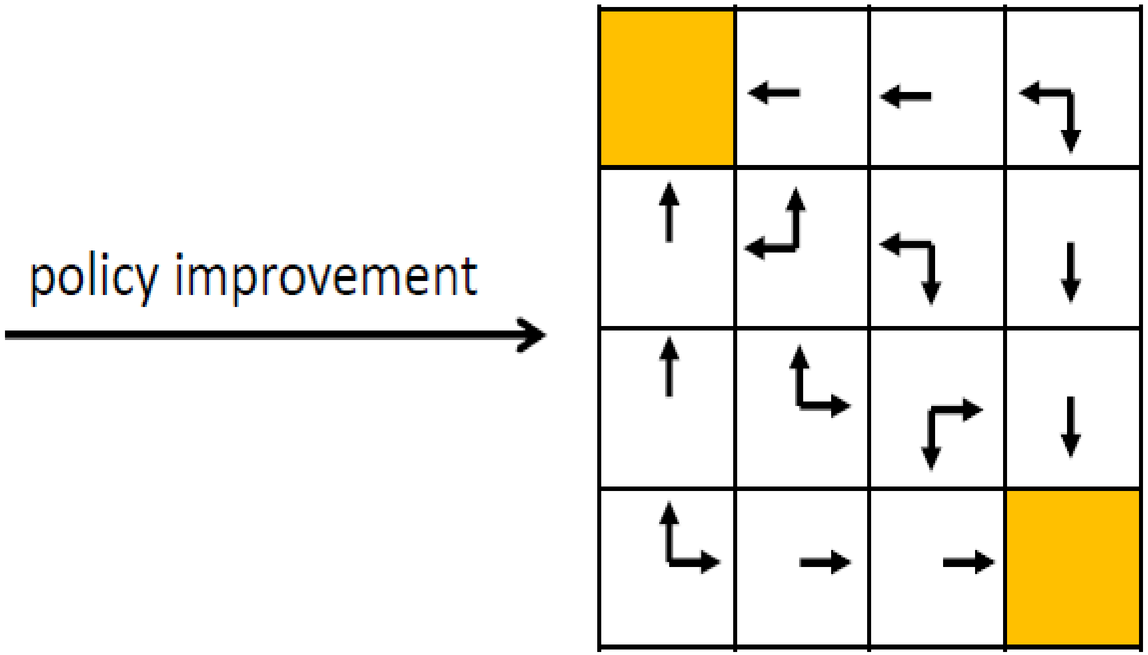
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Policy Iteration



Policy Iteration

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

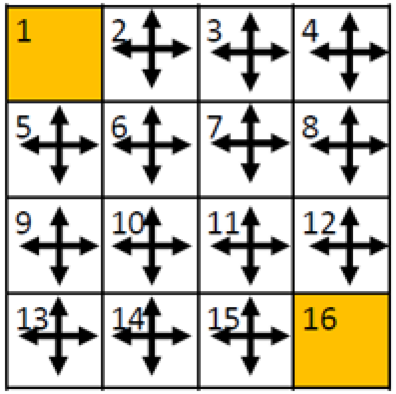


Policy Iteration

Overall, after the policy improvement step using v_{π}, we get the new policy π':

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

π_0



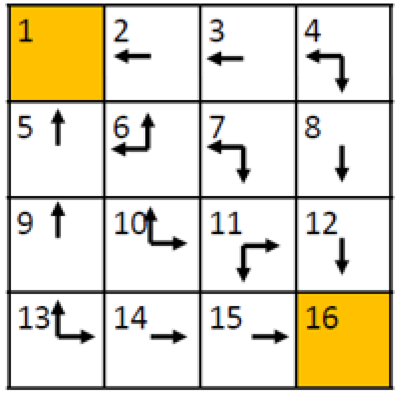
$\xrightarrow{E}$

v_{π_0}

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0

$\xrightarrow{I}$

π_1



Value Iteration

This algorithm is called *value iteration*. It can be written as a particularly simple update operation that combines the policy improvement and truncated policy evaluation steps:

$$\begin{aligned} v_{k+1}(s) &\doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_k(s')\Big], \end{aligned} \tag{4.10}$$

for all $s \in \mathcal{S}$. For arbitrary v_0, the sequence $\{v_k\}$ can be shown to converge to v_* under the same conditions that guarantee the existence of v_*.

- *does a single iteration of policy evaluation at each step*
- *Then, for each state, it takes the maximum action value to be the estimated state value*

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $\quad v \leftarrow V(s)$
| $\quad V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$
| $\quad \Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r|s, a)\big[r + \gamma V(s')\big]$

Asynchronous DP

Major drawback is it involves operations over the entire state set of the MDP, that is, they require sweeps of the state set

Asynchronous DP

- are in-place iterative DP algorithms that do not do systematic sweeps of the state set
- Update the values of states in any order whatsoever, using whatever values of other states are available
- The values of some states may be updated several times before the values of others are updated once

Asynchronous algorithms

- make it easier to intermix computation with real-time interaction
- To solve a given MDP, we can run an iterative DP algorithm at the same time that an agent is actually experiencing the MDP.

Efficiency of Dynamic Programming

DP is thought to be of limited applicability because of the curse of dimensionality, the number of states often grows exponentially with the number of state variables.

DP method is

- guaranteed to find an optimal policy in polynomial time even though the total number of (deterministic) policies is k^n
- If n and k denote the number of states and actions, number of computational operations is less than some polynomial function of n and k.

Linear programming methods can also be used to solve MDPs, and in some cases, their worst-case convergence guarantees are better than those of DP methods

References

1. Russell S., and Norvig P., Artificial Intelligence A Modern Approach (3e), Pearson 2010

2. Richard S Sutton, Andrew G Barto, Reinforcement Learning, second edition, MIT Press

3. https://www.coursera.org/learn/fundamentals-of-reinforcement-learning/home/week/1