3. Finite Markov Decision Processes

Reinforcement Learning – RL is all about interacting with the environment to achieve a goal.
Agent – Anything that is interacting with the environment is called as an agent.
Environment – Environment is one to which the agent is interacting.

The Agent–Environment Interface

The reinforcement learning problem is meant to be a straightforward framing of the problem of learning from interaction to achieve a goal. The learner and decision-maker is called the agent. The thing it interacts with, comprising everything outside the agent, is called the environment. These interact continually, the agent selecting actions and the environment responding to those actions and presenting new situations to the agent. The environment also gives rise to rewards, special numerical values that the agent tries to maximize over time. A complete specification of an environment defines a task, one instance of the reinforcement learning problem.

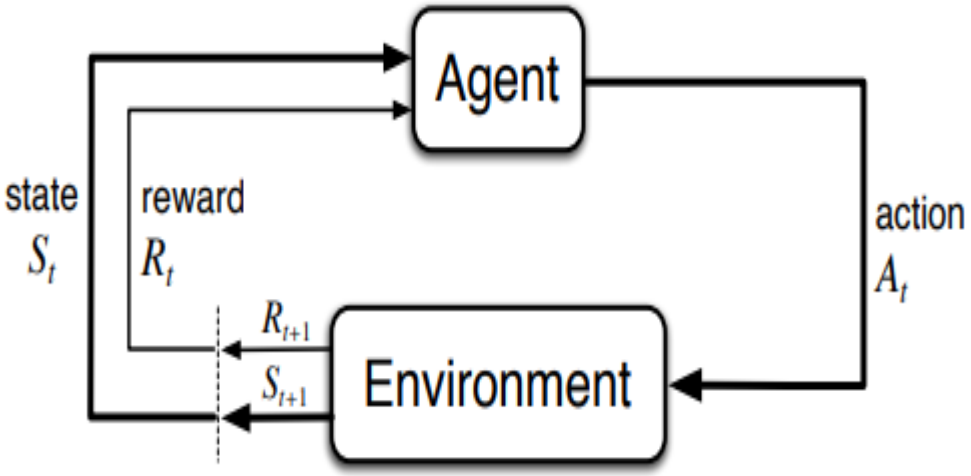


Fig: The agent–environment interaction in reinforcement learning.

The agent and environment interact at each of a sequence of discrete time steps, t = 0, 1, 2, . . . At each time step t, the agent receives some representation of the environment's state, $S_t \in S$, where S is the set of possible states, and on that basis selects an action, $A_t \in A(S_t)$, where $A(S_t)$ is the set of actions available in state S_t. One time step later, in part as a consequence of its action, the agent receives a numerical reward, $R_{t+1} \in R \subset \mathbb{R}$, and finds itself in a new state, S_{t+1}.

At each time step, the agent implements a mapping from states to probabilities of selecting each possible action. This mapping is called the agent's policy and is denoted π_t, where $\pi_t(a|s)$ is the probability that $A_t = a$ if $S_t = s$. Reinforcement learning methods specify how the agent changes its policy as a result of its experience. The agent's goal, roughly speaking, is to maximize the total amount of reward it receives over the long run.

The reinforcement learning framework is a considerable abstraction of the problem of goal-directed learning from interaction. It proposes that whatever the details of the sensory, memory, and control apparatus, and whatever objective one is trying to achieve, any problem of learning goal-directed behaviour can be reduced to three signals passing back and forth between an agent and its environment: one signal to represent the choices made by the agent (the actions), one signal to represent the basis on which the choices are made (the states), and one signal to define the agent's goal (the rewards).

Examples regarding good ways of representing states and actions:

1.Bioreactor

States**: -** *List of sensor readings and symbolic inputs (nutrients and bacteria).*

Actions**: -** *Passing of target temperatures and target stirring rates to the control system.*

Rewards**: -** *Moment-by-moment measures of the rate at which the useful chemical is produced.*

2. Pick-and-Place Robot

States**: -** *The latest readings of joint angles and velocities.*

Actions**: -** *The voltages applied to each motor at each joint.*

Reward**:** *+1 for each object successfully picked up and placed.*

3. Recycling Robot

States**:** - *Weighting, Recharging, Searching.*

Actions**:** - *1. Search for a Can for a certain period of time.*

2. Wait for someone to bring it a Can.

3. Go to its home base to recharge its battery.

Reward**:** *+1 if it secures an empty Can.*

Goals and Rewards

Goal: To maximize the total amount of reward it receives.

Reward: The reward is a simple number, $R_t \in \mathbb{R}$. For example, in chess game, the reward is defined only for terminal states.

- +1 if the agent wins.
- 0 if there is draw.
- -1 if the agent lost the game.

In reinforcement learning, the purpose or goal of the agent is formalized in terms of a special reward signal passing from the environment to the agent. The use of a reward signal to formalize the idea of a goal is one of the most distinctive features of reinforcement learning.

For example, to make a robot learn to walk, researchers have provided reward on each time step proportional to the robot's forward motion. In making a robot learn how to escape from a maze, the reward is often −1 for every time step that passes prior to escape; this encourages the

agent to escape as quickly as possible. To make a robot learn to find and collect empty soda cans for recycling, one might give it a reward of zero most of the time, and then a reward of +1 for each can collected. One might also want to give the robot negative rewards when it bumps into things or when somebody yells at it. For an agent to learn to play checkers or chess, the natural rewards are +1 for winning, −1 for losing, and 0 for drawing and for all nonterminal positions.

Returns

The agent's goal is to maximize the cumulative reward it receives in the long run. If the sequence of rewards received after time step t is denoted R_{t+1}, R_{t+2}, R_{t+3}, . . ., then we seek to maximize the expected return, where the return Gt is defined as some specific function of the reward sequence. In the simplest case the return is the sum of the rewards:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \ldots\ldots + R_T,$$

where T is a final time step. This approach makes sense in applications in which there is a natural notion of final time step, that is, when the agent– environment interaction breaks naturally into subsequence, which we call episodes such as plays of a game, trips through a maze, or any sort of repeated interactions. Each episode ends in a special state called the terminal state.

There are two kinds of tasks:

1) Episodic Task
2) Continuing Task

Episodic task is a task where the interaction ends after a finite number of Time steps.

Continuing task is a task where the interaction never ends i.e. it contains infinite number of Time steps. For example, cleaning robot.

The agent tries to select actions so that the sum of the discounted rewards it receives over the future is maximized. In particular, it chooses A_t to maximize the expected discounted return:

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

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

The 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 has a finite value as long as the reward sequence $\{R_k\}$ is bounded.

If $\gamma = 0$, the agent is "myopic" in being concerned only with maximizing immediate rewards: its objective in this case is to learn how to choose A_t so as to maximize only R_{t+1}.

If $\gamma = 1$, the agent is "Farsighted" it means the agent giving the same weight to all the rewards, whether it is immediate reward or future reward it does not matter.

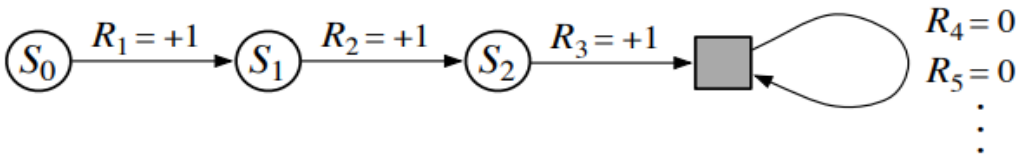
Unified Notation for Episodic and Continuing Tasks

Since there are two different tasks it is required to develop one set of algorithms for episodic task and another set of algorithms for continuing task which might be difficult.

Hence, we need to find a mechanism to combine both episodic and continuing task into single task into single task.

Fortunately, we can unify these two tasks into a single task & can provide a unified notation. This can be done by making a small change to episodic task.

If we introduce a dummy state at the end of an episodic task and make it as an entry point to a continuing task where we generate a reward of 0 only then these two tasks can be unified into 1.



For example, consider the state transition diagram. Here the solid square represents the special absorbing state corresponding to the end of an episode. Starting from S_0, we get the reward sequence +1, +1, +1, 0, 0, 0, . . . Summing these, we get the same return whether we sum over the first T rewards (here T = 3) or over the full infinite sequence.

$$G_t = \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1},$$

Where $T = \infty$ & $\gamma = 1$,

If $T = \infty$ then it is continuing task.

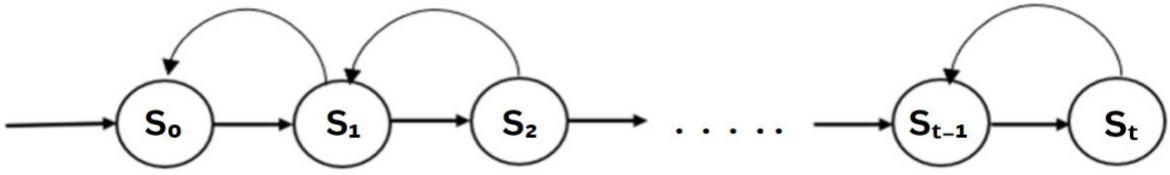
If $T \neq \infty$ & $\gamma = 1$ then it is episodic task.

Markov Property

Markov Property states that the next state depends only on the current state and is independent of past states. In particular, we formally define a property of environments and their state signals that is of particular interest, called the Markov property

It means that the current state encompasses all the information of the past history.

Let $S_0, S_1, S_2, \ldots, S_t$ denote sequence of states as given below



Here S_1 depends only on S_0,

S_2 depends only on S_1

S_3 depends only on S_2

S_t depends only on S_{t-1}

In MDP, the dynamics of the environment can be defined by the following probability distribution.

$$\Pr\{R_{t+1} = r, S_{t+1} = s' \mid S_0, A_0, R_1, \ldots, S_{t-1}, A_{t-1}, R_t, S_t, A_t\},$$

If the state signal has the Markov property, on the other hand, then the environment's response at t + 1 depends only on the state and action representations at t, in which case the environment's dynamics can be defined by specifying only

$$p(s', r|s, a) = \Pr\{R_{t+1} = r, S_{t+1} = s' \mid S_t, A_t\},$$

for all s′, r, and histories, S_0, A_0, R_1, ..., S_{t-1}, A_{t-1}, R_t, S_t, A_t. In this case, the environment and task as a whole are also said to have the Markov property.

Markov Decision Processes

Markov Decision Process is a mathematical framework used to describe on environment in a decision-making scenario where outcomes are random. A reinforcement learning task that satisfies the Markov property is called a Markov decision process, or MDP. If the state and action spaces are finite, then it is called a finite Markov decision process (finite MDP). Finite MDPs are particularly important to the theory of reinforcement learning.

In MDP there are 3 basic elements

1. S = Set of States
2. A = Set of Actions
3. R = Reward

Given any state and action s and a, the probability of each possible pair of next state and reward, s′, r, is denoted

$$p(s', r|s, a) = \Pr\{S_{t+1}=s', R_{t+1} = r \mid S_t=s, A_t=a\}.$$

The equation for computing the reward is given

$$r(s, a) = \mathbb{E}[R_{t+1} \mid S_t=s, A_t=a] = \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} p(s', r|s, a),$$

The state-transition probabilities,

$$p(s'|s, a) = \Pr\{S_{t+1}=s' \mid S_t=s, A_t=a\} = \sum_{r\in\mathcal{R}} p(s', r|s, a),$$

The expected rewards for state–action–next-state triples,

$$r(s,a,s') = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a, S_{t+1} = s'] = \frac{\sum_{r \in \mathcal{R}} r\, p(s', r|s, a)}{p(s'|s, a)}.$$

Example1: Recycling Robot MDP

The recycling robot can be turned into a simple example of an MDP by simplifying it and providing some more details. (Our aim is to produce a simple example, not a particularly realistic one.) Recall that the agent makes a decision at times determined by external events (or by other parts of the robot's control system). At each such time the robot decides whether it should (1) actively search for a can, (2) remain stationary and wait for someone to bring it a can, or (3) go back to home base to recharge its battery. Suppose the environment works as follows. The best way to find cans is to actively search for them, but this runs down the robot's battery, whereas waiting does not. Whenever the robot is searching, the possibility exists that its battery will become depleted. In this case the robot must shut down and wait to be rescued (producing a low reward).

The agent makes its decisions solely as a function of the energy level of the battery. It can distinguish two levels, high and low, so that the state set is **S = {high, low}.** Let us call the possible decisions the agent's actions wait, search, and recharge. 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 agent's action sets are

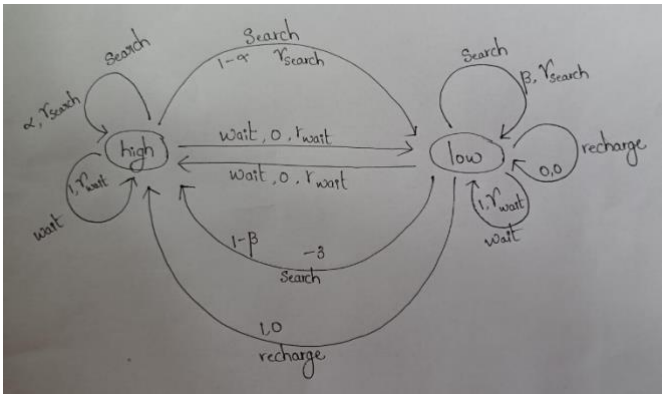
A(high) = {search, wait}

A(low) = {search, wait, recharge}

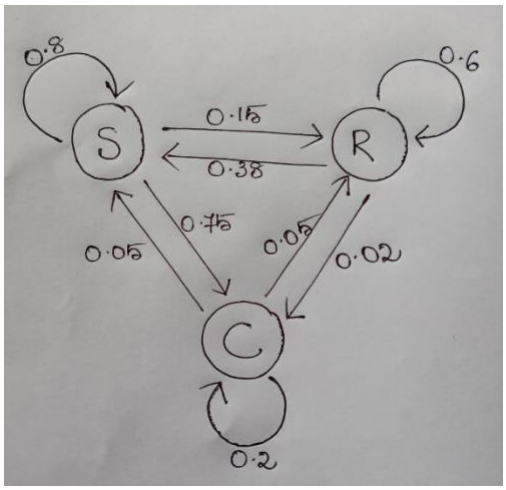
Transition Table

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

Transition Diagram



Example2: Weather



Today \ Tomorrow	*Sunny*	*Rainy*	*Cloudy*
Sunny	0.8	0.15	0.05
Rainy	0.38	0.6	0.02
Cloudy	0.75	0.05	0.2

Q1. Given that today is sunny then what is the probability that tomorrow is sunny and day after tomorrow is rainy?

Sol. T_1 T_2 T_3

S S R

$P(T_3=R, T_2=S \mid T_1=S) = P(T_3=R \mid T_2=S) * P(T_2=S \mid T_1=S)$

$= 0.15 * 0.8$

$= 0.12$

The probability that tomorrow is sunny and day after tomorrow is rainy when today is sunny is 12%.

Q2. Given that today is cloudy and yesterday was rainy then what is the probability that tomorrow is sunny?

Sol. T_0 T_1 T_2

R C S

$P(T_2=S, T_1=C \mid T_0=R) = P(T_2=S \mid T_1=C) * P(T_1=C \mid T_0=R)$

$= 0.75 * 0.02$

$= 0.015$

The probability that tomorrow is sunny when today is cloudy and yesterday was rainy is 1.5%

Q3. Compute the probability for the following sequence.

S R R R C C

Sol. $P(S) * P(R|S) * P(R|R) * P(R|R) * P(C|R) * P(C|C)$

$= 0.75 * 0.15 * 0.6 * 0.6 * 0.02 * 0.2$

$= 0.000162000$

The probability for the given sequence is 0.0162%

Value Functions

All reinforcement learning algorithms involve estimating *value functions*- functions of states (or of state–action pairs) that estimate how good it is for the agent to be in a given state (or how good it is to perform a given action in a given state).

There are 2 kinds of value functions:

1. State value function
2. Action value function

1.) State Value Function:

It estimates how good the state is under some policy π.

It is denoted by $V_\pi(s)$.

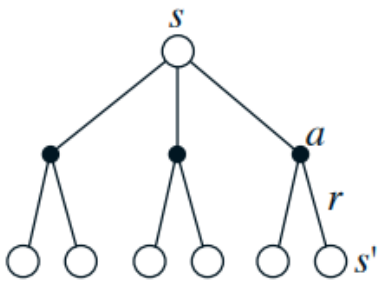
$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \;\middle|\; S_t = s\right],$$

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. The value of the terminal state, if any, is always zero. We call the function V_π the state-value function for policy π.

Bellman Equation for state value function

$$\begin{aligned}
v_\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \;\middle|\; S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \;\middle|\; S_t = s\right] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a)\left[r + \gamma \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \;\middle|\; S_{t+1} = s'\right]\right] \\
&= \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a)\left[r + \gamma v_\pi(s')\right],
\end{aligned}$$

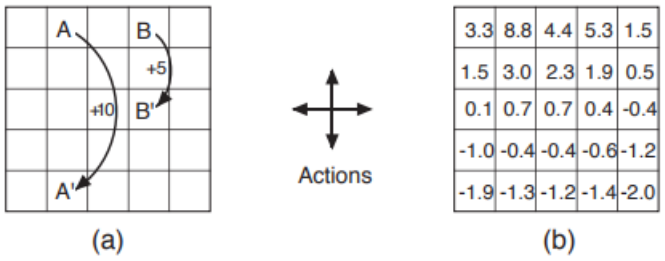
Backup Diagram for $V_\pi(s)$



Starting from state *s*, the root node at the top, the agent could take any of some set of actions. The environment could respond with one of several next states, s′, along with a reward, r. The Bellman equation averages over all the possibilities, weighting each by its probability of occurring. It states that the value of the start state must equal the (discounted) value of the expected next state, plus the reward expected along the way. The value function $V_\pi(s)$ is the unique solution to its Bellman equation.

Example:

Consider a 5x5 grid world, where the agent has to move. The agent can perform four actions i.e., up, down, left, right. The reward that agent gets in the states is 0, except in the states A and B. In state A it gets a reward of +10 and state B it gets a reward of +5. When the agent in any state gets out of the grid it penalized i.e., it gets a reward of -1. In addition to this in state A the agent can take any action resulting the state A the agent can take any action resulting the state A′. Similarly in state B the agent can take any action resulting to state B.



Grid example: (a) exceptional reward dynamics; (b) state-value function for the equiprobable random policy.

2.) Action Value Function:

Estimates value of taking an action *a* in state *s*.

It is denoted by $q_*(s, a)$.

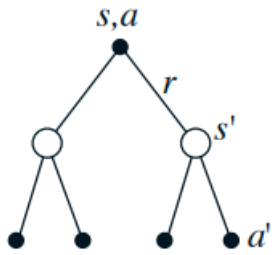
It can be written as follows:

$$q_*(s,a) = \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \;\Big|\; S_t = s, A_t = a\Big]$$

The Bellman optimality equation for q∗ is

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

Backup Diagram for Action Value function



Optimal Value Function

The main objective of a reinforcement learning agent is to find the optimal policy. For any RL problem, there exists many policies.

For example, assume that there are 'n' states and in each state 'k' actions are possible. There number of policies are n^k.

For 5x5 grid world problem

- 25 states
- 4 actions in each state

Therefore, number of policies possible is 25^4.

A policy π is defined to be better than or equal to a policy π′ if its expected return is greater than or equal to that of π′ for all states. In other words, $\pi \geq \pi'$ if and only if $V_\pi(s) \geq V_{\pi'}(s)$ for all $s \in S$. There is always at least one policy that is better than or equal to all other policies. This is an optimal policy. Although there may be more than one, we denote all the optimal policies by π_*. They share the same state-value function, called the optimal state-value function, denoted v_*, and defined as

$$v_*(s) = \max_\pi v_\pi(s),$$

for all $s \in S$.

Optimal policies also share the same optimal action-value function, denoted q_*, and defined as

$$q_*(s,a) = \max_\pi q_\pi(s,a),$$

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

For the state–action pair (s, a), this function gives the expected return for taking action 'a' in state 's' and thereafter following an optimal policy. We can write q_* in terms of v_* as follows:

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

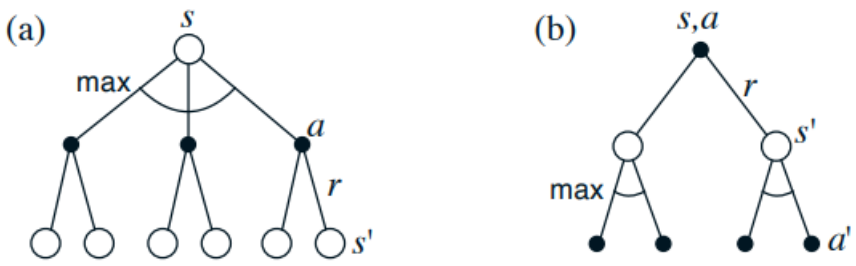
Bellman Equation for Optimal State Value Function

$$
\begin{aligned}
v_*(s) &= \max_{a \in A(s)} q_{\pi_*}(s,a) \\
&= \max_a \mathbb{E}_{\pi^*}[G_t \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi^*}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \;\middle|\; S_t = s, A_t = a\right] \\
&= \max_a \mathbb{E}_{\pi^*}\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \;\middle|\; S_t = s, A_t = a\right] \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_{a \in A(s)} \sum_{s',r} p(s', r|s, a)\big[r + \gamma v_*(s')\big].
\end{aligned}
$$

The Bellman optimality equation for q_* is

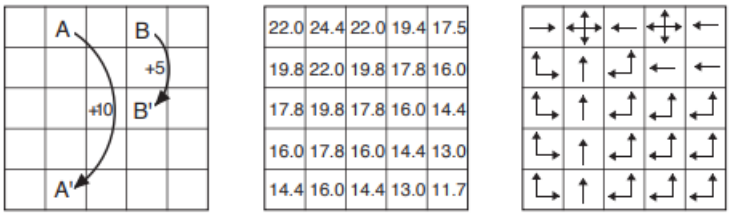
$$
\begin{aligned}
q_*(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \;\middle|\; S_t = s, A_t = a\right] \\
&= \sum_{s',r} p(s', r|s, a)\left[r + \gamma \max_{a'} q_*(s', a')\right].
\end{aligned}
$$

Backup Diagrams



Backup diagrams for (a) v_* and (b) q_*

Example:



a) gridworld b) v_* c) π_*

Optimal solutions to the gridworld example.