FULL REINFORCEMENT LEARNING

BY: DIMPLE BOHRA

CONTENTS

1. Introduction to Full RL
2. Difference between Full RL and Immediate RL
3. Agent, Environment,
4. Goals, Rewards, Return
5. Policy in Full RL
6. Episodic and Continuing Tasks

Reference: Chapter 3 from Sutton and Barto,

http://www-edlab.cs.umass.edu/cs689/lectures/RL%20Lecture%203.pdf

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.

AGENT ENVIRONMENT INTERFACE

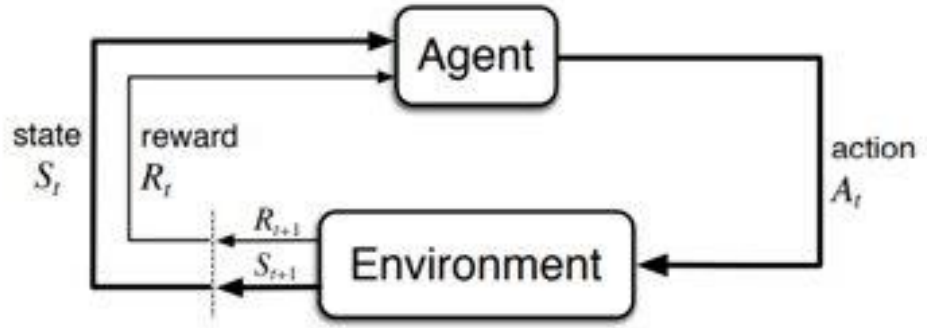


Figure 3.1: The agent–environment interaction in reinforcement learning.

- More specifically, the agent and environment interact at each of a sequence of discrete time steps, t = 0, 1, 2, 3, ...
- 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(St) 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, $R_{t+1} \in R$, and finds itself in a new state, S_{t+1}.

AGENT ENVIRONMENT INTERFACE

- 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.

AGENT ENVIRONMENT INTERFACE

- The boundary between agent and environment is drawn closer to the agent
- For example, the motors and mechanical linkages of a robot and its sensing hardware should usually be considered parts of the environment rather than parts of the agent.
- Similarly, if we apply the framework to a person or animal, the muscles, skeleton, and sensory organs should be considered part of the environment.
- Rewards, too, presumably are computed inside the physical bodies of natural and artificial learning systems but are considered external to the agent.
- The general rule we follow is that anything that cannot be changed arbitrarily by the agent is considered to be outside of it and thus part of its environment.

AGENT ENVIRONMENT INTERFACE

- The agent often knows quite a bit about how its rewards are computed as a function of its actions and the states in which they are taken. But we always consider the reward computation to be external to the agent because it defines the task facing the agent and thus must be beyond its ability to change arbitrarily.
- In fact, in some cases the agent may know everything about how its environment works and still face a difficult reinforcement learning task, just as we may know exactly how a puzzle like Rubik's cube works, but still unable to solve it.
- The agent– environment boundary represents the limit of the agent's absolute control, not of its knowledge

AGENT ENVIRONMENT INTERFACE

- The agent–environment boundary can be located at different places for different purposes.
- In a complicated robot, many different agents may be operating at once, each with its own boundary.
- For example, one agent may make high-level decisions which form part of the states faced by a lower-level agent that implements the high-level decisions.
- In practice, the agent–environment boundary is determined once one has selected particular states, actions, and rewards, and thus has identified a specific decision-making task of interest.

AGENT ENVIRONMENT INTERFACE EXAMPLES

- **Pick-and-Place Robot:** Consider using reinforcement learning to control the motion of a robot arm in a repetitive pick-and-place task. If we want to learn movements that are fast and smooth, the learning agent will have to control the motors directly and have low-latency information about the current positions and velocities of the mechanical linkages. The actions in this case might be the voltages applied to each motor at each joint, and the states might be the latest readings of joint angles and velocities. The reward might be +1 for each object successfully picked up and placed. To encourage smooth movements, on each time step a small, negative reward can be given as a function of the moment-to-moment “jerkiness” of the motion

AGENT ENVIRONMENT INTERFACE EXAMPLES

- **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. This agent has to decide whether the robot should (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. This decision has to be made either periodically or whenever certain events occur, such as finding an empty can.

GOALS AND REWARDS

- 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. At each time step, the reward is a simple number, $R_t \in R$.
- Informally, the agent's goal is to maximize the total amount of reward it receives. This means maximizing not immediate reward, but **cumulative reward in the long run.**
- We can clearly state this informal idea as the reward hypothesis:

➢ *That all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward).*

GOALS AND REWARDS

- The use of a reward signal to formalize the idea of a goal is one of the most distinctive features of reinforcement learning.
- Although formulating goals in terms of reward signals might at first appear limiting, in practice it has proved to be flexible and widely applicable.
- The best way to see this is to consider examples of how it has been, or could be, used.
- **Examples**

1. To make a robot learn to walk, researchers have provided reward on each time step proportional to the robot's forward motion.
2. 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.
3. 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

GOALS AND REWARDS

- In particular, the reward signal is not the place to impart to the agent prior knowledge about how to achieve what we want it to do.
- For example, a chess-playing agent should be rewarded only for actually winning, not for achieving subgoals such as taking its opponent's pieces or gaining control of the center of the board. If achieving these sorts of subgoals were rewarded, then the agent might find a way to achieve them without achieving the real goal. Means it might find a way to take the opponent's pieces even at the cost of losing the game.
- The reward signal is your way of communicating to the robot what you want it to achieve, not how you want it to be achieved.

GOALS AND REWARDS

- Newcomers to reinforcement learning are sometimes surprised that the rewards—which define the goal of learning—**are computed in the environment or in the agent?**
- Example: if the goal concerns a robot's internal energy reservoirs, then these are considered to be part of the environment; if the goal concerns the positions of the robot's limbs, then these too are considered to be part of the environment—that is, the agent's boundary is drawn at the interface between the limbs and their control systems. These things are considered internal to the robot but external to the learning agent.
- For our purposes, it is convenient to place the boundary of the learning agent not at the limit of its physical body, but at the limit of its control

REWARD

- Reward is:

1. Scalar quantity (just a number)
2. Outside direct control of the agent
3. Bounded

BENEFITS OF SCALAR REWARD

- It gels well in some sense with how we do decision making and that's why it also seems natural. Example: whether I should go to college or not? Whether I should book the car now? Etc....
- Having single scalar value makes optimization of solution easy

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 what precise aspect of this sequence do we wish to maximize?
- In general, we seek to maximize the expected return, where the return G_t is defined as some specific function of the reward sequence.
- In the simplest case the return is the sum of the rewards:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T \text{ , where T is a final time step.}$$

RETURNS

- 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 subsequences, which we call episodes
- Examples: 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, followed by a reset to a standard starting state or to a sample from a standard distribution of starting states.
- Tasks with episodes of this kind are called **episodic tasks.** In episodic tasks we sometimes need to distinguish the set of all nonterminal states, denoted S, from the set of all states plus the terminal state, denoted $\mathcal{S}^+$

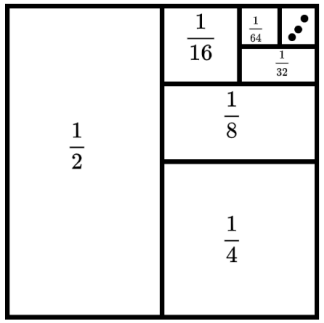
EPISODIC AND CONTINUOUS TASKS

- An episodic task lasts a finite amount of time. For example, playing a single game is an episodic task, which you win or lose. In an episodic task, there might be only a single reward, at the end of the task, and one option is to distribute the reward evenly across all actions taken in that episode.
- In a continuous task, rewards might be assigned with discounting, so more recent reactions receive greater reward, and actions a long time in the past receive a vanishingly small reward.

EXAMPLES OF EPISODIC AND CONTINUOUS TASKS

- Episodic tasks are the tasks that have a terminal state (end). Episodes are considered as agent-environment interactions from initial to final states.
- For example, in a car racing video game, you start the game (initial state) and play the game until it is over (final state). This is called an episode. Once the game is over, you start the next episode by restarting the game, and you will begin from the initial state irrespective of the position you were in the previous game. So, each episode is independent of the other.
- In a continuous task, there is not a terminal state. Continuous tasks will never end.
- For example, a personal assistance robot does not have a terminal state.

DISCOUNTING



- **The discount factor** (a number between 0-1) is a clever way to scale down the rewards more and more after each step so that, the total sum remains bounded. The discounted sum of rewards is called return (G_t) in RL.

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

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

- According to this discounting approach, 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.

DISCOUNTED RETURNS

- 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 each of the agent's actions happened to influence only the immediate reward, not future rewards as well, then a myopic agent could maximize rewards by separately maximizing each immediate reward.
- But in general, acting to maximize immediate reward can reduce access to future rewards so that the return may actually be reduced.
- As γ approaches 1, the objective takes future rewards into account more strongly: the agent becomes more farsighted.

RETURNS

- Example: Pole balancing

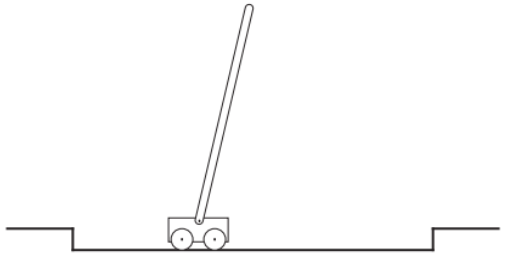


Figure 3.2: The pole-balancing task.

- The objective here is to apply forces to a cart moving along a track so as to keep a pole hinged to the cart from falling over.
- A failure is said to occur if the pole falls past a given angle from vertical or if the cart runs off the track.
- The pole is reset to vertical after each failure. This task could be treated as episodic, where the natural episodes are the repeated attempts to balance the pole.
- The reward in this case could be +1 for every time step on which failure did not occur, so that the return at each time would be the number of steps until failure.

RETURNS

- Example: Pole balancing

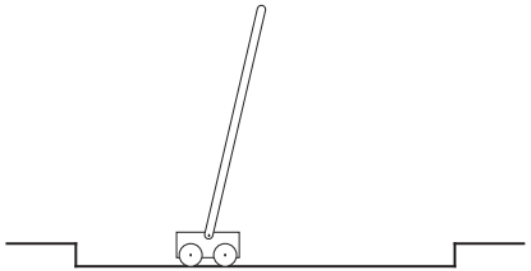


Figure 3.2: The pole-balancing task.

- Alternatively, we could treat pole-balancing as a continuing task, using discounting. In this case the reward would be −1 on each failure and zero at all other times. The return at each time would then be related to $-\gamma^K$

  where K is the number of time steps before failure.

- In either case, the return is maximized by keeping the pole balanced for as long as possible

UNIFIED NOTATION FOR EPISODIC AND CONTINUOUS TASKS

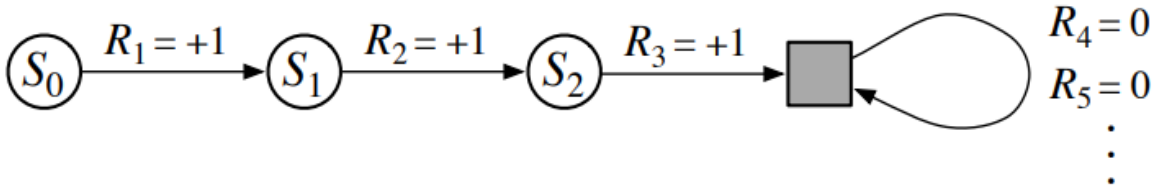
- Episodic tasks requires some additional notation. Rather than one long sequence of time steps, we need to consider a series of episodes, each of which consists of a finite sequence of time steps.
- We number the time steps of each episode starting a new from zero. Therefore, we have to refer not just to S_t , the state representation at time t, but to $\mathbf{S_{t,i}}$, the state representation at time t of episode i (and similarly for $\mathbf{A_{t,i}}$, $\mathbf{R_{t,i}}$, $\mathbf{\Pi_{t,i}}$, $\mathbf{T_i}$, etc.).

UNIFIED NOTATION FOR EPISODIC AND CONTINUOUS TASKS

- We need one convention to obtain a single notation that covers both episodic and continuing tasks
- $G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$, where T is a final time step.

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

- These can be unified by considering episode termination to be the entering of a special absorbing state that transitions only to itself and that generates only rewards of zero



UNIFIED NOTATION FOR EPISODIC AND CONTINUOUS TASKS

- Thus, we can define return using the convention of omitting episode numbers when they are not needed and including the possibility that γ = 1 if the sum remains defined (e.g., because all episodes terminate). Alternatively, we can also write the return as

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

Including the possibility that T = ∞ or γ = 1 (but not both)

POLICY IN FULL RL

- Policies in Reinforcement Learning (RL) are shrouded in a mystery.
- Simply stated, a policy $\pi: s \rightarrow a$ or $\pi(a \mid s)$ is any function that returns a feasible action for a problem.
- For instance, you could simply take the first action that comes to mind, select an action at random, or run a heuristic. However, what makes RL special is that we actively anticipate the downstream impact of decisions and learn from our observations; we therefore expect some intelligence in our policies.

POLICY IN FULL RL

- For example, imagine a robot moves across the room and the task is to get to the target point (x, y), where it gets a reward. Here:
- A room is an *environment*
- Robot's current position is a *state*
- A *policy* is what an agent does to accomplish this task:
  - dumb robots just wander around randomly until they accidentally end up in the right place (policy #1)
  - others may, for some reason, learn to go along the walls most of the route (policy #2)
  - smart robots plan the route in their "head" and go straight to the goal (policy #3)
  - Obviously, some policies are better than others, and there are multiple ways to assess them, namely *state-value function* and *action-value function*.
  - The goal of RL is to learn the best policy. *A policy defines the learning agent's way of behaving at a given time.*

THE MARKOV PROPERTY

- It is a property of environments and their state signals.
- ***State*** *: is the position of the agents at a specific time-step in the environment. So, whenever an agent performs an action, the environment gives the reward and a new state where the agent reached by performing the action.*
- **Actions** can be any decision we want the agent to learn, and state can be anything which can be useful in choosing actions
- ***Transition*** *: Moving from one state to another is called Transition.*
- ***Transition Probability****: The probability that the agent will move from one state to another is called transition probability.*

THE MARKOV PROPERTY

- *The **Markov Property** states that :*

"Future is Independent of the past given the present"

- Mathematically we can express this statement as :

$$P[S_{t+1} \mid S_t] = P[S_{t+1} \mid S_1, \ldots\ldots, S_t]$$

MARKOV PROPERTY

- A state signal that succeeds in retaining all relevant information is said to be Markov, or to have the Markov property

THE MARKOV PROPERTY

- Let us formally define the Markov property for the reinforcement learning problem.
- To keep the mathematics simple, we assume here that there are a finite number of states and reward values. This enables us to work in terms of sums and probabilities rather than integrals and probability densities, but the argument can easily be extended to include continuous states and rewards.
- Consider how a general environment might respond at time t+ 1 to the action taken at time t. In the most general, causal case this response may depend on everything that has happened earlier. In this case the dynamics can be defined only by specifying the complete probability distribution:

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

THE MARKOV PROPERTY

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

- 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

$$p(s', r|s, a) = \Pr\{R_{t+1} = r, S_{t+1} = s' \mid S_t, A_t\} \quad \text{Eq. 2}$$

In other words, a state signal has the Markov property, and is in a Markov state, if and only if equation 2 is equal to equation 1 for all s', r and histories, S0, A0, R1, ..., St−1, At−1, Rt

In this case, the environment and task as a whole are also said to have the Markov property

BENEFITS OF MARKOV PROPERTY

- If an environment has the Markov property, then its one-step dynamics (Eq.2) enable us to predict the next state and expected next reward given the current state and action.
- That is, the best policy for choosing actions as a function of a Markov state is just as good as the best policy for choosing actions as a function of complete histories.
- Even when the state signal is non-Markov, it is still appropriate to think of the state in reinforcement learning as an approximation to a Markov state

EXAMPLE

- In the **pole-balancing task**, a state signal would be Markov if it specified exactly, or made it possible to reconstruct exactly, the position and velocity of the cart along the track, the angle between the cart and the pole, and the rate at which this angle is changing (the angular velocity). In an idealized cart–pole system, this information would be sufficient to exactly predict the future behavior of the cart and pole, given the actions taken by the controller.

MARKOV PROPERTY

- Given any state and action s and a, the probability of each possible pair of next state and reward, s', r, is denoted as:

$$p(s', r|s, a) = \Pr\{S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a\}. \tag{3.6}$$

- These quantities completely specify the dynamics of a finite MDP.

MARKOV PROPERTY

- Given any state and action s and a, the probability of each possible pair of next state and reward, s', r, is denoted as:

$$p(s', r|s, a) = \Pr\{S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a\}. \tag{3.6}$$

- For Markov State from S[t] to S[t+1] i.e. any other successor state , the state transition probability is given by

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

OR

$$\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$$

State Transition Probability

STATE TRANSITION PROBABILITY

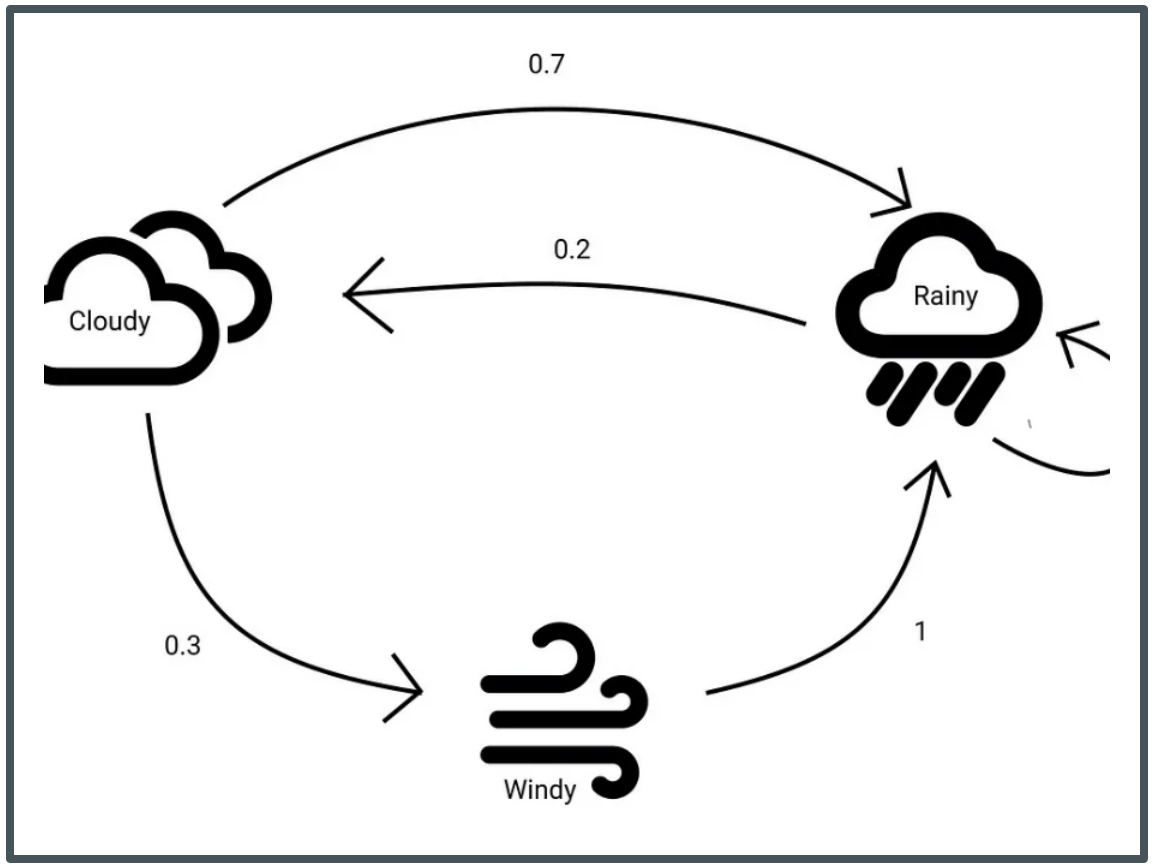
- We can formulate State Transition probability into a State Transition probability matrix by :

$$P = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ p_{21} & p_{22} & \cdots & p_{2n} \\ & \cdots & \cdots & \\ p_{n1} & p_{n2} & \cdots & p_{nn} \end{bmatrix}$$

State Transition Probability Matrix

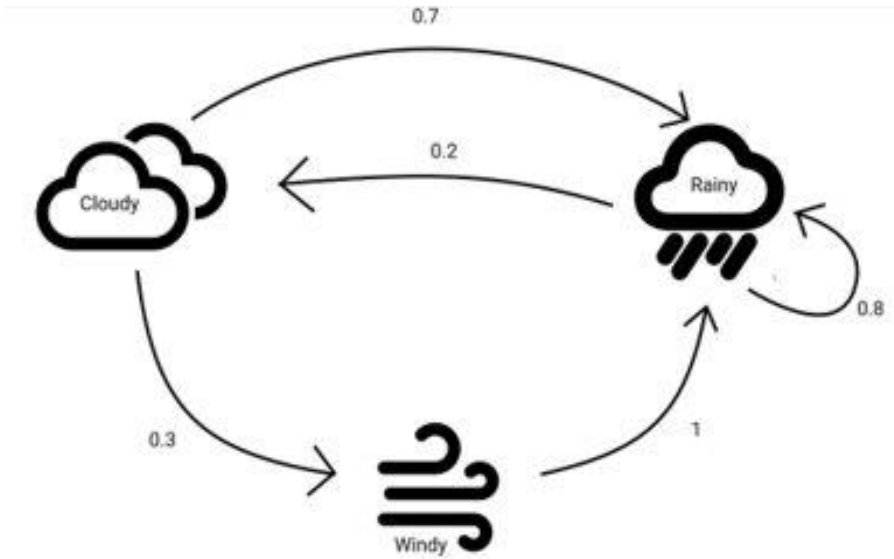
- Each row in the matrix represents the probability from moving from our original or starting state to any successor state. Sum of each row is equal to 1

MARKOV CHAIN



- The Markov Chain consists of a sequence of states that follow the Markov property. This Markov Chain actually is the probabilistic model that depends on the current state to predict the next state.
- To understand Markov Property and Markov Chain, consider weather prediction as an example. If the current state is cloudy, then the next state could be rainy or windy. In the middle of the first state to the next state, there is a probability that we called transition probability.

MARKOV CHAIN



- When the current state is cloudy, 70% of the next state will be rainy, or 30% will be windy. If the current state is windy, there is a 100% possibility the next state will be rainy.
- Then, when the state is Rainy, there is an 80% probability that the next state will keep rainy, and there are 20% will be changed to cloudy.

MARKOV CHAIN

We can show these states and transition probability into a table or a matrix as below:

Current State	Next State	Transition Probability
Cloudy	Rainy	0.7
Cloudy	Windy	0.3
Rainy	Rainy	0.8
Rainy	Cloudy	0.2
Windy	Rainy	1.0

	Cloudy	Rainy	Windy
Cloudy	0.0	0.7	0.3
Rainy	0.2	0.8	0.0
Windy	0.0	1.0	0.0

Markov chain consists of a set of states along with their transition probabilities.

MARKOV REWARD PROCESS

- The Markov Reward Process (MRP) is an extension of the Markov chain with an additional reward function. So, it consists of states, a transition probability, and a reward function.
- This reward function gives us the reward that we get from each state. This function will tell us the reward we obtain in the state cloudy, the reward we obtain in the state of windy, and the rainy state.
- This reward also can be a positive or negative value. Mathematically can define reward as:

$$R_s = E[R_{t+1} \mid S_t]$$

MARKOV DECISION PROCESS

- Discussed about Markov Property, Markov Chain, and Markov Reward Process which are the basics of the Markov Decision Process (MDP). In the Markov Decision Process, we have action as additional from the Markov Reward Process.
- 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.

MARKOV DECISION PROCESS

We define **MRP** as (S,P, R,ɣ) , where :

- S is a set of states,
- P is the Transition Probability Matrix,
- R is the Reward function,
- ɣ is the discount factor

- We define **MDP** as (S,P, R,A,ɣ) , where :

- S is a set of states,
- P is the Transition Probability Matrix,
- R is the Reward function, we saw earlier,
- A is a set of actions
- ɣ is the discout factor

STATE - ACTION VALUE FUNCTION OR Q- FUNCTION

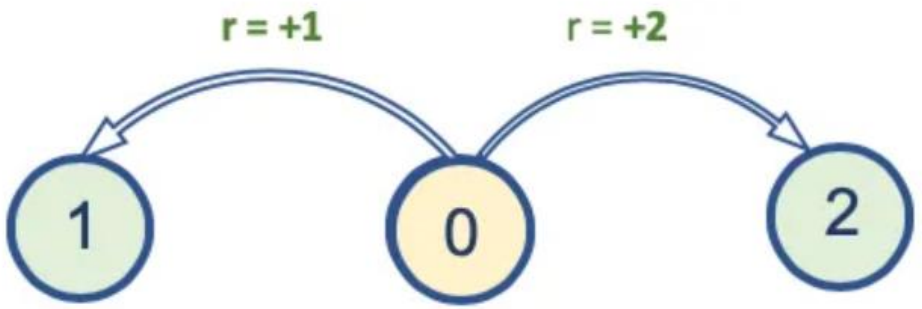
State value function: $$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right]$$

- State-action value function specifies **how good** it is for the agent to take action (a) in a state (s) with a **policy π.**
- Mathematically, we can define the **State-action** value function as :

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

Basically, it tells us the value of performing a certain action(a) in a state(s) with a policy π.

RELATION BETWEEN STATE VALUE AND ACTION FUNCTION

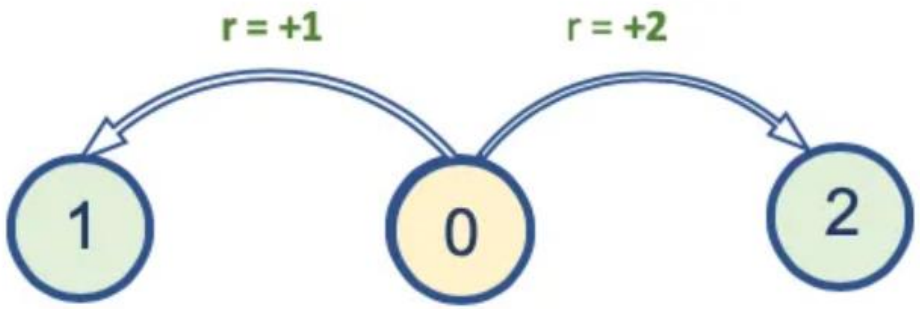


The value of one state is always calculated in terms of some policy that our Agent follows.

Consider some example of policy:

- The Agent always goes left
- The Agent always goes right
- The Agent goes left with a probability of 0.5 and right with a probability of 0.5
- The Agent goes left with a probability of 0.2 and right with a probability of 0.8

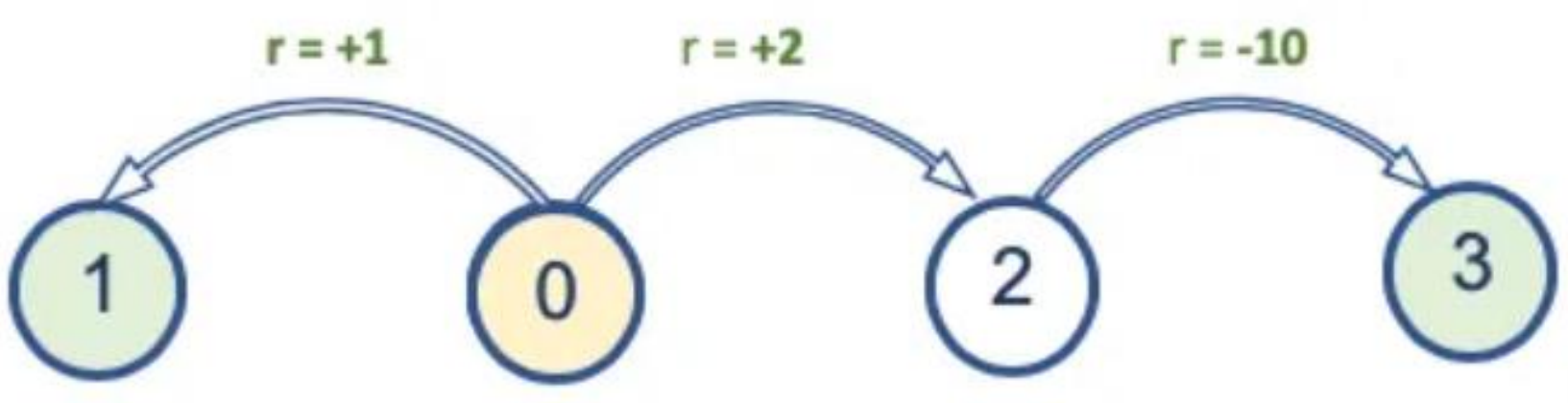
RELATION BETWEEN STATE VALUE AND ACTION FUNCTION



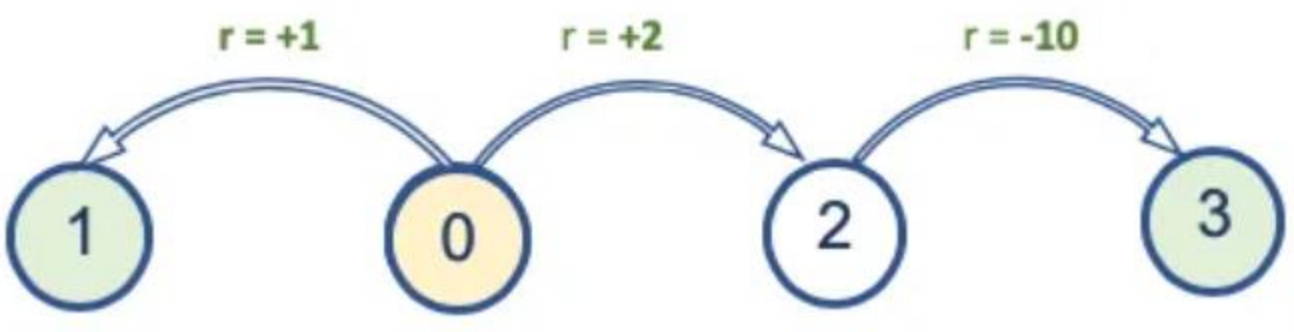
The value of state 0, V(0), in each of the above policies is:

1. The value of state 0 in the case of the “always left” Agent is V(0)=1.0 (every time it goes left, it obtains +1 and the episode ends).
2. For the “always right” Agent, the value of state 0 is V(0)= 2.0 (every time it goes right, it obtains +2 and the episode ends).
3. For the “0.5 left+0.5 right” Agent, the value is V(0)=1.0 x 0.5 + 2.0 x 0.5 = 1.5
4. For the “0.2 left+0.8 right” Agent, the value is V(0)=1.0 x 0.2 + 2.0 x 0.8 = 1.8

RELATION BETWEEN STATE VALUE AND ACTION FUNCTION



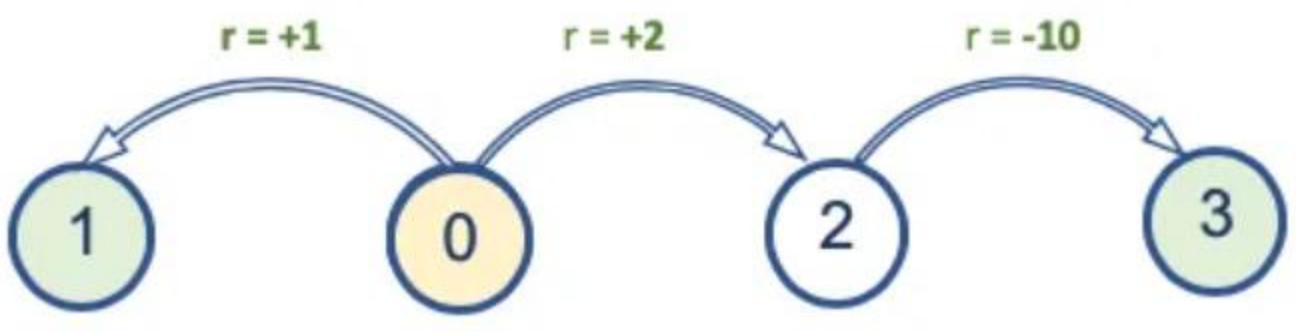
RELATION BETWEEN STATE VALUE AND ACTION FUNCTION



With that addition, for each policy, the V(0) will be:

1. For the “always left”, is the same: V(0)=+1.0
2. For the “always right”: V(0)= 2.0 + (–10.0) = –8.0
3. For the “0.5 left+0.5 right”: V(0)=1.0 * 0.5 + (2.0+(-10.0))* 0.5 = -3.5
4. For the “0.2 left+0.8 right”: V(0)=1.0 * 0.2 + (2.0+(-10.0)) * 0.8 = -6.2

RELATION BETWEEN STATE VALUE AND ACTION FUNCTION



For the “0.5 left+0.5 right”: $V_3(0)=1.0 * 0.5 + \underline{(2.0+(-10.0)})* 0.5 = -3.5$

$q_3(s,a)$

RELATION BETWEEN STATE VALUE AND ACTION FUNCTION

- We denote with $\pi(a|s)$ the probability that a policy, π, selects an action, a, given a current state, s. Note that the sum of probabilities of all outbound actions from s is equal to 1:

$$\sum_a \pi(a|s) = 1$$

- We can assert that the state-value function is equivalent to the sum of the action-value functions of all outgoing (from s) actions a, multiplied by the policy probability of selecting each action:

$$V_\pi(s) = \sum_a \pi(a|s) \cdot Q_\pi(s,a)$$

BELLMAN EQUATION

- BELLMAN Equation is the central element of many Reinforcement Learning algorithms.
- We can say that the Bellman equation decomposes the value function into two parts, the immediate reward plus the discounted future values.
- This equation simplifies the computation of the value function, such that rather than summing over multiple time steps, we can find the optimal solution of a complex problem by breaking it down into simpler, recursive subproblems and finding their optimal solutions.

BELLMAN EQUATION FOR STATE VALUE FUNCTION

$$V_\pi(s) = \sum_a \pi(a|s) \cdot \sum_{s'} P^a_{ss'}(r(s,a) + \gamma V_\pi(s'))$$

OR

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

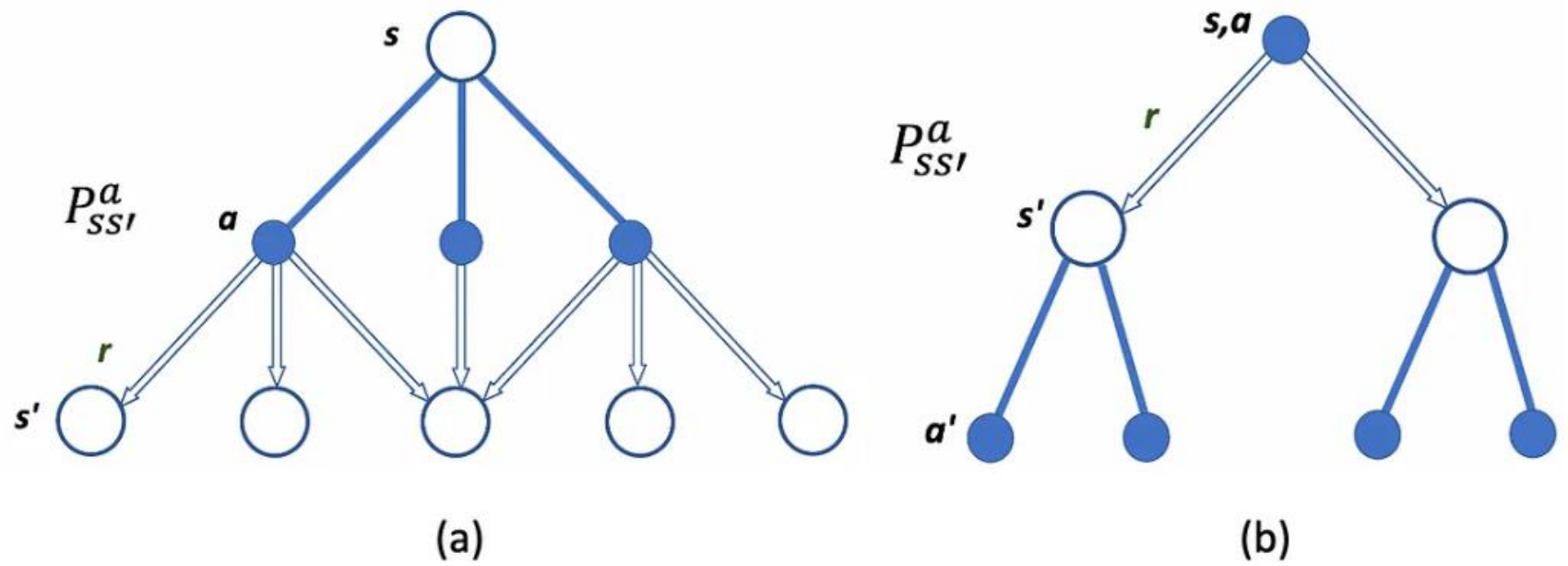
It expresses a relationship between the value of a state and the values of its successor states.

BELLMAN EQUATION FOR STATE VALUE FUNCTION

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

- This equation tells us how to find the value of a state **s** following a policy π.
- We can intuitively see that it recursively breaks down the value computation into an immediate expected reward from the next state, ***r(s,a)***, plus the value of a successor state, ***Vπ(s')***, with a discount factor γ.
- The above equation also expresses the stochasticity of the Environment with the sum over the policy probabilities.

BELLMAN EQUATION



Backup diagrams for (a) $V\pi(s)$ and (b) $Q\pi(s,a)$.

P means the probability of action ***a***, issued in state ***s***, ending up in state ***s'*** (with reward ***r***)

BACK UP DIAGRAMS

- They represent relationships that form the basis of the update or backup operations that are at the heart of reinforcement learning methods.
- These operations transfer value information back to a state (or a state–action pair) from its successor states (or state–action pairs).
- Unlike transition graphs, the state nodes of backup diagrams do not necessarily represent distinct states; for example, a state might be its own successor.
- Explicit arrowheads are not used in these diagrams because time always flows downward in a backup diagram

OPTIMAL VALUE FUNCTIONS

- In an MDP environment, there are many different value functions according to different policies. ***The optimal Value function is one which yields maximum value compared to all other value function***.
- So, mathematically Optimal State-Value Function can be expressed as :

$$v_*(s) = \max_{\pi} v_\pi(s)$$

v∗(s) tells us what is the maximum reward we can get from the system.

OPTIMAL STATE ACTION VALUE FUNCTION

- **Optimal State-Action Value Function** tells us the maximum reward we are going to get if we are in state s and taking action a from there on-wards.
- Mathematically, It can be defined as :

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

OPTIMAL POLICY

- *Optimal Policy is one which results in optimal value function.*
- We find an optimal policy by maximizing over ***q**** (s, a) i.e. our optimal state-action value function.
- We solve q*(s,a) and then we pick the action that gives us most optimal state-action value function(q*(s,a)).
- The above statement can be expressed as:

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \underset{a \in \mathcal{A}}{\text{argmax}}\ q_*(s, a) \\ 0 & \text{otherwise} \end{cases}$$

OPTIMAL POLICY

- The optimal Policy can be expressed as:

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \underset{a \in \mathcal{A}}{\text{argmax}}\ q_*(s,a) \\ 0 & otherwise \end{cases}$$

- This says for a state s we pick the action a with probability 1, if it gives us the maximum q*(s,a). So, if we know q*(s,a) we can get an optimal policy from it.

OPTIMAL VALUE FUNCTION

- 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.
- 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 EQUATION

- 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

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

BELLMAN OPTIMALITY EQUATION

- The Bellman optimality equation for q∗ is:

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

BELLMAN OPTIMALITY AND BELLMAN EXPECTATION EQUATION

Bellman expectation equation →

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

Bellman optimality equation →

$$v_*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s',r} p(s',r|s,a)\left[r + \gamma v_*(s')\right]$$

Bellman Optimality equation is the same as Bellman Expectation Equation, but the only difference is instead of taking the average of the actions our agent can take we take the action with the max value.

BELLMAN OPTIMALITY EQUATION

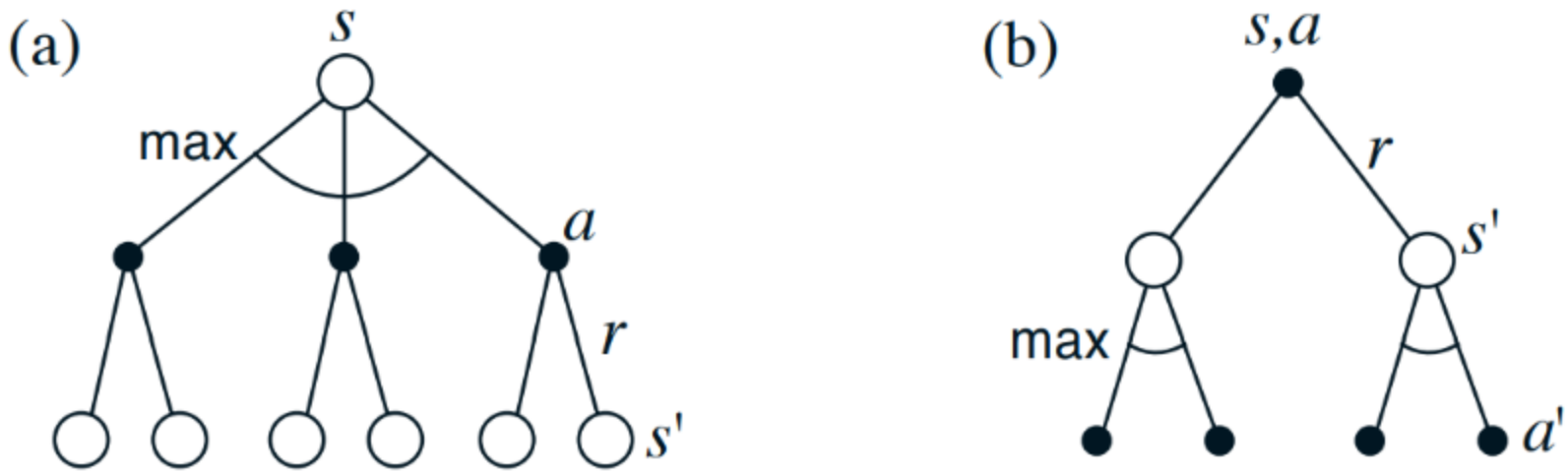


Figure 3.7: Backup diagrams for (a) v_* and (b) q_*

These diagrams spans of future states and actions considered in the Bellman optimality equations for v∗ and q∗. These are the same as the backup diagrams for Vπ and Qπ except that arcs have been added at the agent's choice points to represent that the maximum over that choice is taken rather than the expected value given some policy

EXAMPLE: RECYCLING ROBOT

- At each step, robot has to decide whether it should (1) actively search for a can, (2) wait for someone to bring it a can, or (3) go to home base and recharge.
- Searching is better but runs down the battery; if runs out of power while searching, has to be rescued (which is bad).
- Decisions made on basis of current energy level: high, low.
- Reward = number of cans collected

SOLUTION OF RECYCLING ROBOT PROBLEM

$S = \{\texttt{high}, \texttt{low}\}$

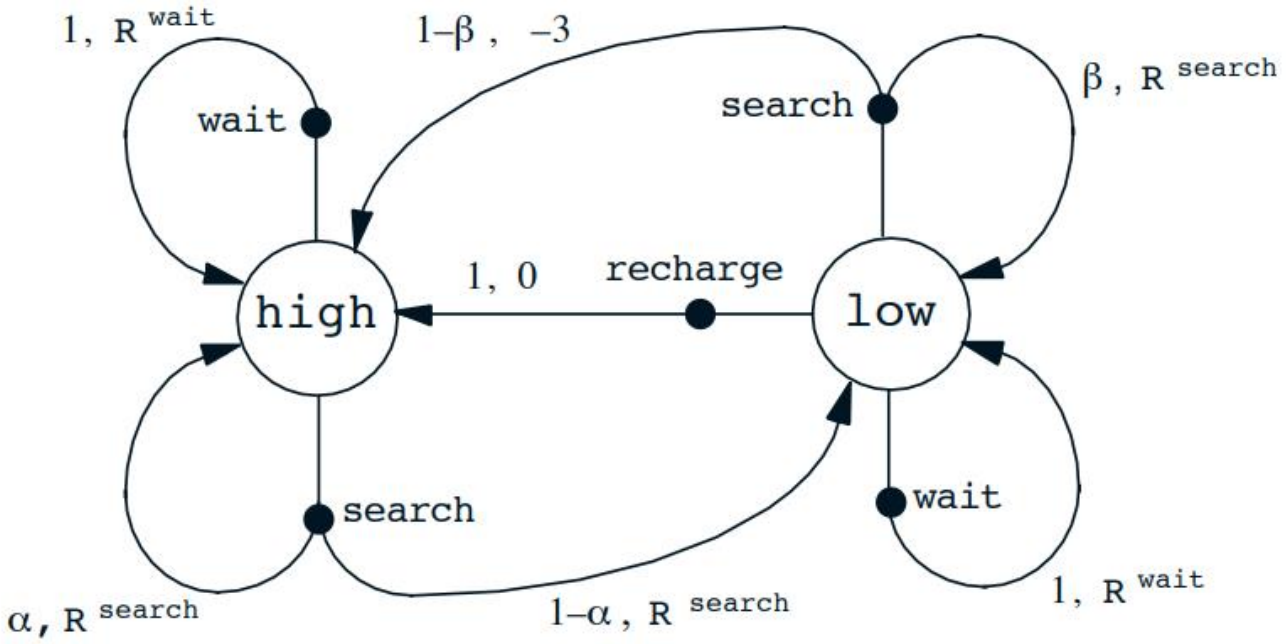
$A(\texttt{high}) = \{\texttt{search}, \texttt{wait}\}$

$A(\texttt{low}) = \{\texttt{search}, \texttt{wait}, \texttt{recharge}\}$

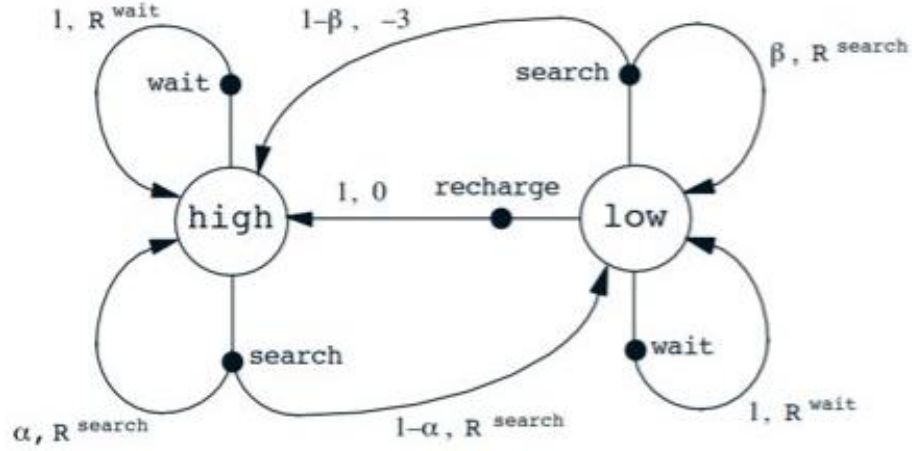
$R^{\texttt{search}}$ = expected no. of cans while searching

$R^{\texttt{wait}}$ = expected no. of cans while waiting

$R^{\texttt{search}} > R^{\texttt{wait}}$



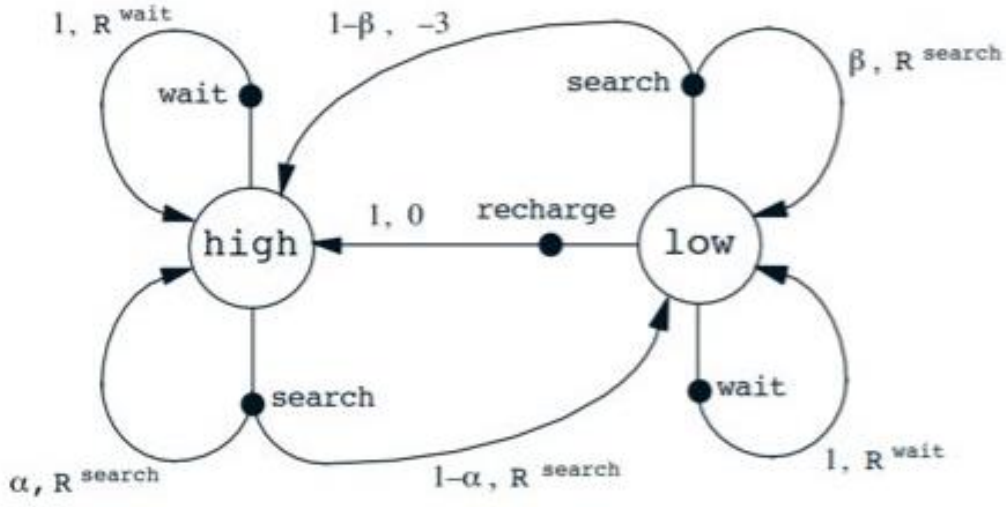
SOLUTION OF RECYCLING ROBOT PROBLEM



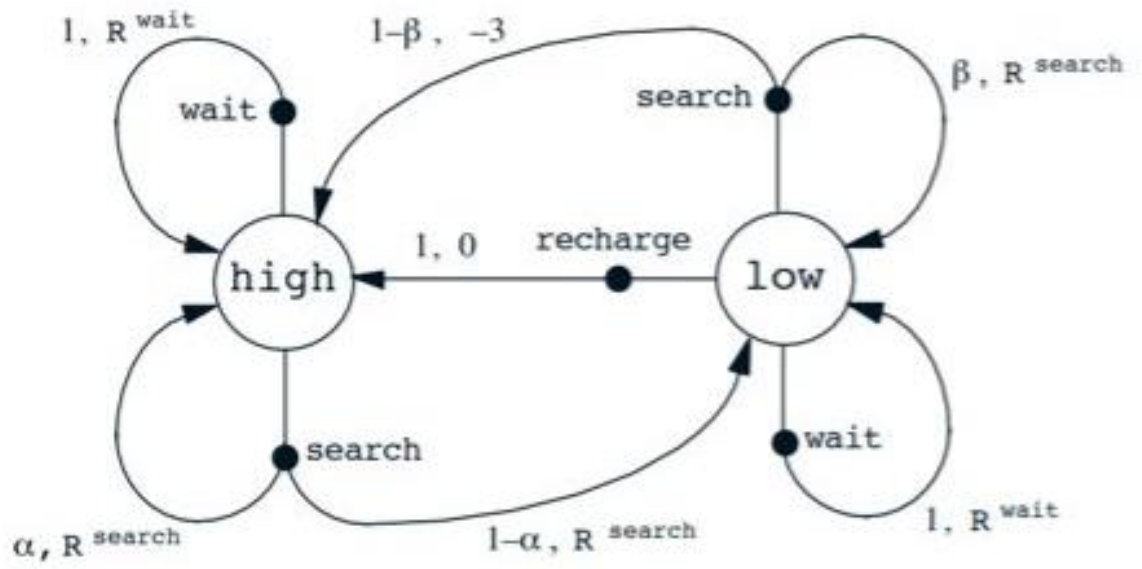
- There are two kinds of nodes: state nodes and action nodes. There is a state node for each possible state (a large open circle labeled by the name of the state), and an action node for each state–action pair (a small solid circle labeled by the name of the action and connected by a line to the state node).
- Starting in state s and taking action a moves you along the line from state node s to action node (s, a).
- Then the environment responds with a transition to the next state's node via one of the arrows leaving action node (s, a). Each arrow corresponds to a triple (s, s', a), where s' is the next state, and we label the arrow with the transition probability, $p(s'|s, a)$, and the expected reward for that transition, $r(s, a, s')$.
- Note that the transition probabilities labeling the arrows leaving an action node always sum to 1.

SOLUTION OF RECYCLING ROBOT PROBLEM

- If the energy level is high, then a period of active search can always be completed without risk of depleting the battery. A period of searching that begins with a high energy level leaves the energy level high with probability α and reduces it to low with probability 1−α.
- On the other hand, a period of searching undertaken when the energy level is low leaves it low with probability β and depletes the battery with probability 1−β.
- In the latter case, the robot must be rescued, and the battery is then recharged back to high.
- Each can collected by the robot counts as a unit reward, whereas a reward of −3 results whenever the robot has to be rescued.

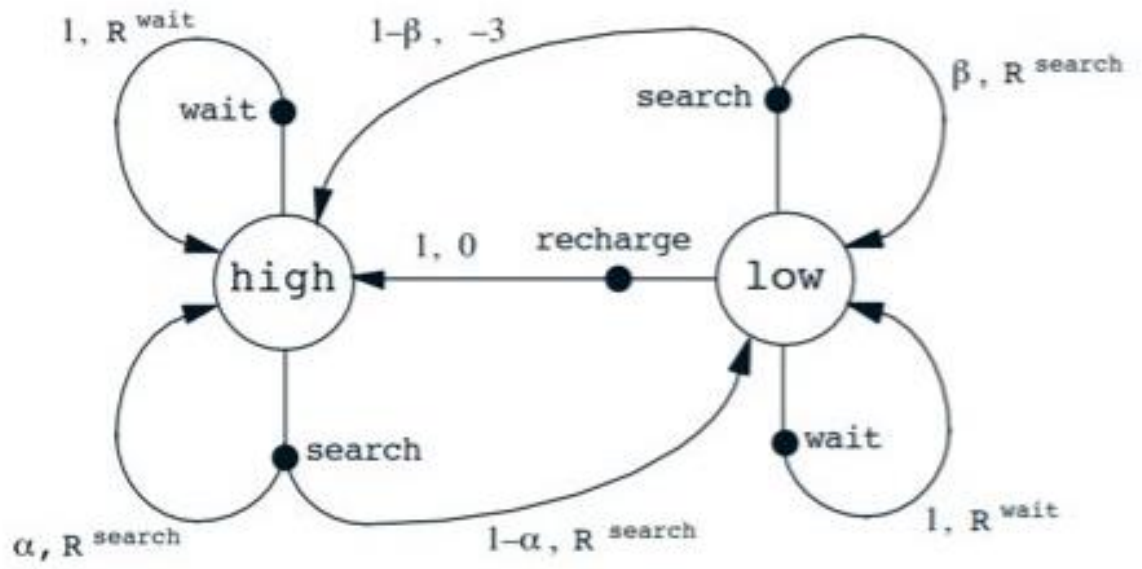


SOLUTION OF RECYCLING ROBOT



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.

SOLUTION OF RECYCLING ROBOT



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

BELLMAN OPTIMALITY EQUATIONS FOR RECYCLING ROBOT

- 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. Since there are only two states, the Bellman optimality equation consists of two equations. The equation for v∗(h) can be written as follows:

$$v_*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s', r} p(s', r|s, a)\left[r + \gamma v_*(s')\right]$$

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

BELLMAN OPTIMALITY EQUATIONS FOR RECYCLING ROBOT

	s	s'	a	$p(s'\|s,a)$	$r(s,a,s')$
	h	h	s	$1-\beta$	-3
	h	l	s	β	r_s
zero	h	h	w	0	r_w
	h	l	w	1	r_w
	l	h	re	1	0
zero	l	l	re	0	0

$$\therefore V_*(h) = \max \begin{cases} p(h|h,s)[r(h,s,h) + \gamma V_*(h)] + \\ p(l|h,s)[r(h,s,l) + \gamma V_*(l)], \\ p(l|h,w)[r(h,w,l) + \gamma V_*(l)], \\ p(h|h,re)[r(h,re,h) + \gamma V_*(h)] \end{cases}$$

$$= \max \begin{cases} (1-\beta)[-3 + \gamma V_*(h)] + \beta[r_s + \gamma V_*(l)], \\ 1[r_w + \gamma V_*(l)], \\ 1[0 + \gamma V_*(h)] \end{cases}$$

$$= \max \begin{cases} \beta r_s - 3(1-\beta) + \gamma[(1-\beta)V_*(h) + \beta V_*(l)], \\ r_w + \gamma V_*(l), \\ \gamma V_*(h) \end{cases}$$

BELLMAN OPTIMALITY EQUATIONS FOR RECYCLING ROBOT

$$v_*(s) = \max_{a \in \mathcal{A}(s)} \sum_{s',r} p(s', r|s, a)\left[r + \gamma v_*(s')\right]$$

Following the same procedure for v∗(l) yields the following 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 rs, rw, α, β, and γ, with $0 \leq \gamma < 1$, $0 \leq \alpha, \beta \leq 1$, there is exactly one pair of numbers, v∗(h) and v∗(l), that simultaneously satisfy these two nonlinear equations

THANK YOU!!!