



Reinforcement Learning : Learning from interaction



Learning to Control

- So far looked at two models of learning
 - Supervised: Classification, Regression, etc.
 - Unsupervised: Clustering, etc.

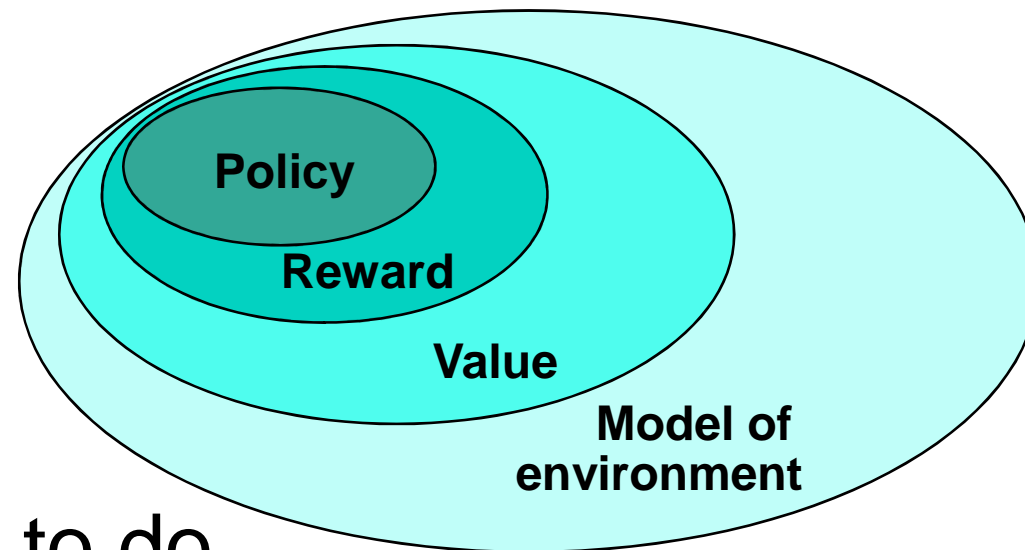
- How did you learn to cycle?
 - Neither of the above
 - Trial and error!
 - Falling down hurts!



Reinforcement learning

- A trial-and-error learning paradigm
 - Rewards and Punishments
- Not just an algorithm but a new paradigm in itself
- Learn about a system –
 - behavior
 - Control
 - from minimal feed back
- Inspired by behavioral psychology

Elements of RL



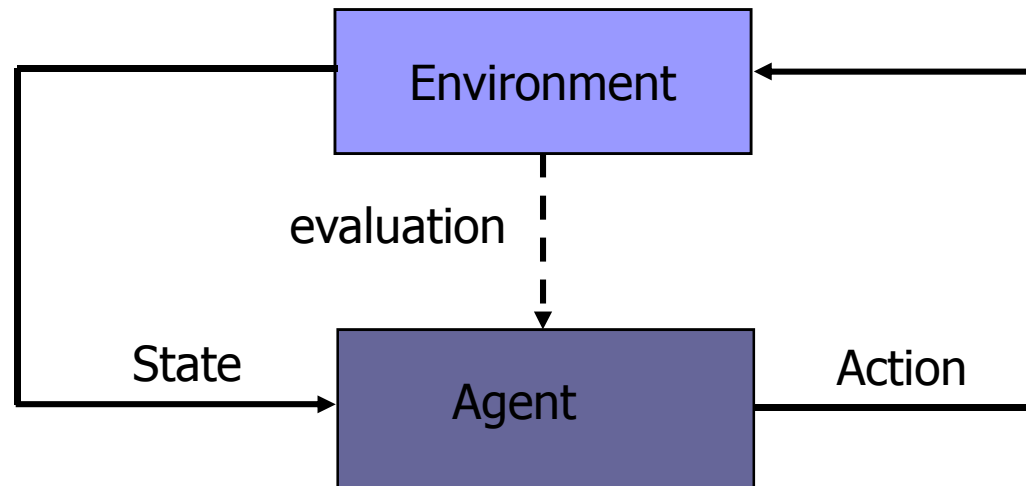
- **Policy**: what to do
- **Reward**: what is good
- **Value**: what is good because it *predicts* reward
- **Model**: what follows what



What is reward?

- The agents need to know that something good has happened when it wins and something bad when it loses. This kind of feed back is called ***reward or reinforcement***.
- The task of *reinforcement learning* is to use observed rewards to learn an optimal policy for the environment.

RL framework



- Learn from close interaction
- Stochastic environment
- Noisy delayed scalar evaluation
- Maximize a measure of long term performance



Model

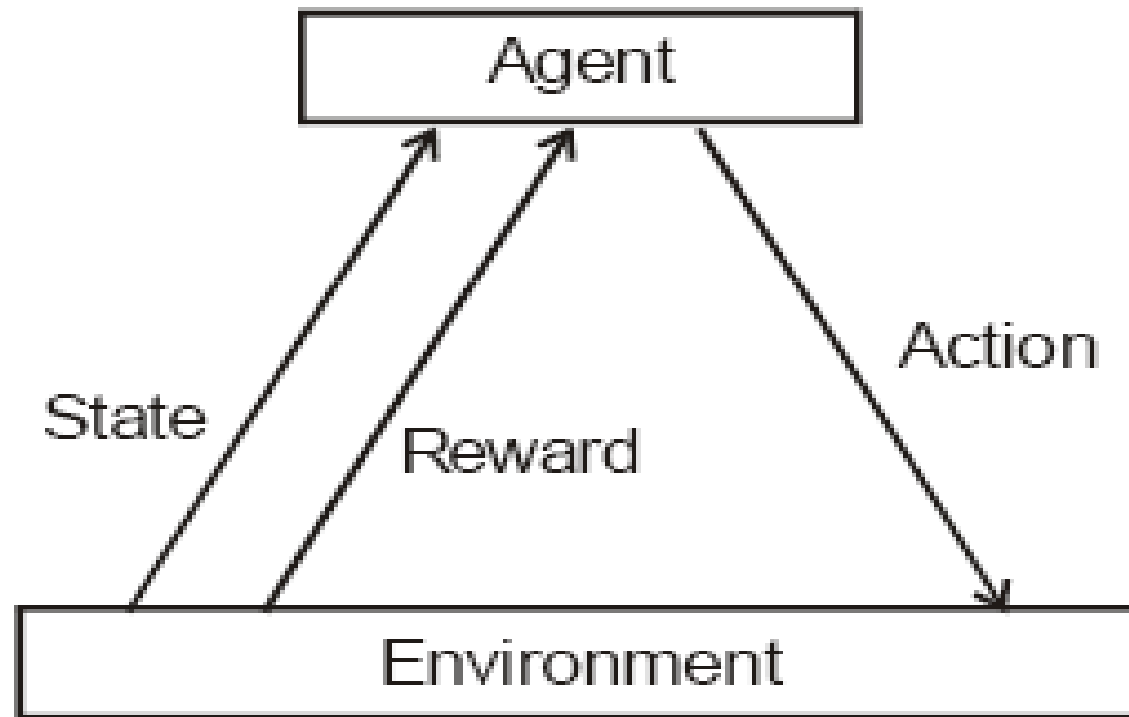
The basic reinforcement learning model consists of:

- a set of environment states S ;
- a set of actions A ; and
- a set of scalar “rewards” in R

Applications:

- robot control
- telecommunications
- games.

Model – Contd...



Model – Contd...



FIGURE 11.1 A robot perceives the current state of its environment through its sensors, and performs actions by moving its motors. The reinforcement learner (agent) within the robot tries to predict the next state and reward.

RL cycle

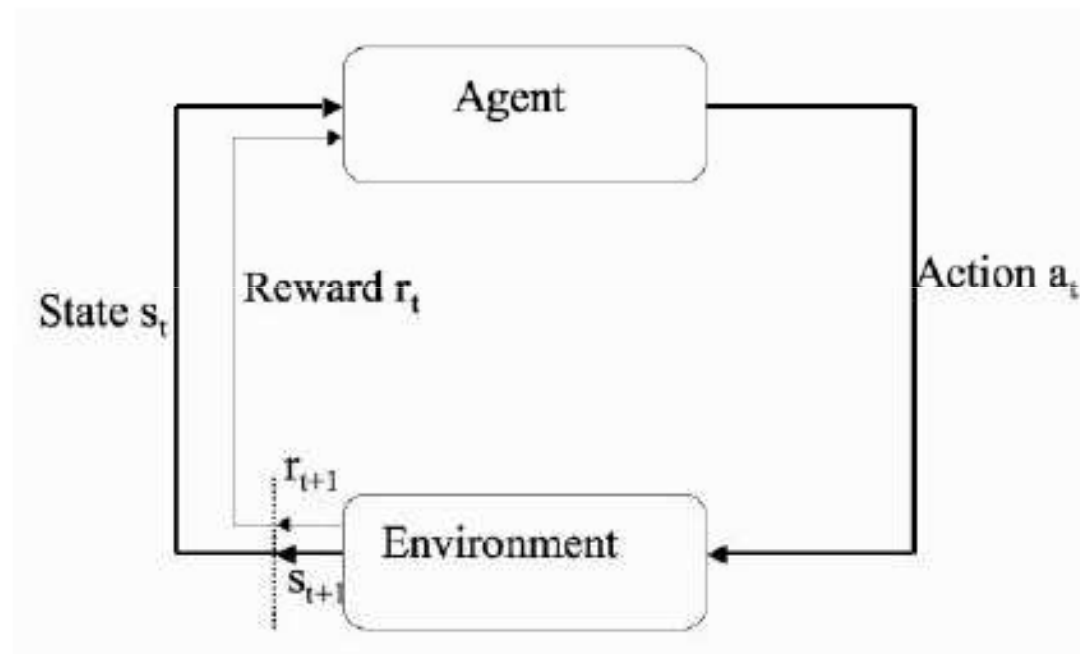


FIGURE 11.2 The reinforcement learning cycle: the learning agent performs action a_t in state s_t and receives reward r_{t+1} from the environment, ending up in state s_{t+1} .



Types of agents

- ***Utility based agents:*** It learns utility function on states, and uses it to select actions that maximize the expected outcome utility.
- ***Q-learning agent:*** It learns an action-value function, or Q-function, giving the expected utility of taking a given action in a given state.
- ***Reflex agent:*** It learns a policy that maps directly from states to action.



Difference between learning types

	Supervised	Unsupervised	Reinforcement
Training data	Data and correct output	Data	States, actions, and rewards
Learning target	Data-output relationship	Patterns in data	Policy
Evaluation	Statistics	Fitness	Reward value
Typical application	Classifiers	Clustering	Controllers



Types of reinforcement learning

- Passive reinforcement learning - the agent's policy is fixed and the task is to learn the utilities of states (or state-action pairs), or model for environment.
- Active reinforcement learning - the agent must decide what actions to take. The main issue in active learning is exploration.

EXAMPLE: GETTING LOST

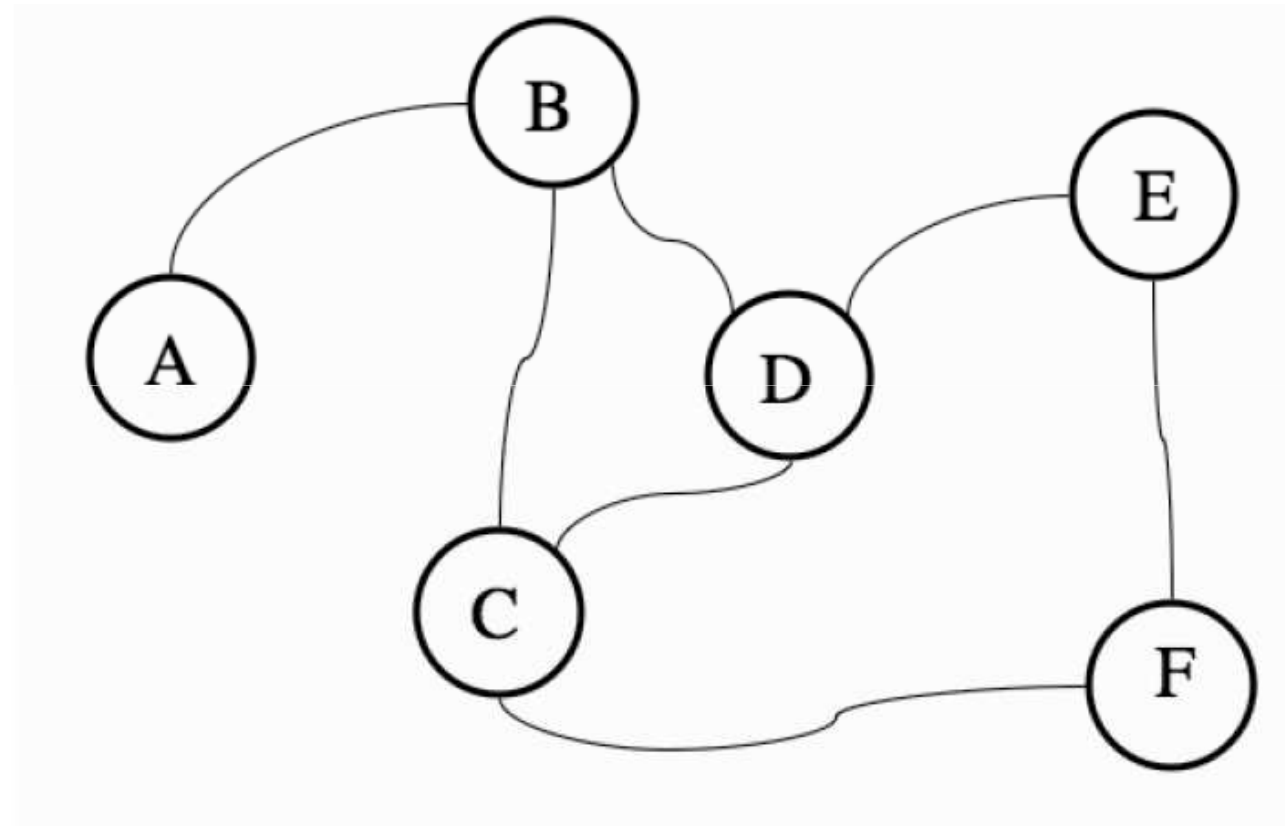


FIGURE 11.3 The old town that you find yourself lost in.



Reward Matrix

Current State	Next State					
	A	B	C	D	E	F
A	-5	0	-	-	-	-
B	0	-5	0	0	-	-
C	-	0	-5	0	-	100
D	-	0	0	-5	0	-
E	-	-	-	0	-5	100
F	-	-	0	-	0	-

State – action space

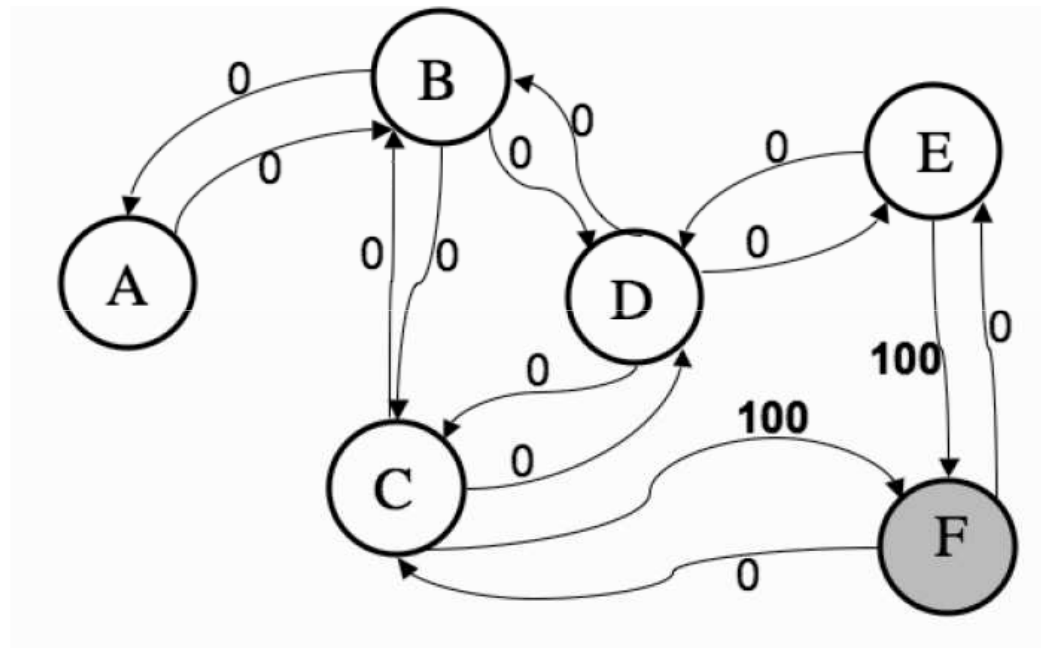


FIGURE 11.4 The *state diagram* if you are correct and the backpacker's is in square (state) F. The connections from each state back into itself (meaning that you don't move) are not shown, to avoid the figure getting too complicated. They are each worth -5 (except for staying in state F, which means that you are in the backpacker's).



Discounting

$$0 \leq \gamma \leq 1$$

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots + \gamma^{k-1} r_k + \dots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}.$$



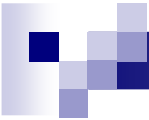
Action Selection

- Greedy: Pick the action that has the highest value of $Q_{s,t}(a)$, so *always choose to exploit* your current knowledge.
- Epsilon-greedy: selection finds better solutions over time than the pure greedy algorithm, since it can explore and find better solutions.
- Soft-max: with temperature as additional parameter, to select the next state.



What is MDP?

- Sequential decision problems for a fully observable environment with a Markovian transition model and additive rewards is called MDP.
- Sequential decision problems in uncertain domain – MDP, defined by a transition model specifying the probabilistic outcomes of actions and a reward function specifying the reward in each state.
- Utility of a state sequence- sum of all the rewards over the sequence, possibly discounted over time.



MDP – contd..

- Solution of a MDP – policy, associates a decision with every state that the agent might reach.
- Optimal policy – maximizes the utility of the state sequences encountered when it is executed.
- Utility of a state – expected utility of the state sequences encountered when an optimal policy is executed, starting in that state.



Markov decision process

- Sequential decision problem
- A finite set of states $s \in \mathcal{S}$
- A finite set of actions $a \in \mathcal{A}$
- Transition model $T(s'|s, a)$ or $T(s, a, s')$
- Reward function $r(s, a)$
- Observe and act in discrete time steps.



Markov property

In general, next state depends on all the earlier states.

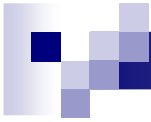
$$s_{t+1} = f(s_t, a_t, s_{t-1}, a_{t-1}, \dots, s_1, a_1, s_0, a_0)$$

Markov property

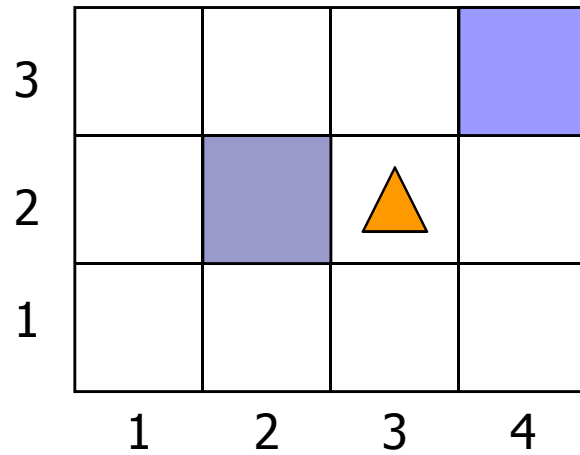
Next state depends only on the present state and present action.

$$s_{t+1} = f(s_t, a_t)$$

The present state s_t has the complete information about the past history.



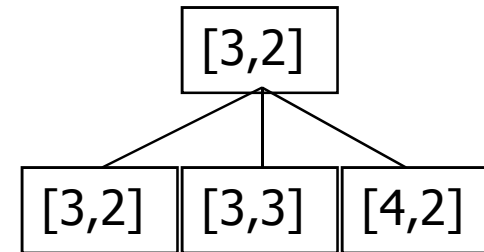
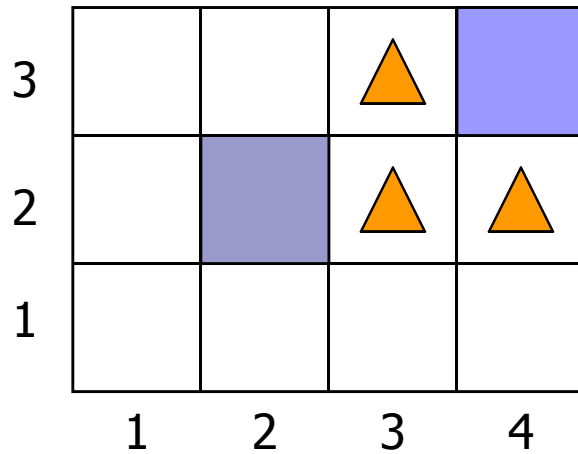
Sequence of Actions



[3,2]

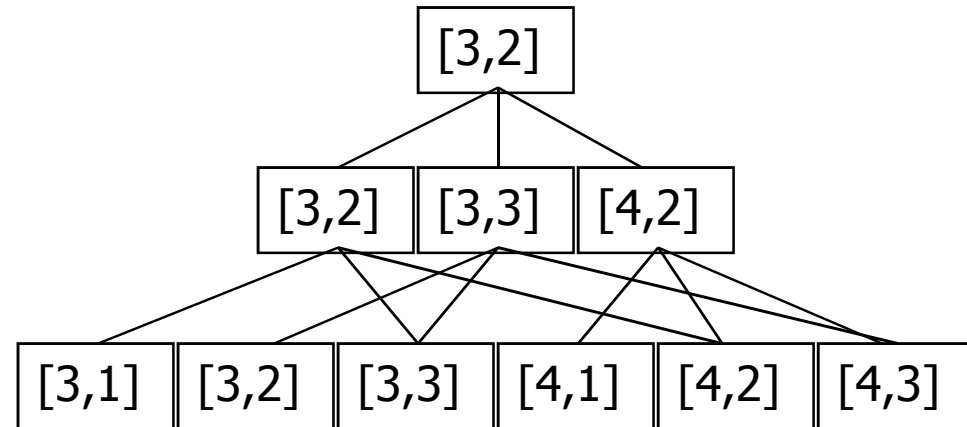
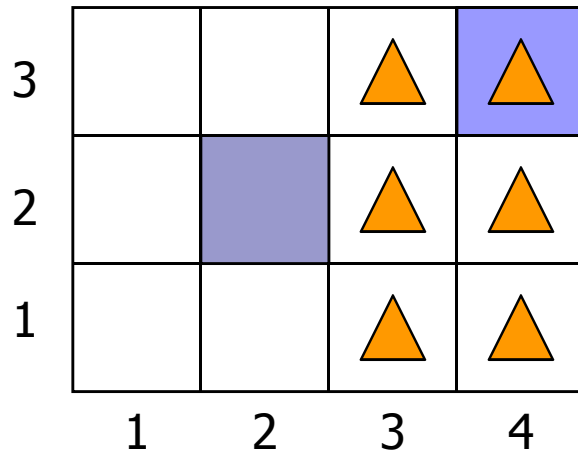
- Planned sequence of actions: (U, R)

Sequence of Actions



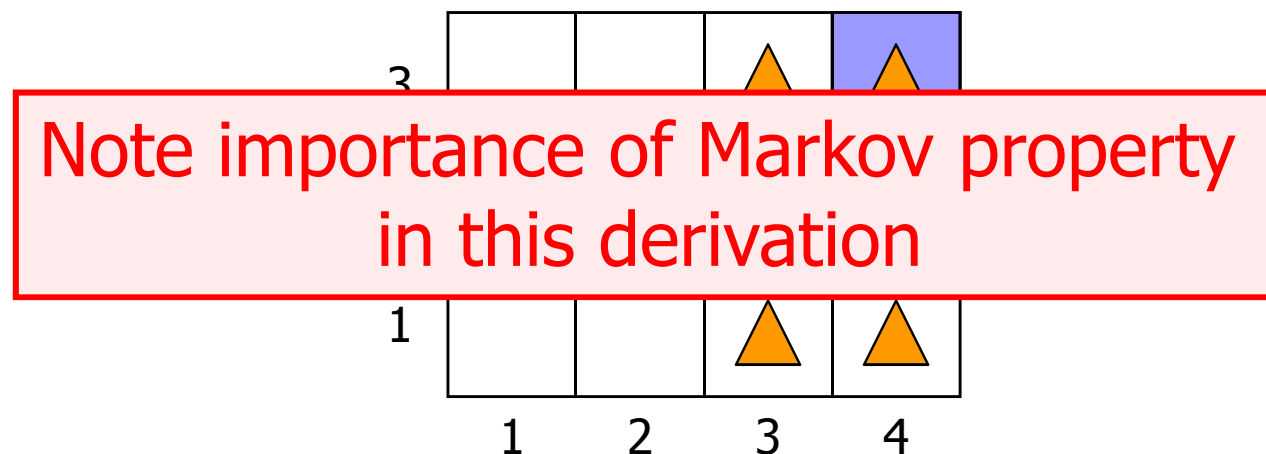
- Planned sequence of actions: (U, R)
- U is executed

Histories



- Planned sequence of actions: (U, R)
- U has been executed
- R is executed
- There are 9 possible sequences of states
 - called histories
- find the no. of possible final states for the robot!

Probability of Reaching the Goal



- $P([4,3] \mid (U,R).[3,2]) =$

$$P([4,3] \mid R.[3,3]) \times P([3,3] \mid U.[3,2])$$

$$+ P([4,3] \mid R.[4,2]) \times P([4,2] \mid U.[3,2])$$
- $P([4,3] \mid R.[3,3]) = 0.8$ • $P([3,3] \mid U.[3,2]) = 0.8$
- $P([4,3] \mid R.[4,2]) = 0.1$ • $P([4,2] \mid U.[3,2]) = 0.1$
- $P([4,3] \mid (U,R).[3,2]) = 0.65$



What action to take in a state?

Policy

- In each state, what action should the agent take? **Policy**.
- For **any** state s , policy $\pi(s)$ gives the action a .
- **Optimal policy**: gives the **optimal action** in every state. Leads to the optimal reward, best outcome.

What is the value of a state?

Value function

- Value of a state s_t according to policy π : sum of the rewards

$$V^\pi(s_t) = r(s_t, \pi(s_t)) + r(s_{t+1}, \pi(s_{t+1})) + r(s_{t+2}, \pi(s_{t+2})) + \dots$$

Expected value: expected sum of rewards

$$V^\pi(s_t) = E[r(s_t, \pi(s_t)) + r(s_{t+1}, \pi(s_{t+1})) + r(s_{t+2}, \pi(s_{t+2})) + \dots]$$

- Discount the future rewards:

Expected discounted reward

$$V^\pi(s_t) = E[r(s_t, \pi(s_t)) + \gamma r(s_{t+1}, \pi(s_{t+1})) + \gamma^2 r(s'_{t+2}, \pi(s_{t+2})) + \dots]$$



Optimal policy

- Find out an optimal policy π^* .
- At any state s , follow the optimal policy. The state s has the maximum value (expected discounted reward) $V^*(s)$.

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$



Calculate optimal value function and optimal policy

- Value of state according to a policy π

$$V^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s'} T(s, a, s') V^\pi(s')$$

Can be used for evaluating policy = value of states according to the policy.

- Value of state according to optimal policy π^*

$$V^*(s) = \max_a [r(s, a) + \gamma \sum_{s'} T(s, a, s') V^*(s')]$$

Can be used for finding out optimal value function.

- Optimal policy

$$\pi^*(s) = \arg \max_a [r(s, a) + \gamma \sum_{s'} T(s, a, s') V^*(s')]$$



Value iteration

- To calculate an optimal policy
- Idea – calculate the utility of each state and then use the state utilities to select an optimal action in each state.



Value iteration

Algorithm: Value Iteration

Input: MDP with S, T, r, γ

Output: π^*

Initialize $V(s)$ arbitrarily, e.g. $V(s) = 0$, for all $s \in S$

repeat

foreach $s \in S$ **do**

$v \leftarrow V^*(s)$

$V^*(s) \leftarrow \max_a [r(s, a) + \gamma \sum_{s'} T(s, a, s') V^*(s')]$

$\pi^*(s) \leftarrow \arg \max_a [r(s, a) + \gamma \sum_{s'} T(s, a, s') V^*(s')]$

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

end

until $\Delta < \epsilon$ policy is good enough



Policy iteration

- Policy evaluation – given a policy calculate the utility of each state when policy to be executed
- Policy improvement – calculate the new policy using forward step



Policy iteration

Algorithm: Policy Iteration (MDP, ϵ)

Input: MDP with S, T, r

Output: π

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

repeat

▷ **Policy evaluation**

$V(s) \leftarrow$ Solve linear equations

$$V^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s'} T(s, \pi(s), s') V^\pi(s')$$

▷ **Policy improvement**

foreach $s \in S$ **do**

$$\pi(s) \leftarrow \arg \max_a [r(s, a) + \gamma \sum_{s'} T(s, a, s') V^\pi(s')]$$

end

until π is unchanged