



NANYANG
TECHNOLOGICAL
UNIVERSITY
SINGAPORE

AI6101

Introduction to AI and AI Ethics

Markov Decision Process

Assoc Prof Bo AN

www.ntu.edu.sg/home/boan

Email: boan@ntu.edu.sg

Office: N4-02b-55





Lesson Outline

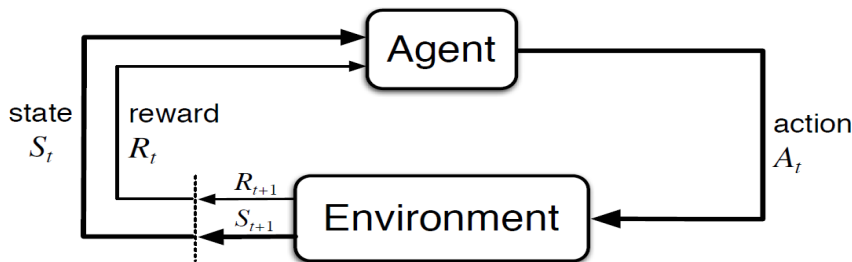
- Introduction
- Markov Decision Process
- Two methods for solving MDP
 - Value iteration
 - Policy iteration
- Temporal difference learning



Introduction

- We consider a framework for decision making under uncertainty
- Markov decision processes (MDPs) and their extensions provide an extremely general way to think about how we can act optimally under uncertainty
- For many medium-sized problems, we can use the techniques from this lecture to compute an optimal decision policy
- For large-scale problems, approximate techniques are often needed (more on these in later lectures), but the paradigm often forms the basis for these approximate methods

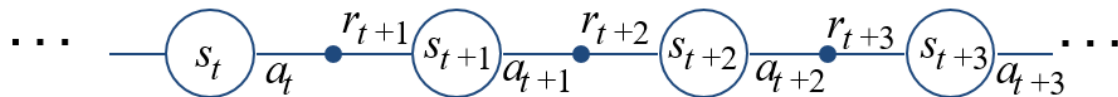
The Agent-Environment Interface



Agent and environment interact at discrete time steps: $t = 0, 1, 2, \dots$

Agent:

1. observes state at step t : $s_t \in \mathcal{S}$
2. Produces action at step t : $a_t \in A(s_t)$
3. Gets resulting reward: $r_{t+1} \in \mathfrak{R}$ and resulting next state: $s_{t+1} \in \mathcal{S}$





Making Complex Decisions

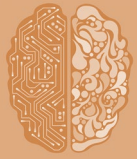
- Make a sequence of decisions
 - Agent's utility depends on a sequence of decisions
 - Sequential Decision Making
- Markov Property
 - Transition properties depend only on the current state, not on previous history (how that state was reached)
 - Markov Decision Processes



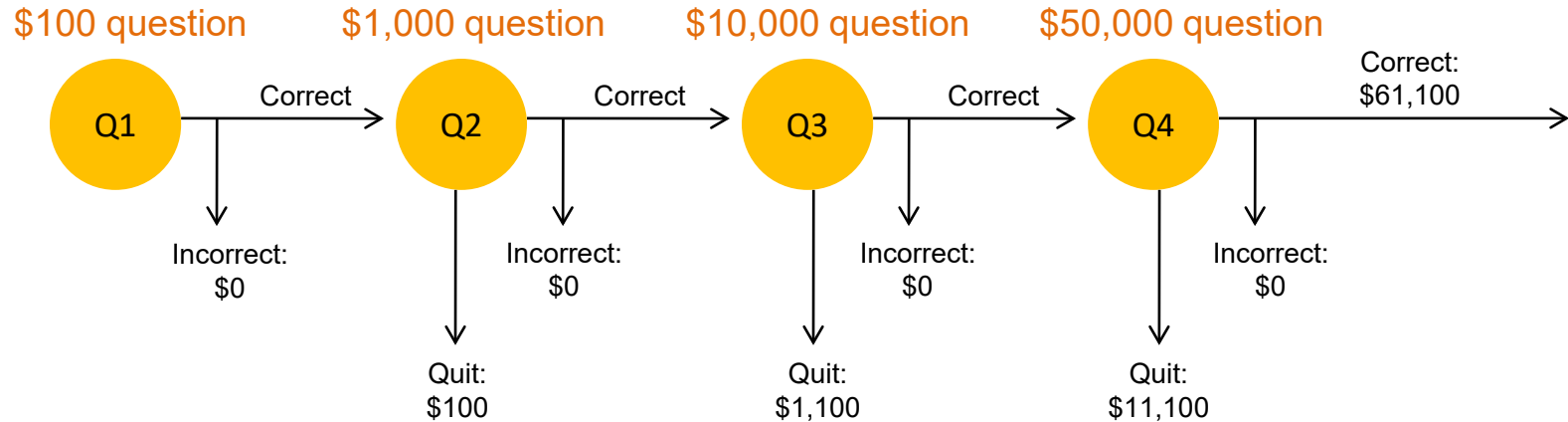
Markov Decision Processes

- Components:
 - **Markov States** s , beginning with initial state s_0
 - **Actions** a
 - Each state s has actions $A(s)$ available from it
 - **Transition model** $P(s' | s, a)$
 - *assumption*: the probability of going to s' from s depends only on s and a and not on any other past actions or states
 - **Reward function** $R(s)$
- **Policy** $\pi(s)$: the action that an agent takes in any given state
 - The “solution” to an MDP

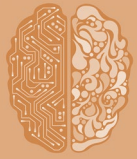
Game Show



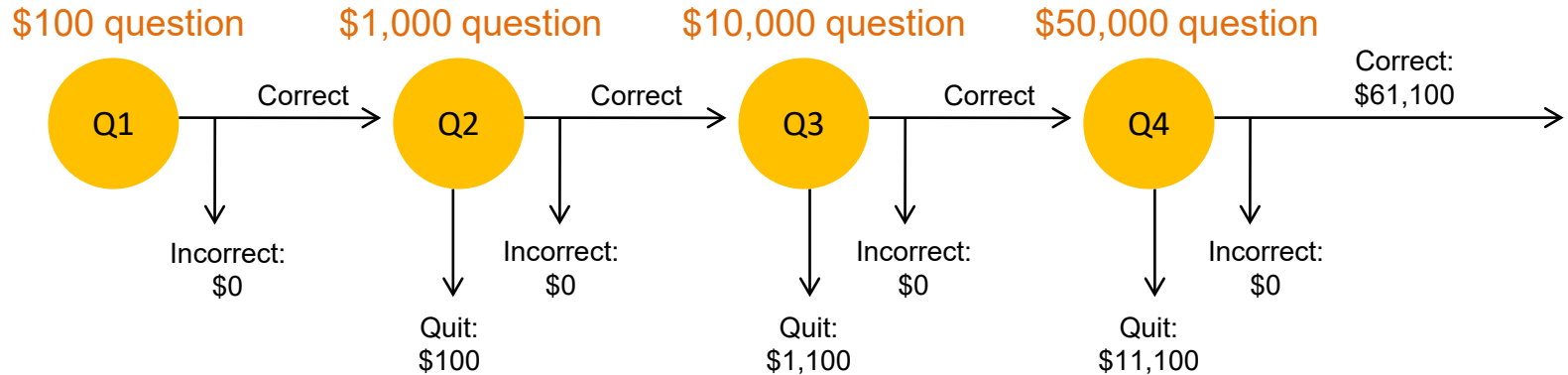
- A series of questions with increasing level of difficulty and increasing payoff
- Decision: at each step, take your earnings and quit, or go for the next question
 - If you answer wrong, you lose everything



Game Show



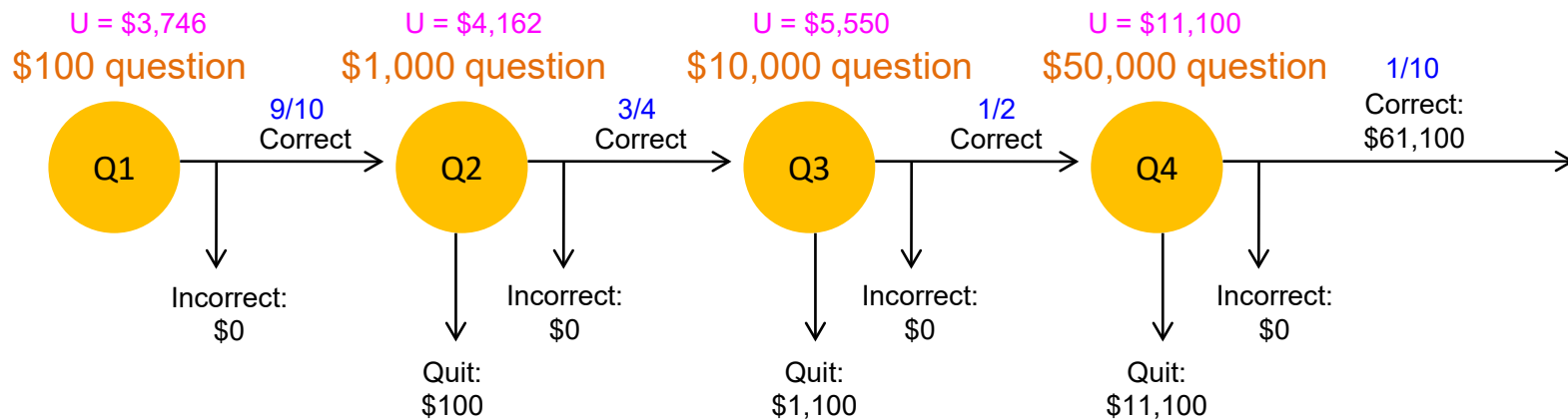
- Consider \$50,000 question
 - Probability of guessing correctly: 1/10
 - Quit or go for the question?
- What is the expected payoff for continuing?
 $0.1 * 61,100 + 0.9 * 0 = 6,110$
- What is the optimal decision?



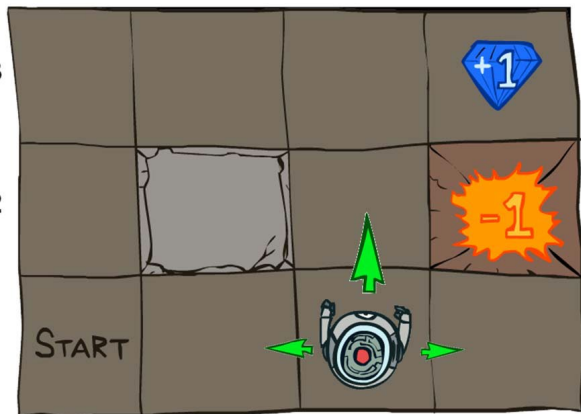
Game Show



- What should we do in Q3?
 - Payoff for quitting: \$1,100
 - Payoff for continuing: $0.5 * \$11,100 = \$5,550$
- What about Q2?
 - \$100 for quitting vs. \$4,162 for continuing
- What about Q1?

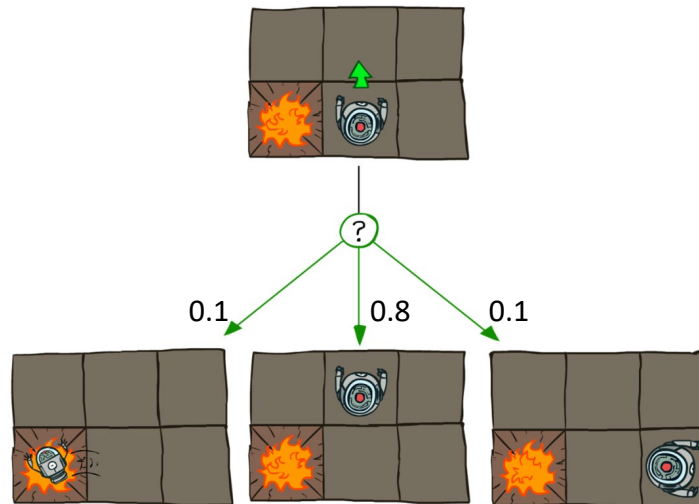


Grid World

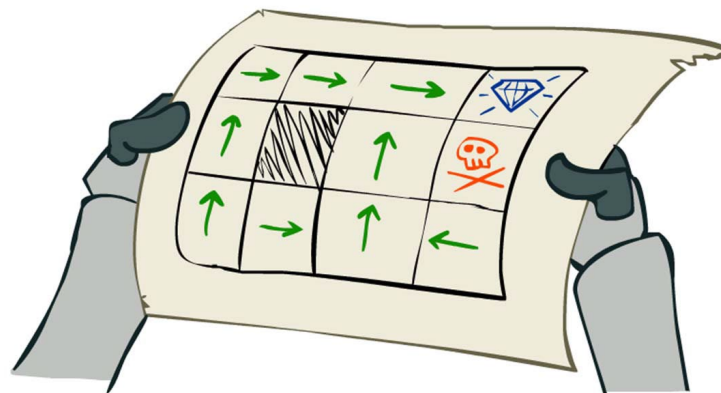
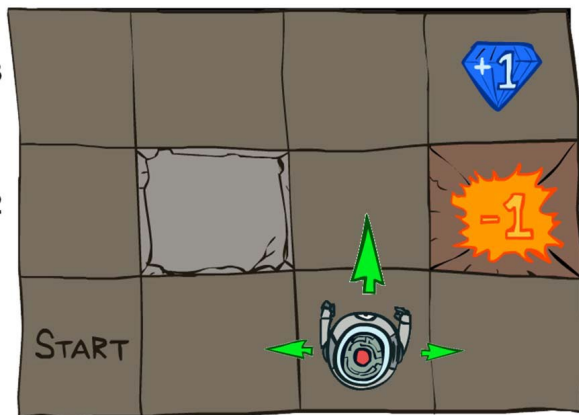


$R(s) = -0.04$ for every
non-terminal state

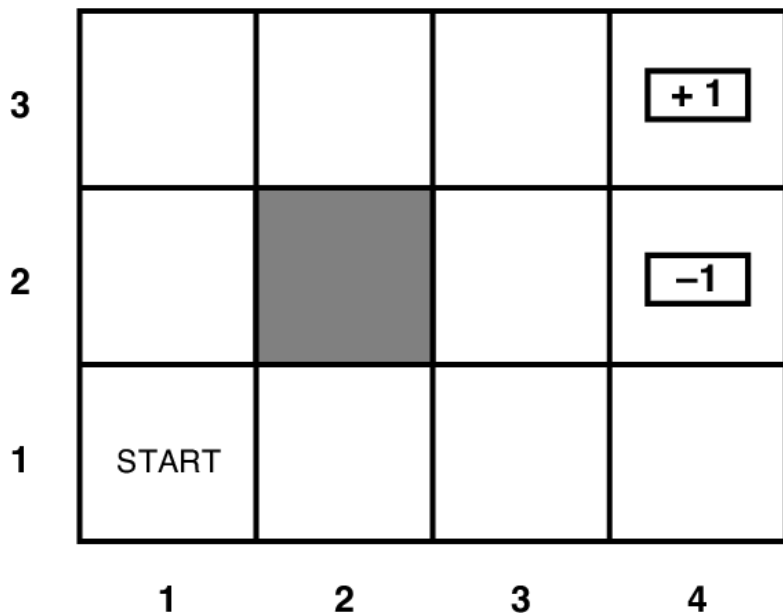
Transition model:



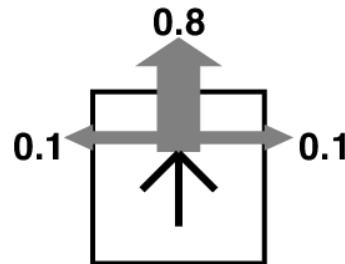
Goal: Policy



Grid World

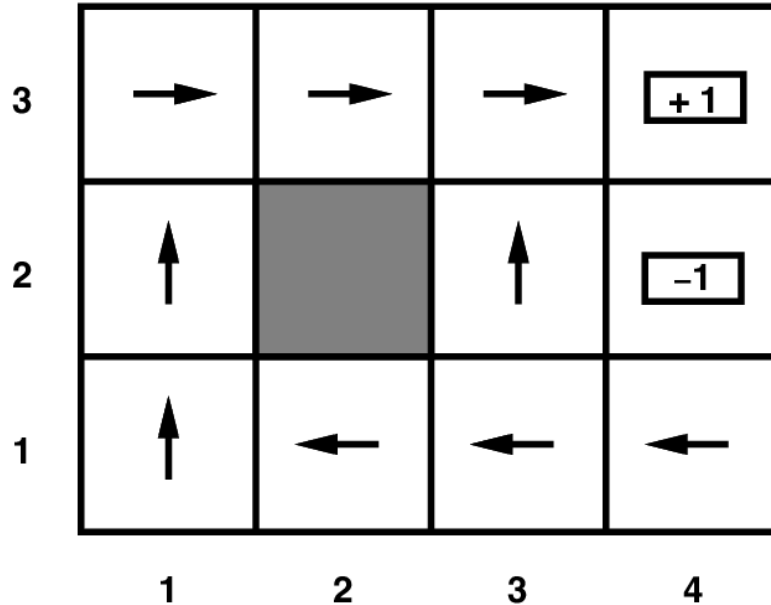


Transition model:



$R(s) = -0.04$ for
every non-terminal
state

Grid World



Optimal policy when
 $R(s) = -0.04$ for every
non-terminal state

Solving MDPs



- MDP components:
 - **States** s
 - **Actions** a
 - **Transition model** $P(s' | s, a)$
 - **Reward function** $R(s)$
- The solution:
 - **Policy** $\pi(s)$ mapping from states to actions
 - How to find the optimal policy?



Maximising Expected Utility

- The optimal policy should maximise the *expected utility* over all possible state sequences produced by following that policy:

$$\sum_{\substack{\text{state sequences} \\ \text{starting from } s_0}} P(\text{sequence})U(\text{sequence})$$

- How to define the utility of a state sequence?
 - Sum of rewards of individual states
 - Problem: infinite state sequences
 - If finite, LP can be applied



Utilities of State Sequences

- Normally, we would define the utility of a state sequence as the sum of the rewards of the individual states
- **Problem:** infinite state sequences
- **Solution:** *discount* the individual state rewards by a factor γ between 0 and 1:

$$\begin{aligned} U([s_0, s_1, s_2, \dots]) &= R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots \\ &= \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \frac{R_{\max}}{1 - \gamma} \quad (0 < \gamma < 1) \end{aligned}$$

- Sooner rewards count more than later rewards
- Makes sure the total utility stays bounded
- Helps algorithms converge

Utilities of States

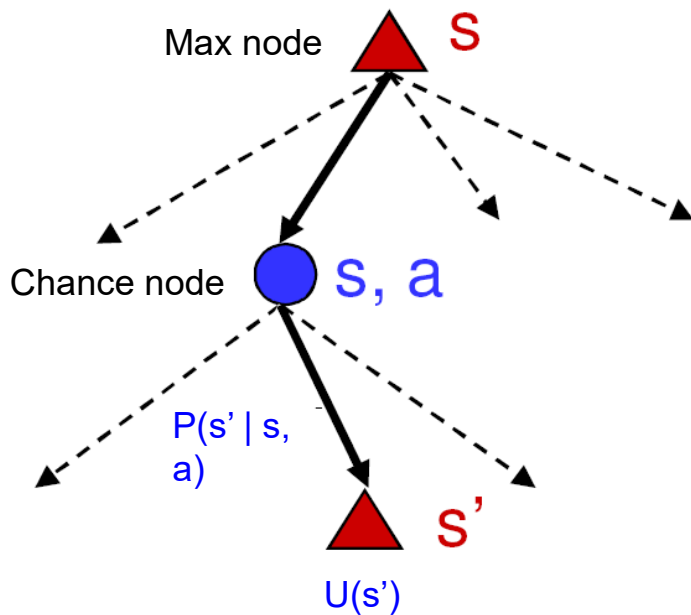


- Expected utility obtained by policy π starting in state s :

$$U^\pi(s) = \sum_{\substack{\text{state sequences} \\ \text{starting from } s}} P(\text{sequence}) U(\text{sequence})$$

- The “true” utility of a state, denoted $U(s)$, is the expected sum of discounted rewards if the agent executes an *optimal* policy starting in state s
- Reminiscent of minimax values of states...

Finding the Utilities of States



- What is the expected utility of taking action a in state s ?

$$\sum_{s'} P(s'|s, a)U(s')$$

- How do we choose the optimal action?

$$\pi^*(s) = \arg \max_{a \in A(s)} \sum_{s'} P(s'|s, a)U(s')$$

- What is the recursive expression for $U(s)$ in terms of the utilities of its successor states?

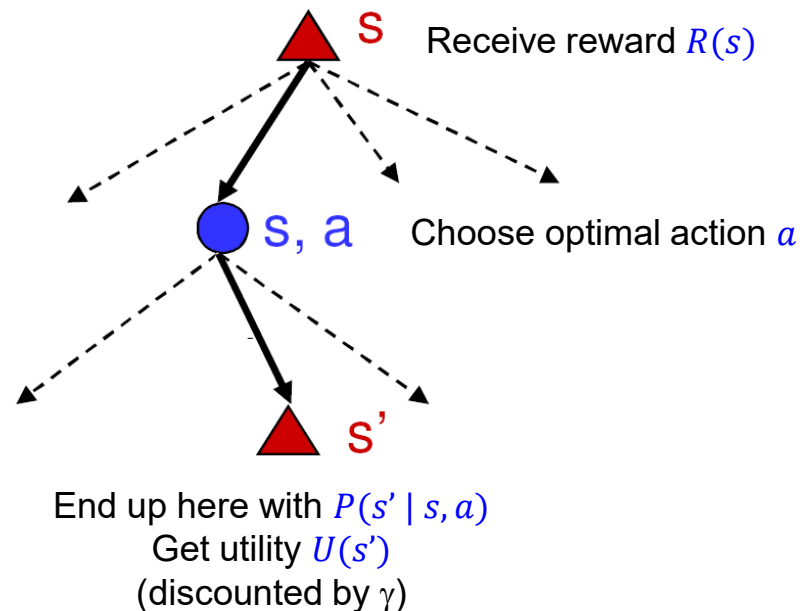
$$U(s) = R(s) + \gamma \max_a \sum_{s'} P(s'|s, a)U(s')$$

The Bellman Equation



- Recursive relationship between the utilities of successive states:

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' | s, a) U(s')$$



The Bellman Equation



- Recursive relationship between the utilities of successive states:

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s, a) U(s')$$

- For N states, we get N equations in N unknowns
 - Solving them solves the MDP
 - We could try to solve them through expectimax search, but that would run into trouble with infinite sequences
 - Instead, we solve them algebraically
 - Two methods: **value iteration** and **policy iteration**



Method 1: Value Iteration

- Start out with every $U(s) = 0$
- Iterate until convergence
 - During the i th iteration, update the utility of each state according to this rule:

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s, a) U_i(s')$$

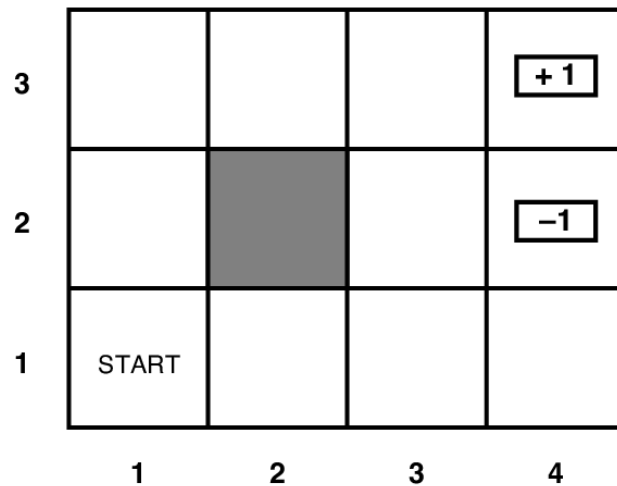
- In the limit of infinitely many iterations, guaranteed to find the correct utility values
 - In practice, don't need an infinite number of iterations...



Value Iteration

- What effect does the update have?

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s, a) U_i(s')$$





Method 2: Policy Iteration

- Start with some initial policy π_0 and alternate between the following steps:
 - **Policy evaluation:** calculate $U^{\pi_i}(s)$ for every state s
 - **Policy improvement:** calculate a new policy π_{i+1} based on the updated utilities

$$\pi^{i+1}(s) = \arg \max_{a \in A(s)} \sum_{s'} P(s'|s, a) U^{\pi_i}(s')$$



TD(Temporal difference) Prediction

Policy Evaluation (the prediction problem):

for a given policy p , compute the state-value function V^π

The simplest TD method, TD(0):

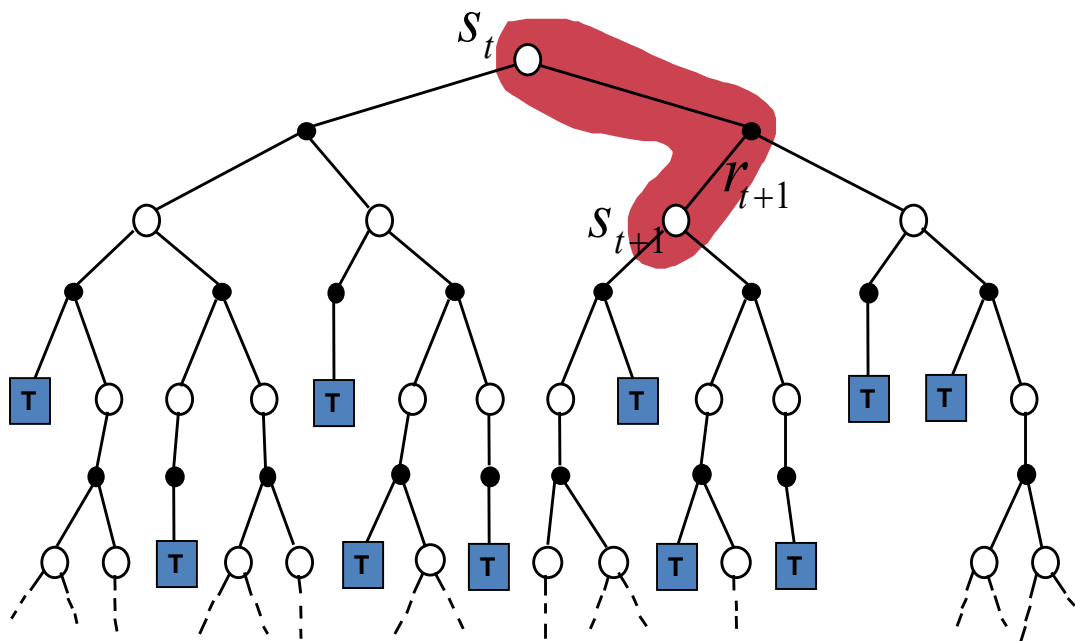
$$V(s_t) \leftarrow V(s_t) + \alpha \left[\underbrace{r_{t+1} + \gamma V(s_{t+1})}_{\text{target}} - V(s_t) \right]$$

target: an estimate of the return



Simplest TD Method

$$V(s_t) \leftarrow V(s_t) + \alpha [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$





Advantages of TD Learning

- TD methods do not require a model of the environment, only experience
- TD methods can be fully incremental
 - You can learn **before** knowing the final outcome
 - Less memory
 - Less peak computation
 - You can learn **without** the final outcome
 - From incomplete sequences

Further Reading

[AAAI'18: http://www.ntu.edu.sg/home/boan/papers/AAAI18_Malmo.pdf]



We Won 2017 Microsoft Collaborative AI Challenge

- Collaborative AI
 - *How can AI agents learn to recognise someone's intent (that is, what they are trying to achieve)?*
 - *How can AI agents learn what behaviours are helpful when working toward a common goal?*
 - *How can they coordinate or communicate with another agent to agree on a shared strategy for problem-solving?*

Further Reading

[AAAI'18: http://www.ntu.edu.sg/home/boan/papers/AAAI18_Malmo.pdf]



- Microsoft Malmo Collaborative AI Challenge
 - *Collaborative mini-game, based on an extension “stag hunt”*
 - *Uncertainty of pig movement*
 - *Unknown type of the other agent*
 - *Detection noise (frequency 25%)*
- Our team HogRider won the challenge (out of more than 80 teams from 26 countries)
 - *learning + game theoretic reasoning + sequential decision making + optimisation*

