



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

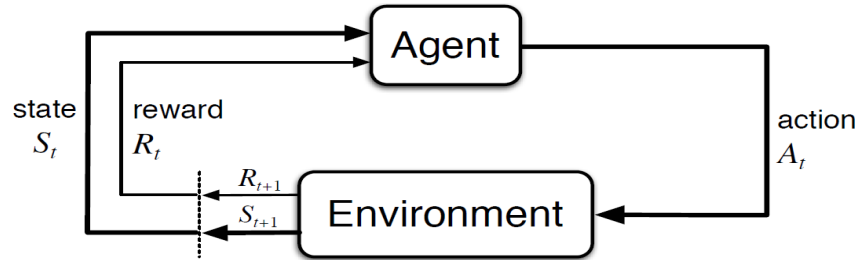
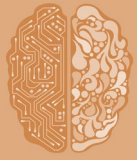
- Introduction
- Markov Decision Process
- Two methods for solving MDP
 - Value iteration
 - Policy iteration



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

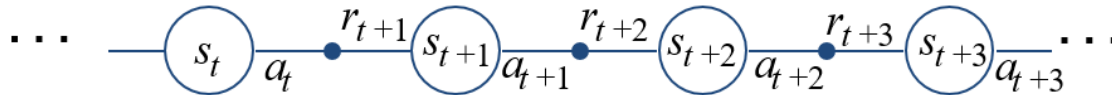
The Agent-Environment Interface



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} and the next state: $s_{t+1} \in \mathcal{S}$





Making Complex Decisions

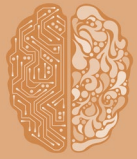
- Make a sequence of decisions
 - Agent's utility depends on a sequence of decisions
 - Sequential Decision Making
- Markov Property
 - Transition properties depend only on the current state, not on previous history (how that state was reached)
 - Markov Decision Processes



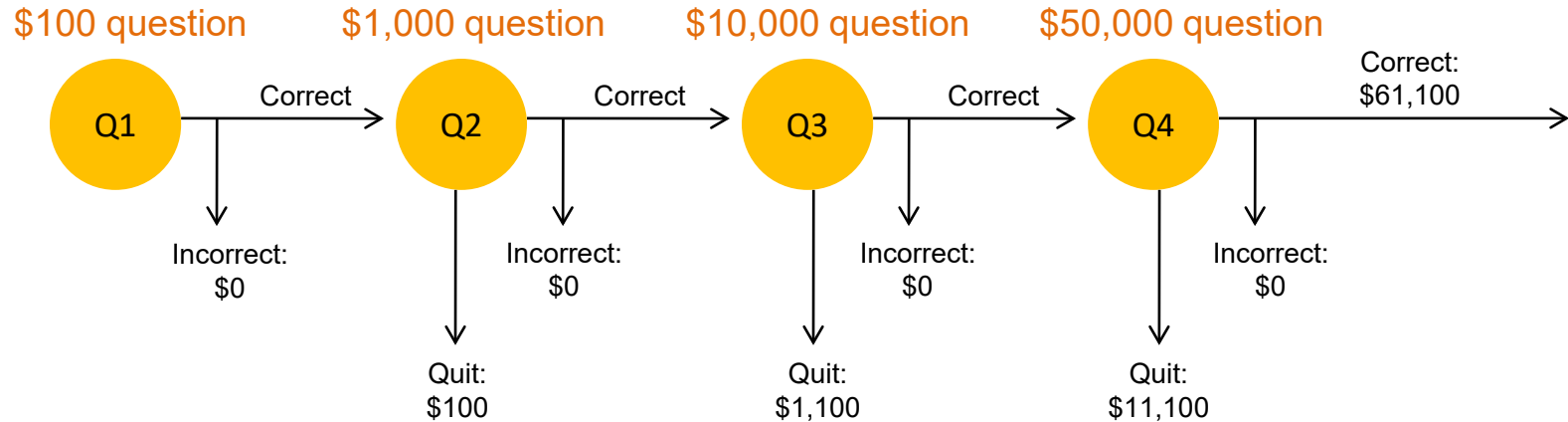
Markov Decision Processes

- Formulate the agent-environment interaction as an MDP
- 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** $R(s)$, or $r(s)$
- **Policy** $\pi(s)$: the action that an agent takes in any given state
 - The “solution” to an MDP

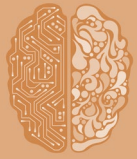
Game Show



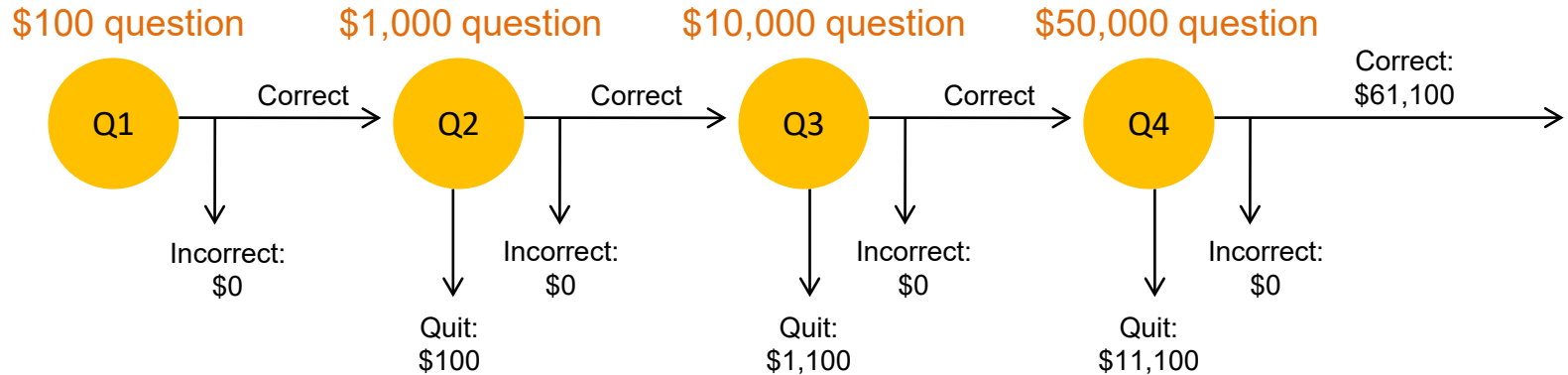
- A series of questions with increasing level of difficulty and increasing payoff
- Decision: at each step, take your earnings and quit, or go for the next question
 - If you answer wrong, you lose everything



Game Show



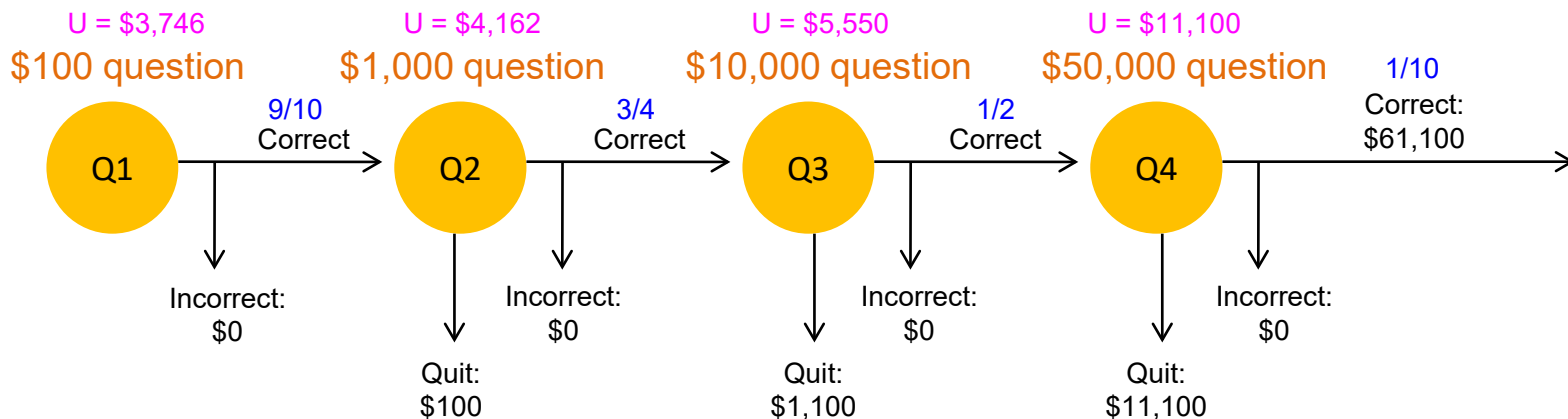
- Consider \$50,000 question
 - Probability of guessing correctly: 1/10
 - Quit or go for the question?
- What is the expected payoff for continuing?
 $0.1 * 61,100 + 0.9 * 0 = 6,110$
- What is the optimal decision?



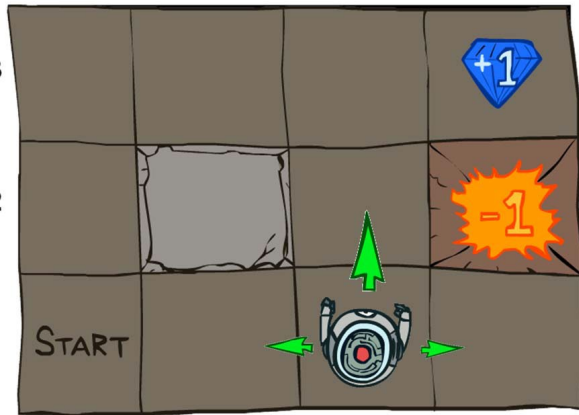
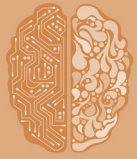
Game Show



- 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?

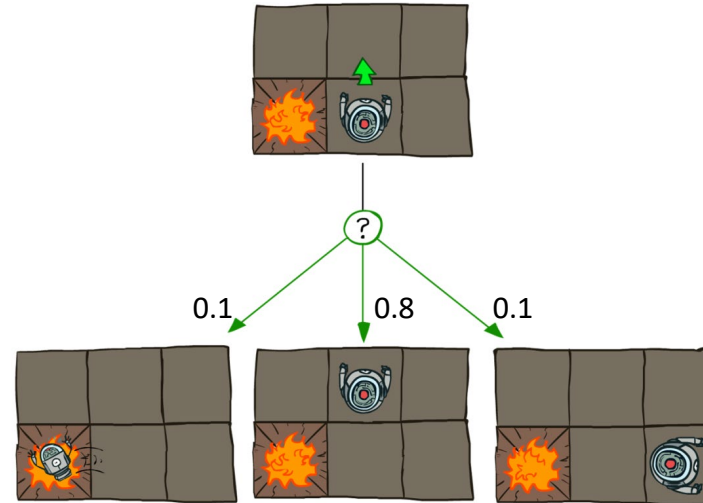


Grid World

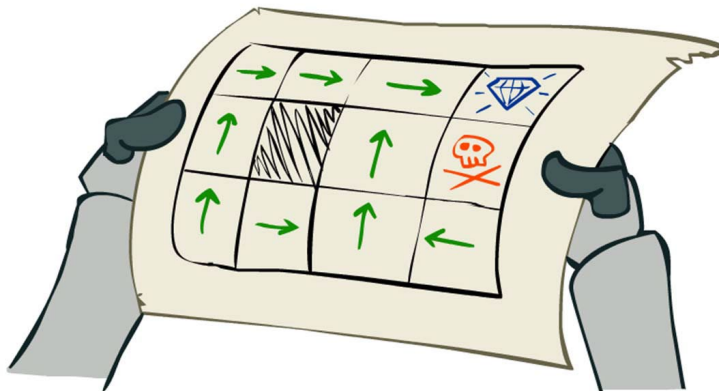
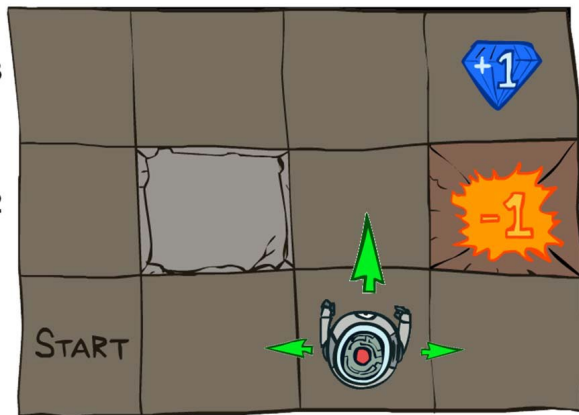


$R(s) = -0.04$ for every
non-terminal state

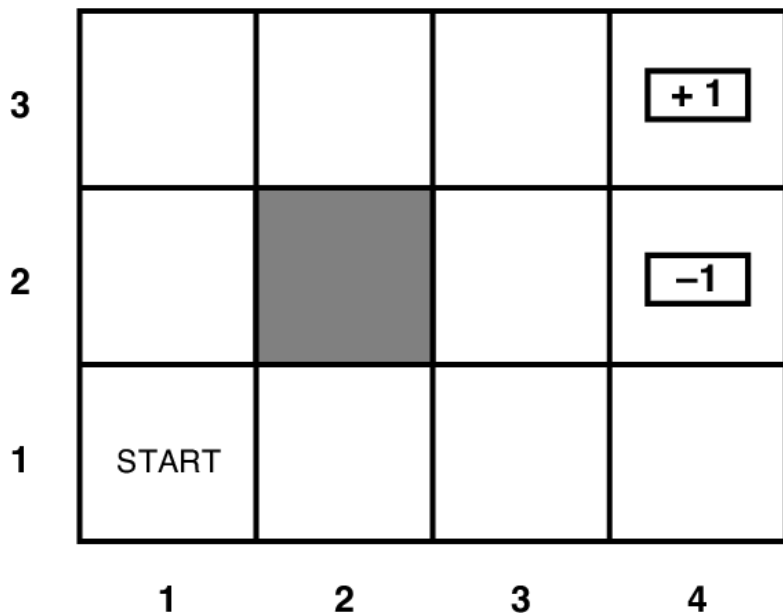
Transition model:



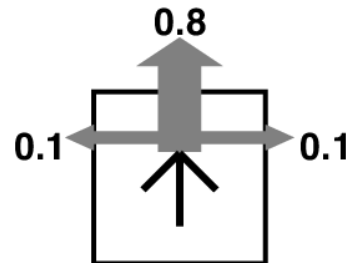
Goal: Policy



Grid World

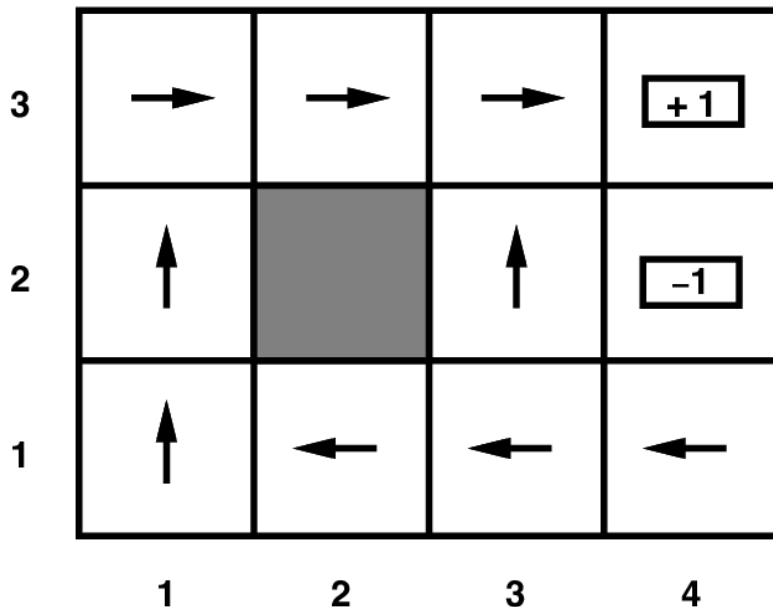


Transition model:



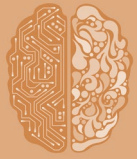
$R(s) = -0.04$ for every non-terminal state

Grid World



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

Atari Video Games



Execute actions
accumulated rewards



to get the maximum

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?

Maximizing Accumulated Rewards



- The optimal policy should maximise the accumulated *rewards* over given a trajectories like $\tau = \langle S_1, A_1, R_1, \dots, S_T, A_T, R_T \rangle$ under some policies:

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \dots \gamma^K R_{t+K} = \sum_{k=0}^K \gamma^k R_{t+k}$$

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



Accumulated Rewards

- Normally, we would define the *accumulated rewards* of trajectories as the discounted sum of the rewards
- **Problem:** infinite time horizon
- **Solution:** *discount* the individual rewards by a factor γ between 0 and 1:

$$\begin{aligned} G_t &= R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \dots \\ &= \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \leq \frac{R_{max}}{1-\gamma} \quad (0 < \gamma < 1) \end{aligned}$$

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



Value Function

- The “true” value of a state, denoted $V(s)$, is the expected sum of discounted rewards if the agent executes an *optimal* policy starting in state s

$$V^\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$$

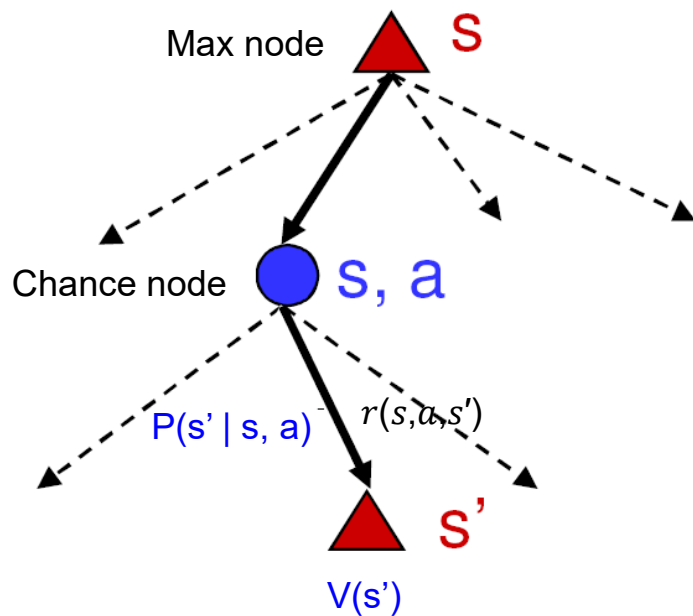
- Similarly, we define the action-value of a state-action pair as

$$Q^\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$$

- The relationship between Q and V

$$V^\pi(s) = \sum_{a \in A} Q^\pi(s, a) \pi(a|s)$$

Finding the Value Function of States



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

$$\sum_{s'} P(s'|s, a)[r(s, a, s') + \gamma * V(s')]$$

- How do we choose the optimal action?

$$\pi^*(s) = \operatorname{argmax}_{a \in A} \sum_{s'} P(s'|s, a)[r(s, a, s') + \gamma V(s')]$$

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

$$V(s) = \max_{a \in A} \sum_{s'} P(s'|s, a)[r(s, a, s') + \gamma V(s')]$$

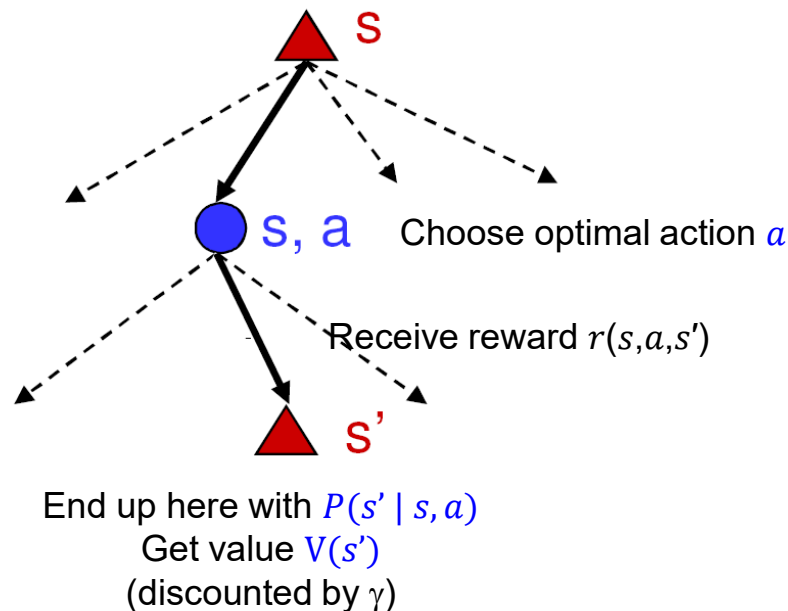
The Bellman Equation



- Recursive relationship between the accumulated rewards of successive states:

$$V(s) = \max_{a \in A} \sum_{s'} P(s'|s, a) [r(s, a, s') + \gamma V(s')]$$

- 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 $V(s) = 0$
- Iterate until convergence
 - During the i th iteration, update the value of each state according to this rule:

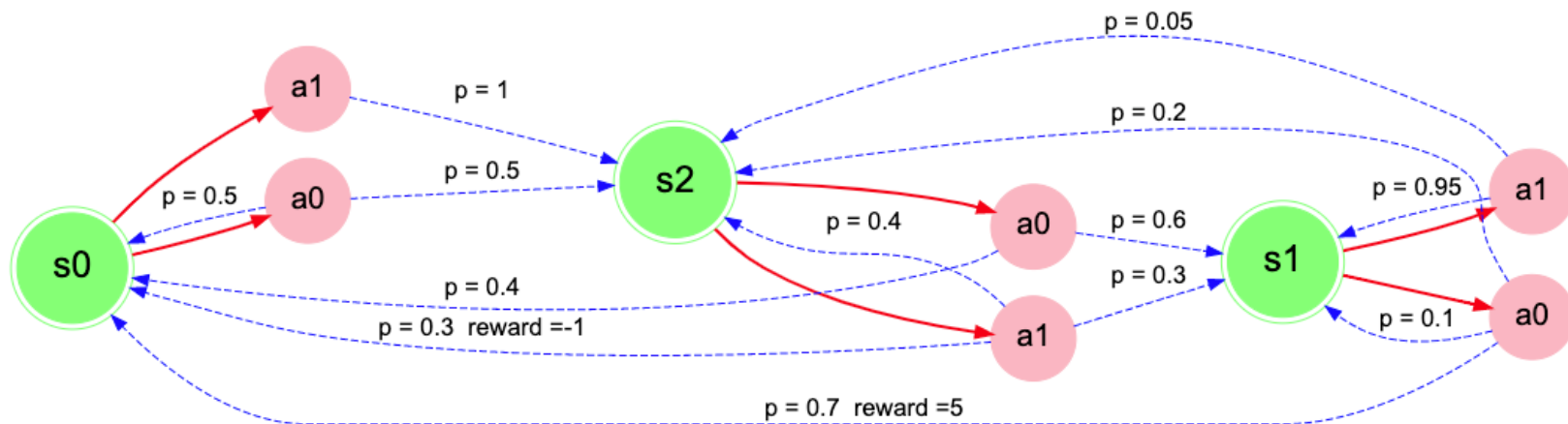
$$V_{i+1}(s) = \max_a \sum_{s' \in S} P(s'|s, a) [R(s, a, s') + \gamma V_i(s')]$$

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



Value Iteration: Example

- A simple example to show how VI works
 - State, action, reward (non-zero) and transition probability are shown in the figure
 - We use this example as an MDP and solve it using VI and PI



These structure can be easily implemented by `dict` of Python or `HashMap` of Java



Value Iteration: Example (cont'd)

- Given the states and actions are finite, we can use matrices to represent the value function $V(s)$
- Pseudo-code of VI
 1. Initialize $V_0(s)$, for all s
 2. For $i = 0, 1, 2 \dots$
 3. $V_{i+1}(s) = \max_a \sum_{s'} P(s'|s, a)[r(s, a, s') + \gamma V_i(s')]$ for all state s



Value Iteration: Example (cont'd)

- How VI works in an iteration? Given iteration $i = 0$

For all state find

$$V_{i+1}(S) = \max_a \sum_{s'} P(s'|s, a) [r(s, a, s') + \gamma V_i(s')]$$

We can instead calculate $Q(s, a)$ values for each s and a and get the best $V(s)$

$$Q_{i+1}(s, a) = \sum_{s'} P(s'|s, a) [r(s, a, s') + \gamma V_i(s')]$$

Finally

$$V_{i+1}(S) = \max_a \sum_{s'} P(s'|s, a) [r(s, a, s') + \gamma V_i(s')] = \max_a Q_{i+1}(s, a)$$



Value Iteration: Example (cont'd)

- We use the previous example and solve it using VI
- Construct a Q table, $Q(s, a)$
 - 3 rows (3 states) and 2 columns (2 actions)
 - $V_0 = [0, 0, 0]$ # 3 states

$$Q_{i+1}(s, a) = \sum_{s'} P(s'|s, a) [r(s, a, s') + \gamma V_i(s')]$$

Q_0			V_0	Q_1			V_1
	a_0	a_1			a_0	a_1	
s_0	0	0	0	s_0	0	0	0
s_1	0	0	0	s_1	3.5	0	3.5
s_2	0	0	0	s_2	0	-0.3	0

$$Q_1(s_1, a_0) = P(s_0|s_1, a_0) [r(s_1, a_0, s_0) + \gamma V_0(s_0)] + \\ P(s_1|s_1, a_0) [r(s_1, a_0, s_1) + \gamma V_0(s_1)] + \\ P(s_2|s_1, a_0) [r(s_1, a_0, s_2) + \gamma V_0(s_2)]$$

$$Q_1(s_1, a_0) = 0.7 * [5 + 0] + 0.1 * [0 + 0] + 0.2 * [0 + 0] = 3.5$$

The initial Q and V table

Repeat this process until it converges



Value Iteration: Example (cont'd)

- Iterating the process (code available on later slides)

iter 0 | $V(s_0) = 0.000$ $V(s_1) = 0.000$ $V(s_2) = 0.000$

iter 1 | $V(s_0) = 0.000$ $V(s_1) = 3.500$ $V(s_2) = 0.000$

iter 2 | $V(s_0) = 0.000$ $V(s_1) = 3.815$ $V(s_2) = 1.890$

iter 3 | $V(s_0) = 1.701$ $V(s_1) = 4.184$ $V(s_2) = 2.060$

... ..

iter 63 | $V(s_0) = 8.020$ $V(s_1) = 11.160$ $V(s_2) = 8.912$

iter 64 | $V(s_0) = 8.021$ $V(s_1) = 11.161$ $V(s_2) = 8.913$

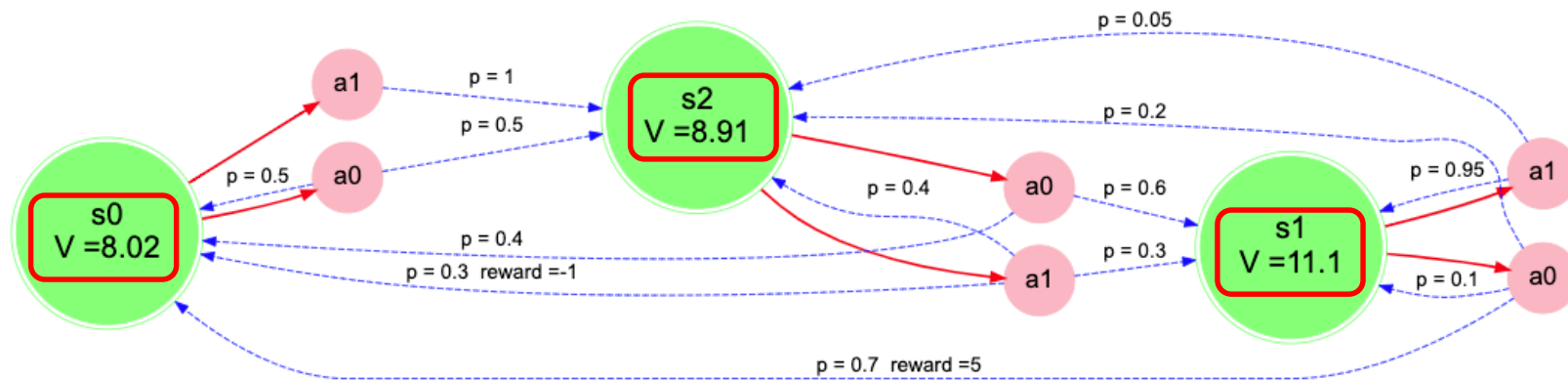
iter 65 | $V(s_0) = 8.022$ $V(s_1) = 11.162$ $V(s_2) = 8.915$

Very good, the
values converge
now



Value Iteration: Example (cont'd)

- Put the optimal values on the graph





Value Iteration: Example (cont'd)

- Use V^* to find optimal policy
 - aka optimal actions in each state

$$\pi^*(s) = \operatorname{argmax}_a \sum_{s'} P(s'|s, a) [r(s, a, s') + \gamma V_i(s')] = \operatorname{argmax}_a Q_i(s, a)$$

π^*		a^*
	s_0	a_1
	s_1	a_0
	s_2	a_0

Done! We get the optimal policy of the example

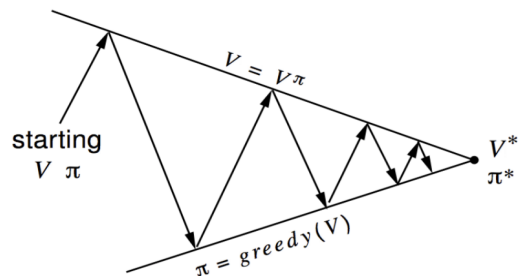
Python Code on Colab: https://colab.research.google.com/drive/1DnYlr3QJxpfs_rR_jAAUrqHZMjvsjGSx?usp=sharing



Method 2: Policy Iteration

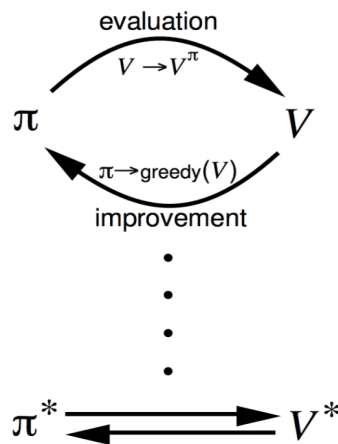
- Start with some initial policy π_0 and alternate between the following steps:
 - Policy evaluation:** calculate $V^{\pi_i}(s)$ for every state s , *like VI*
 - Policy improvement:** calculate a new policy π_{i+1} based on the updated utilities

$$\pi_{i+1}(s) = \operatorname{argmax}_{a \in A} \sum_{s'} P(s'|s, a) [r(s, a, s') + \gamma V^{\pi_i}(s')]$$



Policy evaluation Estimate v_π
Any policy evaluation algorithm

Policy improvement Generate $\pi' \geq \pi$
Any policy improvement algorithm





Policy Iteration: Main Steps

- Unlike VI, policy iteration has to maintain a policy - chosen actions from all states - and estimate V^{π_i} based on this policy.
 - Iterate this process until convergence (like VI)
- Steps of PI
 1. Initialization
 2. Policy Evaluation (calculating the V)
 3. Policy Improvement (calculate the policy π)



Policy Iteration: Detailed Steps

1. Initialization
 - a. Initialize $V(s)$ and $\pi(s)$ for all state s
2. Policy Evaluation (calculate the V)
 - a. Repeat
 - b. $\Delta \leftarrow 0$
 - c. For each state s :
 - d. $v \leftarrow V(s)$
 - e. $V(s) \leftarrow \sum_{s'} p(s'|s, \pi(s)) [R(s, \pi(s), s') + \gamma V(s')]$
 - f. $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
 - g. until $\Delta < \omega$ (a small positive number)
3. Policy Improvement (calculate the new policy π)
 - a. *polycystable* \leftarrow *True*
 - b. For each state s :
 - c. $b \leftarrow \pi(s)$
 - d. $\pi(s) \leftarrow \operatorname{argmax}_{\pi(s)} \sum_{s'} p(s'|s, \pi(s)) [R(s, \pi(s), s') + \gamma V(s')]$
 - e. If $b \neq \pi(s)$, then *polycystable* \leftarrow *False*
 - f. If *polycystable* \leftarrow *True*, then stop; else go to step 2;

Policy Iteration: Policy Evaluation



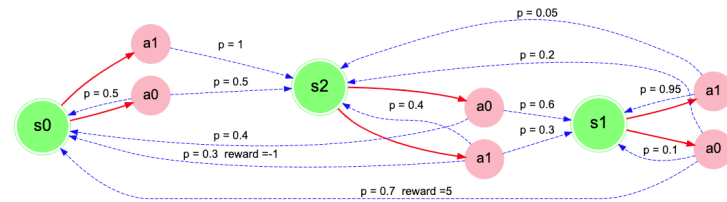
- Use the example used in VI
- Start iteration $i = 0$, initialize random π , $\gamma = 0.99$, and $V(s) = 0$ for all state s

$$V^{\pi_i}(s) = \sum_{s'} P(s, \pi_i(s), s') [R(s, \pi_i(s), s') + \gamma V^{\pi_i}(s')]$$

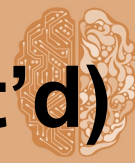
$$V(s_0) \leftarrow P(s_0, \pi(s_0), s_0) [R(s_0, \pi(s_0), s_0) + \gamma V(s_0)] + \\ P(s_0, \pi(s_0), s_1) [R(s_0, \pi(s_0), s_1) + \gamma V(s_1)] + \\ P(s_0, \pi(s_0), s_2) [R(s_0, \pi(s_0), s_2) + \gamma V(s_2)]$$

$$V(s_1) \leftarrow P(s_1, \pi(s_1), s_0) [R(s_1, \pi(s_1), s_0) + \gamma V(s_0)] + \\ P(s_1, \pi(s_1), s_1) [R(s_1, \pi(s_1), s_1) + \gamma V(s_1)] + \\ P(s_1, \pi(s_1), s_2) [R(s_1, \pi(s_1), s_2) + \gamma V(s_2)]$$

$$V(s_2) \leftarrow P(s_2, \pi(s_2), s_0) [R(s_2, \pi(s_2), s_0) + \gamma V(s_0)] + \\ P(s_2, \pi(s_2), s_1) [R(s_2, \pi(s_2), s_1) + \gamma V(s_1)] + \\ P(s_2, \pi(s_2), s_2) [R(s_2, \pi(s_2), s_2) + \gamma V(s_2)]$$



Policy Iteration: Policy Evaluation (cont'd)



- Use the example used in VI
- Start iteration $i = 0$, $\gamma = 0.99$, initialize random $\pi = [a_1, a_0, a_1]$ and $V(s) = 0$ for states s_0 , s_1 and s_2

$$V(s_0) \leftarrow 0.0[0.0 + \gamma V(s_0)] + 0.0[0.0 + \gamma V(s_1)] + 1.0[0.0 + \gamma V(s_2)]$$

$$V(s_1) \leftarrow 0.7[5.0 + \gamma V(s_0)] + 0.1[0.0 + \gamma V(s_1)] + 0.2[0.0 + \gamma V(s_2)]$$

$$V(s_2) \leftarrow 0.3[-1 + \gamma V(s_0)] + 0.3[0.0 + \gamma V(s_1)] + 0.4[0.0 + \gamma V(s_2)]$$



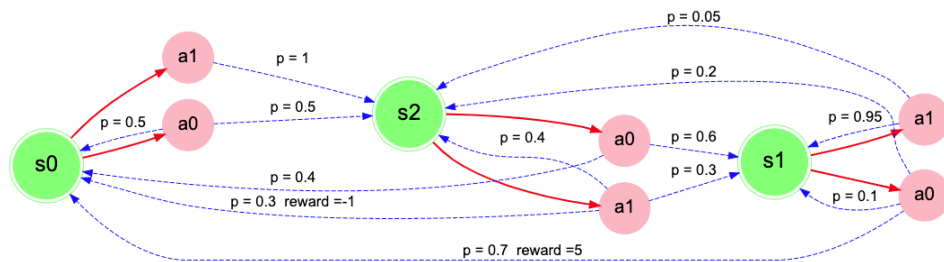
$$V(s_0) = 0$$

$$V(s_1) = 0.7 * 5.0 = 3.5$$

$$V(s_2) = 0.3 * (-1) + 0.3 * (0.99 * 3.5) = 0.7395$$

... loop this process as instructed in step 2 ...

- Values are calculated asynchronously
- The converged values are used for policy improvement

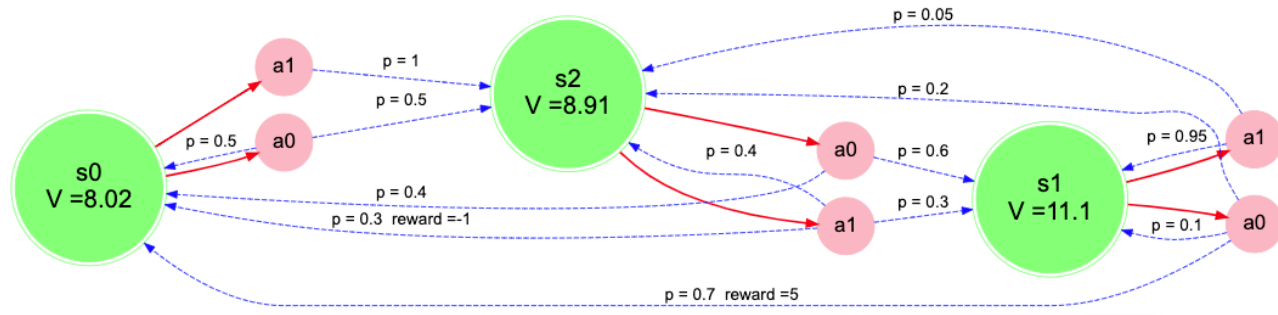


Policy Iteration: Example (cont'd)



- The optimal Q values are
 - Nearly the same as that of VI (as shown in the figure below)

	V^*
s_0	8.03
s_1	11.2
s_2	8.9



- We can easily calculate the optimal policy. Can you try it?

$$\pi^*$$

	a^*
s_0	a_1
s_1	a_0
s_2	a_0

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*

