Reinforcement Learning Fundamentals

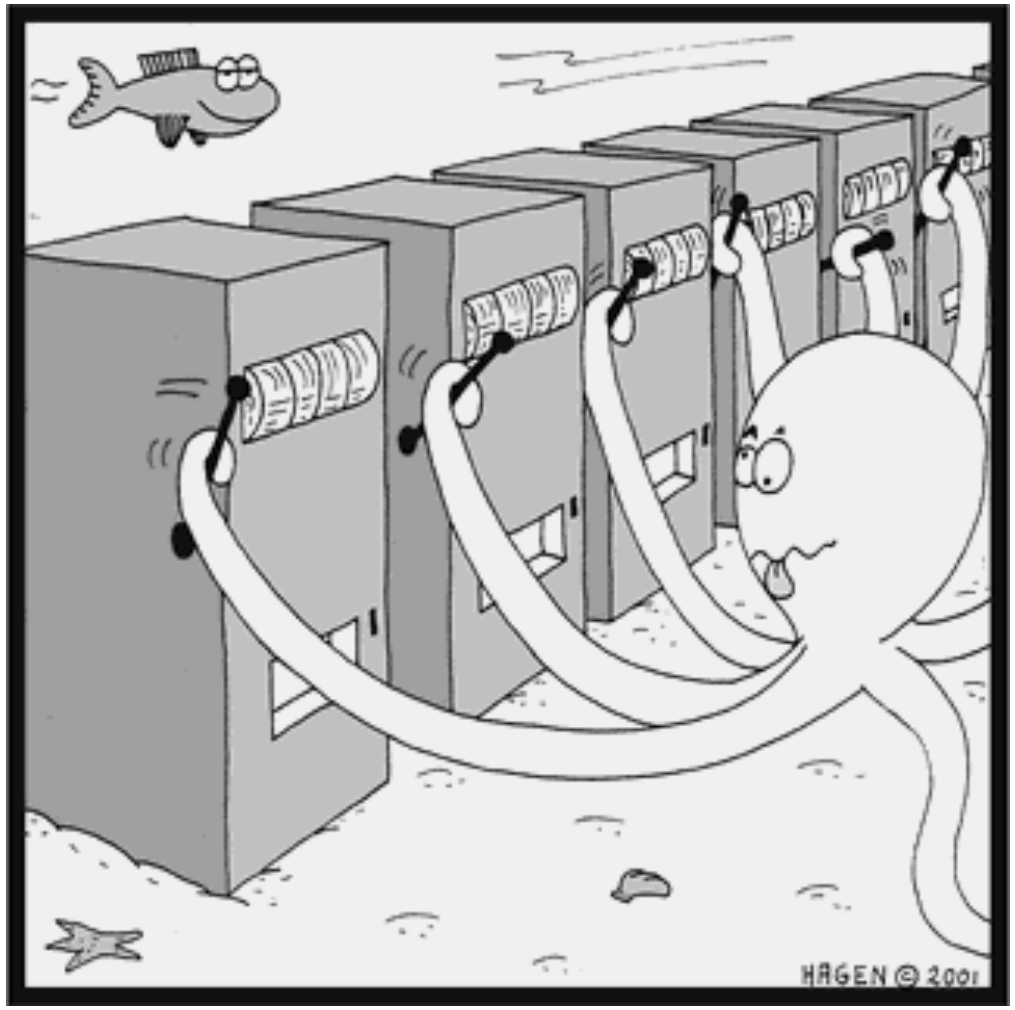
Lecture 7: Multi-armed Bandit

Dr Sandeep Manjanna
Assistant Professor, Plaksha University
sandeep.manjanna@plaksha.edu.in

In today's class…

- Performance metrics
  - Correctness
  - Convergence
  - Sample Efficiency

- Solution methods
  - SoftMax
  - Upper Confidence Bound



Multi-armed Bandit

- n-arm bandit problem is to learn to preferentially select a particular action (arm) from a set of n actions $(1, 2, 3, \ldots, n)$
- Each selection results in Rewards derived from the respective probability distribution
- Arm i has a reward distribution with mean μ_i and

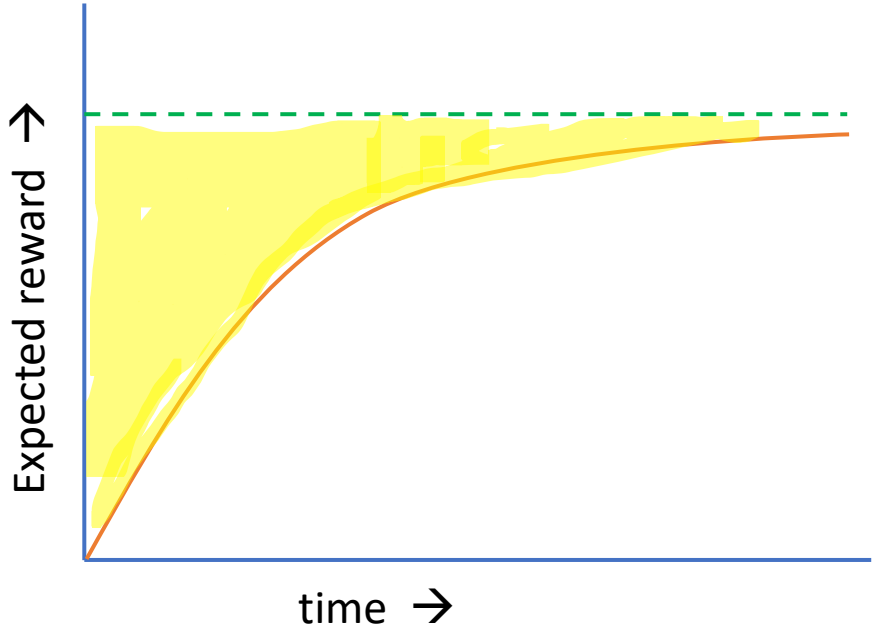
$$\mu^* = \max\{\mu_i\}$$

Performance Metrics

- **Asymptotic Correctness –** Identify the correct arm eventually
    - Gives a guarantee that eventually the algorithm will be selecting an arm that has the highest pay-off.
        - As T tends to infinity.

Performance Metrics

- **Regret Optimality**
  - "disappointed over (something that one has done or failed to do)" – *definition of regret from Oxford Dictionary*



  - Regret optimality refers to increasing the total reward one gets over the process of learning → Minimize regret while learning

Performance Metrics

- **PAC Optimality (Probably Approximately Correct)**
  - Approximately right in Bandit setup:
    - The arm suggested has expected payoff close to the expected payoff of the best arm.
  - Probably – It is either approximately correct or not!
  - (ε, δ) – PAC framework
    - Identification of an ε-optimal arm with probability $1 - \delta$
    - ε-Optimal: Mean of the selected arm satisfies $\mu > \mu^* - \varepsilon$

$$\{ P (q_*(a) \geq (q_*(a_*) - \varepsilon))\} \geq (1 - \delta)$$

True expected value of an action ***a***

True expected value of the best action $\boldsymbol{a_*}$

Performance Metrics

- Asymptotic Correctness → measures **Correctness** of the solution
- Regret Optimality → measures the rate of **Convergence** of the solution
- PAC Optimality (Probably Approximately Correct) → **Sample efficiency** : want to minimize the sample size. Not very much used for practically implementable algorithm.

Solution Approaches

Exploration methods:

$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}}$$

- **Epsilon Greedy:** Select an arm $A_t \doteq \underset{a}{\operatorname{argmax}} Q_t(a)$ with probability (1- ε) and select any arbitrary arm with probability ε.

  - **Some problems:**

    Even if we know that for a certain a_i, $Q_t(a_i) << Q_t(a_*)$, we still sample a_i with a fixed probability.

    - Wasted trials
    - Affects the regret / increased regret

Solution Approaches

Exploration methods:

$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}}$$

- **Epsilon Greedy:** Select an arm $A_t \doteq \underset{a}{\operatorname{argmax}} Q_t(a)$ with probability (1- ε) and select any arbitrary arm with probability ε.

- **SoftMax:**
  - Converts a set of values into probability distribution.
  - Select arms with probability proportional to the current value estimates.

$$\pi_t(a_i) = \frac{\exp(Q_t(a_i)/\tau)}{\sum_j \exp(Q_t(a_j)/\tau)}$$

Temperature parameter τ

- Asymptotic convergence guarantees

Other Approaches

- **Median Elimination** (Even-Dar et at., 2006)
- **Upper Confidence Bound** (UCB by Auer et al., 1998)
- **Thompson Sampling** (Chappelle & Li, 2001)

Incremental Value-function

$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}}$$

For a given action, $Q_n \doteq \dfrac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$

$$\begin{aligned}
Q_{n+1} &= \frac{1}{n}\sum_{i=1}^{n} R_i \\
&= \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\left(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\Big(R_n + (n-1)Q_n\Big) \\
&= \frac{1}{n}\Big(R_n + nQ_n - Q_n\Big) \\
&= Q_n + \frac{1}{n}\Big[R_n - Q_n\Big],
\end{aligned}$$

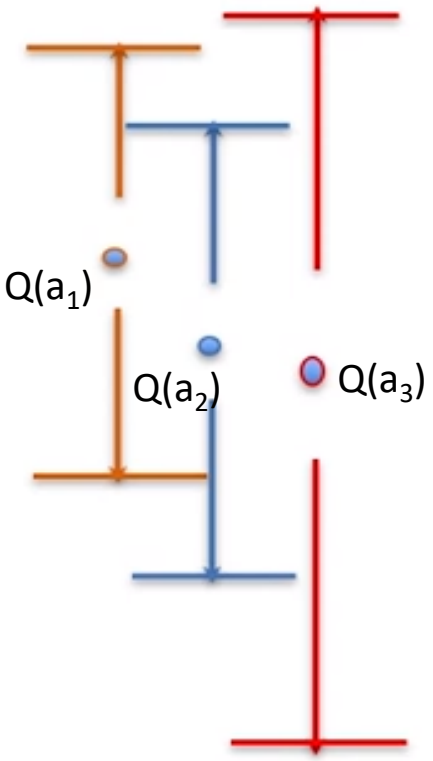
$$NewEstimate \leftarrow OldEstimate + StepSize\left[Target - OldEstimate\right]$$

Simple ε-greedy,

Initialize, for $a = 1$ to k:
$Q(a) \leftarrow 0$
$N(a) \leftarrow 0$

Loop forever:
$A \leftarrow \begin{cases} \operatorname{argmax}_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$
$R \leftarrow bandit(A)$
$N(A) \leftarrow N(A) + 1$
$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\big[R - Q(A)\big]$

Upper Confidence Bound Action Selection



- **SoftMax** will still allocate probability to a_2 and a_3 as the values are close to a_1 . Even after convergence of value function.
- The **confidence interval** is the range of values that you expect your estimate to fall between a certain percentage of the time if you run your experiment again or re-sample the population in the same way.
- The **confidence level** is the percentage of times you expect to reproduce an estimate between the upper and lower bounds of the confidence interval.
- **UCB suggests**: Be greedy with respect to the upper confidence bound.

Upper Confidence Bound Action Selection

(UCB by Auer et al., 1998)

This term is a measure of the **uncertainty** or **variance** in the estimate of a's value.

$$A_t \doteq \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right]$$

- ***ln t*** denotes the natural logarithm of ***t***
- $N_t(a)$ denotes the number of times that action ***a*** has been selected prior to time ***t***
- The number ***c > 0*** controls the degree of exploration

Reinforcement Learning Fundamentals

Lecture 8: Contextual RL and Full RL

Dr Sandeep Manjanna
Assistant Professor, Plaksha University
sandeep.manjanna@plaksha.edu.in

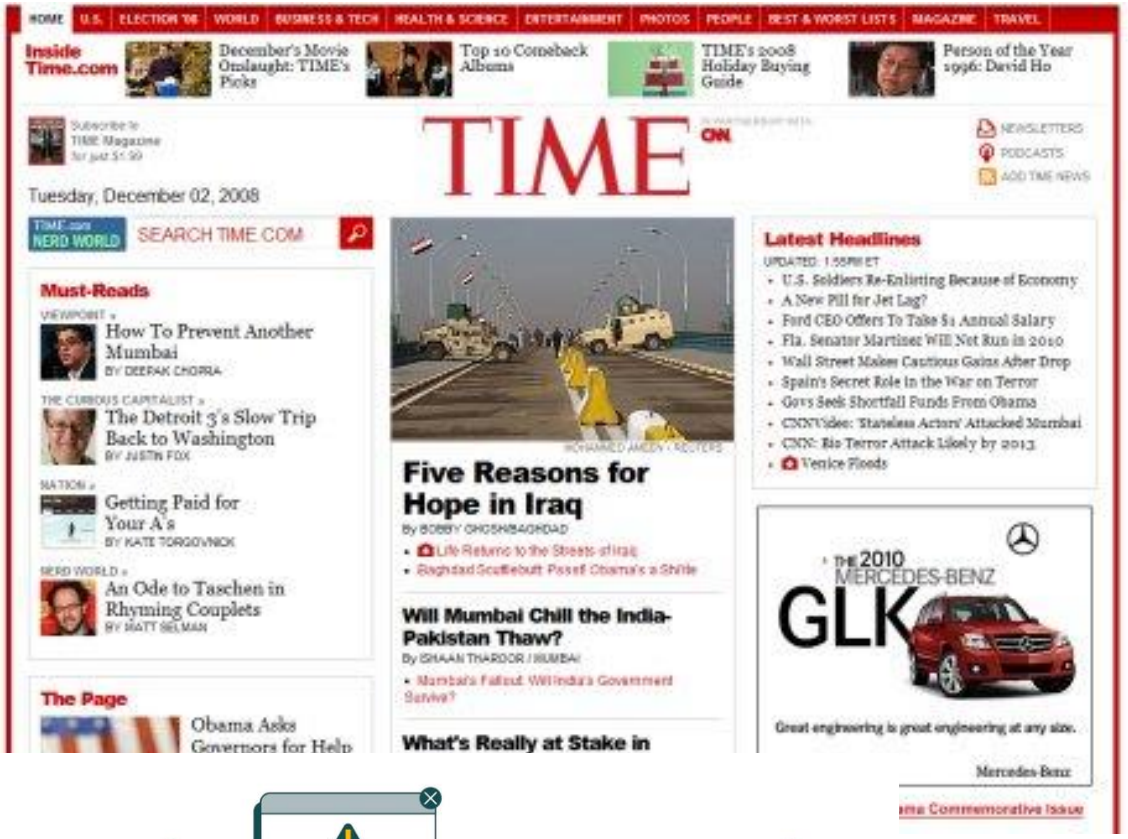
Any implementation of UCB algorithm for Bandit problem?

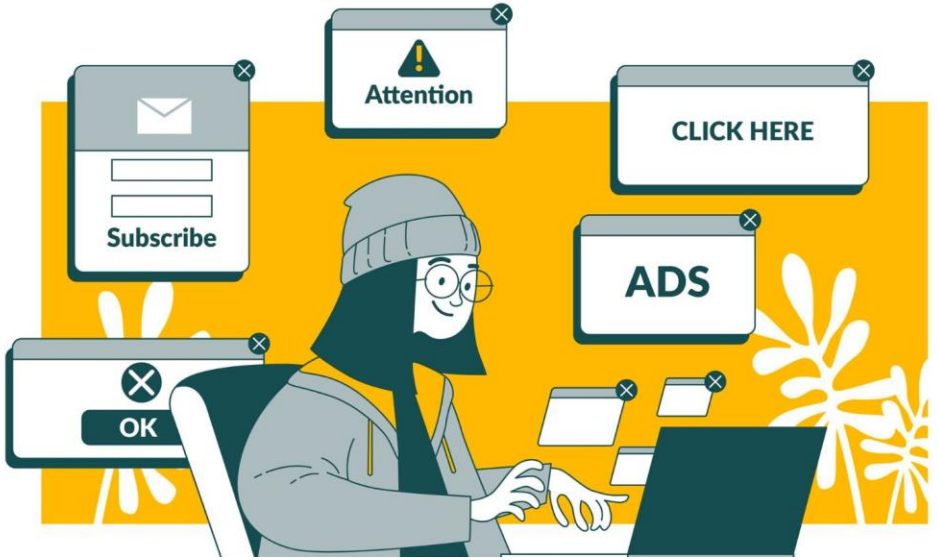
In today's class…

- Contextual Bandit
- Temporal difference
- Full RL Problem

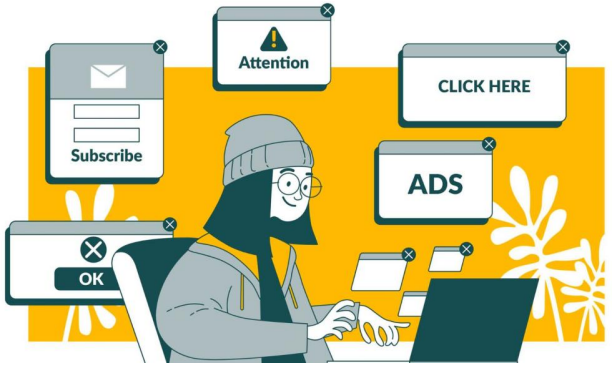
Contextual Bandits

- Customization
  - Different news / ads for different users.
- Different recommendation for different users
  - One UCB for each user?
- **Not a good solution. Why??**
  - Hard to Train
    - Lots and lots of users
    - Less experience with each user
    - Preferences / relevance change over time





Contextual Bandits



- Can we group users into categories? And then run a UCB for each user group.

- How to categorize / what are the possible parameters we can use to categorize?
    - Age, Gender, Browsing behaviour, Location
    - Demographic or behaviour or engagement features, etc.

- We don’t need to know the person X, all we need to know is the attributes of X.

Contextual Bandits

- Now, assume that the parameters of the reward distributions are determined by a set of hyperparameters (features / attributes of the users).
    - μ and σ of the reward distribution are a function of the features / attributes of the user.

- The statistic used to choose an arm is now dependent on these features or attributes.

- Instead of learning ***Q(a)***, we will learn ***Q(s,a)***. Now we will track ***Q(s,a)*** and $n_{s,a}$.
    - ***Q(s,a)*** represents the value of taking action ***a*** in the context ***s***.
    - $n_{s,a}$ represents the number of times action ***a*** is chosen w.r.t context ***s***.

- Can action ***a*** also be represented with a set of features? What's the use?
    - Change of stories / ads will not need to start a new bandit.

Contextual Bandits

- LinUCB by Li et al., in 2010
  - One of the more popular contextual bandit algorithms
  - *Predicted expected reward* assumed to be a linear function of the features
    - Use ridge regression to fit parameters
    - Can derive upper confidence bounds for the regression fit
    - Use UCB like action selection
    - Gives better performance with lesser "training" data
- Contextual bandit is a powerful extension of Bandit setting.

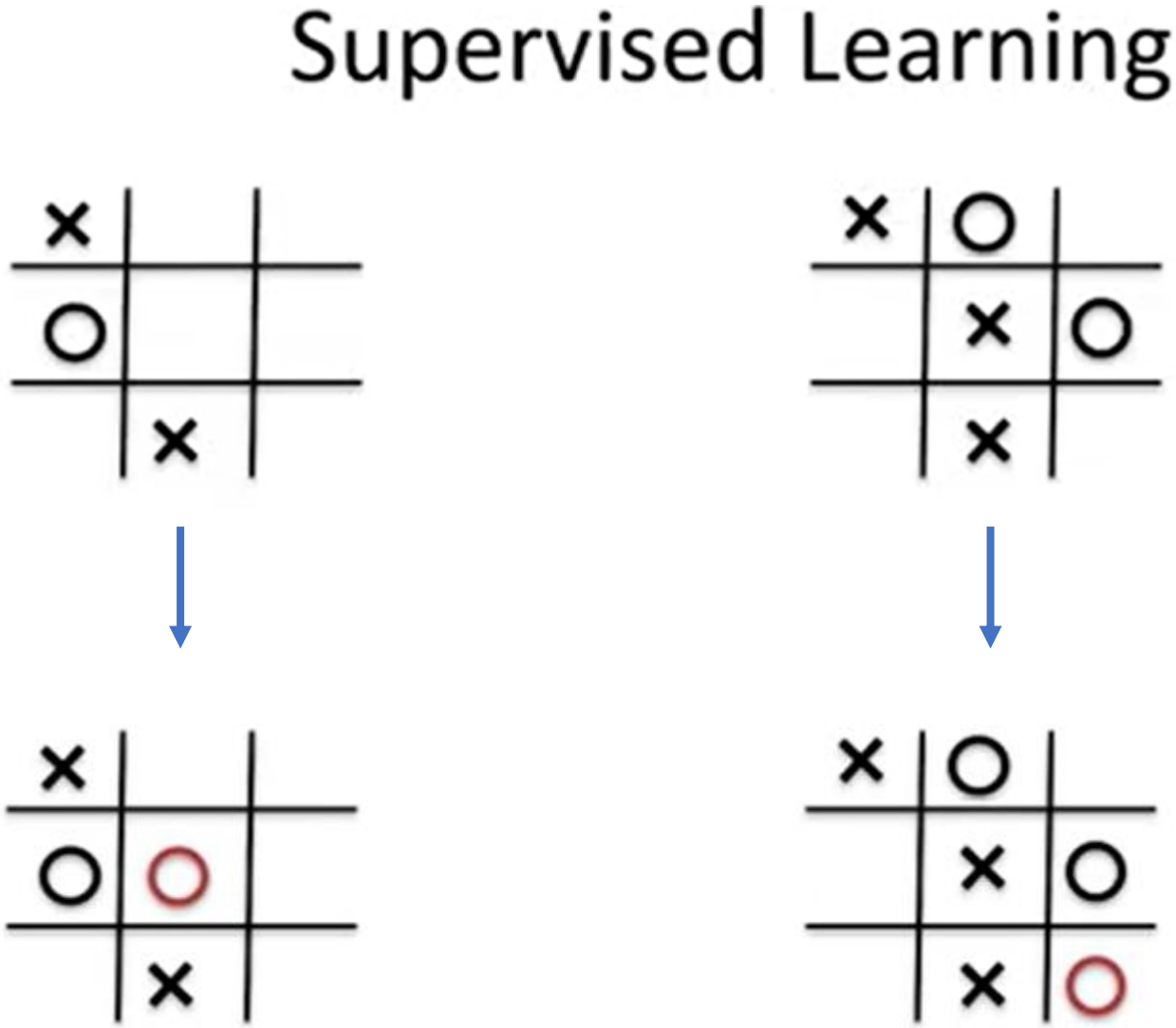
Bandit / Immediate
Only actions

Contextual Bandit
Actions, and Context

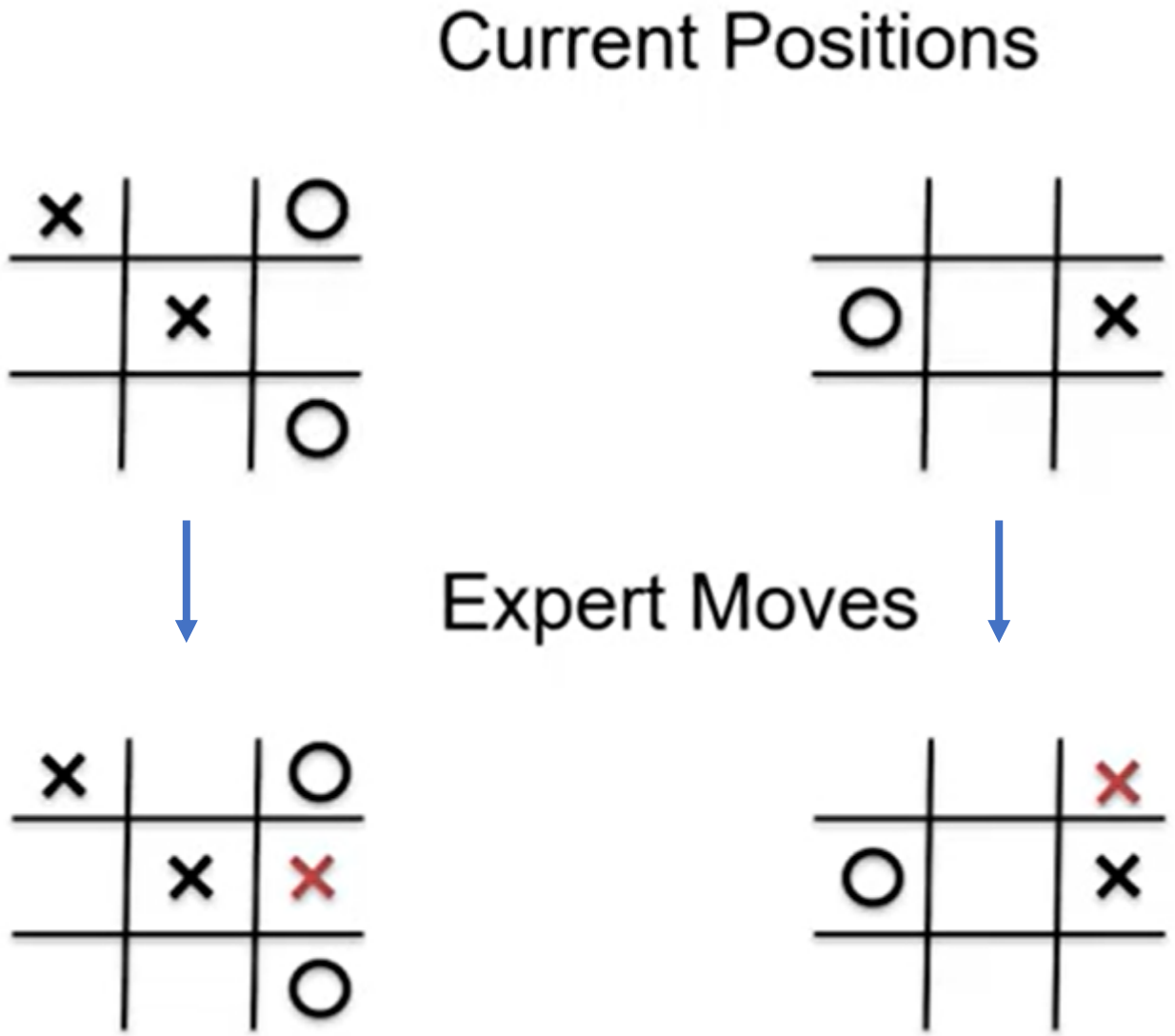
Full RL Problem
Actions, Context, and Sequence

Slide from Prof. Ravindran's course: "Reinforcement Learning."

Example game: Tic-Tac-Toe



Example game: Tic-Tac-Toe



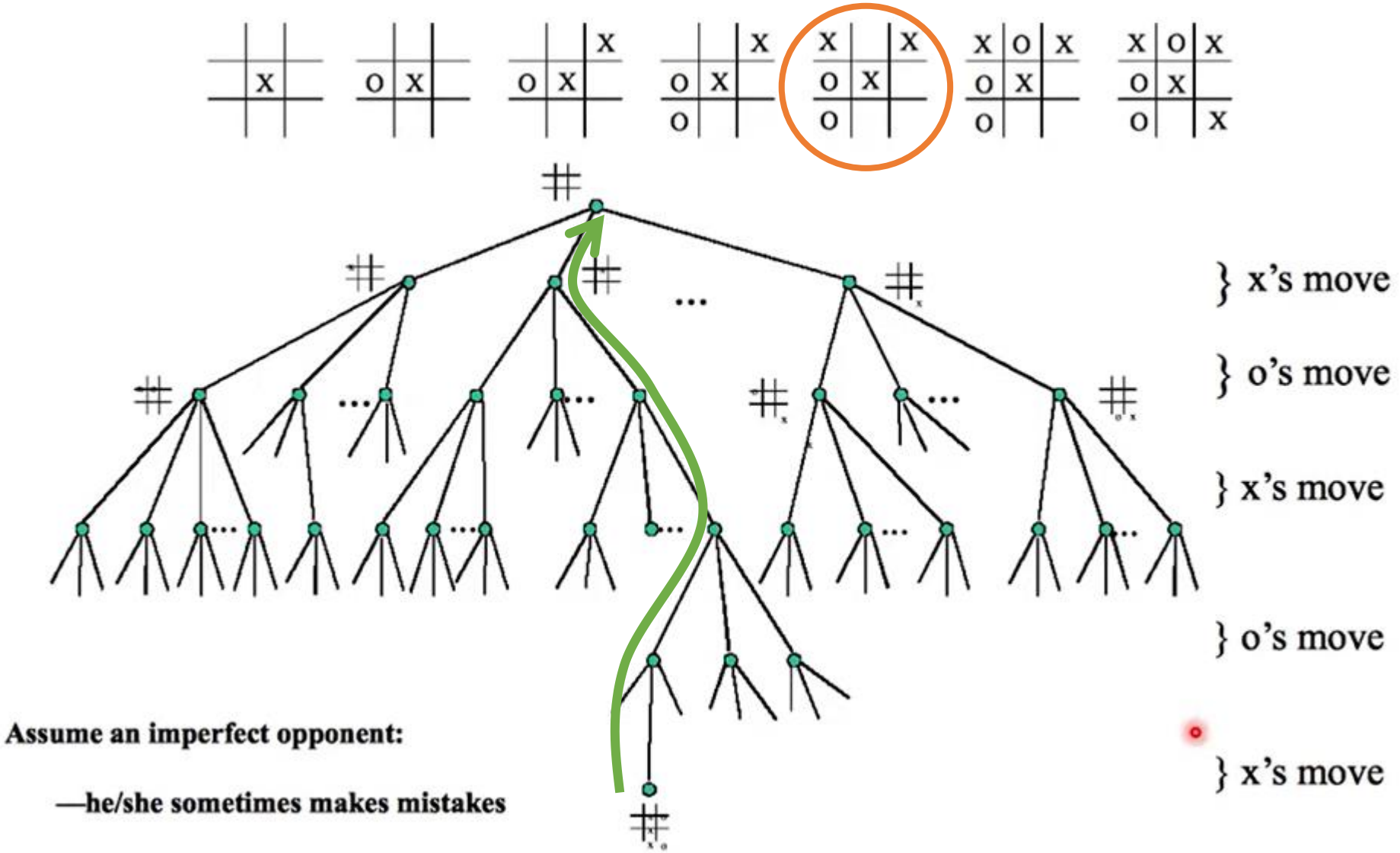
Example game: Tic-Tac-Toe

How to do this with Reinforcement Learning?

- Don’t have to tell how to play. Only inform about the legal moves.
- Learn from evaluation
  - Win gives 1 point
  - Loss gives -1 point
  - Draw gives 0 points
- Learn by playing repeatedly

MENACE (Michie and Chambers in 1960)

Example game: Tic-Tac-Toe



Temporal Difference

Barto, Sutton, Anderson in 1983

Intuition: Prediction of outcomes made at time ***t+1*** is better than the prediction of outcomes made at time ***t***.

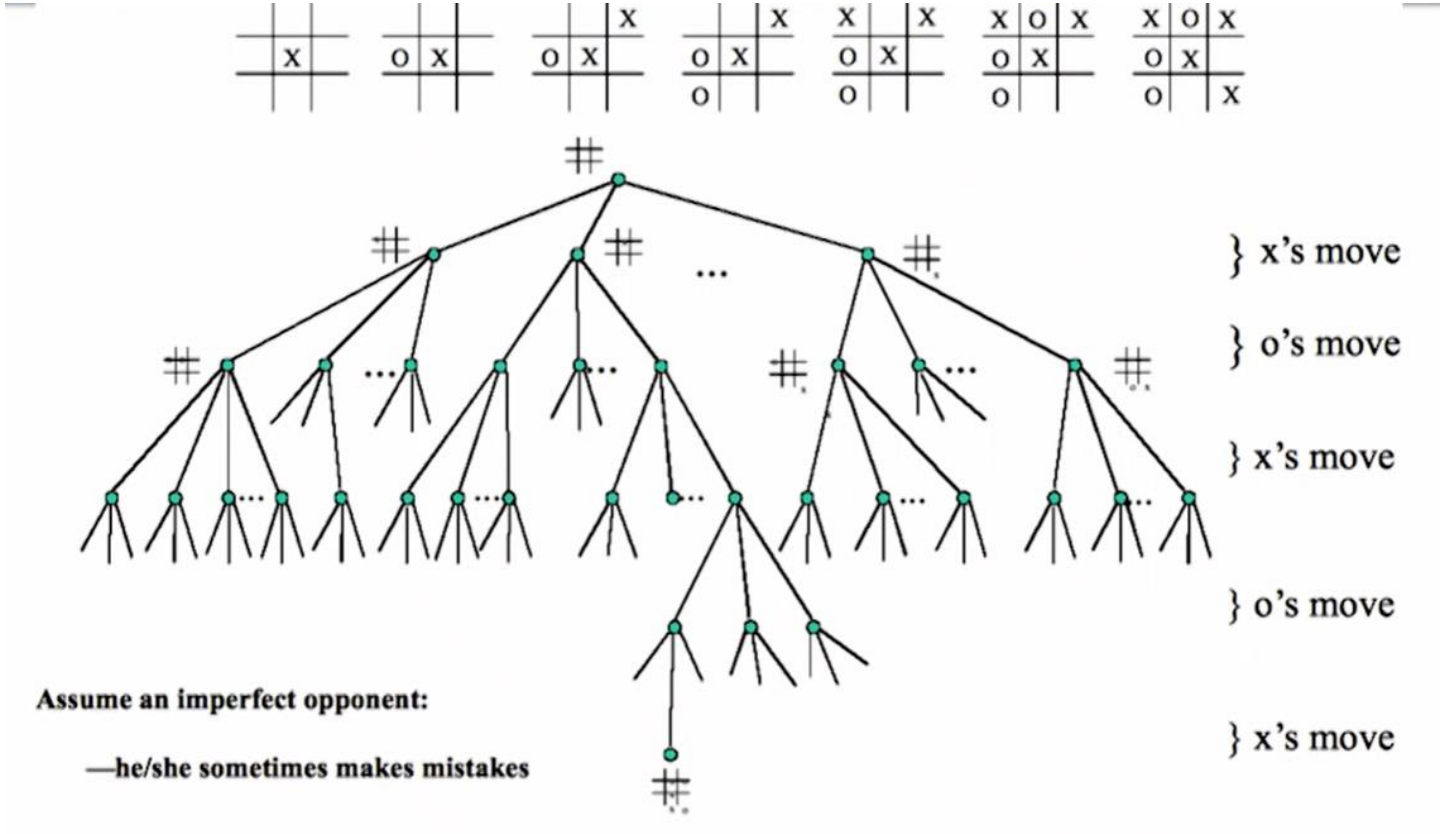
- Hence, the predictions made at later timestep can be used to update the predictions made at earlier timestep.

Created significant impact in behavioral psychology and neuroscience.

TD in Brain (Monkey Experiment).

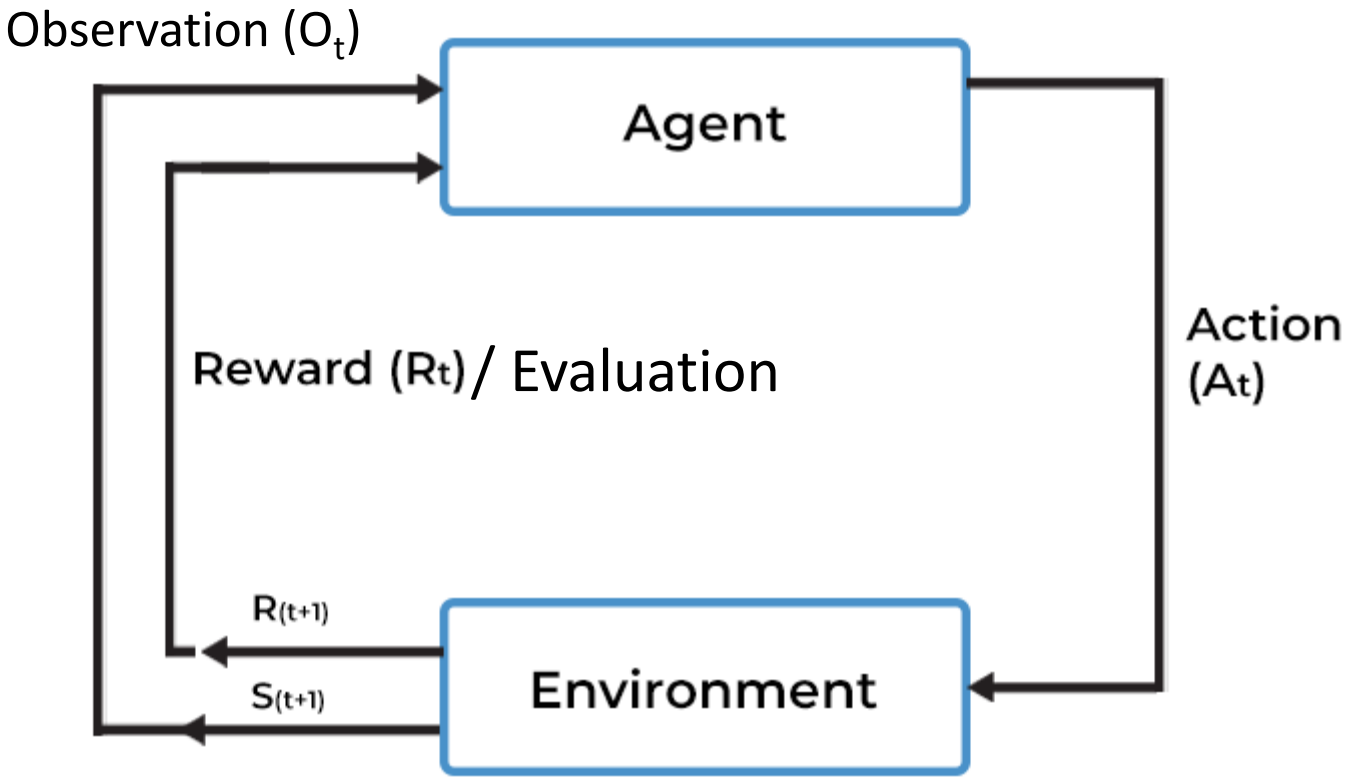
Full RL Problem

Tic-Tac-Toe example:



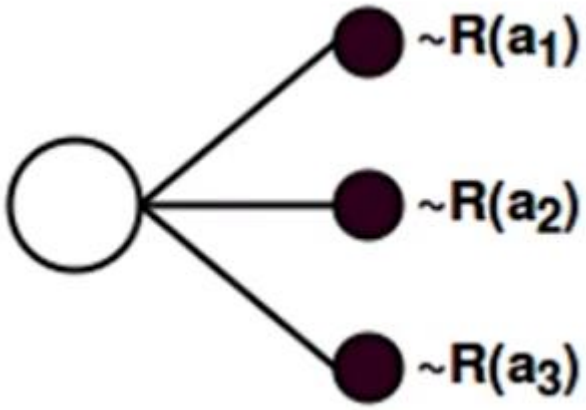
1. Sequence of decisions.
2. Reward is delayed.
3. The second problem in the sequence depends on what action you chose in the first problem.

Full RL Problem

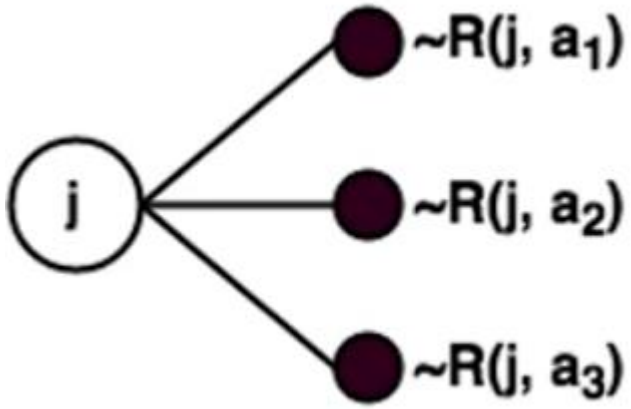


- States
- Environment
- Rewards
- Policy
- Value function
- Model

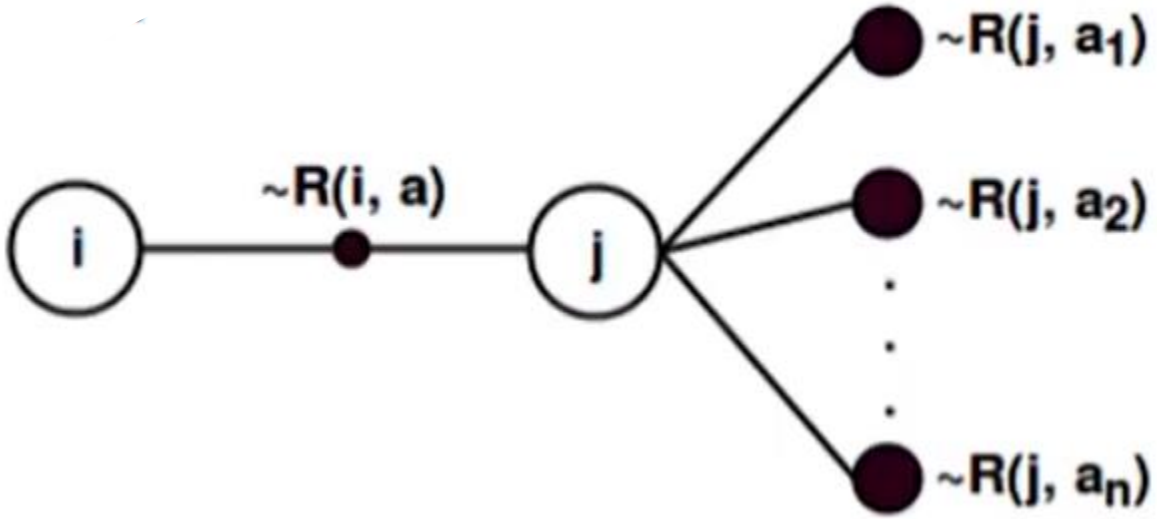
Full RL Problem



Bandit Problem



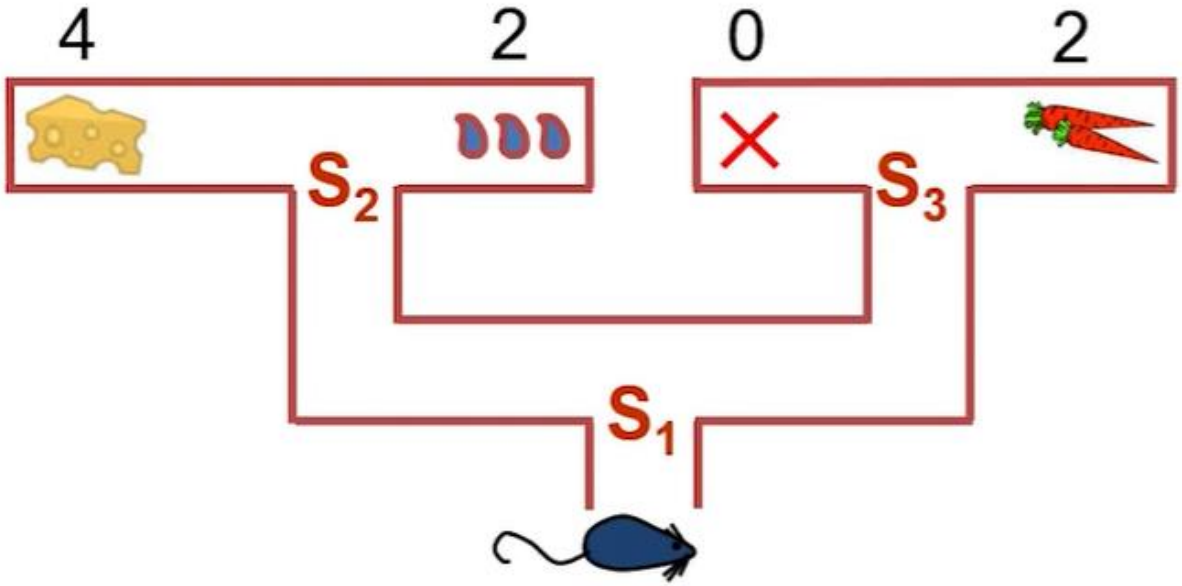
Contextual Bandit



Full RL Problem

Slide from Prof. Ravindran's course: "Reinforcement Learning."

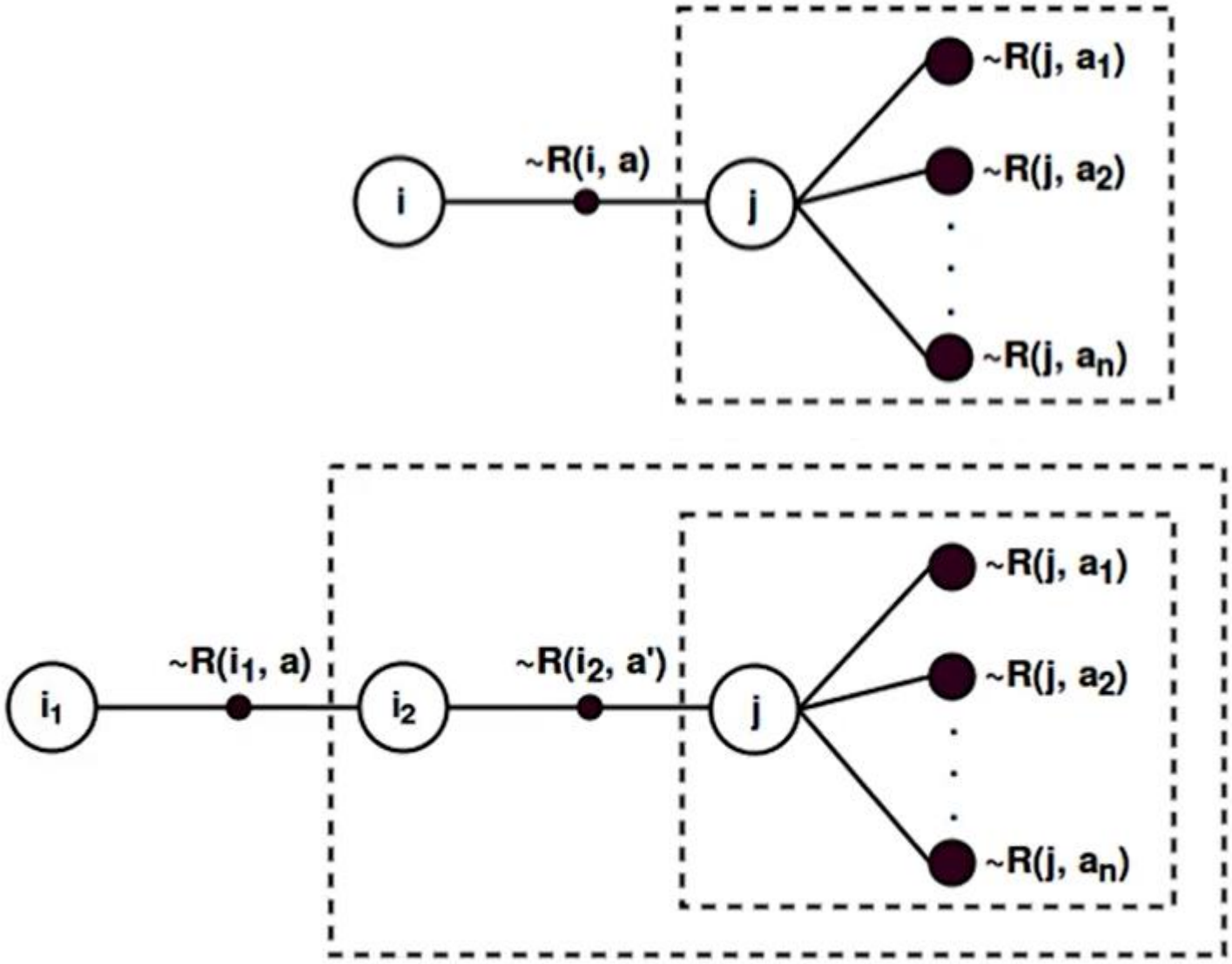
Action at a Temporal Distance



- learning an appropriate action at S_1:
  - –depends on the actions at S_2 and S_3
  - –gains no immediate feedback
- Idea: use prediction as surrogate feedback

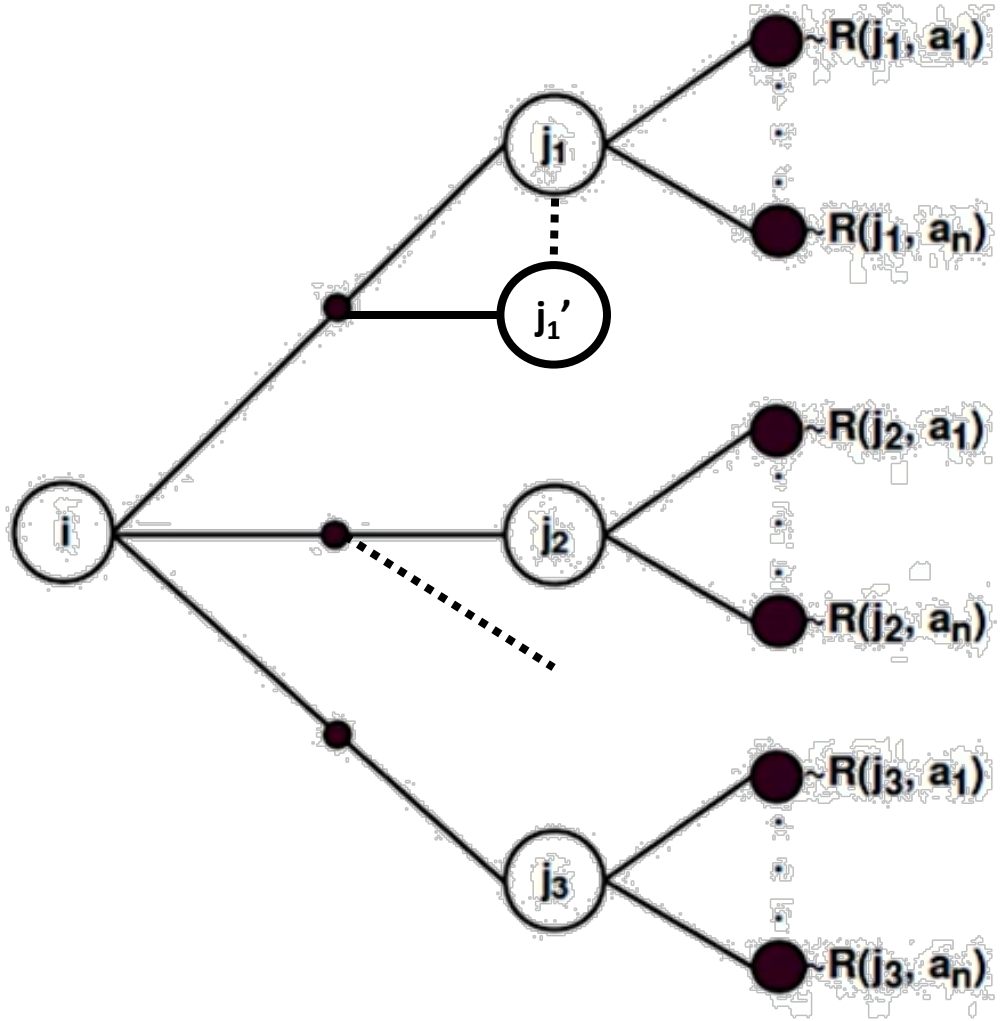
Slide from Prof. Ravindran's course: "Reinforcement Learning."

Full RL Problem



Slide from Prof. Ravindran's course: "Reinforcement Learning."

Full RL Problem



Slide from Prof. Ravindran's course: "Reinforcement Learning."

Reinforcement Learning Fundamentals

Lecture 9: Markov Decision Process (MDP)

Dr Sandeep Manjanna
Assistant Professor, Plaksha University
sandeep.manjanna@plaksha.edu.in

In today's class…

- Class Presentations
  1. UCB vs. Epsilon Greedy
  2. Ridge Regression
  3. MENACE
  4. TD in Brain

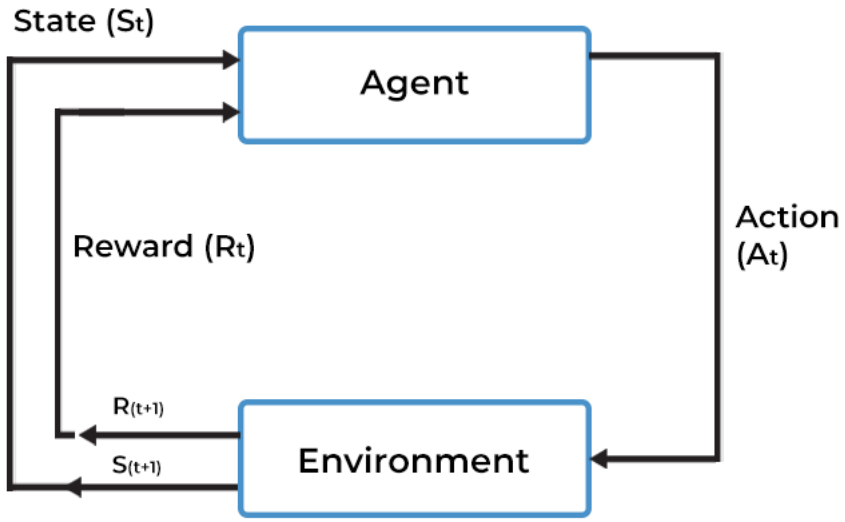
- Markov Property
- Markov Process
- Markov Reward Process
- Markov Decision Process (MDP)

Fully Observable Environments

… extent to which the agent has access to information about the current state of the environment.

- A fully observable environment is one in which the **agent has complete information about the current state of the environment.**
- The agent has direct access to all environmental features that are necessary for making decisions.
- Example?

Board games like chess or checkers.



Full observability: agent directly observes environment state

$$O_t = S_t^a = S_t^e$$

- Agent state = environment state = information state
- Formally, this is a Markov decision process (MDP)

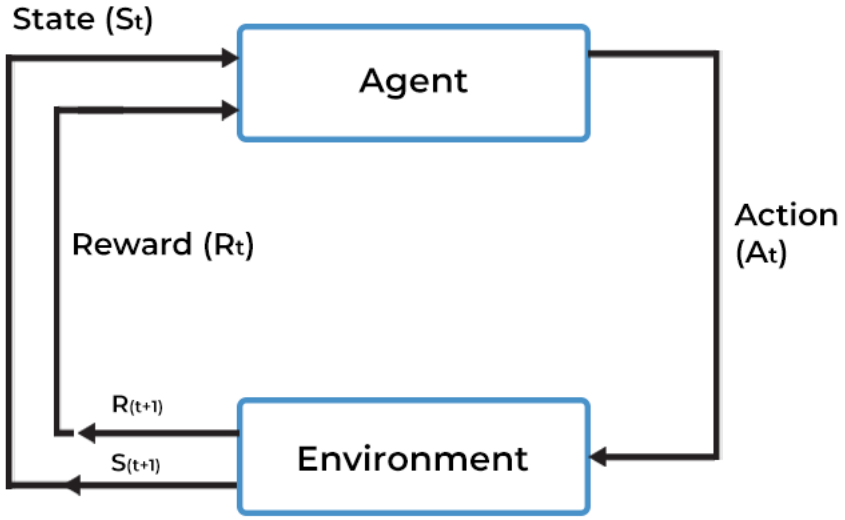
Introduction to MDP

Markov decision processes formally describe an environment for reinforcement learning

Where the environment is fully observable, i.e. The current state completely characterizes the process

- Almost all RL problems can be formalized as MDPs, e.g.
  - Optimal control primarily deals with continuous MDPs
  - Partially observable problems can be converted into MDPs
  - Bandits are MDPs with one state

Agent Environment Interface

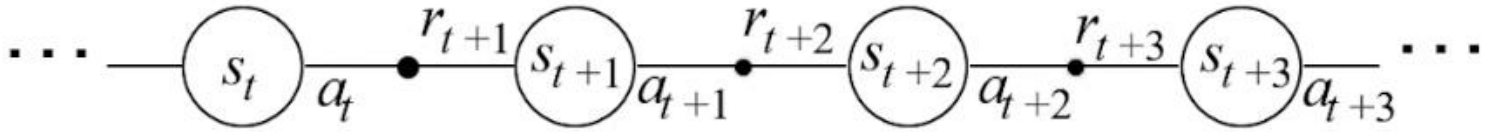


Assumption: Interaction between the agent and the environment happens at discrete time steps: t = 0, 1, 2, …

What is the maximum duration a tic-tac-toe experiment can run for?

- Agent observes state s_t at time t: $s_t \in S$
- Agent takes action at a_t time t: $a_t \in A$
- Agent gets a reward: $r_{t+1} \in \mathbb{R}$
- Agent reaches the next state: $s_{t+1} \in S$

Trajectory:



Markov Property

- “the state” at time t, means whatever information about the environment that is available to the agent at time t.
- The state can include immediate observations, highly processed observations, and structures built over time from a sequence of observations.
- Ideally, a state should summarize past observations so as to retain all essential information.
- “The future is independent of the past given the present”

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

  - The state captures all relevant information from the history
  - Once the state is known, the history may be thrown away
  - i.e. The state is a sufficient statistic of the future

Reinforcement Learning Fundamentals

Lecture 10: Markov Decision Process (MDP)

Dr Sandeep Manjanna
Assistant Professor, Plaksha University
sandeep.manjanna@plaksha.edu.in

In today's class…

- Markov Process
- Markov Reward Process
- Markov Decision Process (MDP)
- Problem to formulation
- Examples

Schedule for Evaluation

Date	Evaluation	Description
2/16/2024	Finalizing Project	Finalize Team, and Project topic
2/23/2024	Project Proposal (5%)	2-page report for Project Proposal. This document is expected to include following but not limited to: • 1/2 page for introduction and related work, 1 page for the problem and the proposed work, 1/4 page for proposed evaluation, 1/4 page for references. Format will be shared with you.
2/28/2024	Quiz 2	
3/15/2024	In-class Exam 1 (10%)	
3/25/2024	Mid Term Progress Report (5%)	4-page report for Mid-term Progress Report. This document is expected to include following but not limited to: • The first two pages contain a copy of your project proposal. The remaining pages include: a status update, presenting what you have accomplished so far (include figures and results), and 1/4 page describing your next steps.
Mar 28th and 29th	Mid Term Presentation (5%)	Progress presentation
4/3/2024	Quiz 3	
4/24/2024	Quiz4	
5/3/2024	Final Project Submission (25%)	Include full code in git hub, 6 page report, multi-media with demo if required. The format for the report will be provided later
May 6th to 10th	Final Project Presentation (10%)	Final Presentation Details will be shared Later
May	In-class Exam 2 (20%)	

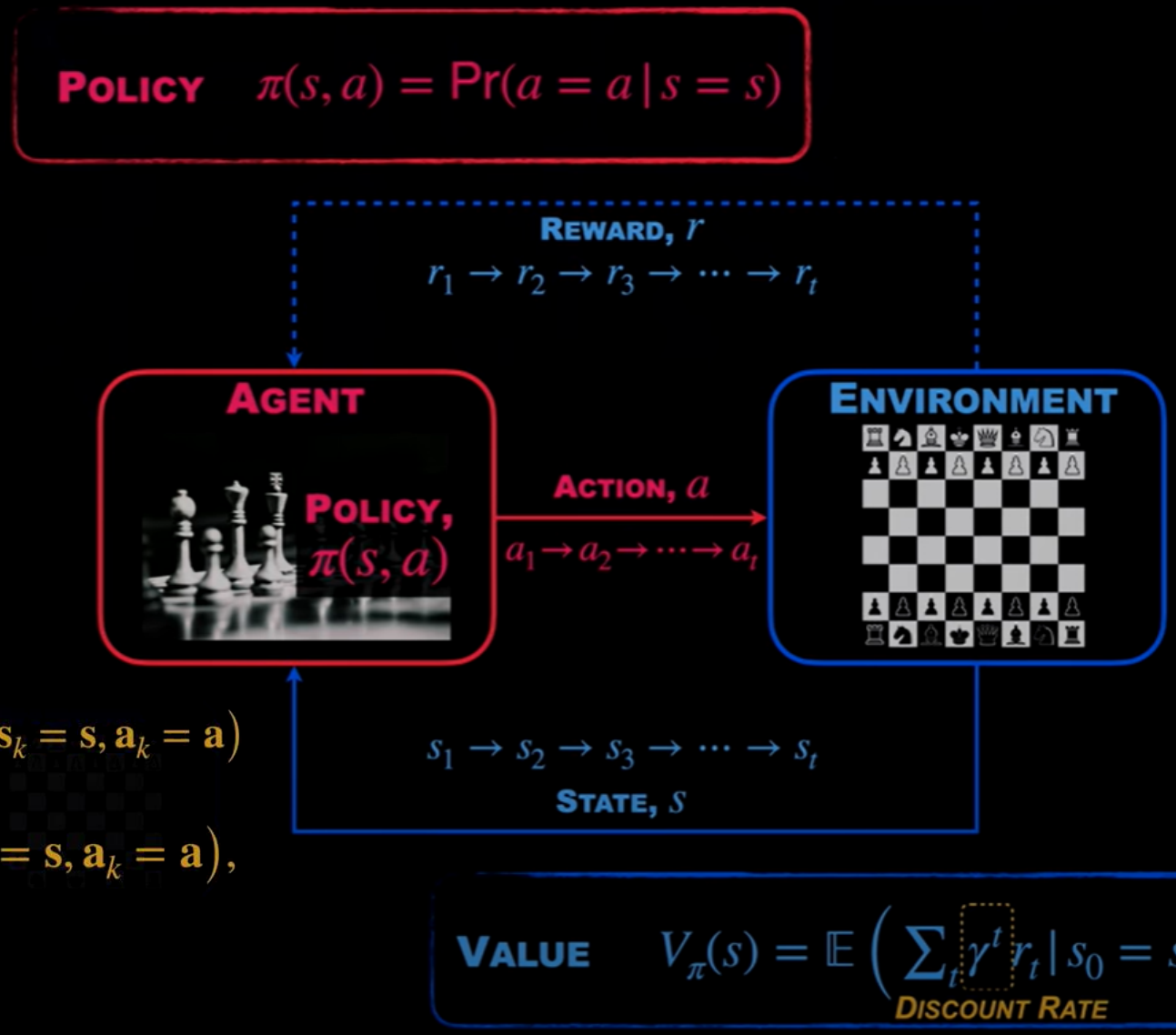
Inside an RL Agent
Model

- A model predicts what the environment will do next
- $\mathcal{P}$ predicts the next state
- $\mathcal{R}$ predicts the next (immediate) reward, e.g.

Transition Model → $\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$

Reward Function / Return → $\mathcal{R}^a_s = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$

Inside an RL Agent



$$R(\mathbf{s}', \mathbf{s}, \mathbf{a}) = \mathbf{Pr}\left(\mathbf{r}_{k+1} \mid \mathbf{s}_{k+1} = \mathbf{s}', \mathbf{s}_k = \mathbf{s}, \mathbf{a}_k = \mathbf{a}\right)$$

$$P(\mathbf{s}', \mathbf{s}, \mathbf{a}) = \mathbf{Pr}\left(\mathbf{s}_{k+1} = \mathbf{s}' \mid \mathbf{s}_k = \mathbf{s}, \mathbf{a}_k = \mathbf{a}\right),$$

Dr Steve Brunton's course : Reinforcement Learning.

Markov Property

- “the state” at time t, means whatever information about the environment that is available to the agent at time t.
- The state can include immediate observations, highly processed observations, and structures built over time from a sequence of observations.
- Ideally, a state should summarize past observations so as to retain all essential information.
- “The future is independent of the past given the present”

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

  - The state captures all relevant information from the history
  - Once the state is known, the history may be thrown away
  - i.e. The state is a sufficient statistic of the future

State Transition Matrix

For a Markov state s and successor state s', the *state transition probability* is defined by

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

State transition matrix $\mathcal{P}$ defines transition probabilities from all states s to all successor states s',

$$\mathcal{P} = \begin{matrix} \\ \textit{from} \end{matrix} \begin{matrix} \textit{to} \\ \end{matrix} \begin{bmatrix} \mathcal{P}_{11} & \dots & \mathcal{P}_{1n} \\ \vdots & & \\ \mathcal{P}_{n1} & \dots & \mathcal{P}_{nn} \end{bmatrix}$$

where each row of the matrix sums to 1.

Slide from David Silver's course: "Reinforcement Learning."

Markov Process

A Markov process is a memoryless random process, i.e. a sequence of random states $S_1, S_2, \ldots$ with the Markov property.

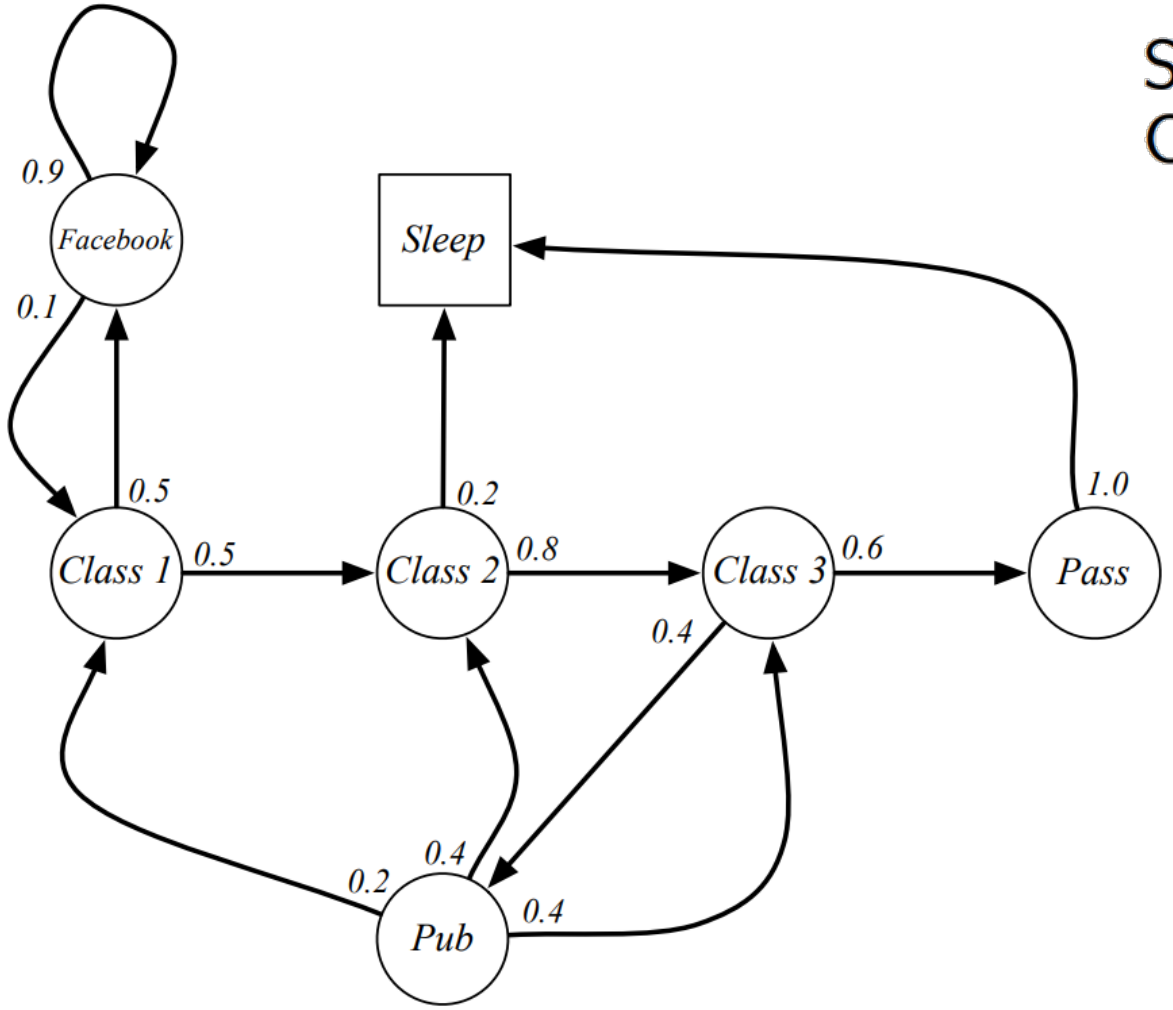
Definition

A *Markov Process* (or *Markov Chain*) is a tuple $\langle \mathcal{S}, \mathcal{P} \rangle$

- $\mathcal{S}$ is a (finite) set of states
- $\mathcal{P}$ is a state transition probability matrix,
  $\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$

Slide from David Silver's course: "Reinforcement Learning."

Markov Process Example



Sample episodes for Student Markov Chain starting from $S_1 = \text{C1}$

$$S_1, S_2, ..., S_T$$

- C1 C2 C3 Pass Sleep
- C1 FB FB C1 C2 Sleep
- C1 C2 C3 Pub C2 C3 Pass Sleep
- C1 FB FB C1 C2 C3 Pub C1 FB FB FB C1 C2 C3 Pub C2 Sleep

Slide from David Silver's course: "Reinforcement Learning."

Markov Reward Process Example

A Markov reward process is a Markov chain with values.

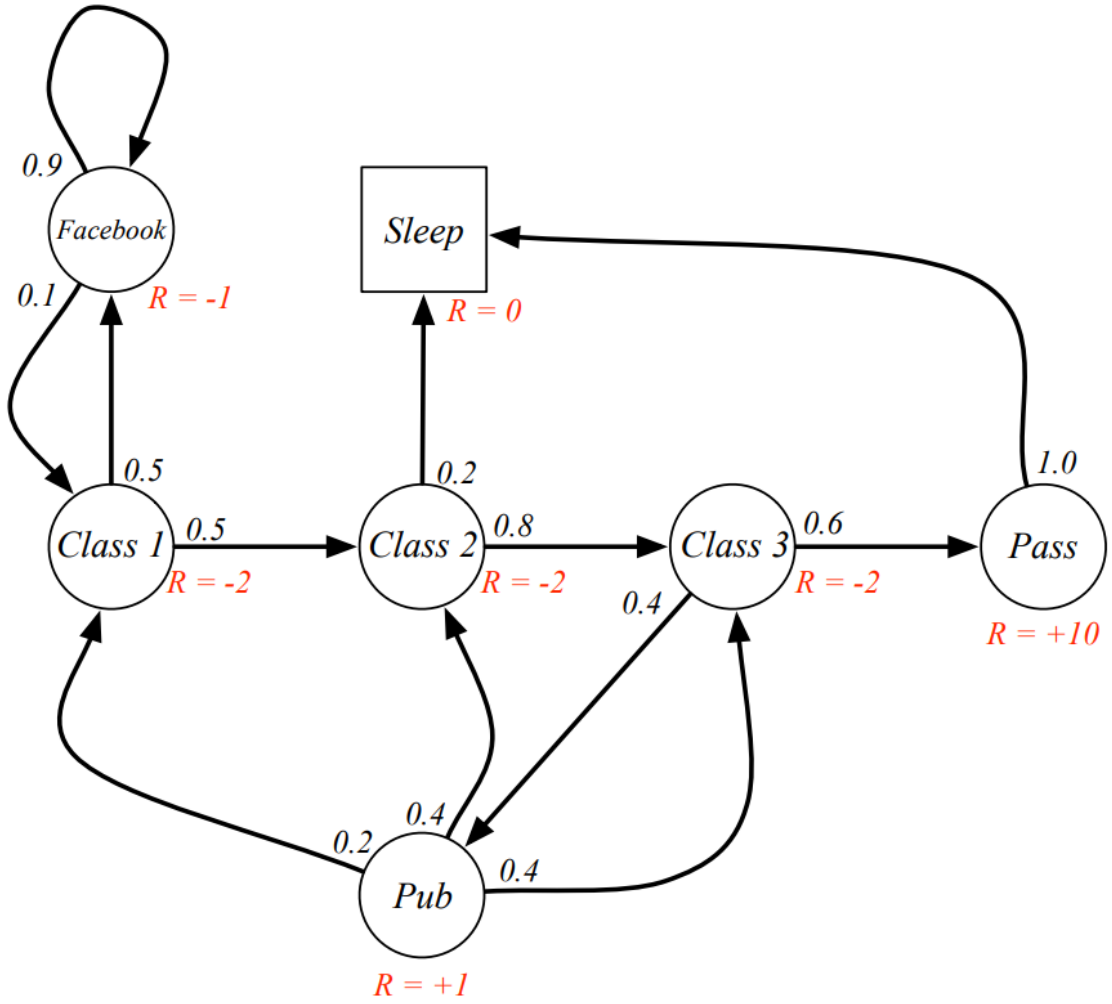
Definition

A *Markov Reward Process* is a tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$

- $\mathcal{S}$ is a finite set of states
- $\mathcal{P}$ is a state transition probability matrix, $\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$
- $\mathcal{R}$ is a reward function, $\mathcal{R}_s = \mathbb{E}\left[R_{t+1} \mid S_t = s\right]$
- γ is a discount factor, $\gamma \in [0, 1]$

Slide from David Silver's course: "Reinforcement Learning."

Markov Reward Process Example



What will be the return for each of these samples?

- C1 C2 C3 Pass Sleep
- C1 FB FB C1 C2 Sleep
- C1 C2 C3 Pub C2 C3 Pass Sleep
- C1 FB FB C1 C2 C3 Pub C1 FB FB FB C1 C2 C3 Pub C2 Sleep

Slide from David Silver's course: "Reinforcement Learning."

Markov Decision Process

- MDP, *M*, is the tuple: $M = \langle S, A, p, r \rangle$
  - S : set of states.
  - A : set of actions.
  - $p : S \times A \times S \to [0,1]$: probability of transition.
  - $r : S \times A \times S \to \mathbb{R}$: expected reward.
- Policy: $\pi : S \times A \to [0,1]$ (can be deterministic)
- Maximize total expected reward
- Learn an *optimal* policy

$$\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$

$$\mathcal{R}^a_s = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$$

How to compute the expected reward?

1. Discrete distribution over r:
2. R is from Real numbers:

Slide from Prof. Ravindran's course: "Reinforcement Learning."

Markov Decision Process

- MDP, *M*, is the tuple: $M = \langle S, A, p, r \rangle$
  - S : set of states.
  - A : set of actions.
  - $p : S \times A \times S \to [0, 1]$: probability of transition.
  - $r : S \times A \times S \to \mathbb{R}$: expected reward.
- Policy: $\pi : S \times A \to [0,1]$ (can be deterministic)
- Maximize total expected reward
- Learn an *optimal* policy

***π (a|s)* or *π (s,a)* = ?**

What does it mean for policy to be deterministic?

Slide from Prof. Ravindran's course: "Reinforcement Learning."

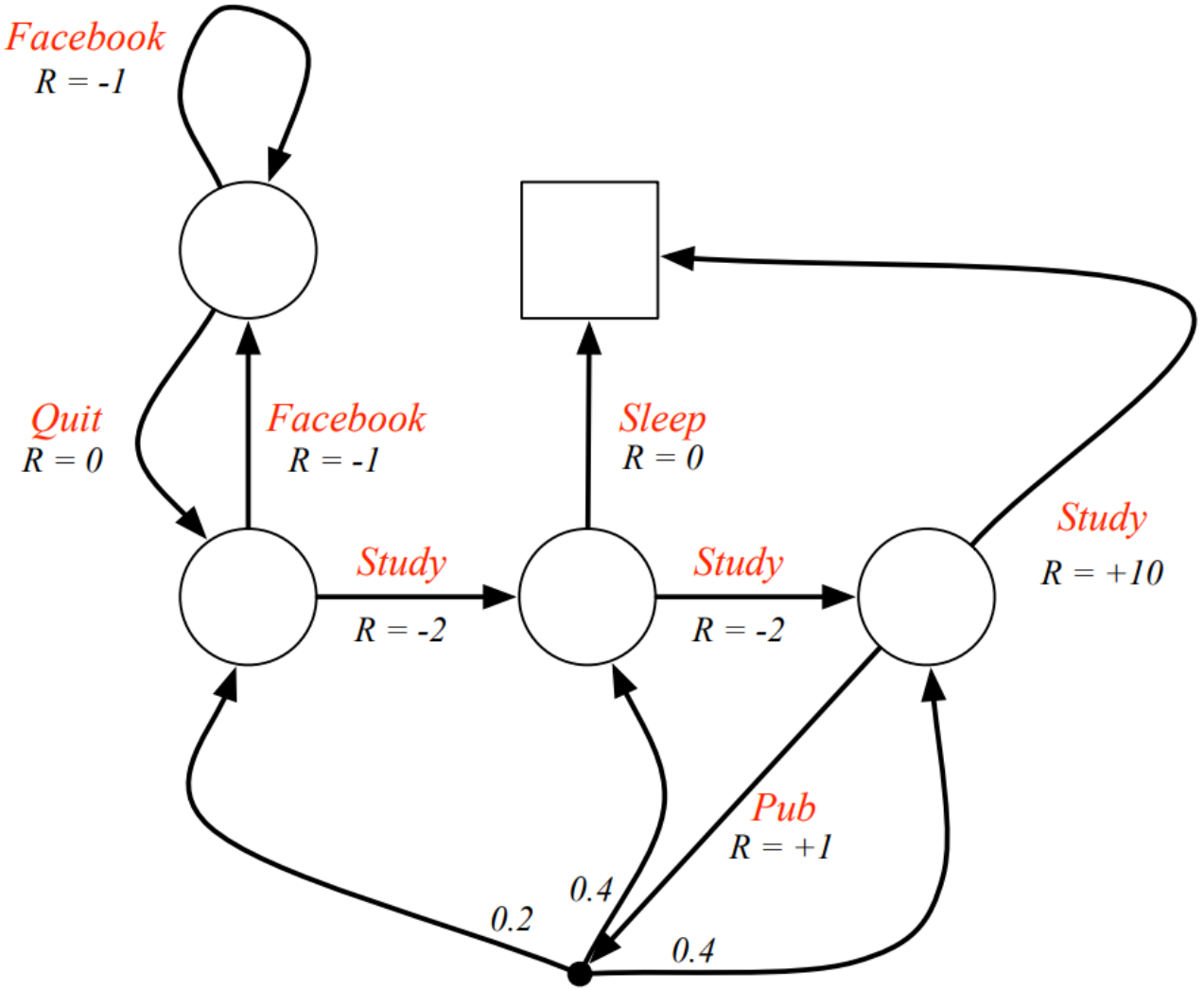
Markov Decision Process

- MDP, *M*, is the tuple: $M = \langle S, A, p, r \rangle$
  - S : set of states.
  - A : set of actions.
  - $p : S \times A \times S \to [0, 1]$: probability of transition.
  - $r : S \times A \times S \to \mathbb{R}$: expected reward.
- Policy: $\pi : S \times A \to [0,1]$ (can be deterministic)
- Maximize total expected reward
- Learn an *optimal* policy

The policy that achieves the maximum total expected reward is called Optimal Policy.

Slide from Prof. Ravindran's course: "Reinforcement Learning."

Markov Reward Process Example



Slide from David Silver's course: "Reinforcement Learning."

Formulating an RL Problem

- States
  - –Enough information to take decisions
  - –Raw inputs often not sufficient
- Actions
  - –The control variables
  - –Discrete – items to recommend, moves in a game
  - –Continuous – torque to a motor
- Rewards
  - –Define the *goal* of the problem

States must follow Markov Property

Different levels of controls in learning to drive example.

Slide from Prof. Ravindran's course: "Reinforcement Learning."

Reinforcement Learning Fundamentals

Lecture 11: MDP Returns and Value Function

Dr Sandeep Manjanna
Assistant Professor, Plaksha University
sandeep.manjanna@plaksha.edu.in

In today's class…

- In-class presentations : Example MDP formulations
- Returns
- Value function

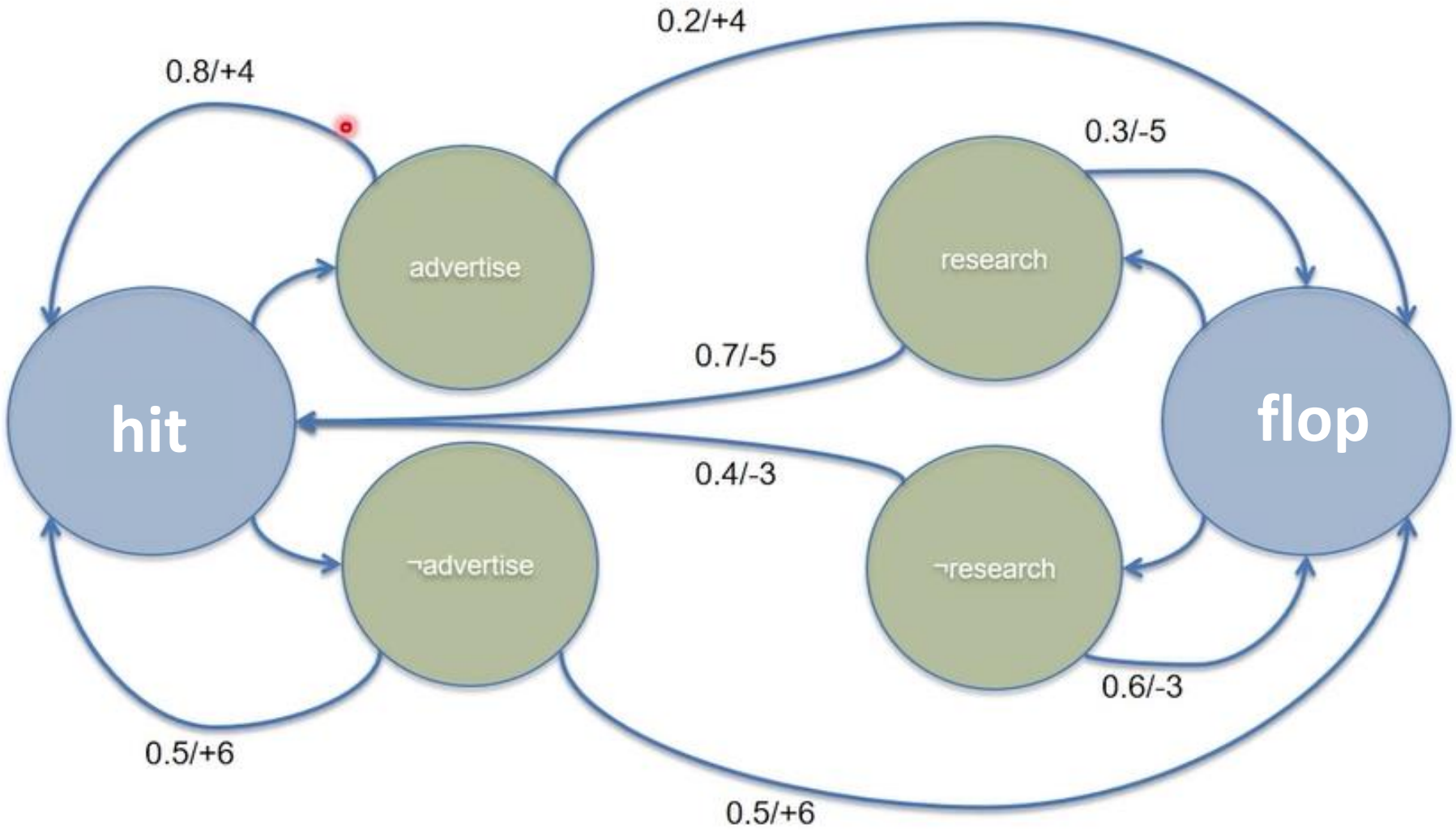
MDP Formulation

1. A web series produced by Netflix can either be a hit, or a flop. If the web series is a hit, the Netflix can advertise that show to get an immediate reward of 4 additional units. On doing so, the show continues to be a hit with a probability of 0.8. However, the show can become a flop with a probability 0.2. Netflix can alternately decide not to advertise a hit show yielding a saving of 6 units, with a 0.5 probability of the show continuing to be a hit.

   In case the show is a flop in the beginning, Netflix has an option to perform audience study to improve the viewership of the show. This investment in the study will cost the company 5 units and with a probability of 0.7, the audience study will lead to the show becoming a hit. Finally, if the company does not perform the study for the flop show, it receives an immediate cost of 3 units, with the show continuing to being a flop with a probability of 0.6.

MDP Formulation

1.



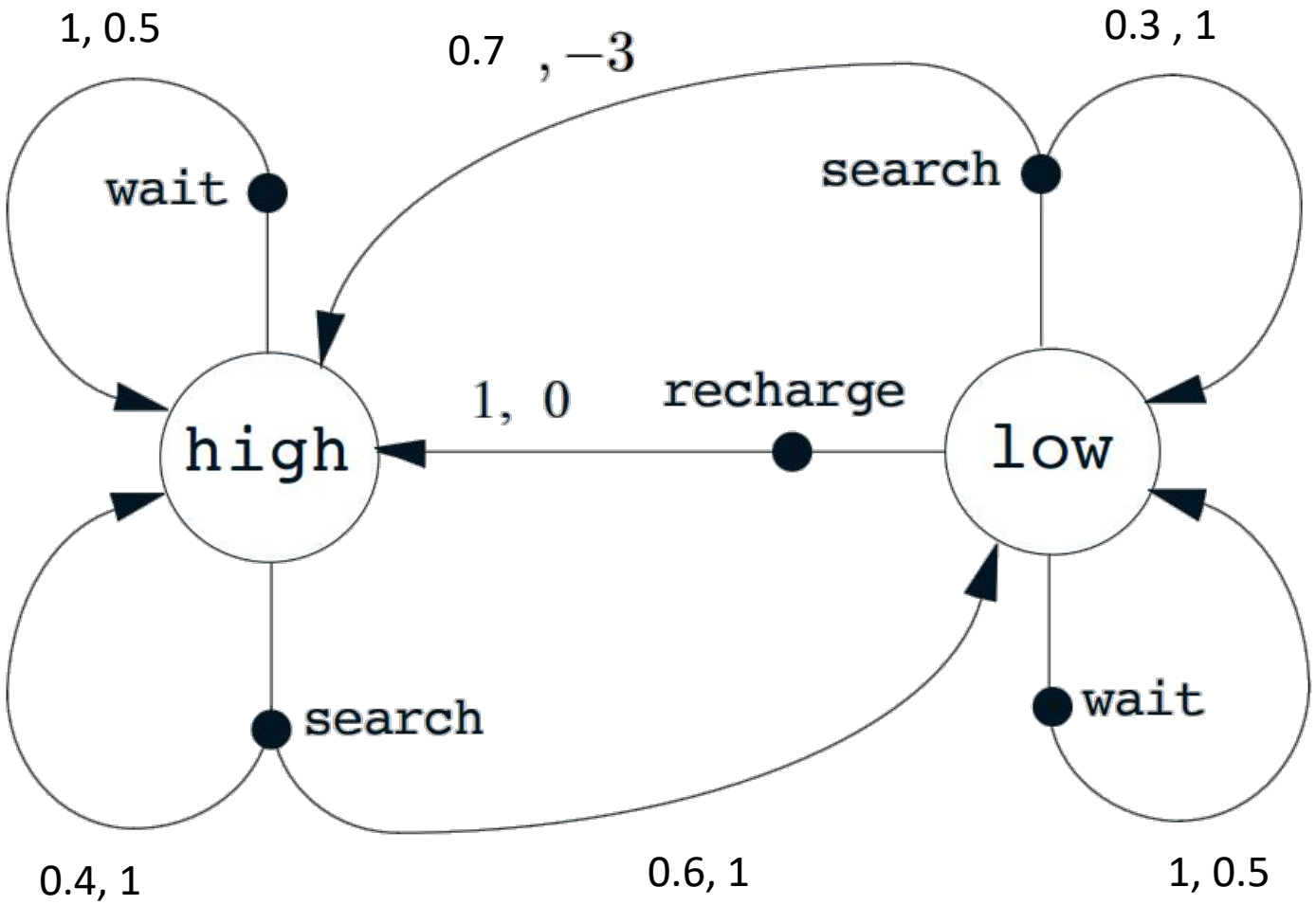
MDP Formulation

2. A mobile robot has the job of collecting empty coffee mugs in an office environment. It has sensors for detecting mugs, 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 mugs are made by a reinforcement learning agent based on the current charge levels (high or low) of the battery.

   The robot can wait for someone to bring the mug to it, in which case it gains an appreciation of 0.5 units. Robot with high battery level can decide to search for the mugs, receive an appreciation of 1 unit, and end up in with high battery level with a probability of 0.4. However, with a probability of 0.6, the robot's battery level goes to low. Once the robot battery is low, it can either go to the charging station to recharge its batteries, or it can decide to go searching for the mugs. But, in this case, with a probability of 0.7, the robot will drain out all the batteries and need to be carried by a human being to get the batteries charged and the robot is penalized by 3 units. And with a probability of 0.3, the robot will successfully search the mug and gain an appreciation of 1 unit.

MDP Formulation

2.



MDP Formulation

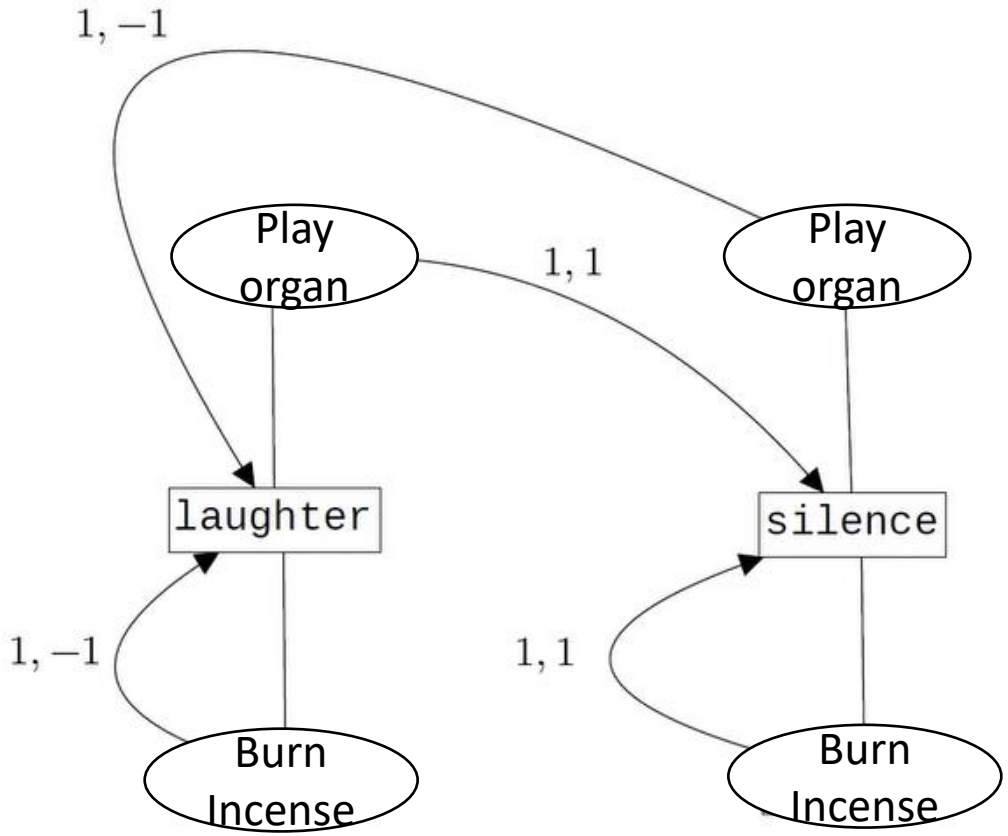
3. Dear Friend,

   Some time ago, I bought this old house, but found it to be haunted by ghostly sardonic laughter. As a result it is hardly habitable. There is hope, however, for by actual testing I have found that this haunting is subject to certain laws, obscure but infallible, and that the laughter can be affected by my playing the organ or burning incense. In each minute, the laughter occurs or not, it shows no degree. What it will do during the ensuing minute depends, in the following exact way, on what has been happening during the preceding minute: Whenever there is laughter, it will continue in the succeeding minute unless I play the organ, in which case it will stop. But continuing to play the organ does not keep the house quiet. I notice, however, that whenever I burn incense when the house is quiet and do not play the organ it remains quiet for the next minute. At this minute of writing, the laughter is going on. Please tell me what manipulations of incense and organ I should make to get that house quiet, and to keep it so.

   Sincerely,
   At Wits End

MDP Formulation

3.



More real-world examples can be found here:
https://towardsdatascience.com/real-world-applications-of-markov-decision-process-mdp-a39685546026

We want to learn a policy …

Policy at step t, π_t :

a mapping from states to action probabilities

$\pi_t(s, a) =$ probability that $a_t = a$ when $s_t = s$

- Reinforcement learning methods specify how the agent changes its policy as a result of experience.
- Roughly, the agent's goal is to get as much reward as it can over the long run.

Slide from Prof. Ravindran's course: "Reinforcement Learning."

Returns

Suppose the sequence of rewards after step t is :

$$r_{t+1}, r_{t+2}, r_{t+3}, \cdots$$

What do we want to maximize?

Slide from Prof. Ravindran's course: "Reinforcement Learning."

Returns

Suppose the sequence of rewards after step t is :

$$r_{t+1}, r_{t+2}, r_{t+3}, \cdots$$

What do we want to maximize?

We want to maximize the **return**, G_t, for each step t.

Episodic tasks: interaction breaks naturally into episodes, e.g., plays of a game, trips through a maze.

$$G_t = r_{t+1} + r_{t+2} + \cdots + r_T,$$

where T is a final time step at which a **terminal state** is reached, ending an episode.

Slide from Prof. Ravindran's course: "Reinforcement Learning."

Returns

Continuing tasks: interaction does not have natural episodes.

Discounted return:

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

where $\gamma, 0 \le \gamma \le 1$, is the **discount rate**.

shortsighted $0 \leftarrow \gamma \rightarrow 1$ farsighted

Will this become an Immediate RL problem?

In general,

we want to maximize the **expected return**, $E\{G_t\}$, for each step t.

Slide from Prof. Ravindran's course: "Reinforcement Learning."

Why discount?

Most Markov reward and decision processes are discounted. Why?

- Mathematically convenient to discount rewards
- Avoids infinite returns in cyclic Markov processes
- Uncertainty about the future may not be fully represented
- If the reward is financial, immediate rewards may earn more interest than delayed rewards
- Animal/human behavior shows preference for immediate reward
- It is sometimes possible to use undiscounted Markov reward processes (i.e. $\gamma = 1$), e.g. if all sequences terminate.

Slide from David Silver's course: "Reinforcement Learning."

Value Function

- Expected future rewards starting from a state (or state-action pair) and following policy π

State - value function for policy π :

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

Slide from Prof. Ravindran's course: "Reinforcement Learning."

Value Function

- Expected future rewards starting from a state (or state-action pair) and following policy π

State - value function for policy π :

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

Action - value function for policy π :

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

Slide from Prof. Ravindran's course: "Reinforcement Learning."

Value Function

- Expected future rewards starting from a state (or state-action pair) and following policy π

State - value function for policy π :

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

Action - value function for policy π :

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

$$v_\pi(s) = \sum_a \pi(a|s) q_\pi(s,a)$$

Slide from Prof. Ravindran's course: "Reinforcement Learning."

Reinforcement Learning Fundamentals

Lecture 12: Bellman Equation

Dr Sandeep Manjanna
Assistant Professor, Plaksha University
sandeep.manjanna@plaksha.edu.in

In today's class…

- Until now,
  - started with states and actions,
  - discrete time dynamics,
  - policies (action probability or mapping from states to actions),
  - MDPs,
  - Returns, and
  - State-Value function and Action-Value function.

- How to solve an MDP? How to use the value functions to solve MDP?

- Bellman Equation

Value Function

Definition

The *state-value function* $v_\pi(s)$ of an MDP is the expected return starting from state s, and then following policy π

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$$

Definition

The *action-value function* $q_\pi(s, a)$ is the expected return starting from state s, taking action a, and then following policy π

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

$$v_\pi(s) = \sum_a \pi(a|s) q_\pi(s, a)$$

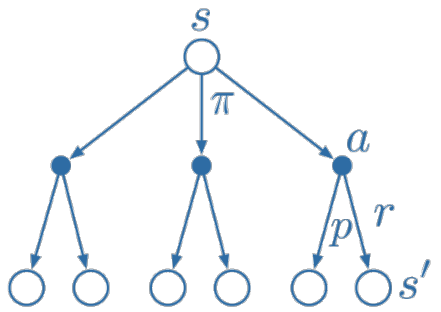
Bellman Equation

Richard E. Bellman

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r \,|\, s, a) \Big[r + \gamma \mathbb{E}_\pi[G_{t+1} | S_{t+1} = s'] \Big] \\
&= \sum_a \pi(a|s) \sum_{s', r} p(s', r \,|\, s, a) \Big[r + \gamma v_\pi(s') \Big], \quad \text{for all } s \in \mathcal{S},
\end{aligned}
$$

- 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.

Bellman Equation



Backup diagram for v_π

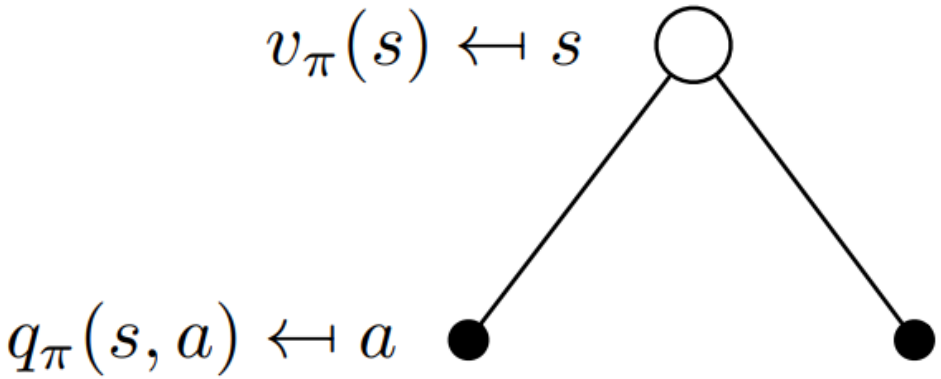
Unlike transition graphs, the state nodes of backup diagrams do not necessarily represent distinct states.

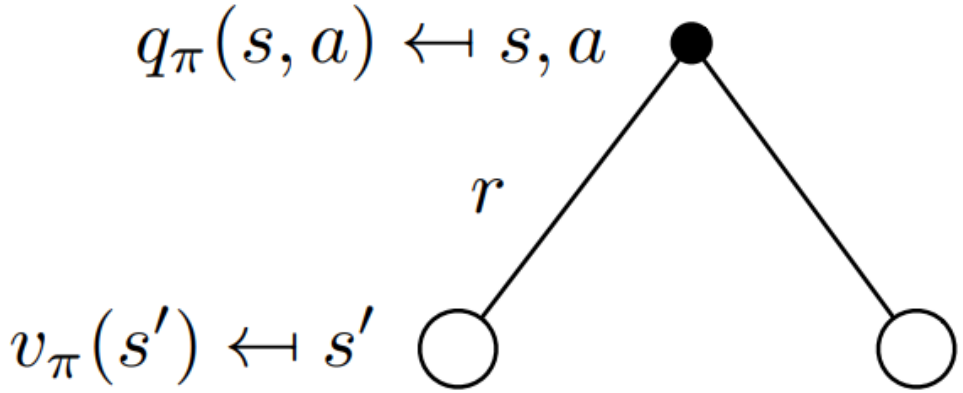
$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r \,|\, s, a) \Big[r + \gamma \mathbb{E}_\pi[G_{t+1} | S_{t+1} = s'] \Big] \\
&= \sum_a \pi(a|s) \sum_{s', r} p(s', r \,|\, s, a) \Big[r + \gamma v_\pi(s') \Big], \quad \text{for all } s \in \mathcal{S},
\end{aligned}
$$

- Linear equation in $|S|$ variables.
- Unique solution exists.

Slide from Prof. Ravindran's course: "Reinforcement Learning."

Bellman Equation





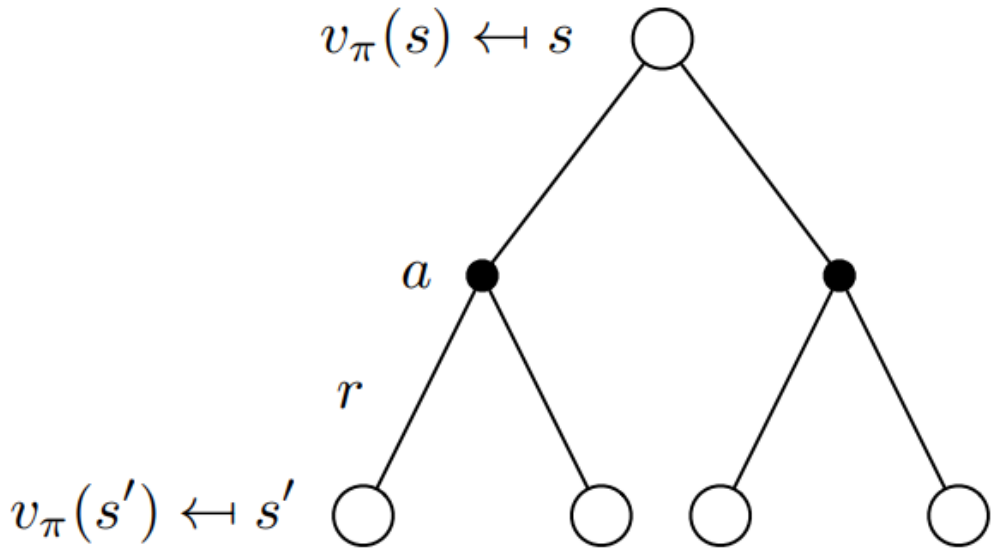
$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$$

$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$

Bellman Equation

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s, a)$$

$$q_\pi(s, a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$



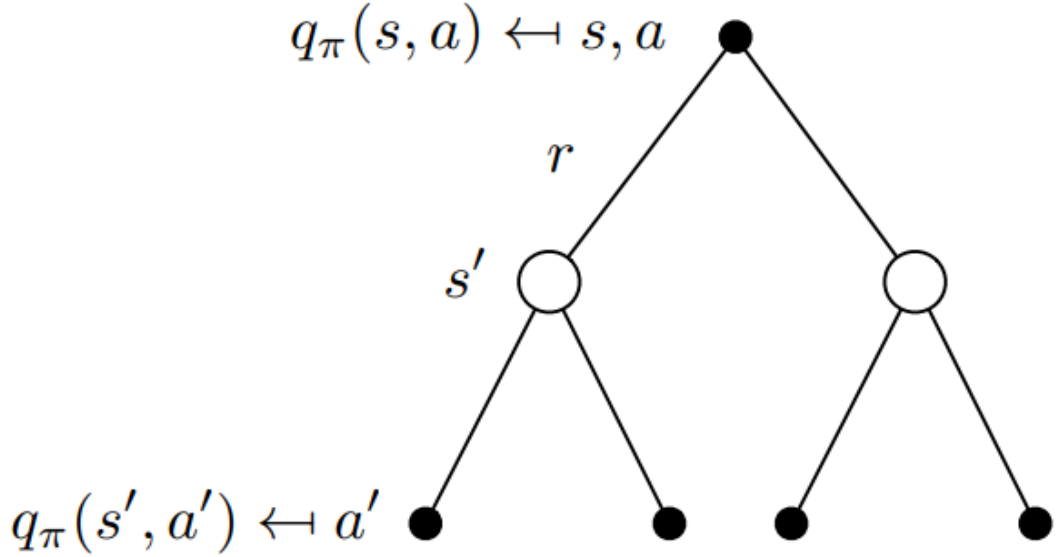
$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s') \right)$$

$$= \sum_a \pi(a|s) \sum_{s', r} p(s', r \,|\, s, a) \Big[r + \gamma v_\pi(s') \Big], \quad \text{for all } s \in \mathcal{S},$$

Bellman Equation

$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s,a)$$

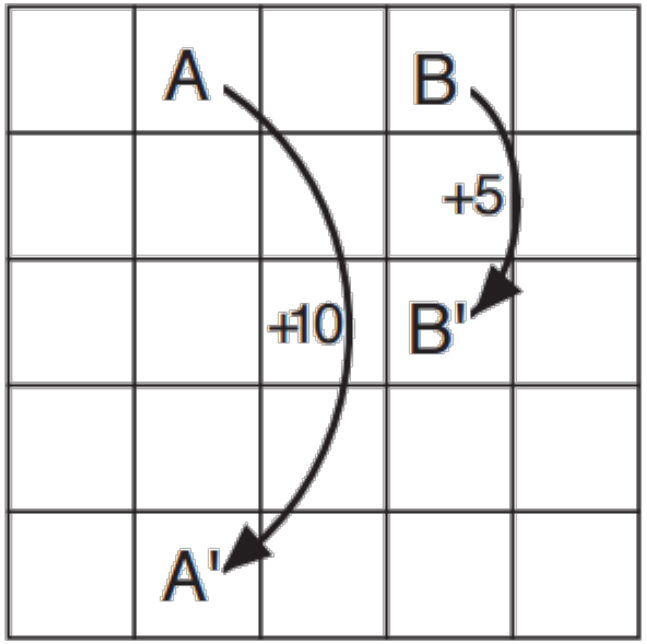
$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_\pi(s')$$

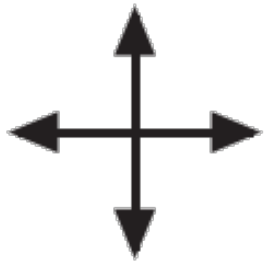


$$q_\pi(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \sum_{a' \in \mathcal{A}} \pi(a'|s') q_\pi(s',a')$$

Bellman Equation

State-value function
for equiprobable
random policy;
$\gamma = 0.9$





Actions

3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

- Actions that would take the agent off the grid leave its location unchanged, but also result in a rewa
- Other

$$= \sum_a \pi(a|s) \sum_{s',r} p(s', r \,|\, s, a) \Big[r + \gamma v_\pi(s') \Big], \quad \text{for all } s \in \mathcal{S},$$

s A and B.

Optimal Value Function

Definition

The *optimal state-value function* $v_*(s)$ is the maximum value function over all policies

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

The *optimal action-value function* $q_*(s, a)$ is the maximum action-value function over all policies

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

- The optimal value function specifies the best possible performance in the MDP.
- An MDP is "solved" when we know the optimal value fn.

Slide from Dr David Silver's course: "Reinforcement Learning."

Optimal Policy

Define a partial ordering over policies

$$\pi \geq \pi' \text{ if } v_\pi(s) \geq v_{\pi'}(s), \forall s$$

Theorem

For any Markov Decision Process

- *There exists an optimal policy π_* that is better than or equal to all other policies, $\pi_* \geq \pi, \forall \pi$*
- *All optimal policies achieve the optimal value function, $v_{\pi_*}(s) = v_*(s)$*
- *All optimal policies achieve the optimal action-value function, $q_{\pi_*}(s, a) = q_*(s, a)$*

Slide from Dr David Silver's course: "Reinforcement Learning."

Optimal Policy

An optimal policy can be found by maximising over $q_*(s, a)$,

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

- There is always a deterministic optimal policy for any MDP
- If we know $q_*(s, a)$, we immediately have the optimal policy

Slide from Dr David Silver's course: "Reinforcement Learning."

Reinforcement Learning Fundamentals

Lecture 13: Bellman Optimality

Dr Sandeep Manjanna
Assistant Professor, Plaksha University
sandeep.manjanna@plaksha.edu.in

Announcements

- There will be 4 quizzes in total and best 3 will be considered for grade calculation. **NO make-up quiz for any reason**.
- The Midterm project presentation will be online.
- There will be an in-class exam on the **15th of March.**

In today's class…

Until now…

- State and Action Value Functions
- Derived Bellman Equation

- Bellman Optimality: Optimal Value function and Optimal Policy
- Policy Extraction
- Value Iteration

Bellman Equation

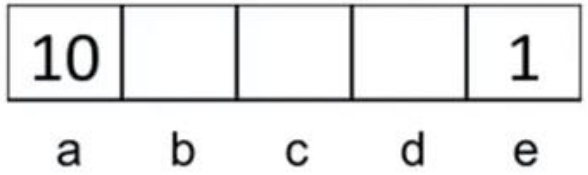
$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \Big[r + \gamma \mathbb{E}_\pi[G_{t+1} | S_{t+1} = s'] \Big] \\
&= \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) \Big[r + \gamma v_\pi(s') \Big], \quad \text{for all } s \in \mathcal{S},
\end{aligned}
$$

Richard E. Bellman

- The Bellman equation averages over all the possibilities, weighting each by its probability of occurring.
- Linear equation in $|S|$ variables.
- Unique solution exists.

Optimal Policy

- Given:



  - Actions: East, West, and Exit (only available in exit states a, e)
  - Transitions: deterministic

- Quiz 1: For $\gamma = 1$, what is the optimal policy?

- Quiz 2: For $\gamma = 0.1$, what is the optimal policy?

- Quiz 3: For which γ are West and East equally good when in state d?

Optimal Policy

Define a partial ordering over policies

$$\pi \geq \pi' \text{ if } v_\pi(s) \geq v_{\pi'}(s), \forall s$$

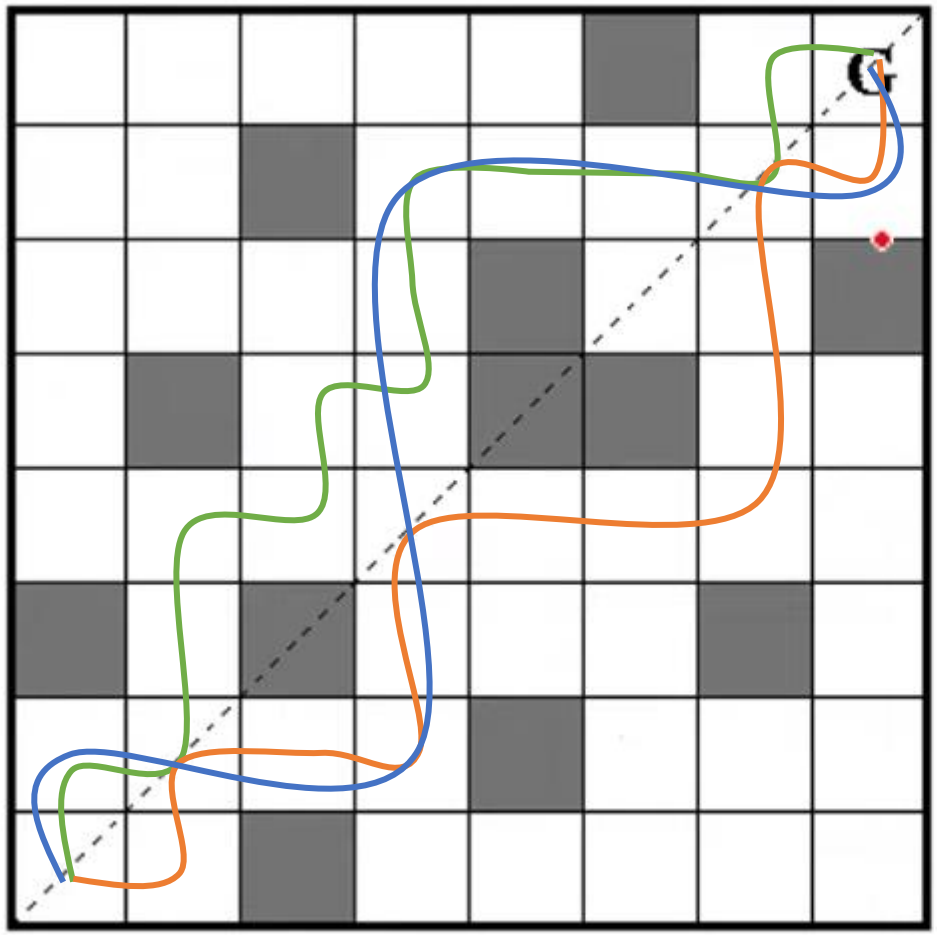
Theorem

For any Markov Decision Process

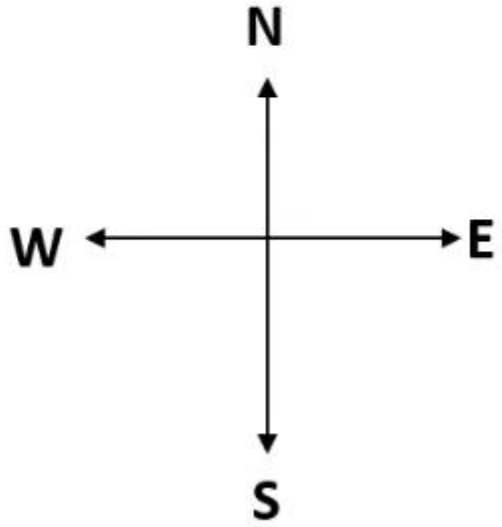
- *There exists an optimal policy π_* that is better than or equal to all other policies, $\pi_* \geq \pi, \forall \pi$*
- *All optimal policies achieve the optimal value function, $v_{\pi_*}(s) = v_*(s)$*
- *All optimal policies achieve the optimal action-value function, $q_{\pi_*}(s, a) = q_*(s, a)$*

Slide from Dr David Silver's course: "Reinforcement Learning."

Optimality Example



$$M = \langle S, A, p, r \rangle$$



Many optimal policies, but only one optimal value function!

Optimal Value Function

Definition

The *optimal state-value function* $v_*(s)$ is the maximum value function over all policies

$$v_*(s) = \max_\pi v_\pi(s) \quad \text{for all } s \in \mathcal{S}$$

The *optimal action-value function* $q_*(s, a)$ is the maximum action-value function over all policies

$$q_*(s, a) = \max_\pi q_\pi(s, a) \quad \text{for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A}$$

- The optimal value function specifies the best possible performance in the MDP.
- An MDP is "solved" when we know the optimal value fn.

Slide from Dr David Silver's course: "Reinforcement Learning."

Optimal Policy

An optimal policy can be found by maximising over $q_*(s, a)$,

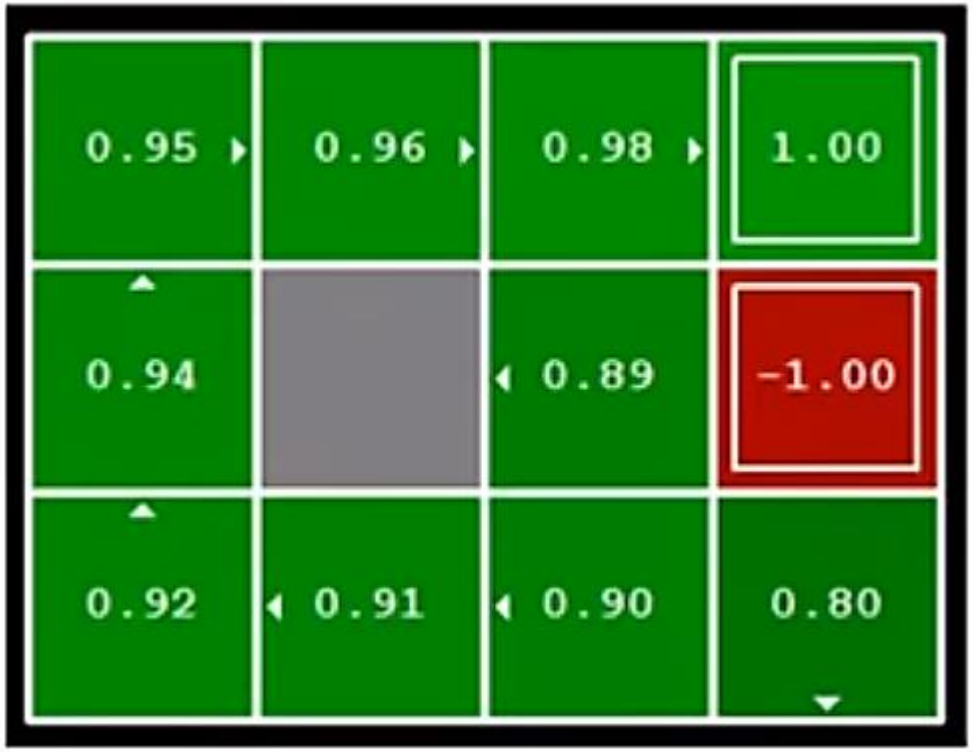
$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \underset{a \in \mathcal{A}}{\text{argmax}}\ q_*(s, a) \\ 0 & \textit{otherwise} \end{cases}$$

- There is always a deterministic optimal policy for any MDP
- If we know $q_*(s, a)$, we immediately have the optimal policy

- Optimal value function is unique for an MDP.
- Hence, many solution approaches try to find an optimal value function, instead of directly finding the optimal policy.

Why are these useful?

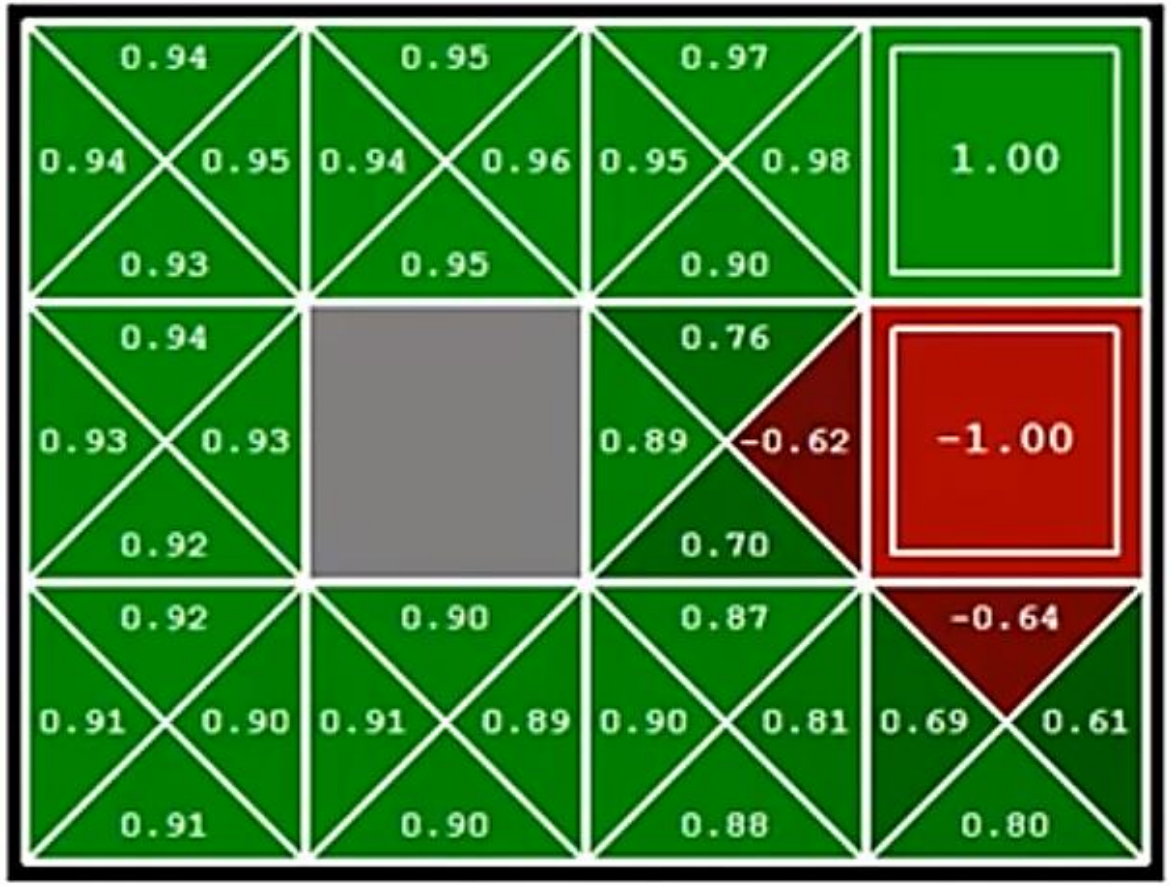
Given the optimal values v_*, how to get the optimal policy?



- We can use one-step-lookahead search to get the long-term optimal action.
- This is called **policy extraction**, since it gets the policy implied by the value.

Why are these useful?

Given the optimal q-values q_*,



- Don’t even need to do one-step-lookahead search.

Bellman Optimality Equation for v_*

$$v_\pi(s) = \sum_a \pi(a|s) \underbrace{\sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big]}_{q_\pi(s,a)}, \quad \text{for all } s \in \mathcal{S},$$

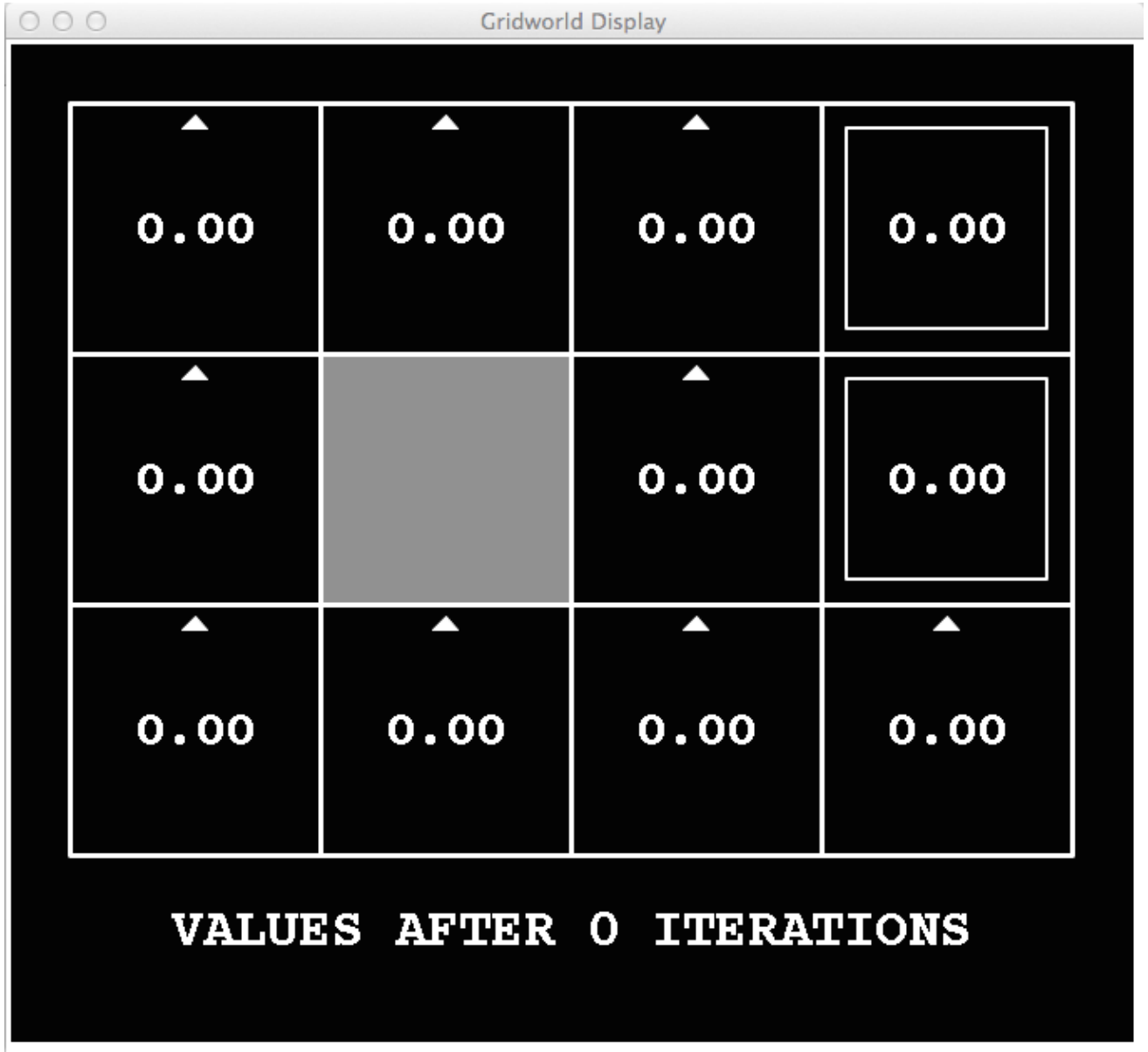
The value of a state under an optimal policy must equal the expected return for the best action from that state:

$$\begin{aligned} v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\ &= \max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma v_*(s')\big]. \end{aligned}$$

Similarly,

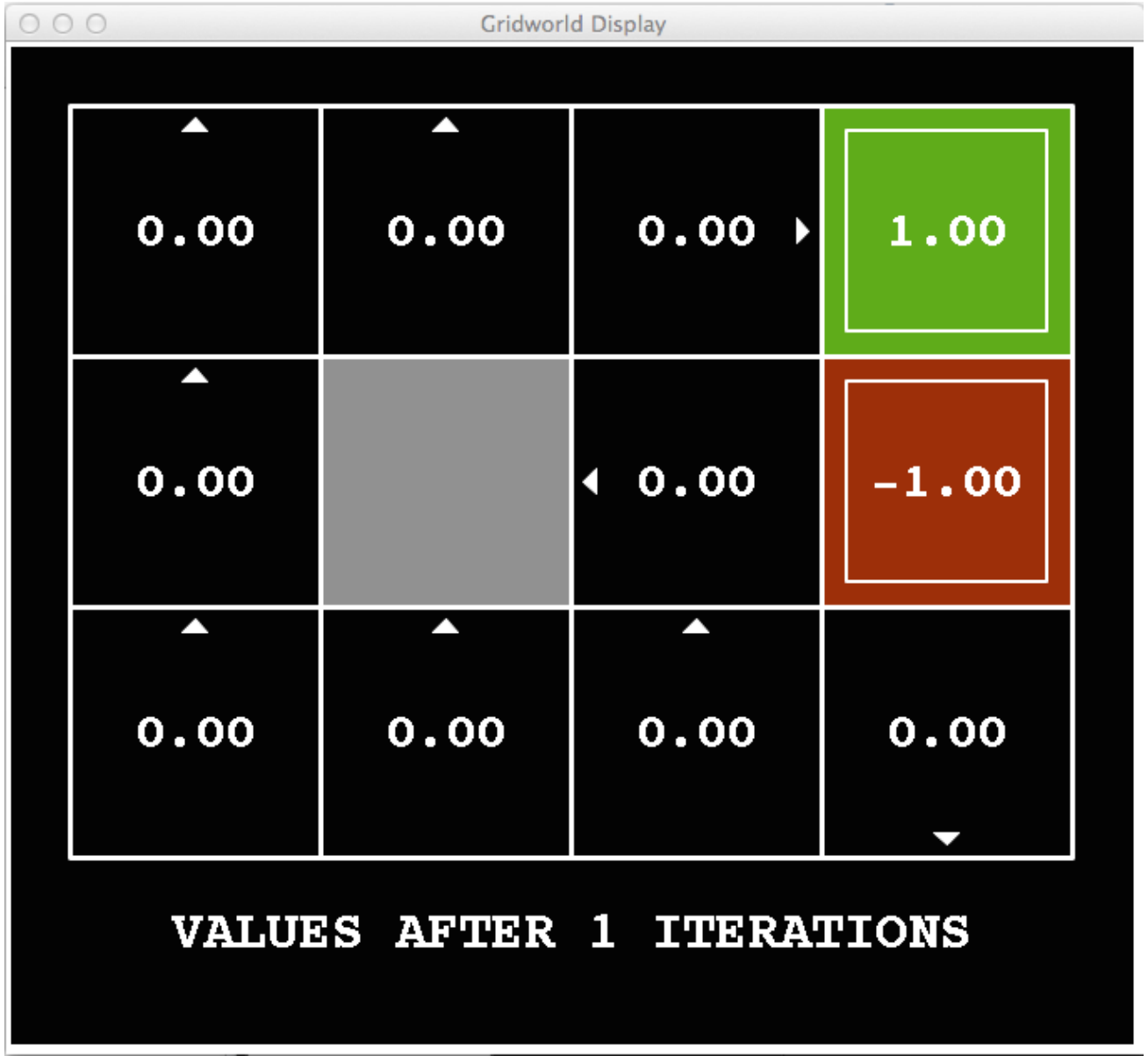
$$\begin{aligned} q_*(s,a) &= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1},a') \;\Big|\; S_t = s, A_t = a\Big] \\ &= \sum_{s',r} p(s',r|s,a)\Big[r + \gamma \max_{a'} q_*(s',a')\Big]. \end{aligned}$$

Value Iteration



Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration



Noise = 0.2
Discount = 0.9
Living reward = 0

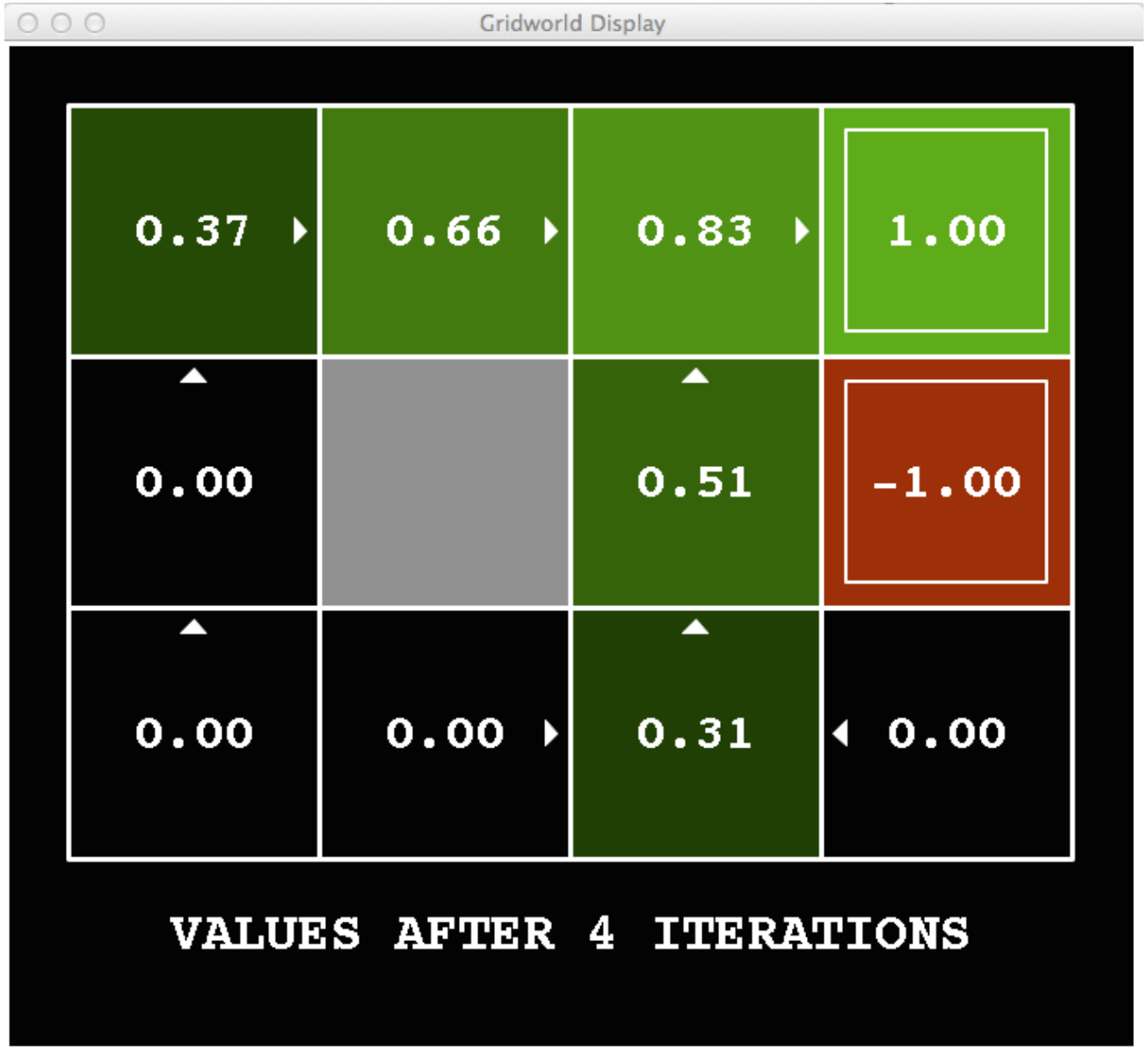
Value Iteration

Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration

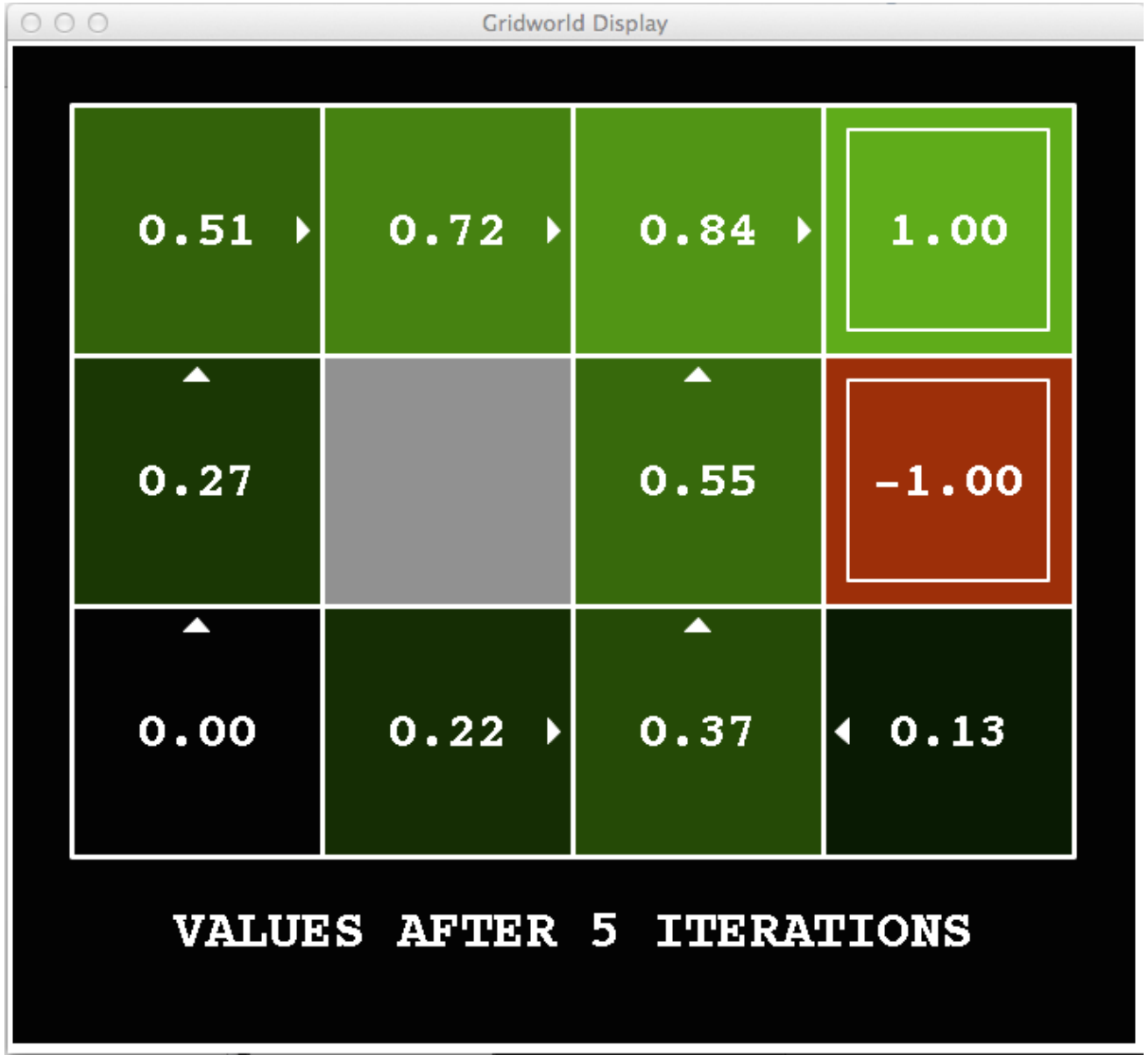
Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration



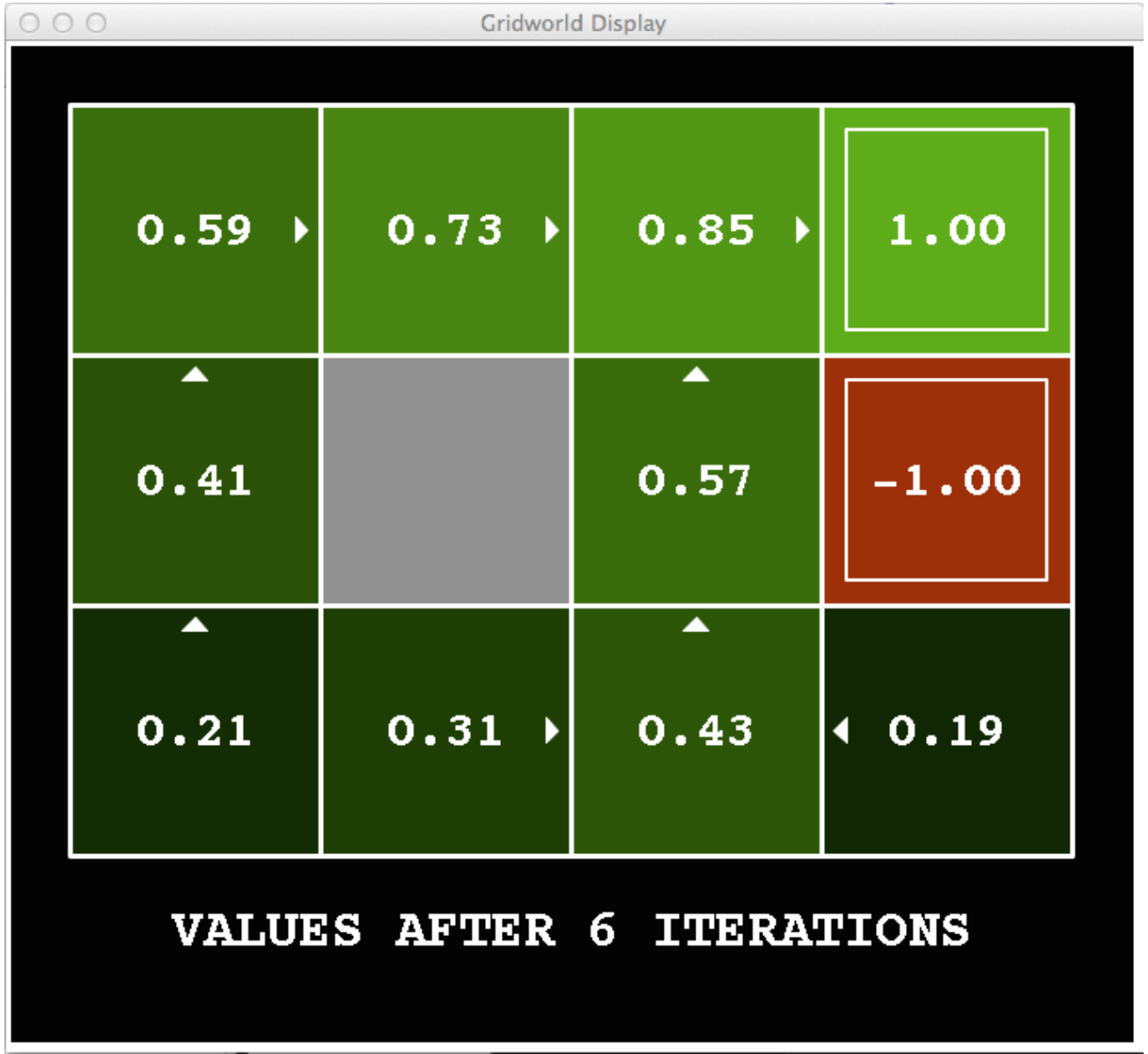
Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration



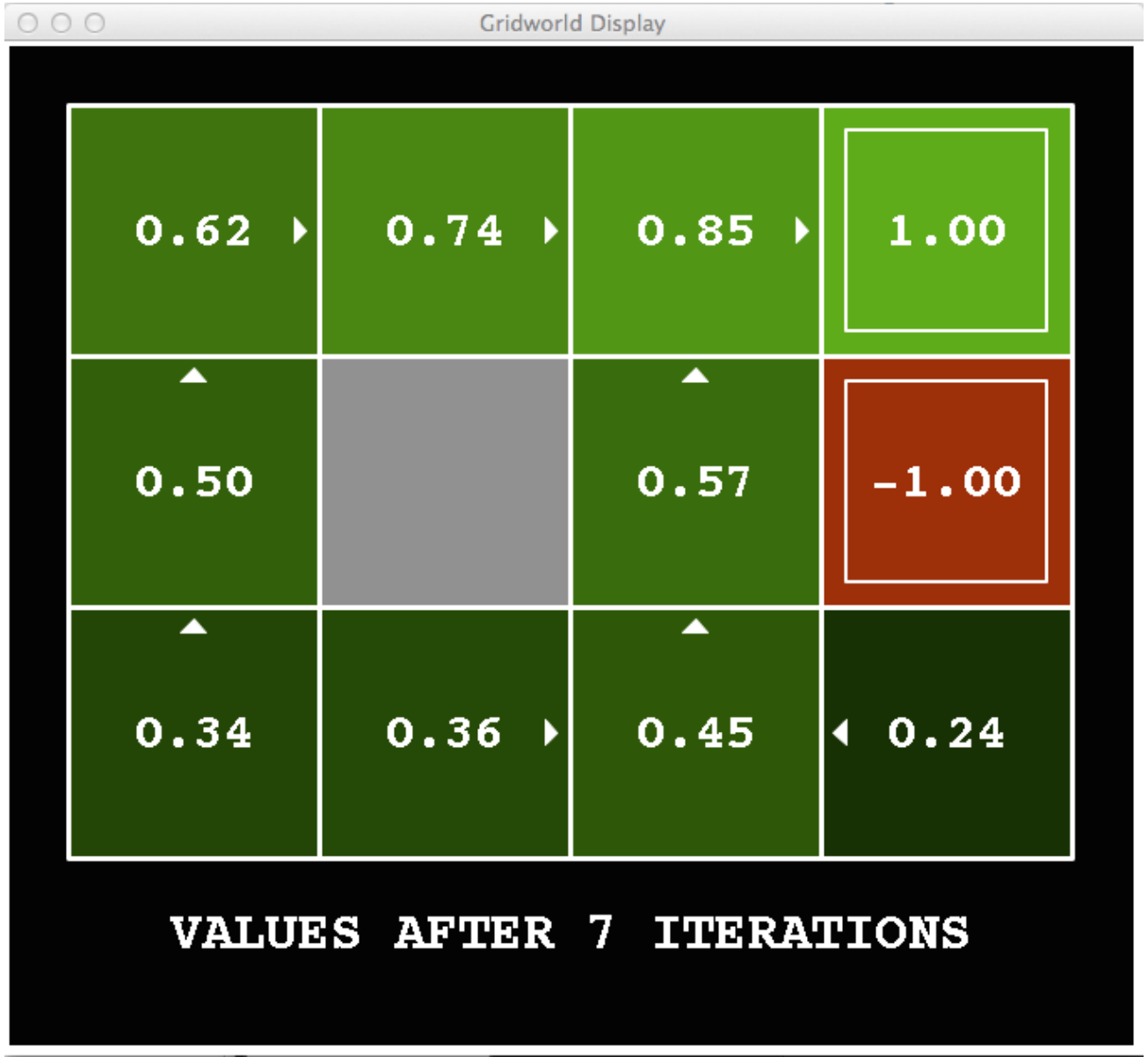
Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration



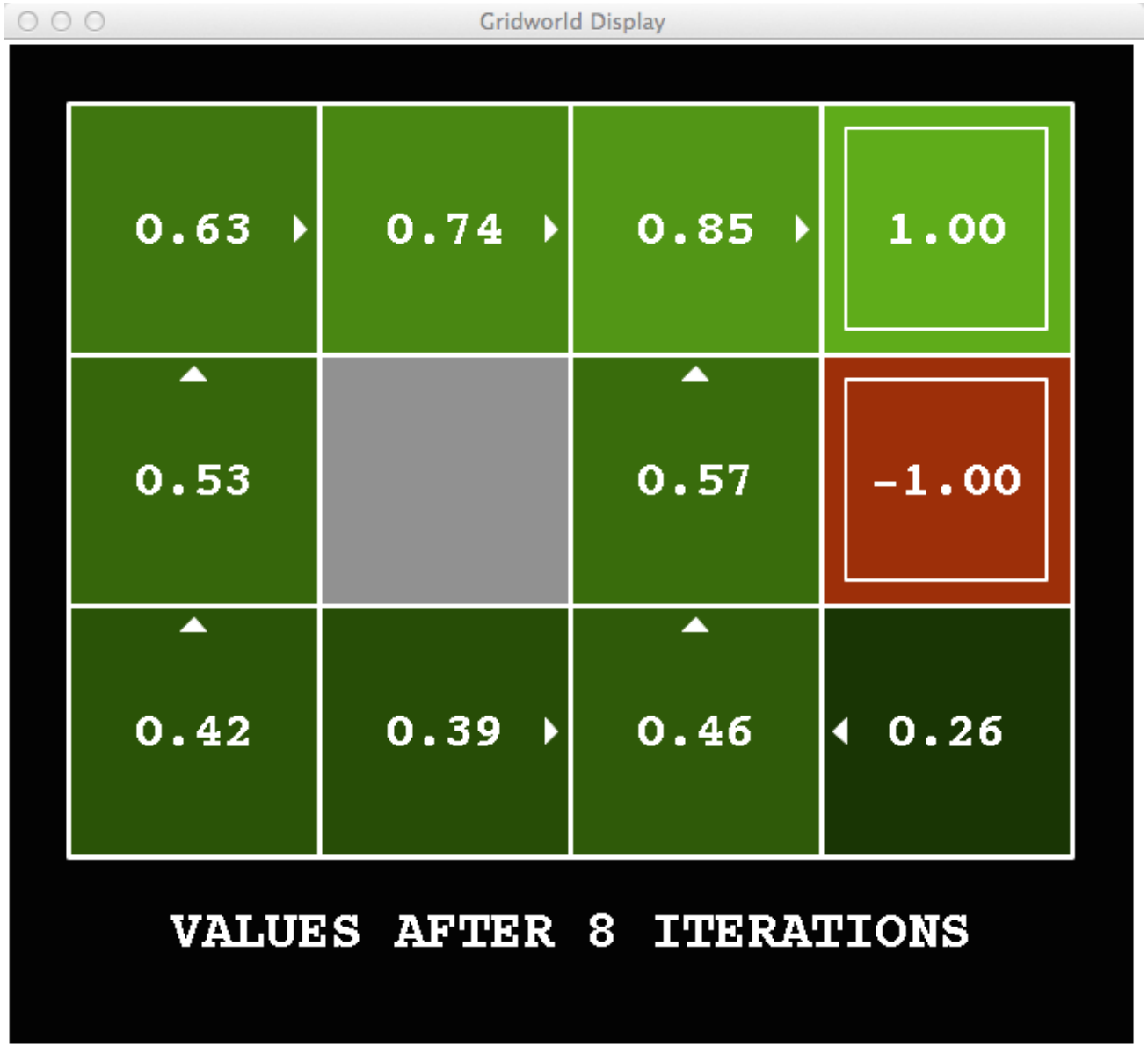
Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration



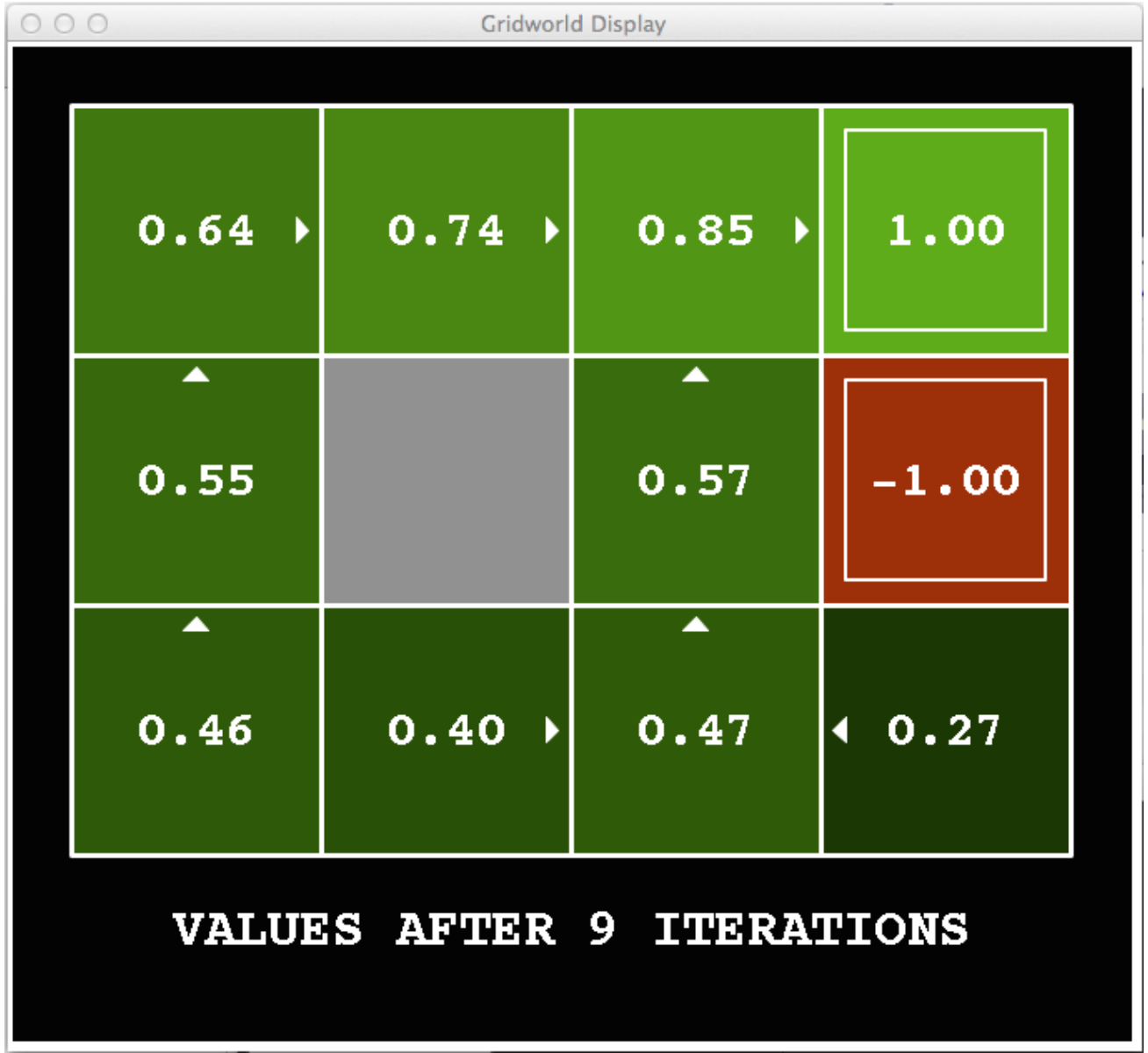
Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration



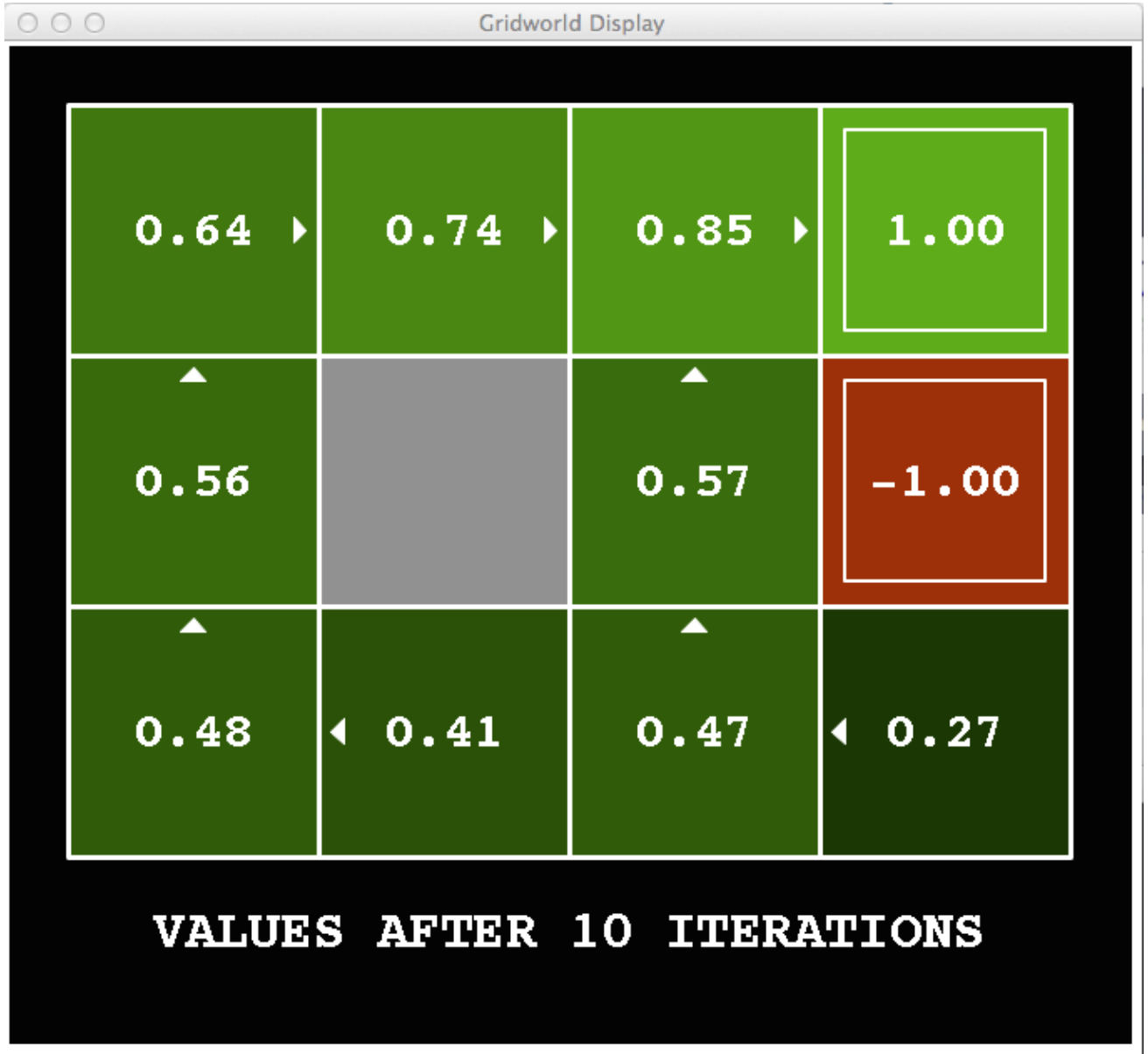
Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration



Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration

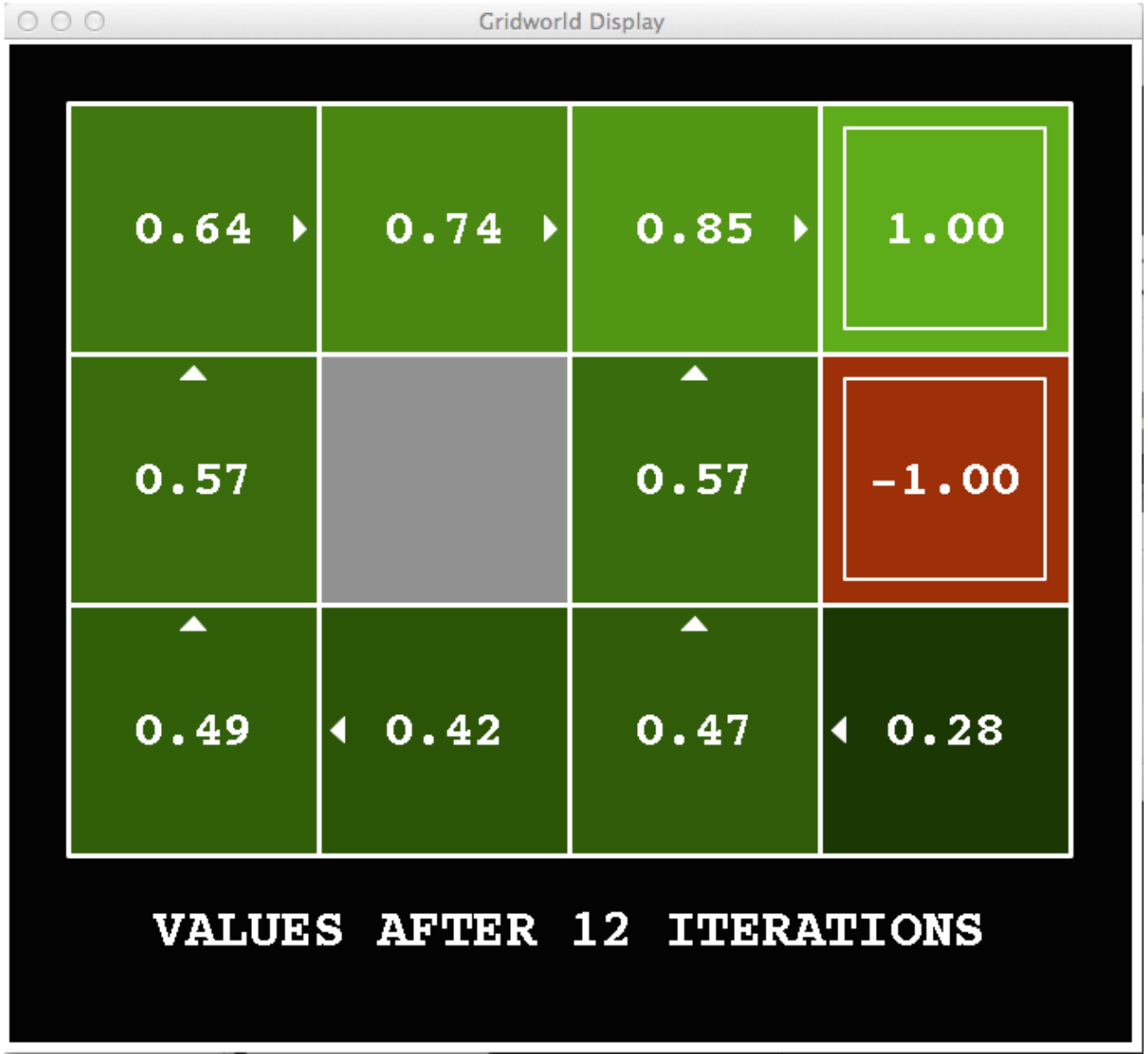


Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration

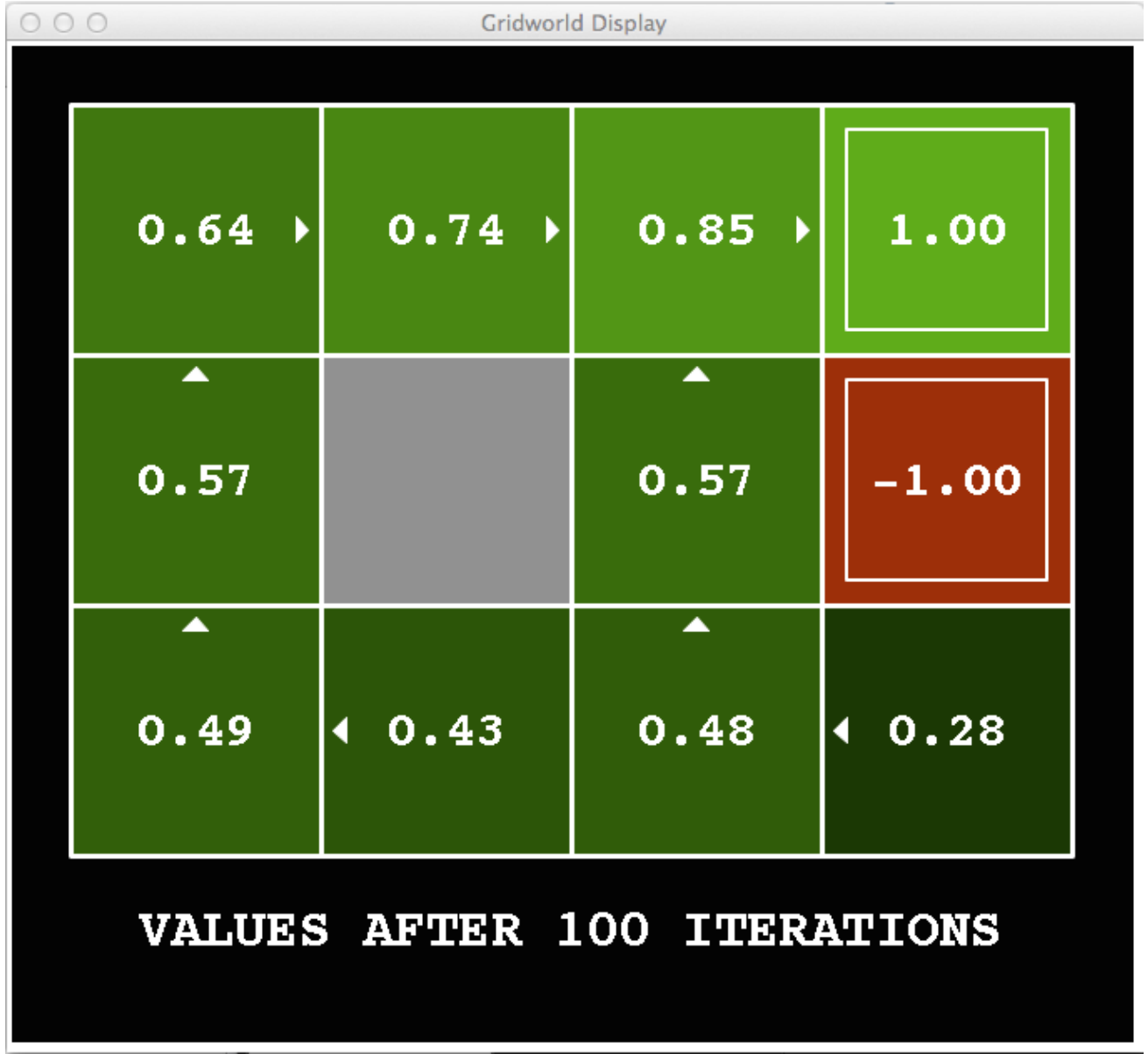
Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration



Noise = 0.2
Discount = 0.9
Living reward = 0

Value Iteration



Noise = 0.2
Discount = 0.9
Living reward = 0

Reinforcement Learning Fundamentals

Lecture 14: Value Iteration and Policy Iteration

Dr Sandeep Manjanna
Assistant Professor, Plaksha University
sandeep.manjanna@plaksha.edu.in

Announcements

- Project Proposal due at 10:00 pm tonight **23rd Feb 2024.**

In today's class…

Until now…

- Bellman Optimality:
  - Optimal Value function and Optimal Policy
- Policy Extraction

- Value Iteration
- Policy Iteration

Bellman Optimality Equation for v_*

$$v_\pi(s) = \sum_a \pi(a|s) \underbrace{\sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_\pi(s')\Big]}_{q_\pi(s,a)}, \quad \text{for all } s \in \mathcal{S},$$

The value of a state under an optimal policy must equal the expected return for the best action from that state:

$$\begin{aligned} v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\ &= \max_a \sum_{s',r} p(s',r|s,a)\big[r + \gamma v_*(s')\big]. \end{aligned}$$

Similarly,

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

How to solve these MDPs?
Dynamic Programming

- Stochastic dynamic programming represents the problem in the **form of a Bellman equation**.
- The aim is to **compute a policy** prescribing how to act optimally in the face of uncertainty.
- We will look at DP-based approaches to find V_π and V_*.
- Needs complete specification of system dynamics. Hence, does not scale well.

Slide from Prof. Ravindran's course: "Reinforcement Learning."

How to solve these MDPs?
Dynamic Programming

- DP is the solution method of choice for MDPs
  - Requires complete knowledge of system dynamics (transition matrix and rewards)
  - Computationally expensive
  - Curse of dimensionality
  - Guaranteed to converge!

- We will use DP approaches to solve both **evaluation** and **control**.

Going from π to V_π

Finding V_*

Slide from Prof. Ravindran's course: "Reinforcement Learning."

Value Iteration

- Turning the Bellman optimality equation into an update rule.

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

$$
\begin{aligned}
v_{k+1}(s) &\doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_k(s') \right],
\end{aligned}
$$

Every iteration, we look at 1-step additional expected return.

First solve for zero step problem, use that to solve the 1-step problem, in-turn use that to solve the 2-step problem…

Slide from Prof. Ravindran's course: "Reinforcement Learning."

Value Iteration

Assumption is that the ***S*** and ***A*** are small enough so that we can loop over all $s \in S$ and $a \in A$.

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
  $\Delta \leftarrow 0$
  Loop for each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

In-place updates.
It results in asynchronous computation.

Output a deterministic policy, $\pi \approx \pi_*$, such that
  $\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$

Implement Value Iteration on a simple grid-world example and compare the effect of discount factor, size of the world, convergence threshold, reward function, and transition probability on the values achieved.

Reinforcement Learning Fundamentals

Lecture 15: Policy Iteration

Dr Sandeep Manjanna
Assistant Professor, Plaksha University
sandeep.manjanna@plaksha.edu.in

Policy Evaluation

- For a given policy π, compute the state value function v_π
- Recall Bellman equation for v_π:

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

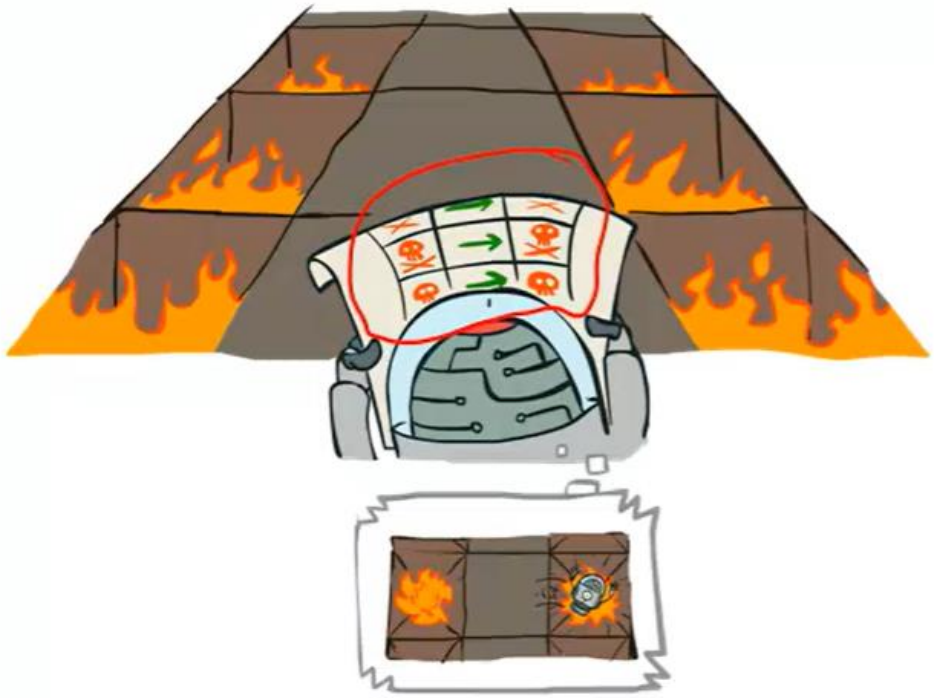
  - a system of $|S|$ simultaneous linear equations
  - solve iteratively **?**

First solve for zero step problem, use that to solve the 1-step problem, in-turn use that to solve the 2-step problem… following the policy π

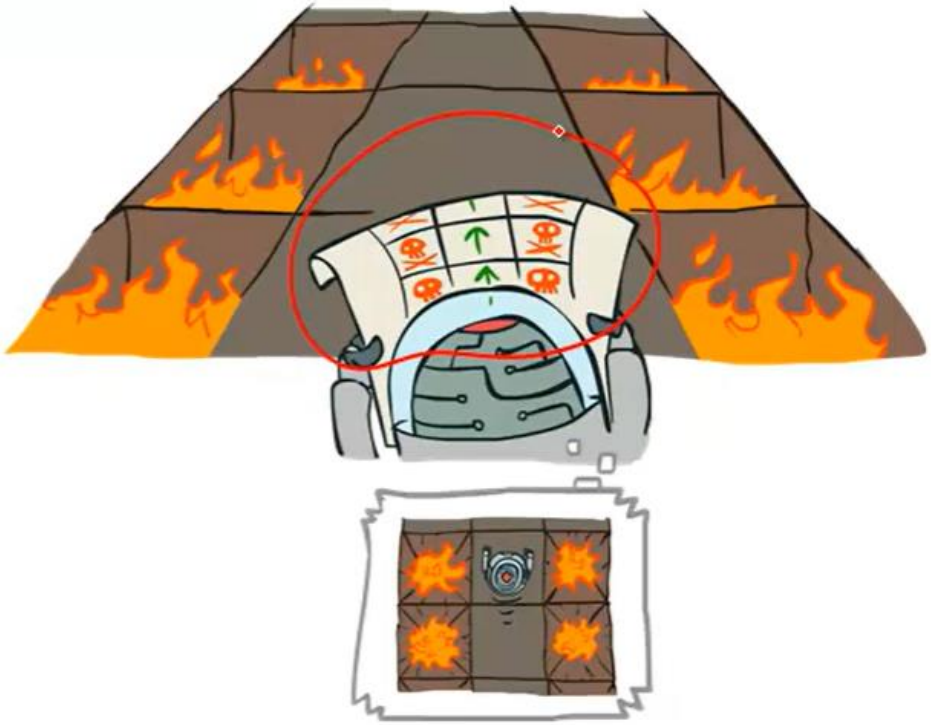
Slide from Prof. Ravindran's course: "Reinforcement Learning."

Policy Evaluation Example

Always Go Right

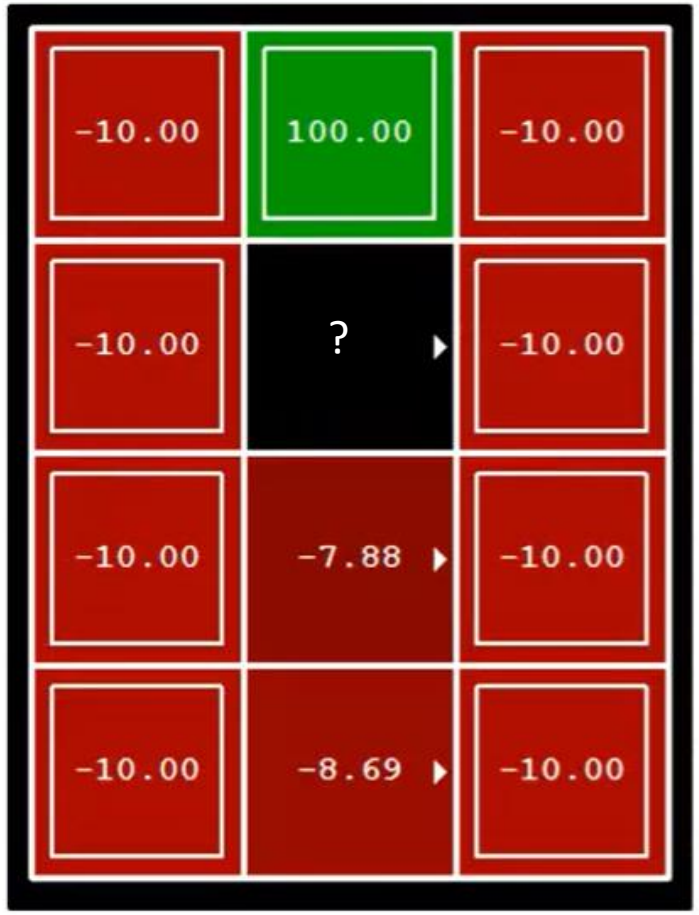


Always Go Forward

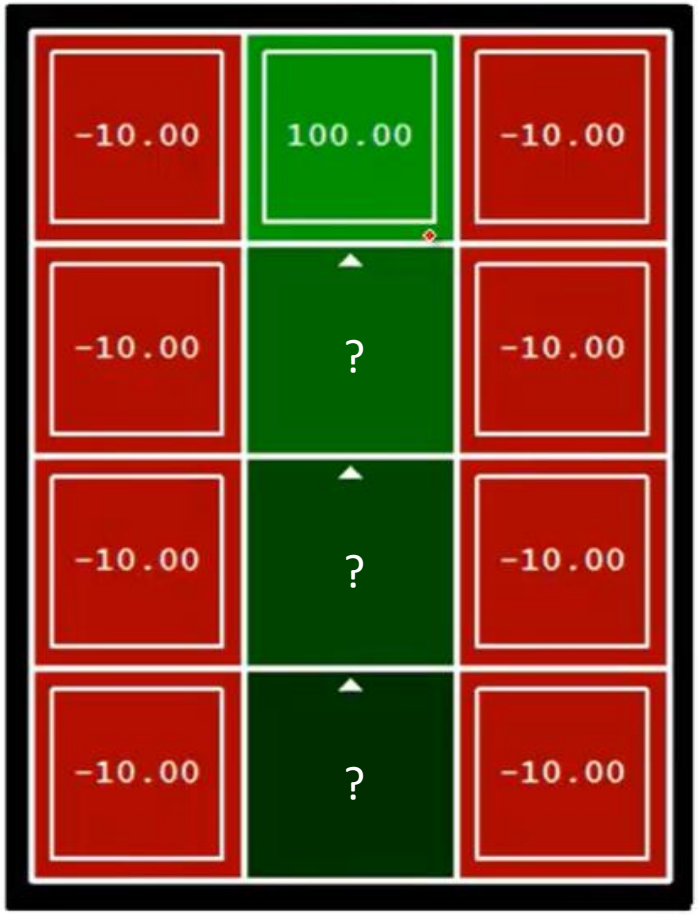


Policy Evaluation Example

Always Go Right



Always Go Forward



Policy Evaluation Example

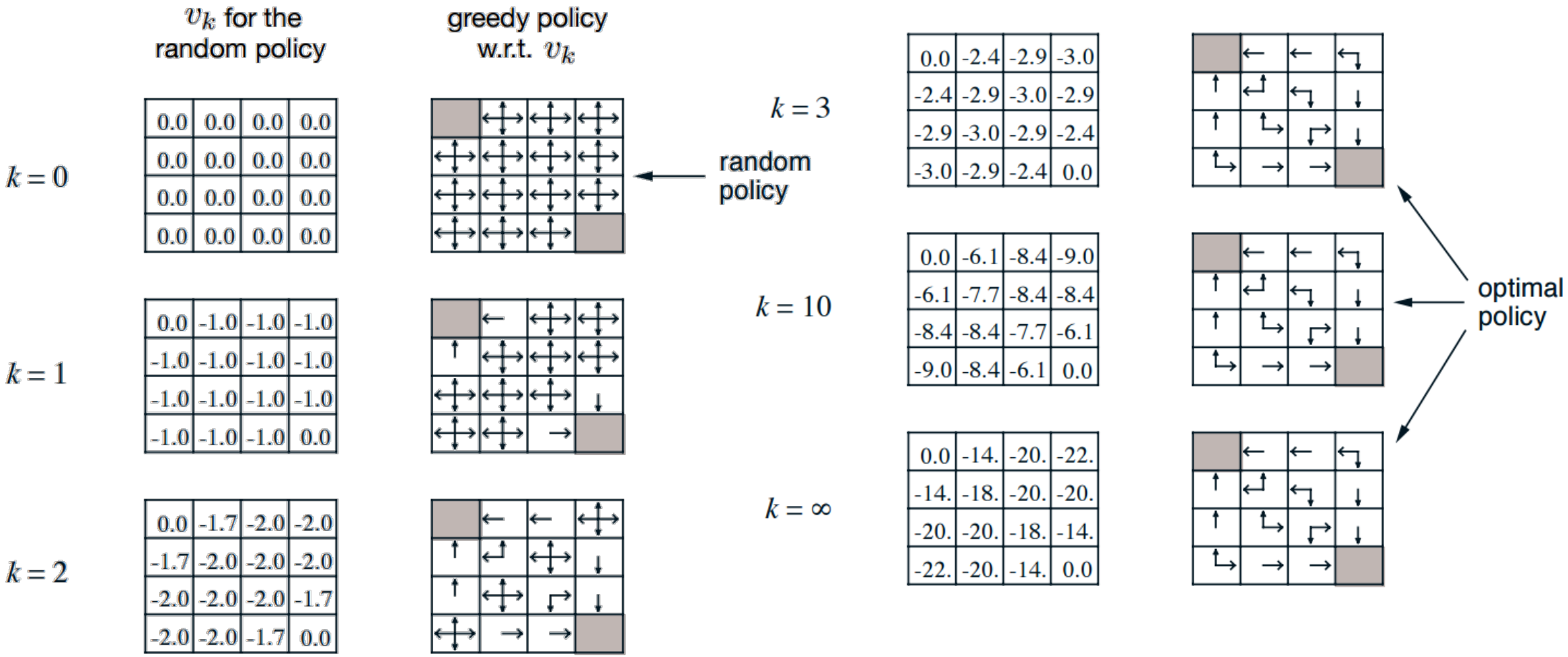


Figure 4.1: Convergence of iterative policy evaluation on a small gridworld. The left column is the sequence of approximations of the state-value function for the random policy (all actions equally likely). The right column is the sequence of greedy policies corresponding to the value function estimates (arrows are shown for all actions achieving the maximum, and the numbers shown are rounded to two significant digits). The last policy is guaranteed only to be an improvement over the random policy, but in this case it, and all policies after the third iteration, are optimal.

Policy Improvement

Finite MDPs have at least one optimal deterministic policy.

- Suppose we have computed v_π for an arbitrary deterministic policy π
- For a given state s, would it be better to choose an action $a \neq \pi(s)$?
- The value of doing a in state s is:

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

- It is better to switch to action a for state s if and only if

$$q_\pi(s,a) > v_\pi(s)$$

Slide from Prof. Ravindran's course: "Reinforcement Learning."

Policy Improvement

Do this for all states to get a new policy π' that is **greedy** with respect to v_π:

$$\pi'(s) = \arg\max_a q_\pi(s,a)$$
$$= \arg\max_a \sum_{s',r} p(s',r \mid s,a)\,[r + \gamma v_\pi(s')]$$

Then, $v_{\pi'} \geq v_\pi$

Slide from Prof. Ravindran's course: "Reinforcement Learning."

Policy Iteration

What if $v_{\pi'} = v_\pi$? Then, for all $s, \in \mathcal{S}$, we have

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

But this is the Bellman Optimality equation.
So $v_{\pi'} = v_*$ and both π and π' are optimal policies.

$$\pi_0 \xrightarrow{\text{E}} v_{\pi_0} \xrightarrow{\text{I}} \pi_1 \xrightarrow{\text{E}} v_{\pi_1} \xrightarrow{\text{I}} \pi_2 \xrightarrow{\text{E}} \cdots \xrightarrow{\text{I}} \pi_* \xrightarrow{\text{E}} v_*$$

policy evaluation

policy improvement
"greedification"

Slide from Prof. Ravindran's course: "Reinforcement Learning."

Policy Iteration

- Alternative approach for optimal values:
  - Step 1: Policy evaluation: calculate utilities for some fixed policy (not optimal utilities!) until convergence
  - Step 2: Policy improvement: update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values
  - Repeat steps until policy converges

Policy Iteration

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

2. Policy Evaluation
   Loop:
   $\Delta \leftarrow 0$
   Loop for each $s \in \mathcal{S}$:
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\left[r + \gamma V(s')\right]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

Implement Policy Iteration on a simple grid-world example and compare the effect of discount factor, size of the world, convergence threshold, reward function, and transition probability on the values achieved.

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
   old-action $\leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
   If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Break ties consistently