UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

3547 Intelligent Agents & Reinforcement Learning

Module 6: Intro to Reinforcement Learning & Finite Markov Decision Processes

Course Plan

Module Titles
Module 1 – Introduction to Intelligent Agents
Module 2 – Search
Module 3 – Logical Inference
Module 4 – Planning and Knowledge Representation
Module 5 – Probabilistic Reasoning
Module 6 – **Current Focus: Intro to Reinforcement Learning and Finite Markov Decision Processes**
Module 7 – Dynamic Programming and Monte Carlo Methods
Module 8 – Temporal Difference Learning
Module 9 – Function Approximation for RL
Module 10 – Deep Reinforcement Learning and Policy Gradient Methods
Module 11 – Introduction to Advanced DRL
Module 12 – Presentations (no content)

Learning Outcomes for this Module

- Enable you to contrast Reinforcement Learning with the other AI techniques you've learned so far in the course
- Learn to recognize when RL is appropriate vs. other methods
- Discuss how to integrate RL with other methods to build a better AI than any of the techniques would allow individually
- Get some hands-on experience with introductory RL example code

Topics for this Module

- **6.1** Intro to Reinforcement Learning
- **6.2** Multi-armed Bandits
- **6.3** Finite Markov Decision Processes
- **6.4** WumpusWorld with RL
- **6.5** Resources and Wrap-up

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Module 6 – Section 1

Intro to Reinforcement Learning

Taking Sequential Actions: Learning vs. Planning

- Planning: Computing a “good” sequence of decisions using knowledge of the environment
  - The agent has a model of the environment
  - The agent computes a sequence of actions without interacting with the environment
- Reinforcement Learning: Learning to make the right sequence of decisions (as to which action to take at each step) to maximize a desired result
  - The environment is fully or partially unknown to the agent
  - The agent interacts with the environment through trial-and-error
  - The agent uses what it learns to improve its policy

How is RL Different than Supervised Learning?

- No dataset of examples to learn a function from
- Feedback is sporadic and perhaps delayed
- Time is an explicit factor
- The agent interacts with and can affect the environment

Applications

- Machine control
  - Aircraft
  - Power plants
  - Self-driving cars
  - Robots
- Entertainment
  - Game bots
- Finance
  - Investment strategy
- Automated service
  - Chatbots
- Marketing
  - eCommerce ads

Game Examples

- Backgammon
- Go
- Atari
- Texas Hold ‘em
- Starcraft

Tic-Tac-Toe and Hexapawn in Reinforcement Learning

MichaelMaggs [CC BY-SA 2.5 (https://creativecommons.org/licenses/by-sa/2.5)]

The Four Key Elements of Reinforcement Learning

- **Policy**
  - A mapping from perceived states of the environment to agent actions
  - Can be a simple lookup to a complex computation such as a search
- **Reward Signal**
  - A percept from the environment on each time step that provides feedback
  - The agent's objective is to maximize the total reward it receives over the long run
  - The reward signal is used to update the policy so in the future the agent chooses better actions

The Four Key Elements (Cont'd)

- **Value Function**
  - A measure of the future expected reward given the current perceived state of the environment
  - Gives a value to the *long-term* desirability of each state
- **Model**
  - As with logical agents, the better the model of the environment, the better the agent can plan its course of action

How is This Different Than Previous Agents?

- We needed to program-in a strategy or value function for our logical agents
- They didn't learn from experience (other than possibly tracking changes in the environment during a single run)
- Minimax assumes the opponent plays optimally, so cannot exploit recurring weaknesses in an opponent's strategy
- Evolutionary algorithms retain a kind of learning from generation-to-generation, but play many games with a small set of fixed policies, then try some random variations on the best, whereas RL learns purposefully from every game and can potentially take advantage of what it learns during the course of play

Reinforcement Learning in a Nutshell

- For a minimax chess, go or tic-tac-toe player, we search some number of moves ahead, estimate the value of the alternative legal moves and select the one with the highest value
- In RL
  - We learn a value function for each state of (or pattern of pieces on) the board by playing many games
  - We mostly take the best (called **greedy**) move but occasionally take an exploratory move to experience a new line of play
  - For greedy moves we update the value of the previous state to be a little closer to the value of the next state, (called **backing up**)

$$V(S_t) \leftarrow V(S_t) + \alpha(V(S_{t+1}) - V(S_t))$$

Module 6 – Section 2

Multi-armed Bandits

Multi-armed Bandits

- A simplified initial puzzle used to introduce RL
- Choose between k options which randomly produce a reward taken from a stationary distribution dependent on the action taken
- Objective: Maximize total reward over some time period

Action Values

- The value q_* of an action a at time t is the expected value of the reward at t given a

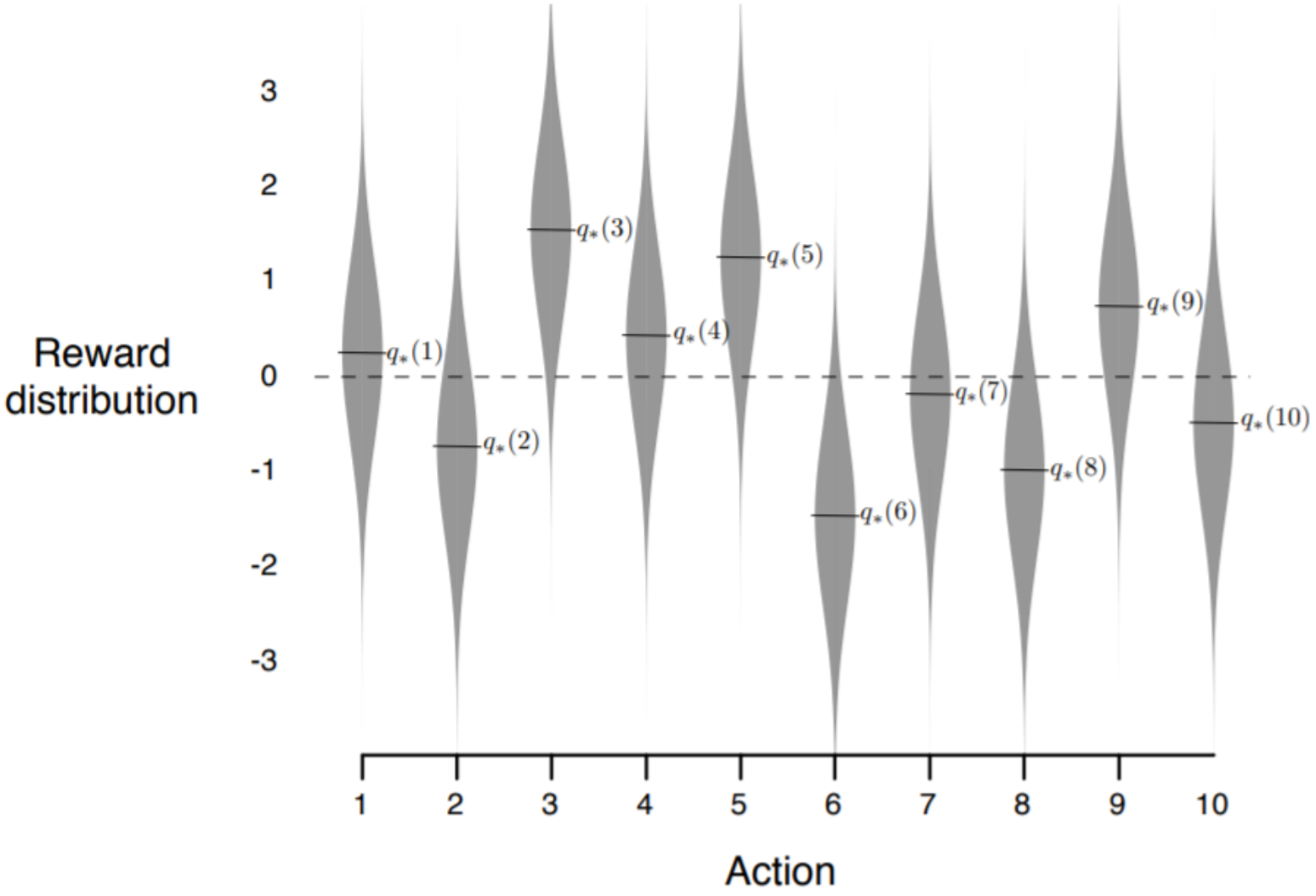
$$q_*(a) = E(R_t|A_t = a)$$

- We don't know $q_*(a)$ so we look for $Q_t(a)$, an estimate of $q_*(a)$
- We call the action available at any time step with the largest $Q_t(a)$ the **greedy** action
- We can be greedy and **exploit**, or choose another action and **explore**

Action Value Methods

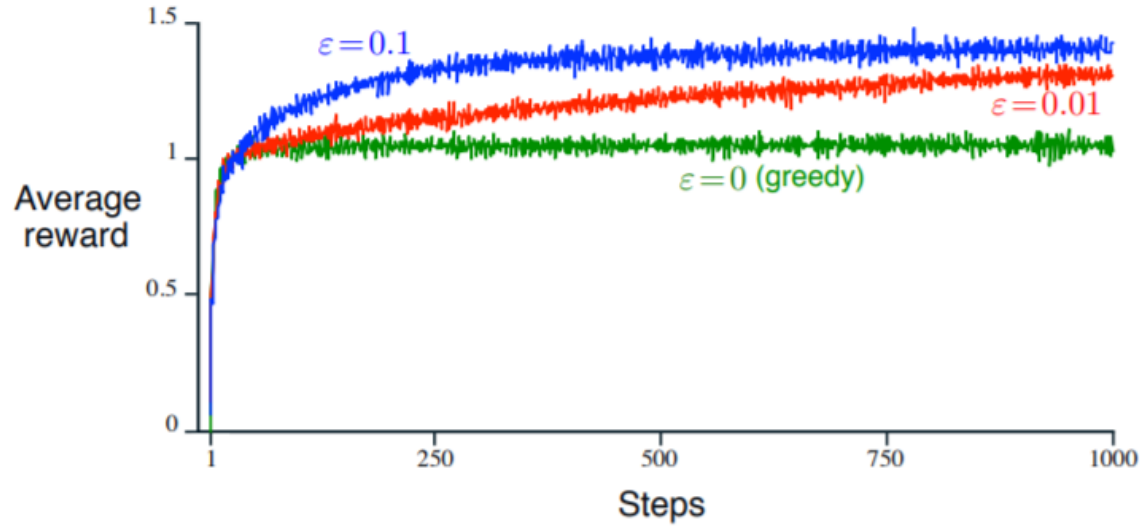
- Basic idea for finding $Q_t(a)$: Average the reward received over many examples of taking the action
- If we want to be greedy, we can always take the action with the largest $Q_t(a)$ i.e. $A_t = \underset{a}{\operatorname{argmax}}\, Q_t(a)$
- Better if we can be mostly greedy but occasionally explore
- Methods that are mostly greedy but instead choose to explore with a small probability ε are called **ε-greedy**
- Given enough experience, **ε-**greedy methods are guaranteed to cause $Q_t(a)$ to converge to $q_*(a)$

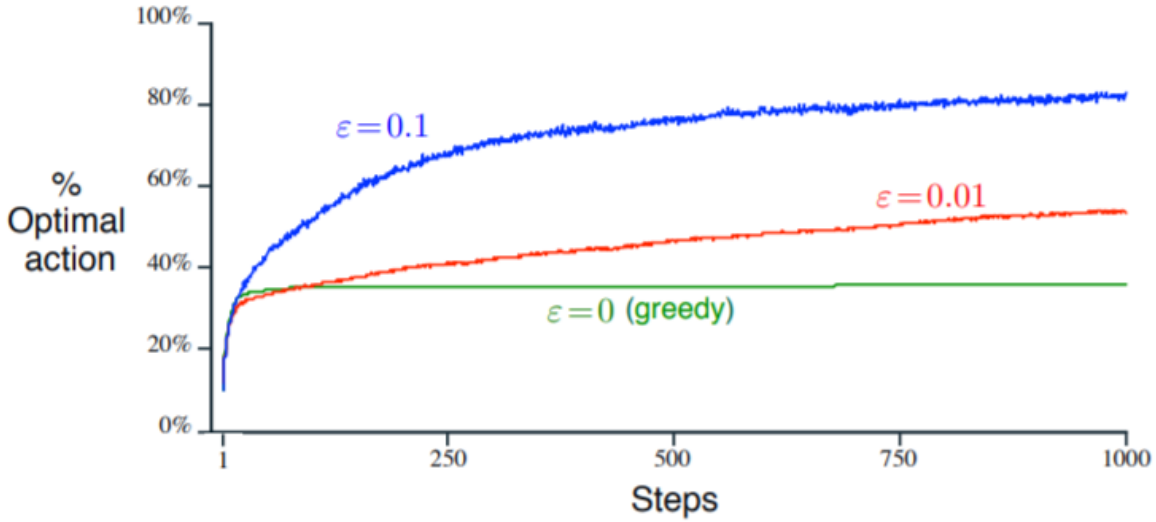
Example: 10-Armed Bandit



Source: Sutton & Barto p. 28

Greedy vs. ε-Greedy Methods





Source: Sutton & Barto p. 29

Updating $Q_t(a)$

- Fortunately we can update an average with a new observation easily and cheaply:

$$Q_{n+1} = Q_n + \frac{1}{n}(R_n - Q_n)$$

- We can start with a large ε and decrease it over time, or adaptively increase/decrease it if we know when the reward distributions are less/more stationary

Module 6 – Section 3

Finite Markov Decision Processes

Finite Markov Decision Processes

- The overall setting is much as before:
  - We have an agent which interacts with the environment by taking actions
  - The environment has a state which may be partially or fully visible
  - Time is considered to proceed in discrete steps
- But now we have a numerical reward signal which is our key interest
- We start in a state S_o and take action A_0
- We then receive reward R_1 and find ourselves in state S_1
- To be an MDP, the random variables R_t and S_t must have well-defined discrete distributions dependent only on the preceding state and action
- To be a finite MDP, the sets S, A and R must all be finite

MDP Dynamics

$$p(s', r|s, a) = \Pr(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a)$$

- This is an ordinary deterministic function
- It defines the **dynamics** of the MDP
- It specifies a probability distribution over (s', r) for each (s, a) the agent might find itself in

Goals and Rewards

- In RL the notion of a goal state is encapsulated in the reward signal; the agent will seek states that have high payoffs
- We are interested in long-term total reward and so we design our agent to maximize it
- Often we will penalize an agent for wasting time
- Prior knowledge about how to achieve the goal is better placed in the initial policy or initial value functions than in the reward signal

Returns and Episodes

- The **return** $G_t = R_{t+1} + R_{t+2} + \cdots + R_T$ where T is the final time step
- We want to maximize the expected return
- Interactions with the environment can be **continuing** or **episodic**
- We can consider all episodes as ending in a single terminal state denoted S^+ (but with different rewards in different cases)
- In the continuing case T and G could be infinite so we should discount future returns:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$$

where γ is a parameter between 0 and 1

Policies and Value Functions

- Value functions are functions of either states or state-action pairs
- They are defined with respect to the agent's behavior, called a **policy**
- If an agent is following policy π at time t then $\pi(a|s)$ gives the probability of taking action a given that the environment is in state s
- Then the **state-value function** for that policy is

$$v_\pi(s) = E_\pi(G_t|S_t = s)$$

and the **action-value function** is

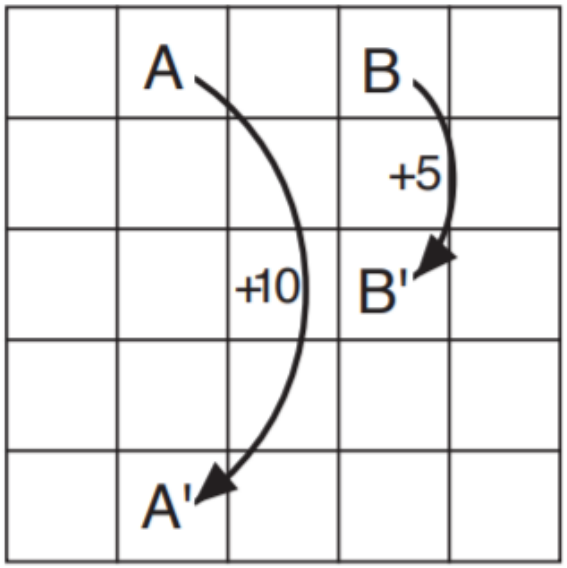
$$q_\pi(s,a) = E_\pi(G_t|S_t = s, A_t = a)$$

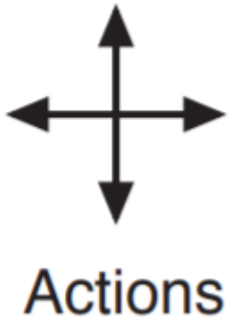
Prediction and Control

- Reinforcement Learning has two interacting sub-problems: Prediction and Control

- **Prediction**: Given a policy, how valuable is it to be in each state?
- **Control**: Given a state-value function what is the best policy?

Gridworld: Prediction



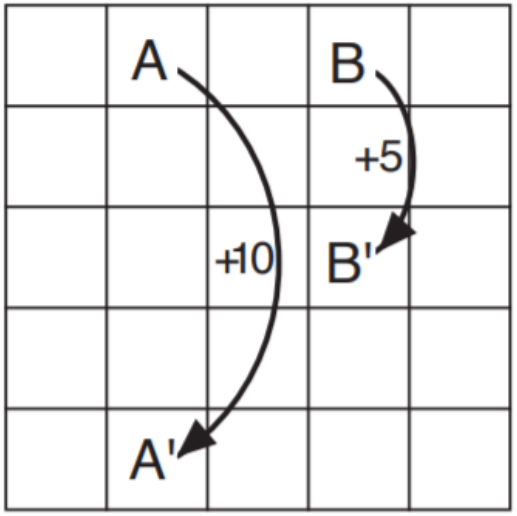


3.3	8.8	4.4	5.3	1.5
1.5	3.0	2.3	1.9	0.5
0.1	0.7	0.7	0.4	-0.4
-1.0	-0.4	-0.4	-0.6	-1.2
-1.9	-1.3	-1.2	-1.4	-2.0

See https://github.com/ShangtongZhang/reinforcement-learning-an-introduction/blob/master/chapter03/grid_world.py

Source: Sutton and Barto p. 60

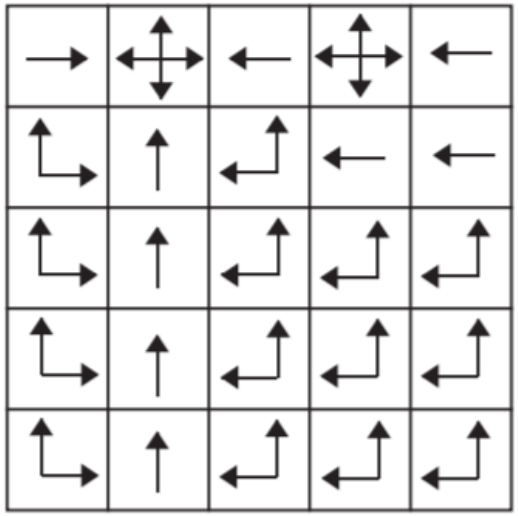
Gridworld: Control



Gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

v_*



π_*

Source: Sutton and Barto p. 65

Module 6 – Section 4

WumpusWorld with RL

Applying RL to WumpusWorld

- RL allows actions to be probabilistic
  - Couldn't do that with logic programming
  - Could with probabilistic programming
- We can learn by playing many times
  - Previous techniques required us to program-in strategy
  - We'll have to play a vast number of games to learn all the best actions in every situation
- A key question is how we define the states
  - We need the states to have the Markov property
- Logical agents are excellent at planning the return to the maze entrance; RL would need to learn a sequence of actions to return to the entrance

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Module 6 – Section 5

Resources and Wrap-up

Resources

- Sutton & Barto, Reinforcement Learning, 2nd Ed., MIT Press. 2018.
- Richard Sutton's website: http://incompleteideas.net/
- https://gym.openai.com/
- Algorithms of Reinforcement Learning: https://sites.ualberta.ca/~szepesva/papers/RLAlgsInMDPs.pdf

Summary

- RL is particularly applicable when:
  - We want to learn a set of behaviours
  - The environment provides sporadic feedback on whether the agent is doing the right things
  - There are many existing examples of complete games/tasks to learn from or self-play is possible
  - The environment is not changing quickly
  - Planning/inference/constraint solving is not required
- RL learns by attributing the value of later states to earlier ones so that good early states will be chosen to lead to good late states

Next Week

- Dynamic Programming: Computing an optimum policy when we have complete knowledge of the environment
- Monte Carlo prediction: Discovering one when we don't

Any questions?

UNIVERSITY OF TORONTO
SCHOOL OF CONTINUING STUDIES

Thank You

Thank you for choosing the University of Toronto School of Continuing Studies