NUS | Computing
National University of Singapore

Reinforcement Learning

CS4246/CS5446

AI Planning and Decision Making

Please join:
pollev.com/anarayan

This lecture will be recorded!

Topics

- Reward Based Learning and Planning (23.1)
- Learning - Passive RL (23.2)
  - Monte Carlo learning: Direct utility estimation (23.2.1)
  - Adaptive dynamic programming (23.2.2)
  - Temporal difference learning (23.2.3)
- Planning - Active RL (23.3)
  - Active adaptive dynamic programming
  - Exploration vs exploitation (23.3.1)
  - Temporal difference Q-learning and SARSA (23.3.3)

Recall: Sequential Decision Problems

- What are sequential decision problems?
  - An agent’s utility depends on a sequence of decisions
  - Incorporate utilities, uncertainty, and sensing
  - Search and planning problems are special cases
  - Decision (Planning) Models:
    - Markov decision process (MDP)
    - Partially observable Markov decision process (POMDP)
    - Reinforcement learning: sequential decision making + learning

Solving Sequential Decision Problems

- Decision (Planning) Problem or Model
  - Appropriate abstraction of states, actions, uncertain effects, goals (wrt costs and values or preferences), and time horizon + observations (through sensing)
- Decision Algorithm
  - Input: a problem
  - Output: a solution in the form of an optimal action sequence or policy over time horizon
- Decision Solution
  - An action sequence or solution from an initial state to the goal state(s)
    - An optional solution or action sequence; OR
    - An optimal policy that specifies "best" action in each state wrt to costs or values or preferences
  - (Optional) A goal state that satisfies certain properties

Recall: Decision Making under Uncertainty

- Decision Model:
  - Actions: $a \in A$
  - Uncertain current state: $s \in S$ with probability of reaching: $P(s)$
  - Transition model of uncertain action outcome or effects:
    $P(s'|s,a)$ – probability that action a in state s reaches state s'
  - Outcome of applying action a:
    $\text{Result}(a)$ – random variable whose values are outcome states
  - Probability of outcome state s', conditioning on that action a is executed:
    $P(\text{Result}(a) = s') = \sum_s P(s)P(s'|s,a)$
  - Preferences captured by a utility function:
    $U(s)$ – assigns a single number to express the desirability of a state s

Recall: Markov Decision Process (MDP)

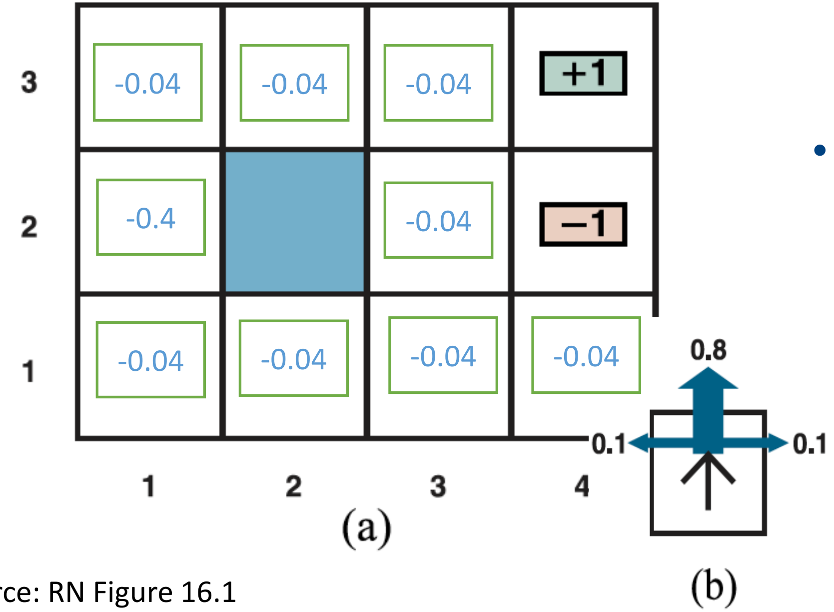
- Formally:
  - An MDP $M \triangleq (S, A, T, R)$ consists of:
  - A set S of states
  - A set A of actions
  - A transition function $T: S \times A \times S \rightarrow [0,1]$ such that:

$$\forall s \in S, \forall a \in A: \sum_{s\prime \in S} T(s, a, s') = \sum_{s\prime \in S} P(s'|s, a) = 1$$

  - A reward function $R: S \rightarrow \Re$ or $R: S \times A \times S \rightarrow \Re$

  Solution is a policy – a function to recommend an action in each state: $\pi: S \rightarrow A$
    - Solution involves careful balancing of risk and reward

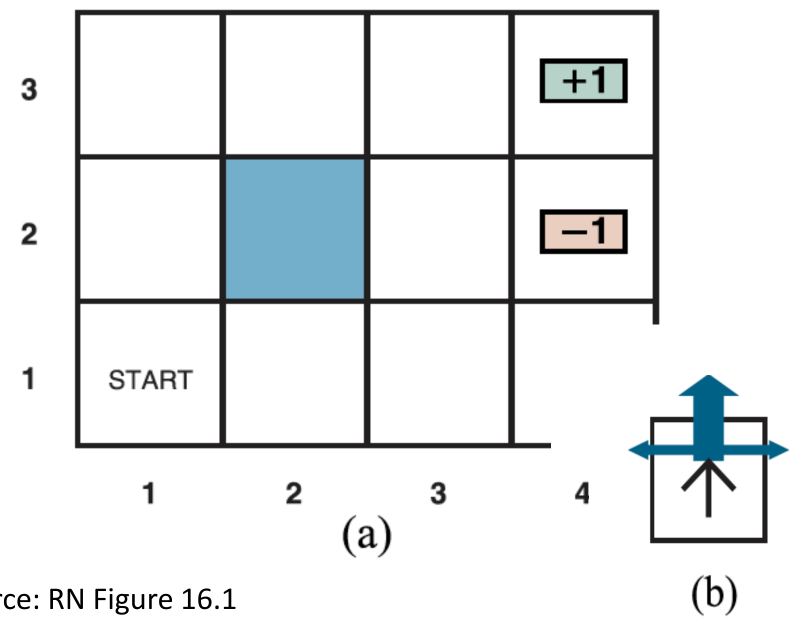
Example: Navigation in Grid World



Source: RN Figure 16.1

- Transition model:
  - Define (stochastic) outcome of each action
  - $P(s'|s,a)$ – probability of reaching state s' if action
- Reward model:
  - Define reward received for every transition
  - $R(s,a,s')$ – For every transition from s to s' via action a; AND/OR
  - $R(s)$ – For any transition into state s
  - Rewards may be positive or negative, but bound by $\pm R_{max}$
  - Utility function $U(s)$ depends on the sequence of states and actions – environment history – sum of rewards of the states in the sequence

Example: Navigation



Source: RN Figure 16.1

- Transition model:
  - Unknown
- Reward model:
  - Unknown

Reinforcement Learning

- Decision making in complex and uncertain environments
  - Learning to behave proficiently in unfamiliar environment, given only percepts and occasional rewards
  - E.g., playing chess, flying helicopter, exploring Mars, assisting elderly
- Assume:
  - Environment is a Markov Decision Process (MDP)
  - Optimal policy – policy that maximizes the expected total reward

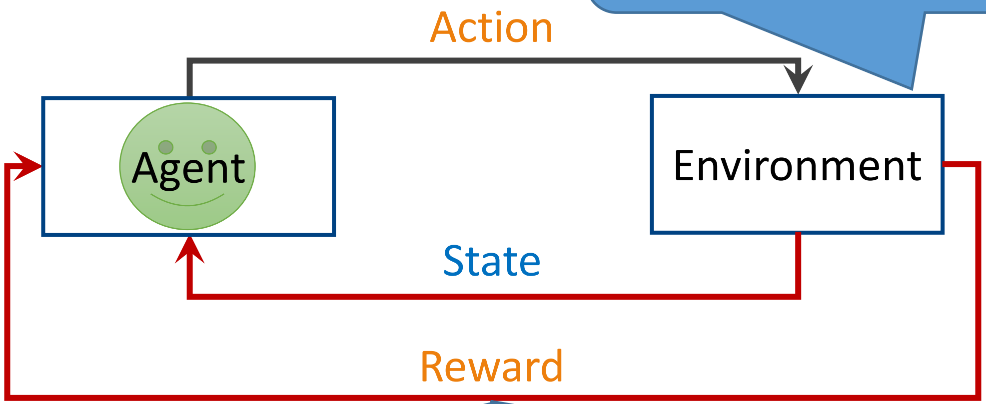
Rely on the Markov property

- Reinforcement learning
  - Use observed rewards to learn an optimal policy for the environment
  - Often with no knowledge of the environment model or reward function
    - E.g., don't know the game rules, just play until Win/Loss declared
    - Would this for a good approach for an AI tutor?
  - Influences from psychology, neuroscience, operations research, and optimal control theory.

Reinforcement Learning

- The agent – environment loop

Fully observable environment based on percepts
Unknown environment
Unknown probabilistic action outcomes



Reward or Reinforcement as feedback for learning – achieved along the way or at the end

"Hardwired" or given rewards, e.g., hunger and pain, pleasure and food

Part of the input perception; agent must differentiate & recognize it as a *reward*

Types of RL Agents

- Reinforcement learning
  - [PL] Passive Learning: Learning Utility functions
  - [AL] Active Learning : Learning to Plan or Decide
  - [MB] Model-based: Learn transition model to solve problem
  - [MF] Model- free: Do not (need to) learn transition model to solve problem
- Passive learning agent
  - Has a fixed policy that determines his behavior
  - sf policy evaluation in policy iteration
- Active learning agent
  - Must decide what actions to take
  - sf value iteration

Passive Learning

Predict utility function (value function) of state given a fixed policy

Passive Learning

Main idea:

- Agent executes a set of trials in environment using policy π; starts in $(1,1)$ until reaches terminal state
- Percepts supply both current state and reward received in state
- Use reward information to learning expected utility $U^{\pi}(s)$associated with each non-terminal state s

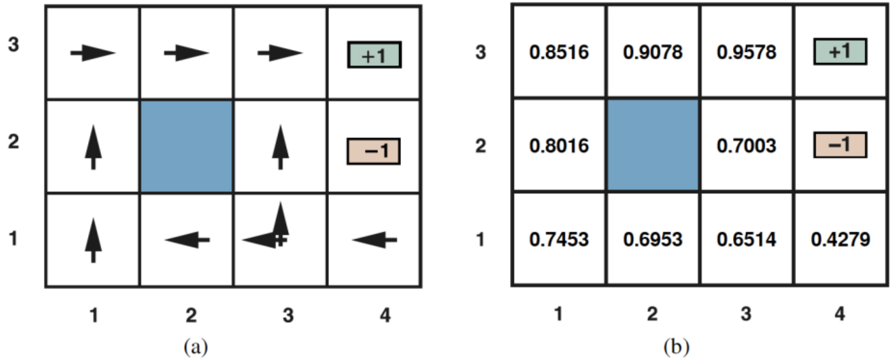
$$U^{\pi}(s) = E[\sum_{t=0}^{\infty} \gamma^t R(S_t)]$$

where $R(s)$ is reward for a state s, S_t is a random variable indicating state reached at time t when executing policy π

  - sf policy evaluation (in policy iteration)

- Assume $\gamma = 1$ in examples that follow

Learning Utility Function



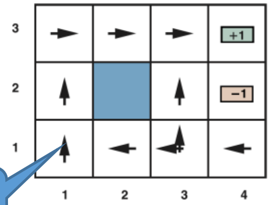
What are the percepts in each state?

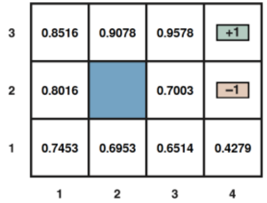
Source: RN Figure 23.1

- The policy π is fixed:
  - Agent always executes $\pi(s)$ in state s
- Goal: Learn utility function or value function $U^{\pi}(s)$ from observations
  - Unknown transition model, $P(s'|s, a)$
  - Unknown reward function, $R(s)$

If known, simply do policy evaluation!

Learning Utility Function





- Agent executes a series of trials using π, e.g.,
  - Trial 1: $(1,1) \xrightarrow[Up]{\text{-.04}} (1,2) \xrightarrow[Up]{\text{-.04}} (1,3) \xrightarrow[Right]{\text{-.04}} (1,2) \xrightarrow[Up]{\text{-.04}} (1,3) \xrightarrow[Right]{\text{-.04}} (2,3) \xrightarrow[Right]{\text{-.04}} (3,3) \xrightarrow[Right]{\text{+1}} (4,3)$
  - Trial 2: $(1,1) \xrightarrow[Up]{\text{-.04}} (1,2) \xrightarrow[Up]{\text{-.04}} (1,3) \xrightarrow[Right]{\text{-.04}} (2,3) \xrightarrow[Right]{\text{-.04}} (3,3) \xrightarrow[Right]{\text{-.04}} (3,2) \xrightarrow[Up]{\text{-.04}} (3,3) \xrightarrow[Right]{\text{+1}} (4,3)$
  - Trial 3: $(1,1) \xrightarrow[Up]{\text{-.04}} (1,2) \xrightarrow[Up]{\text{-.04}} (1,3) \xrightarrow[Right]{\text{-.04}} (2,3) \xrightarrow[Right]{\text{-.04}} (3,3) \xrightarrow[Right]{\text{-.04}} (3,2) \xrightarrow[Up]{\text{-1}} (4,2)$
- Utility or value of a state:
  - Expected total reward from that state onwards
  - Also called expected reward-to-go / expected return
- Utility/value of s under π is:

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t) | S_0 = s\right]$$

Assume
$\gamma = 1$

Example: Monte Carlo Learning

$$(1,1) \xrightarrow[Up]{-.04} (1,2) \xrightarrow[Up]{-.04} (1,3) \xrightarrow[Right]{-.04} (1,2) \xrightarrow[Up]{-.04} (1,3) \xrightarrow[Right]{-.04} (2,3) \xrightarrow[Right]{-.04} (3,3) \xrightarrow[Right]{+1} (4,3)$$

$$(1,1) \xrightarrow[Up]{-.04} (1,2) \xrightarrow[Up]{-.04} (1,3) \xrightarrow[Right]{-.04} (2,3) \xrightarrow[Right]{-.04} (3,3) \xrightarrow[Right]{-.04} (3,2) \xrightarrow[Up]{-.04} (3,3) \xrightarrow[Right]{+1} (4,3)$$

$$(1,1) \xrightarrow[Up]{-.04} (1,2) \xrightarrow[Up]{-.04} (1,3) \xrightarrow[Right]{-.04} (2,3) \xrightarrow[Right]{-.04} (3,3) \xrightarrow[Right]{-.04} (3,2) \xrightarrow[Up]{-1} (4,2)$$

- Each trial provides a “sample” of expected return for each state visited
  - In the first trial, sample total reward:
    - For $(1,1) =$
    - For $(1,2) =$
  - Overall:
    - $U^{\pi}(1,1) =$
    - $U^{\pi}(1,2)) =$

Monte Carlo Learning

$$(1,1) \xrightarrow[Up]{\textbf{-.04}} (1,2) \xrightarrow[Up]{\textbf{-.04}} \boxed{(1,3)} \xrightarrow[Right]{\textbf{-.04}} (1,2) \xrightarrow[Up]{\textbf{-.04}} \boxed{(1,3)} \xrightarrow[Right]{\textbf{-.04}} (2,3) \xrightarrow[Right]{\textbf{-.04}} (3,3) \xrightarrow[Right]{\textbf{+1}} (4,3)$$

$$(1,1) \xrightarrow[Up]{\textbf{-.04}} (1,2) \xrightarrow[Up]{\textbf{-.04}} (1,3) \xrightarrow[Right]{\textbf{-.04}} (2,3) \xrightarrow[Right]{\textbf{-.04}} (3,3) \xrightarrow[Right]{\textbf{-.04}} (3,2) \xrightarrow[Up]{\textbf{-.04}} (3,3) \xrightarrow[Right]{\textbf{+1}} (4,3)$$

$$(1,1) \xrightarrow[Up]{\textbf{-.04}} (1,2) \xrightarrow[Up]{\textbf{-.04}} (1,3) \xrightarrow[Right]{\textbf{-.04}} (2,3) \xrightarrow[Right]{\textbf{-.04}} (3,3) \xrightarrow[Right]{\textbf{-.04}} (3,2) \xrightarrow[Up]{\textbf{-1}} (4,2)$$

- Main idea:
  - Maintain a running average of expected return for each state in a table
  - Run infinitely many trials; value converges to the true expected value
  - Number of visits may be treated differently: e.g., first visit vs. every visit
    - E.g., consider only 0.8 for (1,2); 0.88 is ignored
    - E.g., consider both 0.8 and 0.88 for (1,2)
    - Both converge to the true expected value in the limit

Monte Carlo Learning

Also called Direct Utility Estimation:

An instance of supervised learning - Each example has state as input and observed return (unbiased estimate) as output

$$\big((x_1, y_1), u_1\big), \big((x_2, y_2), u_2\big), \dots, \big((x_n, y_n), u_n\big)$$

where u_j is the measured utility of the j^{th} example

Very slow convergence:

Need to wait till the end of the episode before learning can begin

By ignoring constraints, searches much larger space, including utility functions that violate the Bellman equations for a fixed policy:

$$U^\pi(s) = \sum_{s'} P(s'|s, \pi(s))[R(s, \pi(s), s') + \gamma U^\pi(s')]$$

Quiz

Quiz answer

Quiz answer

Exercise

What is the Monte Carlo estimate of the value for state (1,1) after the trials shown?

$$(1,1) \underset{Up}{\xrightarrow{\textbf{-.04}}} (1,2) \underset{Up}{\xrightarrow{\textbf{-.04}}} (1,3) \underset{Right}{\xrightarrow{\textbf{-.04}}} (1,2) \underset{Up}{\xrightarrow{\textbf{-.04}}} (1,3) \underset{Right}{\xrightarrow{\textbf{-.04}}} (2,3) \underset{Right}{\xrightarrow{\textbf{-.04}}} (3,3) \underset{Right}{\xrightarrow{\textbf{+1}}} (4,3)$$

$$(1,1) \underset{Up}{\xrightarrow{\textbf{-.04}}} (1,2) \underset{Up}{\xrightarrow{\textbf{-.04}}} (1,3) \underset{Right}{\xrightarrow{\textbf{-.04}}} (2,3) \underset{Right}{\xrightarrow{\textbf{-.04}}} (3,3) \underset{Right}{\xrightarrow{\textbf{-.04}}} (3,2) \underset{Up}{\xrightarrow{\textbf{-.04}}} (3,3) \underset{Right}{\xrightarrow{\textbf{+1}}} (4,3)$$

$$(1,1) \underset{Up}{\xrightarrow{\textbf{-.04}}} (1,2) \underset{Up}{\xrightarrow{\textbf{-.04}}} (1,3) \underset{Right}{\xrightarrow{\textbf{-.04}}} (2,3) \underset{Right}{\xrightarrow{\textbf{-.04}}} (3,3) \underset{Right}{\xrightarrow{\textbf{-.04}}} (3,2) \underset{Up}{\xrightarrow{\textbf{-1}}} (4,2)$$

Example: Adaptive Dynamic Programming

$$(1,1) \xrightarrow[Up]{-.04} (1,2) \xrightarrow[Up]{-.04} (1,3) \xrightarrow[Right]{-.04} (1,2) \xrightarrow[Up]{-.04} (1,3) \xrightarrow[Right]{-.04} (2,3) \xrightarrow[Right]{-.04} (3,3) \xrightarrow[Right]{+1} (4,3)$$

$$(1,1) \xrightarrow[Up]{-.04} (1,2) \xrightarrow[Up]{-.04} (1,3) \xrightarrow[Right]{-.04} (2,3) \xrightarrow[Right]{-.04} (3,3) \xrightarrow[Right]{-.04} (3,2) \xrightarrow[Up]{-.04} (3,3) \xrightarrow[Right]{+1} (4,3)$$

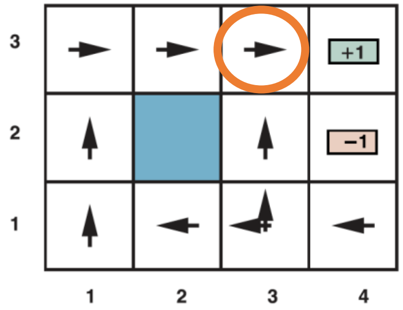
$$(1,1) \xrightarrow[Up]{-.04} (1,2) \xrightarrow[Up]{-.04} (1,3) \xrightarrow[Right]{-.04} (2,3) \xrightarrow[Right]{-.04} (3,3) \xrightarrow[Right]{-.04} (3,2) \xrightarrow[Up]{-1} (4,2)$$

What is the transition probability function from $s = (3,3)$?

To compute:

Why are all the actions $Right$?

- $P(s' = (3,2) | s = (3,3), a = Right) = \frac{1}{3}$
- $P(s' = (4,3) | s = (3,3), a = Right) = \frac{2}{3}$



Adaptive Dynamic Programming (ADP)

- Learn transition function T empirically through experience
  - Count action outcomes for each s, a Learn by counting and average!
  - Normalize to give estimates of transition probabilities $P(s'|s, a)$
- Learn reward function $R(s, a, s')$ upon entering state s'
- Solve MDP with learned T
  - Given π, perform policy evaluation for all s:

$$U^{\pi}(s) = \sum_{s'} P(s'|s, \pi(s))[R(s, \pi(s), s') + \gamma U^{\pi}(s')]$$

  - Solve $|S|$ linear equations with $|S|$ unknowns in $O(|S|^3)$ time
- Notes:
  - Can learn $P(s'|s, a)$ using supervised learning
  - Exploits constraints among utilities of states

ADP-Learner

function PASSIVE-ADP-LEARNER(*percept*) **returns** an action
  inputs: *percept*, a percept indicating the current state s' and reward signal r
  persistent: π, a fixed policy
    mdp, an MDP with model P, rewards R, actions A, discount γ
    U, a table of utilities for states, initially empty
    $N_{s'|s,a}$, a table of outcome count vectors indexed by state and action, initially zero
    s, a, the previous state and action, initially null

Using tabular representation

  if s' is new **then** $U[s'] \leftarrow 0$
  if s is not null **then**
    increment $N_{s'|s,a}[s, a][s']$
    $R[s, a, s'] \leftarrow r$
    add a to $A[s]$
    $\boxed{\mathbf{P}(\cdot \mid s, a) \leftarrow \text{NORMALIZE}(N_{s'|s,a}[s, a])}$
    $U \leftarrow \text{POLICYEVALUATION}(\pi, U, mdp)$
    $s, a \leftarrow s', \pi[s']$
    return a

Reward function R

Transition function T

MLE

Source: RN Figure 23.2

Example: Temporal-Difference (TD) Learning

$$(1,1) \xrightarrow[Up]{-.04} (1,2) \xrightarrow[Up]{-.04} (1,3) \xrightarrow[Right]{-.04} (1,2) \xrightarrow[Up]{-.04} (1,3) \xrightarrow[Right]{-.04} (2,3) \xrightarrow[Right]{-.04} (3,3) \xrightarrow[Right]{+1} (4,3)$$

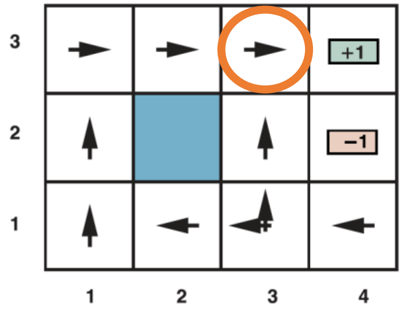
$$(1,1) \xrightarrow[Up]{-.04} (1,2) \xrightarrow[Up]{-.04} (1,3) \xrightarrow[Right]{-.04} (2,3) \xrightarrow[Right]{-.04} (3,3) \xrightarrow[Right]{-.04} (3,2) \xrightarrow[Up]{-.04} (3,3) \xrightarrow[Right]{+1} (4,3)$$

$$(1,1) \xrightarrow[Up]{-.04} (1,2) \xrightarrow[Up]{-.04} (1,3) \xrightarrow[Right]{-.04} (2,3) \xrightarrow[Right]{-.04} (3,3) \xrightarrow[Right]{-.04} (3,2) \xrightarrow[Up]{-1} (4,2)$$

After first trial:

- $U^{\pi}(1,3) = \frac{0.84+0.92}{2} = 0.88$
- $U^{\pi}(2,3) = 0.96$

- In the long run, must obey:
  - $U^{\pi}(1,3) = -0.04 + U^{\pi}(2,3)$
- Hence:
  - $U^{\pi}(1,3) = 0.92$ Current value of 0.88 needs updating!



Temporal Difference Learning (TD)

Update rule uses difference in utlities between successive states

- Temporal difference learning (TD)
  - For a transition from state s to s', TD learning does:

    $$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha\big(R(s,\pi(s),s') + \gamma U^{\pi}(s') - U^{\pi}(s)\big)$$

  - where, α is the learning rate
- In TD learning:
  - TD target: $R(s,\pi(s),s') + \gamma U^{\pi}(s')$
  - TD term or error: $R(s,\pi(s),s') + \gamma U^{\pi}(s') - U^{\pi}(s)$
    - TD term is error signal, update is intended to reduce error
  - Increases $U^{\pi}(s)$ if $R(s,\pi(s),s') + \gamma U^{\pi}(s')$ is larger than $U^{\pi}(s)$; decreases otherwise
  - Converges if α decreases appropriately with the number of times the state has visited. E.g., $\alpha(n) = O\left(\frac{1}{n}\right)$

Temporal-Difference (TD) Learning

- Main idea:
  - Adjust utility estimates towards ideal (local) equilibrium with correct utility estimates
  - In passive learning equilibrium is given by the Bellman equation for a fixed policy:

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

- TD update:
  - Involves only observed successor s', average value of $U^\pi(s)$ will converge to correct value
  - If learning rate α is changed from fixed parameter to decreasing function of increasing number of visits to a state, then $U^\pi(s)$ converges
- TD vs MC and ADP:
  - Exploits more of the Bellman equation constraints than MC
  - Does not need to learn the model (here refers to transition function T), unlike ADP

TD-Learner (Model-Free RL)

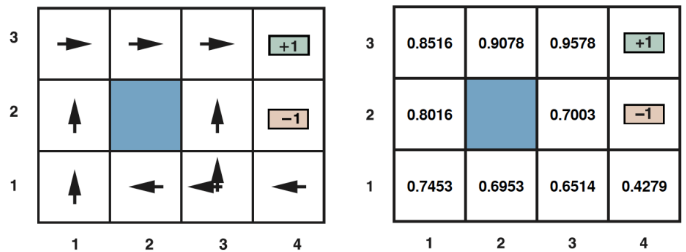
function PASSIVE-TD-LEARNER($percept$) **returns** an action
inputs: $percept$, a percept indicating the current state s' and reward signal r
persistent: π, a fixed policy
s, the previous state, initially null
U, a table of utilities for states, initially empty
N_s, a table of frequencies for states, initially zero

if s' is new **then** $U[s'] \leftarrow 0$
if s is not null **then**
increment $N_s[s]$
$U[s] \leftarrow U[s] + \alpha(N_s[s]) \times (r + \gamma\, U[s'] - U[s])$
$s \leftarrow s'$
return $\pi[s']$

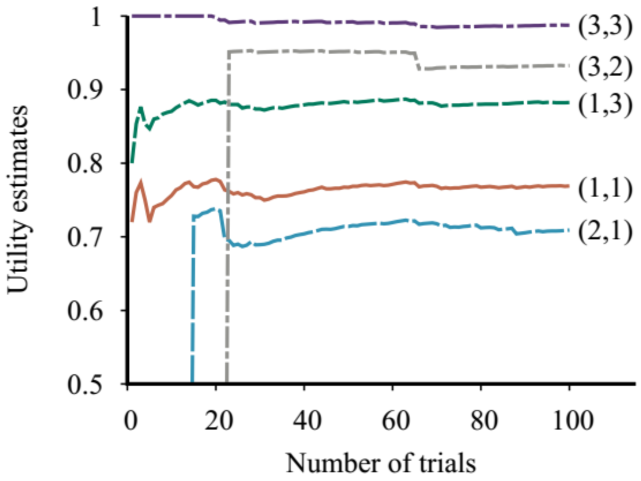
- TD needs only observed transitions to do the updates; no transition model is needed!
- Learn from every experience – update each time we experience s, $\pi(s)$, and s'

Source: RN Figure 23.4

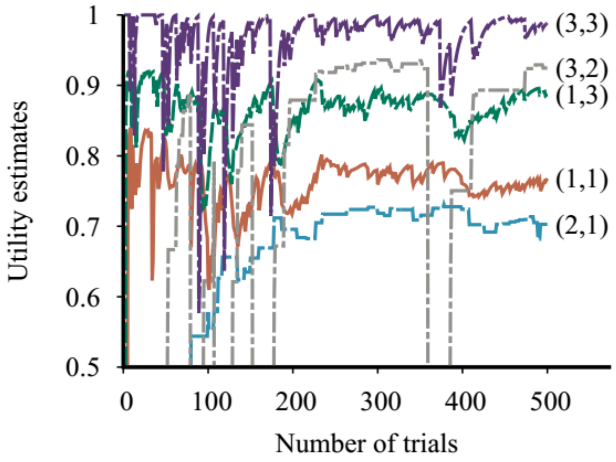
Example: ADP vs TD



(a) (b)



Adaptive Dynamic Programming



Temporal Difference Learning

Source: RN Figure 23.3 (a) and 23.5 (a)

ADP vs TD/MC

- ADP
    - Learns the model and then solves for the value
- TD/MC
    - Don’t need a model
    - Can work with measurements from real-world or simulator
- ADP
    - More data efficient
    - Requires less data from the real-world
- TD
    - Doesn’t need to compute expectation
    - Computationally more efficient

Active Learning

Choose to act optimally based on prediction of utility or value of states
Exploration vs exploitation, Q-learning, SARSA

Active Learning

- Main ideas:
  - T and R unknown; can choose any action a at each step
  - Goal: Learn optimal policy π^* that obey the Bellman equations:

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

- In summary:
  - Learner makes choices
  - Tradeoff between exploration vs. exploitation:
  - If learned $\hat{T}$ is no the true T, optimal policy obtained using $\hat{T}$ may be suboptimal wrt T, possibly incurring large policy loss

From Passive to Active ADP-Learner

function PASSIVE-ADP-LEARNER(*percept*) **returns** an action
  inputs: *percept*, a percept indicating the current state s' and reward signal r
  persistent: $\boxed{\pi\text{, a fixed policy}}$ ← π not fixed
    mdp, an MDP with model P, rewards R, actions A, discount γ
    U, a table of utilities for states, initially empty
    $N_{s'|s,a}$, a table of outcome count vectors indexed by state and action, initially zero
    s, a, the previous state and action, initially null

  if s' is new **then** $U[s'] \leftarrow 0$
  if s is not null **then**
    increment $N_{s'|s,a}[s, a][s']$
    $R[s, a, s'] \leftarrow r$
    add a to $A[s]$
    $\mathbf{P}(\cdot \mid s, a) \leftarrow$ NORMALIZE($N_{s'|s,a}[s, a]$)
    $\boxed{U \leftarrow \text{POLICYEVALUATION}(\pi, U, mdp)}$ ← $U \leftarrow \text{PolicyIteration}(\pi, U, mdp)$
    $s, a \leftarrow s', \pi[s']$
    return a

Reward function R

Transition function T

Exercise:

Is this a good reinforcement learning algorithm?

- Replace POLICY-EVALUATION step by $\pi \leftarrow$ POLICY-ITERATION(mdp) above
- π is no longer fixed; changes as transitions & rewards are learned

Source: RN Figure 23.2

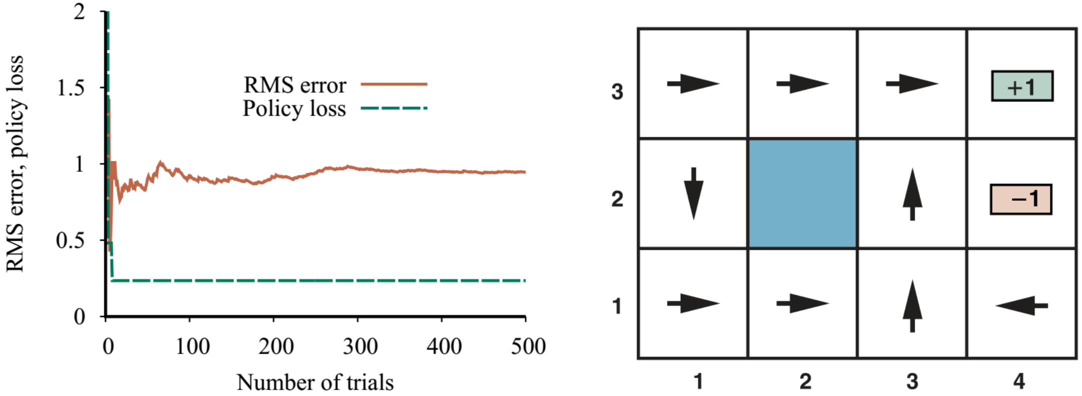
Active Adaptive Dynamic Programming

- For active-ADP:
    - Learn model T with outcome probabilities for all actions, not just for a fixed policy.
    - Learn utilities defined by optimal policy that obey the Bellman equation
    - Then, run VI or PI to determine optimal policy
    - Extract an optimal action by one-step look-ahead to maximize expected utility
- Note:
    - Learner has choice of actions
    - Actions don't just maximize rewards for given current learned model T, they also affect the percepts (states and rewards) received
    - Should learner then just execute action recommended by optimal policy? Why?

Greedy ADP Learner

RMS error in the utility estimates averaged over non-terminal states

Policy loss: $U^{\pi^*} - U^{\pi}$



- Resulting algorithm is greedy w.r.t the policy
- Policy resulting from executing action suggested by the learned model
  - Note, for e.g., at $(2, 1)$ it goes right instead of left
  - Policy reasonable, but not optimal
- Agent is greedy – sometimes doesn’t try better policies sufficiently after finding a suboptimal one

Source: RN Figure 23.6

Exploration Vs Exploitation

Hope to learn something new about the problem!

- Actions in RL gain reward and help to learn a better model
  - By improving the model, better reward may potentially be obtained
- Need to trade off
  - Exploration: Choose action to learn the true model T (by affecting the states and rewards received) to receive greater rewards in future
  - Exploitation: Given current learned model T, choose action to maximize the reward (as reflected in current utility estimates)
- Main questions:
  - How to balance exploration and exploitation to maximize long-term expected rewards?
  - When to “stop” learning the model?
- With greater understanding, less exploration is necessary!

Greedy in the Limit of Infinite Exploration

- GLIE: A scheme for balancing exploration and exploitation
  - Try each action in each state an unbounded number of times to avoid a finite probability of missing an optimal action
  - Eventually become greedy so that it is optimal w.r.t the true model
- GLIE-based schemes for forcing exploration in RL:
  - ϵ-greedy exploration
    - Choose the greedy action with probability $(1-\epsilon)$
    - Choose a random action with probability ϵ
  - GLIE based ϵ-greedy eventually converges to the optimal, but can be slow
    - One solution: lower ε over time (e.g., $\epsilon = \frac{1}{t}$)
    - Another solution: use exploration functions!

Optimism

- Alternatively, balance exploration and exploitation using greedy action selection w.r.t an optimistic estimate of the utility, $U^+(s)$
- One way to construct $U^+(s)$ is to use an exploration function $f(u, n)$ with the following update:

$$U^+(s) \leftarrow \max_a f\left(\sum_{s'} P(s'|s,a)[R(s,\mathrm{a},\mathrm{s}') + \gamma U^+(s)], N(s,a)\right)$$

  - Where, $N(s,a)$ is the number of times action a has been tried in state s

Exploration Functions

- When to explore:
  - Random actions: explore a fixed amount
  - Better idea: explore areas where “badness” has not been established
- Exploration function: f
  - Takes a value estimate (u) and a count (n), and returns an optimistic utility $f(u, n)$
  - Determines how greed (preference for high values of u) is traded off against curiosity (preference for actions that have not been tried often with low n)
  - f should be increasing in u and decreasing in n (form is not important)

Example: Exploration Functions

- An exploration function $f(u, n)$ that is increasing with u and decreasing with n:

$$f(u, n) = \begin{cases} R^+ & \text{if } n < N_e \\ u & \text{otherwise} \end{cases}$$

Where:

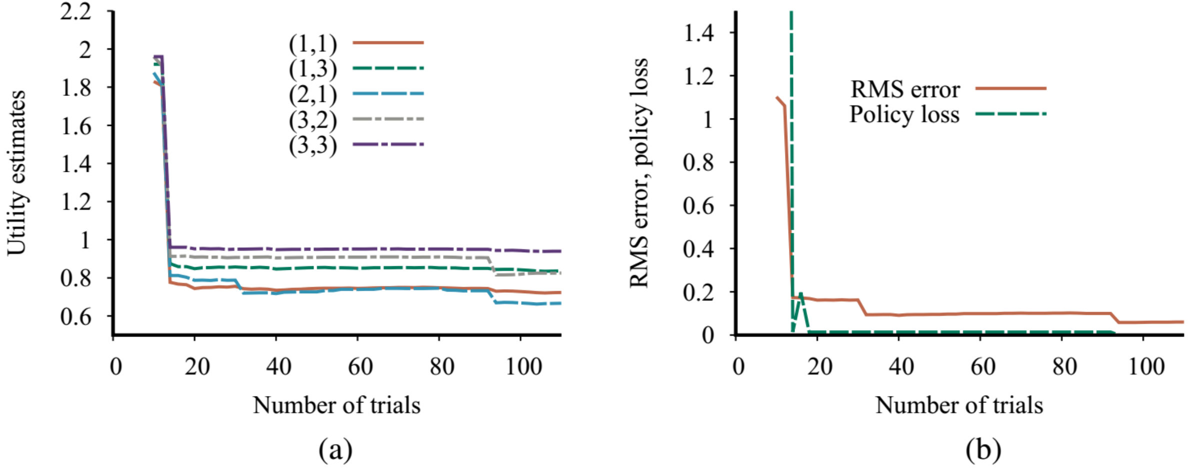
- R^+ is an optimistic estimate of the best possible reward
- N_e is the parameter

Agent tries each state-action pairs N_e times

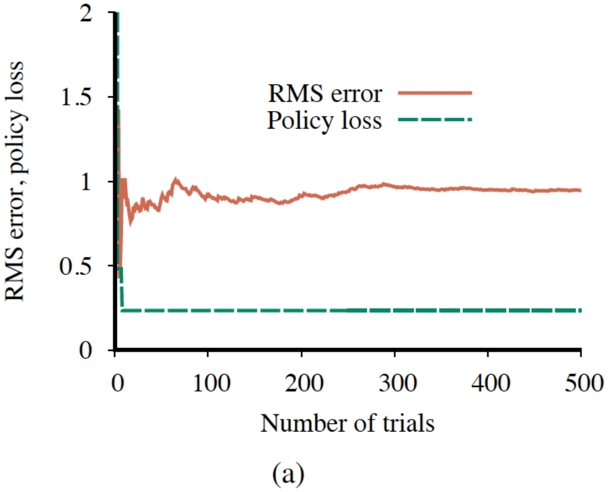
- Example: RMAX [1]
  - Works well for small problems, has guarantees

[1] Brafman, R.I. and M. Tennenholtz, *R-max - a general polynomial time algorithm for near-optimal reinforcement learning.* J. Mach. Learn. Res., 2003. **3**(null): p. 213–231.

Example: Navigation in Grid World



Exploring ADP



Greedy ADP

- Exploring ADP using exploration function with: $R^+ = 2, N_e = 5$

Source: RN Figure 23.6 (a) 23.7 (a) and (b)

Q-Learning: Learning Action-Utility Functions

- Q-function – Action-utility function $Q(s,a)$
  - Expected total discounted reward if action a is taken in state s
  - Q-functions (also called Q-values) are related to utilities by: $U(s) = \max_a Q(s,a)$
- Q-Learning
  - Model-free learning: Agent can act optimally knowing Q-function; no need for look-ahead based on a transition model
  - The Bellman equation for Q-functions:

$$Q(s,a) = \sum_{s'} P(s'|s,a)\,[R(s,a,s') + \gamma \max_{a'} Q(s',a')]$$

  - Uses TD update to learn action-utility functions or Q-functions:

TD Term or Error

$$Q(s,a) \leftarrow Q(s,a) + \alpha\boxed{[R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)]}$$

Q-learning: Active TD Learning

function Q-LEARNING-AGENT(*percept*) **returns** an action
  inputs: *percept*, a percept indicating the current state s' and reward signal r
  persistent: Q, a table of action values indexed by state and action, initially zero
    N_{sa}, a table of frequencies for state–action pairs, initially zero
    s, a, the previous state and action, initially null

TD Update

  if s is not null **then**

Exploration function

    increment $N_{sa}[s, a]$
    $Q[s, a] \leftarrow Q[s, a] + \alpha(N_{sa}[s, a])\boxed{(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])}$
  $s, a \leftarrow s', \text{argmax}_{a'} f(Q[s', a'], N_{sa}[s', a'])$

TD Term or Error

  return a

- TD Q-learning does not need a transition model, $P(s'|s, a)$ for learning or action selection

Source: RN Figure 23.8

SARSA

- State-Action-Reward-State-Action (SARSA):
  - Uses TD for prediction (evaluation) and ϵ-greedy for action selection

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R(s,a,s') + \gamma Q(s',a') - Q(s,a)]$$

where a' is the action taken at state s'

- Note:
  - Rule is applied at the end of each quintuple (s, a, r, s', a')
  - With appropriately decreasing ϵ, SARSA converges to the optimal policy

Q-Learning vs SARSA

controller *Q-learning*(S, A, γ, α)
  Inputs
    S is a set of states
    A is a set of actions
    γ the discount
    α is the step size
  Local
    real array $Q[S, A]$
    states s, s'
    action a
  initialize $Q[S, A]$ arbitrarily
  observe current state s
  repeat
    select an action a
    $do(a)$
    observe reward r and state s'
    $Q[s,a] := Q[s,a] + \alpha * (r + \gamma * \max_{a'} Q[s', a'] - Q[s,a])$
    $s := s'$
  until termination

Figure 12.3: *Q*-learning controller

controller *SARSA*(S, A, γ, α)
  Inputs
    S is a set of states
    A is a set of actions
    γ the discount
    α is the step size
  Local
    real array $Q[S, A]$
    state s, s'
    action a, a'
  initialize $Q[S, A]$ arbitrarily
  observe current state s
  select an action a using a policy based on Q
  repeat
    $do(a)$
    observe reward r and state s'
    select an action a' using a policy based on Q
    $Q[s,a] := Q[s,a] + \alpha * (r + \gamma * Q[s', a'] - Q[s,a])$
    $s := s'$
    $a := a'$
  until termination

Figure 12.5: SARSA: on-policy reinforcement learning

Source: Poole and Macwark, AI Foundations of Computational Agents, 2e, 2018

On-policy vs Off-policy Learning[1]

- On-policy learning – data used for learning is generated by the policy being learned
- Situations where data isn't generated by the policy currently being learned
  - Data collected using currently running policy
  - Explore using a different policy
- In off-policy methods:
  - Generate data using a behavior policy
  - Try to learn a target policy

What is the issue here?

[1]Material for this section are mostly from Sutton & Barto

Off-Policy and On-Policy Learning

- Q-learning update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$$

- SARSA update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big(R(s,a,s') + \gamma Q(s',a') - Q(s,a)\big)$$

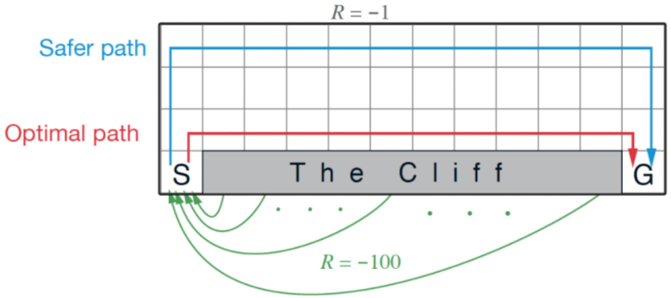
- Q-learning is off-policy – Target of Q-learning uses the max over possible actions a' at s'
  - Doesn't need the actual action in the next state in the update
  - Can converge to the optimal policy even when trained off-policy
    - Needs appropriately decaying α
    - Each action is taken in each state infinitely often
- SARSA is on-policy – Target uses Q-function from the policy that is running
  - Using Q-function from another state-action pair will give incorrect target value

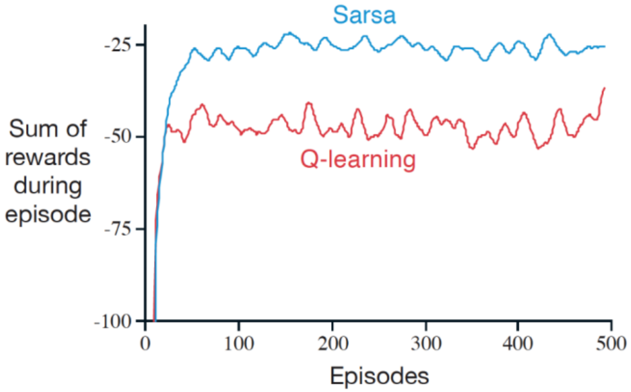
Q-Learning vs SARSA

- Similarities:
  - If agent is greedy and always takes action with the best Q-function: the two are identical
  - Both converge slowly as local updates do not enforce consistency among all the Q-functions via model
- Differences:
  - Q-learning backs up Q-function from the best action in s'
  - SARSA waits until an action is taken and backs up Q-function for that action
  - If exploration yields a negative reward: SARSA penalizes the action, Q-learning does not

Example: Cliff Walking

From Sutton & Barto



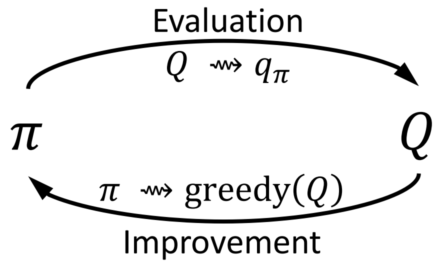


Source: SB: Example 6.6

- Actions: left, right, up, down
- Reward: -1 for each step; -100 if you fall off
- SARSA & Q-learning used with ϵ-greedy; $\epsilon = 0.1$
  - Why does SARSA outperform Q-learning?
  - What happens if GLIE exploration is used?

Generalized Policy Iteration

- How can MC and TD prediction methods be used to do control?
- Can use the idea of generalized policy iteration (GPI)



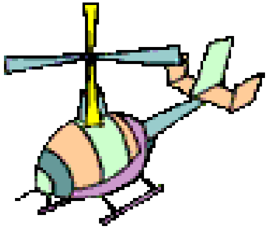
- Repeatedly:
  - Do some policy evaluation (using MC or TD), then use greedy action with respect to the current policy (policy improvement) to get more samples

Summary: Reinforcement Learning

- Passive Learning – fixed policy
  - [MB/MF] Monte-Carlo prediction (MC): Direct utility estimation
  - [MB] Adaptive dynamic programming (ADP)
  - [MF] Temporal difference methods (TD)
- Active Learning – optimal policy
  - [MB] Active Adaptive dynamic programming (ADP)
  - [MF] Active Temporal Difference methods (TD)
    - Q-Learning
    - SARSA

Applications of Reinforcement Learning

- Game playing
  - Checkers (1959, 1967)
  - Backgammon (TD-Gammon 1992)
  - Atari Games (2015)
  - Go (2019)
  - Starcraft II: https://youtu.be/gEyBzcPU5-w
  - DOTA2 (2019)
- Robotic control
  - Cart-pole balancing problem (inverted pendulum) (1968)
  - Helicopter flight (2001, 2004, …)
  - Rover in unfamiliar place (planet) …
  - Autonomous robot naviation: https://youtu.be/KyA2uTIQfxw



- Potential real world applications:
  - Power grid management
  - Generate product recommendations
  - Cargo management
  - Diagnosis and treatment of infectious disease
  - Drug discovery
  - Assistive care and education
  - …

Homework

- Readings:
  - RN: 23.1, 23.2. 23.3.1, 23.3.2
  - *SB: 6.2, 6.4, 6.5* (Optional)
- Some classical papers and recent surveys on RL

  (Journal articles publicly available online or through NUS Library e-Resources)
  - L. P. Kaelbling, M. L. Littman, & A. W. Moore. Reinforcement Learning: A Survey. JAIR, 4:237-285, 1996
  - Wirth, C., et al., *A survey of preference-based reinforcement learning methods*. J. Mach. Learn. Res., 2017. **18**(1): p. 4945–4990.