Reinforcement Learning

Introduction

- Vijay Chakilam

Multi-Armed Bandits

- A learning problem where one is faced repeatedly with a choice among k different options or actions.
- Each choice results in a random numerical reward that depends on the option/action chosen.
- The objective is to maximize the expected total reward over some time period.
- Examples:
  - Digital Advertising
  - Personalization - A/B Testing

Multi-Armed Bandits

- The original form of k-armed bandit problem is named by analogy to a slot machine.
- Rewards are the payoffs for hitting the jackpot.
- Win rate of levers is unknown.
- Discover best bandit by playing and collecting data.
- Balance explore (collecting data) + exploit (playing best-so-far lever)

Action-Value Methods

- Value of an action is the expected or mean reward given that that action is selected.

$$q_*(a) = \mathbb{E}[R_t \mid A_t = a]$$

- Sample average method:
  - A natural way to estimate the true value of an action is the mean reward when that action is selected.

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

Exploit vs. Explore: Action selection rules

- Exploiting:
  - At any time step, always select the action whose estimated value is greatest.
  - Greedy actions.

$$A_t \doteq \arg\max_a Q_t(a).$$

- Exploring:
  - Instead, select one of the other actions, to improve the estimates of the non-greedy actions.

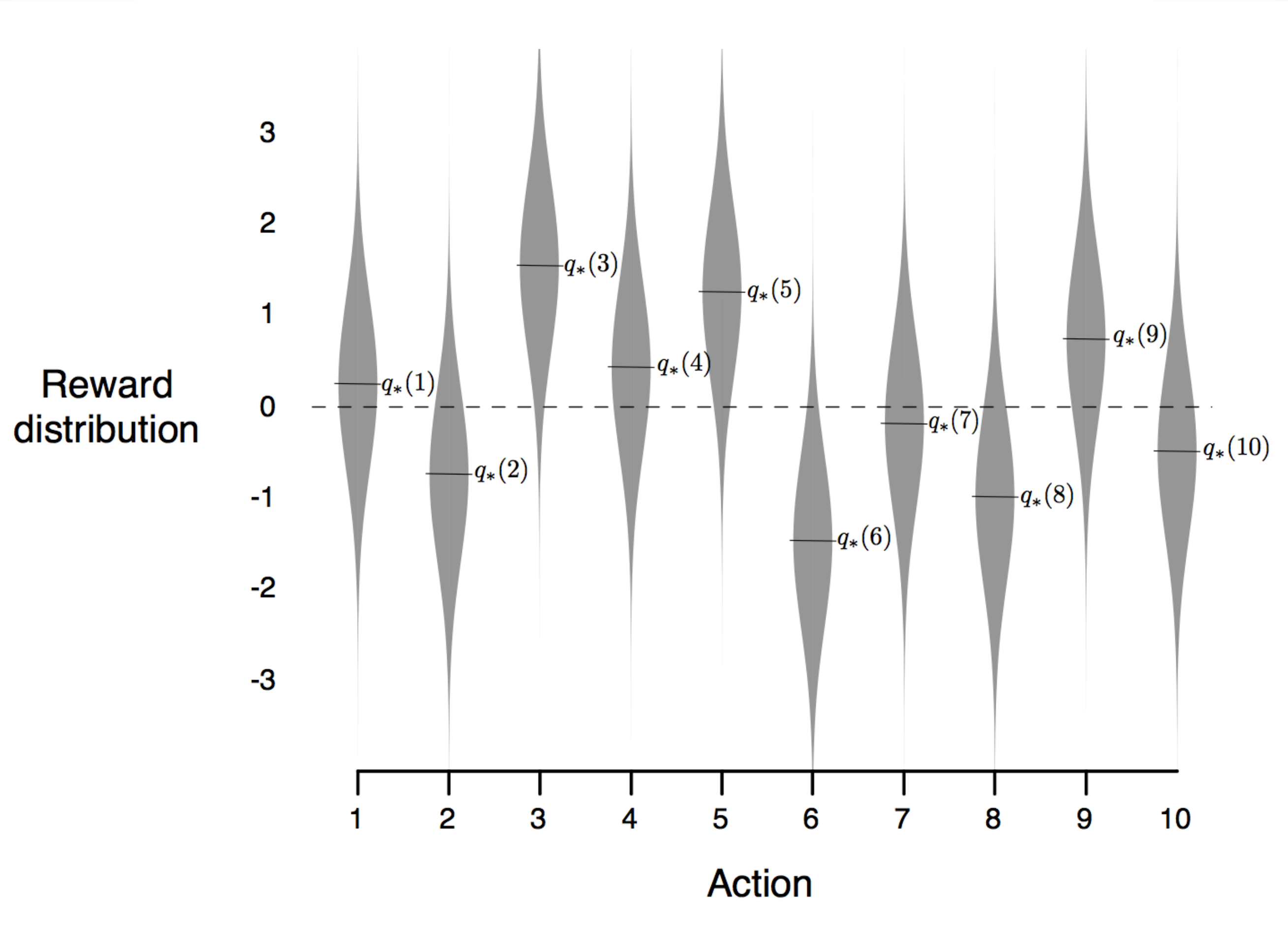
Exploit vs. Explore: Action selection rules

- Epsilon greedy rule:
    - Choose a small number as a probability of exploration
    - Pseudo code:

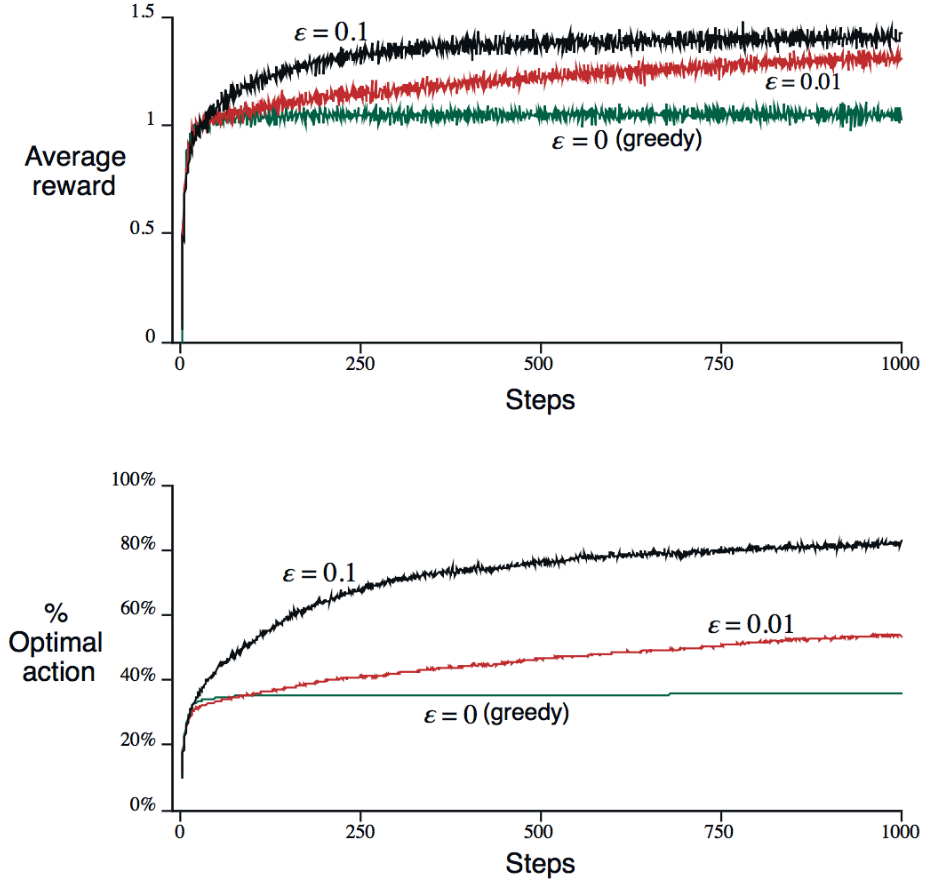
```
p = random()
if p < epsilon:
    pull random arm
else:
    pull current-best arm
```

- Eventually, we'll discover which arm is the true best, since this allows us to update every arm's estimate.

10-armed testbed



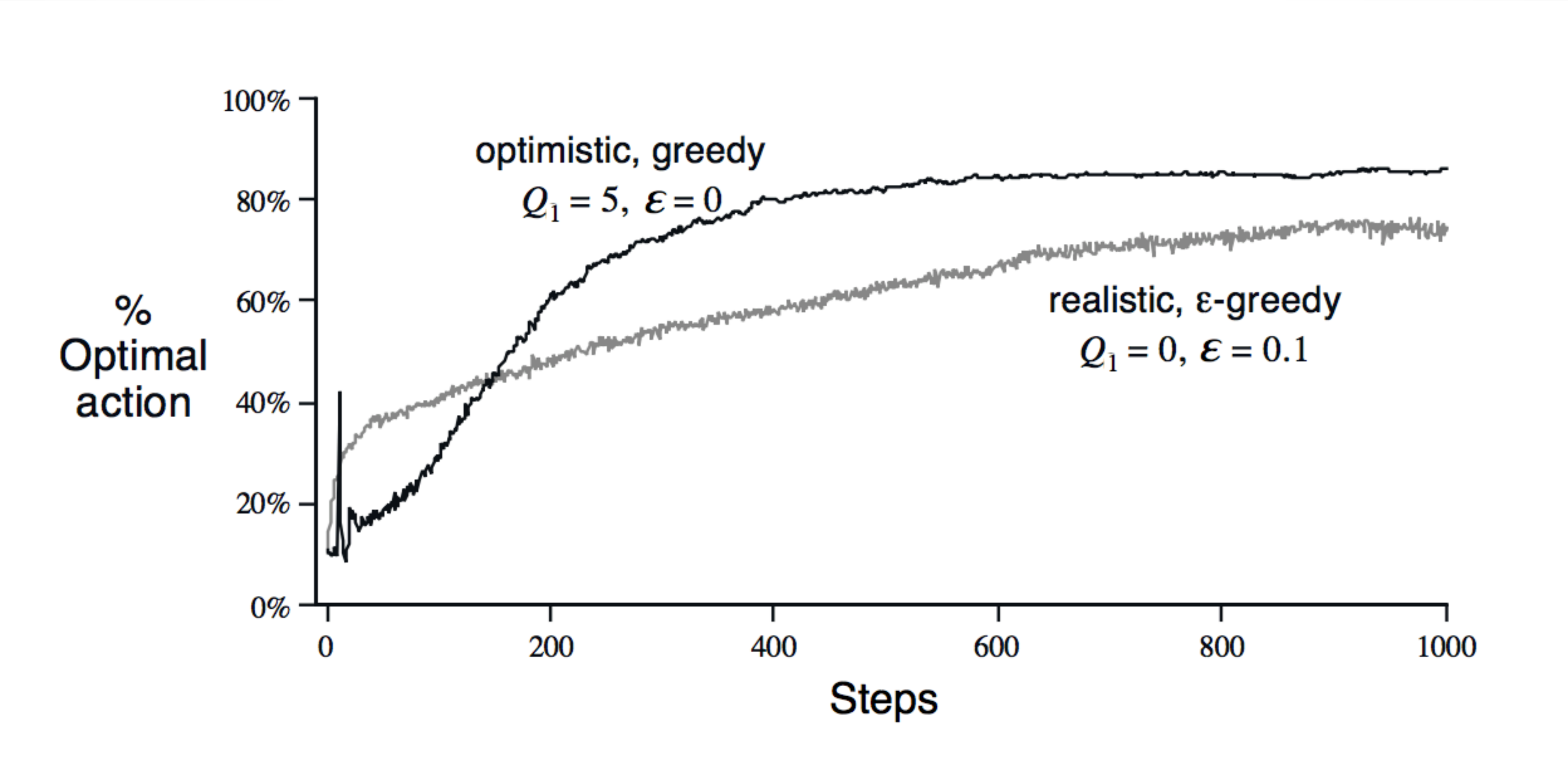
Exploit vs. Explore: Action selection rules



Exploit vs. Explore: Action selection rules

- Optimistic Initial Value:
- Suppose we know the true mean of each bandit is << 10.
- Pick a high ceiling as an estimate.
- If a bandit isn't explored enough, its sample mean will remain high, causing the algorithm to explore it more.
- Even though the initial sample is very high, as the bandit is explored, all collected data will cause the estimate to go down.
- All means will eventually settle into their true values.

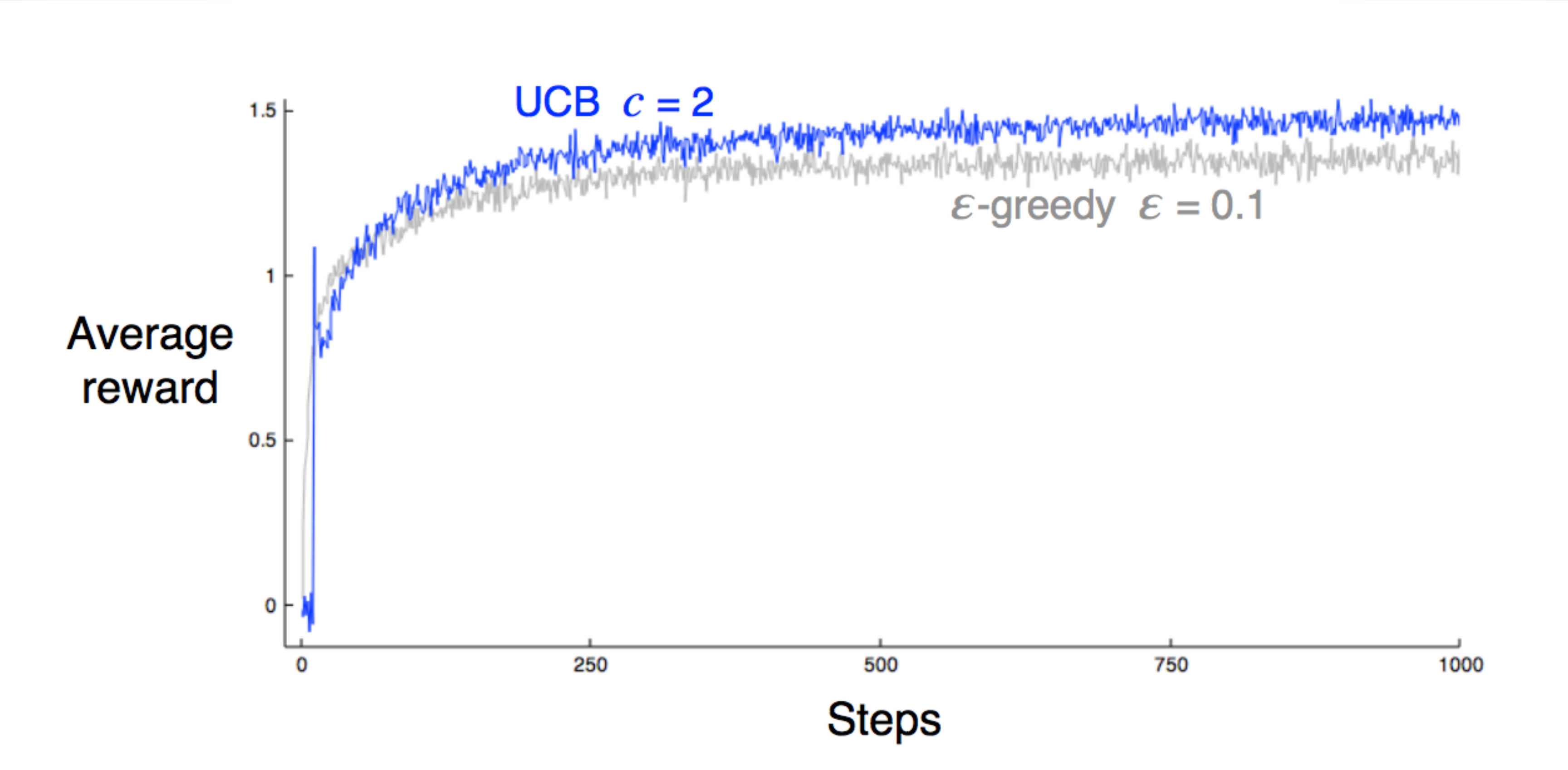
Exploit vs. Explore: Action selection rules



Exploit vs. Explore: Action selection rules

- Upper Confidence Bound: $A_t \doteq \underset{a}{\arg\max} \left[Q_t(a) + c\sqrt{\frac{\log t}{N_t(a)}} \right]$
- Similar to the optimistic initial value, be greedy w.r.t the UCB estimate.
- If $N_t(a)$ is small, the upper bound is high and if it is large, the UCB is low.
- Since log t grows more slowly than $N_t(a)$, enough samples would have been collected by the time the upper bounds eventually shrink.
- Converges to purely greedy.

Exploit vs. Explore: Action selection rules



Action-Value Methods: Incremental Implementation

- Consider the estimate of an action's value after its i^{th} selection

$$Q_n = \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}.$$

- Manipulate to devise incremental formula:

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

Action-Value Methods: Nonstationary problem

- Exponential/Recency-weighted average method.

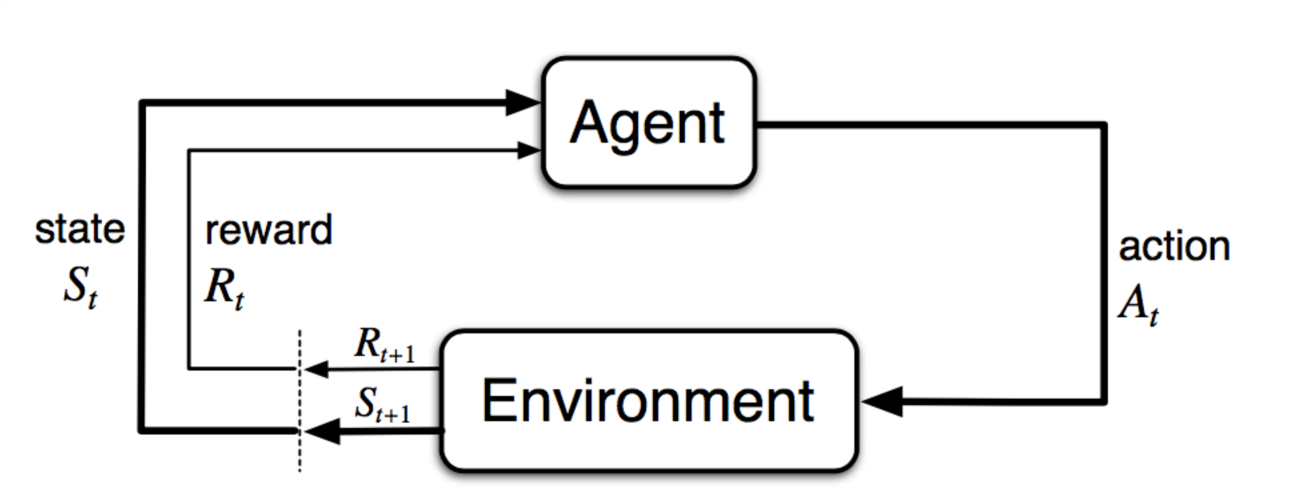
$$
\begin{aligned}
Q_{n+1} &\doteq Q_n + \alpha\Big[R_n - Q_n\Big] \\
&= \alpha R_n + (1-\alpha)Q_n \\
&= \alpha R_n + (1-\alpha)\left[\alpha R_{n-1} + (1-\alpha)Q_{n-1}\right] \\
&= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2 Q_{n-1} \\
&= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2 \alpha R_{n-2} + \\
&\qquad\qquad \cdots + (1-\alpha)^{n-1}\alpha R_1 + (1-\alpha)^n Q_1 \\
&= (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i.
\end{aligned}
$$

Action-Value Methods: Convergence Criterion

- Q will converge for $\sum_{n=1}^{\infty} \alpha_n(a) = \infty$ and $\sum_{n=1}^{\infty} \alpha_n^2(a) < \infty.$
- The first condition is required to guarantee that the steps are large enough to eventually overcome any initial conditions or random fluctuations.
- The second condition guarantees that eventually the steps become small enough to assure convergence.
- Q doesn't converge for a constant step-size parameter.

Reinforcement Learning

- Elements of a Reinforcement Learning problem

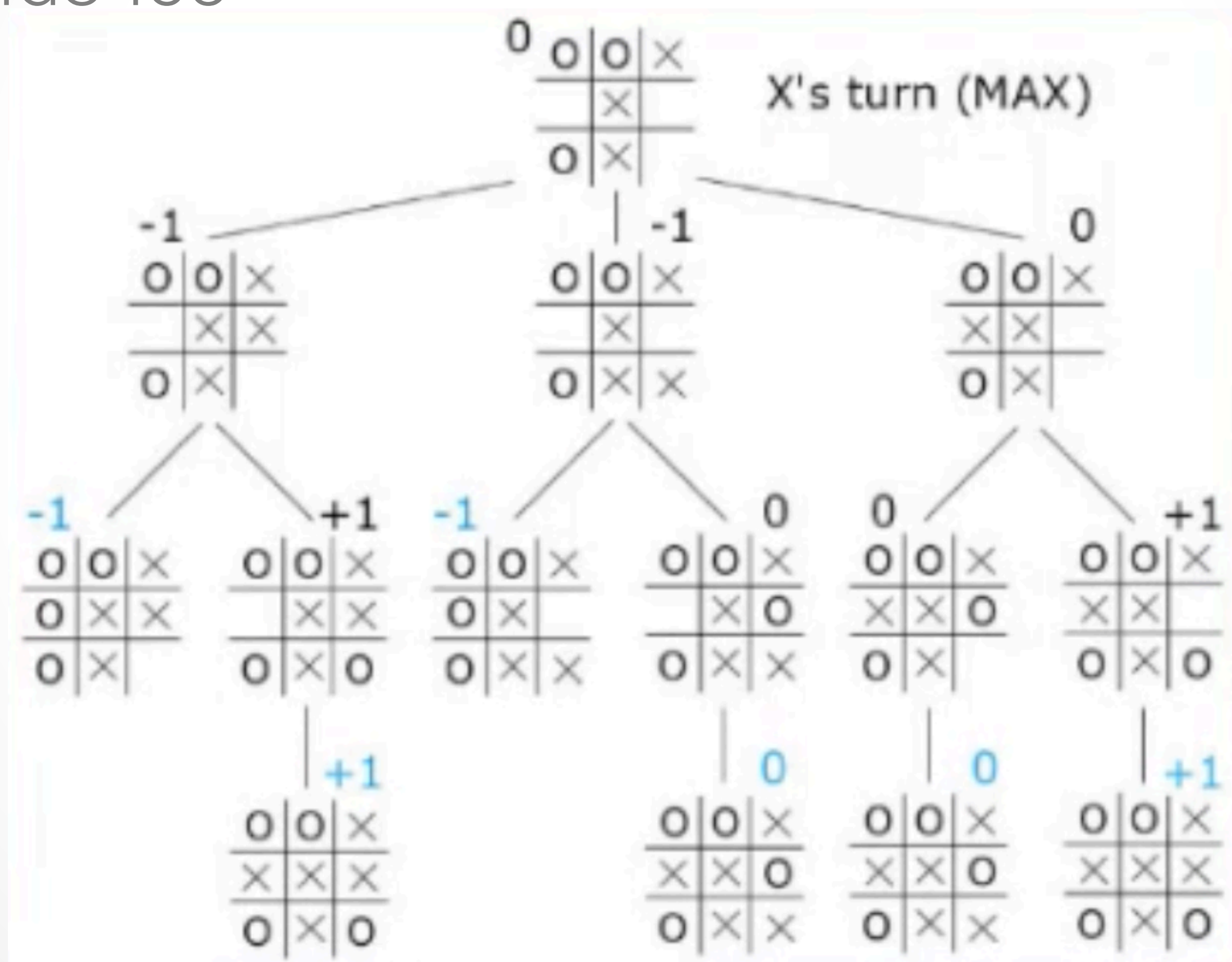


Elements of a Reinforcement Learning problem

- Agent interacts with Environment.
- State is a specific configuration of the environment the agent is sensing (may not be the entire environment)
- Actions are what agents can do that affect its state.
- Actions result in next states along with possible rewards.
- Rewards tell how good the actions were.

Reinforcement Learning: Examples

- Tic-Tac-Toe



Reinforcement Learning: Examples

- Recycle Robot

- At each time step, the robot decides whether it should
  - actively search for a can,
  - remain stationary and wait for someone to bring it a can, or
  - go back to home base to recharge its battery.

- The agent makes its decisions solely as a function of the energy level of the battery.

- The state space is the energy level of the battery = {high, low}

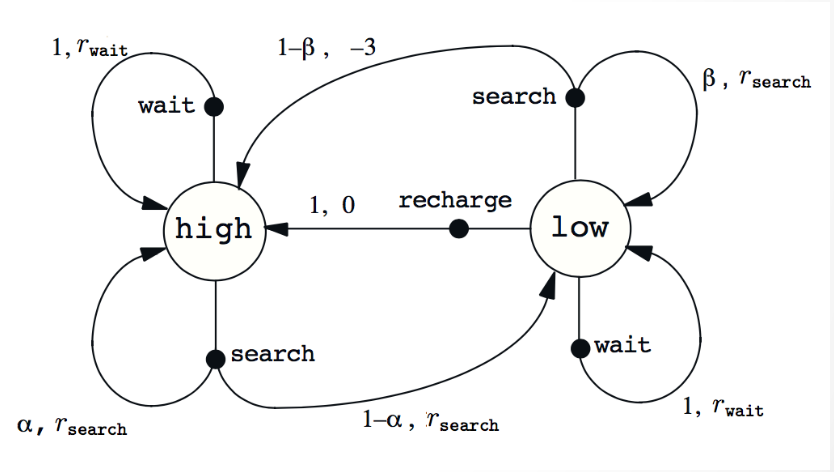
- A(high) = {search, wait}
- A(low) = {search, wait, recharge}

Reinforcement Learning: Examples

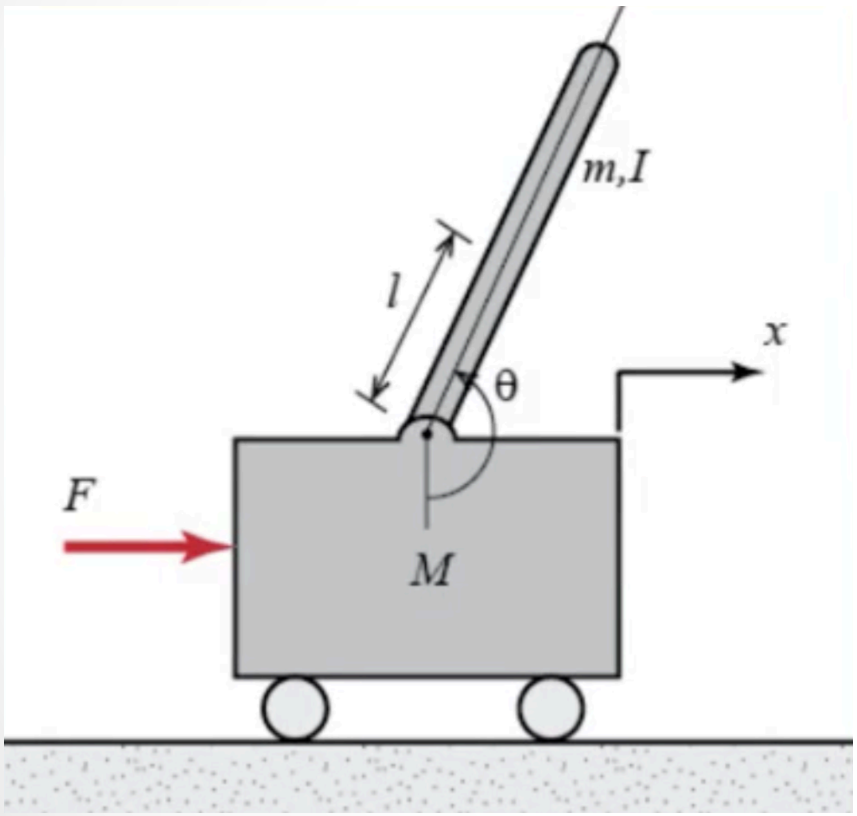
Transition Probabilities

s	s'	a	$p(s'\|s,a)$	$r(s,a,s')$
high	high	search	α	$r_{\texttt{search}}$
high	low	search	$1-\alpha$	$r_{\texttt{search}}$
low	high	search	$1-\beta$	-3
low	low	search	β	$r_{\texttt{search}}$
high	high	wait	1	$r_{\texttt{wait}}$
high	low	wait	0	$r_{\texttt{wait}}$
low	high	wait	0	$r_{\texttt{wait}}$
low	low	wait	1	$r_{\texttt{wait}}$
low	high	recharge	1	0
low	low	recharge	0	0.

Transition Graph



Reinforcement Learning: Examples



- Cart Pole
- Inverted Pendulum
- Unstable system
- Episode starts with pole vertical, falls soon.
- Agent: move to keep the pole within certain angle.
- Continuous state space.

Markov Property

- A state signal that succeeds in retaining all relevant information is said to be Markov.
- Consider how a general environment might respond at time t+1 to the action taken at time t:

$$\Pr\{S_{t+1} = s', R_{t+1} = r \mid S_0, A_0, R_1, \ldots, S_{t-1}, A_{t-1}, R_t, S_t, A_t\}$$

- If the state signal has Markov property, the response at t+1 depends only on the state and action representations at time t:

$$p(s', r | s, a) \doteq \Pr\{S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a\}$$

Markov Property

- From the conditional joint distribution of the state and reward at time t+1, other dynamics of the system such as the expected rewards for state-action pairs and the state transition probabilities can be calculated as:

$$r(s,a) \doteq \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r | s, a)$$

$$p(s'|s,a) \doteq \Pr\{S_{t+1} = s' \mid S_t = s, A_t = a\} = \sum_{r \in \mathcal{R}} p(s', r | s, a)$$

$$r(s,a,s') \doteq \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a, S_{t+1} = s'\right] = \frac{\sum_{r \in \mathcal{R}} r\, p(s', r | s, a)}{p(s'|s,a)}$$

Markov Decision Process

- A Markov Decision Process is defined by:
    - Set of all states
    - Set of all actions
    - Set of all rewards
    - State transition probabilities
    - Discount factor (gamma)
- The idea of a discount factor is to 'discount' the value of a reward that is obtained in the future.
- The goal is to maximize total future reward and the further in the future the reward is, the harder it is to predict.

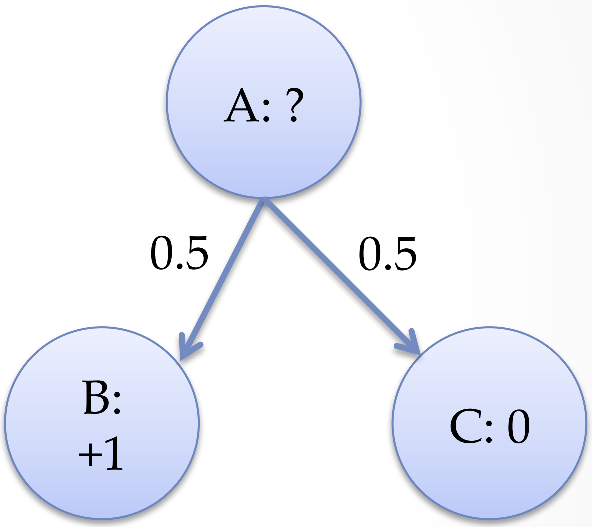
$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

Policy

- Policy is a mapping from from each state and action to the probability of taking an action in a state.
- Policy is what defines what actions to do in what states.
- Technically, not part of the MDP itself, but along with the value function, forms the solution to the problem.
- Examples:
  - Epsilon greedy
  - UCB

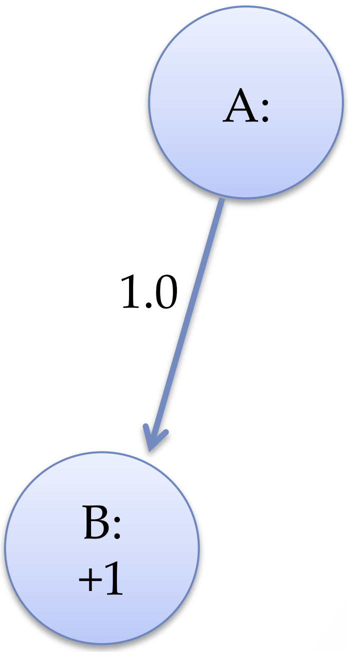
Value Functions

- Two possible states from A: B or C
- 50% chance of ending up in either.
- Value of state A:
  - V(A) = 0.5*1+0.5*0 = 0.5



Value Functions

- Only one possible state from A: B
- Value of state A:
  - V(A) = 1.0*1 = 1.0
- Values tells us the future goodness of a state.



Value Functions

- The value of a state under a policy is defined as:

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

- This is called the state-value function.
- Similarly, we define action-value function as the value of taking an action in a state under a policy.

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

Bellman Equation

- A fundamental property of value functions is that they satisfy certain recursive relationships.

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \,\middle|\, S_t = s\right] \\
&= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \,\middle|\, S_{t+1} = s'\right]\right] \\
&= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \Big[r + \gamma v_\pi(s')\Big], \quad \forall s \in \mathcal{S}
\end{aligned}
$$

Optimal policy; Optimal Value

- Value functions define a partial ordering over policies.
- There is always at least one policy that is better than or equal to all other policies.

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

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

- We can also write the optimal action-value function in terms of the optimal state-value function as:

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]$$

V(s) vs. Q(s, a)

- Finding values given a fixed policy is called prediction problem.
- Finding the optimal policy is called as a control problem.
- The action-value function is better suited for the control problem, since it tells us what the best action is given a state.
- The state-value function requires to perform all the actions to determine the best action.

Solving the MDPs

- Solving the prediction problem
  - Evaluating the values under a given policy
- Solving the control problem

    while not converged:
        evaluate values under current policy
        improve policy by taking argmax over the action-values

- Some methods:
  - Dynamic Programming
  - Monte Carlo methods
  - Temporal Difference methods
  - Approximation methods

Dynamic Programming

- We need to loop through all the states on every iteration.
- Impractical for large and infinite state space problems.
- Calculating the joint distribution of future state and rewards could become infeasible.
- Doesn’t learn from experience.

Monte Carlo Methods

- Unlike Dynamic Programming, Monte Carlo methods learn from experience.
- Expected values can be approximated by sample means.

$$V(s) = E[G(t) \mid S(t) = s] \approx \frac{1}{N} \sum_{i=1}^{N} G_{i,s}$$

- Requires many episodes of experience.
- MC methods can leave many states unexplored.

Temporal Difference Methods

- Estimate returns based on the current value function.
- Instead of calculating the sample mean, TD uses the current reward and the next state value.
- Enables online learning.

Approximation Methods

- DP, MC and TD methods are studied in the context of tabular methods.
- The value functions are stored as dictionaries.
- Can't scale to large and infinite state spaces.
- Use function approximation methods to approximate the values functions instead.

Summary

- Three most important distinguishing characteristics of Reinforcement Learning:
  - Being closed-loop (system's actions influence its later inputs)
  - Not having direct instructions as to what action to take
  - The consequences of actions play out over extended time periods.

- A very important challenge that arise in reinforcement learning and not in other kinds of learning is the trade off between exploration and exploitation.

References

- Richard Sutton and Andrew Barto, *Reinforcement Learning: An Introduction*
  http://incompleteideas.net/sutton/book/the-book-2nd.html

- Andrew Barto, *Reinforcement Learning and its relationship with Supervised Learning*
  http://www-anw.cs.umass.edu/pubs/2004/barto_d_04.pdf

- Andrej Karpathy, *Deep Reinforcement Learning*
  http://karpathy.github.io/2016/05/31/rl/

- *Deep Learning Courses*
  https://deeplearningcourses.com/