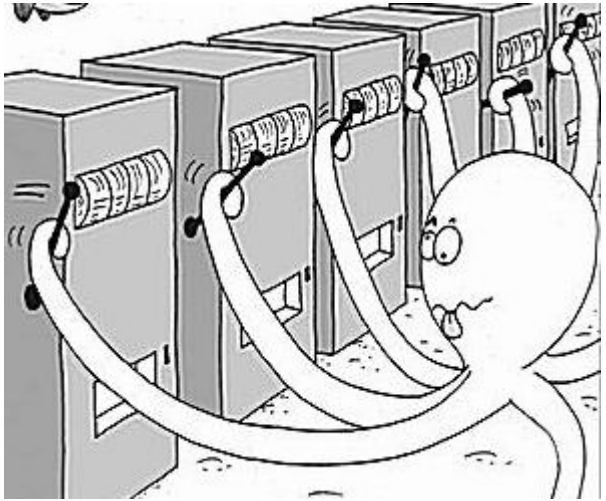
COMP 4752
Computational Intelligence

Lecture 15

Reinforcement Learning

N-Armed Bandit Problem

- Repeatedly make a choice among n different actions
- After each action you receive a reward from a stationary probability distribution depending on the action
- Objective is to maximize your expected total reward over a number of action selections



N-Armed Bandit Problem

- Name is an analogy to slot machines: “One-armed bandit”
- You have a limited amount of money, and you try to win as much as possible

- How do we select which levers to pull?

Exploitation vs. Exploration

- If we maintain an estimate of action values, at any time there is one greatest
  - The ‘Greedy Action’
- Exploitation: Choosing the Greedy Action
  - Maximizes single action returns
- Exploration: Choosing a non-greedy action to improve your action estimates
  - Required for future reward maximization
- How to balance exploitation vs. exploration?

Action-Value Methods

- $Q^*(a)$ = Actual Value of action a
  - “Actual Value” = Mean Reward
- $Q_t(a)$ = Estimate of $Q^*(a)$ after play t

$Q_t(a)$ Sample Average Estimation

- Natural way of calculating $Q_t(a)$ is to average the rewards received so far after a number of plays
- If at play t, action a has been chosen k_a times, yielding rewards r_1, r_2, …, r_{k_a} then:

$$Q_t(a) = (r_1 + r_2 + \ldots + r_{k_a}) / k_a$$

- If $k_a = 0$, define Q_t as some default, $Q_t(a) = 0$
- As k_a gets large, $Q_t(a)$ converges to $Q^*(a)$
- This is called the “sample average” method

Greedy Action Selection

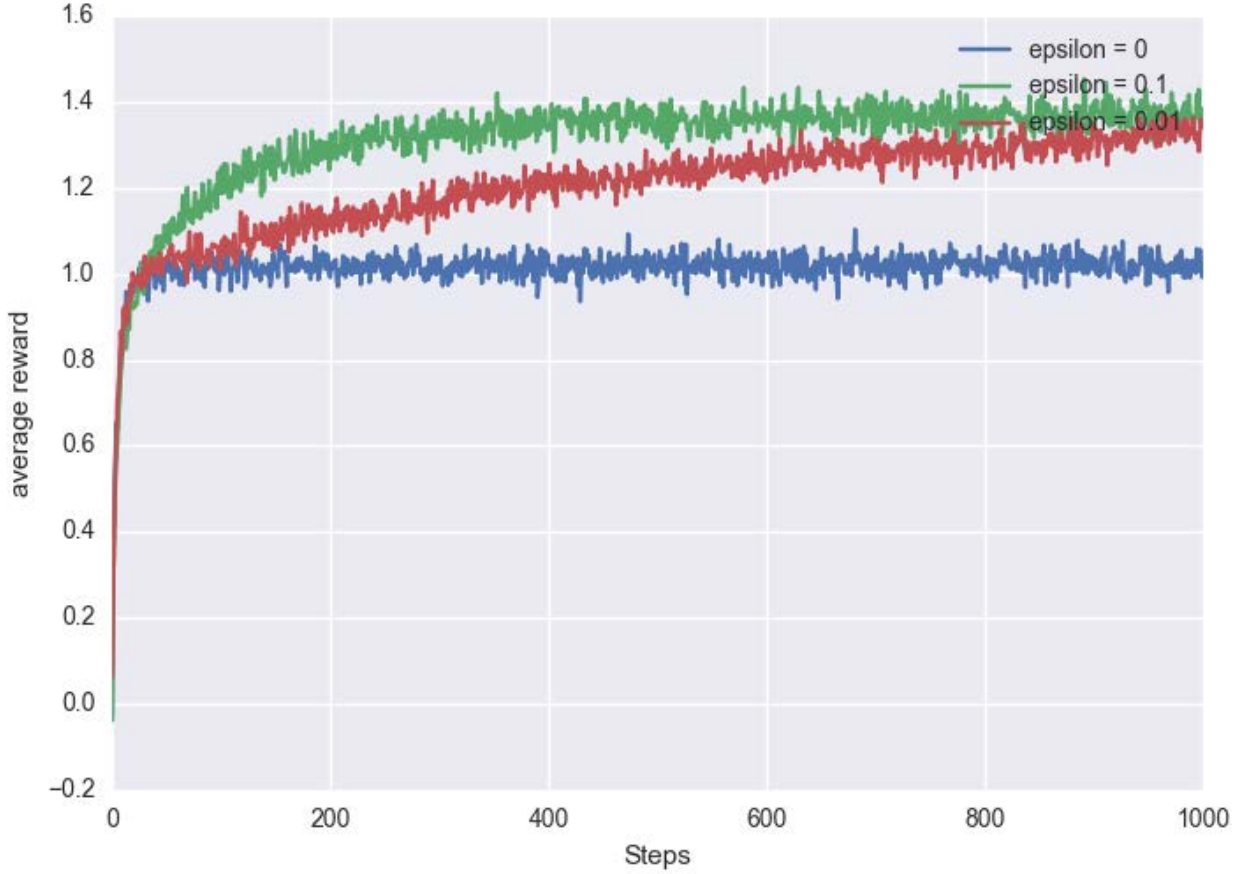
- How to select an action from value estimations?
- Simples way: Greedy Selection
- Select on play t, a greedy action A* for which:

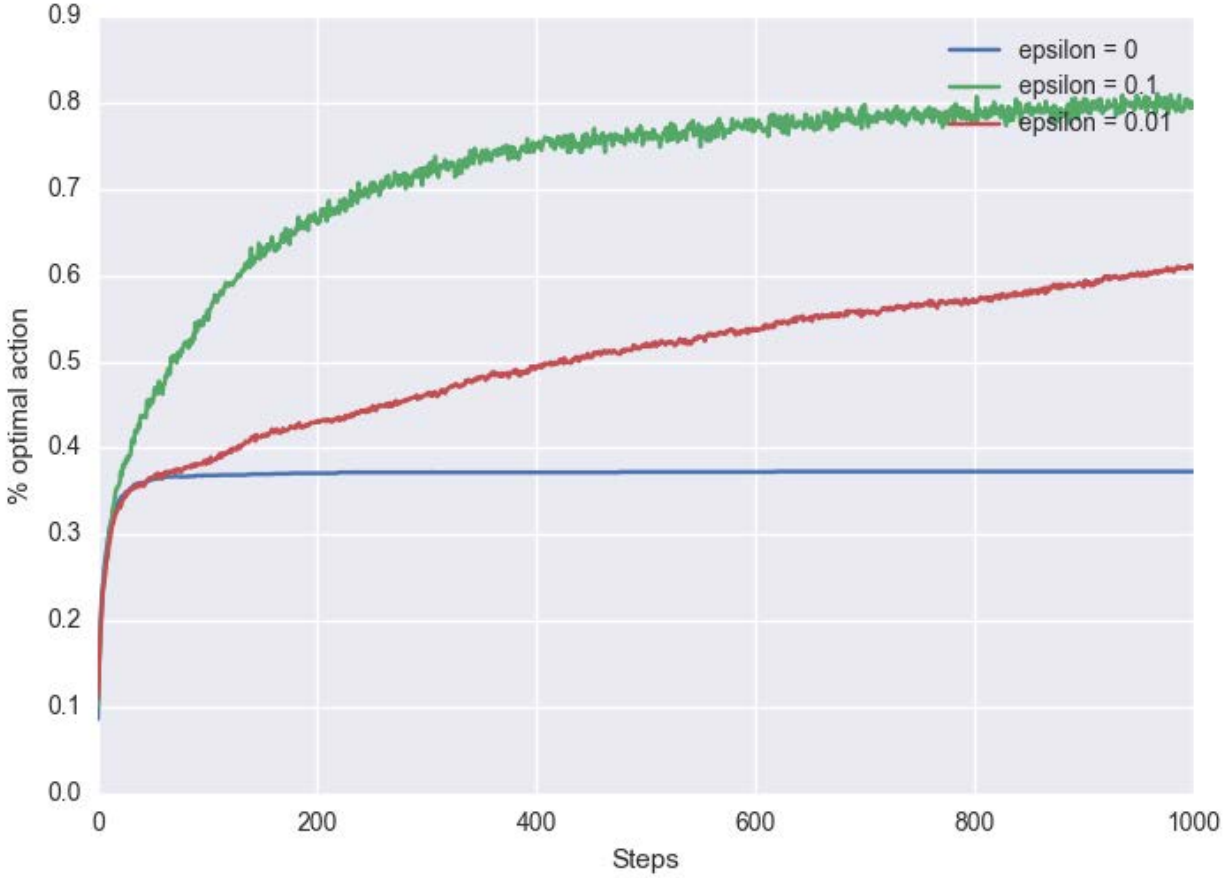
$$Q_t(a^*) = \max_a Q_t(a)$$

- This method always exploits current knowledge to maximize immediate reward
- No sampling or exploration to determine values of another action to see if they may be better

ε-Greedy Selection

- To add exploration, choose a random action with small probability ε
- In the limit, as the number of plays increases, each action will be sampled infinite times
- This guarantees k_a -> infinity, and $Q_t(a)$ converges to Q*(a)
- In theory this works, but in practice it may take a very, very long time to converge


epsilon = 0
epsilon = 0.1
epsilon = 0.01
average reward
Steps
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0
200
400
600
800
1000


epsilon = 0
epsilon = 0.1
epsilon = 0.01
% optimal action
Steps
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
0
200
400
600
800
1000

Incremental Action-Value Est.

- Recall: $Q_t(a) = (r_1 + r_2 + ... + r_{k_a}) \ / \ k_a$
- Problem: Memory and computational requirements grow over time
- Let’s derive an incremental formula so that memory is no longer an issue

Incremental Implementation

Q_k = average of first k rewards

$$Q_{k+1} = \frac{1}{k+1} \sum_{i=1}^{k} r_i$$

$$= Q_k + \frac{1}{k+1}(r_{k+1} - Q_k)$$

NewEstimate = OldEstimate + StepSize(Target-OldEstimate)

Nonstationary Problems

- Averaging works fine for stationary environments, but not if it changes over time
- Want to weight recent rewards more than old ones
- Use a constant step-size parameter $0 < \alpha \leq 1$

$$Q_{k+1} = Q_k + \alpha (r_{k+1} - Q_k)$$

Incremental Update Example

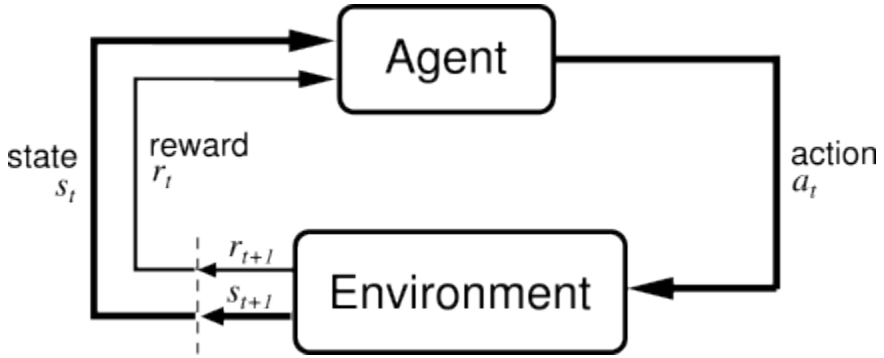
- $Q_{k+1} = Q_k + \alpha (r_{k+1} - Q_k)$
- New estimate pulled toward r_{k+1} by α

- $Q_k = 50, \alpha = 1, r_{k+1} = 100$
- $Q_{k+1} = 50 + 1*(100-50) = 50+50 = 100$

- $Q_k = 50, \alpha = 0.5, r_{k+1} = 100$
- $Q_{k+1} = 50 + 0.5*(100-50) = 75$

Agent-Environment Interface

- Sequence of time steps
  - t=0, 1, 2, 3, …
- At each time step t:
  - Agent perceives state s ∈ S
  - Selections action a_t ∈ $A(s_t)$
  - One time step later gets r_{t+1} ∈ R
  - Finds itself in new state s_{t+1}
- Agent selects action from its policy π
  - General: $\pi_t(s,a)$ = Probability a_t=a if s_t=s



Goals and Rewards

- In RL, purpose is defined by reward r_t
- Reward often used to define ‘goal state’
- RL objective = maximize cumulative reward

- Example: Path-Finding
  - Reward is -1 at every non-goal state
  - Reward is 1 at the goal state
  - Maximizing goal minimizes path length

Returns

- Objective: maximize long term reward
- How can we formally define this?
- Reward Sequence: r_{t+1}, r_{t+2}, r_{t+3}, ...
- Objective: maximize expected return
  - Return R_t is a function of reward sequence

Returns

- Simplest case: Return is sum of rewards:
  - $R_t = r_{t+1} + r_{t+2} + r_{t+3} + ... + r_T$
  - T is the *final time step*

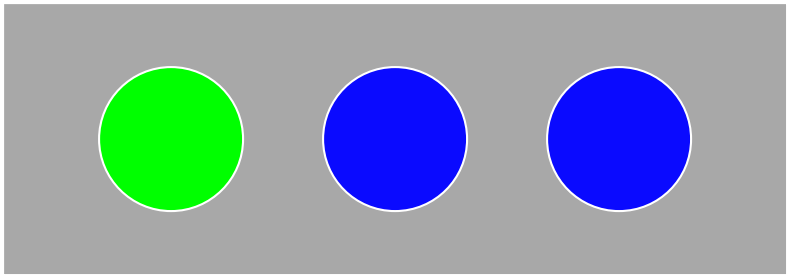
- In general we add a discounting factor γ
  - γ balances importance between present / future reward
  - $R_t = \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + ...$
  - $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$

The Markov Property

- In the RL framework, agent makes decisions as a function of the state of an environment
- Recall: the ‘state’ of an environment is whatever info is available to the agent
- If a state contains all relevant information for making a decision, it is said to be Markov, or to have the Markov Property
- Intuitively: “no state history required”

Markov Examples

- 3 balls in a box, 2 blue 1 green
- One ball is drawn from the box at random
- Next day, someone asks what is the probability of drawing a red ball
- Not Markov



Markov Examples

Markov Property Definition

- In general, environment state and rewards may be defined by action sequence
- $Pr\{s_{t+1}=s', r_{t+1}=r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0\}$

- If Markov Property holds, environment response at t+1 depends only on state and action at t
- $Pr\{s_{t+1}=s', r_{t+1}=r \mid s_t, a_t\}$

Markov Decision Process (MDP)

- An RL task that satisfies the Markov Property is called a Markov Decision Process (MDP)

- If state and action spaces are finite, it is called a finite MDP

MDP Definition

- MDP = (S, A, P, R, γ)
  - S = finite set of states
  - A = finite set of actions (A_s legal from s)
  - $P_a(s,s') = Pr(s_{t+1}=s' \mid s_t=s, a_t=a)$
  - $R_a(s,s')$ = reward after transition s to s'
  - $\gamma \in [0,1]$ = discount factor

MDP Problem / Objective

- Choose a policy that π maximizes Return:
- $R_t = \sum_{k=0}^{\infty} \gamma_k \, r_{t+k+1}$

- Can be solved by dynamic programming
  - Assignment 3 = Grid World DP