Reinforcement Learning

Dr. Fayyaz ul Amir Afsar Minhas
PIEAS Biomedical Informatics Research Lab
Department of Computer and Information Sciences
Pakistan Institute of Engineering & Applied Sciences
PO Nilore, Islamabad, Pakistan
http://faculty.pieas.edu.pk/fayyaz/

Basics

- How can we learn a probability?
  - Maximum Likelihood Estimation
  - Example
    - Given a coin, we want to estimate what is the probability of it giving “heads”
    - What is the actual probability p
    - How can it be estimated?
      - By tossing the coin N times
      - Counting the number of times it lands “Heads” H
      - The estimate is: H/N

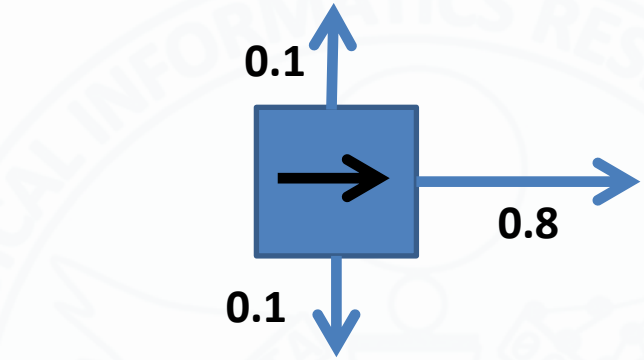
Basics

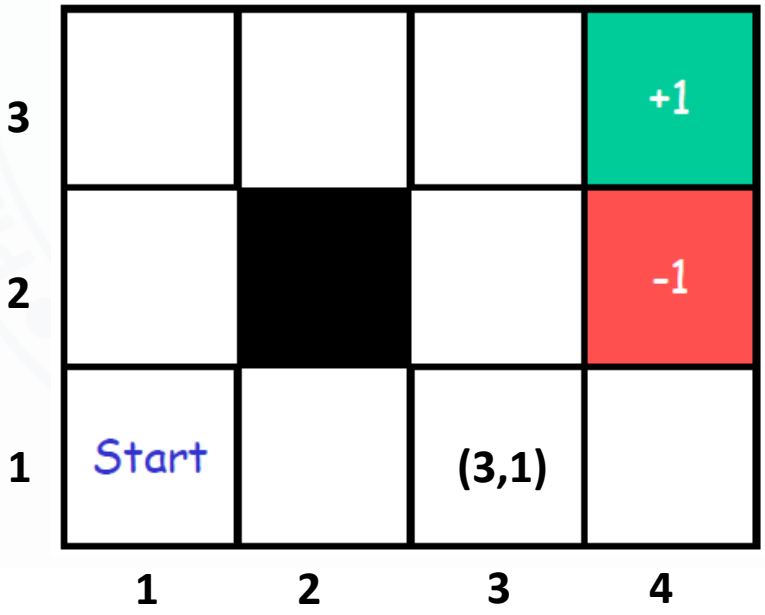
- Running Average

$$CMA_{n+1} = \frac{x_{n+1} + n \cdot CMA_n}{n+1}$$

- K-Armed Bandit
  - Assume a machine than when one of its K buttons is pressed, gives you M_k dollars associated with that button
  - Each time the button is pressed it may return slightly different rewards
  - Your objective is to make the most money in 1000 trials
  - Which button will you press?
    - In the beginning, we would be pressing different buttons randomly
    - Over time, we will find out the optimal button
    - Exploration vs. Exploitation

Environment





Revision

- Agent in a Markov Decision Process
  - A set of states S
  - A set of actions in each state A(s)
    - An action causes a transition
  - A transition probability model (may be unknown!)
    - P(s'|s,a) is the probability of reaching state s' from s by action a
  - Each state has a reward R(s)
  - The probability of reaching a state is dependent only on the previous state and the action taken in that state

Revision

- The solution is represented in terms of a policy
  - Tells the action to be taken in that state: $\pi(s)$
- For a given policy, each state has a utility
  - Equal to the reward of that state and discounted rewards for future states (i.e., expected utility of the next state)

$$U^{\pi}(s) = \mathrm{E}[\sum_{t=0}^{\infty} \gamma^t R(S_t)] = \mathrm{E}[R(s) + \sum_{t=1}^{\infty} \gamma^t R(S_t)]$$

$$= \mathrm{E}\left[R(s) + \gamma \sum_{t=0}^{\infty} \gamma^t R(S_{t+1})\right] = R(s) + \gamma U^{\pi}(next)$$

$$U^{\pi}(next) = \sum_{s' \in S} P(s'|s, \pi(s)) U(s')$$

Revision

- Objective of MDP
  - Find an optimal policy
  - How?
    - By using the Maximum Expected Utility Principle
      - Pick a policy that maximizes the expected utility
        - » $\pi^* = argmax_\pi U^\pi(s)$
    - To find the optimal policy when the transition model is known, we can use:
      - Value Iteration
      - Policy Iteration
  - What to do when the transition model is not known?
    - Some how “learn” it!!

Reinforcement Learning

- How can an agent learn from success and failure?
  - The agent is placed in an environment and must learn how to reach a goal
    - Encompasses all of Artificial Intelligence
- Objectives
  - Passive RL
    - Fixed policy, learn the utilities
  - Active RL
    - Find the policy too
      - The agent must “explore” the environment and the effects its actions have on the environment

Passive Reinforcement Learning

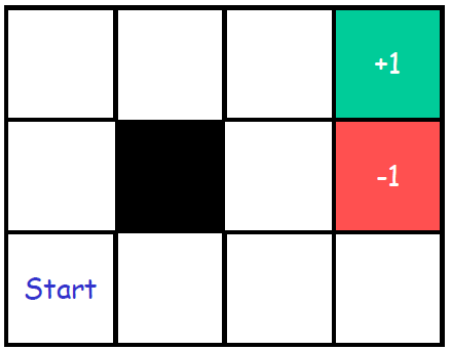
- Given: Policy, Reward Function, States, Actions
- Unknown: Transition model
- Objective: Estimate utilities of different states
- Techniques
  - Direct Utility Estimation
  - Adaptive Dynamic Programming
  - Temporal Difference Learning

Active Reinforcement Learning

- Given: Policy, Reward Function, States, Actions
- Unknown: Transition model
- Objective: Optimal Policy and a “Q” function that maps actions in a state to utilities
- Techniques
  - Q-Learning
  - SARSA
  - Policy Estimation

Direct Utility Estimation

- For a given policy, generate a number of trials
  - Each trial is a sequence of states terminating in a terminal state
  - The agent knows the state it is in and the reward it gets in each state
- Examples
  - T1: $(1,1)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (2,3)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (4,3)_{+1}$
  - T2: $(1,1)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (2,3)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (3,2)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (4,3)_{+1}$
  - T3: $(1,1)_{-0.04} \rightarrow (2,1)_{-0.04} \rightarrow (3,1)_{-0.04} \rightarrow (3,2)_{-0.04} \rightarrow (4,2)_{-1}$



Direct Utility Estimation

- The utility of a state is the expected total reward received from that state onwards
- Based on the 3 trials what is the utility of every state?
  - Use a running average $$CMA_{n+1} = \frac{x_{n+1} + n \cdot CMA_n}{n+1}$$

Trial\State	(1,1)	(1,2)	(1,3)	…
T1:	0.72	(0.84,0.76)	0.88	…
Utility Estimate	(0.72,1)	(0.8,2)	(0.88,1)	
T2:	0.72	0.76	0.88	
Utility Estimate	(0.72,2)	(0.786,3)	(0.88,2)	
…				

Direct Utility Estimation

- Over a large number of trials, the estimates will converge to the optimal values

- Assumption:
  - Utilities of states are independent
- Requires large number of trials

Adaptive Dynamic Programming

- Estimates the transition probabilities
- And then uses policy evaluation

POLCY EVALUATION

Repeat "k" times
for s in S:

$$U(s) = R(s) + \gamma \sum_{s' \in S} P(s'|s, \pi_i(s)) U(s')$$

Adaptive Dynamic Programming

function PASSIVE-ADP-AGENT(*percept*) **returns** an action

inputs: *percept*, a percept indicating the current state s' and reward signal r'

static: π, a fixed policy

mdp, an MDP with model T, rewards R, discount γ

U, a table of utilities, initially empty

N_{sa}, a table of frequencies for state-action pairs, initially zero

$N_{sas'}$, a table of frequencies for state-action-state triples, initially zero

s, *a* the previous state and action, initially null

if s' is new **then do** $U[s'] \leftarrow r'$; $R[s'] \leftarrow r'$

if *s* is not null, **then do**

increment $N_{sa}[s,a]$ and $N_{sas'}[s,a,s']$

for each *t* such that $N_{sas'}[s,a,t]$ is nonzero do

$T[s,a,t] \leftarrow N_{sas'}[s,a,t] \,/\, N_{sa}[s,a]$

$U \leftarrow$ POLICY-EVALUATION(π, *U*, *mdp*)

if TERMINAL?[s'] then *s*, *a* ← null **else** *s*, *a* ← s', $\pi[s']$

return a

Update reward function

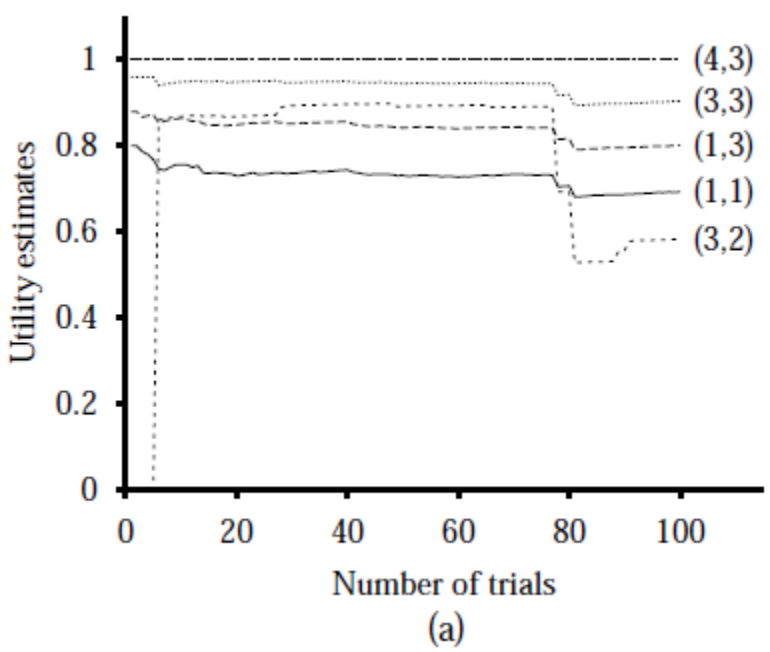
Update transition model

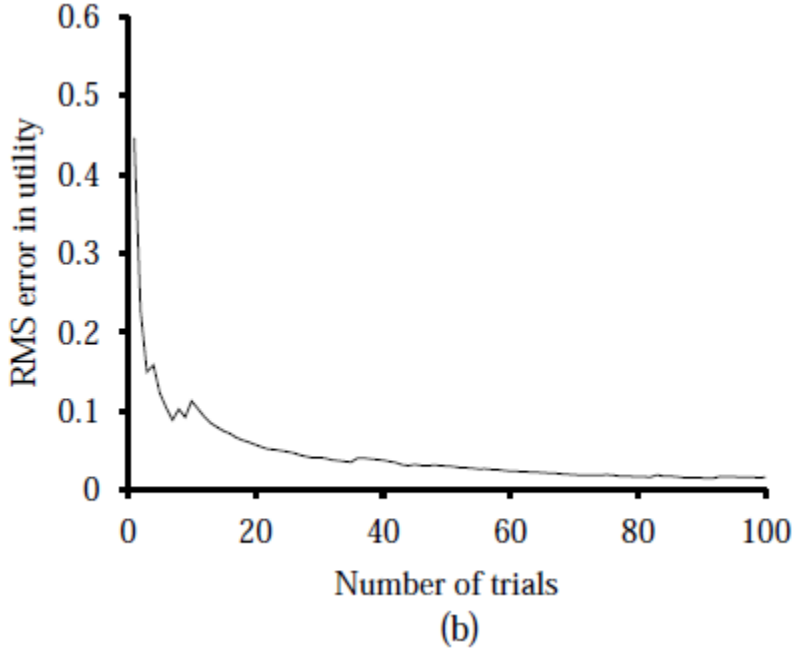
Adaptive Dynamic Programming

POLCY EVALUATION

Repeat "k" times
for s in S:

$$U(s) = R(s) + \gamma \sum_{s' \in S} P(s'|s, \pi_i(s))U(s')$$





Temporal Difference Learning

- Use observed transitions to adjust the utilities of the observed states so that they agree with the Bellman Equation
- Assume that we end up in state s' from state s, then we know that

$$U^{\pi}(s) = R(s) + \gamma U^{\pi}(s')$$

- If we have estimated values of $U^{\pi}(s')$ and $U^{\pi}(s)$, then we can update the estimate of $U^{\pi}(s)$ by using a linear combination of the previous value of $U^{\pi}(s)$ and $R(s) + \gamma U^{\pi}(s')$ as follows

$$U^{\pi}(s) \leftarrow (1-\alpha)U^{\pi}(s) + \alpha\big(R(s) + \gamma U^{\pi}(s')\big)$$

Or equivalently,

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

α is called the learning rate and it should be decreased as more and more samples are collected because our confidence in the estimate grows.

Example

- Assume that an action in a trial takes us from (1,3) to (2,3) and our current estimates are:

  U(1,3) = 0.84

  U(2,3) = 0.92

  Then the 'new' value of U(1,3) is:

  U(1,3) = -0.04 + 0.92 = 0.88

  Thus, the updated value of U(1,3) will now be based on 0.88 and 0.84 and will be:

$$U^{\pi}(1,3) \leftarrow U^{\pi}(1,3) + \alpha\big(R(1,3) + \gamma U^{\pi}(2,3) - U^{\pi}(1,3)\big)$$

Temporal Difference Learning

function PASSIVE-TD-AGENT($percept$) **returns** an action

inputs: $percept$, a percept indicating the current state s' and reward signal r'

persistent: π, a fixed policy

U, a table of utilities, initially empty

N_s, a table of frequencies for states, initially zero

s, a, r, the previous state, action, and reward, initially null

if s' is new **then** $U[s'] \leftarrow r'$

if s is not null **then**

increment $N_s[s]$

$U[s] \leftarrow U[s] + \alpha(N_s[s])(r + \gamma\, U[s'] - U[s])$

if s'.TERMINAL? **then** $s, a, r \leftarrow$ null **else** $s, a, r \leftarrow s', \pi[s'], r'$

return a

TD-Agent

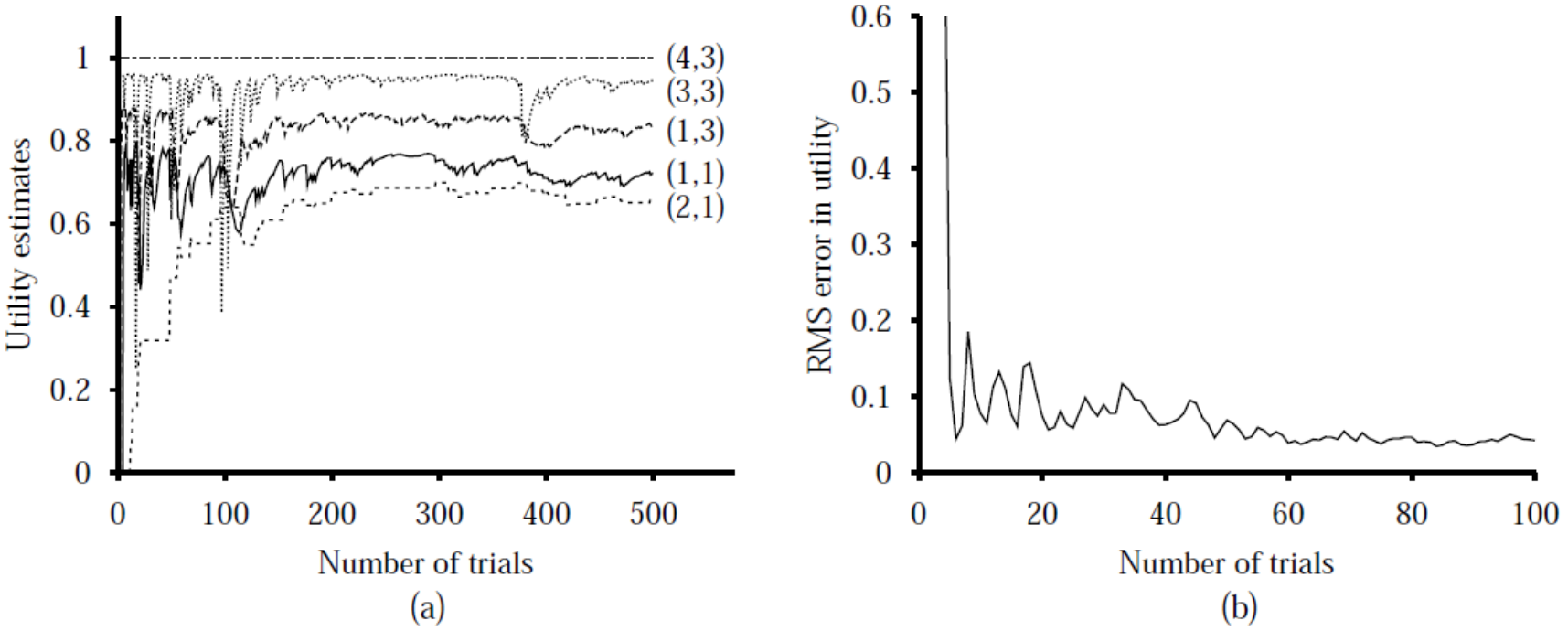


Figure 21.5 **FILES: .** The TD learning curves for the 4×3 world. (a) The utility estimates for a selected subset of states, as a function of the number of trials. (b) The root-mean-square error in the estimate for $U(1,1)$, averaged over 20 runs of 500 trials each. Only the first 100 trials are shown to enable comparison with Figure 21.3.

TD-Agent

- Note that the TD-Agent does not learn the transition model and hence TDRL is also called ‘model free’ learning

Active Reinforcement Learning

- We do not know the model, the utilities or the policy
- How can this be done?
  - Pick a policy at random
  - Follow the policy to find the transition/utility values
  - Update the policy
  - Follow the policy to find the transition/utility values
  - …
- Uses the current policy to make moves in estimating the transition/utility values
  - But the estimates are not correct
  - Therefore, the policy will not be correct
  - The method will get stuck at a suboptimal policy or it might not explore all options
    - It's afraid of trying new things!!!

Active Reinforcement Learning

- Thus, the agent should first ‘explore’ different options before settling or ‘exploiting’ the optimal one
- Initially, the policy should have an element of randomness which gradually reduces over time as the agent becomes more and more confident in its view of the world
- How to implement this?

Active Reinforcement Learning

- We know that in the event there is no optimal policy, we have

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

  - See: Value Iteration
- To allow 'seemingly' suboptimal actions, we change the above equation to

$$U(s) = R(s) + \gamma max_{a \in A(s)} f\left(\sum_{s' \in S} P(s'|s,a)U(s'), N(s,a)\right)$$

$N(s,a)$ is the frequency of state-action pair (s,a) and $f(u,n)$ is a function called the exploration function. It controls the tradeoff between greed and curiosity. It should be increasing in u and decreasing in n.

The exploration function

- The exploration function allows an otherwise suboptimal action to be optimal if it has not been sampled enough times by returning an optimistic utility in such cases

$$U(s) = R(s) + \gamma max_{a \in A(s)} f\left(\sum_{s' \in S} P(s'|s,a)U(s'), N(s,a)\right)$$

- Example

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

Q-Learning

- In TD-Learning, we already had a policy and we knew the action to be taken. This gave us the following update equation

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

- However, in active learning we must also obtain what the optimal action is. So instead of utilities, which are specific to states, we use an action-utility representation $Q(s, a)$ which is directly related to the utility of the state by

$$U(s) = max_a Q(s, a)$$

Q-values vs. Utilities

- Utility is for a state, Q-value is for a state-action pair
- Utility tells us the desirability of a state, Q-value tells us the desirability of an action in a state
- Similar to utility values

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

- Bellman Equations can also be written for Q-values

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

- Notice that: $U(s) = max_a Q(s,a)$

Q-Learning

- Similar to TD-Learning, the update equation becomes

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

  – At each step, pick the action that you think is the best based on the Q value estimates

- SARSA

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

  – Uses the Q value of the action a' that is *actually* taken in state s'

Pseudo code

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 values for state-actions, initially zero

N_{sa}, a table of frequencies for state-action pairs, initially zero

s,a,r, the previous state, action and reward, initially null

If Terminal?(s) **then** $Q(s', a') \leftarrow r'$

If s is not null **then**

increment $N_{sa}(s, a)$

Update:$Q(s, a) \leftarrow Q(s, a) + \alpha(N_{sa}(s, a))\big(R(s) + \gamma max_{a'} Q(s', a') - Q(s, a)\big)$

s,a,r $\leftarrow$ s', $\arg max_{a'} f\big(Q(s', a'), N_{sa}(s', a')\big), r'$

Return a

Q-Function Approximation

- Instead of using a table, we can also use a function approximation scheme such as a Neural network to return Q(s,a) given a representation of s and a

Policy Search

- Directly search for a policy that maximizes the utility/Q-value

Coding

- Coding
  - https://github.com/aimacode/aima-python/blob/master/rl.ipynb

Applications

- Robot Walking
- Game Play
  - Algorithm plays against itself to learn what moves to take in different states
  - Uses Neural Networks
- Web document searches
  - Minimizes the number of web-crawls
  - Only crawl when needed

Dynamic Marble Control

End of Lecture

Humans Are the World's Best Pattern-Recognition Machines, But for How Long?

http://bigthink.com/endless-innovation/humans-are-the-worlds-best-pattern-recognition-machines-but-for-how-long