Reinforcement Learning

Lecture Notes

Machine Learning, CMPT 882

Instructor: Dr. Oliver Schulte

Spring 2004

Scribe: Mohamed Soliman

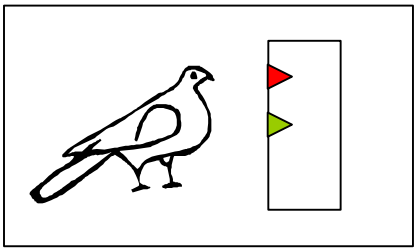
March 18th, 23rd, and 25th, 2004

Contents

Introduction
  The Backgammon World
  Example: TD-GAMMON
Control Learning, in General
Reinforcement Learning Problem
  Discounted Cumulative, infinite horizon:
  The Average Reward Model
Markov Decision Processes
Agent's Learning Task
  Definitions
  Challenges
Computing Optimal Policy Given Model
  Simple Situation
  Bellman Backup - Example
  Behavior of Value Iteration
Model-Free Learning: Q Function
  Q-Learning Algorithm (Deterministic World)
  Convergence
Final Notes:

Introduction

We want to think about learner acting in the environment and how the learner will achieve it. (What actions to take). Some observations are obtained throughout studies in analyzing the behavior of animals and how this can help to obtain a different model of machine learning. Consider an example training a pigeon.



In this experiment, a pigeon is presented with a board that has two specks; green and red. The environment is set so as to give the bird food only when it specks on the correct spot.

If Red speck → food
If Green speck → no food

Then how long does it take the pigeon to figure out how to get food? We want to know exactly how. In this case the pigeon learns while doing actions. When the action leads to a reward (food), the bird will continue to speck on red so as to obtain the reward (food). This method of learning is learning by reward and penalties (no food on green specks). It is also about **combing actions with learning.** As long as there is no direct teacher, the bird will have to learn from its experience on acting within its surrounding environment. Eventually, the bird will do more specks on the red. We can abstract the above as:

	FEATURE	**ACTION**		**REWARD**
If	**Red**	**Specks**	→	**Food (Reward)**
If	**Green**	**Specks**	→	**No Food (No Reward)**

State
We wanted the bird to learn causal relationships; given feature F, action A causes reward R. Similarly the learner can perceive a combination of features and comprehend them as a *state* (Example: there will be food only when specks on red while the bell is ringing). Based on state, in reinforcement learning, learners will choose the action that can give the maximum reward.

Reward

In the above example, learning occurred by giving the learner reward for choosing certain actions. The learners' behavior will be directed towards actions that produce most rewards. When the learner repeats the action while receiving the reward (food) one time after the other, the motivation of repeating the action will be reinforced. Therefore, having values for rewards and determining their values for the desired actions are central to this class of learning problems[1].

Policy

Also in this example, the animal observes the results of taking simple actions then evaluates the result to reach the goal in subsequent actions. Generally, we can think of a plan of action (called ***Policy***) instead of a single action that gives the best reward. Moreover, the reward may be delayed. For example, walking across the corridor, there are multiple decisions (actions) to be taken to reach a certain destination (reward).

Computational Problem:
How to find the course of actions (policy) that maximizes the overall reward?

As long as we are concerned with the course of actions (policy), the learner will observe short-term versus long-term rewards. In general, the more we know the outcomes (rewards and penalties), we can anticipate the best possible course of actions (policy) in as long-term as possible.

Generally, the learner will be an agent that tries to find the policy that maximized the overall reward in the long run. Therefore, the agent will be faced with finding the single step actions while looking for the delayed reward. Finding the immediate reward based on a single-step actions is named the ***credit assignment problem.***

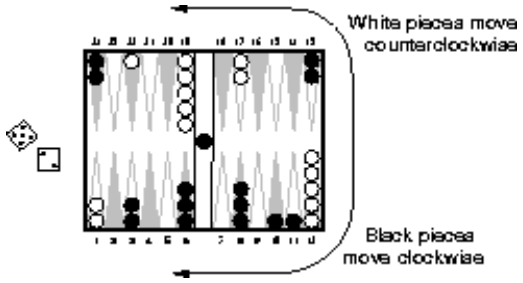
Therefore, the decision will be how far to know about the long-term[2] reward based on the sequence of actions that the learner would take. This will be discussed in the next section. Let's take a look at another example, backgammon playing.

[1] A related problem called "The Credit Assignment Problem" is about determining which actions caused what rewards.

[2] In Reinforcement Learning, there is the notion of the discount factor, discussed later, that captures the effect of looking far in the long run

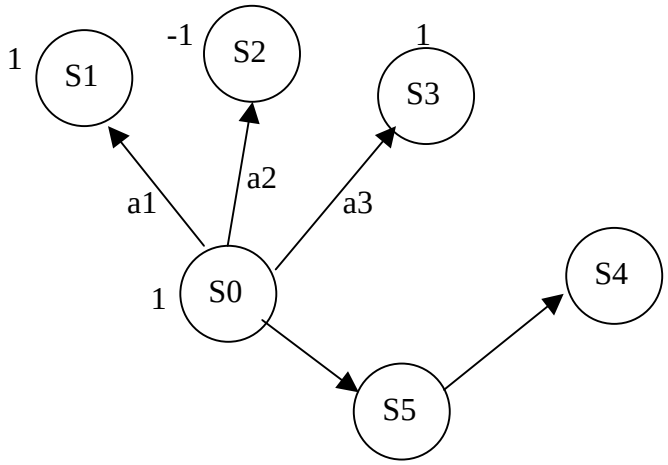
The Backgammon World

Let's consider learning to play backgammon using reinforcement learning. In this game, each of two players in turn rolls two dices and moves two of 15 pieces based on the total amount of the result. This process will be repeated until reaching the end of the backgammon board where the player will start removing the pieces. The winner is the player who removes all the 15 pieces first.



A Backgammon position [Sutton and Barto]

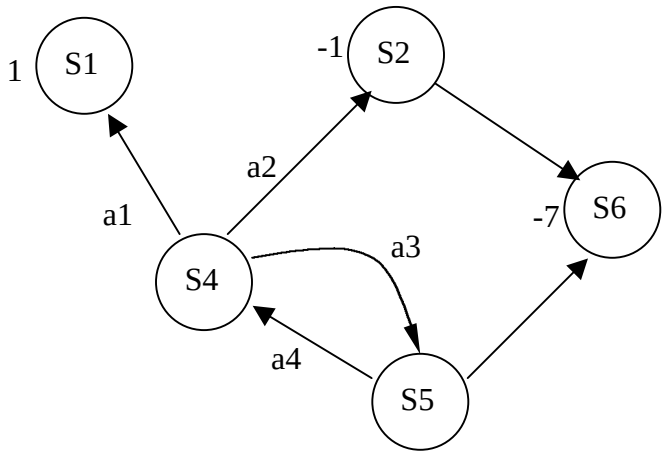
The complexity of this game arises as each player has options on moving the pieces for different possible locations. When a white piece hits a black piece, the black piece will be placed in the middle to re-enter the race from the beginning. This is a penalty to the black player as it reduces the chance of winning. We can model this as a reinforcement learning problem. What we need is the course of action (policy) on how to play to win the game. (States = Board Configuration). We can also think (from each player perspective) about the states and the transitions based on a single move. For example:



Transitions from one state to another occur by the player taking an action (rolling the dice and moving pieces). Such action can lead to win (good move) or the player can take a bad move. We want the learner to learn to win the backgammon game. It is possible to assign values to state transitions so that when it moves to a new state (by move), the player can get a positive value (reward) or be punished by receiving a negative (A move that permits

a hit by the other player). Therefore, if the learner tries to win the game by accumulating rewards, it will do the actions that lead to a win.

For example, state transitions could be sketched as follows:



We can indicate the states of wins of being absorbing states with rewards while loss states are absorbing states with losses (negative reward). If the player is at state S5, he will try to move to state S4 in order to get a better reward than moving to state S6 which moves it closer to the win state. Notice that we also want to discourage loops.

Since another player is involved, one player can't anticipate which state (or sequence of states) she will experience in response to a move(s). Therefore, we can base reward calculation upon the utility of being at certain a state to be equal to the chance or probability of winning.

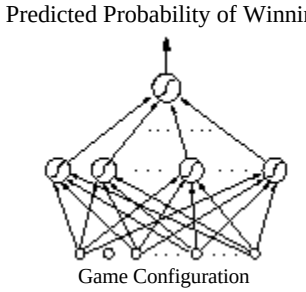
Example: TD-GAMMON

Based on the results by Gerry Teasauro 1995, and since then, TD-GAMMON probably became the most famous Reinforcement Learning application. The TD-gammon program learns to play the backgammon game. It has the following characteristics:

- Gives 100 as an immediate reward for wins
- Give a penalty of 100 (immediate reward of –100) for lose and
- 0 for all other states

The TD-gammon starts from a random play and it is trained by playing many times (1.5 million games) against itself. Currently, it has a winning rate comparable to best human players.

Since there are two players each has 15 pieces with several alternatives according to results of the dice roll, the state space will be enormous. Also, finding the immediate action if win or lose is difficult to obtain. We need a function approximation technique to obtain state utility (probability of winning) based on the game configuration. TD-gammon used a backpropagation neural network since neural networks are proven to provide good function approximations. The resultant prediction can be used to predict the rewards according to certain sates. It is easy to determine the utility of the final state.



For example, the values obtained from NNET for two states are:

Value (S_{10}) = 30%
Value (S_{11}) = 11%

Therefore, the learner will prefer the action that leads to $S_{10,}$ which has a higher utility. In general, the RI agent will take the action leading to state with maximum utility. Here RI is used as well to compute the utility for intermediate states (e.g. a result of a single move in backgammon).

Control Learning, in General

Reinforcement Learning is suitable for a class of problems that requires learning for control. I.e. learning to choose actions, eg,

- Learning to play game (backgammon, chess, Tic-Tac-Toe, etc)
- Robot learning to dock on battery charger
- Learning to choose actions to optimize factory output (manufacturing applications where rewards only known after online manufacturing decisions)

The above-mentioned problems share the following characteristics:

- **Delayed reward:** The agent doesn't observe an immediate reward. Instead, the reward and feedback about the action are calculable from the later states.

- **Opportunity for active exploration:** When the agent can't find information about the possible actions and the corresponding rewards, the agent has the choice

of doing actions while observing the results. The agent will explore the environment by visiting new states and finding the corresponding rewards. Although this may lead the agent to discover the optimal policy, the agent can also take undesirable actions. The penalties incurred of exploration suggest the agent to balance exploration and exploitation to exploit information of actions/rewards that it already has.

- **State is only partially observable:** The agent might not be able obtain a view of all information related to the state due to several reasons. One of which is the big state space, which could be to the scale of millions.

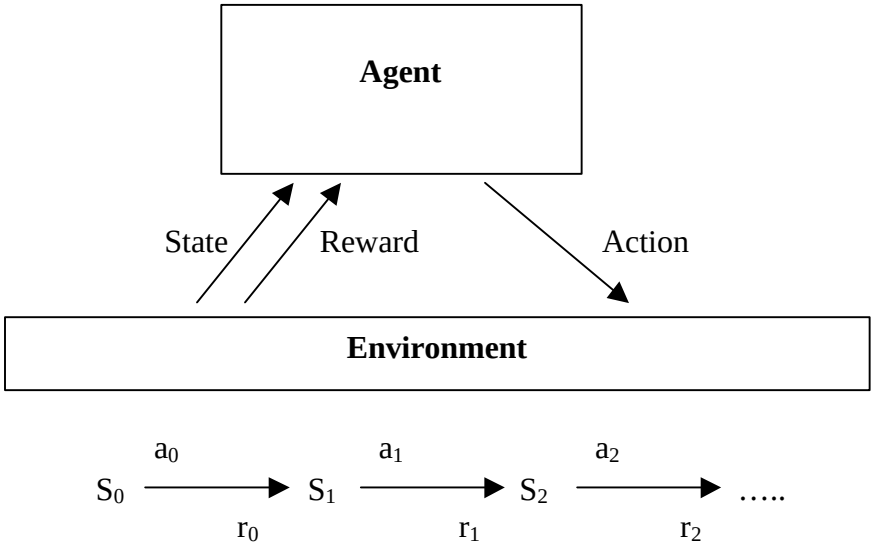
- **May need to learn multiple tasks:** For example, in robot learning two arms need to simultaneously move in a certain manner to get to the required state. Another example is that the agent needs to learn to dock on its battery charger and to learn to navigate through corridors simultaneously.

Using RI has gained success in the problem domains discussed above. For example

+ World class TD-Backgammon program [Tesauro'95]
+ Checkers program [Samuel'59]
+ Job-Shop scheduling [Zhang/Dietterich'95]
+ Real-time scheduling of passenger elevators [Crites/Barto]

Reinforcement Learning Problem

Here the learner is represented by an agent learns by interaction with the environment. The agent performs actions to the environment that changes its states. In order to know about the effectiveness of the actions taken, the agent measures the rewards achieved. The goal of the agent is to select the course of actions (policy) that obtains the maximum overall reward.



In other words, the agent's goal can be formulated to maximize:

$r_0 + g\, r_1 + g^2 r_2 + ..., where..0 \le g \le 1$

There are three reward models:

- Discounted cumulative, infinite horizon: $\sum_0^\infty g^i r_{t+i}$
- Cumulative, finite horizon: $\sum_0^h r_{t+i}$
- Average reward model: $\lim_{h\to\infty} \frac{1}{h} \sum_0^h r_{t+i}$

Discounted Cumulative, infinite horizon:

As noted earlier, rewards may be delayed; therefore the agent is faced with tradeoffs of taking actions on the short-term versus on the long term. To capture this notion, the concept of the discount factor ã is used. This concept is a standard practice in business accounting for decisions on assets. In an oil well example where the value of the well this year is 1 Million, next year for example is 1 million, etc… then which year is the best for selling the well? Here if the decision is delayed for infinity, it may imply the maximum overall reward. However, since there is a probability of breakage, the decision to sell the well earlier is based on the accumulation of the discount factor and the probability of failure. I.e. later reward gets discounted

The Average Reward Model

Consider the sequence of immediate rewards and the agent can calculate the average reward as

1	2	3	1
1	1.5	2	7/4

Instead of using the discount factor (infinite discounted model), the agent may use the average reward function to calculate the rewards in response to a course of actions.

$$\lim_{h\to\infty} \frac{1}{h} \sum_0^h r_{t+i}$$

Markov Decision Processes

Assume:

1. Finite set of states S and
2. Finite set of actions A

Since the agent is assumed to observe state and choose action, then it receives an immediate reward $r_t = r(S_t)$ and the state will change to S_{t+1}. Markov Assumption states that r_t, S_{t+1} depend only on current state(s) action. This assumption can simplify the memory requirements of the agent (example). Based on this assumption, state transitions can be formed as a matrix mapping from current state St to future states S_{t+1} with values of the utilities[3].

	S1	S2	S3
S1	0.5	0.7	..
S2	..	..	..
S3	..	..	..

Agent's Learning Task

Definitions

Policy: $p : S \to A$ (A sequence of actions based on current state that performs state transitions)

Optimal Policy: ð* is the policy that achieves maximal reward

Utility: (value) of policy ð at $S_0 \in S$

$$\equiv \sum_{t=0}^{\infty} g^{\,t} r(s_t)$$

where $S_{t+1} = d(S_t, p(S_t))$

Expected Utility of policy ð at S

$$U_p(s) = r(s) + g \sum_{s'} P(s' \mid s, p(s)).U_p(s')$$

[3] Any deterministic planning problem can be modeled as MDP (Markov decision problem)

Goal: Computing optimal policy $p^* = \arg\max_p E_s[U_p(s)]$

Challenges

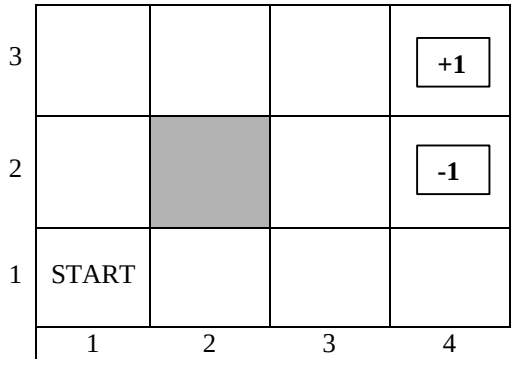
The challenges faced with the RI agent can be summarized at:

- **Challenge 1:** Finding optimal policy given model
  - Value Iteration
  - Policy Iteration
    - **Challenge 1A:** Evaluating fixed policy
    - Adaptive Dynamic Programming
    - TD-learning; TD(ë)-learning
- **Challenge 2:** Scaling to large state spaces
  - Generalization
- **Challenge 3:** Finding optimal policy NOT given model
  - Estimations + Challenge 1
  - Q-learning

Computing Optimal Policy Given Model

Simple Situation

Consider this example: an agent starts at location [1,1] moves {North, South, East, West} on the grid until it terminates on reaching [4,2] or [4,3]. [4,3] is the goal while [4,2] is a loss spot.



The agent has a transition model: $M_{ij}^a = P(S_{t+1} = j \mid S_t = i, A=a)$

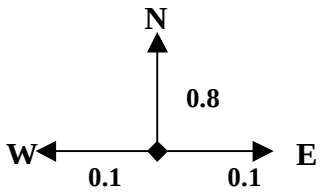
This is the probability of reaching state j is agent performs action a from state i.. using r(s) = immediate reward of state s. The agent (acting rationally) will maximize expected utility:

$$U(s_i) = r(s_i) + \max_a \sum_a M_{ij}{}^a . U(s_j)$$

and with the discount factor will be:

$$U(s_i) = r(s_i) + g \max_a \sum_a M_{ij}{}^a . U(s_j)$$

If actions have deterministic effects (i.e. $M_{ij}{}^a$ is either 0 or 1), this will be a standard planning problem. However, agent actions are not reliable. The agent can move {North, South, East, West}, however with non-deterministic effects. For example, the agent taking actions moving north implies the probability of the actual transition north is 0.8 as given below:



Moreover, for this model, the agent uses an **immediate reward** that leads to a final state. For the above example, it will be:

$$\{r(s_{t+1}) \begin{cases} -\frac{1}{25} & if S_{t+1} \neq [4,2] or [4,3] \\ 1-\frac{1}{25} & if S_{t+1} = [4,3] \\ -1-\frac{1}{25} & if S_{t+1} = [4,2] \end{cases}$$

The **Utility function** can be defined as the sequence of action + states (cumulative immediate reward)

$$U([s_0, a_1, s_1, ..., a_n, s_n]) = \sum_t r(s_{t+1})$$

$$= \begin{cases} 1-\frac{n}{25} if s_n = [4,3] \\ -1-\frac{n}{25} if s_n = [4,2] \end{cases}$$

Therefore, the agent will use the EPISODE that gives the maximum utility. In this case, the agent needs to do actions (moves) and after each, it will determine the resulting state[4]. Based on states, the agent just needs to know the optimal action. Therefore, the agent searches the optimal policy (ð*, ð:State→Action)

[4] In this section we are working with observable (accessible) environments. I.e. the agent can determine the state.

Theorem (Bellman and Dreyfus)

The optimal policy ð*(s) is the one that maximizes expected utility U*(.)

$$p^*(s_i) := \arg\max_a \sum_j Mi_{ij}{}^a . U^*(s_j)$$

$$U^*(s_i) := r(s_i) + \max_a \sum_j Mi_{ij}{}^a . U^*(s_j)$$

Therefore, the agent needs U* too obtain the optimal policy. The **Value iteration algorithm** can be used to estimate U.

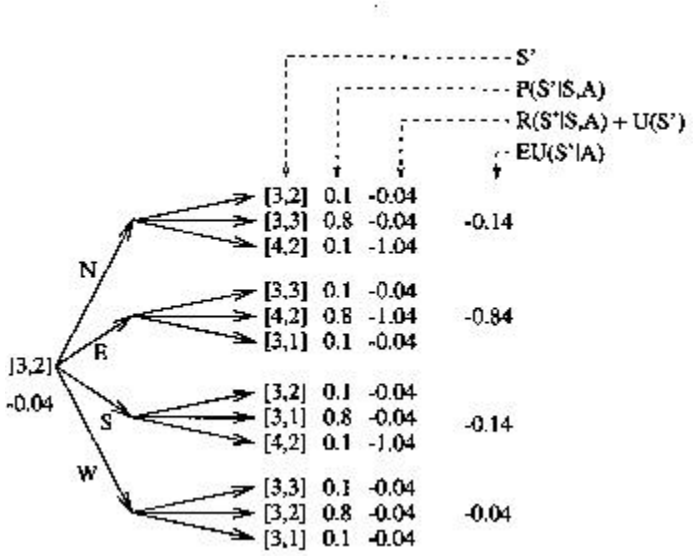
Function ValueIteration(…)
U(S) := r(S) for all S
While U is changing **do**
  for each state s_i **do**

$$U(s_i) := r(s_i) + \max_a \sum_j Mi_{ij}{}^a . U(s_j)$$

  end
end
return (U(.))

It is guaranteed that U converges to stable values. Each update of U is called Bellman backup. An example of Bellman backup for the agent at state [3,2] is given below

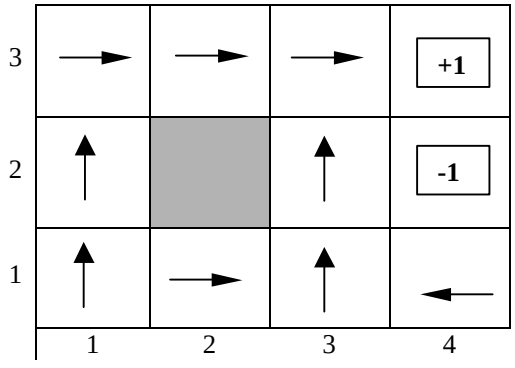
Bellman Backup - Example



Performing value iteration on the grid example yields the following utility values (optimal value function):

3	0.812	0.868	0.912	**+1**
2	0.762		0.660	**-1**
1	0.705	0.655	0.611	0.388
	1	2	3	4

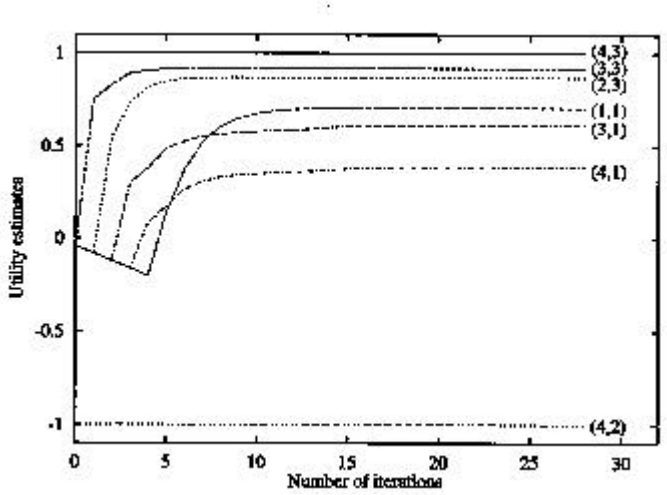
And therefore, the optimal policy will be:



Behavior of Value Iteration

Convergence: The above mentioned algorithm is guaranteed convergence to optimal.

Notice how fast utility estimate converges to the actual. For example, [4,3] immediately is at the actual with value 1 since it is the goal state. The next fastest convergence is for the adjacent state ([3,3]), etc… see figure below.



Exploration versus exploitation: The agent will need to do both. For example, we moved to a new city, Vancouver, in exploration for higher payoffs. On the other hand exploiting an already known action from a state might be considered (known payoff).

Model-Free Learning: Q Function

Unlike the previous model where the agent has a model of the utility function, the agent might not presume such model. Similar to the value function, the agent calculates rewards using the Q function by iteratively visiting the different system states; given below:

$$Q(s,a) \equiv r(s) + gU^*(d(s,a))$$

Based on Q, the agent can choose the optimal action ð* even without knowing ä:

$$p^* = \arg\max_a \{[r(s) + gU^*(d(s,a))]\}$$

$$= \arg\max_{a} \{Q(s,a)\}$$

Thus The Q function is an on-line approximation to the value iteration. We need the agent to learn Q(s,a) to find the optimal policy that is the target of the Q-learning algorithm.

Q-Learning Algorithm (Deterministic World)

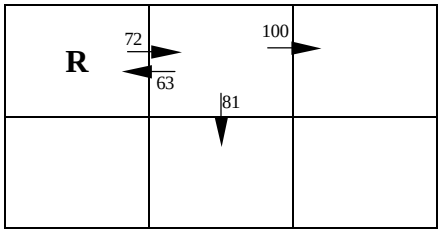
- For each s, a
  Initialize table entry

- Observe current state s

- Do forever:

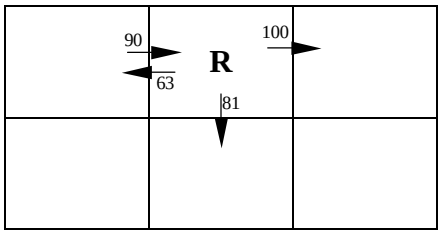
  Select an action a and execute it
  Receive immediate reward r = r(s)
  Observe new state s'=ä(s,a)
  Update table entry for $\hat{Q}(s,a)$
  $\hat{Q}(s,a) \leftarrow r + g \max_{a'} \{\hat{Q}(s',a')\}$
  s ← s'

Notice that $\hat{Q}$ is an approximation to Q. An example of updating $\hat{Q}$ assuming the initial state given below (ã=0.9):



Initial state s_1

After one step, the values of $\hat{Q}$ will be:



Next state s_2

$$\hat{Q}(s1, ar) \leftarrow r(s_1) + g \max_{a'}\{\hat{Q}(s', a'), a'\}$$
$$= 0 + 0.9 \max\{63, 81, 100\}$$
$$= 90$$

Notes about Q-Learning:

1. If rewards > 0 then
   a. $(\forall s, a, n) \quad \hat{Q}_{n+1}(s, a) \geq \hat{Q}_n(s, a)$
   b. $(\forall s, a, n) \quad 0 \leq \hat{Q}_n(s, a) \leq Q(s, a)$
2. $\hat{Q}$ Converges to Q

We can always work with rewards > 0 since it is possible from utility theory to rescale the utility function. For example, rescaling u to be u'=au + b where for a, b constants, a>0, is possible. If u(1) is to chocolate and u(2) is to vanilla, then vanilla will be preferred. Rescaling the utility function will still hold the property of favoring vanilla

It can be proved that $\hat{Q}$ converges to Q for the deterministic case when the agent visit (s,a) infinitely often.

Convergence

Let $\hat{Q}_n$ be the table after n updates and

$\ddot{A}_n$ be the maximum error in $\hat{Q}_n$

$$= \max_{s,a}\{|\hat{Q}_n(s, a) - Q(s, a)|\}$$

Claim: After each interval,

$$\Delta_{n+fi} \leq g\Delta_n$$

i.e. the largest error in $\hat{Q}$ is reduced by ã.

Proof:

$$
\begin{aligned}
&|\hat{Q}_{n+1}(s,a) - Q(s,a)| \\
&\quad = |\,(r(s) + g \max_{a'} \hat{Q}_n(s',a')) - (r(s) + g \max_{a'} Q(s',a'))\,| \\
&\quad = g\,|\max_{a'} \hat{Q}_n(s',a') - \max_{a'} Q(s',a')| \\
&\quad \le g \max_{a'} |\hat{Q}_n(s',a') - Q(s',a')| \\
&\quad \le g \max_{s'',a'} |\hat{Q}_n(s'',a') - Q(s'',a')| \\
&\quad \le g\Delta_n
\end{aligned}
$$

Uses: $|\max_a f_1(a)| - \max_a f_2(a)| \quad \le \quad \max_a |f_1(a) - f_2(a)|$

i.e., $\hat{Q}$ Converges to Q

Final Notes

Reinforcement Learning has the following dimensions:

Accessibility: In accessible environments, the learner has complete state information from its perceptions. Sometimes, the agent may have access to only a subset of features of the state, but not all. In such partially observable situations, the agent might or might not be interested in full estimation of the state (e.g. EM algorithm and POMDP). Sometimes, it is to the health of the agent memory to save the agent from considering irrelevant information.
When Rewards: Are rewards delayed to only TERMINAL states, or it available for intermediate states
Prior Knowledge: Varies depending on the agent's prior knowledge about the model ä(s,a) and r(s,a). Otherwise, the agent has to learn this as well as utility information?
Determinism: In deterministic environments, the state is determined by current state and the action. I.E. $P(s_{t+1}|s_t,a_t) \in \{0,1\}$
Fixed or Changing Policy: Given fixed policy, the agent will "passively" watches world, trying to learn utility of different states. The "Active" agent changes policy.
Discount: The discount factor captures the relative importance of current reward versus the future reward (if ã=1 future rewards are more important then for ã <1)

- There is a good web site of Dr. Richard Sutton at the University of Alberta (http://www.cs.ualberta.ca/~sutton/) this site has interesting links to research in reinforcement learning as well as a link to the book by Sutton and Barto in Reinforcement Learning (http://www.cs.ualberta.ca/~sutton/book/the-book.html)

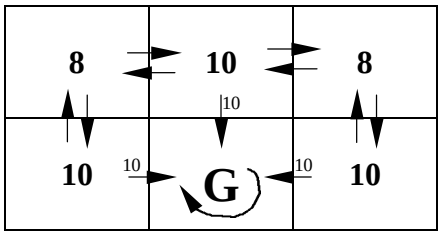
Assignment 7 Solution

13.2.

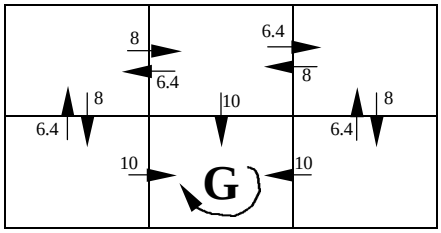
Reward = 10 for all labeled transitions and 0 for all unlabeled transitions

13.2 (a)

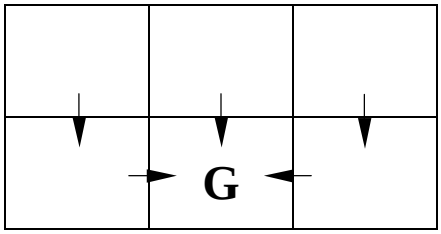
ã = 0.8



V* (s)

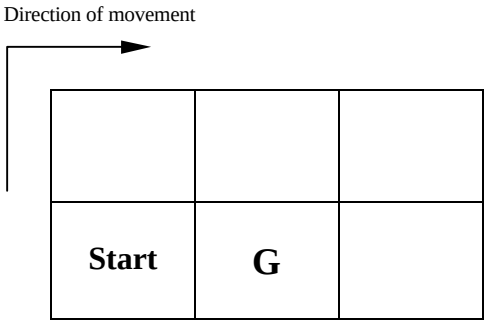


Q(s,a)

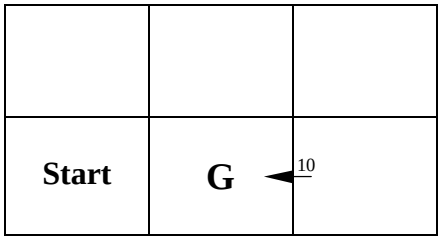


One optimal policy

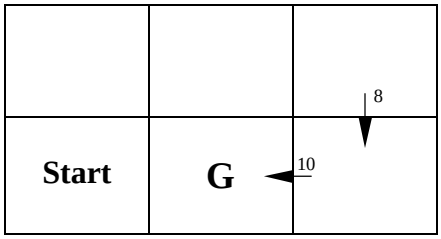
13.2 (c)



And assuming the initial $\hat{Q}$ values is initialized to zero

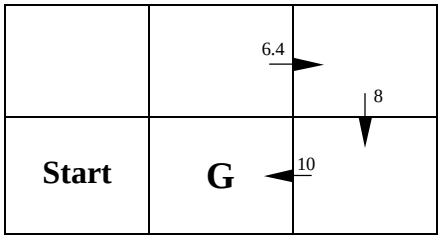


After the first episode, only the $\hat{Q}$ values corresponding the transition indicated will have an update.



After the second episode

$$\hat{Q} \leftarrow r + g \max_{a'} \hat{Q}(s', a')$$



After the third episode