Imperial College London

October 18, 2018

Department of Computing

MLNC – Machine Learning & Neural Computation – Dr Aldo Faisal

Lab Assignment 1

Lab assignment: Understanding of MDPs

Consider a stair climbing MDP made of 7 states $\mathcal{S} = \{s_1, \ldots, s_7\}$. The top of the states and the bottom of the states are absorbing states s_1, s_7. Reaching the bottom yields you $+100$ reward, reaching the bottom yields you -100 reward. Each step up gives you -10 reward (effort), each step down give you 10 reward. The two possible actions $\mathcal{A} = \{Down, Up\}$ are deterministic. We do not specify a specific γ.

1. Draw the MDP as a graph (by hand)
2. Using below Matlab code as template for specifying an MDP, specify above Stair Climbing MDP (You will have to fix the code).
3. Implement code that takes a deterministic policy (i.e. $\pi(s) = a$) and computes the value function for this policy for $\gamma = \frac{1}{2}$
4. Compute for the "Always Down" policy and the "Always Up" policy the two value functions (code it so that it works for an arbitrary $\gamma \in [0, 1]$.).
5. Plot how varying γ will make an "always up" policy be more rewarding than an "always down" if you start in s_4.
6. Think about a way to find a better deterministic policy (e.g. by choosing actions so as to be greedy on the value function).

```
%%% Basic code to specify an MDP
%%% Learning in Autonomous Systems coursework
%%% Aldo Faisal (2015), Imperial College London
function [S, A, T, R, StateNames, ActionNames, Absorbing] = StairClimbingMDP()
% States are:  { s1 <-- s2 <=> s3 <=> s4 <=> s5 <=> s6 --> s7 ];
S = 7;
StateNames =  ['s1'; 's2'; 's3'; 's4'; 's5'; 's6'; 's7'];

% Actions are: {L,R} --> {1, 2 }
A = 2;
ActionNames =  ['L'; 'R'];

% Matrix indicating absorbing states
Absorbing = [
%P  1   2   3   4   5  6  7  G   <-- STATES
       1   0    0   0  0  0  1
];

% load transition
T = transition_matrix()

```

```
% load reward matrix
R = reward_matrix(S,A)

%----------------------------------------------------------------------

% the transition subfunction
function prob = transition_function(priorState, action, postState)
% reward function (defined locally)
T = transition_matrix()
prob = T(postState,priorState,action)

% get the transition matrix
function T = transition_matrix()
TL = [
% MODIFY HERE
% 1 ...7     <-- FROM STATE
1   1   0   0   0   0   0 ; % 1 TO STATE
0   0   1   0   0   0   0 ; % .
0   0   0   1   0   0   0 ; % .
0   0   0   0   1   0   0 ; % .
0   0   0   0   0   1   0 ; % .
0   0   0   0   0   0   0 ; % .
0   0   0   0   0   0   1 ; % 7
];
TR = [
% MODIFY HERE
% 1 ...7     <-- FROM STATE
1   0   0   0   0   0   0 ; % 1 TO STATE
0   0   0   0   0   0   0 ; % .
0   1   0   0   0   0   0 ; %  .
0   0   1   0   0   0   0 ; % .
0   0   0   1   0   0   0 ; % .
0   0   0   0   1   0   0 ; % .
0   0   0   0   0   1   1 ; % 7
];
T = cat(3, TL, TR); %transition probabilities for each action


%----------------------------------------------------------------------

% the locally defined reward function
function rew = reward_function(priorState, action, postState)
% reward function (defined locally)
% MODIFY HERE
if ((priorState == 2) && (action == 1) && (postState == 1))
    rew = -1;
elseif ((priorState == 6) && (action == 2) && (postState == 7))
    rew = 1;
elseif (action == 1)
    rew = 0;
else
    rew = 0;
end

% get the reward matrix
function R = reward_matrix(S, A)
% i.e. 11x11 matrix of rewards for being in state s,
%performing action a and ending in state s'
R = zeros(S, S, A);
for i = 1:S
   for j = 1:A
      for k = 1:S
         R(k, i, j) = reward_function(i, j, k);
      end
```

```
    end
end
```