Problem Set III

Karan Chawla
karangc2@illinois.edu

Course Instructor: **Prof. G. Chowdhary**

April 28, 2017

[1]

[1] This assignment was done with Wyatt McAllistar

1 Question 1

Consider a 5 by 5 grid world. We would like to find a path from one end of the grid to the other end, that is from location (1,1) to location (5,5). A reward of 1 is obtained when the agent reaches (5,5), a reward of -0.01 is obtained for every transition. There are 5 actions: stay, go left, go up, go right, go down. Each action results in a stochastic transition, with 90% probability of the intended transition, and 10% probability of a random transition.
What would be the state transition matrix, what would be the reward function? Assuming the state transition and reward function is known, implement the following algorithms to find this path:
1. Value iteration (Algorithm 2 from Geramifard et al).
2. TBVI (Algorithm 3 from Geramifard et al.).
If the state transition and the reward matrices are not known, implement the following reinforcement learning algorithm:
1. SARSA (Algorithm 6 from Geramifard et al.).
For all algorithms, implement the following value function approximations:
1. Tabular.
2. Radial Basis Function network, limit the maximum number of basis to 20.
3. (Bonus) Gaussian Process (5 points extra for each algorithm you implement GP for).
An implementation of Q learning with Tabular, Radial Basis, and GP function approximation has been provided. The main file is *gridworldQLGPmain.m*. It has been commented for your review. You need not use this implementation if you do not wish to.
For implementing the SARSA algorithm, you could change the Q learning update law with the SARSA one in the given code.
For implementing the MDP algorithms, you have you change the code to use the transition and reward models in the computation of the policy. Else, you can write code from scratch, which in this case could be easier since an episodic treatment that is done in the RL code is not needed for solving MDPs when the reward and transition models are known.
Note that the transitions in this grid world are stochastic, as opposed to the deterministic transitions in the grid world in Problem Set 2.
Present your answer as plots and a discussion. For the dynamic programming problems, you will evaluate the performance of your algorithm over a series of 100 Monte-Carlo runs. The stochasticity comes from the random transitions.
The RL algorithms are set to run in 5 executions of 200 episodes each with 100 evaluations (samples).
The files given to you plot the results with their mean and standard deviations. This is the kind of plot you should be presenting.

Solution

For value iteration, the policy and reward converged for both tabular and radial basis function implementations, and are shown in Table 1, and Table 2 below.

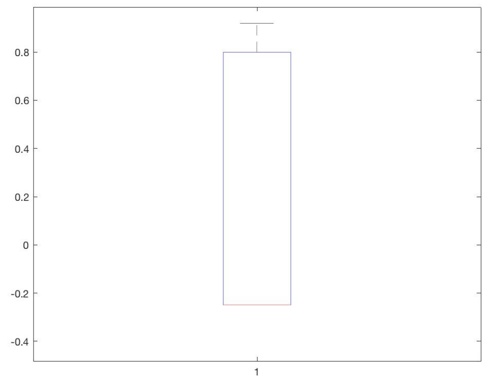
Table 1: Value Iteration Tabular

Value					Policy				
0.7372	0.8129	0.8979	0.9912	1.0000	2	2	2	2	1
0.6729	0.7409	0.8169	0.9001	0.9912	2	2	2	2	3
0.6108	0.6717	0.7409	0.8169	0.8979	3	2	2	3	3
0.5539	0.6085	0.6717	0.7409	0.8129	3	2	3	3	3
0.5029	0.5539	0.6108	0.6729	0.7372	2	2	2	3	3

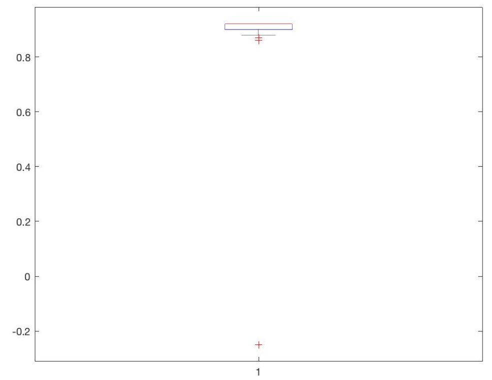
Table 2: Value Iteration RBF

Value					Policy				
0.5675	0.5823	0.6363	0.8334	1.0000	2	2	2	2	1
0.5772	0.6014	0.6379	0.7443	0.8333	2	2	2	3	3
0.5729	0.6002	0.6255	0.6373	0.6361	2	2	3	3	3
0.5491	0.5719	0.5996	0.6011	0.5836	3	3	3	3	3
0.5313	0.5490	0.5727	0.5785	0.5707	3	2	3	3	3

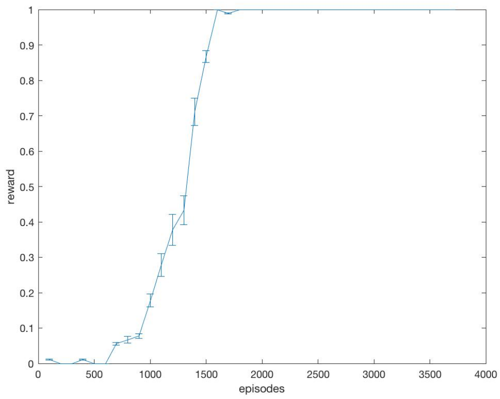
The figures on the following page show the six learning algorithms. It can see, there is a large difference between on policy and off policy learning for this trivial example. Because the state space is really small, and because there are a range of optimal paths, the on-policy algorithms such as SARSA and TBVI do not converge well. In the case of the RBF implementation, TBVI will converge, but SARSA never does not. Q learning always converges as it is an off-policy method.



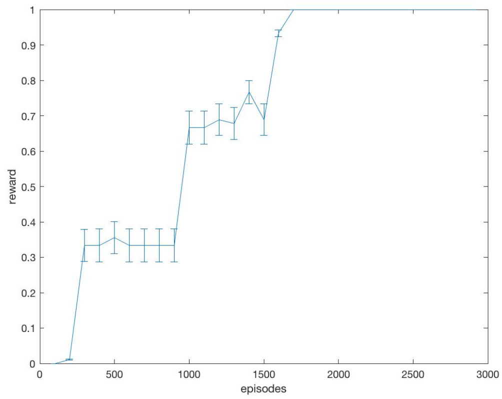
(a) TBVI Tabular



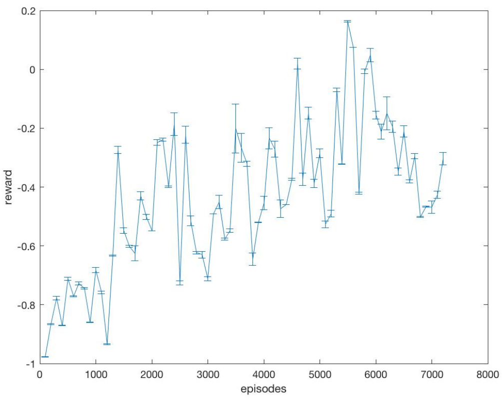
(b) TBVI RBF



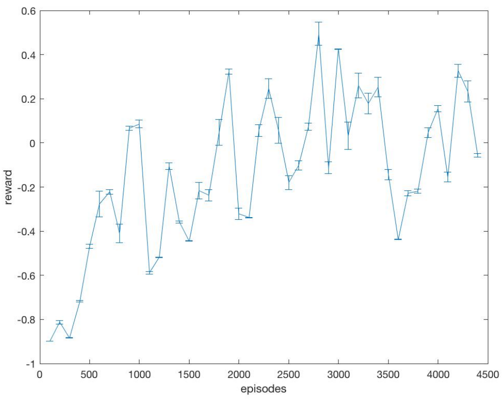
(c) Q Tabular



(d) Q RBF



(e) SARSA Tabular



(f) SARSA RBF

Figure 1.1: Learning Algorithms

The figures above shown the six learning algorithms.

Matlab Code for Value Iteration

```
clear all; close all; clc;

%Initialize Variables
basis         = 1;                       % basis
N_grid        = 5;                       % grid size
N_state       = N_grid*N_grid;           % state size
N_act         = 5;                       % action size
noise         = 0.1;                     % transition uncertainty
gamma         = 0.9;                     % discount factor
eta           = 0.05;                    % tolerance
alpha0        = 0.5;                     % initial Learning Rate
alpha_exp     = 0.5;                     % learning rate

% create value function tablse and policy
v             = zeros(N_state, 1);       % value function
old_v         = zeros(N_state, 1);       % previous value function
policy        = zeros(N_state,1);        % policy

%Create state transition matrix
P             = zeros(N_state,N_state,N_act); %(A, Sold, Snew)

params.N_grid = N_grid;
params.N_state = N_state;
params.rbf_c  = [1 1; 1 5; 2 4; 3 3; 4 2; 5 1; 5 5];
params.s      = size(params.rbf_c,1)+1;  % RBF standard deviation
params.mu     = ones(params.s,1)*1;      % RBF average
params.bw     = 1;                       % RBF bias parmeter

for s=1:N_state
    %states: stay, right, up, left, down
    new_s(1)  = s;                       % stay
    new_s(2)  = s+1*(mod(s,N_grid)~=0);  % right
    new_s(3)  = s+N_grid*(s+N_grid<=N_state); % up
    new_s(4)  = s-1*(mod(s,N_grid)~=1);  % left
    new_s(5)  = s-N_grid*(s-N_grid>=1);  % down

    % for all actions
    for a=1:N_act
        % compute transition probability for desired
        P(s,new_s(a),a) = (1-noise);
        % for all actions
        for a_rand=1:N_act
            % compute uncertain transition probability
            P(s,new_s(a_rand),a) = P(s,new_s(a_rand),a) + noise/(N_act-1);
        end
    end
end

% choose basis


% tabular
if basis==0
```

```
    % initialize delta
    delta=eta;
    % while delta is not within the tolerance
    while delta>=eta
        % initialize delta
        delta=0;
        % create a temporary set of value and policy vectors
        va = zeros(N_state, N_act);
        v_ = zeros(N_state,1);
        %for all initial states
        for s=1:N_state
            % for all actions
            for a=1:N_act
                % for all final states
                for s_prime=1:N_state
                    % compute the new value function
                    va(s,a)=va(s,a)+P(s,s_prime,a)*(reward(s_prime)+gamma*v(s_prime,:));
                end
            end
            % store the new maximum value and corresponding action
            [v_(s,:), action] = max(va(s,:));
            policy(s,:) = action;
            old_v(s,:) = v(s,:);
            % update the value function and recompute delta
            v(s,:) = v_(s,:);
            delta = max(delta,abs(v(s,:)-old_v(s,:)));
        end
    end

elseif basis==1 % RBF
    % initialize theta, phi, and delta
    theta = zeros(params.s,1);
    phi = VI_RBF(params);
    delta=eta;
    i=1;
    while delta>=eta
        delta=0;
        for s=1:N_state
            v=phi(s,:)*theta;
            %find action that maximizes value
            va=zeros(1,N_act);
            for a=1:N_act
                va(a)=P(s,:,a)*(reward(1:N_state)'+gamma*phi*theta);
            end
            v_=max(va);
            % update RBF
            alpha = alpha0/i^alpha_exp;
            theta_old = theta;
            theta = theta + alpha*(v_-v)*phi(s,:)';
            theta=theta/norm(theta);
            % update delta
            delta = max(delta,max(theta-theta_old));
        end
        i=i+1;
```

```
    end

    % update value function using RBF
    v = phi*theta;
    Va=zeros(N_state,N_act);
    %compute maximum value
    for a=1:N_act
        Va(:,a)=P(:,:,a)*v;
    end
    %update policy;
    [~, policy]=max(Va,[],2);
end

% display output
V_new=flipud(reshape(v,[N_grid N_grid])')/max(v);
policy=flipud(reshape(policy,[N_grid N_grid])');
display(V_new)
display(policy)
```

Matlab Code for TBVI

```
%% Gridworld Simulation
clear all;
close all;
clc;
%% Gridworld Parameters
basis             = 1;              % 0 for tabular, 1 for RBF
N_grid            = 5;              % size of grid
N_state           = N_grid*N_grid;  % size of state
N_act             = 5;              % number of actions
N_eps             = 20;             % max length of trajectory
N_time            = 1000;           % length of time
N_converge        = 6;              % number of convergences
N_exec            = 5;              % number of executions
N_dim             = 2;              % state dimension

eta               = 0.01;           % convergence tol
s_init            = [1;1];          % initial state
a_init            = 2;              % initial action
s_goal            = [N_grid;N_grid]; % goal state

rew_goal          = 1;              % goal reward
rew_trans         = -0.01;          % transition reward
noise             = 0.1;            % transition uncertainty

params.N_grid     = N_grid;         % size of grid
params.s_goal     = s_goal;         % goal state
params.rew_goal   = rew_goal;       % goal reward
params.rew_trans  = rew_trans;      % transition reward
params.N_dim      = N_dim;          % state dimension
params.N_act      = N_act;          % number of actions
params.noise      = noise;          % transition uncertainty
```

```
%% Learning Parameters
gamma            = 0.9;           % discount factor
params.gamma     = gamma;         % discount factor

alpha_init       = 0.5;           % initial learning rate
alpha_dec        = 0.5;           % decay rate
mu               = 1;
eps_init         = 0.8;           % initial exploration rate
eps_dec          = 0.1;           % exploration rate decay

max_points       = 25;            % max centres allowed for RBF
tol              = 1e-4;          % tolerance

if basis==0
    params.N_s = N_state;                % Number state-features
    params.N_sa = params.N_s*params.N_act; % Number state-action-features
    params.state_action_slicing_on=1;
    gpr = onlineGP_RL(0,0,0,0,params);
    params.basis=0;
elseif basis==1
    params.c = [5 5; 1 5; 5 1; 1 1]';
    params.N_s = size(params.c,2)+1;     % Number state-features
    rbf_mu = ones(params.N_s,1)*mu;      % RBF mu
    params.mu=ones(params.N_s,1)*1;      % RBF mu
    params.bw=1;                         % RBF bias
    params.N_sa = params.N_s*params.N_act; % Number state-action-features
    params.state_action_slicing_on=1;
    gpr = onlineGP_RL(0,0,0,0,params);
    params.basis=1;
end

%% Algorithm Execution
%Create state transition matrix
P=zeros(N_act,N_state,N_state);
for s=1:N_state
    %states: stay, right, up, left, down
    new_s(1) = s;                          % stay
    new_s(2) = s+1*(mod(s,N_grid)~=0);     % right
    new_s(3) = s+N_grid*(s+N_grid<=N_state); % up
    new_s(4) = s-1*(mod(s,N_grid)~=1);     % left
    new_s(5) = s-N_grid*(s-N_grid>=1);     % down

    % for all actions
    for a=1:N_act
        % compute probability of desired transition
        P(s,new_s(a),a) = (1-noise);
        % for all actions
        for a_rand=1:N_act
            % compute probability of uncertain transition
            P(s,new_s(a_rand),a) = P(s,new_s(a_rand),a) + noise/N_act;
        end
    end
end
```

```

% initialize
theta = zeros(params.N_sa,1);
ctr = 0;
eval_ctr = 0;
n_conv=0;
i=0;

% while the time is less than the required time
% while the number of convergences is less that that required
while i<=N_time && n_conv<N_converge
    % initialize
    s_old = s_init;
    break_cmd = 0;
    delta = 0;
    k=1;
    % while less than the number of episodes
    % while not commanded to break
    while k<=N_eps && ~break_cmd
        ctr = ctr+1;

        % set the exploration proability
        p_eps = eps_init/(ctr)^eps_dec;

        % check whether to explore
        r = sample_discrete([p_eps 1-p_eps]);

        % explpre
        if r==1
            p = 1/N_act.*ones(1,N_act);
            action = sample_discrete(p);
        % exploit
        else
            [Q_opt,action] = Q_greedy_act(theta,s_old,params,gpr);
        end

        % compute next state and reward
        s_old_lin = sub2ind([N_grid N_grid],s_old(1),s_old(2));
        s_new=zeros(2,N_exec);
        rew=zeros(1,N_exec);
        % for all executions
        for j=1:N_exec
            % calculate next state
            s_new(1,j) = sample_discrete(P(s_old_lin,:,action));
            [s_new(1,j),s_new(2,j)]=ind2sub([N_grid N_grid],s_new(1,j));
            % calculate reward
            [rew(j),break_cmd] = reward2(s_new(:,j),params);
        end
        % recompute learning rate
        alpha = alpha_init/ctr^alpha_dec;
        % recompute feature vector
        phi_old = Q_feature(s_old,action,params);
        % calculate the value
        val_old = Q_value(theta,s_old,action,params);
```

```
        v_new = 0;
        % for all executions
        for j=1:N_exec
            % compute optimal action
            [Q_opt,a_op] = Q_greedy_act(theta,s_new(:,j),params,gpr);
            % compute new value
            v_new=v_new+rew(j)+gamma*Q_value(theta,s_new(:,j),a_op,params);
        end
        v_new = v_new/N_exec;
        % compute error
        err = (v_new - val_old);

        % update basis vector
        theta_old=theta;
        theta = theta + alpha*(err.*phi_old);
        delta = max(delta,abs(max(theta-theta_old)));

        % reset state
        s_old = s_new(:,1);
        k=k+1;
    end
    % check for break condition
    if (delta<eta && break_cmd)
        n_conv = n_conv+1;
    end
    i=i+1;
end

% update policy
policy=zeros(N_grid,N_grid);
for i=1:N_grid
    for j=1:N_grid
        [~,policy(i,j)] = Q_greedy_act(theta,[i;j],params,gpr);
    end
    policy(N_grid,N_grid)=1;
end
policy = flipud(policy');

%% Monte Carlo runs
rew_eval = zeros(1,100);
for eval_ctr = 1:100

    s_old = s_init;
    for ctr=1:N_state

        [~,action] = Q_greedy_act(theta,s_old,params,gpr);
        s_next = gridworld_trans(s_old,action,params);
        [rew,break_cmd] = reward2(s_next,params);
        rew_eval(eval_ctr) = rew_eval(eval_ctr) + rew;

        if break_cmd
            break;
        end

```

```
        s_old = s_next;
    end
end
boxplot(rew_eval);
rew_mean=mean(rew_eval);
rew_var=var(rew_eval);
display(rew_mean);
display(rew_var);
```

Matlab Code for SARSA

```
%% Gridworld Simulation
clear all;
close all;
clc;
%% Gridworld Parameters
basis                = 1;                  % 0 for tabular, 1 for RBF
N_grid               = 5;                  % grid size
N_state              = N_grid*N_grid;      % state size
N_act                = 5;                  % number of actions

N_dim                = 2;                  % state dimension
s_init               = [1;1];              % initial state
a_init               = 2;                  % initial action
s_goal               = [N_grid;N_grid];    % goal state

N_obstacle           = 0;                  % number of obstacles
obs_list             = [];                 % coordinates of obstacles

rew_goal             = 1;                  % reward for goal
rew_trans            = -0.01;              % reward for transition
noise                = 0.1;                % transition uncertainty

params.N_grid        = N_grid;             % grid size
params.s_goal        = s_goal;             % goal state
params.rew_goal      = rew_goal;           % reward for goal
params.rew_trans     = rew_trans;          % reward for transition
params.N_dim         = N_dim;              % state dimension
params.N_act         = N_act;              % number of actions
params.noise         = noise;              % transition uncertainty

%% Learning Parameters
N_length             = 100;                % length of episode
N_eps                = 200;                % number of episodes
N_exec               = 3;                  % number of executions
N_freq               = 100;                % frequency of evaluation
N_eval               = 30;                 % evaluation interations
N_budget             = 25;                 % max points in stack

gamma                = 0.9;                % discount factor
params.gamma         = gamma;              % discount factor

```

```
params.N_budget   = N_budget;            % max points in stack
data_method       = 2;                   % 1 For cylic and 2 for SVD
params.epsilon_data_select=0.2;
stack_index       = 0;                   % initial stack index
points_in_stack   = 0;                   % initial stack length

alpha_init        = 0.5;                 % initial learning rate
alpha_dec         = 0.5;                 % learning rate decay
eps_init          = 0.8;                 % initial exploration rate
eps_dec           = 0.1;                 % exploration rate decay
N_pts             = 25;                  % max centres allowed for RBF
tol               = 1e-4;                % tolerance

if basis==0
    params.N_s = N_state;                % state-features
    params.N_sa = params.N_s*params.N_act; % state-action-features
    params.basis=basis;
    params.state_action_slicing_on=1;
    params.basis=0;
    gpr = onlineGP_RL(0,0,0,0,params);
elseif basis==1
    params.c = [5 5; 1 5; 5 1; 1 1]';
    params.N_s = size(params.c,2)+1;     % number of state-features
    params.mu=ones(params.N_s,1)*1;      % RBF mu
    params.bw=1;                         % RBF bias
    params.N_sa = params.N_s*params.N_act; % state-action features
    params.state_action_slicing_on = 1;
    params.basis = 1;
    gpr = onlineGP_RL(0,0,0,0,params);
end

%% Algorithm Execution
rew_exec = zeros(N_exec,1);
eval_ctr = zeros(N_exec,1);

% for all executions
for i =1:N_exec
    % reset Q function
    theta = zeros(params.N_sa,1);

    step_ctr = 1;
    % for all episodes
    for j = 1:N_eps
        %fprintf('Episode: %d/%d, Execution: %d/%d \n',j,N_eps,i,N_exec);

        % reset to initial state
        s_old = s_init;

        % set exploration probability
        p_eps = eps_init/(step_ctr)^eps_dec;
        % check if exploring
        r = sample_discrete([p_eps 1-p_eps]);
        % explore
        if r==1
```

```
            p = 1/N_act.*ones(1,N_act);
            action = sample_discrete(p);
        % exploit
        else
            [Q_op,action] = Q_greedy_act(theta,s_old,params,gpr);
        end
        % for lenfth of evaluation
        for k = 1: N_length
            % check if it is time to evaluate
            if(mod(step_ctr,N_freq) == 0)
                % evaluate reward
                eval_ctr(i) = eval_ctr(i) + 1;
                rew_eval = zeros(1,N_eval);
                % for number of evals
                for eval_count = 1:N_eval
                    % reset state
                    s_prv = s_init;
                    % for legnth of evaluation
                    for step_count = 1:N_length
                        % calculate optimal action
                        [Q_op,action]=Q_greedy_act(theta,s_prv,params,gpr);
                        s_next = gridworld_trans(s_prv,action,params);
                        % calculate reward
                        [rew, break_cmd] = reward2(s_next,params);
                        rew_eval(eval_count) = rew_eval(eval_count)+rew;
                        % check break condition
                        if break_cmd
                            break;
                        end
                        % update state
                        s_prv = s_next;
                    end
                end
                % update reward
                rew_exec(i,eval_ctr(i)) = mean(rew_eval);
            end
            step_ctr = step_ctr + 1;

            % get nextsState
            s_new = gridworld_trans(s_old,action,params);

            % calculate reward
            [rew,break_cmd] = reward2(s_new,params);

            % set exploration rate
            p_eps = eps_init/(step_ctr)^eps_dec;
            % check if exploring
            r = sample_discrete([p_eps 1-p_eps]);
            % explore
            if r==1
                p = 1/N_act.*ones(1,N_act);
                a_new = sample_discrete(p);
            % exploit
            else
```

```
                % calculation optimal action
                [Q_op,a_new] = Q_greedy_act(theta,s_old,params,gpr);
            end

            % calculate learning rate
            alpha = alpha_init/(step_ctr)^alpha_dec;
            % calculate feature vector
            phi_old = Q_feature(s_old,action,params);
            % calculate value
            v_old = Q_value(theta,s_old,action,params);
            % update value
            v_new = Q_value(theta,s_new,a_new,params);
            % compute error
            err = (rew + gamma*v_new - v_old);
            % update RBF parameter
            theta = theta + alpha*(err.*phi_old);

            % reset state
            s_old = s_new;
            action = a_new;

            % check break comdition
            if break_cmd
                break;
            end
        end
    end
end

%% Post Process
% find minimum number of evaluations
min_eval = min(eval_ctr);
rew_exec = rew_exec(:,1:min_eval);
rew_total = zeros(1,min_eval);
std_total = zeros(1,min_eval);

% update reward
for m =1:min_eval
    rew_total(m) = mean(rew_exec(:,m));
    std = var(rew_exec(:,m));
    std_total(m) = 0.1*std;
end

%% Plots
t = 1:min_eval;
t = t.*N_freq;
errorbar(t,rew_total,std_total);
xlabel('episodes')
ylabel('reward')
```