

HW4\HW4_5264_AmandaMarlow.jl

```
1 using DMUStudent.HW4: gw
2 using LinearAlgebra: I
3 using CommonRLInterface: render, actions, act!, observe, reset!, AbstractEnv, observations,
  terminated, clone
4 using SparseArrays
5 using Statistics: mean
6 using Plots
7 using StaticArrays: SA
8 using StatsBase
9
10 #####
11 # SARSA- $\lambda$  #
12 #####
13
14 function sarsa_lambda_episode!(Q, env;  $\epsilon=0.10$ ,  $\gamma=0.99$ ,  $\alpha=0.05$ ,  $\lambda=0.9$ )
15
16     start = time()
17
18     function policy(s)
19         if rand() <  $\epsilon$ 
20             return rand(actions(env))
21         else
22             return argmax(a->Q[(s, a)], actions(env))
23         end
24     end
25
26     s = observe(env)
27     a = policy(s)
28     r = act!(env, a)
29     sp = observe(env)
30     hist = [s]
31     N = Dict{Tuple{s, a}, Int}((s, a) => 0.0)
32
33     while !terminated(env)
34         ap = policy(sp)
35
36         N[(s, a)] = get(N, (s, a), 0.0) + 1
37
38          $\delta = r + \gamma Q[(sp, ap)] - Q[(s, a)]$ 
39
40         for ((s, a), n) in N
41             Q[(s, a)] +=  $\alpha \delta * n$ 
42             N[(s, a)] *=  $\gamma * \lambda$ 
43         end
44
45         s = sp
46         a = ap
47         r = act!(env, a)
48         sp = observe(env)
49         push!(hist, sp)
50     end
51
52     N[(s, a)] = get(N, (s, a), 0.0) + 1
```

```

53     δ = r - Q[(s, a)]
54
55     for ((s, a), n) in N
56         Q[(s, a)] += α*δ*n
57         N[(s, a)] *= γ*λ
58     end
59
60     return (hist=hist, Q = copy(Q), time=time()-start)
61 end
62
63 function sarsa_lambda!(env; n_episodes=100, kwargs...)
64     Q = Dict{Tuple{String, String}, Float64}()
65     episodes = []
66
67     for i in 1:n_episodes
68         reset!(env)
69         push!(episodes, sarsa_lambda_episode!(Q, env;
70                                             ε=max(0.01, 1-i/n_episodes),
71                                             kwargs...))
72     end
73
74     return episodes
75 end
76
77 function evaluate(env, policy, n_episodes=1000, max_steps=1000, γ=1.0)
78     returns = Float64[]
79     for _ in 1:n_episodes
80         t = 0
81         r = 0.0
82         reset!(env)
83         s = observe(env)
84         while !terminated(env)
85             a = policy(s)
86             r += γ^t*act!(env, a)
87             s = observe(env)
88             t += 1
89         end
90         push!(returns, r)
91     end
92     return returns
93 end
94
95
96 #####
97 # Policy Gradient
98 #####
99
100 function gradLogPi(env, theta, a)
101     A = collect(actions(env))
102     if a == A[1]
103         gradPolicy = [1 - exp(theta[1])/sum(exp.(theta)), -exp(theta[1])/sum(exp.(theta)), -
exp(theta[1])/sum(exp.(theta)), -exp(theta[1])/sum(exp.(theta))]
104     elseif a == A[2]
105         gradPolicy = [-exp(theta[2])/sum(exp.(theta)), 1 - exp(theta[2])/sum(exp.(theta)), -
exp(theta[2])/sum(exp.(theta)), -exp(theta[2])/sum(exp.(theta))]
106     elseif a == A[3]

```

```

107     gradPolicy = [-exp(theta[3])/sum(exp.(theta)), - exp(theta[3])/sum(exp.(theta)), 1 -
exp(theta[3])/sum(exp.(theta)), -exp(theta[3])/sum(exp.(theta))]
108     elseif a == A[4]
109         gradPolicy = [-exp(theta[4])/sum(exp.(theta)), -exp(theta[4])/sum(exp.(theta)), -
exp(theta[4])/sum(exp.(theta)), 1 - exp(theta[4])/sum(exp.(theta))]
110     else
111         throw(error("not a valid action"))
112     end
113
114     return gradPolicy
115 end
116
117 function policyGradEpisode!(env, θ, α)
118
119     start = time()
120
121     A = collect(actions(env))
122     function policy(s,θ)
123         theta = θ[s]
124         tot = sum(exp.(theta))
125         P = zeros(Float64, 4)
126         for i = 1:4
127             P[i] = exp(theta[i])/tot
128         end
129         samp = rand(Float64,1)[1]
130         if samp <= P[1]
131             a = A[1]
132         elseif samp <= P[2]+P[1]
133             a = A[2]
134         elseif samp <= P[3]+P[2]+P[1]
135             a = A[3]
136         else
137             a = A[4]
138         end
139         return a
140     end
141
142     update = []
143     for i = 1:10
144         path = []
145         gradPolicy = []
146         d = 0
147         R = 0
148         while !terminated(env)
149             d +=1
150             s = observe(env)
151             a = policy(s, θ)
152             r = act!(env, a)
153             path = push!(path, (s,a,r))
154             R += r
155             push!(gradPolicy, gradLogPi(env, θ[s], a))
156         end
157
158         for k in 1:d
159             gradU = gradPolicy[k]*R
160             s = path[k][1]
161             push!(update, (s, gradU))

```

```

162         R = R - path[k][3]
163     end
164 end
165 hist = []
166 for q in eachindex(update)
167     s = update[q][1]
168     gradU = update[q][2]
169     θ[s] += α*gradU/10
170     push!(hist,[s])
171 end
172
173 return (hist=hist, θ = copy(θ), time=time()-start, policy = policy)
174 end
175
176 function policyGradient(env, n_episodes, α)
177     θ = Dict{(s) => 0.5*ones(4) for s in observations(env))
178     episodes = []
179
180     for i in 1:n_episodes
181         reset!(env)
182         push!(episodes, policyGradEpisode!(env, θ, α))
183     end
184
185     return episodes
186 end
187
188 function learningCurve_steps(env,episodes, n_episodes)
189     p1 = plot(xlabel="steps in environment", ylabel="avg return")
190     n = convert{Int64,floor(n_episodes/10))
191     stop = n_episodes
192     for (name, eps) in episodes
193         if(name == "SARSA-λ")
194             Q = Dict{(s, a) => 0.0 for s in observations(env), a in actions(env))
195             xs = [0]
196             ys = [mean(evaluate(env, s->argmax(a->Q[(s, a)], actions(env))))]
197             for i in n:n:min(stop, length(eps))
198                 newsteps = sum(length(ep.hist) for ep in eps[i-n+1:i])
199                 push!(xs, last(xs) + newsteps)
200                 Q = eps[i].Q
201                 push!(ys, mean(evaluate(env, s->argmax(a->Q[(s, a)], actions(env))))))
202             end
203         else
204             xs = [0]
205             thetas = Dict{(s) => 0.5*ones(4) for s in observations(env))
206             ys = [mean(evaluate(env, s->eps[1].policy(s,thetas)))]
207             for i in n:n:min(stop, length(eps))
208                 newsteps = sum(length(ep.hist) for ep in eps[i-n+1:i])
209                 push!(xs, last(xs) + newsteps)
210                 thetas = eps[i].θ
211                 push!(ys, mean(evaluate(env, s->eps[i].policy(s, thetas)))))
212             end
213         end
214         plot!(p1, xs, ys, label=name)
215     end
216     display(p1)
217 end

```

```

218
219 function learningCurve_time(env,episodes, n_episodes)
220     p2 = plot(xlabel="wall clock time", ylabel="avg return")
221     n = convert{Int64,floor(n_episodes/10)}
222     stop = n_episodes
223     for (name, eps) in episodes
224         if(name == "SARSA-λ")
225             Q = Dict{(s, a) => 0.0 for s in observations(env), a in actions(env))
226             xs = [0.0]
227             ys = [mean(evaluate(env, s->argmax(a->Q[(s, a)], actions(env))))]
228             for i in n:n:min(stop, length(eps))
229                 newtime = sum(ep.time for ep in eps[i-n+1:i])
230                 push!(xs, last(xs) + newtime)
231                 Q = eps[i].Q
232                 push!(ys, mean(evaluate(env, s->argmax(a->Q[(s, a)], actions(env)))))
233             end
234         else
235             xs = [0.0]
236             thetas = Dict{(s) => 0.5*ones(4) for s in observations(env))
237             ys = [mean(evaluate(env, s->eps[1].policy(s, thetas)))]
238             for i in n:n:min(stop, length(eps))
239                 newtime = sum(ep.time for ep in eps[i-n+1:i])
240                 push!(xs, last(xs) + newtime)
241                 thetas = eps[i].θ
242                 push!(ys, mean(evaluate(env, s->eps[i].policy(s, thetas)))))
243             end
244         end
245         plot!(p2, xs, ys, label=name)
246     end
247     display(p2)
248 end
249
250 env = gw
251 n_eps= 150000
252 alpha=0.6
253 PolicyGrad_episodes = policyGradient(env, n_eps, alpha)
254 lambda_episodes = sarsa_lambda!(env, n_episodes=n_eps, α=0.1, λ=0.3)
255 display(render(env))
256 episodes = Dict("Policy Gradient"=>PolicyGrad_episodes, "SARSA-λ"=>lambda_episodes)
257 learningCurve_steps(env, episodes, n_eps)
258 learningCurve_time(env, episodes, n_eps)
259

```