HW4\HW4_5264_AmandaMarlow.jl

```
using DMUStudent.HW4: gw
    using LinearAlgebra: I
    using CommonRLInterface: render, actions, act!, observe, reset!, AbstractEnv, observations,
    terminated, clone
    using SparseArrays
    using Statistics: mean
 6
    using Plots
 7
    using StaticArrays: SA
    using StatsBase
 8
 9
10
    ############
    # SARSA-λ #
11
12
    ############
13
    function sarsa_lambda_episode!(Q, env; \epsilon=0.10, \gamma=0.99, \alpha=0.05, \lambda=0.9)
14
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)
         r = act!(env, a)
28
29
         sp = observe(env)
30
         hist = [s]
31
         N = Dict((s, a) \Rightarrow 0.0)
32
33
         while !terminated(env)
34
             ap = policy(sp)
35
36
             N[(s, a)] = get(N, (s, a), 0.0) + 1
37
             \delta = r + \gamma * Q[(sp, ap)] - Q[(s, a)]
38
39
             for ((s, a), n) in N
40
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)
             push!(hist, sp)
49
50
         end
51
52
         N[(s, a)] = get(N, (s, a), 0.0) + 1
```

```
53
          \delta = r - Q[(s, a)]
 54
 55
          for ((s, a), n) in N
              Q[(s, a)] += \alpha*\delta*n
 56
 57
              N[(s, a)] *= \gamma * \lambda
 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...)
          Q = Dict((s, a) \Rightarrow 0.0 \text{ for } s \text{ in observations(env)}, a \text{ in actions(env)})
 64
 65
          episodes = []
 66
 67
          for i in 1:n episodes
 68
              reset!(env)
 69
              push!(episodes, sarsa_lambda_episode!(Q, env;
 70
                                                         \in = \max(0.01, 1-i/n \text{ episodes}),
 71
                                                         kwargs...))
 72
          end
 73
 74
          return episodes
 75
     end
 76
 77
     function evaluate(env, policy, n_episodes=1000, max_steps=1000, \gamma=1.0)
 78
          returns = Float64[]
          for _ in 1:n_episodes
 79
              t = 0
 80
              r = 0.0
 81
 82
              reset!(env)
              s = observe(env)
 83
              while !terminated(env)
 84
                   a = policy(s)
 85
                   r += \gamma^*act!(env, a)
 86
 87
                   s = observe(env)
 88
                   t += 1
 89
 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))]
          elseif a == A[3]
106
```

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

```
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
              \theta[s] += \alpha*gradU/10
170
              push!(hist,[s])
171
         end
172
173
         return (hist=hist, \theta = \text{copy}(\theta), time=time()-start, policy = policy)
174
     end
175
     function policyGradient(env, n episodes, \alpha)
176
177
         \theta = Dict((s) \Rightarrow 0.5*ones(4) for s in observations(env))
         episodes = []
178
179
180
         for i in 1:n episodes
181
              reset!(env)
182
              push!(episodes, policyGradEpisode!(env, \theta, \alpha))
183
         end
184
185
         return episodes
     end
186
187
188
     function learningCurve steps(env,episodes, n episodes)
         p1 = plot(xlabel="steps in environment", ylabel="avg return")
189
190
         n = convert(Int64,floor(n episodes/10))
         stop = n episodes
191
192
         for (name, eps) in episodes
              if(name == "SARSA-\lambda")
193
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))))]
                  for i in n:n:min(stop, length(eps))
197
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
              else
203
204
                  xs = [0]
                  thetas = Dict((s) \Rightarrow 0.5*ones(4) for s in observations(env))
205
                  ys = [mean(evaluate(env, s->eps[1].policy(s,thetas)))]
206
                  for i in n:n:min(stop, length(eps))
207
208
                       newsteps = sum(length(ep.hist) for ep in eps[i-n+1:i])
209
                       push!(xs, last(xs) + newsteps)
210
                      thetas = eps[i].\theta
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
     function learningCurve time(env,episodes, n episodes)
219
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-\lambda")
225
                  Q = Dict((s, a) => 0.0 \text{ for } s \text{ in observations(env)}, a \text{ in } actions(env))
                 xs = [0.0]
226
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])
                      push!(xs, last(xs) + newtime)
230
                      Q = eps[i].Q
231
232
                      push!(ys, mean(evaluate(env, s->argmax(a->Q[(s, a)], actions(env)))))
233
                  end
             else
234
235
                  xs = [0.0]
                  thetas = Dict((s) => 0.5*ones(4) for s in observations(env))
236
237
                 ys = [mean(evaluate(env, s->eps[1].policy(s, thetas)))]
                  for i in n:n:min(stop, length(eps))
238
239
                      newtime = sum(ep.time for ep in eps[i-n+1:i])
                      push!(xs, last(xs) + newtime)
240
                      thetas = eps[i].\theta
241
242
                      push!(ys, mean(evaluate(env, s->eps[i].policy(s, thetas))))
243
                  end
244
             end
             plot!(p2, xs, ys, label=name)
245
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, \alpha=0.1, \lambda=0.3)
255
     display(render(env))
     episodes = Dict("Policy Gradient"=>PolicyGrad_episodes, "SARSA-λ"=>lambda_episodes)
256
257
     learningCurve steps(env, episodes, n eps)
258
     learningCurve time(env, episodes, n eps)
259
```