Reinforcement Learning: An Introduction - Chapter 7

Thomas Hopkins

Exercise 7.1

The larger random walk task was most likely used because it would allow for larger n steps to be taken and tested. A smaller walk would most likely shift the value of n down. The change from the left-side end state from 0 to -1 would cause larger n to be more useful since a reward other than 0 is computed along the n-step return if it reaches that terminal state.

Exercise 7.2

On-line methods most likely worked better than off-line methods because better estimates for values were used earlier in the learning process. Waiting until the end of the episode delays the ability to use the most up-to-date information. On the other hand, on-line methods allow this new information to be used immediately.

Exercise 7.3

The only explanation I can think of is that 3-step returns are particularly noisy for this choice of initialization and number of states in the random walk. So, using a higher learning rate, α, causes the error to increase sharply over the first 10 episodes.

Exercise 7.4

The equation for the weighting at time t given by λ and its half-life τ_λ is

$$(1-\lambda)\lambda^t = (1-\lambda)(\frac{1}{2})^{t/\tau_\lambda}$$

$$\Rightarrow \lambda^t = (\frac{1}{2})^{t/\tau_\lambda}$$

$$\Rightarrow t\ln(\lambda) = \frac{t}{\tau_\lambda}\ln(\frac{1}{2})$$

$$\Rightarrow \tau_\lambda = -\frac{\ln(2)}{\ln(\lambda)}$$

Exercise 7.5

$$\Delta V_t(s_t) = \alpha \delta_t e_t(s)$$

$$= \alpha(r_{t+1} + \gamma V_t(s_{t+1}) - V_{t-1}(s_t))(\sum_{k=0}^{t}(\gamma\lambda)^{t-k} I_{ss_k})$$

from the derviation on page 177, we have

$$= \alpha[R_t^\lambda - V_{t-1}(s_t)]$$

This method would benefit from being able to update values during the episode. As described before, this helps in using recent information sooner. In this case, however, you will only get the immediate information of the return on-line. It might be worse since the other on-line method would be able to use the updated values of each state in its update as well.

An experiment to assess the relative merits would be to create two environments. One would have large immediate rewards and the other would have small immediate rewards. Run both

Exercise 7.6

For accumuulating traces, the update is:

$$e_t(s,a) = \begin{cases} \gamma\lambda e_{t-1}(s) + 1 & s_t = s, a_t = a \\ \gamma\lambda e_{t-1}(s) & s_t = s, a_t = a \end{cases}$$

With two $a = wrong$ then one $a = right$, we have

$$e_1(s, wrong) = 1$$

$$e_1(s, right) = 0$$

$$e_2(s, wrong) = \gamma\lambda + 1$$

$$e_2(s, right) = 0$$

$$e_3(s, wrong) = \gamma\lambda(\gamma\lambda + 1)$$

$$e_3(s, right) = 1$$

Assuming $\gamma = 1$ we have

$$\lambda(\lambda + 1) > 1$$

$$\Rightarrow \lambda^2 + \lambda - 1 > 0$$

$$\Rightarrow \lambda > -\frac{1-\sqrt{5}}{2}$$

If $\lambda > 0.618$, then $wrong$ will have a larger eligibility trace than $right$.

Exercise 7.7

In [2]:

```
mutable struct StateActionPair
    state::Int
    action::String
    trace::Float64
    value::Float64
end

# 1 is right, 2 is wrong for actions
function sarsa_lambda(num_states::Int; num_episodes = 100, alpha = 0.1,
                      gamma = 1.0, lambda = 0.9, epsilon = 0.5, replacing_tra
    env = [[StateActionPair(i, "right", 0.0, 0.0),
            StateActionPair(i, "wrong", 0.0, 0.0)]
           for i = 1:num_states]

    for e = 1:num_episodes
        s = env[1]
        sap = rand(s)
        terminated = false
        while !terminated
            new_s = s
            reward = 0.0
            if sap.action == "right"
                if sap.state == num_states
                    terminated = true
                    new_s =
                    reward = 1.0
                else
                    new_s = env[sap.state + 1]
                end
            end
            if terminated
                new_sap = StateActionPair(num_states + 1, "none", 0.0, 0.0)
            else
                new_sap = new_s[argmax([new_s[1].value, new_s[2].value])]
                if rand() < epsilon
                    new_sap = rand(new_s)
                end
            end
            delta = reward + gamma * new_sap.value - sap.value
            if replacing_trace
                sap.trace = 1
            else
                sap.trace = sap.trace + 1
            end
            for i = 1:num_states
                for j = 1:2
                    env[i][j].value = env[i][j].value + alpha * delta * env[i
                    env[i][j].trace = gamma * lambda * env[i][j].trace
                end
            end
            sap = new_sap
        end
    end
    return env
end

function display_values(env)
    println("Values for each state")
```

```
    println("--------------------")
    for s in env
        println("State: $(s[1].state)")
        println("Right: $(s[1].value)")
        println("Wrong: $(s[2].value)")
    end
end;
```

In [10]:

```
using Random

Random.seed!(32)
N = 5
episodes = 10
alpha = 0.9
epsilon = 0.1
accumulate_env = sarsa_lambda(N; num_episodes = episodes, alpha = alpha, epsi
replacing_env = sarsa_lambda(N; num_episodes = episodes, alpha = alpha, epsil
println("Accumulating Trace")
display_values(accumulate_env)
println("\nReplacing Trace")
display_values(replacing_env)
```

```
Accumulating Trace
Values for each state
---------------------
State: 1
Right: -2.1463301237429342e9
Wrong: 3.1251120520893908e9
State: 2
Right: -1.9868580042852964e9
Wrong: 5.301282924569049e8
State: 3
Right: -1.756881357948165e8
Wrong: 6.227890555493832e8
State: 4
Right: -6.194646549525633e8
Wrong: -51971.52634669248
State: 5
Right: -1.117016311869059e9
Wrong: 2.1175084255286857e7

Replacing Trace
Values for each state
---------------------
State: 1
Right: 0.9674348569717787
Wrong: 0.9039216250956357
State: 2
Right: 0.9840439068929528
Wrong: 0.0
State: 3
Right: 0.9994466628362726
Wrong: 0.2238675776586064
State: 4
Right: 1.0042490635732428
Wrong: 0.0
State: 5
```

```
Right: 1.0074991628992664
Wrong: 0.0
```

By trying different values of α, we see that replacing traces allow for a much larger value while accumulating traces are unusable at larger values.

Exercise 7.8

The backup diagram would be the same as Figure 7.10 since the weighting does not account for repeated states. The change would be in how you combine the different weights to get the real trace for a particular state-action pair. This would have to be a maximum of 1 for replacing traces.

Exercise 7.9

```
Initialize V(s) arbitrarily and e(s) = 0, for all s in S
Repeat (for each episode):
    Initialize s
    Repeat (for each step of episode):
        a <- action given by pi for s
        Take action a, observe reward, r, and next state, s'
        delta <- r + gamma * V(s') - V(s)
        e(s) <- e(s) + 1
        For all s:
            if e(s) > c:
                V(s) <- V(s) + alpha * delta * e(s)
            e(s) <- gamma * lambda * e(s)
        s <- s'
    until s is terminal
```

Exercise 7.10

This proof is very similar to that in Section 7.4 and Exercise 7.5 above except that we now have a variable λ term that depends on t. The only change to the proof is the addition of a subscript to λ indicating which time step the particular λ_t is referring to.