**Video Lecture Notes**

project starts at 14:36

use the numbers that are distributed with the code

smallest number of steps to goal is 16 steps

line 133 sleep statement slows it down. remove to increase speed

constantly calls the query of Q learner and

questions start at 29:30

Testing you should be able to solve this maze in 36 steps.

then we’ll turn on dyna and expect convergence to a smaller number more quickly.

reward is for moving into the goal state, not for being in the goal state

each robot movement you see is one call to the query()

query() updates the Q table:

Assume we have T

T[s, a ,s’] – the probability given s and a that we’ll end up in s’

so sum of all T over s’ = 1

the correct Q table will be mainly 0’s and occasional 1. Because if you’re in a particular cell and you take a particular action, eg. up, the prob = 1 that you’ll end up in the cell above.

when up against an obstacle, and try to go past it, the test learner doesn’t let you go by, so the Transition matrix T, eg. (55, N, 1) (Same part) and all else 0.

… also selling stocks may not always go up 1%, might go down 2%

Assume we have R[s, a] = reward in state s for action a

can Hallucinate the experience tuple:

1. randomly select s

2. randomly select a

3. consult T to find s’

**to find s’ :**

**generate rand # between 0,1 np.random.random()**

**looking T[s,a,?] find s’**

**s and a are fixed. use a for loop to accumulate the sum of the probabilities as you iterate over all possible s’**

**when that sum reaches the value of rand, that’s the action you select**

**more precisely: you should randomly select s’ according to the distribution. One of those next states is going to be the most likely.**

**in our case there’s going to be one of the s’ T = 1, so if you draw randomly you’ll always pick that state**

**Example:**

for simplicity, assume we have a small state space , numStates =4

T[s,a,0]

T[s,a,1] , 2, 3 .

T[ … 0] = 0

T1 = 0.6

T2 = .2

T3 = .2

So this is a dist. of probs.

so we want to randomly pull one of these states out of its distribution according to how likely they are

the most likely next state is 1.

rand = .72

**use for loop and step through sums of probs until >= rand**

**so 0, .6 , .8**

**so choose action 2**

now do this 100 times, and count number of times picked state 1

would’ve picked state 1 about 60 times, 2 20 times, and 3 20 times

end example

back to above (contds. from step 3)

**1. randomly select s**

**2. randomly select a**

**3. consult T to find s’**

**4. r = R[s,a]**

**and that’s everything you need to update Q table**

**50:52**

**How to get T?**

**remember your query function is always being called by the state that you’re in**

Observe s, a, s’

suggestion: create a new matrix: Tc (T count)

In constructor, set Tc = 0.000001

Tc [s ,a, s’] += 1

How to compute T[s,a,s'] from Tc?

T[s, a, s’] = Tc[s,a,s’] / sum(T[s,a,:])

Assume we’re in **Within query()**

<s,a,s’,r>

update Q-table

throw dice, randomly select action according to rar

decay rar rar = rar \* radr

otherwise choose action according to Q

**Dyna part:**

update Tc[s,a,s’] +=1

update T[s,a, :]

update R[s,a] = (1 – alpha\_r) \* R[s, a] + alpha\_r \* r

for in range(0, dyna)

select random s, a

draw s’ from T(s,a,:]

r=R[s,a]

update Q with <s,a,s’,r>

return action

Questions:

For the update let alpha\_r = 1 and then it would fully believe the rewards.

My suggestion is to initialize Tc = 0.000001 then compute T from that count and it will end up with uniform probabilities everywhere.