Cse 573

Ubit :v33

Ub#:50292192

Introduction:

In this project, we combine reinforcement learning and deep learning .Our task is to train Tom using Agent in grid environment to catch jerry in shortest route. This problem is solved using DQN .

Understanding and Implementation:

Environment :

Our environment a grid of size 5x5 where Tom is initialized at (4,4) an jerry at (0,0). We assume the position of jerry to be fixed. we represent states using four parameters fx,fy,px,py where (fx,fy) are the co-ordinates of jerry(fixed) and (px ,py) are co-ordinates of Tom. Variation of <fx,fy,px,py> gives several states. In this case we have 4 different actions up down right and left. Each every kind of action depending on the previous state and next state generates a reward. For this environment we define reward values as 0 if Tom's position remains unchanged , -1 if tom doesn't reach jerry and 1 if tom reaches jerry . Thus for every action we have a <S,A,R,S'> sequence .

Agent :

agent will learn from the environment by interacting with it and receiving rewards for performing actions. It .All the changes to the environment and training the model is done by agent .

Deep -Q -network:

In Q-learning we define a function *Q(s, a)* representing the maximum discounted future reward when we perform action **a** in state **s**, and continue optimally from that point on.

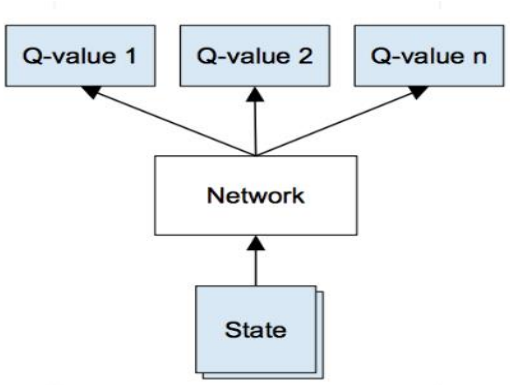
$$Q(s_t, a_t) = max\ R_{t+1}$$

Q(s, a) gives the best possible score at the end of the game after performing action a in state **s** . This is done by iteratively

```
repeat
      select and carry out an action a
      observe reward r and new state s'
      Q[s,a] = Q[s,a] + α(r + γ max_a' Q[s',a'] - Q[s,a])
      s = s'
until terminated
```

This is performed iteratively until we get optimum q table . In the q table rows represent different states and coloumns represent different action. **For a complex reinforcement problem we have very large number of states and updating each state is not feasible method.Thus we use deep q network**

In deep Q networks our nueral network is the q function which takes the state as input and gives q-values for all possible actions. In this project we use 2 hidden layers and input layer has 4 inputs (fx,fy,px,py) and ouputs 4 q values for each action



Exploration , exploitation, epsilon

Action can be taken in two possible ways . One is to take random actions and explore the environment and observe. another way is to take an action using optimal q value. We want to use both the methods But initially Q-network is initialized randomly, then its predictions are initially random as well. If we pick an action with the highest Q-value, the action will be random and the agent performs crude "exploration". As a Q-function converges, it returns more consistent Q-values and the amount of exploration decreases. So we define a parameter **epsilon** and decrease the value of epsilon exponentially . proportional to the value of epsilon we decide the rate of exploration vs exploitation.

$$\epsilon=\epsilon_{min}+(\epsilon_{max}-\epsilon_{min})*e^{-\lambda|S|},$$

where $\epsilon_{min},\epsilon_{max}\in[0,1]$
λ - hyperparameter for epsilon
$|S|$ - total number of steps

memory :

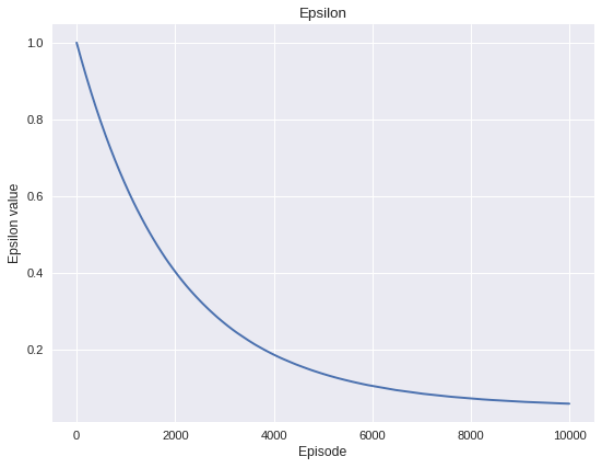
The problem with training the nueral networks with online data is that the network is overfitted .Once DNN is overfitted, it's hard to produce various experiences. To solve this problem, we use the concept of experience Replay . Replay stores experiences including state transitions, rewards and actions, which are necessary data to perform Q learning, and makes mini-batches to update neural networks. Storing all experience in a replay buffer allows us to train on more independent samples. We just draw a batch of transitions from the **buffer(memory**) at random and train on that. This helps break the temporal correlation of training samples.

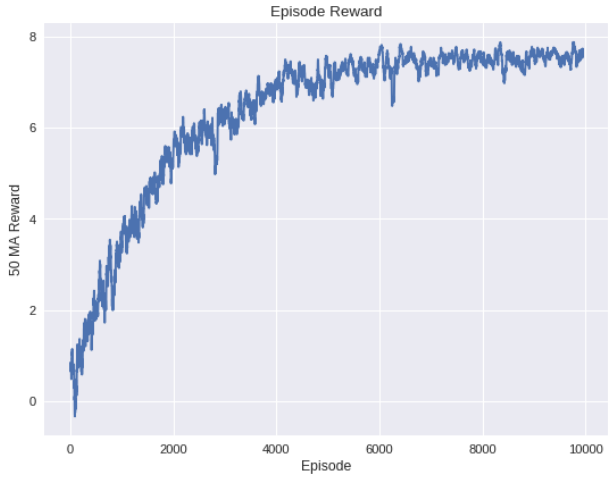
Results

Epsilon-max/min	Episodes	lambda	gamma	Episode reward mean	time
1/0.0005	10000	0.0005	0.5	6.49	888.3
1/0.005	5000	0.0005	0.99	5.03	458
3/0.0005	10000	0.0005	0.99	4.49	921
2/0.0005	10000	0.0005	0.99	5.132	797
1/o.1	10000	0.0005	0.99	6.04	813.26
1/0.0005	10000	0.005	0.99	7.386	801
1/0.06	10000	0.0005	0.1	6.49	786.75
1/0.5	10000	0.0005	0.99	3.60	839.62

Let us analyse each of the result .

1. default values



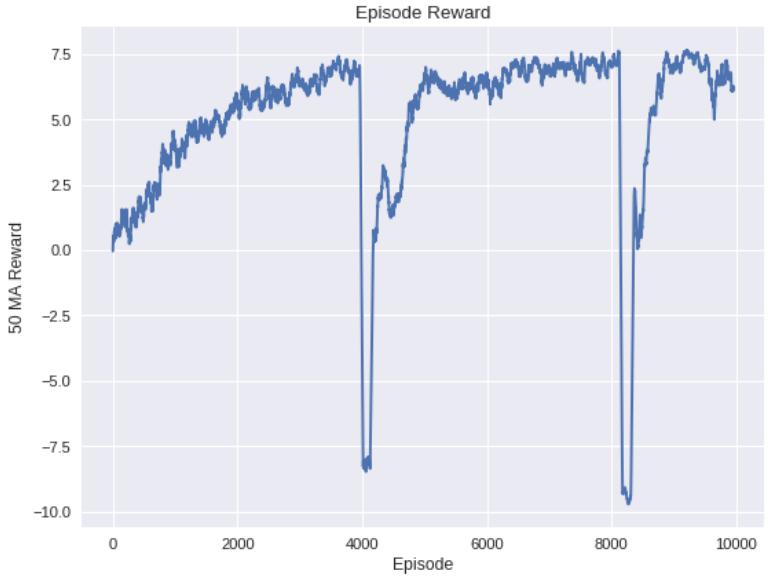


We observe that the value of epsilion decreases exponentially .
From the episode reward graph we can see that the model converges around 7000

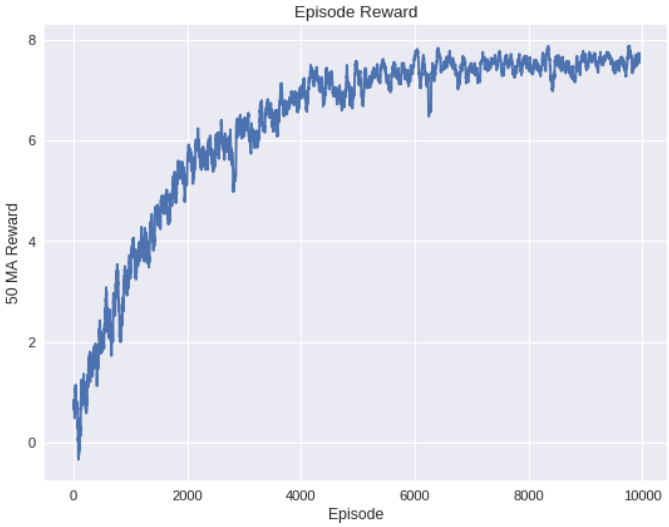
Row no: 2
For 5000 episodes the model does not converge which reflects in the mean episode reward (5.03)

Row no: 1,3,5
from the table we see that as we increase the value of the max-epsilon the mean episode reward decreases
From rows 1,2,6

We see that decreasing the gamma seems to have no huge change in the mean reward. As the no of states and actions are small in number taking action based on immediate reward (decreasing discount factor) might not have much impact on mean reward.
Upon increasing the no of neurons in the network to 400, 400 the value of episode mean reward was 4.20



Mean episode graph for network with (400,400,400 nuerons in 3 hidden layers)



Time Elapsed: 496.22s

Epsilon 0.12899716648889398

Last Episode Reward: 4

Episode Reward Rolling Mean: 2.7301295786144273

when we decrease the grid size to 3 we see that there is significant decrease in the time elapsed.

When we increase the lamda from 0.0005 to 0.005 we observe that the mean episode reward is 7.43

Optimum parameters : lamda =0.005 , episodes =7000, gamma = 0.99, epsilon _max = 1 , epsilon min = 0.005

Writing task 1

A:

- Initially the q values of the q function are random.so if we act according to q table then, The agent might be stuck performing non-optimal actions. Unless it has already found the optimal policy, exploring actions which do not have the highest Q-value might allow it to find a better policy.

B:

- The ε-greedy strategy is to select the greedy action (one that maximizes Q[s,a]) all but ε of the time and to select a random action ε of the time, where $0 \leq \varepsilon \leq 1$.

- An alternative is "optimism in the face of uncertainty": initialize the Q-function to values that encourage exploration. If the Q-values are initialized to high values, the unexplored areas will look good, so that a greedy search will tend to explore.

Writing task2

Q(st; at) = rt + l * max Q(st+1; a)

l = 0.99

State	Up	Down	Right	Left
0	3.90	3.94	3.94	3.90
1	2.94	2.97	2.97	2.90
2	1.94	1.99	1.99	1.94
3	0.97	1	0.99	0.97
4	0	0	0	0

Now let's see each state individually,

S0	S1b	Pb
S1a	S2	S3b

Pa S3a Goal

S1a and S1b is same due to symmetry and therefore considered as S1 only.

Pa and Pb are same due to symmetry and therefore considered as P only.

S3a and S3b is same due to symmetry and therefore considered as S3 only.

Now for state S4, going to any direction will yield zero reward and therefore, we fill the above table with zeros.

Now for state S3b,

Action UP

Qt3 = -1 + 0.99 * max QtP = -1 + 0.99*1.99 = -1 + 1.97 = 0.97

Action DOWN

Qt3 = 1 + 0.99 * max QtG = 1

Action LEFT

Qt3 = -1 + 0.99 * max Qt2 = - 1 + 0.99*1.99 = 0.97

Action RIGHT

Qt3 = 0 + 0.99 * max Qt3 = 0.99

Now for state S2,

Action UP

Qt2 = -1 + 0.99 * max Qt1 = -1 + 0.99 *2.97 = 1.94

Action DOWN

Qt2 = 1 + 0.99 * max Qt3 = 1.99

Action RIGHT

Qt2 = 1 + 0.99 * max Qt3 = 1.99

Action LEFT

Qt2 = -1 + 0.99 * max Qt1 = -1 + 0.99 *2.97 = 1.94

Now for state S1b,

Action UP

Qt1 = 0 + 0.99 * max Qt1 = 0.99 *2.97 = 2.94

Action DOWN

Qt1 = 1 + 0.99 * max Qt2 = 1+ 0.99*1.99 = 1+1.97 = 2.97

Action LEFT

Qt1 = -1 + 0.99 * max Qt0 = -1 + .99*3.94 = 2.90

Action RIGHT

Qt1 = 1 + 0.99 * max QtP = 1+ 0.99*1.99 = 1+1.97 = 2.97

Now for state Pb,

Action UP

QtP = 0 + 0.99 * max QtP = 1.97

Action DOWN

QtP = 1 + 0.99 * max Qt3 = 1.99

Action RIGHT

QtP = 0 + 0.99 * max QtP = 1.97

Action LEFT

QtP = -1 + 0.99 * max Qt1 = -1 + 0.99*2.97 = 1.94

Now for state S0,

Action UP

Qt0 = 0 + 0.99 * max Qt0 = 3.90

Action DOWN

Qt0 = 1 + 0.99 * max Qt1 = 1+ 0.99*2.97 = 3.94

Action RIGHT

Qt0 = 1 + 0.99 * max Qt1 =1+ 0.99*2.97 = 3.94

Action LEFT

Qt0 = 0 + 0.99 * max Qt0 = 3.90

References:

www.aispace.org

https://ai.intel.com/demystifying-deep-reinforcement-learning/