Q-Learning with UCB Exploration

IISc, M.Tech (Online) Aug 2022, EA-277o

11 Dec 2022

Anisha
Arkadip Basu
Naveen Rupdrapa
Yatish Reddy

Motivation

- To improve the performance of **Model Free RL algorithm** for infinite horizon MDP

- Deal with infinite horizon MDPs without access to simulator

- Before the paper "Q-LEARNING WITH UCB EXPLORATION IS SAMPLE EFFICIENT FOR INFINITE-HORIZON MDP" was published. The best know sample complexity of exploration was achieved by delayed Q-learning $\tilde{O}(\frac{SA}{\epsilon^4(1-\gamma)^8})$

  Big O Tilde, log factors can be ignored

- *Several model-based algorithms have been proposed for infinite horizon MDP. However, there still exists a considerable gap between the state-of-the-art algorithm and the theoretical lower bound regarding* 1/(1- γ) factor

- *The performance measure <u>cannot be</u> a straightforward extension of the sample complexity defined by finite horizon setting.*

Q-Learning

Q-learning lets the agent use the environment's rewards to learn, over time, the best action to take in a given state.

The values store in the Q-table are called a Q-values, and they map to a (state, action) combination.

Q-values are initialized to an arbitrary value, and as the agent exposes itself to the environment and receives different rewards by executing different actions, the Q-values are updated using the equation**:**

$$Q(state, action) \leftarrow (1-\alpha)Q(state, action) + \alpha\left(reward + \gamma \max_{a} Q(next\ state, all\ actions)\right)$$

a (alpha) is the learning rate, γ (gamma) is the discount factor.

After enough random exploration of actions, the Q-values tend to converge serving our agent as an action-value function which can be exploited to pick the most optimal action from a given state.

Q learning with UCB exploration

Q-learning Algorithm:

- For $t = 1, 2, \cdots$
  - Act $a_t = \arg\max_a Q(s_t, a)$,
  - $k \leftarrow$ number of times (s_t, a_t) is visited,
  - $Q(s_t, a_t) \leftarrow (1 - \alpha_k)Q(s_t, a_t) + \alpha_k \left[r_t + \gamma V(s_{t+1})\right]$.

Q-learning with Hoeffding-style UCB exploration bonus:

- For $t = 1, 2, \cdots$
  - Act $a_t = \arg\max_a Q(s_t, a)$,
  - $Q(s_t, a_t) \leftarrow (1 - \alpha_k)Q(s_t, a_t) + \alpha_k \left[r_t + \gamma V(s_{t+1}) + \tilde{\Theta}\left(\sqrt{\frac{1}{(1-\gamma)^3 k}}\right)\right]$.

Terminologies

- ***Sample complexity of Exploration:*** *Sample complexity of Exploration of an algorithm is defined as the number of time steps t such that the non-stationary policy π_t at time t, is not ε-optimal for current state s_t.*

$$V^{\pi_t}(s_t) < V^*(s_t) - \epsilon.$$

- ***Probably Approximately Correct in Markov Decision Processes (PAC-MDP):*** *An algorithm is said to be PAC-MDP if, for any ε and δ, the sample complexity of ALG is less than some polynomial in the relevant quantities* $(S, A, 1/\epsilon, 1/\delta, 1/(1-\gamma))$ *with probability at least 1-δ.*

- ***Bellman equation:*** long-term- reward in a given action is equal to the reward from the current action combined with the expected reward from the future actions taken at the following time.

$$\begin{cases} V^{\pi_t}(s) = Q^{\pi_t}(s, \pi_t(s)) \\ Q^{\pi_t}(s, a) := \left(r_t + \gamma \mathbb{P} V^{\pi_{t+1}}\right)(s, a), \end{cases} \quad \begin{cases} V^*(s) = Q^*(s, \pi^*(s)) \\ Q^*(s, a) := \left(r_t + \gamma \mathbb{P} V^*\right)(s, a), \end{cases}$$

Complexity measurement

- The main bottleneck in the infinite horizon setting, the agent may enter under-explored regions at any time period, and sample complexity of exploration characterizes the performance at all states the agent enters.
- First we need to establish convenient sufficient conditions for being -optimal at timestep t and state s_t

i.e.

$$V^*(s_t) - V^{\pi_t}(s_t) \leq \epsilon.$$

Infinite Q-learning with UCB

3.1 ALGORITHM

Algorithm 1 Infinite Q-learning with UCB

Parameters: ϵ, γ, δ
Initialize $Q(s,a), \hat{Q}(s,a) \leftarrow \frac{1}{1-\gamma}$, $N(s,a) \leftarrow 0$, $\epsilon_1 \leftarrow \frac{\epsilon}{24RM \ln \frac{1}{1-\gamma}}$, $H \leftarrow \frac{\ln 1/((1-\gamma)\epsilon_1)}{\ln 1/\gamma}$.
Define $\iota(k) = \ln(SA(k+1)(k+2)/\delta)$, $\alpha_k = \frac{H+1}{H+k}$.
for $t = 1, 2, \ldots$ **do**
  Take action $a_t \leftarrow \arg\max_{a'} \hat{Q}(s_t, a')$
  Receive reward r_t and transit to s_{t+1}
  $N(s_t, a_t) \leftarrow N(s_t, a_t) + 1$
  $k \leftarrow N(s_t, a_t)$, $b_k \leftarrow \frac{c_2}{1-\gamma}\sqrt{\frac{H\iota(k)}{k}}$ ▷ c_2 is a constant and can be set to $4\sqrt{2}$
  $\hat{V}(s_{t+1}) \leftarrow \max_{a \in A} \hat{Q}(s_{t+1}, a)$
  $Q(s_t, a_t) \leftarrow (1 - \alpha_k)Q(s_t, a_t) + \alpha_k \left[r(s_t, a_t) + b_k + \gamma \hat{V}(s_{t+1}) \right]$
  $\hat{Q}(s_t, a_t) \leftarrow \min(\hat{Q}(s_t, a_t), Q(s_t, a_t))$
end for

UCB Q-learning algorithm (Algorithm 1) maintains an optimistic estimation of action value function Q(s, a) and its historical minimum value Qˆ(s, a).

Complexity of the Algorithm

- With some mathematical theorems and assumptions it has been proved in the paper that with probability 1 − δ, the number of time steps such that $\left(V^{*} - V^{\pi}\right)(s_t) > \epsilon$ is

$$\tilde{O}\left(\frac{SA\ln 1/\delta}{\epsilon^{2}(1-\gamma)^{7}}\right)$$

- The complexity is better than delayed q learning.

Demo : Taxi problem

There are 500 discrete states since there are 25 taxi positions, 5 possible locations of the passenger (including the case when the passenger is the taxi), and 4 destination locations

Rewards: There is a reward of -1 for each action and an additional reward of +20 for delievering the passenger. There is a reward of -10 for executing actions "pickup" and "dropoff" illegally.
Rendering:

```
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

- The filled square represents the taxi, which is yellow without a passenger and green with a passenger.
- The pipe ("|") represents a wall which the taxi cannot cross.
- R, G, Y, B are the possible pickup and destination locations.
- Action Space: 6, State Size: 500

Actions:

- 0: move south
- 1: move north
- 2: move east
- 3: move west
- 4: pickup passenger
- 5: dropoff passenger

Fine tuning

- ½<gamma<1
- R=(math.log(3/(epsilon_hyp*(1-gamma)))/(1-gamma))
- M=2*math.log2(1/((1-gamma)*epsilon_hyp))
- epsilon1=(epsilon_hyp/(24*R*M*math.log(1/(1-gamma))))
- H=(math.log(1/(epsilon1*(1-gamma)))/math.log(1/gamma))
- qcap_table = np.full((state_size, action_size),q_initial_value)
- n_table = np.zeros((state_size, action_size))
- q_table = np.zeros((state_size, action_size))

- max_epsilon = 1.0
- min_epsilon = 0.01
- decay_rate = 0.01
- gamma = 0.9
- c2=4*math.sqrt(2)

Changes made to Algorithm

```
for episode in range(total_ep):
    state = env.reset()
    step = 0
    done = False

    for step in range(max_steps):

        exp_exp_tradeoff = random.uniform(0,1)
        #Take action at ← arg maxa0 Q^(st, a0)
        if exp_exp_tradeoff > epsilon:
            action = np.argmax(qcap_table[state, :])
        else:
            action = env.action_space.sample()

        # Receive reward rt and transit to st+1
        new_state, reward, done, info = env.step(action)

        #N(st, at) ← N(st, at) + 1
        n_table[state, action] =n_table[state, action]+1

        #k ← N(st, at)
        k=n_table[state, action]

        #update bk
        bk=(c2/(1-gamma))*math.sqrt((H*getlk(k,delta))/k)

        #action_max = np.argmax(qcap_table[new_state, :])
        vcapstplus1=np.max(qcap_table[new_state])
        # Update Q(s,a):= Q(s,a) + lr [R(s,a) + gamma * max Q(s',a') - Q(s,a)]
        alphak=getaplhak(H,k)
        q_table[state, action] = (1-alphak)*q_table[state, action] + alphak * (reward + bk+gamma * vcapstplus1)



        #Q^(st, at) ← min(Q^(st, at), Q(st, at))

        qcap_table[state, action]=min(q_table[state, action],qcap_table[state, action])
        state = new_state
        if done == True:
            break

        episode += 1

        epsilon = min_epsilon + (max_epsilon - min_epsilon) * np.exp(-decay_rate * episode)
```

Colab results

Total Episodes	Mean Score
15000	-1093.4
30000	-858.5
50000	-712.2
200000	-522.6

Reference

1. https://arxiv.org/abs/1901.09311
2. https://openreview.net/pdf?id=BkglSTNFDB
4. https://paperswithcode.com/task/q-learning
5. https://blog.paperspace.com/getting-started-with-openai-gym/
6. https://www.researchgate.net/publication/335805245_Q-Learning_Algorithms_A_Comprehensive_Classification_and_Applications

Code

https://www.kaggle.com/anishabhushan/q-learning-taxi-implementation-ucb-final

Thank You!

Anisha
Arkadip Basu
Naveen Rupdrapa
Yatish Reddy

IISc, M.Tech (Online)
Reinforcement Learning
EA-277o Aug 2022 batch