"به نام خدا"

تمرین چهارم درس یادگیری تعاملی

فردین عباسی 810199456

دانشکده مهندسی برق و کامپیوتر
دانشکدگان فنی
دانشگاه تهران

پاییز 1402

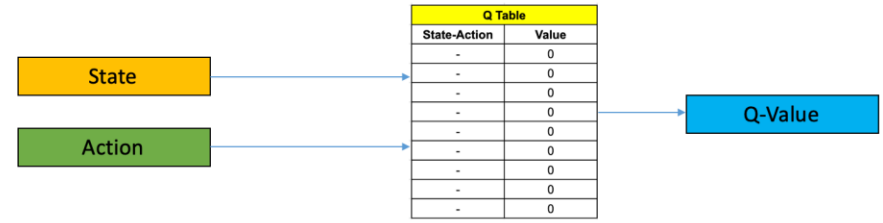
سوالات تحلیلی

1.

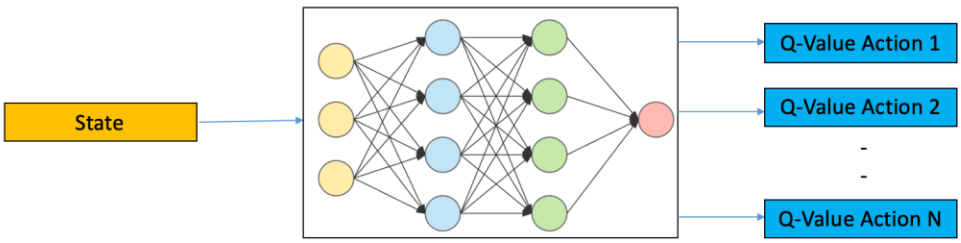
در بسیاری از کاربرد های دنیای واقعی استیت ها و یا اکشن ها فضای پیوسته ای دارند و استفاده از الگوریتم های کلاسیک یادگیری تعاملی مانند Q-Learning که برای تمامی حالت و اکشن ها یک جدول ارزش تهیه میکند عملا غیر ممکن است و نیاز به حافظه بسیار بزرگی دارد در نتیجه در عوض یک شبکه عصبی آموزش می دهیم که بتواند پارامتر ها را به گونه ای انتخاب کند که ارزش استیت-اکشن های متفاوت را به درستی به ما خروجی دهد. در مقابل این سود بزرگ RL Deep می توان گفت پیچیدگی الگوریتم های آن یک عیب به شمار می رود. به عبارتی هم از منظر پیاده سازی الگوریتم و انتخاب درست پارامتر ها (تعداد لایه ها و...) و هم زمان آموزش این الگوریتم ها بار محاسباتی زیادی دارند.

2.

در الگوریتم DQN به دلیل اینکه بر خلاف الگوریتم Q-Learning که از جدول برای ذخیره Q-values استفاده می کند، از روش های شبکه عصبی برای محاسبه Q-Values بهره می برد سبب می شود که بتوان از این الگوریتم در فضا های با حالت پیوسته و ابعاد بالا نیز بهره برد.



Q Learning



Deep Q Learning

برای اطلاعات بیشتر توصیه می شود این کتابچه تعاملی را مطالعه نمایید.

3.

در الگوریتم های Deep RL اگر قرار بود پس از هر حرکت شبکه را آپدیت کنیم به دلیل وابستگی حرکات پشت سر هم در محیط، فرآیند ما هم از نظر بازدهی و هم از نظر کیفیت آموزش دچار اختلال میشود. در نتیجه برای رفع این مشکل در هر مرحله (r,'s,a,s) در یک صف ذخیره شده و هر بار برای آموزش شبکه با سمپل برداری رندوم از آن جدول شبکه را آموزش میدهیم. با این تغییر که به آن replay Experience گفته می شود در واقع وابستگی بین اعمال و نتایج پشت سر هم را از بین میبریم و نتایج بسیار بهتری در همگرایی به سیاست بهینه کسب خواهیم کرد.

سؤال پیاده‌سازی

بخش اول

1.

Action Space

Num	Action	Unit
0	apply -1 torque to the actuated joint	torque (N m)
1	apply 0 torque to the actuated joint	torque (N m)
2	apply 1 torque to the actuated joint	torque (N m)

فضای اکشن ها گسسته و شامل گشتاور اعمالی به مفصل میانی است. فضای اکشن ها شامل 3 حالت، یک واحد گشتاور در خلاف جهت، بدون اعمال گشتاور و یک واحد گشتاور در جهت است.

Reward

هدف این است که با اعمال گشتاور مناسب به پاندول در حالت ایستا اولیه، بتوان آن را به ارتفاع مشخصی رساند. تا زمانی که به این ارتفاع نرسد پاداش(مجازات) 1- دریافت خواهد کرد.

Episode End

اگر عامل به ارتفاع مشخص شده برسد و یا به عبارتی $-cos(\theta_1) - cos(\theta_2 + \theta_1) > 1$ حاصل شود اپیزود Terminated می شود، در غیر اینصورت اگر طول اپیزود از 500 مرحله بیشتر شود، اپیزود Truncated می شود.

Observation Space

Num	Observation	Min	Max
0	Cosine of `theta1`	-1	1
1	Sine of `theta1`	-1	1
2	Cosine of `theta2`	-1	1
3	Sine of `theta2`	-1	1
4	Angular velocity of `theta1`	~ -12.567 (-4 * pi)	~ 12.567 (4 * pi)
5	Angular velocity of `theta2`	~ -28.274 (-9 * pi)	~ 28.274 (9 * pi)

فضای مشاهده عامل پیوسته و 6 بعدی است که شامل توابع مثلثاتی sin, cos از زوایای هر دو مفصل است.
همچنین سرعت زاویه ای هر دو مفصل نیز جزء فضای مشاهده است.

2.

همانطور که پیش تر ذکر شد فضای اکشن عامل گسسته و فضای مشاهده شامل ویژگی های فیزیکی پیوسته است.

بخش دوم

1.

شبه کد الگوریتم DQN به شرح زیر است:

Initialize network Q
Initialize target network $\hat{Q}$
Initialize experience replay memory D
Initialize the *Agent* to interact with the Environment
while *not converged* **do**

  /* Sample phase
  $\epsilon \leftarrow$ setting new epsilon with ϵ-decay
  Choose an action a from state s using policy ϵ-greedy(Q)
  Agent takes action a, observe reward r, and next state s'
  Store transition $(s, a, r, s', done)$ in the experience replay memory D

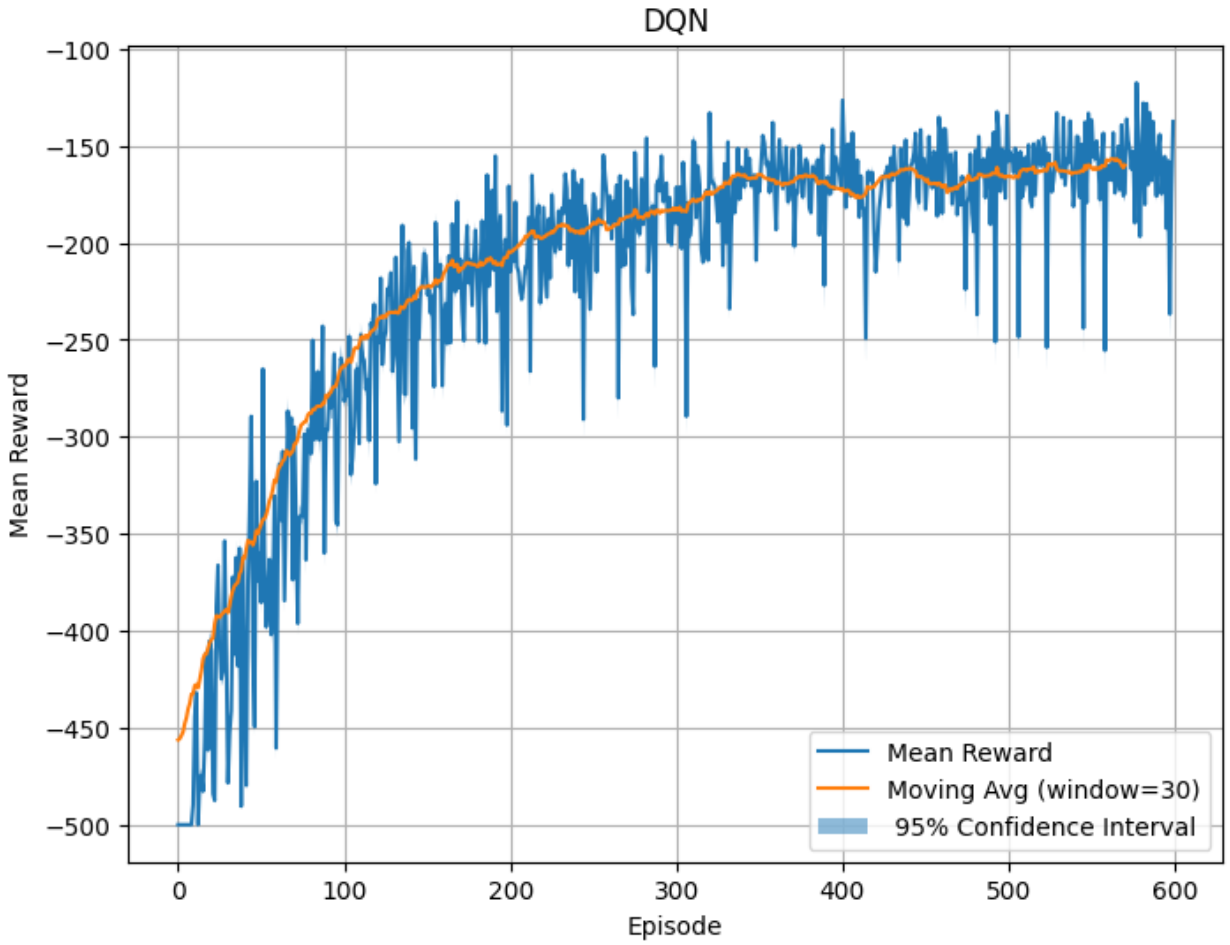
  if *enough experiences in* D **then**
    /* Learn phase
    Sample a random *minibatch* of N transitions from D
    for *every transition* $(s_i, a_i, r_i, s'_i, done_i)$ *in minibatch* **do**
      if $done_i$ **then**
        $y_i = r_i$
      else
        $y_i = r_i + \gamma \max_{a' \in \mathcal{A}} \hat{Q}(s'_i, a')$
      end
    end
    Calculate the loss $\mathcal{L} = 1/\mathrm{N} \sum_{i=0}^{N-1} (Q(s_i, a_i) - y_i)^2$
    Update Q using the SGD algorithm by minimizing the loss $\mathcal{L}$
    Every C steps, copy weights from Q to $\hat{Q}$
  end
end

برای بروزرسانی Q_{target} از soft update به شرح زیر استفاده شد:

$$\theta_{target} = \tau * \theta_{local} + (1 - \tau) * \theta_{target}$$

2.

نمودار پاداش در حین یادگیری عامل مطابق زیر می باشد:



همانگونه که مشاهده می شود عامل موفق به یادگیری محیط شده است.

3.

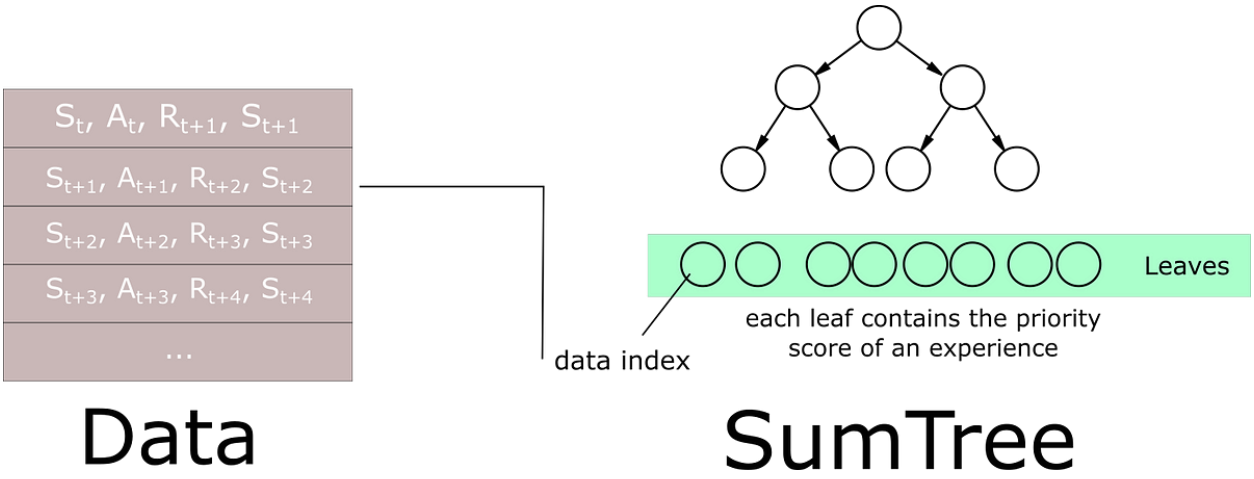
نحوه عملکرد عامل پس از یادگیری مطابق زیر ضبط شده است:

4.

- Buffer:
  - buffer_size = 10000
  - batch_size = 128
- Q-Network:
  - hidden_layers = [32, 64]
  - activation_func = ReLU
- DQN agent:
  - epsilon = 1
    - decay_rate = 0.995
    - min_epsilon = 0.01
  - learning_rate = 3e-4
  - gamma (discount factor) = 0.99
  - τ = 0.005
    - $\theta_{target} = \tau * \theta_{local} + (1 - \tau) * \theta_{target}$

بخش سوم

در ابتدای یادگیری تجربه های با عملکرد ضعیف بیشتر رخ می دهند و هنگامی که از بافر نمونه می گیریم به احتمال بیشتری در نمونه موجود و در نتیجه بیشتر آنها را یاد میگیریم. برای جبران این عدم توازن از Prioritized Experience Replay Buffer ها استفاده می کنیم.



i-th	Definition
Probability: ***P(i)***	$\frac{(P_i+\epsilon)^\alpha}{\sum_{j=0}^{N} (P_j+\epsilon)^\alpha}$
Weight: w_i	$\frac{(\frac{1}{N}\times\frac{1}{P(i)})^\beta}{(\frac{1}{N}\times\frac{1}{min_j P(i)})^\beta} \rightarrow (\frac{min_j(p_j)}{P(i)})^\beta$

- α: Prioritized parameter. 0 means uniform sampling.
- β: Compensation parameter. 1 means fully compensate (i.e. Importance Sampling)
- ϵ: Small value to avoid priority.

یکی دیگر از بهبود های الگوریتم DQN الگوریتم Dueling DQN است.
در این الگوریتم Q-Value مطابق زیر بازنویسی می شود:

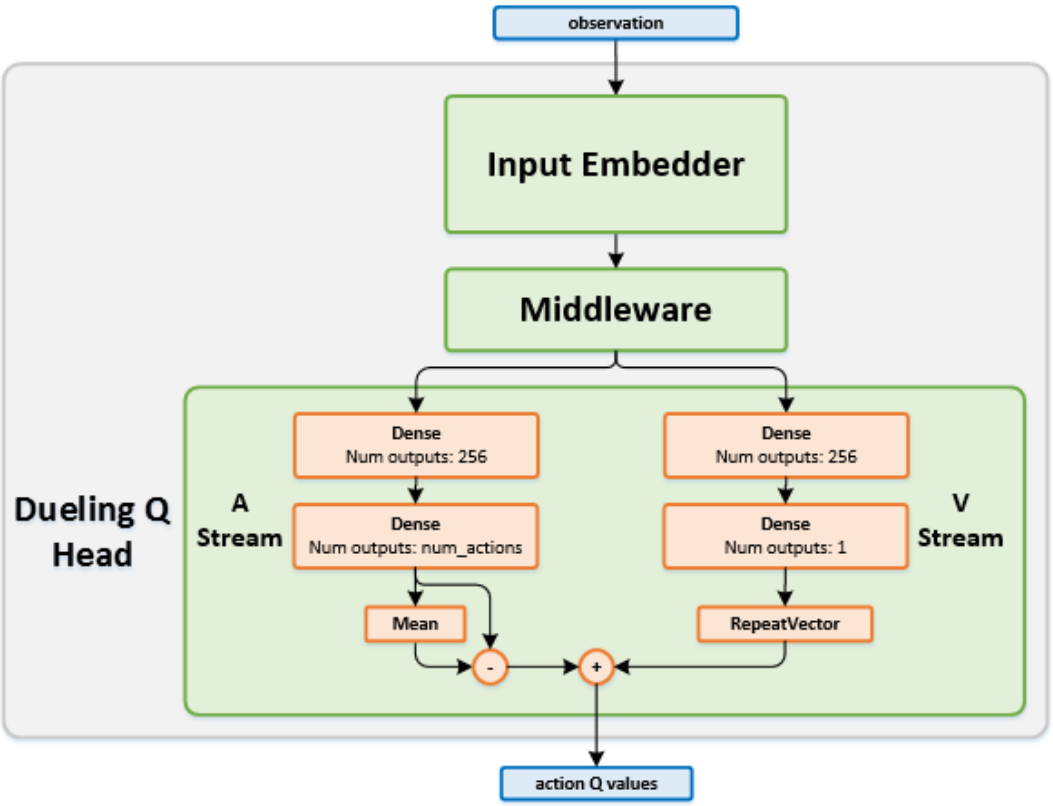
$$Q(s,a) = V(s) + (A(s,a) - \frac{1}{|A|}\sum_{a} \quad A(s,a))$$

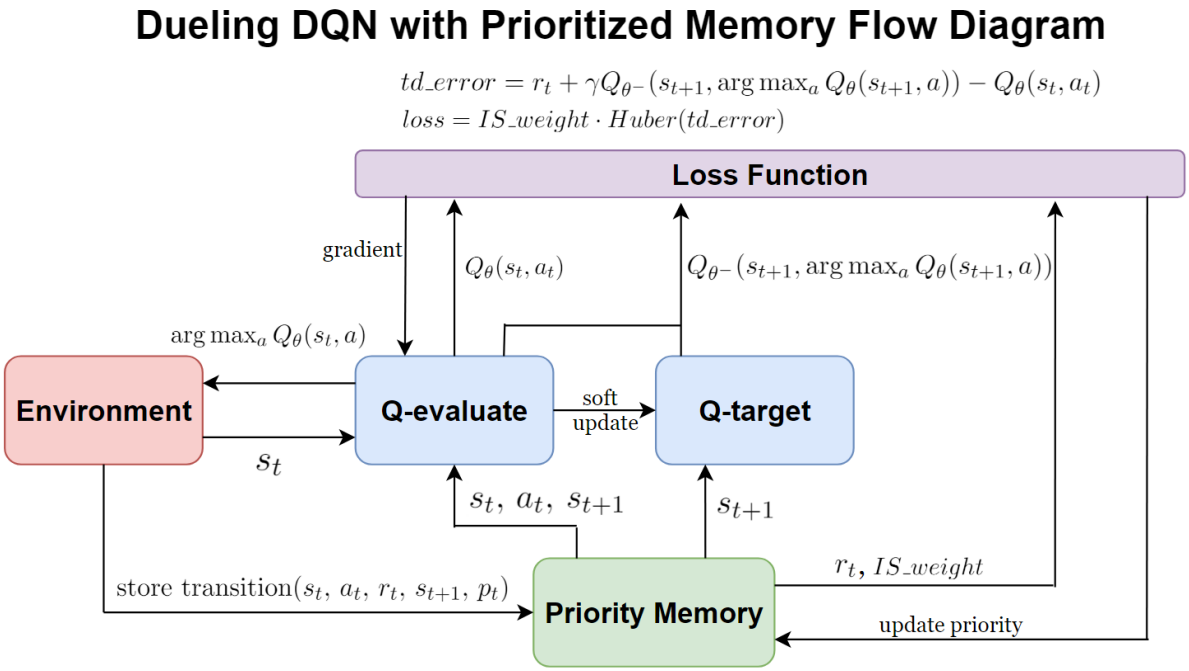
شبه کد این الگوریتم به شرح زیر است:

Algorithm 1 Deep Q Network with Prioritized Experience Replay

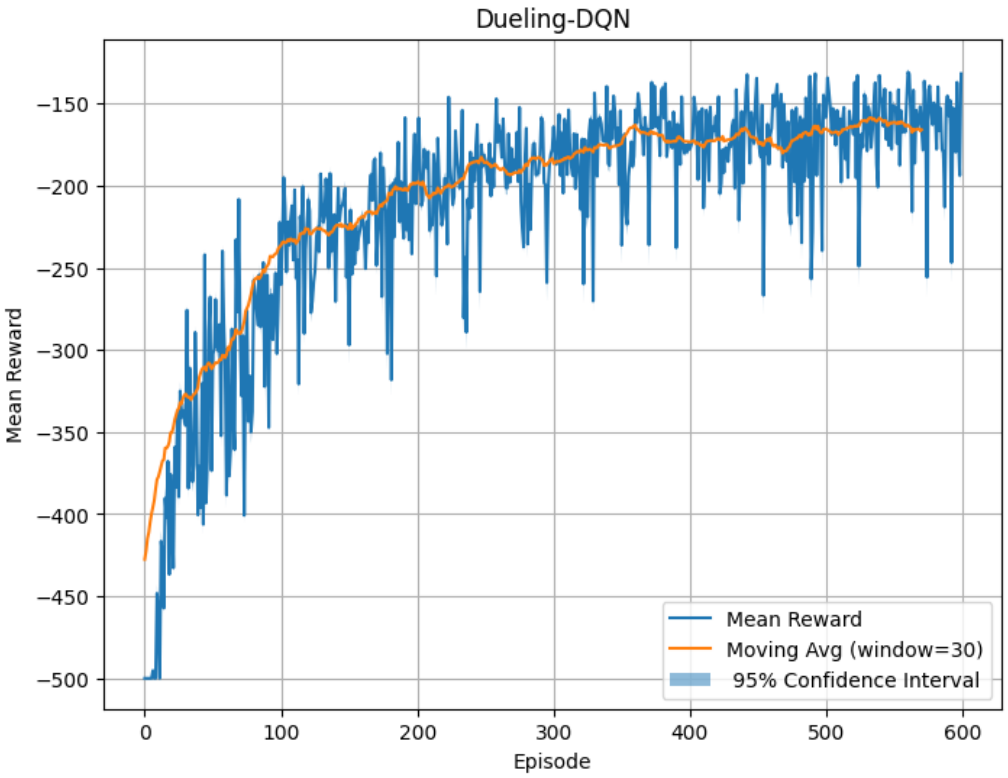
Initialize prioritized replay memory R to capacity N, minibatch size M, replay interval K
Initialize play time T, episode E
Initialize learning rate η, reward decay γ, interpolation parameter τ, epsilon-greedy parameter ϵ
Initialize exponents α, β, small value ζ
Initialize action-value function Q, evaluate net weight θ, target net weight θ^-
for $episode = 1$ to E **do**
    Reset environment and agent to random initial state s_0
    for $t = 1$ to T **do**
        With probability ϵ select random action a_t, otherwise select $a_t = \arg\max_a Q_\theta(s_t, a)$
        Agent execute action a_t in environment and observe reward r_t and next state s_{t+1}
        Set $p_t = \max_{i<t} p_i$ and store transition$(s_t, a_t, r_t, s_{t+1}, p_t)$
        if $t \mod K = 0$ **then**
            for $j = 1$ to M **do**
                Sample $transition_j \propto P(j) = p_j^\alpha / \Sigma_i p_i^\alpha$
                Compute importance-sampling weight (IS_weight) $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$
                Set $y_i = \begin{cases} y_j = r_j & \text{for terminal state } s_{j+1} \\ y_j = r_j + \gamma Q_{\theta^-}(s_{j+1}, \arg\max_a Q_\theta(s_{j+1}, a)) & \text{for non-terminal state } s_{j+1} \end{cases}$
                Compute TD-error $\delta_j = y_j - Q_\theta(s_j, a_j)$ and update priority $p_j \leftarrow |\delta_j| + \zeta$
                Accumulate weight-change $\triangle \leftarrow \triangle + w_j \cdot \delta_j \cdot \nabla_\theta Q_\theta(s_j, a_j)$
            end for
            Update weights $\theta \leftarrow \theta + \eta \cdot \triangle$, and reset $\triangle$ to 0
            Soft update target net weight $\theta^- \leftarrow \tau \cdot \theta + (1 - \tau) \cdot \theta^-$
            Decrease ϵ.
        end if
        if s_t is not terminal state **then**
            Move on to next state $s_t \leftarrow s_{t+1}$
        end if
    end for
end for

معماری کلی شبکه نیز به شرح زیر است:





نتیجه یادگیری عامل با الگوریتم Dueling-DQN مطابق زیر می باشد:



شیب یادگیری عامل نسبت به الگوریتم DQN مقداری افزایش داشته است اما توجه داشته باشید که از تکنیک های بهبود الگوریتم ها به طور کلی نمی توان انتظار بهبود چشمگیر داشت.

عملکرد عامل Dueling-DQN در یک اپیزود مطابق زیر ضبط شده است:

منابع

Dueling DQN - EN. (n.d.). 위키독스

ChienTeLee. (n.d.). *GitHub - ChienTeLee/dueling_dqn_lunar_lander*. GitHub.

Cyoon. (n.d.-c). *GitHub - cyoon1729/deep-Q-networks: Implementations of algorithms from the Q-learning family. Implementations include: DQN, DDQN, Dueling DQN, PER+DQN, Noisy DQN, C51*. GitHub.