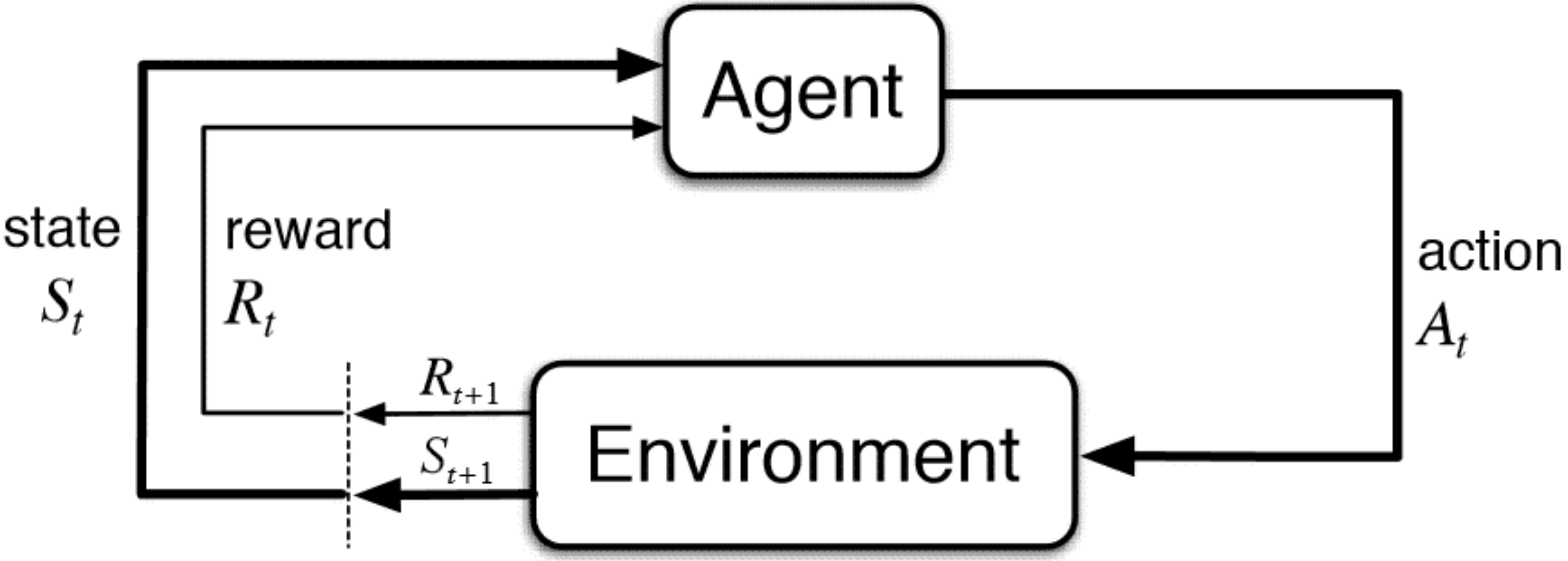
Deep Q-Learning

Mì AI

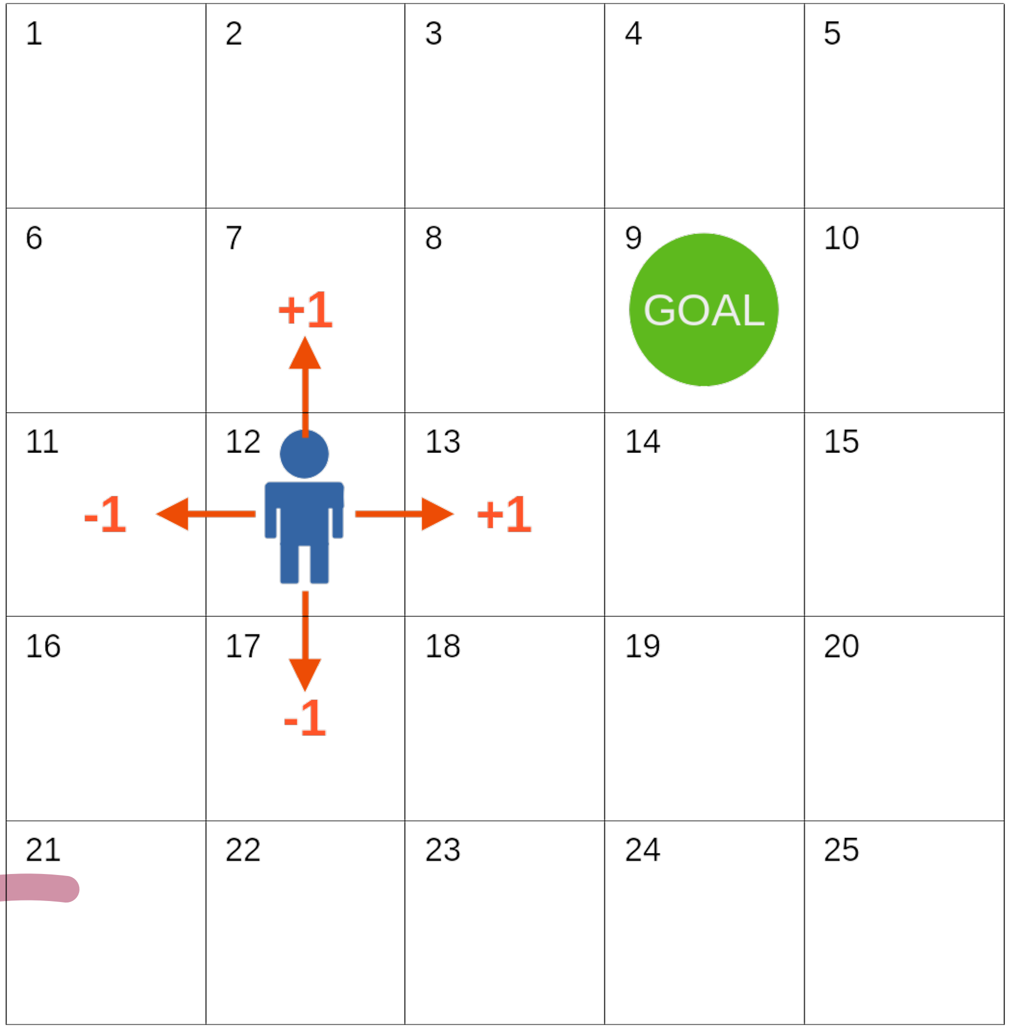
Q-Learning


Dạy máy tính chơi...
GAME!
Q-LEARNING
và bài toán đưa xe về đích
1:10:54
Tìm hiểu và dạy máy tính chơi game với Q - Learning - Mì AI
7.9K views • 1 year ago
Mì AI
17.2K subscribers
Xin chào các bạn, rất vui vì các bạn đã ghé thăm vlog Mì AI của tôi! Hãy join cùng cộng đồng Mì AI nhé! #MìAI Fanpage: ...

Q-Learning



Q-Learning



	↑	↓	←	→
1	-	+1	-	+1
2	-	+1	-1	+1
3	-	+1	-1	+1
4	-	+1	-1	-1
5	-	+1	+1	-
...				
23	+1	-	-1	+1
24	+1	-	-1	-1
25	+1	-	+1	-

Q-Learning

Actions

States

	A_1	A_2	...	A_M
S_1	$Q(S_1, A_1)$	$Q(S_1, A_2)$		$Q(S_1, A_M)$
S_2	$Q(S_2, A_1)$	$Q(S_2, A_2)$		$Q(S_2, A_M)$
⋮			⋱	⋮
S_N	$Q(S_N, A_1)$	$Q(S_N, A_2)$	...	$Q(S_N, A_M)$

Source: https://en.wikipedia.org/wiki/Q-learning

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\bigg(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \bigg)}^{\text{learned value}}$$

Q-Learning

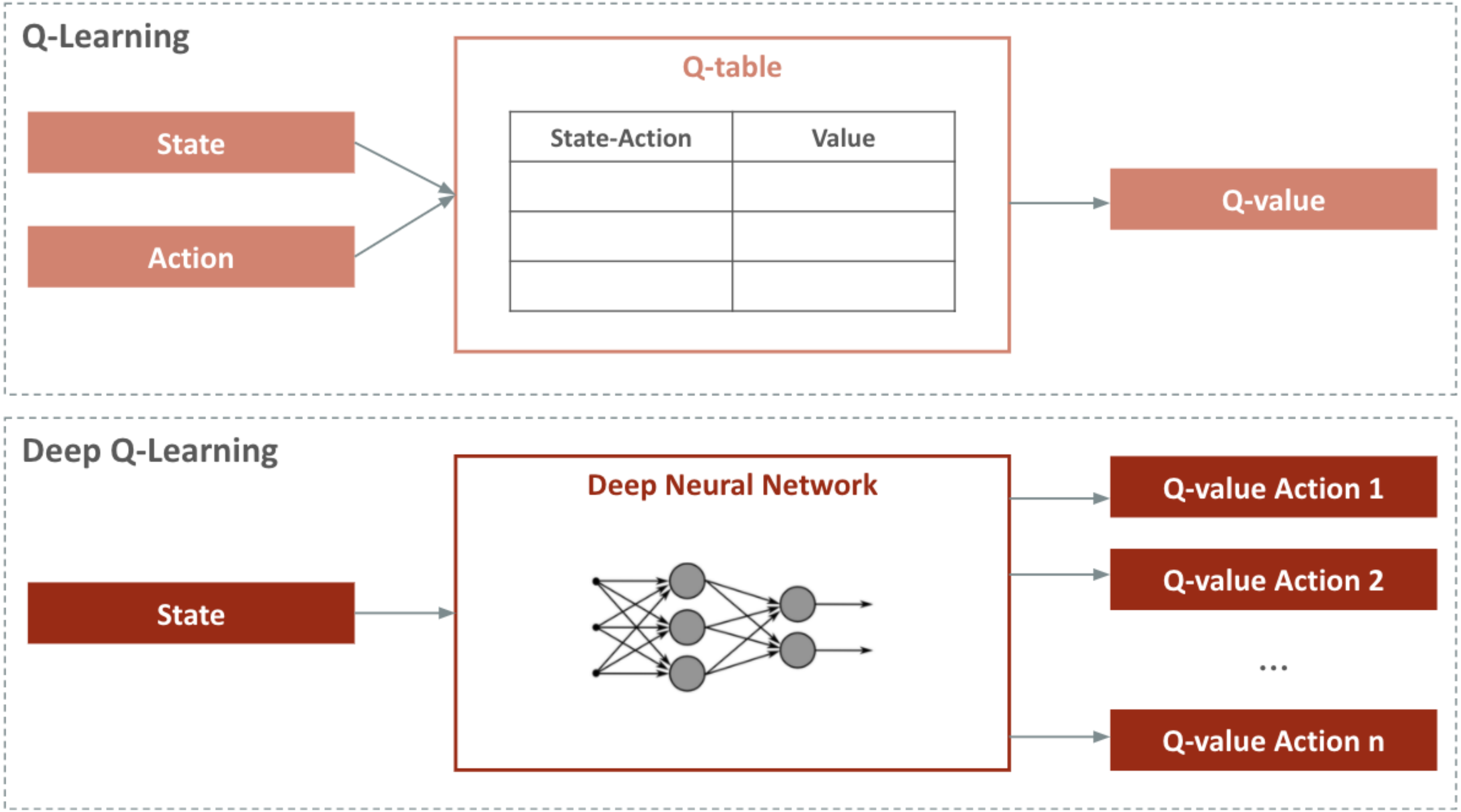
Game Board:

Current state (s): 0 0 0
0 1 0

Q Table: $\gamma = 0.95$

	0 0 0 1 0 0	0 0 0 0 1 0	0 0 0 0 0 1	1 0 0 0 0 0	0 1 0 0 0 0	0 0 1 0 0 0
↑	0.2	0.3	1.0	-0.22	-0.3	0.0
↓	-0.5	-0.4	-0.2	-0.04	-0.02	0.0
→	0.21	0.4	-0.3	0.5	1.0	0.0
←	-0.6	-0.1	-0.1	-0.31	-0.01	0.0

Deep Q-Learning



Why DQN?

- In an environment with a continuous state space it is impossible to go through all the possible states and actions repeatedly, since there are an infinite number of them and the Q-Table would be too big.
- DQN solves this problem by approximating the Q-Function through a Neural Network and learning from previous training experiences, so that the agent can learn more times from experiences already lived without the need to live them again, as well as avoiding the excessive computational cost of calculating and updating the Q-Table for continuous state spaces.

DQN Components

Main Neural Network

The Main NN tries to predict the expected return of taking each action for the given state.

Train and update every episodes

DQN Components

Target Neural Network

The Target Neural Network is used to get the target value for calculating the loss and optimizing it.

Will be updated every N timesteps with the weights of the main network.

DQN Components

Replay Buffer

The Replay Buffer is a list that is filled with the experiences lived by the agent.

An experience is represented by the **current state**, the **action taken in the current state**, the **reward** obtained after taking that action, whether it is a **terminal state** or not, and the **next state reached** after taking the action.

DQN Components

- State size
- Action size
- Gamma
- Episode
- Number of steps
- Epsilon value, epsilon decay
- Learning rate
- Target NN update rate

Action	*Action Number*
0	Push cart left
1	Push cart right

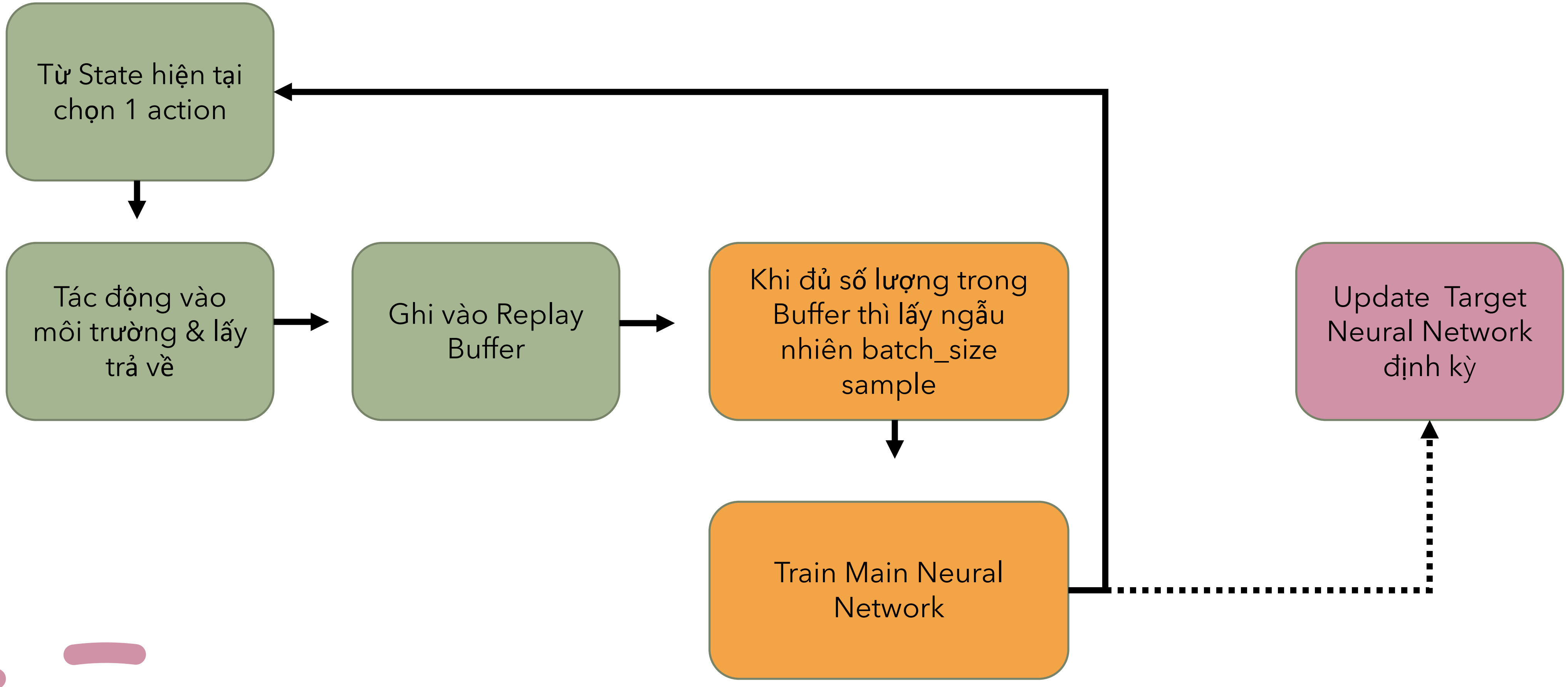
Index In array	*Meaning*	*Min Value*	*Max value*
0	Cart Position on x axis	-4.8	4.8
1	Cart Velocity on x axis	$-\infty$	∞
2	Pole Angle	-0.418 rad	0.418 rad
3	Pole Angular Velocity	$-\infty$	∞

DQN Flow

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

DQN Flow


Từ State hiện tại chọn 1 action
Tác động vào môi trường & lấy trả về
Ghi vào Replay Buffer
Khi đủ số lượng trong Buffer thì lấy ngẫu nhiên batch_size sample
Train Main Neural Network
Update Target Neural Network định kỳ

Select Action

Epsilon Greedy Policy

$$1.\ \textit{Generate random number } x \textit{ between } 0 \textit{ and } 1$$

$$2.\ action = \begin{cases} random\ action & \text{if } x < \varepsilon \\ action\ with\ best\ Q-Value & \text{if } x \geqslant \varepsilon \end{cases}$$

Train Main NN

$$Loss = \Big(\underbrace{r}_{reward} + \gamma * \underbrace{\max_{a'} Q(s', a')}_{\substack{\text{Target Network output} \\ \text{for next state and action}}} - \underbrace{Q(s, a)}_{\substack{\text{Main Network output} \\ \text{for current state and action}}} \Big)^2$$

$$\text{Set } y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$$

Let’s go!