Journal of Automation and Intelligence xxx (xxxx) xxx

Contents lists available at ScienceDirect

Journal of Automation and Intelligence

journal homepage: www.keaipublishing.com/en/journals/journal-of-automation-and-intelligence/

# Sampled-data control through model-free reinforcement learning with effective experience replay

Bo Xiao [a,*], H.K. Lam [b], Xiaojie Su [c], Ziwei Wang [d], Frank P.-W. Lo [e], Shihong Chen [a], Eric Yeatman [a]

[a] *Department of Electrical and Electronic Engineering, Imperial College London, London SW7 2AZ, UK*
[b] *Department of Engineering, King's College London, London WC2B 4BG, UK*
[c] *Department of Automation, Chongqing University, Shapingba District, Chong Qing, China*
[d] *School of Engineering, Lancaster University, LA1 4YW, UK*
[e] *Hamlyn Centre, Imperial College London, London SW7 2AZ, UK*

ARTICLE INFO

*Keywords:*
Reinforcement learning
Neural networks
Sampled-data control
Model-free
Effective experience replay

ABSTRACT

Reinforcement Learning (RL) based control algorithms can learn the control strategies for nonlinear and uncertain environment during interacting with it. Guided by the rewards generated by environment, a RL agent can learn the control strategy directly in a model-free way instead of investigating the dynamic model of the environment. In the paper, we propose the sampled-data RL control strategy to reduce the computational demand. In the sampled-data control strategy, the whole control system is of a hybrid structure, in which the plant is of continuous structure while the controller (RL agent) adopts a discrete structure. Given that the continuous states of the plant will be the input of the agent, the state–action value function is approximated by the fully connected feed-forward neural networks (FCFFNN). Instead of learning the controller at every step during the interaction with the environment, the learning and acting stages are decoupled to learn the control strategy more effectively through experience replay. In the acting stage, the most effective experience obtained during the interaction with the environment will be stored and during the learning stage, the stored experience will be replayed to customized times, which helps enhance the experience replay process.

The effectiveness of proposed approach will be verified by simulation examples.

## 1. Introduction

Reinforcement learning (RL) algorithms generally contain three components, namely, agents, environment and reward functions. The environment of RL is generally represented by Markov decision processes (MDPs) framework. Through interacting with the environment, the agent can lean the optimal behaviour (action picking) which maximizes the obtained rewards from the environment [1]. RL can be also treated as the semi-supervised learning approach since the agent learns from delayed rewards without a supervisor to guide its behaviour at every step [2]. Well-learned RL agent is expected to pick the best action according to the value function at different observed states of the environment. From the optimal control perceptive, RL agent picking the best action during the interaction with environment can be regarded as the controller generating input to control the plant optimally.

During the decades, RL has been successfully applied to solve the control problems, but most of those works such as [3–11] adopted RL techniques to regulate the control system by solving the Hamilton–Jacobi–Bellman (HJB) equation, which demands theoretical understanding and investigation of the optimal control system. In contrast, recently, the model-free reinforcement learning has been adopted in intelligent control design by only interacting with the environment without solving the HJB equation [1,12–16]. Besides, there are also some work utilizing reinforcement learning for control design in real world can be found in [17–20] and the references therein. Q-learning, as one of the most widely applied off-policy RL algorithm, is able to find the solution to the nonlinear and uncertain environment through interacting with environment [1,14,21]. However, the basic structure of Q-learning requires the finite discrete states and discrete control inputs to develop the state–action value table (Q-table), which is the exact value function of the states and actions. This requirement restricts the applications of Q-learning to the infinite continuous state case since it is not possible to calculate a Q-table with infinity dimensions numerically. An alternative solution to this problem is to use the

* Corresponding author.
*E-mail addresses:* b.xiao@imperial.ac.uk (B. Xiao), hak-keung.lam@kcl.ac.uk (H.K. Lam), suxiaojie@cqu.edu.cn (X. Su), z.wang82@lancaster.ac.uk (Z. Wang), po.lo15@imperial.ac.uk (F.P.-W. Lo), shihong.chen19@imperial.ac.uk (S. Chen), e.yeatman@imperial.ac.uk (E. Yeatman).

https://doi.org/10.1016/j.jai.2023.100018
Received 19 December 2022; Received in revised form 20 January 2023; Accepted 24 January 2023

function approximator to approximate the state–action values. In this approach, the Q-table will be represented by the approximation function which can receive the continuous states as its input. One of the most popular choices of the approximation function is neural network (NN) thanks to its global approximation of continuous functions at any accuracy [22,23]. When the state–action value function is approximated by NN, the value function is represented by the Q-network and RL will turn out to be approximate reinforcement learning (ARL). In ARL, the states of the environment and control input will be the input of the Q-network while the estimated state–action value will be the output of the Q-network. During the control process, the Q-network is used to guide the agent to pick the control input according to the approximated state–action value function.

When the state–action values are approximated by a nonlinear function like NN, the reinforcement learning process is usually unstable and the Q-network is apt to blow out during the training process. The techniques of target network and experience replay are generally used to stabilize the training process of the NN [13,14,24]. It is well-known that the state–action value will update itself by the current reward and the largest state–action value of the next step. When the target network is adopted, the largest state–action value of the next step is calculated through the target network instead of the Q-network itself. After a certain times of state–action value updates of Q-network, the target network will be replaced by the Q-network. The replacement can also be soft, in [14], the soft update of the target network was reported, in which only partial of the target network is updated by Q-network, which further smooths the training process. The technique of experience replay was firstly reported in [25] for RL approximated by NN. The main idea of experience replay technique is to store the state transitions, control input and rewards as the experience in the experience buffer (EB). Through experience replay, the experience samples will be randomly shuffled and re-sampled out as mini-batch to train the nonlinear approximator, thus the correlations among the sequential data can be removed and the sample efficiency can be improved. The technique of experience replay is essential in the control of continuous plant, where the sequential state transitions are highly related.

In the works in most of the literatures such as [13,14], the learning and acting are integrated together, the update of state–action values takes place at every step and all the recent transitions are recorded in the EB. However, for the episodic cases, it might not be the best idea to train the nonlinear approximator once a step. It helps improve the training process by customized times of experience replay since more freedom has been introduced. On the other hand, the dynamic model of environment for the RL agent sometimes is valid only for the specific operation domain. However, due to the exploration of Q-learning and non-optimal actions from under-learnt agent, it is unavoidable for the controller to drive the system into some undesired states. In this case, it is not efficient to store all the transitions during simulation as experience. Furthermore, the states far away from the operation domain generally do not make sense in the real applications. When NN is used to approximate the value function, those outliers rise the unnecessary computational cost and make the approximating NN even more difficult to converge through gradient descend. To address the aforementioned issues, we separate the acting and learning into two independent stages for every control episode. In the acting stage, the agent is only to drive the transitions in system to experience different states, when the states are too far away from the valid operation domain, the experience will be discarded and will not be stored in the EB. During the acting stage, the current state, next state, control input and reward will be recorded into the EB. During the learning stage, the Q-network will be updated through the experience recorded during the acting process. Given that the learning and acting stages are decoupled, the number of experience replay times for one episode can be chosen freely, which helps improve the training efficiency. This claim will be verified by the simulation examples.

When the digital controller is considered to be applied to control the continuous plant, the whole closed-loop control system turns out to be a hybrid system, where the plant is continuous while the controller is discrete. In this case, sampled-data control strategy can be a good candidate to reduce the control cost and make the digital control possible [26]. Due to zero-order-hold (ZOH) process, the control input from the sampled-data controller is of the form of staircase signal. To deal with the hybrid dynamics in the sampled-data control systems, in the context of intelligent control, the input-delay approach [27], equivalent jump system [28], discretization approach [29], digital redesign [30,31] and the exact discrete-time design approach [32] can be used for stability analysis and control design. In addition, as mentioned above, the RL agent based on Q-learning needs to check the state–action values for all the possible actions for the current state before it can pick out the action. In the sampled-data control strategy, the control input generated by the RL agent will be held for a period of time, thus the frequency of action picking can be reduced during the control process. Therefore, the computational demand of the controller application can be reduced significantly especially when the size of Q-network and action set are huge. In this paper, the sampled-data control problem will be reformed into MDPs and the RL agent will be trained to fulfil the control objectives by solving the MDPs through interacting with the environment. The contribution and novelty of the paper can be summarized as follows:

1. The control problem of continuous plant is formed as MDPs and RL algorithm is proposed and improved to train the RL controller purely from interactions with the environment without investigating system dynamic.
2. Acting and learning stages in reinforcement learning are decoupled in the proposed algorithm to have more freedom to the experience replay and help enhance the experience replay process.
3. The most effective experience is stored in the EB, which saves the buffer memory and facilitates the learning process.
4. Sampled-data control strategy is proposed to control the continuous plant using discrete controller, which benefits the digital application and helps reduce the computational demand of the control applications.

The rest of the paper is organized as follows: In Section 2, the preliminaries of MDPs, sampled-data control system, Q-learning, adaptive $\epsilon$-greedy policy, ARL will be introduced. In Section 3, the RL algorithm of the sampled-data control strategy will be discussed in details. In Section 4, two simulation examples are presented to verify the proposed control strategy. In Section 5, a conclusion is drawn.

## 2. Preliminaries

In this section, the MDPs, sampled-data control system, Q-learning, adaptive $\epsilon$-greedy policy, ARL will be introduced.

### 2.1. Markov decision processes

An MDP is a mathematical framework based on the state set $\mathcal{X}$ and the related finite action set $\mathcal{U}$. If and only if the state $\mathbf{x}(t)$ in $\mathcal{X}$ captures all the relative information from history, the state $\mathbf{x}(t)$ is Markov. The transition function between two Markov states can be defined as: $f : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$. In addition, an Markov reward process (MRP) is defined as the tuple $\langle \mathcal{X}, f, r, \gamma \rangle$, where $r$ is the scalar reward function and $\gamma$ is the discount factor. The reward function is defined as $r : \mathcal{X} \times \mathcal{U} \times \mathcal{X} \to \mathbb{R}$. $\gamma \in [0, 1]$ determines how we value the importance of current rewards and future rewards. $\gamma$ also stabilizes the total return in infinite timestep cases. The MRP with control input/action is defined as an MDP: $\langle \mathcal{X}, \mathcal{U}, f, r, \gamma \rangle$. The objective of RL agent is to find the optimal control policy which maximizes the accumulated rewards (return) $R$. Considering the observed states $\mathbf{x}(t) \in \mathcal{X}$ and admissible control input $\mathbf{u}(t) \in \mathcal{U}$, the following sampled-data control system can be formed into MDPs. To evaluate the value function, the RL agent will try to solve the MDP. When the value function is obtained, the RL agent can pick the control input according to it, thus to fulfil the control objective.

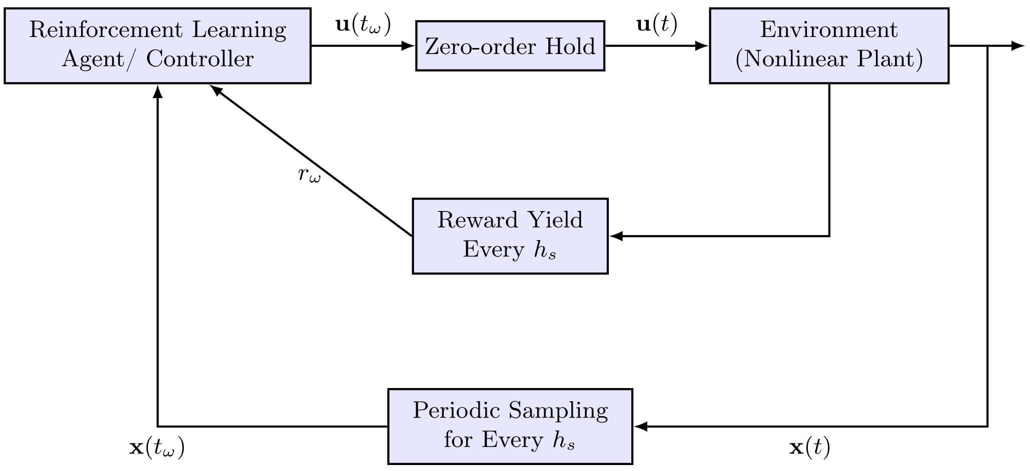

**Fig. 1.** Diagram of the sampled-data based control system.

### 2.2. Sampled-data control system

A sampled-data control system is generally formed by four components, namely the continuous nonlinear plant, RL agent (controller), ZOH and the sampling part. The structure of sampled-data control system can be viewed in Fig. 1. In the figure, the sampled-data control design is combined the RL algorithm to better elaborate the mechanism of the control system.

It is noteworthy that only the sampled states of the nonlinear plant are observed by the RL controller to generate the sampled control input and the sampled control input will be held by the ZOH to control the continuous nonlinear plant. The dynamics of the sampled-data control system can be described as following:

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathbf{A}(\mathbf{x}(t))\mathbf{x}(t) + \mathbf{B}(\mathbf{x}(t))\mathbf{u}(t) \\ \mathbf{u}(t) = \mathbf{u}(t_\omega), \quad \forall t,\ t_\omega < t \le t_{\omega+1}, \end{cases} \tag{1}$$

where $\mathbf{x}(t) \in \mathcal{X} \subset \mathbb{R}^n$ is the state variable of the nonlinear plant, $\mathbf{u}(t) \in \mathcal{U} \subset \mathbb{R}^m$ is the sampled control input. $\mathbf{A}(\mathbf{x}(t)) \in \mathbb{R}^{n\times n}$ is the nonlinear system matrix while $\mathbf{B}(\mathbf{x}(t)) \in \mathbb{R}^{n\times m}$ is the nonlinear input matrix. The sampled-data RL controller is adopted to fulfil the control objective, which takes the sampled state of the nonlinear plant $\mathbf{x}(t_\omega)$ as input, and outputs $\mathbf{u}(t_\omega)$ as the control input. $t_\omega = t - \tau(t)$, for $t_\omega < t \le t_{\omega+1}$, $0 < \tau(t) < h_s$, where $h_s$ is the sampling interval.

From (1), the next sampled state $\mathbf{x}(t_{\omega+1})$ of the continuous system can be calculated as:

$$\mathbf{x}(t_{\omega+1}) = \int_{t_\omega}^{t_{\omega+1}} \mathbf{A}(\mathbf{x}(t))\mathbf{x}(t) + \mathbf{B}(\mathbf{x}(t))\mathbf{u}(t_\omega)dt. \tag{2}$$

In (2), it can be found that $\mathbf{x}(t_{\omega+1})$ depends only on the sampled state $\mathbf{x}(t_\omega)$ and $\mathbf{u}(t_\omega)$ instead of the full history of $\mathbf{x}(t)$, which makes $\mathbf{x}(t_\omega)$ Markov. Associated with proper designed discount factor $\gamma$ and reward function $r$, the sampled-data control problem can be built as MDPs problem.

**Remark 1.** The dynamic model is used as the environment and only used to generate the state transitions and rewards. The agent learns to control the nonlinear plant directly through interactions with the dynamic model instead of investigating the dynamic model.

**Notation.** To make concise of notations, in those cases without ambiguity, the sampled states and control input $\mathbf{x}(t_\omega)$, $\mathbf{x}(t_{\omega+1})$ and $\mathbf{u}(t_\omega)$ will be written as $\mathbf{x}_\omega$, $\mathbf{x}_{\omega+1}$ and $\mathbf{u}_\omega$, respectively.

### 2.3. Q-learning

In RL, the behaviour of the RL agent is determined by the control policy defined as $\pi(\mathbf{u}|\mathbf{x}) = P(\mathbf{u}_\omega = \mathbf{u}|\mathbf{x}_\omega = \mathbf{x})$. The policy $\pi(\mathbf{u}|\mathbf{x})$ represents the probability distribution of the control input $\mathbf{u}_\omega$ according to the observed system state $\mathbf{x}_\omega$. $r_\omega$ is the reward obtained after yielding the control input during the $\omega$-th timestep. As shown in Fig. 1, the RL agent takes the observed state $\mathbf{x}_\omega$ as input and gives out the control input $\mathbf{u}_\omega$, in the same time, receives the reward $r_\omega$ at timestep $\omega$. With the reward function, the state–action value $Q(\mathbf{x}_\omega, \mathbf{u}_\omega)$ in RL is defined as the expectation of return $R_\omega$ which starting with state $\mathbf{x}_\omega$ and control input $\mathbf{u}_\omega$. The state–action value function under policy $\pi$ can be calculated as following:

$$\begin{aligned} Q^\pi(\mathbf{x}, \mathbf{u}) &= \mathbb{E}_\pi[R_\omega | \mathbf{x}_\omega = \mathbf{x}, \mathbf{u}_\omega = \mathbf{u}] \\ &= \mathbb{E}_\pi[r_\omega + \gamma r_{\omega+1} + \gamma^2 r_{\omega+2} + \cdots | \mathbf{x}_\omega = \mathbf{x}, \mathbf{u}_\omega = \mathbf{u}] \\ &= \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r_{\omega+k} | \mathbf{x}_\omega = \mathbf{x}, \mathbf{u}_\omega = \mathbf{u}] \end{aligned} \tag{3}$$

and the state–action value function can be also rewritten as the Bellman equation (BE):

$$Q^\pi(\mathbf{x}_\omega, \mathbf{u}_\omega) = \mathbb{E}_\pi[r_\omega + \gamma \mathbb{E}_\pi[Q^\pi(\mathbf{x}_{\omega+1}, \mathbf{u}_{\omega+1})]]. \tag{4}$$

The goal of Q-learning is to find the optimal control policy $\pi$, which maximizes the state–action value function. The optimal state–action value function can be viewed as following:

$$\begin{aligned} Q^*(\mathbf{x}, \mathbf{u}) = \max_\pi \mathbb{E}_\pi[&r_\omega + \gamma r_{\omega+1} + \gamma^2 r_{\omega+2} \\ &+ \ldots | \mathbf{x}_\omega = \mathbf{x}, \mathbf{u}_\omega = \mathbf{u}]. \end{aligned} \tag{5}$$

To maximize the return obtained by the RL agent, the Q-learning algorithm updates the state–action values according to the reward function in an off-policy style as:

$$\begin{aligned} Q(\mathbf{x}_\omega, \mathbf{u}_\omega) \leftarrow &(1-\alpha)Q(\mathbf{x}_\omega, \mathbf{u}_\omega) \\ &+ \alpha(r_\omega + \gamma \max_{\mathbf{u}_k}\{Q(\mathbf{x}_{\omega+1}, \mathbf{u}_k)\}), \end{aligned} \tag{6}$$

where $\alpha \in (0, 1]$ is the learning rate, which determines the learning speed of Q-learning.

To ensure the Q-learning explores enough along without loss of convergence, the adaptive $\epsilon$-greedy policy is adopted in this paper. To adaptively change the value of the exploration rate $\epsilon$, we start from large value of $\epsilon$ and then reduce it gradually after episodes being completed.

$$\mathbf{u}_\omega = \begin{cases} \arg\max_{\mathbf{u}_k}\{Q(\mathbf{x}_\omega, \mathbf{u}_k)\}, \text{ with probability: } 1-\epsilon, \\ \text{random } \mathbf{u}_k, \text{ with probability: } \epsilon. \end{cases} \tag{7}$$

The Q-learning algorithm is summarized in Algorithm 1.

**Algorithm 1** Algorithm for Q-learning

```
1: Set the state-action values randomly
2: for Every episode do
3:     Initialise state $\mathbf{x}_0$
4:     for Current timestep $\omega$ do
5:         Pick action $\mathbf{u}_\omega$ through $\epsilon$-greedy policy and observe the next state $\mathbf{x}_{\omega+1}$
6:         Update $Q(\mathbf{x}_\omega, \mathbf{u}_\omega) \leftarrow (1-\alpha)Q(\mathbf{x}_\omega, \mathbf{u}_\omega) + \alpha(r_\omega + \gamma \max_{\mathbf{u}_k}\{Q(\mathbf{x}_{\omega+1}, \mathbf{u}_k)\})$
7:         $\mathbf{x}_\omega \leftarrow \mathbf{x}_{\omega+1}$
8:     end for
9: end for
```

### 2.4. Approximate reinforcement learning

When the state set $\mathcal{X}$ and action set $\mathcal{U}$ are too large, calculation of the exact state–action value function $Q(\mathbf{x}, \mathbf{u})$ is difficult and might be impossible in many cases. One alternative is using function approximator to approximate the exact state–action value function. As an universal approximator, a 3-layer fully connected feed-forward neural network (FCFFNN) can be used with nonlinear activation functions for the Q-network to approximate $Q(\mathbf{x}, \mathbf{u})$ [23]. When the value function is approximated, RL will become ARL. The structure of the 3-layer NN used to represent the state–action value function can be viewed in Fig. 2.

In Fig. 2, the state $\mathbf{x} = [x_1, x_2, \dots, x_n]^T$ and action $\mathbf{u} = [u_1, u_2, \dots, u_m]^T$ are the input of the NN while the estimated state–action value function $\hat{Q}(\mathbf{x}, \mathbf{u})$ is the output of the NN. $af_1(.)$ is the activation function for the hidden layer, which is generally a nonlinear function to grant nonlinear approximation ability. $af_2(.)$ is the activation function for the output layer, which can be a linear function. $w_{11}^{(1)}, \dots, w_{h,n+m}^{(1)}, b_1^{(1)}, \dots, b_h^{(1)}$ are the weights and biases between the input and hidden layers. The number of parameters between the input and hidden layers is easy to calculate and it is $h\times(n+m+1)$. In the same way, $w_{11}^{(2)}, \dots, w_{1h}^{(2)}, b^{(2)}$ are the weights and bias between the hidden and output layers. The number of parameters between the hidden layer and output layers is $h+1$ and the total number of parameters in the whole network is $h \times (n+m+2)+1$.

The output of the NN is calculated in a feed-forward way. The output of the $j$th hidden node, $j = 1, \dots, h$, can be calculated as:

$$g_j = af_1(\sum_{i=1}^{n} x_i w_{ji}^{(1)} + \sum_{i=1}^{m} u_i w_{j,n+i}^{(1)} + b_j^{(1)}), \tag{8}$$

and the estimated state–action value, which is the final output of the NN after is calculated as:

$$\hat{Q}(\mathbf{x}, \mathbf{u}) = af_2(\sum_{i=1}^{h} g_i w_{1i}^{(2)} + b^{(2)}). \tag{9}$$

Along with the Q-network approximating the value function, the target network in the same structure with the Q-network. To lighten the notation burden, the set of whole parameters of the Q-network will be written as $\theta^Q$ and the set of whole parameters of the target network will be as $\theta^{Q'}$ in the following context. The updated approximation of the state–action value for the current state $\mathbf{x}_\omega$ and control input $\mathbf{u}_\omega$ is written as $\hat{Q}(\mathbf{x}_\omega, \mathbf{u}_\omega|\theta^Q)$. The Q-network implicitly determines the control policy, in which the control input $\mathbf{u}_k$ is generated according to the observation $\mathbf{x}_\omega$. The sampled control input will be held for a period of the sample interval and drives the current sampled state $\mathbf{x}_\omega$ to the next sampled state $\mathbf{x}_{\omega+1}$, also reward $r_\omega$ will be received from the environment according to the reward function. Then the target network takes the state $\mathbf{x}_{\omega+1}$ and the control input set $\mathcal{U}$ as the input to calculate the output $\max_{\mathbf{u}_k}\{\hat{Q}(\mathbf{x}_{\omega+1}, \mathbf{u}_\omega|\theta^{Q'})\}$ to estimate largest state–action value for the next state $\mathbf{x}_{\omega+1}$. When $\max_{\mathbf{u}_k}\{\hat{Q}(\mathbf{x}_{\omega+1}, \mathbf{u}_\omega|\theta^{Q'})\}$ and $r_\omega$ are available, the state–action value $\hat{Q}(\mathbf{x}_\omega, \mathbf{u}_\omega|\theta^Q)$ of the current state–action pair can be updated according to the target network and obtained reward. The algorithm for updating of the weights of the NN is presented in detail in the next section.

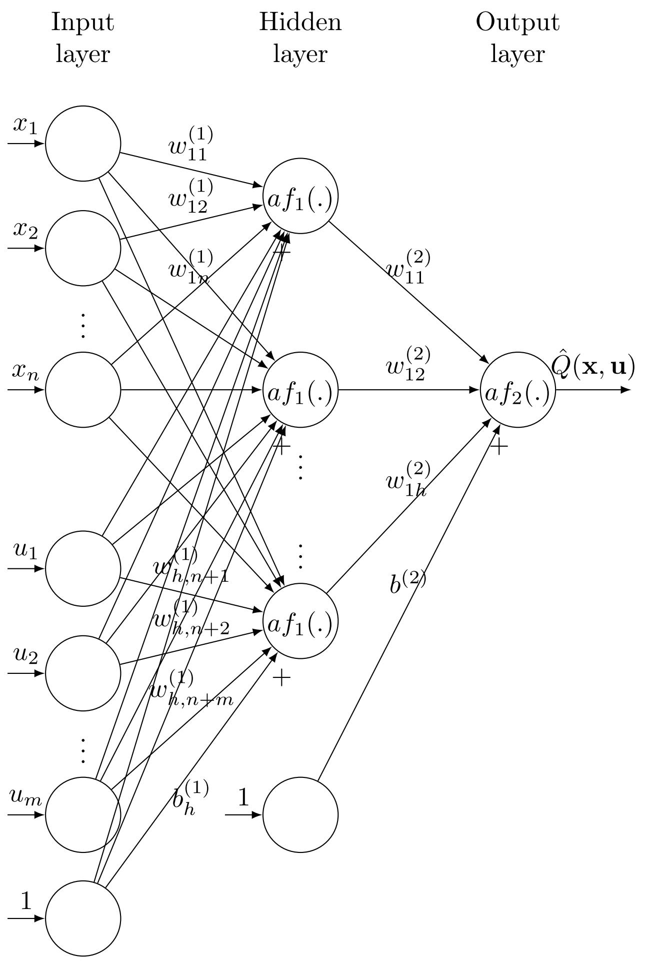

**Fig. 2.** Structure of the 3-layer fully connected feed-forward neural network.

## 3. Control strategy learning algorithm

The update of weights of the target network and Q-network through interaction with the environment is summarized in Algorithm 2 and the algorithm is divided into two parts. The first part of the algorithm is the acting stage. In the acting stage, the agent will follow the current control policy indicated by the current Q-network to generate control input. The control input drives the current sample state $\mathbf{x}_\omega$ to transit to the next sample state $\mathbf{x}_{\omega+1}$. Unlike the traditional way in [12–14], in which the agent learns after every state transition, there is no learning process takes place in the acting stage. In the acting stage, the requirements on current and next states will be established to determine whether the state transition will be stored as part of the experience. When the dynamic model of the environment is available, the state transition along with the control input and reward given by the environment will be recorded into the EB only if the current state and the next state both satisfy the requirements.

In the learning stage, the weights of target network and Q-network will be updated through the effective experience replay. Since the acting and learning phases are decoupled, it is possible to perform a customized number of experience replay at the learning stage. The target network is helpful to adaptively update and stabilized the weights $\theta^Q$ in the Q-network . The weights set $\theta^{Q'}$ of the target network will be frozen to constants for $T_Q$ times of updating of the Q-network. The weights set $\theta^{Q'}$ will be updated by $\theta^{Q'}$ and $\theta^Q$ softly after $T_Q$ times updating of $\theta^Q$.

**Algorithm 2** Algorithm for the effective experience replay

```
1: Choose the same structure for the target network and Q-network
2: Randomly initialize the weights of target network and Q-network by the same parameters $\theta^{Q_0}$:
       $\theta^{Q'} \leftarrow \theta^{Q_0}, \theta^{Q} \leftarrow \theta^{Q_0}$
3: Initialize the experience buffer (EB)
4: Initialize the control input set $\mathbf{u}$
5: Choose the constants: Max_Episode, Max_Time_Step, Max_Replay_Times, $T_Q$, $N$, $\tau$
6: Initialize the counter $C_Q = 0$
7: for Episode = 1:Max_Episode do
8:     Initialize $\epsilon$ for the $\epsilon$-greedy exploration
9:     Initialize $\alpha^Q$ for adaptive learning
10:    Generate control input through the $\epsilon$-greedy policy determined by the current Q-network
11:    Acting Stage:
12:    for t = 1:Max_Time_Step do
13:        Generate control input $\mathbf{u}_t$ through the $\epsilon$-greedy policy determined by the current Q-network
14:        Hold control input for a period of time to observe the reward $r_t$ and new state $\mathbf{x}_{t+1}$
15:        if $\mathbf{x}_t$ and $\mathbf{x}_{t+1}$ both satisfy the requirements then
16:            Store transition $(\mathbf{x}_t, \mathbf{u}_t, r_t, \mathbf{x}_{t+1})$ in the EB
17:        end if
18:    end for
19:    Learning Stage:
20:    for Experience_Replay_Times = 1:Max_Replay_Times do
21:        Randomly sample a mini-batch of $N$ transitions $(\mathbf{x}_i, \mathbf{u}_i, r_i, \mathbf{x}_{i+1})$ from the EB
22:        calculate the Q-target as:
               $\bar{Q}_i = r_i + \gamma \max_{\mathbf{u}_k}\{\hat{Q}(\mathbf{x}_{i+1}, \mathbf{u}_k|\theta^{Q'})\}$
23:        Calculate the cost function:
               $J(\theta^Q) = \frac{1}{2N}\sum_{i=1}^{N}(\bar{Q}_i - \hat{Q}(\mathbf{x}_i, \mathbf{u}_i)|\theta^Q)^2$
24:        Calculate the mini-batch gradient:
               $\nabla_{\theta^Q} J(\theta^Q) = -\frac{1}{N}\sum_{i=1}^{N}(\bar{Q}_i - \hat{Q}(\mathbf{x}_i, \mathbf{u}_i)|\theta^Q)\frac{\partial \hat{Q}(\mathbf{x}_i, \mathbf{u}_i|\theta^Q)}{\partial \theta^Q}$
25:        Update the weights of the Q-network:
               $\theta^Q \leftarrow \theta^Q - \alpha^Q \nabla_{\theta^Q} J(\theta^Q)$
           ,
               $C_Q \longleftarrow C_Q + 1$
26:        if $C_Q \div T_Q = 0$ then
27:            Update the weights of the target network:
                   $\theta^{Q'} \leftarrow (1-\tau)\theta^{Q'} + \tau\theta^Q$
28:        end if
29:    end for
30: end for
```

In the update of weights $\theta^Q$, the target value for the $\omega$-th sample in the supervised learning of neural network $\bar{Q}_\omega$ is defined as

$$\bar{Q}_\omega = r_\omega + \gamma \max_{\mathbf{u}_k}\{\hat{Q}(\mathbf{x}_{\omega+1}, \mathbf{u}_k|\theta^{Q'})\}, \tag{10}$$

where $\theta^{Q'}$ is the weights of the target neural network.

To obtain more accurate estimated gradient, the mini-batch gradient decent approach can be applied instead of the stochastic gradient descent (SGD) approach. In the mini-batch gradient descend, $N$ experience samples from the EB are taken to calculate the estimated gradient to minimize the cost function. The cost function is chosen as the mean square error (MSE). The definition of MSE can be viewed as following:

$$J(\theta^Q) = \frac{1}{2N}\sum_{i=1}^{N}(\bar{Q}_i - \hat{Q}(\mathbf{x}_i, \mathbf{u}_i)|\theta^Q)^2. \tag{11}$$

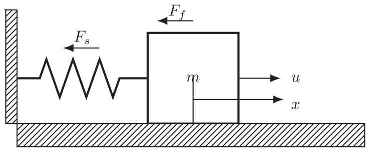

**Fig. 3.** Mass–spring–damping system.

To minimize the cost function, the mini-batch gradient of weights in cost function $J(\theta^Q)$ is calculated as:

$$\nabla_{\theta^Q} J(\theta^Q) = -\frac{1}{N}\sum_{i=1}^{N}(\bar{Q}_i - \hat{Q}(\mathbf{x}_i, \mathbf{u}_i)|\theta^Q)\frac{\partial \hat{Q}(\mathbf{x}_i, \mathbf{u}_i|\theta^Q)}{\partial \theta^Q}. \tag{12}$$

Then, the weights $\theta^Q$ in the NN can be updated according to the obtained mini-batch gradient:

$$\theta^Q \leftarrow \theta^Q - \alpha^Q \nabla_{\theta^Q} J(\theta^Q), \tag{13}$$

where $\alpha^Q$ is a small real number working as the learning rate of the neural network. $\alpha^Q$ does not have to be a constant during the whole training process. Using gradually reduced value of $\alpha^Q$ help accelerate the training process in the early episodes and stabilize the network in the late episodes. It is also worth mentioning that when $N = 1$, the mini-batch gradient descent will be reduced to SGD.

After every $T_Q$ times updating of $\theta^Q$, the weights of the target network $\theta^{Q'}$ will be updated by $\theta^Q$ softly [14] as follows:

$$\theta^{Q'} \leftarrow (1-\tau)\theta^{Q'} + \tau\theta^Q, \tag{14}$$

where $\tau \in (0, 1]$ is a constant determining the updating speed of the target network.

## 4. Simulation examples

In the simulation example section, there are two practical examples provided to verify the effectiveness of the proposed approach.

### 4.1. Mass–spring–damping system

Considering the mass–spring–damping system as shown in Fig. 3, the dynamic is given in the following equation according to Newton's law of motion [33]:

$$m\ddot{x} + F_f + F_s = u(t), \tag{15}$$

where $F_f = c\dot{x}$ is the resistive force due to friction and $F_s = k(1+a^2x^2)x$ is the restoring force of the spring, in which $k$ and $a$ are constants. $u(t)$ is the external force to control the mass–spring damping system.

In the mass–spring damping system simulation, the parameters representation and setting can be viewed in Table 1.

Then operation domain of the mass–spring damping system is considered for $x_1(t)$ in $[-3, 3]$ and $x_2(t)$ in $[-5, 5]$ and the state transitions satisfy $x_1(t)$ in $[-3, 3]$ and $x_2(t)$ in $[-10, 10]$ will be stored in the EB as part of experience.

The control objective in this example is to drive the position and the velocity of the system both to 0, i.e., $\mathbf{x}^0 = \mathbf{0}$.

Considering $\mathbf{x}(t) = [x_1(t), x_2(t)]^T = [x, \dot{x}]^T$, the dynamics in (15) can be rewritten in the state space form:

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathbf{A}(\mathbf{x}(t))\mathbf{x}(t) + \mathbf{B}u(t_\omega), \\ u(t) = u(t_\omega), \quad \forall t,\ t_\omega < t \le t_{\omega+1}, \\ \mathbf{x}(t_{\omega+1}) = \int_{t_\omega}^{t_{\omega+1}} \mathbf{A}(\mathbf{x}(t))\mathbf{x}(t) + \mathbf{B}u(t_\omega)dt. \end{cases} \tag{16}$$

In (16):

$$\mathbf{A}(\mathbf{x}(t)) = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m}(1 + a^2x_1(t)^2) & -\frac{c}{m} \end{bmatrix},$$

**Table 1**
Table of parameters and their values.

| Parameter | Symbol | Value |
|---|---|---|
| Mass | $m$ | 1 kg |
| Harding spring constant 1 | $k$ | 6 |
| Harding spring constant 2 | $a$ | 0.3 |
| Friction constant | c | 2 |
| Discount factor | $\gamma$ | 0.99 |
| Sampling interval | $h_s$ | 0.1s |
| Activation function 1 | $af_1$ | logsig |
| Activation function 2 | $af_2$ | linear |
| Number of weights in NN | $\|\theta\|$ | 50 |
| Mini-batch number | $N$ | 12 |
| Range of action space | $[\min\{\mathcal{U}\}, \max\{\mathcal{U}\}]$ | $[-20, 20]$ |
| Size of action set | $\|\mathcal{U}\|$ | 10 |
| Number of episodes | Max_Episodes | 500 |
| Number of time-steps | Max_Time_Step | 500 |
| Experience replay times | Max_Replay_Times | 500 |
| Network update threshold | $T_Q$ | 200 |
| Soft update parameter | $\tau$ | 0.5 |
| Learning rate of NN | $\alpha^Q$ | $\min(\max(0.005, \frac{1}{\text{episode}}), 0.05)$ |
| Exploration rate | $\epsilon$ | $\min(0.5, \frac{50}{\text{episode}})$ |

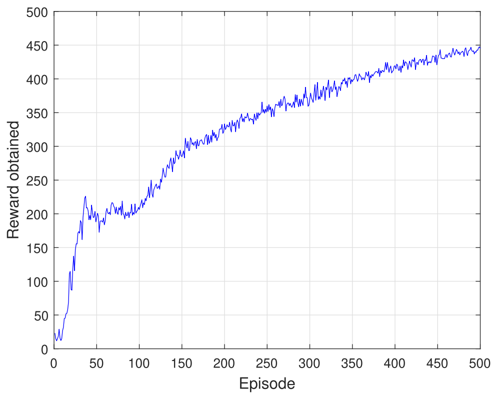

**Fig. 4.** Learning performance of the mass–spring damping system.

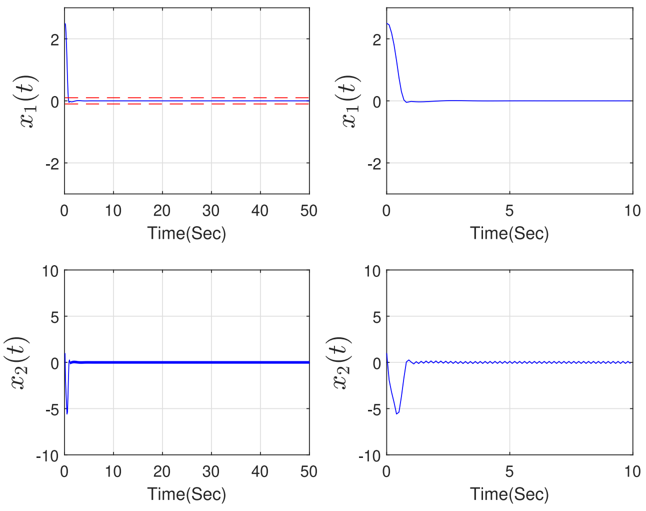

**Fig. 5.** State response of $x_1(t)$ and $x_2(t)$ for the mass–spring damping system. In the top sub-figures, the time responses of $x_1(t)$ are demonstrated under different time scale. In the below sub-figures, the time-responses of $x_2(t)$ are demonstrated under different time scale. The dashed lines in the top left sub-figure are the reward threshold in this simulation.

$$\mathbf{B} = \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix}.$$

To form the control problem into MDPs, the discount factor $\gamma$ is set as 0.99 and reward function for the $\omega$-th action at the $\omega$-th sampled state $\mathbf{x}(t_\omega)$ is set as:

$$r_\omega = \begin{cases} 1, & \text{if} \quad \|x_1(t_{\omega+1}) - x_1^o\|_2 \le 0.1 \\ 0, & \text{if} \quad \|x_1(t_{\omega+1}) - x_1^o\|_2 > 0.1 \end{cases}. \tag{17}$$

The reward function is defined to guide the learning of sampled-data RL controller. The reward function is equivalent to require the sampled-data RL controller to drive state of the control system to the target as fast as possible.

To guarantee that the RL agent explores the state space as much as possible, the initial state of the mass–spring damping system is chosen randomly for every episode: $\mathbf{x}_1(0)$ is chosen randomly from $[-3, 3]$ and $\mathbf{x}_2(0)$ is chosen randomly from $[-5, 5]$. Since the initial states are randomly picked, the simulations have been conducted 10 times and the average reward obtained in episodes is treated as the learning performance. The learning performance can be viewed in Fig. 4.

Since the maximum number of time step in one episode as shown in Table 1 is 500, therefore, the theoretical maximum reward can be obtained in one episode is 500. From Fig. 4, it can be found that the reward obtained in one episode is approaching the optimum in 500 episodes, which is an indicator that the sampled-data RL controller is optimized during the learning.

To have a better understanding of state-time relationship, the state response of the mass–spring damping system can be viewed in Figs. 5 and 6. In the state response simulation, the initial state is chosen as $[x_1(t), x_2(t)]^T = [2.5, 1]^T$ with sampling interval as 0.1s. It can be found that the RL agent is able to stabilize the system quickly (within 1 sec) subject to reasonable control input (within ±20 N).

**Remark 2.** For the purpose of comparison, the learning algorithm in [12–14] with integrated acting and learning processes is conducted under the same parameters setting as in Table 1. The simulations have been conducted 10 times as before. As shown in Fig. 7, both algorithms worked well in this example. Although, the learning speed for the first episodes is slightly higher for the proposed algorithm, no significant advantage of the proposed algorithm is observed. Nevertheless, in the next example, we will show the learning advantage of the proposed algorithm.

In addition, the sampled-data RL controller under different sampling intervals is investigated to show the effect of different sampling intervals on control performance. The phase portraits of the dynamics of mass–spring damping system can be viewed in Fig. 8. It can be clearly viewed that in the figure, all points in vector field (represented by blue arrows) swirl into the target state. Also, when the sampling interval

**Table 2**
Table of parameters and their values.

| Parameter | Symbol | Value |
|---|---|---|
| Gravitational acceleration | $g$ | 9.8 m/s$^2$ |
| Mass of the pendulum | $m_p$ | 1 kg |
| Mass of the cart | $m_C$ | 18 kg |
| Length of the pendulum | $2S$ | 1 m |
| Discount factor | $\gamma$ | 0.99 |
| Sampling interval | $h_s$ | 0.05s |
| Activation function 1 | $af_1$ | logsig |
| Activation function 2 | $af_2$ | linear |
| Number of weights in NN | $\lvert\theta\rvert$ | 98 |
| Mini-batch number | $N$ | 12 |
| Range of action space | $[\min\{\mathcal{U}\}, \max\{\mathcal{U}\}]$ | $[-100, 100]$ |
| Size of action set | $\lvert\mathcal{U}\rvert$ | 10 |
| Number of episodes | Max_Episodes | 500 |
| Number of time-steps | Max_Time_Step | 2400 |
| Experience replay times | Max_Replay_Times | 1500 |
| Network update threshold | $T_Q$ | 200 |
| Soft update parameter | $\tau$ | 0.5 |
| Learning rate of NN | $\alpha^Q$ | $\min(\max(0.005, \frac{1}{\text{episode}}), 0.05)$ |
| Exploration rate | $\epsilon$ | $\min(0.5, \frac{50}{\text{episode}})$ |

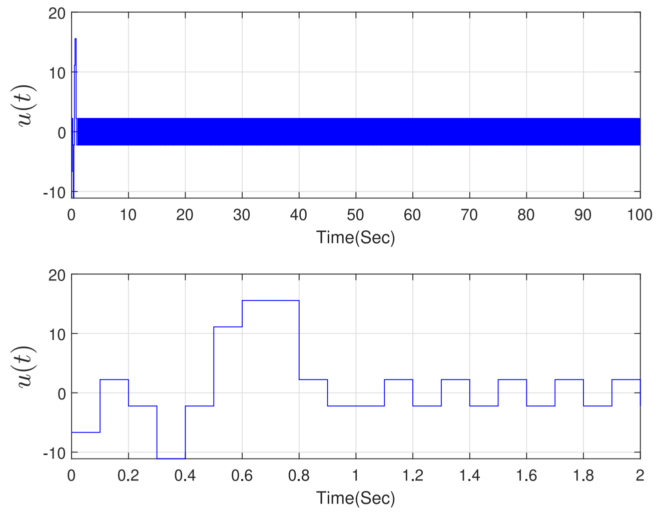

**Fig. 6.** Control input for the sampling interval 0.1 s. In the bottom sub-figure, the control input will be held as constant for 0.1s.

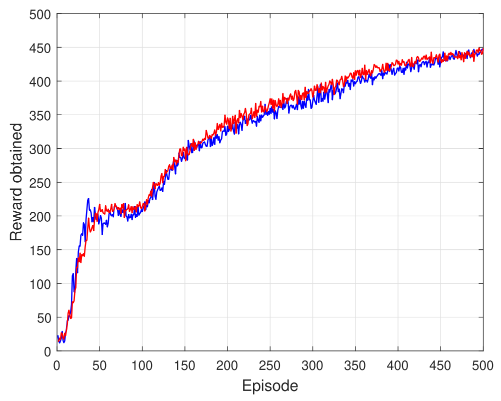

**Fig. 7.** Learning performance comparison for the decoupled method and the integrated method. The blue curve represents the learning performance of mass–spring damping system under decoupled learning algorithm. The red curve represents the learning performance of mass–spring damping system under integrated learning algorithm.

is small (0.02 s), the phase flow (represented by red curves) follows the vector field smoothly and swirls into the target state from different initial start state quickly. The simulation in phase space has been shown in the top left sub-figure. In the case that sampling interval is large (0.2 s), the phase flow fluctuates more in the vector field and detours a little to the target state from different initial states. The simulation in phase space has been shown in the bottom right sub-figure.

### 4.2. Inverted pendulum

In this section, we will consider the inverted pendulum on the cart as shown in Fig. 9. The dynamic of the inverted pendulum control system can be viewed in (18).

$$\ddot{x}_1(t) = \frac{g\sin(x_1(t)) - am_pS\dot{x}_1(t)^2\sin(2x_1(t))/2}{4S/3 - am_pS\cos^2(x_1(t))} - \frac{a\cos(x_1(t))u(t)}{4S/3 - am_pS\cos^2(x_1(t))} \tag{18}$$

where $a = \frac{1}{m_p+M_c}$, $m_p$ is the mass of the pendulum, $M_c$ is the mass of the cart, the length of the pendulum is $2S$, and $u(t)$ is the force applied on the cart. The parameter definitions and values in (18) can be viewed in Table 2.

In this example, the inverted pendulum system is controlled through the state feedback approach with the control objective that to drive the all states to 0, i.e. $\mathbf{x}^o = \mathbf{0}$. Instead of learning the exact control input directly, the RL agent in this example is to learn the feedback gain $k(t_\omega)$ of the inverted pendulum system and the control input is defined as $u(t_\omega) = k(t_\omega)\cdot(x_1^o - x_1(t_\omega))$. To explore more in the state space, the initial state $x_1(t)$ of the inverted pendulum for every episode is randomly chosen from $[-\frac{5\pi}{12}, \frac{5\pi}{12}]$. The operation domain of the inverted pendulum system is considered for $x_1(t)$ in $[-\frac{5\pi}{12}, \frac{5\pi}{12}]$ and $x_2(t)$ in $[-10, 10]$. The states satisfy $x_1(t)$ in $[-\frac{6\pi}{12}, \frac{6\pi}{12}]$ and $x_2(t)$ in $[-10, 10]$ will be stored in the EB as part of experience.

let $\mathbf{x}(t) = [x_1(t), x_2(t)]^T$ and the dynamic equation can be rewritten as the state-space form:

$$\begin{cases} \dot{\mathbf{x}}(t) = \mathbf{A}(\mathbf{x}(t))\mathbf{x}(t) + \mathbf{B}(\mathbf{x}(t))u(t), \\ u(t) = u(t_\omega), \quad \forall t,\ t_\omega < t \le t_{\omega+1}, \\ \mathbf{x}(t_{\omega+1}) = \int_{t_\omega}^{t_{\omega+1}} \mathbf{A}(\mathbf{x}(t))\mathbf{x}(t) + \mathbf{B}(\mathbf{x}(t))u(t_\omega)dt. \end{cases} \tag{19}$$

where

$$\mathbf{A}(\mathbf{x}(t)) = \begin{bmatrix} 0 & 1 \\ f_1(\mathbf{x}(t)) & 0 \end{bmatrix},$$

$$\mathbf{B}(\mathbf{x}(t)) = \begin{bmatrix} 0 \\ f_2(\mathbf{x}(t)) \end{bmatrix}.$$

$f_1(\mathbf{x}(t))$ and $f_2(\mathbf{x}(t))$ in $\mathbf{A}(\mathbf{x}(t))$ and $\mathbf{B}(\mathbf{x}(t))$ are defined as:

$$f_1(\mathbf{x}(t)) = \frac{g - am_pSx_2(t)^2\cos(x_1(t))}{4S/3 - am_pS\cos^2(x_1(t))}\left(\frac{\sin(x_1(t))}{x_1(t)}\right)$$

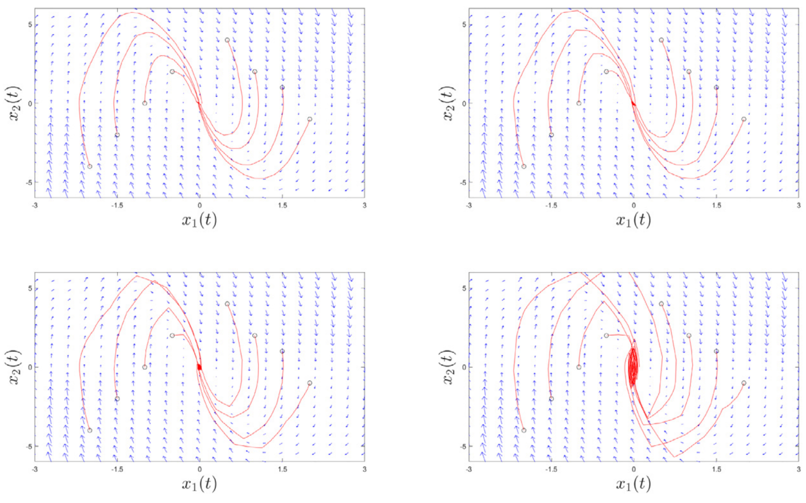

**Fig. 8.** Phase portrait of $x_1(t)$ and $x_2(t)$ for the mass–spring damping system from different initial states. The blue arrows represent the direction and magnitude of the vector field. The red curves represent the phase flow. The initial states of trajectories are denoted by black circles. The sampling intervals for the sub-figure on the top left, top right, bottom left, bottom right are 0.02 s, 0.05 s, 0.1 s and 0.2 s, respectively.

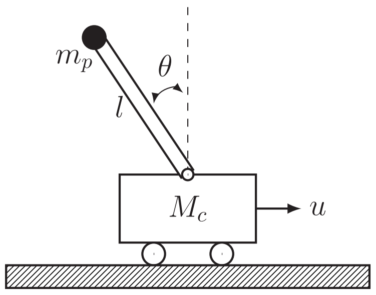

**Fig. 9.** Inverted pendulum system.

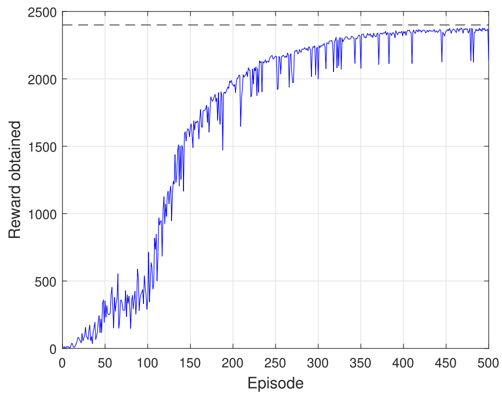

**Fig. 10.** Learning performance of the inverted pendulum system.

and

$$f_2(\mathbf{x}(t)) = \frac{-a\cos(x_1(t))}{4S/3 - am_pS\cos^2(x_1(t))}.$$

To form the control problem into MDPs, the discount factor $\gamma$ is set as 0.99 and the reward function for the $\omega$-th action at state $\mathbf{x}_{t_\omega}$ is set as:

$$r_\omega = \begin{cases} 1, & \text{if} \quad \|x_1(t_{\omega+1}) - x_1^o\|_2 \le \dfrac{1}{36}\pi \\ 0, & \text{if} \quad \|x_1(t_{\omega+1}) - x_1^o\|_2 > \dfrac{1}{36}\pi \end{cases}, \tag{20}$$

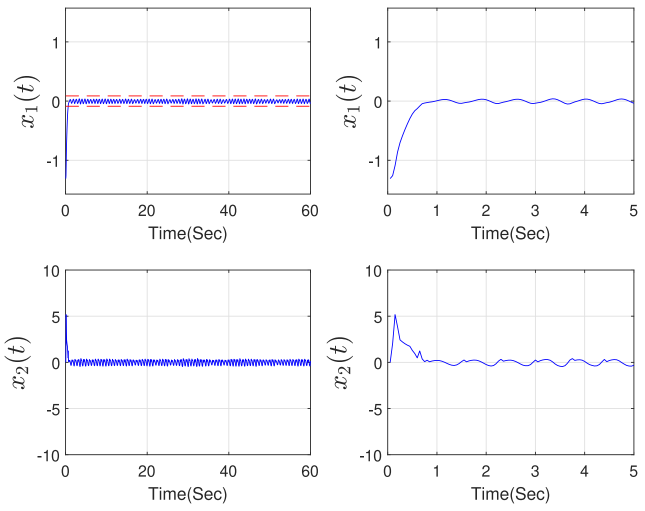

**Fig. 11.** State response of $x_1(t)$ and $x_2(t)$ for the inverted pendulum system. In the top sub-figures, the time responses of $x_1(t)$ are demonstrated under different time scale. In the below sub-figures, the time-responses of $x_2(t)$ are demonstrated under different time scale. The dashed lines in the top left sub-figure are the reward threshold in this simulation.

which demonstrates that only the states very close to target state will receive the reward while others will not, thus the sampled-data RL controller will drive the state of control system to the target as fast as possible.

Since the initial position of the inverted pendulum is randomly picked, which could affect the learning process. The learning of the sampled-data control strategy is conducted 10 times and the average learning curve can be viewed in Fig. 10.

In Fig. 10, the blue curve represents the rewards obtained in one episode while the dashed black line represents the possible maximum rewards could be obtained in one episode. Since the maximum number of time step in one episode as shown in Table 2 is 2400, therefore, the theoretical maximum rewards can be obtained in one episode is 2400. From Fig. 10, it can be found that the reward obtained in one episode is approximating the optimum in 500 episodes, which is an indicator that the sampled-data controller is optimized during the learning.

As in the first example, to show the state-time relationship, the time response and the control input of the inverted pendulum system can be

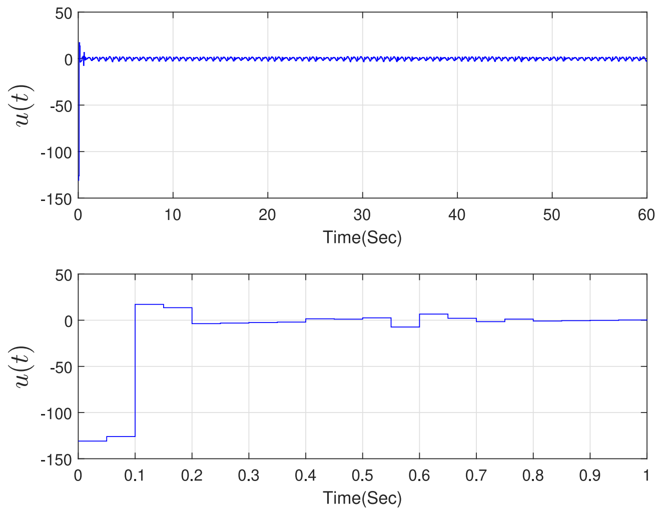

**Fig. 12.** Control input for the sampling interval 0.05 s. In the bottom sub-figure, the control input will be held as constant for 0.05s.

viewed in Figs. 11 and 12. In the time response simulation, the initial state is chosen as $[x_1(t), x_2(t)] = [-\frac{5\pi}{12}, 0]$ with sampling interval 0.05s. From the figures, it can be found that the RL agent is able to fulfil the control objective within 1 second and the control input is within reasonable range ($[-150\text{N}, 50\text{N}]$).

**Remark 3.** Along the same way to compare in the first example, the learning algorithm in [12–14] with integrated acting and the learning processes is conducted under the same parameters setting as in Table 2. The simulations have been conducted 10 times as before. From the learning performance comparison shown in Fig. 14, it can be found that the RL agent only starts to improve after 1000 episodes for the integrated algorithm (represented by red curve) while the RL agent approaches the optimum in 500 episodes for the de-coupled algorithm (represented by blue curve). This comparison demonstrates the effectiveness of the proposed approach. Since one episode means conducting the experiment once in real-life applications, to reduce the number of episodes can reduce the cost of experiments especially when the experiments are expensive to conduct.

As in the first example, the sampled-data RL controller under different sampling intervals is investigated to show the effect of different sampling intervals the control performance. The phase portraits of the inverted pendulum can be viewed in Fig. 13, it can be found that the control system can be stabilized from different initial states since all the phase flow (represented by red curves) swirls into the target state. It can be also found that when the sampling interval is small (0.01s), the phase flow goes to the target state with little fluctuation from different initial start states. The simulation in phase space has been shown in the top left sub-figure. In the case that sampling interval is large (0.05s), the phase flow fluctuates more in the vector field (represented by blue arrows) and detours a little into the target state from initial states. The simulation in phase space has been shown in the bottom right sub-figure.

**Remark 4.** The phase portraits for both examples demonstrate the effect of sampling intervals on control performance. In the case that the sampling interval is small, the RL agent is able to control the system more smoothly while in the case that sampling interval is large, the control process is chattering more and under the danger of failure. However, for the sampled-data RL controller with small sampling intervals, the computational demand is higher since the RL controller will yield control input at every short sampling interval. The computational cost can be reduced by applying larger sampling intervals, but the control performance will be reduced. Therefore, compromise has to be made on the control performance and computational cost for different requirements. Properly designed sampling interval should guarantee the decent control performance with acceptable computational cost.

## 5. Conclusions

In this paper, the sampled-data control strategy through model-free reinforcement learning with effective experience replay is proposed. In the proposed strategy, the sampled-data control problem has been formed as the MDP problem. An intelligent RL agent is to learn the optimal control strategy through interacting with the environment. In the proposed RL algorithm, the acting and learning stages are decoupled to allow the customized experience replay, which enhanced the effectiveness of experience replay in terms of both the number of experience replay times and the most effective experience. In addition, the sampled-data control strategy is applied to reduce the computational demand of the RL agent as well as make the digital application possible. The effect of sampling control input has been discussed in detail. Through the simulation examples, the effectiveness of the proposed approach is verified.

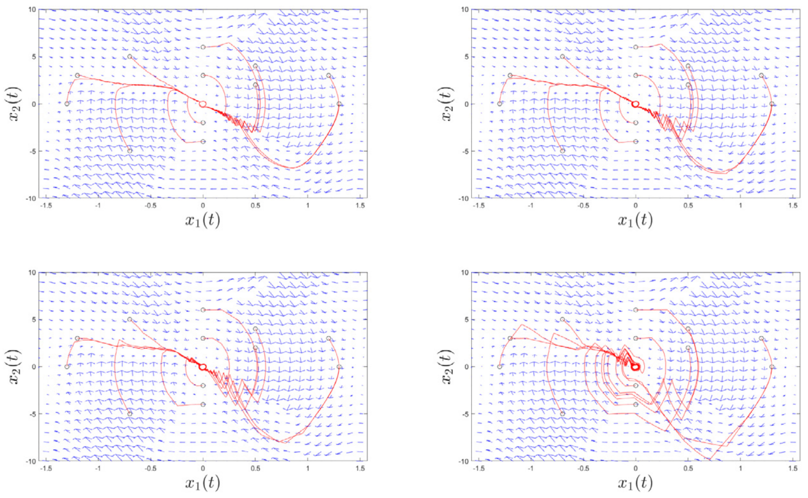

**Fig. 13.** Phase plot of $x_1(t)$ and $x_2(t)$ for the inverted pendulum system from different initial states. The initial states of trajectories are denoted by black circles. The blue arrows represent the direction and magnitude of the vector field. The red curves represent the phase flow. The sampling intervals for the sub-figure on the top left, top right, bottom left, bottom right are 0.01s, 0.02s, 0.03s and 0.05s, respectively.

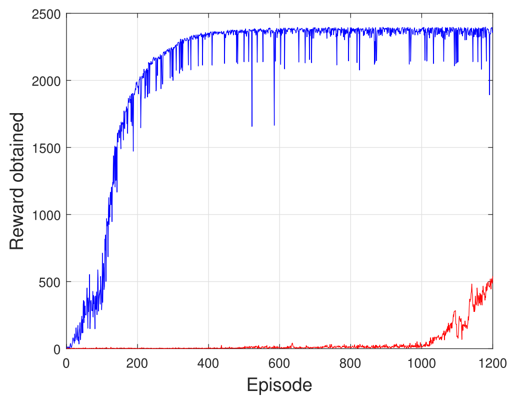

**Fig. 14.** Learning performance comparison for the decoupled method and the integrated method. The red curve represents the learning performance of mass–spring damping system under de-coupled learning algorithm. The blue curve represents the learning performance of inverted pendulum system under integrated learning algorithm.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

No data was used for the research described in the article.

## Acknowledgments

This work was partially supported by Imperial College London, King's College London and Engineering and Physical Sciences Research Council (EPSRC).

## References

[1] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, vol. 1, (1) MIT press Cambridge, 1998.
[2] C.J.C.H. Watkins, Learning from delayed rewards (Ph.D. thesis), King's College, Cambridge, 1989.
[3] K. Doya, Reinforcement learning in continuous time and space, Neural Comput. 12 (1) (2000) 219–245.
[4] M. Abu-Khalaf, F.L. Lewis, Nearly optimal control laws for nonlinear systems with saturating actuators using a neural network HJB approach, Automatica 41 (5) (2005) 779–791.
[5] D. Vrabie, F. Lewis, Neural network approach to continuous-time direct adaptive optimal control for partially unknown nonlinear systems, Neural Netw. 22 (3) (2009) 237–246.
[6] J. Fu, H. He, X. Zhou, Adaptive learning and control for MIMO system based on adaptive dynamic programming, IEEE Trans. Neural Netw. Learn. Syst. 22 (7) (2011) 1133–1148.
[7] S. Bhasin, R. Kamalapurkar, M. Johnson, K.G. Vamvoudakis, F.L. Lewis, W.E. Dixon, A novel actor-critic-identifier architecture for approximate optimal control of uncertain nonlinear systems, Automatica 49 (1) (2013) 82–92.
[8] B. Kiumarsi, F.L. Lewis, H. Modares, A. Karimpour, M.-B. Naghibi-Sistani, Reinforcement Q-learning for optimal tracking control of linear discrete-time systems with unknown dynamics, Automatica 50 (4) (2014) 1167–1175.
[9] D. Liu, X. Yang, D. Wang, Q. Wei, Reinforcement-learning-based robust controller design for continuous-time uncertain nonlinear systems subject to input constraints, IEEE Trans. Cybern. 45 (7) (2015) 1372–1385.
[10] B. Kiumarsi, K.G. Vamvoudakis, H. Modares, F.L. Lewis, Optimal and autonomous control using reinforcement learning: A survey, IEEE Trans. Neural Netw. Learn. Syst. (2017).
[11] D. Wang, D. Liu, Y. Zhang, H. Li, Neural network robust tracking control with adaptive critic framework for uncertain nonlinear systems, Neural Netw. 97 (2018) 11–18.
[12] S. Adam, L. Busoniu, R. Babuska, Experience replay for real-time reinforcement learning control, IEEE Trans. Syst., Man, Cybern. C, Appl. Rev. 42 (2) (2012) 201–212.
[13] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529.
[14] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, 2015, arXiv preprint arXiv:1509.02971.
[15] B. Luo, D. Liu, T. Huang, D. Wang, Model-free optimal tracking control via critic-only Q-learning, IEEE Trans. Neural Netw. Learn. Syst. 27 (10) (2016) 2134–2144.
[16] B. Xiao, H.K. Lam, C. Xuan, Z. Wang, E.M. Yeatman, Optimization for interval type-2 polynomial fuzzy systems: A deep reinforcement learning approach, IEEE Trans. Artif. Intell. (2022).
[17] P. Abbeel, A. Coates, M. Quigley, A. Ng, An application of reinforcement learning to aerobatic helicopter flight, Adv. Neural Inf. Process. Syst. 19 (2006).
[18] J. Kober, J.A. Bagnell, J. Peters, Reinforcement learning in robotics: A survey, Int. J. Robot. Res. 32 (11) (2013) 1238–1274.
[19] S.C. Bacha, W. Bai, Z. Wang, B. Xiao, E.M. Yeatman, Deep reinforcement learning-based control framework for multilateral telesurgery, IEEE Trans. Med. Robot. Bionics 4 (2) (2022) 352–355.
[20] Z. Wang, W. Bai, Z. Chen, B. Xiao, B. Liang, E.M. Yeatman, Multiple-pilot collaboration for advanced remote intervention using reinforcement learning, in: IECON 2021–47th Annual Conference of the IEEE Industrial Electronics Society, IEEE, 2021, pp. 1–6.
[21] C.J. Watkins, P. Dayan, Q-learning, Mach. Learn. 8 (3–4) (1992) 279–292.
[22] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Netw. 2 (5) (1989) 359–366.
[23] K. Hornik, M. Stinchcombe, H. White, Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks, Neural Netw. 3 (5) (1990) 551–560.
[24] P. WawrzyńSki, A.K. Tanwani, Autonomous reinforcement learning with experience replay, Neural Netw. 41 (2013) 156–167.
[25] L.-J. Lin, Self-improving reactive agents based on reinforcement learning, planning and teaching, Mach. Learn. 8 (3–4) (1992) 293–321.
[26] T. Chen, B.A. Francis, Optimal Sampled-Data Control Systems, Springer Science & Business Media, 2012.
[27] H.K. Lam, F.H. Leung, Design and stabilization of sampled-data neural-network-based control systems, IEEE Trans. Syst., Man Cybern. B, Cybern. 36 (5) (2006) 995–1005.
[28] H. Katayama, A. Ichikawa, $H_\infty$ control for sampled-data nonlinear systems described by Takagi-Sugeno fuzzy systems, Fuzzy Sets and Systems 148 (3) (2004) 431–452.
[29] X. Jiang, On sampled-data fuzzy control design approach for T-S model-based fuzzy systems by using discretization approach, Inform. Sci. 296 (2015) 307–314.
[30] H.J. Lee, D. Wan Kim, Intelligent digital redesign revisited: Approximate discretization and stability limitation, Fuzzy Sets and Systems 159 (23) (2008) 3221–3231.
[31] H.C. Sung, D.W. Kim, J.B. Park, Y.H. Joo, Robust digital control of fuzzy systems with parametric uncertainties: LMI-based digital redesign approach, Fuzzy Sets and Systems 161 (6) (2010) 919–933.
[32] D.W. Kim, H.J. Lee, Sampled-data observer-based output-feedback fuzzy stabilization of nonlinear systems: exact discrete-time design approach, Fuzzy Sets and Systems 201 (2012) 20–39.
[33] H.K. Khalil, Noninear systems, Prentice-Hall, New Jersey 2 (5) (1996) 5–1.