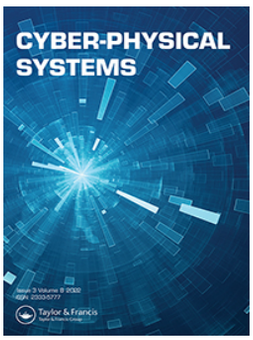

Cyber-Physical Systems

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/tcyb20

# Q-Learning Based Optimisation Framework for Real-Time Mixed-Task Scheduling

Tianchuang Meng, Jin Huang, Huiqian Li, Zengkun Li, Yu Jiang & Zhihua Zhong

**To cite this article:** Tianchuang Meng, Jin Huang, Huiqian Li, Zengkun Li, Yu Jiang & Zhihua Zhong (2022) Q-Learning Based Optimisation Framework for Real-Time Mixed-Task Scheduling, Cyber-Physical Systems, 8:3, 173-191, DOI: 10.1080/23335777.2021.1900922

**To link to this article:** https://doi.org/10.1080/23335777.2021.1900922

Published online: 04 Apr 2021.

CYBER-PHYSICAL SYSTEMS
2022, VOL. 8, NO. 3, 173–192
https://doi.org/10.1080/23335777.2021.1900922

Taylor & Francis
Taylor & Francis Group

ARTICLE



# Q-Learning Based Optimisation Framework for Real-Time Mixed-Task Scheduling

Tianchuang Meng[a], Jin Huang[a], Huiqian Li[a], Zengkun Li[b], Yu Jiang[c] and Zhihua Zhong[a]

[a]School of Vehicle and Mobility, Tsinghua University, Beijing, China; [b]School of Software, Tsinghua University, Beijing, China; [c]College of Computer Science and Technology, Jilin University, Changchun, China

**ABSTRACT**

Mixed periodic and aperiodic tasks with explicit deterministic or probabilistic timing requirements are becoming increasingly deployed in real-time industry control systems. Such systems pose significant challenges to the scheduling algorithms because the failure of scheduling can be catastrophic. In the past decades, significant research effort has been dedicated on the scheduling problems, and various scheduling algorithms were proposed to meet various system requirements and task loads. However, a single fixed scheduling algorithm usually cannot fully satisfy the requirements for a dynamic mixed-task-set, which is commonly found in modern complex real-time control systems. It is thus extremely hard for engineers to design a set of scheduling solutions to guarantee the correctness and optimality under all conditions. Aiming at optimising the scheduling performance in a real-time control system, this paper proposes a Q-learning-based optimisation framework to select proper scheduling algorithms for the mixed-task-set. Built on a three-layer perceptron network, our Q-learning framework is able to efficiently and effectively choose scheduling algorithms that dynamically adapt to the characteristics of task-sets. Experimental results using real-world data proved the effectiveness of the proposed framework.

**ARTICLE HISTORY**
Received 7 October 2020
Accepted 20 February 2021

**KEYWORDS**
Q-learning; optimisation; mixed-task scheduling; real-time systems; automatic train operation system

## 1. Introduction

Real-time systems are pervasively used to control physical processes ranging from automobile ignition systems to nuclear power plants. With the rising complexity of applications, an increasing number of industry-strength real-time systems deploy both periodic and aperiodic tasks. Such tasks typically have explicit deterministic or probabilistic timing requirements [1]. Periodic tasks are characterised by regular arrival times and hard deadlines, while aperiodic tasks are featured by irregular arrival times and either soft or hard

**CONTACT** Jin Huang huangjin@tsinghua.edu.cn School of Vehicle and Mobility, Tsinghua University, Beijing, China

deadlines. For such systems, the correctness of system functioning depends upon not only the results of computation but also on the execution order of such computations.

The Automatic Train Operation (ATO) system is a typical real-time industry control system that automatically drives the train according to the railway routes, train properties, traffic lights as well as the interactions with human drivers or dispatchers [2]. As the most sustainable means of modern transportation, modern railway is consistently growing more intelligent and environmental friendly. Towards such an objective, an essential momentum is the autonomous driving of railway vehicles to enable energy efficiency, punctuality and safety. Since the ATO system is an automatic intelligent system, many valuable research works in automotive engineering can serve as good references for the development of the ATO system [3,4]. The ATO system is a typical complex mixed-task system because it involves a rich set of periodic and aperiodic tasks for various trip planning and real-time control problems. For example, the periodic tasks of the system involve the collection and calculation of real-time information, heartbeat feedback signal processing, fault diagnosis [3], the information collection and transmission of input board and output board, etc. The non-periodic tasks are mainly caused by some random events, including the handle events of exception, railway transit signal, temporary trip planning, route extraction and other service tasks. In addition, the nature of railway transportation makes it a safety-critical system. The response time of various tasks should be as short as possible to guarantee the driving safety. Meanwhile, the system scheduling overhead should also be emphasised besides the response time to reach a better comprehensive performance. As a result, the scheduling of the ATO system considering both the response time and scheduling overhead serves as an excellent example for industry-strength mixed-task real-time systems.

**Related works**. A large body of research has been dedicated to the scheduling problems of mixed aperiodic and periodic tasks. Next, we discuss the main scheduling algorithms with respect to periodic and aperiodic tasks.

For periodic tasks in the ATO system, the commonly used scheduling algorithms include rate monotonic (RM) scheduling algorithm [5], deadline monotonic (DM) scheduling algorithm, earliest deadline first (EDF) scheduling algorithm [6], least slack first (LST) algorithm, etc. According to the working mechanism, these algorithms can be mainly categorised into static scheduling algorithms and dynamic scheduling algorithms. RM and DM belong to static scheduling algorithms, while EDF and LST belong to dynamic scheduling algorithms. The priority of periodic tasks is the main feature in the RM scheduling algorithm. More specifically, the shorter the period of the task, the higher the priority it receives. This is because the tasks with shorter periods come more frequently than those with longer periods and thus are easier to accumulate. Therefore, RM is expected to schedule tasks with shorter periods as quickly as

possible. RM turns out to be very popular in real-time static priority scheduling systems with high security for its simplicity and high efficiency. However, RM lacks flexibility and its CPU utilisation has upper limitations for the reason of static scheduling. Unlike RM, DM determines priority by deadline rather than by period. The shorter the deadline, the more urgent the task should be and thus the higher the priority of the task. The static scheduling algorithms mentioned earlier use static variables to determine the priority of tasks, and the dynamic scheduling algorithms are determined by dynamic variables. In the EDF algorithm, the priority is assigned to the remaining execution time of the task. The task with fewer remaining execution times receives higher priority. EDF has high performance in single-core processors. LST is a dynamic scheduling algorithm, which determines priority by the length of slack time. The difference between deadline and the rest execution time is a good indicator of how idle the task is. The larger the difference, the lower the priority. LST works well especially when the system is under heavy workload. However, frequent task switching scenarios (jolting phenomenon) may occur when the priority of the tasks changes rapidly, which may have great impact on the performance of the system. Unless otherwise specified, this paper adopts RM algorithm for the scheduling of periodic tasks for its reliability and simplicity.

For the aperiodic tasks, the existing scheduling algorithms can mainly be classified into three categories: the background (BG) algorithms, the polling-server-based algorithms and the slack-stealing-based algorithms [7].

Firstly, the BG algorithms simply give the priority to the scheduling of periodic tasks and merely deal with the aperiodic tasks in the BG of the periodic tasks when the CPU is idle. Such simple algorithms tend to excessively emphasise the importance of periodic tasks and can be ineffective when the purpose of the scheduling is to meet the deadline of period tasks and reduce the response time of aperiodic tasks [8].

Secondly, the polling-server-based algorithms transform the aperiodic tasks into periodic tasks by creating a periodic polling server task which is responsible for scheduling the aperiodic tasks. The periodic polling server task can schedule aperiodic tasks when it has positive processing capacity. Due to the short period, high priority and long processing time of this periodic polling server task, the aperiodic tasks can get more opportunities to be scheduled, thus greatly reducing the response time of the aperiodic tasks under the condition of ensuring the scheduling of the periodic tasks. Different polling-server-based algorithms arise from different approaches to supplementing and maintaining the processing capacity of the server. Typical algorithms in this category include the Priority Exchange (PE) algorithm [9], the Deferrable Server (DS) algorithm [10], the Sporadic Server (SS) algorithm [11] and Total Bandwidth Server (TBS) algorithm. The priority of the periodic polling server is generally set to high in order to reduce the response time of the aperiodic tasks. Recent research in such track also targets to reduce response time and energy consumption of

mixed-task-set using a PE server [12], to explore the adaptive scheduling by using predictive execution time [13], as well as the dynamic voltage scaling of mixed-task-sets in priority-driven systems that involve different algorithms, e.g. DS, SS and TBS [14]. Such polling-server-based algorithms were also applied to the real-time scheduling for mixed-task-set on multicore processors [15]. However, when the task of the periodic server missed or delayed in coordinating the aperiodic random arrived tasks, idle operations will have to appear. In such a case, either the aperiodic tasks have to wait for the return of the server task, or the higher priority server task runs vacuously.

Thirdly, the slack-stealing-based algorithms, which attempt to allocate time for aperiodic tasks by stealing the processing time from the periodic tasks without causing the missing the respective deadlines, prove to be the most effective ones (at least theoretically). Endeavours on optimisation and simplification of such algorithms include the Optimal Fixed-Priority (OFP) algorithm [8], the Optimal Dynamic Driven algorithm [16], etc. However, the algorithms are difficult to implement and often require considerable computation resources in practice.

General introductions and reviews on recent development of the scheduling algorithms can refer to References [17–20]. It should be noted that the scheduling of mixed-task-set is also becoming more essential in the process scheduling on multicore CPUs recently [21–23]. The above mentioned real-time scheduling algorithms exhibit varying trade-offs in terms of scheduling performance, computing efficiency and characteristics of tasks. The choosing of a proper scheduling algorithm, however, is application-dependent. In fact, the design of scheduling algorithm has to adapt to the underlying tasks.

**This Research**. In this paper, we propose a Q-learning-based framework for optimising the scheduling solution of mixed-task real-time control systems. The basic idea is to leverage the effectiveness of the reinforcement learning (RL) approach to derive an optimised scheduling solution that adapts to the characteristics of task-set. The framework is based on a three-layer perceptron network, which is computation-efficient and capable of approximating a wide range of functions. To the best of the authors' knowledge, this work is the first one to endeavour on optimising the scheduling problems of mixed-task real-time control applications by leveraging the success of the RL framework.

## 2. Problem description

### *2.1. The scheduling problem*

We focus on the scheduling of mixed-tasks in the ATO system in this study. Specifically, the task-set consists of mixed-tasks consisting of both periodic tasks with hard time constraints and aperiodic tasks with soft constraints. The objective is to schedule the mixed-tasks in such a way that all periodic task deadlines

are met and the response time for the aperiodic tasks are as short as possible. The processing of tasks in a frame consists of three parts: the scheduling overhead, the task execution and the postprocessing.

### 2.2. Markovian decision process analysis

Consider a real-time system with $n$ periodic tasks $T_1, \cdots, T_n$, one can calculate the hyper-period $H$ of the task-set as the least common multiple of various periods. Actually, the whole processing time of a task-set can be taken as scheduled in many series hyper-periods, $H_1, \cdots, H_m$. The aperiodic tasks can be referred as the tasks that come randomly in a hyper-period $H_i (i = 1, 2, \cdots, m)$. The scheduler can use different algorithms according to the state of every hyper-period. In this way, we transform this complex scheduling problem into a multistage decision-making process. Note that these stages are correlated with each other consecutively, that is, the state of the present stage depends on that of the preceding stage. As a result, according to the definition, the problem can be formulated as a Markovian decision process (MDP).

We use a four-tuple $K(T_{per}, T_{arr}, T_{exe}, T_{end})$ to represent a task in the scheduling process, where $T_{per}$ is the period of the task, $T_{arr}$ is the time of the arrival of the task, $T_{exe}$ represents the time of the worst execution of the task and $T_{end}$ represents the finish time of the task. Here, $T_{per}$ of an aperiodic task is set to be 0.

For periodic tasks in a hyper-period $H$, they are generally scheduled according to a fixed priority algorithm, e.g. the First-In-First-Out (FIFO) scheduling algorithm or the RM scheduling algorithm [5]. Assuming that the deadline of the periodic tasks is the time when they comes again, the scheduling results of each periodic task should match the following constraint:

$$T_{end} \leq T_{arr} + T_{per}. \tag{1}$$

The aperiodic tasks in a hyper-period $H$ can be scheduled with any choice of priority, even dynamically. The response time $N$ of an aperiodic task by using the algorithm $i$ can be described as

$$N^i = T^i_{end} - T_{arr} - T^i_{exe}. \tag{2}$$

The response time of the aperiodic tasks in a hyper-period $H_j$ by employing the algorithm $i$ can be represented as

$$R^i_j = \sum N^i. \tag{3}$$

Define the complexity index of the scheduling overhead and post-processing of an algorithm $i$ in a hyper-period $H_j$ as $C^i_j$. Then, the optimisation objective of the whole scheduling process can be described as

$$\min\left(\sum_{j=1}^{m}(R_j^i + \alpha C_j^i)\right), \tag{4}$$

subject to the real-time constraints of both periodic tasks and aperiodic tasks. Here $\alpha$ is the weight on how much we stress on the complexity index relative to the response time. We simply set $\alpha = 1$ in this paper. Follow such track, to optimise the scheduling problem, we can find a strategy to select the algorithms $i_1, \cdots, i_m$ in the hyper-periods $H_1, \cdots, H_m$, respectively, to achieve the goal of (4) under the following assumptions: 1) there is limited buffer space for the aperiodic tasks; 2) tasks do not suspend themselves or synchronise with any other task.

### 2.3. Existing algorithms analysis

As discussed earlier, the objective is to schedule the mixed-tasks in such a way that all periodic task deadlines are met and the response times for the aperiodic tasks are as short as possible. The processing of tasks in a frame consists of three parts: the scheduling overhead, the task execution and the post-processing.

Generally speaking, the BG schedule algorithm usually incurs the lowest schedule overhead. The polling-server-based algorithm takes a high priority to schedule the aperiodic tasks to reduce the response time of aperiodic tasks. They usually perform better under the situations where the frame starts with the aperiodic tasks exactly. The slack-stealing-based algorithms compute the maximum time and record the related information in real time. They induce a high scheduling overhead but generally offer the best level of performance.

To further illustrate different performances of these algorithms, we give a couple of examples by constructing a mixed-task real-time system and comparing scheduling results of different algorithms. We assume that the real-time system consists of two periodic tasks and two aperiodic tasks which are, respectively, represented as four-tuple forms, $K_1(6,\ 0,\ 2,\ T_{end1})$, $K_2(8,\ 1,\ 2,\ T_{end2})$, $K_3(0,\ 4,\ 3,\ T_{end3})$ and $K_4(0,\ 12,\ 1,\ T_{end4})$. As we have declared, we adopt RM algorithm for the scheduling of periodic tasks for its reliability and simplicity. With regard to aperiodic task scheduling algorithms, we choose BG, DS, SS and OFP to, respectively, represent the three kinds of algorithms mentioned earlier. Figure 1 illustrates the scheduling results of BG, DS, SS, OFP, respectively. All periodic tasks are scheduled in the same way. As for the total response time of the aperiodic tasks, BG returns 4, DS and SS returns 2, and OFP returns 0. The results accurately reflect the different performance of those algorithms discussed above.

In summary, concerning the trade-off between performance and computing efficiency, there are no generically optimal scheduling solutions. Next, we will

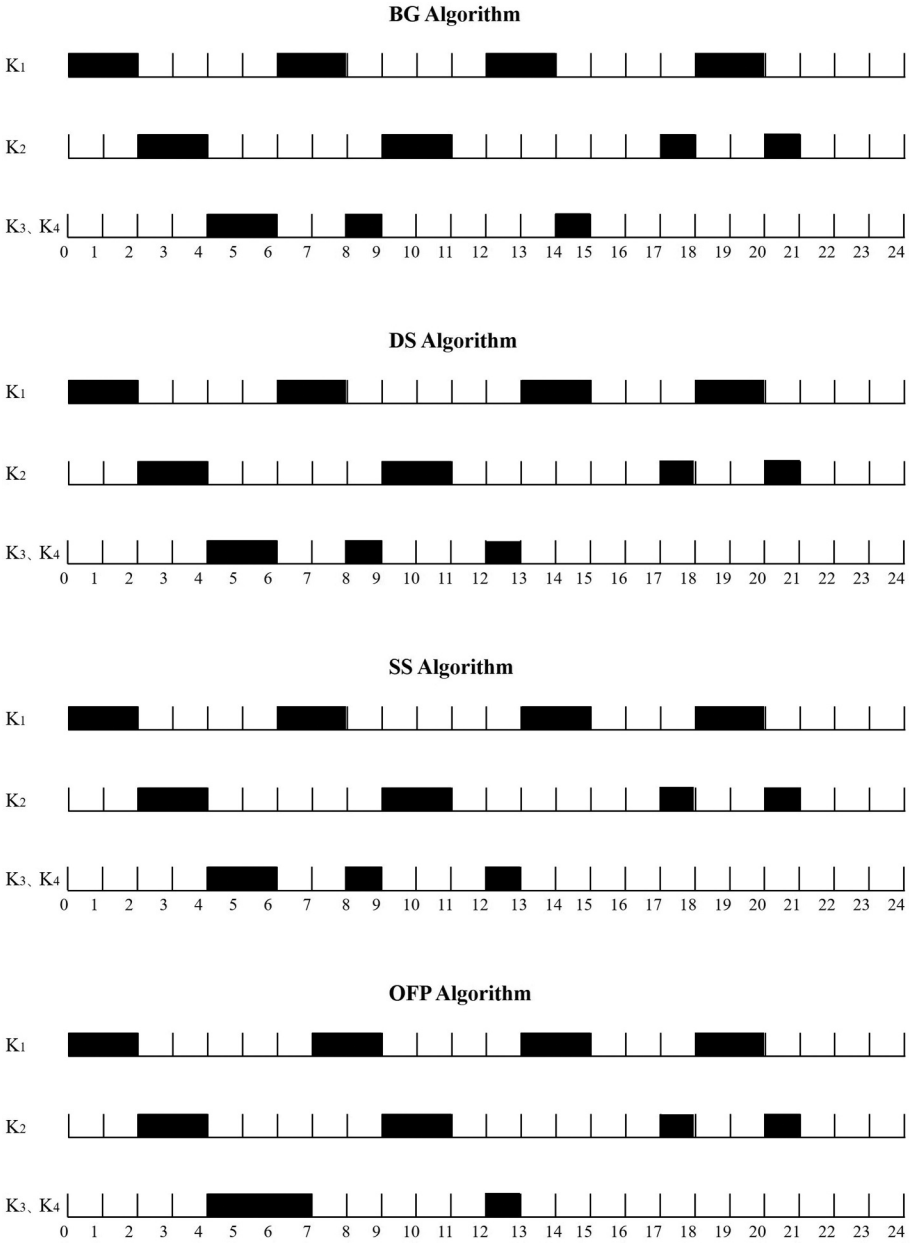


**Figure 1.** Scheduling results of BG, DS, SS, OFP.

further introduce how a task schedule optimisation problem can be formulated, with its properties of MDP being explored.

## 3. The Q-learning algorithm

RL is powerful framework to solve problems with the properties of MDP. RL is quite different from supervised learning, which is the kind of learning studied in most current research in the field of machine learning. Supervised learning learns from a training set of labelled examples provided by a knowledgeable external supervisor. Each example in the training set consists of a description of a situation together with a label specifying the correct action the system should take to that situation. RL is also different from unsupervised learning in machine learning, which is typically about finding structure hidden in collections of unlabelled data. As opposed to supervised learning which learns the knowledge from the examples labelled in advance, RL learns by incrementally interacting with the environment and responding to the receipt of rewards or penalties based on the impact each action has on the systems [24]. General terminologies in RL can be referred to Table 1.

Initialize environment E;
**for** *each episode (training cycle)* **do**
  **while** $s \neq s_{end}$ **do**
    **if** $random(0,1) \leq \xi$ **then**
      a = random();
    **else**
      a = $\max_{a'} Q(s,a')$;
    $(r, s') = \text{E.step}(s, a)$;
    $Q(s,a) = Q(s,a) + \alpha(r + \gamma \max Q(s',a') - Q(s,a))$;
    $s = s'$;
return $Q$;

$Q$-learning, one algorithm of RL, attempts to learn the state-action pair value $Q(s,a)$, whose value represents the maximum discounted reward for the pair of state $s$ and action $a$. The update rule of $Q(s,a)$ is generally described as follows:

$$Q(s,a) = Q(s,a) + \beta(r + \gamma \max Q(s' a' - Q(s,a)) \quad (5)$$

where $\beta$ is a learning rate that controls how much the difference between previous $Q$-value and the new $Q$-value is taken into account, and $\gamma$ is a factor of future rewards discount. The $Q$-value keeps on iterating based on Equation (5) until it converges [25]. This is the classical $Q$-learning algorithm as shown in Algorithm 1.

**Table 1.** Terminologies in reinforcement learning.

| Terminology | Explanation |
|---|---|
| $a$ | The current action |
| $a'$ | The next action |
| $s$ | The current state |
| $s'$ | The next state |
| $r$ | A reinforcement signal(reward or penalty) |
| $A$ | The action set |
| $S$ | The state space |

In the classical case, the *Q*-function is implemented with a table, in which there is one entry for each pair of state and action. It requires that the space of states and actions must be limited. However, the aperiodic tasks coming randomly suggest that the state space *S* is continuous and infinite in our case. In practice, the generic Q-learning approach is impractical without any generalisation as the *Q*-value is estimated separately for each action. The neural network can be a good solution to the problem of unlimited state and can be a good approximation to *Q*-value function. The above Q-learning rule thus can be directly implemented in a neural network [26]. Since no direct assignment of *Q*-values can be made in the neural network, an error function is introduced to measure the difference between the current *Q*-value $Q_c(s,a)$ and the expected *Q*-value $Q_t(s,a)$. For example, a squared-error can be defined as

$$e = (Q_c(s,a) - Q_t(s,a))^2. \tag{6}$$

As a result, gradient descent techniques (e.g. the back-propagation learning rule) can be applied to adjust the weights of a neural network in order to minimise the error *e*. In this work, Q-learning with neural-network was employed to solve the scheduling optimisation problem. The detail solution to this problem will be introduced in the next section.

## 4. Mixed-tasks scheduling optimisation framework with Q-learning

The architecture of Q-learning with neural network realising RL framework in this specific problem is illustrated in Figure 2. Similar to the original RL framework, there are two interacting members: the agent and the environment. Each member may have many elements with independent functions. The task-set that collects and stores the task sequence can provide the information of the state combined with the execution result of last scheduling. The rest of them related to the action, state and reward of Q-learning will be introduced as follows.

### *4.1. Action*

The actions to be selected include four algorithms: the BG, DS, SS and OFP. Among these four algorithms, the BG algorithm is the most straightforward one, as the aperiodic tasks can only be executed when the processor is idle (i.e. when no periodic tasks are running). The DS algorithm and the SS algorithm belong to the category of polling-server-based algorithms. The basic idea of the DS algorithm is to allocate a specific server task with a capacity to execute the aperiodic tasks. If there is no pending aperiodic task when the server task is enabled, a certain strategy ensures the server to maintain the capacity so that the aperiodic task can be processed in a timely manner when it arrives. In addition,

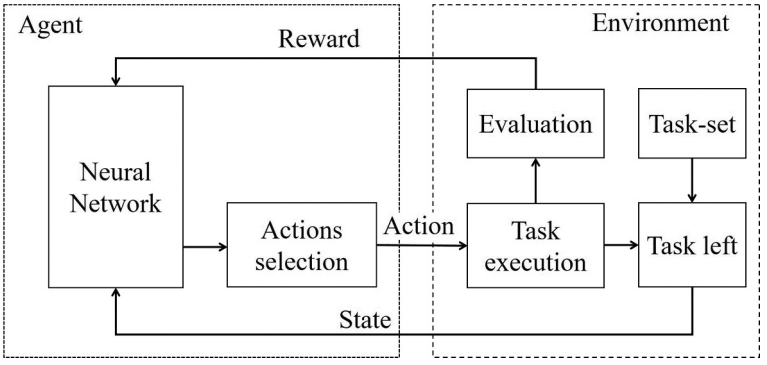


**Figure 2.** The framework of Q-learning with neural network.

the server periodic task replenishes the whole capacity at the start of each period. The SS algorithm also needs a specific task to serve aperiodic tasks. Compared with the DS algorithm, the SS algorithm does not add all the aperiodic tasks execution time to the server at the beginning of each period but keeps the ability to determine when and how much the server task replenishes the capacity based on a certain strategy that is related to the execution of the aperiodic task. The OFP algorithm records and computes the related date of every period and aperiodic task to "stealing slack" from the periodic tasks. Although proved to be optimal for the particular fixed priority, the OFP algorithm is expensive in both memory space and computing time. The characteristics of the above four algorithms are shown as in Table 2.

One often uses the greedy strategy to choose an action from the action set $A$, when $Q(s,a)$ has converged to the true value. However, most researchers adopt the $\xi$-greedy exploration, which selects an action in the training process with a probability $\xi$. Actually, it is more reasonable to decrease $\xi$ over time since the greedy selection is more believable while the random probability becomes smaller in the iterations. Here we set the parameter decreasing rule during scheduling according to the following equation:

$$\xi = 0.99 * \xi. \tag{7}$$

**Table 2.** The feature of the four algorithms that are taken as actions.

| | Performance | Computation | Memory | Realizatiom |
|---|---|---|---|---|
| BG | Poor | Excellent | Excellent | Excellent |
| DS | Good | Excellent | Excellent | Excellent |
| SS | Good | Good | Good | Good |
| OFP | Excellent | Poor | Poor | Poor |

### 4.2. State and neural network

The state represents how many tasks are still in waiting state for the execution, including the tasks left by the last execution and new tasks that are coming. Here only the aperiodic tasks are necessary to be emphasised in the scheduling process as the periodic tasks do not change over time. Similarly, each aperiodic task $T$ only has two meaningful attributes $T_{arr}$ and $T_{exe}$. For instance, given two periodic tasks and six aperiodic tasks at most in one hyper-period $H$, the continuous state space is a 12-dimensional space. Note that the data of the aperiodic tasks are sorted with certain rules, e.g. FIFO, to guarantee the consistency of the input orders.

The main reason we adopt the neural network is that it is able to represent continuous and large state space and perform generalisation. Here, we choose the popular multilayer perceptron (MLP) [27] as the neural network for the training use. As a feedforward neural network, the MLP is capable of approximating general functions (e.g. continuous and integrable functions) [28]. In this work, we use a three-layer perceptron network with basic structures demonstrated by Figure 3 to approximate the true value of $Q$-value. Typically, we need as many perceptron networks as actions to train the $Q$-value. However, we take only the expression of the state as input and alternatively take the Q-value of all possible actions as output as shown in Figure 4. The advantage of this approach is that only one forward pass through the network is needed when we perform a $Q$-value updating or pick the action with the highest Q-value. And then $Q$-values for all actions are readily available. The action taken in each state, i.e. the algorithm chosen at each hyper period, is chosen according to the maximum Q-value principle, that is, $a = \max_{a'} Q(s, a')$

### 4.3. Reward

The RL process should be able to provide the immediate reward, which can be determined by the degree of the completion of tasks, the complexity of

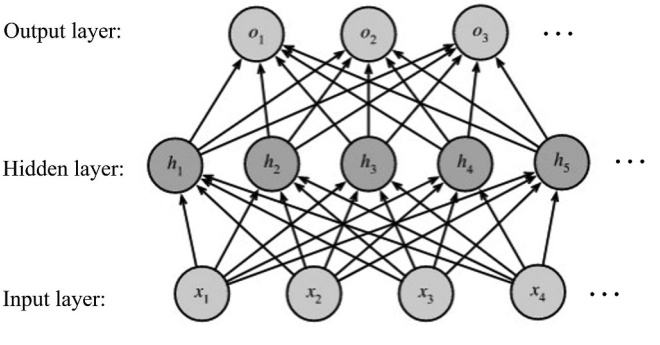


**Figure 3.** The three-layer perceptron network with basic structures.

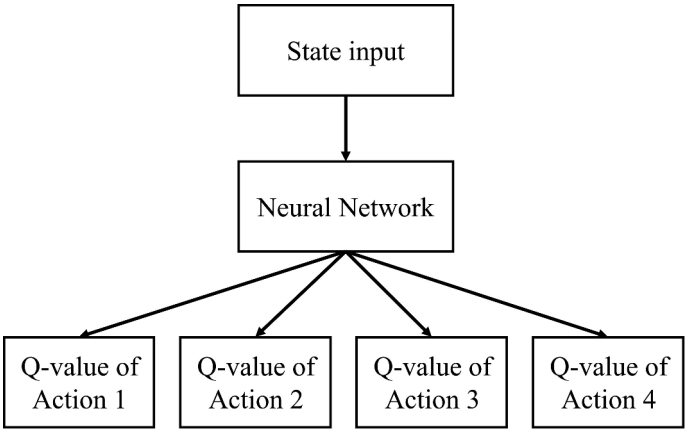


**Figure 4.** Schematic of the network structure.

computation, the requirement of memory and so on. Here we evaluate the periodic task and aperiodic task independently, and then combine them as the schedule result.

For periodic tasks, the goal of scheduling is to guarantee all the tasks are finished before deadline. The deadline is the time stamp when the next periodic task comes. If all periodic tasks are scheduled successfully, we give a positive reward $p$, otherwise, a negative reward. That is

$$r_p = \begin{cases} p & \text{all periodic task finished} \\ -p & \text{others.} \end{cases} \tag{8}$$

For aperiodic task $i$, $i = 1, 2, \cdots, w$, define

$$M^i = \begin{cases} N^i / T^i_{exe} & \text{aperiodic task finished} \\ \delta * T^i_{com} / T^i_{exe} & \text{others,} \end{cases} \tag{9}$$

where $\delta$ is an adjustment factor, $N^i$, $T^i_{exe}$ and $T^i_{com}$ represent the response time, the total execution time and the finished time of the aperiodic task, respectively. Then, every aperiodic task can be evaluated by the reward $r_{ap}$ simply as follows:

$$r_{ap} = \eta * (1 - l) * \sum_{i=1}^{w} M^i, \tag{10}$$

where $\eta$ is also an adjustment factor and $l$ reflects the system load for executing the task-set and scheduling.

**Table 3.** The complexity index of different algorithms in memory and computation.

| | BG | DS | SS | OFP |
|---|---|---|---|---|
| $r_c$ | −3 | 0 | 1 | 3 |

Initialize environment E;
Initialize $Q$ network with random weights;
Initialize state $s, \xi$;
**for** *each episode (training cycle)* **do**
  **while** $s \neq s_{end}$ **do**
    **if** $random \leq \xi$ **then**
      $a = \text{random}()$;
    **else**
      $a = \max_{a'} Q(s, a')$;
    $(r, s') = \text{E.step}(s, a)$;
    $Q_c(s, a) = Q.\text{run}(s, a)$;
    $Q_t(s, a) = Q_c(s, a) + \beta(r + \gamma \max Q(s', a') - Q_c(s, a))$;
    training $Q$ network with $Q_t(s, a), Q_c(s, a)$;
    $s = s'$;
    $\xi = \xi * 0.99$;
return $Q$ network;

Let $r_c$ represent the complexity index which comprehensively considers the cost of computation and requirements of memory. The value of $r_c$ for different algorithms can be defined as in Table 3, according to the estimation of the characteristics of the algorithms. The final reward can then be defined as

$$r = r_p + r_{ap} + r_c. \tag{11}$$

Algorithm 2 shows the training procedure of the proposed framework.

## 5. Experiments

### 5.1. The ATO system

In this work, the ATO system is chosen as the target mixed-task real-time control system for scheduling optimisation. We elaborate the system to analyse the requirements of the mixed-task scheduling problem. As a typical real-time control system, the ATO system automatically drives a train under various conditions. The main blocks contained in the ATO system include the trip planning unit, the automatic control unit, the parameter adaption unit and other necessary assistant modules. The trip planning unit derives a specific route according to the digital railway map for a specific trip. It also creates a reference trip plan as the optimisation problem stated above and performs temporary trip planning for dynamically adjusted trip under temporary situations. The automatic control unit serves the real-time control tasks with

**Table 4.** The number of neurons in different layers.

| | Input layer | Hidden layer | Output layer |
|---|---|---|---|
| Number | 12 | 6 | 4 |

specified control algorithms. The parameter adaptation unit adjusts the parameters according to the learned factors of certain trip while the input key parameters are deviated from the right value. The assistant functions include the operation log, failure recovery and other services.

### 5.2. The ATO system task analysis

In this work, the experiments are performed on the ATO system that we developed for a given model of locomotives. The hardware-in-loop testing platform shown in Figure 5 is used to simulate all the situations that the ATO system may face. We hereby employ the testing platform to analyse the parameters of tasks in the core processing board.

The tasks that we collect from the core processing board of the ATO system are composed with both periodic tasks and aperiodic tasks. The periodic tasks mainly include the tasks that handle real-time control and heartbeat feedback, while the aperiodic tasks include the handle events of exception, railway transit signal, temporary trip planning, route extraction and other service tasks, as listed below:

$$\textit{ATO tasks}\begin{cases}\text{periodic tasks}\begin{cases}\text{realtime control}\\ \text{heartbeat feedback}\end{cases}\\ \text{aperiodic tasks}\begin{cases}\text{exception control}\\ \text{transit signal handle}\\ \text{temporary optimization}\\ \text{service tasks}\end{cases}\end{cases}$$

Specifically, there are two periodic tasks with a period of 10 and 14, respectively, and four aperiodic tasks at most in one hyper-period $H$ in such case. The execution time of periodic tasks is 1 and all periodic tasks are ready at the beginning of hyper-period $H$. According to their execution times, the aperiodic tasks can be divided into two types: the short aperiodic tasks that can be processed in a hyper-period and the long aperiodic tasks that may not be fitted in one hyper-period. And, $6/7$ of the collected task-sets for training are short aperiodic tasks. The different proportion of the short aperiodic tasks represents different scenarios. Both short aperiodic tasks and the long aperiodic tasks have a random arrival time. All the mixed tasks in the core processing board can be manually scheduled without violation of real-time constraints. Next, we take the tasks for experiments.

### 5.3. Experimental design

According the task analysis in the last subsection, we simulate certain task-sets that are in accord with the real working circumstances in the core processing board. Tasks with the format of $K(T_{per}, T_{arr}, T_{exe})$ are generated for simulating the

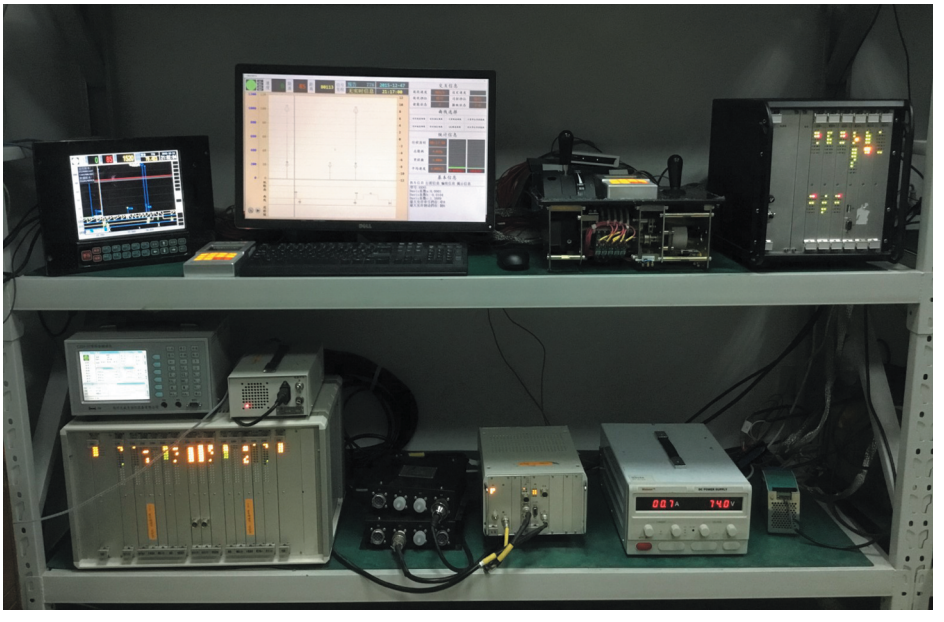

**Figure 5.** The hardware-in-loop testing platform that we used to analyse possible tasks in the core processing board.

process of the tasks while the parameter $T_{end}$ can be calculated. Here we randomly generate three task-sets, namely Set 1, Set 2 and Set 3, with the proportion of the short aperiodic tasks over the total tasks be $5/6$, $6/7$ and $7/8$, for testing purposes. In the three task-sets, Set 2 is to mimic the tasks in the core processing board of the developed ATO system while Set 1 and Set 3 are used for comparisons. These data-sets with different task loads will serve as the training data for exploring proper scheduling algorithms.

The proposed mixed-tasks scheduling algorithm optimisation framework is carried out for verification. In the experiment, we choose the four popular scheduling algorithms mentioned above, i.e. the BG, DS, SS and OFP algorithms, for comparison. The training work is carried out on a server computer, while the evaluation of the complexity index is implemented on the target board, i.e. the core processing board, with an ARM CPU of 666 MHz.

### *5.4. Training*

The training process was carried out under the above environment with the convergence behaviours assessed in each episode, i.e. an iteration of training cycle. As the RL algorithm learns the experience through repeating iterations, relative data can be used to evaluate the progress of the training process. The total reward in every cycle was observed first. It tends to be rather messy in the training process, as shown in Figure 6. This is mainly due to the total reward is

a temporary evaluation to a subprocess and make sense for the learning process [29].

The action-value function $Q$, which provides an estimation of how much discounted reward the agent can obtain by following its policy from any given state, serves as the direct indicator for the training process. The training process can be terminated once the value of the $Q$ function converges. For simplicity, the total of the maximum prediction $Q$-value for certain set of the states in every episode are counted, as shown in Figure 7. It can be seen that the predicted $Q$-value is rather random at the beginning, but increases to be more smooth than the total reward obtained by the agent. When the number of the episode is over $4,000$, the selected $Q$-value is converging to a stable line. It suggests that the neural network in the scheduling optimisation framework offers good convergence property although a theoretical guarantee for convergence is still missing. The optimisation performance can then be evaluated compared to the original four scheduling algorithms, e.g. the BG, DS, SS and OFP.

### 5.5. Evaluation and analysis

The scheduling performance is evaluated in terms of the overall response time of the aperiodic tasks, the overall scheduling cost, as well as the value of the objective function. Specifically, the overall response time of the aperiodic tasks is the sum of the response time of the aperiodic tasks in each hyper-period, i.e. $\sum_{j=1}^{m} R_j^i$, where $R_j^i$ is defined in Equations 2 and 3. The overall scheduling cost is

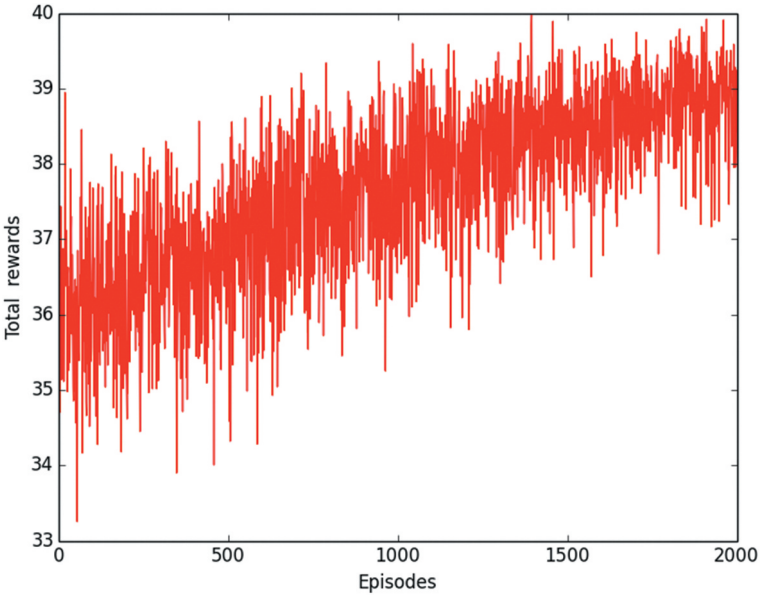


**Figure 6.** The variation of the total rewards with the episodes.

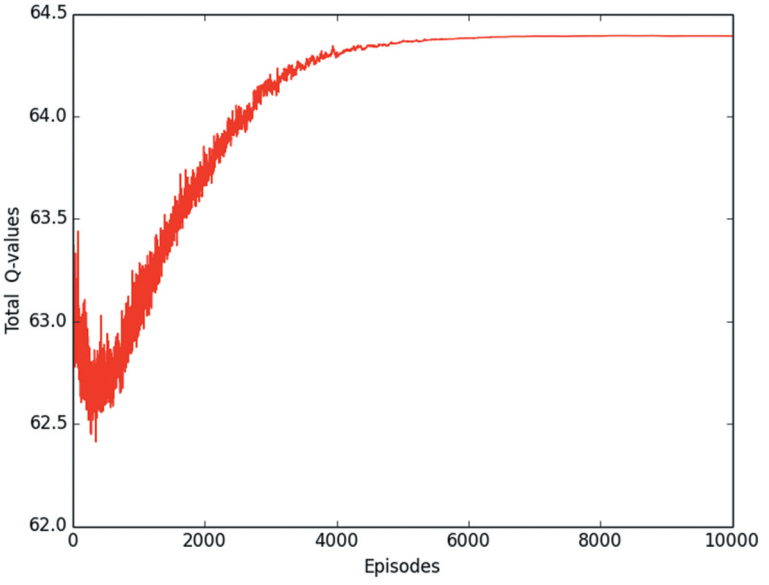


**Figure 7.** The convergence of the total *Q*-value with the episodes.

the sum of the complexity index considering the scheduling overhead and post-processing time in each hyper-period, i.e. $\sum_{j=1}^{m} C_j^i$. Since we have set the weight $a = 1$ in Equation 4), the value of the objective function is equal to $\sum_{j=1}^{m}(R_j^i + C_j^i)$. The comparison results are listed as in Tables 5–7, respectively. We also evaluated the experiments result in view of the space complexity (i.e. memory cost) as listed in Table 8.

The experimental results suggest that the proposed framework can either get better response time or less scheduling cost than a single fixed algorithm. The value of the objective function in Table 7 clearly shows the advantage of the proposed framework considering both response time of aperiodic tasks and the scheduling cost. The proposed approach always searches for the best suitable algorithm for a certain situation and is rather flexible for different mixed-tasks in view of the task loads and combinations.

## 6. Conclusion

Different scheduling algorithms are generally associated with varying scheduling cost and system performance. Aiming at optimising the scheduling

**Table 5.** The response time of the different algorithms in scheduling the three task-sets.

| | BG | DS | SS | OFP | Proposed |
|---|---|---|---|---|---|
| Set 1 | 1956 | 1324 | 1546 | 472 | 1053 |
| Set 2 | 2072 | 1564 | 1801 | 507 | 1246 |
| Set 3 | 2183 | 2175 | 2516 | 679 | 1635 |

**Table 6.** The complexity indexes (the scheduling overhead and post-processing time) of the algorithms on the three task-sets.

| | BG | DS | SS | OFP | Proposed |
|---|---|---|---|---|---|
| Set 1 | 974 | 1418 | 1238 | 2782 | 1592 |
| Set 2 | 1026 | 1468 | 1278 | 2790 | 1664 |
| Set 3 | 1122 | 1534 | 1298 | 2978 | 1620 |

**Table 7.** The value of the optimisation objective on the three task-sets.

| | BG | DS | SS | OFP | Proposed |
|---|---|---|---|---|---|
| Set 1 | 2930 | 2742 | 2784 | 3254 | 2645 |
| Set 2 | 3098 | 3032 | 3079 | 3297 | 2910 |
| Set 3 | 3305 | 3709 | 3814 | 3657 | 3255 |

**Table 8.** The space complexity of the different algorithms.

| Algorithms | BG | DS | SS | OFP | Proposed |
|---|---|---|---|---|---|
| Space complexity | o(1) | o(1) | o($n$) | o($n$) | o(1) to o($n$) |

considering both scheduling cost and the scheduling performance in a real-time control system, this paper proposes a Q-learning framework to elevate system performance by selecting proper scheduling algorithms for the mixed-tasks according to the dynamic task loads and compositions. A computation-efficient three-layer perceptron network is chosen to perform the Q-learning process. Experimental results suggest that the proposed framework is rather flexible and fit for different mixed-tasks in view of the task loads and combinations, making it effective for deriving scheduling solutions for highly complex mixed-task real-time systems.

## Disclosure statement

No potential conflict of interest was reported by the authors.

## ORCID

Tianchuang Meng http://orcid.org/0000-0001-5774-624X

## References

[1] Sha L, Abdelzaher T, Årzén KE, et al. Real time scheduling theory: a historical perspective. Real-Time Syst. 2004;28(2–3):101–155.
[2] Ning B. Advanced train control systems. Southampton, UK: WIT press; 2010.
[3] Shi Q, Zhang H. Fault diagnosis of an autonomous vehicle with an improved SVM algorithm subject to unbalanced datasets. IEEE Trans Ind Electron. 2020. DOI:10.1109/TIE.2020.2994868

[4] Chen J, Shuai Z, Zhang H, et al. Path following control of autonomous four-wheel-independent-drive electric vehicles via second-order sliding mode and nonlinear disturbance observer techniques. IEEE Trans Ind Electron. 2020;68(3):2460–2469.
[5] Liu CL, Layland JW. Scheduling algorithms for multiprogramming in a hard-real-time environment. J ACM (JACM). 1973;20(1):46–61.
[6] López JM, Díaz JL, García DF. Utilization bounds for EDF scheduling on real-time multiprocessor systems. Real-Time Syst. 2004;28(1):39–68.
[7] Buttazzo G. Hard real-time computing systems: predictable scheduling algorithms and applications. Vol. 24. Berlin, German: Springer Science & Business Media; 2011.
[8] Lehoczky JP, Ramos-Thuel S. An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems. RTSS. Vol. 92. 1992.
[9] Lehoczky JP. Enhanced aperiodic responsiveness in hard real-time environments. Unknown Host Publication Title. IEEE. 1987:261-270.
[10] Sprunt B, Sha L, Lehoczky J. Aperiodic task scheduling for hard-real-time systems. Real-Time Syst. 1989;1(1):27–60.
[11] Strosnider JK. Highly responsive real time token rings. Ph.D. Thesis, Carnegie Mellon Univ., Pittsburgh, USA. Aug. 1988.
[12] Mehariya S, Songara D, Mitra V, et al. An approach for a reduced response time and energy consumption in mixed task set using a priority exchange server. 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI). IEEE; 2014.
[13] Tanaka K. Adaptive total bandwidth server: using predictive execution time. International Embedded Systems Symposium. Berlin, Heidelberg: Springer; 2013.
[14] Shin D, Kim J. Dynamic voltage scaling of mixed task sets in priority-driven systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 2006;25(3):438-453.
[15] Digalwar M, Gahukar P, Mohan S. Design and development of a real time scheduling algorithm for mixed task set on multi-core processors. 2014 Seventh International Conference on Contemporary Computing (IC3). IEEE; 2014.
[16] Jun H, Yufang S. An algorithm to improve the responsive performance for scheduling soft-aperiodic tasks. J Software. 1998;9(10):721–727.
[17] Burns A, Davis R. Mixed criticality systems-a review. Department of Computer Science, University of York, York, England: Tech Rep; 2013.
[18] Ghor HE, Chetto M, Chehade RH. A real-time scheduling framework for embedded systems with environmental energy harvesting. Comput Electr Eng. 2011;37(4):498–510.
[19] Mohammadi A, Akl SG. Scheduling algorithms for real-time systems. School of Computing, Queens University, Kinston, Canada: Tech Rep; 2005.
[20] Cho S, Lee SK, Ahn S, et al. Efficient real-time scheduling algorithms for multiprocessor systems. IEICE Trans Commun. 2002;85(12):2859–2867.
[21] Mishra G, Gurumurthy K. Dynamic task scheduling on multicore automotive ecus. Int J VLSI Design Commun Syst. 2014;5(6):1.
[22] Zeng S, He B, Jiang J. A scheduling algorithm for synchronization task in embedded multicore systems? J Comput Inf Syst. 2014;10(19):8531–8541.
[23] Jiang J, Zeng S, Qiu B. A task scheduling model and method for control-oriented multicore systems. The 27th Chinese Control and Decision Conference (2015 CCDC). IEEE; 2015.
[24] Wang YC, Usher JM. Application of reinforcement learning for agent-based production scheduling. Eng Appl Artif Intell. 2005;18(1):73–82.
[25] Watkins CJ, Dayan P. Q-learning. Mach Learn. 1992;8(3–4):279–292.

[26] Gaskett C, Wettergreen D, Zelinsky A. Q-learning in continuous state and action spaces. Australasian Joint Conference on Artificial Intelligence. Berlin, Heidelberg: Springer; 1999.
[27] Linggard R, Myers D, Nightingale C. Neural networks for vision, speech and natural language. Vol. 1. Berlin, German: Springer Science & Business Media; 2012.
[28] Hornik K, Stinchcombe M, White H. Multilayer feedforward networks are universal approximators. Neural Networks. 1989;2(5):359–366.
[29] Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning. arXiv Preprint. 2013;arXiv:13125602.