Energy-Efficient Scheduling for Real-Time Systems Based on Deep Q-Learning Model

Qingchen Zhang, Man Lin, Laurence T. Yang, Zhikui Chen, and Peng Li

Abstract—Energy saving is a critical and challenging issue for real-time systems in embedded devices because of their limited energy supply. To reduce the energy consumption, a hybrid dynamic voltage and frequency scaling (DVFS) scheduling based on Q-learning (QL-HDS) was proposed by combining energy-efficient DVFS techniques. However, QL-HDS discretizes the system state parameters with a certain step size, resulting in a poor distinction of the system states. More importantly, it is difficult for QL-HDS to learn a system for various task sets with a Q-table and limited training sets. In this paper, an energy-efficient scheduling scheme based on deep Q-learning model is proposed for periodic tasks in real-time systems (DQL-EES). Specially, a deep Q-learning model is designed by combining a stacked auto-encoder and a Q-learning model. In the deep Q-learning model, the stacked auto-encoder is used to replace the Q-function for learning the Q-value of each DVFS technology for any system state. Furthermore, a training strategy is devised to learn the parameters of the deep Q-learning model based on the experience replay scheme. Finally, the performance of the proposed scheme is evaluated by comparison with QL-HDS on different simulation task sets. Results demonstrated that the proposed algorithm can save average 4.2% energy than QL-HDS.

Index Terms—Energy consumption, stacked auto-encoder, dynamic voltage and frequency scaling, Q-learning

1 Introduction

Recently, an explosive growth in computer technology and internet of things has promoted the development of embedded systems [1], [2], [3], [4], [5]. An embedded system is developed for a special function by combining computer hardware and software that is usually fixed in capability or programmable [6]. Nowadays, embedded systems have been successfully applied in many areas, such as industrial internet of things application, mobile cloud computing and big data processing [7], [8], [9], [10], [11], [12], [13], [14], [15]. With the rapid development of embedded systems in last decades, smart mobile devices have become ubiquitous in our life [16]. Representative mobile devices include smart phones, tablets and point of sales that are profoundly changing our lifestyle. Currently, the mobile devices are providing more and more functions, involving video playing, WEB browsing, WIFI communication, game and email. A large number of functions in the mobile devices result in the increase of power consumption [17].

The power of smart mobile devices is mainly supplied by the battery in most cases. However, the power stored in the battery is usually limited, causing the constraints in the reliability and life of the mobile devices [18]. Generally, two methods to prolong the usage time of the battery are to add the battery capacity and to reduce the power consumption. Unfortunately, it is very difficult to increase the power capacity of the battery because the power capacity of the battery depends on the chemical characteristics of the battery materials. In the past thirty years, the power capacity of the battery is only added by 3 to 4 times. In addition, increasing the power capacity of the battery requires to grow the volume, weight and cost, which will reduce the mobility and convenience of the mobile devices. Therefore, improving the energy efficiency to reduce the power consumption has become an important research topic for smart mobile devices with embedded systems [19], [20], [21].

Typically, most power of smart mobile devices is consumed by processors including CPU and GPU. For example, the energy consumption count of various components of a DELL XPS M1330 labtop is shown in Fig. 1 [22].

From Fig. 1, CPU and GPU account for more than 50 percent power consumption of the labtop at its maximum settings.

Currently, most of tasks run on the processors in embedded systems are real-time tasks that typically have timing constraints [23]. Specially, they must be accomplished before their deadlines, which is safety critical for some key areas such as aircraft controls. In addition, the tasks in the embedded systems are usually periodic. Generally speaking, the periodic tasks are repeatedly run at regular intervals. Therefore, this paper focuses on the energy-efficient scheduling for periodic multi-tasks in real-time systems. Specially, this paper aims to design the

• Q. Zhang and L. T. Yang are with the School of Electronic Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China, and the Department of Computer Science, St. Francis Xavier University, Antigonish, NS B2G 2W5, Canada. E-mail: qzhang@stfx.ca, ltyang@gmail.com.
• M. Lin is with the Department of Computer Science, St. Francis Xavier University, Antigonish, NS B2G 2W5, Canada. E-mail: mlin@stfx.ca.
• Z. Chen and P. Li are with the School of Software Technology, Dalian University of Technology, Dalian 116620, China. E-mail: zkchen@dlut.edu.cn, lipeng2015@mail.dlut.edu.cn.

*Manuscript received 11 June 2017; revised 29 July 2017; accepted 20 Aug. 2017. Date of publication 24 Aug. 2017; date of current version 6 Mar. 2019. (Corresponding author: Laurence T. Yang.)
Recommended for acceptance by M. Qiu and S.-Y. Kung.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TSUSC.2017.2743704*

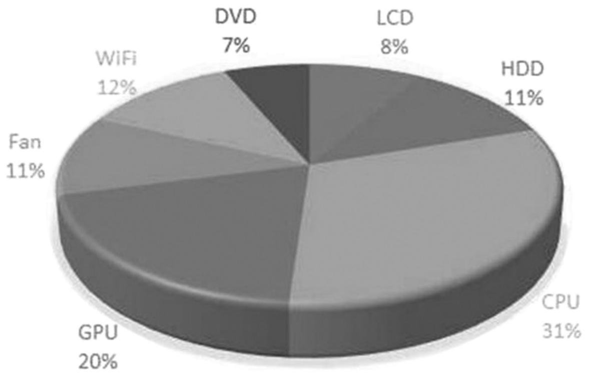


Fig. 1. Energy consumption distribution of one Dell labtop.

energy scheduling scheme to reduce as much as power consumption.

Many methods have been presented to reduce the energy consumption in recent years. The most widely used two methods are dynamic power management (DPM) [24] and dynamic voltage and frequency scaling (DVFS) which have been successfully applied to the scheduling in the embedded real-time systems [25]. DPM adjusts the work modes according to the system workload to minimize the power consumption [24]. In detail, DPM switches the system to the sleep mode with the lowest energy to save energy when the system is in the idle intervals.

However, DPM will consume additional energy due to the mode-switching. Therefore, it is worth to switch the mode only if the idle interval is long enough.

Generally, the energy consumption of a processor includes the dynamic energy consumption because of switching activities and the static energy consumption because of the leakage current. The dynamic consumption usually accounts for the major part of the total energy consumption. It is approximately proportional to the square of operating voltage and the clock frequency [22]. DVFS dynamically adjusts the voltage and the frequency of the system to reduce the energy consumption. The most representative DVFS technologies include cycle-conserving (CC), look-ahead (LA), dynamic reclaiming algorithm (DRA) [26], [27]. They adjust the voltage and the frequency depending on the system state parameters for instance dynamic slack and system utilization. Power efficiency of different DVFS technologies varies under the different system states. Therefore, any single DVFS technology could not guarantee the minimal power consumption under all system states [22], [28].

To combine different DVFS technologies, i.e., CC, LA, DRA and AGR, for power consumption reduction, Islam and Lin [22] proposed a Q-learning-based hybrid scheduling algorithm (QL-HDS) which uses the dynamic slack (DS) and the system utilization (SU) as the system state. QL-HDS defines a function (called Q-function) to calculate a Q-value for each DVFS technology depending on the current system state. Furthermore, the DVFS technology with the smallest Q-value is selected to adjust the voltage and the frequency in the next hyperperiod. Although simulations validated that QL-HDS outperformed any single DVFS technology in reducing the power consumption, it still has two limitations. First, QL-HDS discretizes the values of the dynamic slack and the system utilization with the step size of 0.1, thus it could not learn the Q-values with the continues state parameters. However, most of the values of the system state parameters are concentrated in a certain range such as $[0.4, 0.7]$ for a specific task set. Therefore, DL-HDS could not distinguish the system states effectively. For example, QL-HDS views two system sates $\{SU = 0.407, DS = 0.514\}$ and $\{SU = 0.407, DS = 0.549\}$ as the same one system state with $\{SU = 0.4, DS = 0.5\}$. Second, it is difficult for QL-HDS to learn the Q-values of each DVFS technology for all the system states on limited training sets, leading to failure in searching the Q-table when a new system state appears.

In this paper, an energy-efficient scheduling algorithm based on the deep Q-learning model is proposed for saving energy. Specially, a deep Q-learning model is designed to learn the Q-values of each DVFS technology for different system sates by combining a stacked auto-encoder (SAE) [29] and a Q-learning model [30]. The stacked auto-encoder is a typical deep learning model and furthermore it has been successfully used in many applications such as image classification and natural language processing [31], [32]. More importantly, it has been used to construct the deep reinforcement learning model for traffic prediction [30]. The deep Q-learning model can distinguish the system states effectively by learning the Q-values of each DVFS technology under the continues state parameters. More importantly, the deep Q-learning model uses the stacked auto-encoder model to replace the Q-function for learning the Q-value of each DVFS technology for any system state after training the parameters. Furthermore, a training scheme is devised to learn the parameters of the deep Q-learning model, based on the experience replay strategy. Finally, the performance of the proposed algorithm is evaluated by comparing with QL-HDS on different simulation task sets. Results demonstrated that our proposed algorithm can save average 4.2 percent energy than QL-HDS.

In summary, there are three contributions in the paper, listed as:

- A deep Q-learning model is proposed for the energy-efficient scheduling in the real-time systems (DQL-EES) by combining a stacked auto-encoder and a Q-learning model. The stacked auto-encoder is substituted for the Q-function to learn the Q-value of each DVFS technology for any system state that is described by the dynamic slack and the system utilization.
- A training scheme is designed to learn the parameters of the deep Q-learning model based on the experience replay strategy. Specially, the experience tuples are collected and stored into the relay memory as the training samples.
- Several experiments are conducted to evaluate the performance of the proposed model on different simulation task sets by comparison with QL-HDS. Results demonstrate that the proposed method can reduce average 4.2 percent energy consumption than QL-HDS on the simulation task sets.

In the reminder of the paper, the related works on DVFS techniques are presented in Section 2. The system models are listed as the preliminary of the proposed model in Section 3 and the deep Q-learning model for energy-efficient scheduling is illustrated in Section 4. The training method for the parameters of the deep Q-learning model is described in

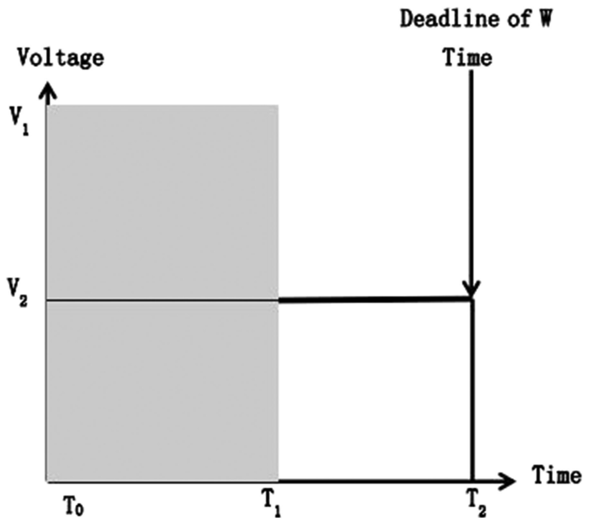


Fig. 2. One example of the DVFS technique.

Section 5 and the simulation results are reported in Section 6. Finally, the paper is concluded in Section 7.

2 Related Works

In this section, the related works about DVFS techniques and the learning-based DVFS techniques are presented. The DVFS techniques are presented first, followed by the learning-based DVFS techniques.

2.1 DVFS Technologies

Generally, the power consumption of processors mainly includes static energy consumption and dynamic energy consumption. The dynamic energy consumption is typically resulted in switching activities, approximately proportional to the square of the operating voltage and the clock frequency. The dynamic voltage and frequency scaling reduces the dynamic energy consumption by adjusting the operating voltage and the clock frequency dynamically according to the system state such as the system utilization, workload and the dynamic slack. Fig. 2 shows an example of the DVFS technique.

Assume that the deadline of the task W is T_2. T_1 is the actual completion time when W is performed at the high operating voltage of V_1. S_1 denotes the idle interval after W is completed and the processor does not consume energy during S_1. T_2 represents the actual completion time when W is executed at the low operating voltage of V_2. Assume that $V_2 = V_1/2$, $f_2 = f_1/2$ and $T_2 - T_1 = T_1 - T_0$, where f_1 and f_2 represent the clock frequency when W is executed at the operating voltage of V_1 and V_2, representatively. Under the ideal condition, decreasing the operating voltage from V_2 to V_1 for W can save 75 percent energy.

DVFS has been proved to be an effective technology for energy consumption reduction. In the past few years, many DVFS techniques have been developed for energy optimization. The most representative DVFS techniques are static, cycle-conserving (CC) and look-ahead (LA) that are based on the earliest deadline first (EDF) scheduling strategy [27].

The static scheme calculates the utilization of the scheduled task set according to offline parameters including the periods and execution time, and furthermore sets the voltage and the frequency accordingly. The static scheme does not adjust the voltage and the frequency during the execution set. However, the actual execution time might vary in different periods, so the static scheme could not usually achieve the energy optimization. To tackle this problem, CC adopts the notation of the dynamic utilization to adjust the voltage and the frequency. Specially, CC determines the system utilization depending on the worst case execution time (WCET) at the beginning of every period, and then updates the system utilization dynamically during the scheduling of the tasks depending on the actual execution time (AET). Since WCET is always longer than AET, CC sets a high clock frequency at the beginning of each period and decreases the frequency gradually when tasks are completed. Different from CC, LA sets a low clock frequency at the beginning of each period and performs as much work as possible under the low frequency to save energy. When the deadline comes near, LA increases the clock frequency gradually to guarantee the deadline.

More recently, Aydin et al. [26] developed two DVFS techniques, called dynamic reclaiming algorithm (DRA) and aggressive speed adjustment (AGR), representatively. DRA is according to detecting early completions and decreasing the clock frequency for other tasks to save the energy while guaranteing the deadlines. Specially, DRA orders a data structure called α-queue based on the improved earliest deadline first (EDF*) scheduling strategy. Once one new task arrives, its WCET is put into α-queue at S_{opt} frequency. The utilization of the scheduled task set stays the same at S_{opt} frequency. AGR is a variant of DRA. When multiple tasks exist in the α-queue and they are assumed to finish before the next task arrival time, AGR transfers the processor time among the tasks in the α-queue. Generally, AGR could reduce the power consumption if the idle time is longer because of the convex correlation over the dynamic energy and the clock frequency. However, if the static energy consumption is comparable with the dynamic energy consumption, AGR will consume more power. In addition, DRA and AGR might result in more power dissipation since the system is always alive.

Lawitzky et al. [33] developed a DVFS scheme by combining the real time with the power management in real-time systems. It appends the number of the dynamic slack of an accomplished task with the budget of the next task and sets the clock frequency. Lu and Guo [34] investigated the DVFS techniques for multi-core scheduling and presented a DVFS algorithm to adjust the clock frequency for every scheduled task according to the utilization of the task set and available cores.

Islam and Lin [22] compared the power efficiency of several DVFS techniques in real-time systems. They concluded that the power efficiency of each DVFS technique is significantly dependent upon the system state such as the system utilization and the dynamic slack. No single DVFS could achieve the energy optimization under all system states.

2.2 DVFS Technologies Based on Learning

The power efficiency of a DVFS technology is significantly dependent upon the different system states. To improve the robustness for energy saving for the different applications and hardware, learning-based methods have been incorporated into the system-level energy management in recent years. Specially, a good power management scheme should be able to learn the best strategy to reduce the energy consumption for different systems from previous experiences. Therefore, some DVFS technologies based on learning methods have been investigated in last few years for system-level energy management.

For example, Bhatti et al. [28] presented a scheme for interplaying of DVFS technologies in real-time systems according to online learning. This scheme adjusts the frequency and the voltage depending on power consumption and performance penalty that are learned by previous experiences. Once the dynamic slack happens to change, this scheme will take the decision to select the best expert. One similar method developed by Dhiman and Rosing uses the clock frequency levels as different experts that are also chosen according to the energy consumption and performance penalty [35]. Jung and Pedram applied the supervised learning strategy to the power management scheme for multiprocessor systems [36]. Specially, they utilize the supervised learning strategy to analyze the system state from input features and learn the best voltage-frequency configures with a precalculated policy table. However, a precalculated policy table and the limited training task sets could not cover any environment, so it is difficult to learn a system to apply to all workloads and hardware configurations.

Most recently, the reinforcement learning methods have been applied to energy optimization. For instance, a reinforcement learning-based scheme was presented to optimize the temperature and power consumption. Afterwards, a robust reinforcement learning-based scheme was developed to set the operating voltage and the clock frequency dynamically without requiring prior information of the workload. However, these methods mainly focus on the non-real-time systems. That means, they does not consider the deadlines of the scheduled tasks.

3 System Models

3.1 Task Model

In embedded systems, most of tasks are periodic and real-time tasks. A periodic real-time task is repeatedly performed at regular intervals and must be completed before its deadline. A task set T that consists of n periodic real-time tasks is usually defined by the following equation [22].

$$T = \{\tau_1(p_1, \omega_2), \tau_2(p_2, \omega_2), \ldots, \tau_n(p_n, \omega_i)\}. \tag{1}$$

In Eq. (1), τ_i denotes the ith task. Each task τ_i has a period p_i and a worst case execution time (wcet) ω_i which indicates the maximum execution time for completing the task τ_i at the highest clock frequency. In addition, the periodic real-time task has a utilization that is calculated by Eq. (2).

$$u_i = \omega_i / p_i. \tag{2}$$

Therefore, the total utilization U of the task set T is calculated by Eq. (3).

$$U = \sum\nolimits_{i=1}^{n} \omega_i / p_i. \tag{3}$$

In every period, one new job is produced by each task. For example, τ_i^j denotes the new job produced by the ith task τ_i in the jth period. For the task set T, there are many jobs in a hyperperiod (hyp) that is defined by the following equation.

$$hyp = h \times LCM(p_1, p_2, \ldots, p_n). \tag{4}$$

where LCM denotes the least common multiple of all the periods of the tasks in the task set T and h denotes the hth hyperperiod.

The jobs generated in a hyperperiod are scheduled according to their priorities. In this paper, the earliest deadline first scheduling strategy is selected to set the priority of each job. Specially, the job with the earliest period has the highest priority.

TABLE 1
Technique Constants of the 70 nm Technology

Constant	Value	Constant	Value
K_1	0.063	V_{bs0}	0
K_2	0.153	α	1.5
K_3	5.38	V_{th1}	0.244
K_4	1.83	I_{jun}	4.8×10^{10}
K_5	4.19	C_e	0.43×10^{9}
K_6	5.26	L_d	37

3.2 Energy Model

For the processors based on complementary metal oxide semiconductor (CMOS), the major energy consumption part is the dynamic energy consumption that can be calculated by the following equation [22].

$$P_{dynamic} = C_e \times V_{dd}^2 \times f, \tag{5}$$

where C_e denotes the switching capacity, V_{dd} denotes the operating voltage, and f represents the clock frequency that can be calculated by Eq. (6).

$$f = \frac{(V_{dd} - V_{th})^{\alpha}}{L_d \times K_6}, \tag{6}$$

where L_d, K_6 and α are the technique constants determined by the processor fabrication technique, and V_{th} denotes the threshold voltage represented by

$$V_{th} = V_{th1} - K_1 \times V_{dd} - K_2 \times V_{bs}. \tag{7}$$

In Eq. (7), V_{th1}, K_1 and K_2 are technique constants, and V_{bs} denotes the the body bias voltage.

The static energy consumption can be given by

$$P_{static} = V_{dd} \times I_{subn} + |V_{bs}| \times I_{jun}, \tag{8}$$

where I_{jun} denotes the reverse bias junction current and I_{subn} denotes the subthreshold leakage current that can be calculated by Eq. (9).

$$I_{subn} = K_3 \times e^{K_4 \times V_{dd}} \times e^{K_5 V_{bs}}. \tag{9}$$

In Eq. (9), K_3, K_4 and K_5 are the technique constants.

Table 1 shows the technique constants of the 70 nm process technology used in the energy model.

Therefore, if a task has the execution time T, it consumes the total energy

$$E_n = \lambda \times P_{dynamic} \times T + \rho \times P_{static} \times T, \tag{10}$$

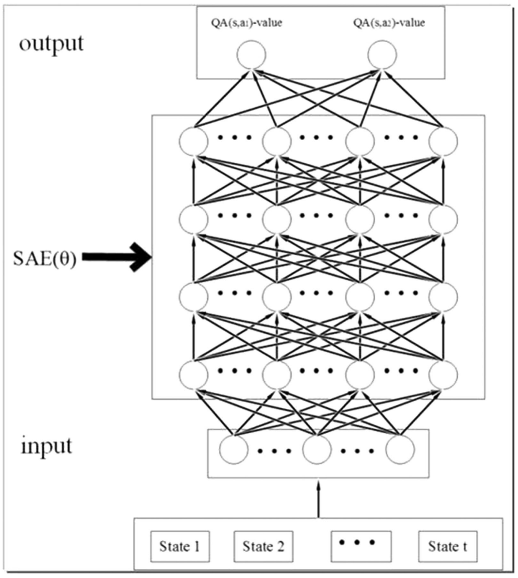


Fig. 3. Architecture of the deep Q-learning model.

where λ and ρ denote the weights and they are usually the constants determined by the processor fabrication technology.

4 Deep Q-Learning Model for Energy-Efficient Scheduling in Real-Time Systems

Given a task set T that is composed of n periodic real-time tasks and three DVFS techniques, i.e., CC, LA and DRA, the goal of this paper is to select an appropriate DVFS technique at the beginning of each hyperperiod to configure the operating voltage and the clock frequency so that the power consumption E_n is minimal while all tasks are accomplished before deadlines .

To achieve this goal, a Q-value for each DFVS technique is defined. At the beginning of each hyperperiod, the Q-value for each DVFS technique is calculated based on the current system state, and then the DVFS technique with the lowest Q-value is selected to configure the voltage and the frequency.

To calculate the Q-value for each DVFS technique, a deep Q-learning model is constructed by combining a stacked auto-encoder and a Q-learning model, as presented in Fig. 3.

The deep Q-learning model takes the system state as the input and outputs the Q-value for each DVFS technique. The stacked auto-encoder $SAE(\theta)$ is used to learn the features of each input system state and the Q-learning aims to compute the value of each action given the input system state. Specially, the system state space is defined as $S = \{S_1, S_2, \ldots, S_n\}$. Since the DVFS techniques used in our model adjust the voltage and the frequency mainly depending on the system utilization and the dynamic slack, the system utilization su and the dynamic slack ds are selected as the system state, i.e., $S_i = \{su_i, ds_i\}$.

The dynamic slack usually is calculated by

$$ds = 1 - \frac{Et_{hyp}}{\sum_{i=1}^{n}\left(\frac{hyp}{p_i} \times \omega_i\right)}, \tag{11}$$

where hyp denotes the hyperperiod, p_i and ω_i denote the period and the WCET of the ith task, respectively. In Eq. (11), Et_{hyp} denotes the sum of the actual execution time of every task in a hyperperiod, which can be calculated as

$$Et_{hyp} = \sum_{i=1}^{n}\sum_{j\in hyp} AET(\tau_i^j), \tag{12}$$

where τ_i^j denotes the jth job produced by the ith task in the corresponding hyperperiod.

The system utilization of the task set T with n periodic real-time tasks can be computed by the following:

$$su = \sum_{i=1}^{n}\frac{\omega_i}{p_i}, \tag{13}$$

In the hybrid scheduling algorithm based on reinforcement learning model (QL-HDS) proposed by Islam and Lin [22], the system state s_t is obtained by interacting the environment at the end of the previous hyperperiod and it is used to update the Q-value $Q(s_t, a_t)$ for the DVFS technique represented by a_t using the following Q-function.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_i \times [p_t(s_t, a_t) - Q(s_t, a_t)], \tag{14}$$

where $p_t(s_t, a_t)$ denotes a penalty obtained in the system state s_t with the DVFS technique a_t and $a_t(s_t, a_t)$ represents the learning rate.

In QL-HDS, the penalty is defined as the average power consumption in each hyperperiod, which can be computed by the following

$$p_t(s_t, a_t) = \frac{En(s_t, a_t)}{Et_{hyp}}, \tag{15}$$

where $En(s_t, a_t)$ denotes the total power consumption at the system state s_t using the DVFS technique.

QL-HDS uses a Q-table to store the updated Q-values for each DVFS technique under the different system states. At the beginning of a new hyperperiod, QL-HDS searches the Q-table to choose the DVFS technique with the lowest Q-value according to the current system state to configure the voltage and the frequency.

However, DL-HDS could not distinguish the system states effectively since the system state parameters are continuous. More importantly, it is challenging for QL-HDS to learn the Q-values of each DVFS technology for all the system states on the limited training task sets, leading to the failure in searching the Q-table when a new system state appears.

In the proposed model, the stacked auto-encoder model is used to approximate the Q-function as shown in Fig. 4. Specially, the deep Q-learning model proposed in the paper is stacked by multiple basic auto-encoders.

A basic auto-encoder is a neural network with three layers, i.e., input layer x, hidden layer h and output layer y, as shown in Fig. 4 [29].

Given an input x, the basic auto-encoder encodes the input to the hidden layer h by the following encoding function

$$h = f(W^{(1)}x + b^{(1)}), \tag{16}$$

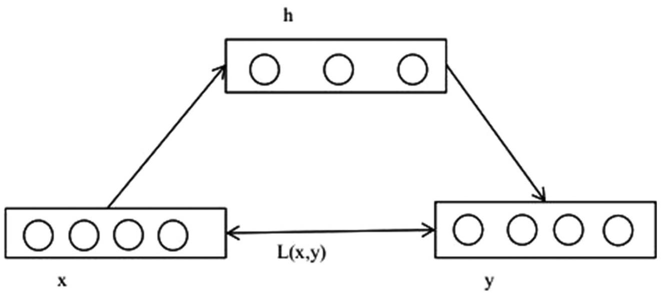


Fig. 4. Basic auto-encoder.

where f an encoder, typically a Sigmoid function $f(x) = 1/(1+e^{-x})$.

Afterwards, the hidden layer h is reconstructed to the output layer y by the following decoding function

$$y = g(W^{(2)}h + b^{(2)}), \tag{17}$$

where g represents the decoding function, typically a Sigmoid function or an identity function.

$\theta = \{W^{(1)}, b^{(1)}; W^{(2)}, b^{(2)}\}$ indicates the parameter set of the basic auto-encoder. The basic auto-encoder trains the parameter set θ by minimizing the reconstruction error $L(x,y) = \frac{1}{2}||y-x||^2$. Typically, the back-propagation algorithm is an effective method to train the parameters of an auto-encoder with n input units and m hidden units,presented in Algorithm 1.

Algorithm 1. Back-propagation Algorithm

```
    Input: ${(X^{(i)}, Y^{(i)})}_{i=1}^{m}$, $\eta$, $threshold$
    Output: $\theta = \{W^{(1)}, b^{(1)}; W^{(2)}, b^{(2)}\}$
1:  for $iteration = 1, 2, \ldots, iterater_{max}$ do
2:    for $example = 1, 2, \ldots, N$ do
3:      for $j = 1, 2, \ldots, m$ do
4:        $z_j^{(2)} = \sum_{i=1}^{n} W_{ji}^{(1)} + b_i^{(1)}$;
5:        $a_j^{(2)} = f(z_j^{(2)})$;
6:    for $i = 1, 2, \ldots, n$ do
7:        $z_i^{(3)} = \sum_{j=1}^{m} W_{ij}^{(2)} a_j^{(2)} + b_i^{(2)}$;
8:        $a_i^{(3)} = f(z_i^{(3)})$;
9:    if $J_{TAE}(\theta) > threshold$ then
10:     for $i = 1, 2, \ldots, n$ do
11:       $\sigma_i^{(3)} = (a_i^{(3)} \cdot (1 - a_i^{(3)})) \cdot (a_i^{(3)} - x_i)$;
12:     for $j = 1, 2, \ldots, m$ do
13:       $\sigma_j^{(2)} = (\sum_{i=1}^{n} W_{ij}^{(2)} \sigma_i^{(3)})(a^{(2)}(1 - a^{(2)}))$;
14:     for $i = 1, 2, \ldots, n$ do
15:       $\Delta b_i^{(2)} = \Delta b_i^{(2)} + \sigma_i^{(3)}$;
16:       for $j = 1, 2, \ldots, m$ do
17:         $\Delta w_{ij}^{(2)} = \Delta w_{ij}^{(2)} + a_j^{(2)} \cdot \sigma_i^{(3)}$;
18:     for $j = 1, 2, \ldots, m$ do
19:       $\Delta b_j^{(1)} = \Delta b_j^{(1)} + \sigma_j^{(2)}$;
20:       for $i = 1, 2, \ldots, n$ do
21:         $\Delta w_{ji}^{(1)} = \Delta w_{ji}^{(1)} + x_i \cdot \sigma_j^{(2)}$;
22:     $W = W - \eta \times (\frac{1}{N}\Delta w)$;
23:     $b = b - \eta \times (\frac{1}{N}\Delta b)$;
```

In Algorithm 1, the forward propagation is executed to calculate the actual output values of the auto-encoder on lines 1-8 while the back-propagation is performed to compute the gradients of the objective function with regard to each parameter on lines 10-21. Finally, the gradient descent is run to update the parameters on lines 22-23.

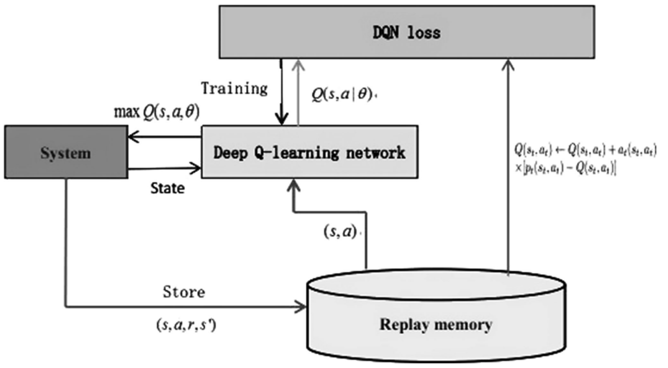


Fig. 5. The scheme to train the parameters of the deep Q-learning model.

The task of this paper is to train the parameters of the deep Q-learning model by combining the stacked auto-encoder and the Q-learning model, which is discussed in the nest section. After obtaining the parameters, the Q-value for each DVFS technique can be calculated according to the current system state. Furthermore, the DVFS technique with the lowest Q-value is chosen to configure the voltage and the frequency.

5 Training Algorithm for the Parameters of Deep Q-Learning Model

In this paper, the experience replay technique is used to train the parameters of the proposed deep Q-learning model. Fig. 5 illustrates the architecture of the proposed scheme for training the parameters of the deep Q-learning model.

To train the parameters of the deep Q-learning model, some records need to be collected and stored into the replay memory as the training samples. Specially, at the beginning of each hyperperiod t, the scheduler observes the system state $s_t = \{su_t, ds_t\}$ and chooses an action a_t that denotes a DVFS technique to adjust the voltage and the frequency. At the end of the hyperperiod that is also the beginning of the next hyperperiod $t+1$, the system state will be altered to $s_{t+1} = \{su_{t+1}, ds_{t+1}\}$, and a penalty $p_t(s_t, a_t)$ calculated by Eq. (15) is obtained. Accordingly, an experience tuple $e_t = (s_t, a_t, p_t(s_t, a_t), s_{t+1})$ can be obtain at the time-step t. In this paper, N experience tuples $E = \{e_1, e_2, \ldots, e_N\}$ are collected and stored into the replay memory to train the parameters of the deep Q-learning model.

The parameters of the deep Q-learning model are trained by two stages, i.e., pre-training stage and Q-learning stage.

$Q(s, a, \theta)$ is used to represent Q-function approximated by the deep Q-learning model, with $\theta = [\theta_{pre}, \theta_{ql}]$ where θ_{pre} and θ_{ql} for the stacked auto-encoder part and the last layer, respectively. In the pre-training stage, the greedy layerwise strategy is used to train the parameters of each auto-encoder from bottom to up for obtaining the initial parameters θ_{pre} of the deep Q-learning model. In detail, the system state parameters are fed into the input to learn the first hidden representation. Afterwards, the first hidden layer is used as the input to learn the second hidden representation. This process is performed from the bottom to up until all the hidden layers are trained for the parameters.

TABLE 2
Task Details

Task No.	number of jobs	period	wcet
1	60	50	6.31
2	100	30	0.89
3	30	100	12.92
4	50	60	4.88
5	25	120	15.63
6	200	15	0.29
7	600	5	0.45
8	600	5	0.93
9	24	125	13.55
10	300	10	0.04
11	30	100	2.53
12	25	120	4.21
13	40	75	8.26
14	75	40	4.09
15	75	40	8.26
16	200	15	1.19
17	120	25	4.15
18	40	75	11.53
19	600	5	0.66
20	24	125	8.88

In the Q-learning stage, the parameters $\theta = [\theta_{pre}, \theta_{ql}]$ are trained by minimizing the following loss function over m samples selected randomly from the replay memory.

$$L(\theta) = \sum_{i=1}^{m} [Q(s_i, a_i, \theta) - (Q(s_i, a_i) + \alpha_i \times [p_i(s_i, a_i) - Q(s_i, a_i)])]^2. \tag{18}$$

Therefore, the training algorithm for the parameters of the deep Q-learning model is outlined in Algorithm 2.

Algorithm 2. Training Algorithm for the Parameters of the Deep Q-Learning Model

```
1:  if $pretrain == true$ then
2:      Pick up some state samples to pre-train the SAE with $\theta_{pre}$;
3:      Initialize approximator of Q-function with $\theta = [\theta_{pre}, \theta_{ql}]$;
4:  else
5:      Initialize parameters of Q-function randomly;
6:  for $episode = 1, \ldots, N$ do
7:      Initialize an episode;
8:      for $t = 1, \ldots, T$ do
9:          Observe the current system state $s_t$;
10:         Select a DVFS technique $a_t$;
11:         Calculate penalty $p_t(s_t, a_t)$ and observe the next state
            $s_{t+1}$;
12:         Store $e_t = (s_t, a_t, p_t(s_t, a_t), s_{t+1})$ into replay memory;
13:         Calculate the Q-value: ;
14:             $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_i$
                $\times [p_t(s_t, a_t) - Q(s_t, a_t)]$;
15:         Perform the back-propagation algorithm to train
            $\theta = [\theta_{pre}, \theta_{ql}]$;
```

The time complexity of Algorithm 2 is dominated by two parts, i.e., collecting experience tuples and performing back-propagation algorithm. To collect one experience tuple, Et_{hyp} needs to be updated at each scheduling event, which requires $O(1)$ time complexity. Therefore, collecting N experience tuples cost $O(N)$ time complexity. For a basic auto-encoder with n input units and m hidden units, its complexity is $O(Mmn)$ where M denotes the training samples selected from the replay memory to perform the back-propagation algorithm. The storage complexity of Algorithm 2 is also dominated by two parts, i.e., replay memory and the size of the deep reinforcement learning model. Specially, N experience tuples requires $O(N)$ space complexity. For the the deep reinforcement learning model with k hidden layers, it has a space complexity of $O((k+1)mn)$.

6 Performance Evaluation

In the experiment, the proposed deep Q-learning model (DQL-EES) is evaluated by comparison with the hybrid DVFS scheduling based on reinforcement learning (QL-HDS). To evaluate the performance of the deep Q-learning model, a simulator followed [22] is developed by using JAVA. The simulator could schedule a large number of periodic real-time tasks and set the system to different voltage and frequency. The simulator takes a task set as input and outputs the power consumption.

In the experiment, 20 different tasks are generated randomly, and their details are listed in Table 2. Furthermore, the 20 tasks is used to generate 4 different task sets which have 3 tasks, 10 tasks, 14 tasks and 20 tasks, representatively. The power consumption caused by the task sets scheduled on the simulator are reported to evaluate the performance of the proposed model, which is a widely used validation methodology in some previous works [22].

From Table 2, each task has different parameters with others, consisting of period, WCET, and number of jobs. Therefore, every task set has distinct system utilization and the dynamic slack at different periods. Furthermore, the power consumption is normalized by following [22] with regards to the highest power consumption when reporting the simulation results.

To make the comparison fair with QL-HDS, three DVFS techniques, i.e., CC, LA and DRA, are used in the proposed method. Furthermore, the proposed method uses the same scheduling algorithm, namely earliest deadline first, with QL-HDS, in the experiments.

6.1 Varying Number of Hidden Layers and Hidden Units

First, the impact of the number of hidden layers and hidden units are investigated to the performance of our proposed model. Specially, 3 kinds of deep Q-learning models are constructed with one hidden layer, two hidden layers and three hidden layers, respectively, for scheduling the tasks on the simulator. Each kind of deep Q-learning model that has different number of hidden units are run for five times, each with different random initial parameters. To report the results clear, DQL-n is used to represent the deep Q-learning model with n hidden layers for energy-efficient scheduling in real-time systems. The results of average energy consumption are shown in Tables 3–6.

Two remarkable results can be observed from the above tables. First, for the deep Q-learning model with the same number of hidden layers, the energy consumption reduces slightly as the number of hidden units increases in most

TABLE 3
Power Consumption for Scheduling Task Set 1

	4	8	12	16
DQL-1	0.9755	0.9732	0.9718	0.9717
DQL-2	0.9319	0.9294	0.9277	0.9278
DQL-3	1	0.9982	0.9954	0.9951

TABLE 4
Power Consumption for Scheduling Task Set 2

	4	8	12	16
DQL-1	0.9903	0.9891	0.9864	0.9859
DQL-2	0.9278	0.9251	0.9219	0.9219
DQL-3	1	0.9969	0.9957	0.9958

TABLE 5
Power Consumption for Scheduling Task Set 3

	4	8	12	16
DQL-1	0.9487	0.9452	0.9431	0.9429
DQL-2	0.8943	0.8941	0.8902	0.8900
DQL-3	1	0.9942	0.9926	0.9923

cases. This result implies that adding hidden units can improve the performance of the deep Q-learning models since adding hidden units can approximate the Q-function more effectively. However, when the number of hidden units is more than 12, the energy consumption of each model remains almost unchanged. Such observation indicates that 12 hidden units are enough to approximate the Q-function in the experiments. Second, the deep Q-learning model with two hidden layers performs significantly better than the deep Q-learning model with one hidden layer, demonstrating that the performance of the deep Q-learning model is affected by the number of hidden layers. Specially, adding hidden layers is able to improve the performance of the deep Q-learning model effectively. However, the deep Q-learning model with 3 hidden layers performs worst. This is maybe because 3 hidden layers result in the over-fitting.

6.2 Comparison between QL-HDS and DQL-EPS

In this section, the performance comparison between QL-HDS and DQL-EES for power consumption scheduling in real-time systems are discussed. Table 7 shows the average power consumption of each model with 12 hidden units for scheduling 4 task sets.

From the results, the deep Q-learning model with two hidden layers (DQL-2) performs best. This is expected because DQL-2 yields the best approximation of Q-function. Furthermore, DQL-2 saves 5.63 percent energy than QL-HDS in average for task set 2, and even in the worst case (for task set 4), DQL-2 still saves average 2.48 percent energy than QL-HDS. For the four task sets, DQL-2 saves 4.20 percent energy than QL-HDS in average. Such observations indicate that the proposed method performs better than QL-HDS for energy-efficient scheduling in real-time systems. In addition, DQL-1 performs almost the same with QL-HDS and DQL-3 performs slightly worse than QL-HDS due to its over-fitting.

TABLE 6
Power Consumption for Scheduling Task Set 4

	4	8	12	16
DQL-1	0.9531	0.9501	0.9485	0.9482
DQL-2	0.9296	0.9282	0.9253	0.9254
DQL-3	1	0.9984	0.9968	0.9967

TABLE 7
Average Power Consumption of Each Model

	DQL-1	DQL-2	DQL-3	QL-HDS
Taskset 1	0.9765	0.9323	1	0.9781
Taskset 2	0.9901	0.9259	1	0.9822
Taskset 3	0.9503	0.8967	1	0.9478
Taskset 4	0.9513	0.9283	1	0.9531

TABLE 8
Average Execution Time of Each Model

	DQL-1	DQL-2	DQL-3	QL-HDS
Taskset 1	0.79	0.81	1	0.77
Taskset 2	0.72	0.79	1	0.71
Taskset 3	0.82	0.94	1	0.79
Taskset 4	0.69	0.81	1	0.68

6.3 Execution Time

Table 8 presents the execution time of QL-HDS for learning the Q-table and the execution time of DQL-EES for training the parameters of the deep Q-learning model.

From Table 8, the execution time of DQL-EES for training the parameters of the deep Q-learning model is longer than that of QL-HDS for learning the Q-table. This is because the execution time of the proposed model includes two parts, i.e., the time for picking up the experience tuples and the time for training the parameters. When the deep Q-learning model has only one hidden layer and few hidden units, it does not require much time to train parameters. So, the execution time of DQL-1 is slightly longer than that of QL-HDS. However, DQL-2 and DQL-3 need to take very long to pre-train and fine-tune parameters, so they require significantly more execution time than QL-HDS. Specially, DQL-2 takes about 12.5 percent time overhead in average compared with QL-HDS.

7 Conclusion

In this paper, a deep Q-learning model is presented for energy-efficient scheduling in real-time systems by combining a stacked auto-encoder and a Q-learning model. One property of the proposed deep Q-learning model is the use of the stacked auto-encoder to replace the Q-function of the conventional Q-learning model, so that the proposed scheme is able to learn the Q-value of each DVFS technology for any system state after training the parameters. Furthermore, since the proposed scheme could learn the Q-value for each DVFS technique under the continues state parameters, it is more effective to distinguish the system states than QL-HDS. To train the parameters of the deep Q-learning model, a training scheme based on the experience replay strategy is devised. Simulation results on different task sets demonstrated that the proposed algorithm could save average 4.2 percent energy than QL-HDS.

References

[1] A. Das, G. V. Merrett, M. Tribastone, and B. M. Al-Hashimi, "Workload change point detection for runtime thermal management of embedded systems," *IEEE Trans. Comput.-Aided Des. Integrated Circuits Syst.*, vol. 35, no. 8, pp. 1358–1371, Aug. 2016.

[2] Q. Zhang, C. Zhu, L. T. Yang, Z. Chen, L. Zhao, and P. Li, "An incremental CFS algorithm for clustering large data in industrial internet of things," *IEEE Trans. Industrial Inf.*, vol. 13, no. 3, pp. 1193–1201, Mar. 2017.

[3] X. Wang, L. T. Yang, J. Feng, X. Chen, and M. Jamal Deen, "A tensor-based big service framework for enhanced living environments," *IEEE Cloud Comput.*, vol. 3, no. 6, pp. 36–43, Jun. 2016.

[4] J. Gao, J. Li, Z. Cai, and H. Gao, "Composite event coverage in wireless sensor networks with heterogeneous sensors," in *Proc. IEEE Conf. Comput. Commun.*, 2015, pp. 217–225.

[5] Q. Zhang, L. T. Yang, Z. Chen, and P. Li, "High-order possibilistic c-means algorithms based on tensor decompositions for big data in IoT," *Inf. Fusion*, vol. 39, pp. 72–80, 2017.

[6] Y. Jiang, H. Zhang, Z. Li, Y. Deng, and X. Song, "Design and optimization of multiclocked embedded systems using formal techniques," *IEEE Trans. Industrial Electron.*, vol. 62, no. 2, pp. 1270–1278. Feb. 2015.

[7] Q. Zhang, L. T. Yang, Z. Chen, and P. Li, "An improved deep computation model based on canonical polyadic decomposition," *IEEE Trans. Syst. Man Cybern.: Syst.*, vol. 48, no. 10, pp. 1657–1666, Oct. 2018.

[8] L. Y. Zhang, K.-W. Wong, Y. Zhang, and J. Zhou, "Bi-level protected compressive sampling," *IEEE Trans. Multimedia*, vol. 18, no. 9, pp. 1720–1732, Sep. 2016.

[9] P. Li, Z. Chen, L. T. Yang, L. Zhao, and Q. Zhang, "A Privacy-preserving high-order neuro-fuzzy c-means algorithm with cloud computing," *Neurocomputing*, vol. 256, pp. 82–89, 2017.

[10] L. Zhao, Z. Chen, Y. Hu, G. Min, and Z. Jiang, "Distributed feature selection for efficient economic big data analysis," *IEEE Trans. Big Data*, vol. 4, no. 2, pp. 164–176, Jun. 2018.

[11] Q. Zhang, H. Zhong, L. T. Yang, Z. Chen, and F. Bu, "PPHOCFS: Privacy preserving high-order CFS algorithm on the cloud for clustering multimedia data," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 12, no. 4s, pp. 66:1–66:15, 2016.

[12] A. Malinowski and H. Yu, "Comparison of embedded system design for industrial applications," *IEEE Trans. Ind. Inf.*, vol. 7, no. 2, pp. 244–254, Feb. 2011.

[13] Q. Zhang, L. T. Yang, Z. Chen, and P. Li, "PPHOPCM: Privacy-preserving high-order possibilistic c-means algorithm for big data clustering with cloud computing," *IEEE Trans. Big Data*, to be published. doi: 10.1109/TBDATA.2017.2701816.

[14] H. Li, K. Ota, M. Dong, A. Vasilakos, and K. Nagano, "Multimedia processing pricing strategy in GPU-accelerated cloud computing," *IEEE Trans. Cloud Comput.*, to be published. doi: 10.1109/TCC.2017.2672554.

[15] Q. Zhang, L. T. Yang, Z. Chen, P. Li, and M. Jamal Deen, "Privacy-preserving double-projection deep computation model with crowdsourcing on cloud for big data feature learning," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 2896–2903, Aug. 2018.

[16] M. Z. A. Bhuiyany, G. Wang, J. Wu, X. Xiao, and X. Liu, "Application-oriented sensor network architecture for dependable structural health monitoring," in *Proc. IEEE Pacific Rim Int. Symp. Dependable Comput.*, 2015, pp. 91–98.

[17] B. K. Donohoo, C. Ohlsen, S. Pasricha, Y. Xiang, and C. Anderson, "Context-aware energy enhancements for smart mobile devices," *IEEE Trans. Mobile Comput.*, vol. 13, no. 8, pp. 1720–1732, Aug. 2014.

[18] H. Cai and G. Hu, "Distributed control scheme for package-level state-of-charge balancing of grid-connected battery energy storage system," *IEEE Trans. Ind. Inf.*, vol. 12, no. 5, pp. 1919–1929, Oct. 2016.

[19] X. Lin, Y. Wang, N. Chang, and M. Pedram, "Concurrent task scheduling and dynamic voltage and frequency scaling in a real-time embedded system with energy harvesting," *IEEE Trans. Comput.-Aided Des. Integrated Circuits Syst.*, vol. 35, no. 11, pp. 1890–1902, Nov. 2016.

[20] M. S. H. Talpur, M. Z. A. Bhuiyan, and G. Wang, "Energy-efficient healthcare monitoring with smartphones and IoT technologies," *Int. J. High Performance Comput. Netw.*, vol. 8, no. 2, pp. 186–194, 2015.

[21] H. Li, M. Dong, X. Liao, and H. Jin, "Deduplication-based energy efficient storage system in cloud environment," *Comput. J.*, vol. 58, no. 6, pp. 1373–1383, 2015.

[22] F. Islam and M. Lin, "Hybrid DVFS scheduling for real-time systems based on reinforcement learning," *IEEE Syst. J.*, vol. 11, no. 2, pp. 931–940, Jun. 2017.

[23] S. Ni, Y. Zhuang, Z. Cao, and X. Kong, "Modeling dependability features for real-time embedded systems," *IEEE Trans. Dependable Secure Comput.*, vol. 12, no. 2, pp. 190–203, Mar./Apr. 2015.

[24] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE Trans. Very Large Scale Integration Syst.*, vol. 8, no. 3, pp. 299–316, Jun. 2000.

[25] X. Lin, Y. Wang, Q. Xie, and M. Pedram, "Task scheduling with dynamic voltage and frequency scaling for energy minimization in the mobile cloud computing environment," *IEEE Trans. Serv. Computing*, vol. 8, no. 2, pp. 175–186, Mar./Apr. 2015.

[26] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Time-efficient power-aware scheduling for periodic real-time tasks," *IEEE Trans. Comput.*, vol. 53, no. 5, pp. 584–600, May 2004.

[27] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Proc. 8th ACM Symp. Operating Syst. Principles*, 2001, pp. 89–102.

[28] M. Bhatti, C. Belleudy, and M. Auguin, "Hybrid Power Management in Real Time Embedded Systems: An Interplay of DVFS and DPM Techniques," *Real-Time Syst.*, vol. 47, no. 2, pp. 143–162, 2011.

[29] Q. Zhang, L. T. Yang, and Z. Chen, "Deep computation model for unsupervised feature learning on big data," *IEEE Trans. Serv. Comput.*, vol. 9, no. 1, pp. 161–171, Jan./Feb. 2016.

[30] L. Li, Y. Lv, and F. Wang, "Traffic signal timing via deep reinforcement learning," *IEEE/CAA J. Autom. Sinica*, vol. 3, no. 3, pp. 247–254, Jul. 2016.

[31] Q. Zhang, L. T. Yang, and Z. Chen, "Privacy preserving deep computation model on dloud for big data feature learning," *IEEE Trans. Comput.*, vol. 65, no. 5, pp. 1351–1362, May 2016.

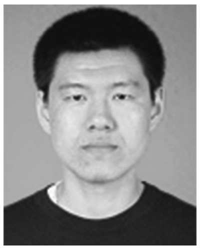
[32] Q. Zhang, L. T. Yang, X. Liu, Z. Chen, and P. Li, "A tucker deep computation model for mobile multimedia feature learning," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 13, no. 3s, pp. 39:1–39:18, 2017.

[33] M. P. Lawitzky, D. C. Snowdon, and S. M. Petters, "Integrating real time and power management in a real system," in *Pro. Operating Syst. Platforms Embedded Real-Time Appl.*, 2008, pp. 1–10.

[34] J. Lu and Y. Guo, "Energy-aware fixed-priority multi-core scheduling for real-time systems," in *Proc. IEEE 17th Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, 2011, pp. 277–281.

[35] G. Dhiman and T. Rosing, "System-level power management using online learning," *IEEE Trans. Comput.-Aided Des. Integrated Circuits Syst.*, vol. 28, no. 5, pp. 676–689, May 2009.

[36] H. Jung and M. Pedram, "Supervised learning based power management for multicore processors," *IEEE Trans. Comput.-Aided Des. Integrated Circuits Syst.*, vol. 29, no. 9, pp. 1395–1408, Sep. 2010.



Qingchen Zhang received the BEng degree from Southwest University, China, and the PhD degree from the Dalian University of Technology. He is currently a postdoc fellow with St. Francis Xavier University. His research interests include big data and deep learning.

Man Lin received the BE degree in computer science and technology from Tsinghua University, Beijing, China, in 1994 and the PhD degree from Linköping University, Linköping, Sweden, in 2000. She is currently a professor of computer science with St. Francis Xavier University, Antigonish, NS, Canada. Her research interests include realtime and embedded system scheduling, power-aware computing, parallel algorithms, and optimization algorithms.

Laurence T. Yang received the BE degree from Tsinghua University, China, and the PhD degree from the University of Victoria, Canada. He is currently a professor with the University of Electronic Science and Technology of China and St. Francis Xavier University, Canada. His research has been supported by the National Sciences and Engineering Research Council, and the Canada Foundation for Innovation. His research interests include parallel and distributed computing, embedded and ubiquitous/pervasive computing, big data, and cyber-physical-social systems.

Zhikui Chen received the BEng degree from Chongqing Normal University, China, and the PhD degree from Chongqing University, China. He is currently a professor with the Dalian University of Technology, China. His research interests include internet of things and big data processing.

Peng Li received the BEng degree from Dezhou University, China. He is working toward the PhD degree with the Dalian University of Technology, China. His research interests include deep learning and big data.