Deep Reinforcement Learning Based Residential Demand Side Management With Edge Computing

Tan Li
City University of Hong Kong
Hong Kong SAR
litan1003@gmail.com

Yuanzhang Xiao
University of Hawaii at Manoa
Honolulu, HI, USA
yxiao8@hawaii.edu

Linqi Song
City University of Hong Kong
Hong Kong SAR
linqi.song@cityu.edu.hk

***Abstract*—Residential demand side management (DSM) is a promising technique to improve the stability and reduce the cost of power systems. However, residential DSM is facing challenges under the ongoing paradigm shift of computation, such as edge computing. With the proliferation of smart appliances (e.g., appliances with computing and data analysis capabilities) and high-performance computing devices (e.g., graphics processing units) in the households, we expect surging residential energy consumption caused by computation. Therefore, it is important to schedule edge computing as well as traditional energy consumption in a smart way, especially when the demand for computation and thus for electricity occurs during the peak hours of electricity consumption.**

In this paper, we investigate an integrated home energy management system (HEMS) who participates in a DSM program and is equipped with an edge computing server. The HEMS aims to maximize the home owner's expected total reward, defined as the reward from completing edge computing tasks minus the cost of electricity consumption, the cost of computation offloading to the cloud, and the penalty of violating the DSM requirements. The particular DSM program considered in this paper, which is a widely-adopted one, requires the household to reduce certain amount of energy consumption within a specified time window. In contrast to well-studied real-time pricing, such a DSM program results in a long-term temporal interdependency (i.e., of a few hours) and thus high-dimensional state space in our formulated Markov decision processes. To address this challenge, we use deep reinforcement learning, more specifically Deep Deterministic Policy Gradient, to solve the problem. Experiments show that our proposed scheme achieves significant performance gains over reasonable baselines.

I. Introduction

A smart grid is an electrical grid that incorporates a communication network enabling assorted smart devices, such as smart meters, sensors, and smart appliances to connect and exchange information with each other and with the central platform as well. The bidirectional flow of information and power improves the operational efficiency of power grids and reduces the consumption of fossil fuels and emissions of green house gases, aiming at building an environmentally-friendly and reliable power system.

In smart grids, demand side management (DSM) is an important mechanism that aims to reduce the electricity load in the power system during peak hours, because a high peak to average ratio of electricity load will result in a dramatic cost increase in power grids [1]. However, the efficiency of DSM might be compromised under a paradigm shift of household energy consumption patterns [2]. Specifically, the households are deploying an rapidly increasing number of edge devices, such as smart appliances (e.g., voice assistant speakers, smart refrigerators), healthcare monitoring devices, and surveillance networks [3], [4]. These devices often require high-performance computation (e.g., voice recognition and synthesis, computer vision), which is preferred to be done locally through edge computing due to latency requirement and privacy concerns [5]. As a result, the surge in edge computing will increase the household energy consumption and make it more challenging for DSM. Therefore, it is important to jointly consider DSM and edge computing in an integrated framework. The importance of such an integrated framework has been recognized by some researchers [3], [4]. However, existing works [3], [4] remain to be high-level discussion without rigorous problem formulation and concrete solutions.

In this paper, we study for the first time how to perform DSM in an integrated network of smart grids with edge computing by making decisions about how to offload computational tasks. We consider a widely-adopted DSM program in this paper, which requires the household to reduce certain amount of energy consumption within a specified time window. One challenge we often face in practice is that the DSM program is not widely accepted by consumers [6] because of the unwillingness to shift the power load. In order to tackle this, we consider the integrated system such that users do not need to (or slightly) perform power load shifting during the effective time period and instead they could perform computational task offloading in order to realize DSM. This is motivated by the fact that computation will consist of a major power consumption in our future smart home and smart appliances. Users can choose to allocate the computational task load among the edge devices and the cloud platform where the computation in edge devices will result in more power consumption and the computation in the cloud platform will result in some monetary cost.

Our main contributions are as follows.

1) We consider the DSM with edge computing and have formulated a Markov Decision Process (MDP) problem of

This work was partially supported by the Hong Kong RGC grant ECS 9048149 and the City University of Hong Kong grant (Project No. 7200594). Xiao was partially supported by NSF IIP Grant No. 1822213.

978-1-5386-8099-5/19/$31.00 ©2019 IEEE

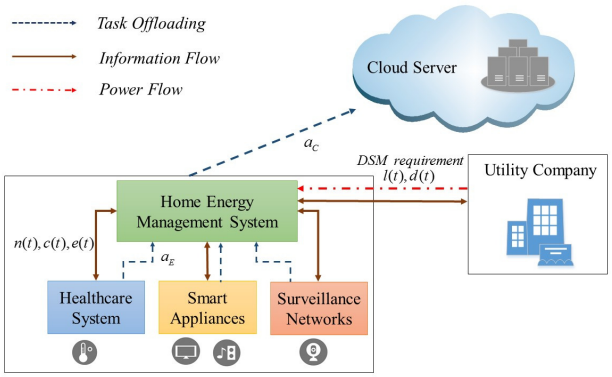


Fig. 1. Residential demand side management with edge computing.

making decisions about how to offload computational tasks among the edge and the cloud.

2) We propose a deep reinforcement learning approach to solve the MDP problem. The proposed method is able to overcome the challenges of dealing with high-dimensional state and action spaces and a large continuous action space in our model. The key is to use deep neural networks in the actor-critic approach of solving MDPs.

3) Our experimental results show that our proposed scheme has a good convergence for residential DSM, and could achieve significant performance gains over other task offloading strategies in a wide range of environmental scenarios.

II. System Model

A. System Setup

We consider a residential user who runs computationally intensive tasks and participates in a demand response program. We illustrate the system in Fig. 1. The user employs an intelligent home energy management system (HEMS) to determine how to perform the tasks, locally (i.e., edge computing) or in the remote cloud (i.e., cloud computing) and to allocate the local computing resources among the tasks scheduled on the edge. The HEMS contains edge computing resources to perform some of the tasks locally (i.e., edge computing) and could also pay to perform some of the tasks in a cloud server (i.e., cloud computing). The HEMS also schedules the energy consumption of the entire household. Moreover, the HEMS acts as the interface between the household and the utility company: it receives demand response signals from the utility company, and monitors and reports the total energy consumption to the utility company.

Next, we build a discrete-time system model with infinite time horizon $t = 1, 2, \ldots$, and describe the tasks and the demand response program in details.

1) Computationally Intensive Tasks: There are a growing number of computationally intensive tasks within a household. Examples of such tasks include smart appliances such as smart refrigerators, which can automatically set the inside temperature by analyzing images of stored items; voice assistant speakers, e.g., Amazon Echo, Google Home, and entertainment devices like smart TVs and game consoles with virtual reality (VR) capability; healthcare monitoring devices such as wrist bands, blood pressure and heart rate monitors, smart watches, and fall detectors; and surveillance networks that need to process a large volume of computer vision data. Common features of these tasks include

- requirement of high-performance computing for voice, image, and video recognition and synthesis;
- desire for edge computing due to low latency requirement and privacy issues;
- high energy consumption from high-performance edge computing.

Bearing these common features in mind, we model these tasks mathematically as follows. At each time t, there are $n(t) \in \mathbb{Z}_+$ tasks to be completed. Each task, indexed by $i = 1, 2, \ldots, n(t)$, is characterized by a tuple (C_i, E_i), where $C_i \in \mathbb{R}_+$ is the total amount of computing resources required for completing task i (i.e., task load) and $E_i \in \mathbb{Z}_+$ is the number of time slots before the task expires. The status of each task i includes the remaining task load $c_i(t) \in [0, C_i]$ and the elapsed time of the task $e_i(t) \in [0, E_i]$. For example, to train a neural network based machine learning model, it usually takes hours to process the newly generated data each day. The HEMS determines the amount $a_{E,i}(t) \in \mathbb{R}_+$ of task offloaded to the edge and the amount $a_{C,i}(t) \in \mathbb{R}_+$ of task offloaded to the cloud for task i in time slot t. Given the task offloading, the status of task i will be updated in time slot $t+1$ according to

$$c_i(t+1) = c_i(t) - a_{E,i}(t) - a_{C,i}(t), \quad (1)$$

$$e_i(t+1) = e_i(t) + 1. \quad (2)$$

For compact presentation, we define the vectors $\boldsymbol{c}(t) = \left[c_1(t), \ldots, c_{n(t)}(t)\right]$ and $\boldsymbol{e}(t) = \left[e_1(t), \ldots, e_{n(t)}(t)\right]$ as the status of all the tasks. We also define the vectors $\boldsymbol{a}_E(t) = \left[a_{E,1}(t), \ldots, a_{E,n(t)}(t)\right]$ and $\boldsymbol{a}_C(t) = \left[a_{C,1}(t), \ldots, a_{C,n(t)}(t)\right]$ as the task offloading decisions. Since there is no need to allocate more resources than those required for completing the tasks, the set of feasible actions at each time slot t is

$$\begin{aligned}\mathcal{A}(t) = \{(\boldsymbol{a}_E(t), \boldsymbol{a}_C(t)) \mid\ & a_{E,i}(t) \geq 0, a_{C,i}(t) \geq 0, \\ & a_{E,i}(t) + a_{C,i}(t) \leq c_i(t),\ i = 1, \ldots, n(t) - 1\}\end{aligned} \quad (3)$$

The energy consumption of each task i at time slot t is

$$\beta \cdot \sum_{i=1}^{n(t)} a_{E,i}(t), \quad (4)$$

where $\beta \in \mathbb{R}_+$ is the amount of electricity consumed per unit usage of edge computing resources.

Remark 1: Our model for computationally intensive tasks is general enough to model other loads in the household. For shiftable load such as electric vehicle charging, we can use the tuple (C_i, E_i) to denote the total energy requirement and the time before the deadline of finishing the charging, use $a_{E,i}(t)$ to denote the energy scheduled at time t, and set $\beta = 1$ and $a_{C,i}(t) = 0$. For non-shiftable load, we can simply set $E_i = 1$.

2) Demand Response Program: The user participates a demand response program that may require the user to consume a reduced amount of electricity within a specified time frame (e.g., a few hours). Such a demand reduction is usually mandated by a contract signed by the user and the utility company. The user is rewarded when signing up for the program and may need to pay a penalty if failing to fulfill the load reduction requirement. At a certain time slot t, the utility company may send a signal $(l(t), d(t))$ to the user, which requires the user to restrict its electricity load to a total amount of $l(t)$ kilowatt hour (kWh) in the next $d(t)$ time slots. The demand response events will not overlap, namely a new event will always happen after the previous event has ended. As a result, the status of the demand response event transits according to

$$l(t+1) = l(t) - \beta \cdot \sum_{i=1}^{n(t)} a_{E,i}(t), \quad (5)$$

$$d(t+1) = d(t) - 1. \quad (6)$$

In other words, we can view $l(t)$ as the "quota" for electricity consumption in the next $d(t)$ time slots, starting from time t. This quota decreases over time as the user consumes more electricity. Note that the computing resources in the cloud will not be counted when calculating the electricity consumption.

B. Formulation of Markov Decision Process

As we can see from Eq. (1)(2)(5)(6), the current HEMS states are completely determined by the previous states and task assignment decisions. Therefore, we can formulate the user's decision making problem as a Markov decision process (MDP) with infinite time horizon.

1) States: The state at each time slot t consists of the price of electricity $p_E(t) \in \mathbb{R}_+$, the price of cloud computing $p_C(t) \in \mathbb{R}_+$, the status of demand response $(l(t), d(t)) \in \mathbb{R}_+ \times \mathbb{Z}_+$, the status of the tasks $(n(t), \boldsymbol{c}(t), \boldsymbol{e}(t)) \in \mathbb{N} \times \mathbb{R}_+^{n(t)} \times \mathbb{R}_+^{n(t)}$. We define the collection of all the above status as the state of the MDP

$$\boldsymbol{s}(t) = (p_E(t), p_C(t), l(t), d(t), n(t), \boldsymbol{c}(t), \boldsymbol{e}(t)). \quad (7)$$

2) Actions: The action at each time slot t specifies how much computing resources, both on the edge and in the cloud, allocated to each task, namely

$$\boldsymbol{a}(t) = (\boldsymbol{a}_E(t), \boldsymbol{a}_C(t)) \in \mathbb{R}_+^{n(t)} \times \mathbb{R}_+^{n(t)}. \quad (8)$$

3) Rewards: The reward at each time slot t has four components. The first component is the positive reward of completing the task on time. For task i, the reward function is written as

$$u_{T,i} : \mathbb{R}_+ \times \mathbb{Z}_+ \to \mathbb{R}_+ \quad (9)$$

$$(c_i(t), e_i(t)) \mapsto u_{T,i}(c_i(t), e_i(t)). \quad (10)$$

In general, the reward is realized only when the task is completed before the deadline. An example reward function could be

$$u_{T,i}(c_i(t), e_i(t)) = \begin{cases} \gamma_i & \text{if } c_i(t) = 0 \text{ and } e_i(t) \le E_i \\ 0 & \text{otherwise} \end{cases}, \quad (11)$$

where $\gamma_i > 0$ is the reward of completing task i.

The second component of the reward is the cost of electricity consumption, which is simply

$$-p_E(t) \cdot \left[\beta \cdot \sum_{i=1}^{n(t)} a_{E,i}(t) \right]. \quad (12)$$

The third component of the reward is the cost of running the computation in the cloud, which can be calculated as

$$-p_C(t) \cdot \sum_{i=1}^{n(t)} a_{C,i}(t). \quad (13)$$

The final component of the reward is the potential penalty of violating the demand response requirement, which is defined as

$$u_D(l(t), d(t)) = \begin{cases} -\rho & \text{if } d(t) = 0 \text{ and } l(t) < 0 \\ 0 & \text{otherwise} \end{cases}, \quad (14)$$

where $\rho > 0$ is the (usually large) penalty of violating the demand response contract. Since we assume that the user has also signed up for the program, the one-time sign-on rebate is not included in the reward.

The net reward function is then

$$u(\boldsymbol{s}(t), \boldsymbol{a}(t)) = \sum_{i=1}^{n(t)} u_{T,i}(c_i(t), e_i(t)) - p_C(t) \cdot \sum_{i=1}^{n(t)} a_{C,i}(t) - p_E(t) \cdot \left[\beta \sum_{i=1}^{n(t)} a_{E,i}(t) \right] - p_D(l(t), d(t)).$$

4) State Transitions: The transition of the task state follows (1), and the transition of the demand response state follows (5). The pricing states $p_E(t)$ and $p_C(t)$ are drawn randomly independently of each other and other state components. The number of tasks $n(t)$ evolves randomly because the number of new tasks is random, namely

$$n(t+1) = n(t) - \sum_{i=1}^{n(t)} \mathbf{1}_{\{c_i(t)=0 \text{ and } e_i(t) \le E_i\}} + m(t+1), \quad (15)$$

where $\mathbf{1}_{\{\cdot\}}$ is the indicator function; $\sum_{i=1}^{n(t)} \mathbf{1}_{\{c_i(t)=0 \text{ and } e_i(t) \le E_i\}}$ is the number of completed tasks in time t; and $m(t+1)$ is the random number of new tasks at time $t+1$. Note that there is some certain stable state transition probability $\mathbb{P}(\boldsymbol{s}|\boldsymbol{s}', \boldsymbol{a})$ for the system from state $\boldsymbol{s}'$ to state $\boldsymbol{s}$ when taking action $\boldsymbol{a}$. This state transition is unknown and needs to be learned.

Without loss of generality, we focus on Markov policies, which depend on the current state and time only. We write a Markov policy as

$$\pi : \boldsymbol{s}(t) \mapsto \boldsymbol{a}(t), \quad (16)$$

and the set of all Markov policies as Π_M.

Our goal is to find the optimal Markov policy that maximizes the expected total reward starting from any initial state $\boldsymbol{s}(1)$, namely

$$\pi^* = \arg\max_{\pi\in\Pi_M} \mathbb{E}^{\pi}\left\{\sum_{t=1}^{\infty}\gamma^t u\left(\boldsymbol{s}(t),\boldsymbol{a}(t)\right)\right\}, \tag{17}$$

where $\gamma \in (0,1)$ is the discount factor, and the expectation $\mathbb{E}^{\pi}$ depends on the policy π.

Note that, here in our reinforcement learning problem, we need to learn the state transition of the MDP and the optimal policy, where at least one of the optimal policies is Markov.

III. Solution

In this section, we first describe the challenges in solving the problem (17), and then our proposed solution based on recent advances in deep reinforcement learning, namely the Deep Deterministic Policy Gradient (DDPG) method.

A. Challenges

Many reinforcement learning algorithms, such as Q-learning [7], solve (17) using the action-value function $Q^{\pi}(\boldsymbol{s}(t),\boldsymbol{a}(t))$, defined as the total reward obtained under policy π when the current state and action are $\boldsymbol{s}(t)$ and $\boldsymbol{a}(t)$. The action-value function obeys the recursive relationship known as the Bellman equation:

$$\begin{aligned} Q^{\pi}(\boldsymbol{s}(t),\boldsymbol{a}(t)) = u\left(\boldsymbol{s}(t),\boldsymbol{a}(t)\right) + \\ \gamma\mathbb{E}_{\boldsymbol{s}(t+1),\boldsymbol{a}(t+1)\sim\pi}\left\{Q^{\pi}\left(\boldsymbol{s}(t+1),\boldsymbol{a}(t+1)\right)\right\}. \end{aligned} \tag{18}$$

Given the optimal action-value function $Q^*(\boldsymbol{s}(t),\boldsymbol{a}(t))$, the optimal Markov policy π^* can be defined as

$$\pi^*\left(\boldsymbol{s}(t)\right) = \arg\max_{\boldsymbol{a}(t)\in\mathcal{A}(t)} Q^*\left(\boldsymbol{s}(t),\boldsymbol{a}(t)\right). \tag{19}$$

There are two challenges in solving the problem (17) using (18) and (19). The first challenge comes with using (18) to find the optimal action-value function $Q^*(\boldsymbol{s}(t),\boldsymbol{a}(t))$ when the numbers of states and actions are large. In our setting, the action $\boldsymbol{a}(t) = \{a_{C,1}, a_{C,2}, ..., a_{C,n(t)}, a_{E,1}, a_{E,2}, ..., a_{E,n(t)}\}$ is a $2n(t)$-dimensional vector, and the state $\boldsymbol{s}(t) = (p_E(t), p_C(t), l(t), d(t), n(t), \boldsymbol{c}(t), \boldsymbol{e}(t))$ is a $(2n(t)+5)$-dimensional vector. Therefore, the numbers of states and actions grow exponentially with the number of tasks, leading to "curse of dimensionality". The second challenge comes with using (19) to find the optimal action under each state when the number of actions is large. In our setting, the offloading actions $a_{C,i}$ and $a_{E,i}$ are continuous (or discrete with high granularity). This means that we need to search through the large action space to find the action with the highest action-value.

Traditional reinforcement learning algorithms such as Q-learning [7] maintain a "Q table" to compute and store the action-value function, which is not scalable. More advanced algorithms such as deep Q network (DQN) [8] uses a deep neural network as a function approximator of the action-value function, in order to address the first challenge. However, it does not overcome the second challenge because it is hard to solve (19) when the optimal action-value function is represented by a deep neural network.

Therefore, we propose to use the DDPG algorithm [9], which addresses both challenges by integrating deep neural network and the actor-critic approach. As shown in Fig. 2, the DDPG network consists of two parts: *Actor* and *Critic* Network. Both parts are composed of deep neural networks and guide each other's optimization process. First, we solve the problem (18) under a high-dimensional state space by using the Critic Network as the action-value function estimator, which is similar to DQN. Moreover, we address the challenge in searching the optimal action (19) under large and continuous action space by using the Actor Network to estimate the optimal action based on deterministic policy gradients.

B. Details of the DDPG Algorithm

The algorithm is shown in Algorithm 1 and Fig. 2. In Fig. 2, we define a parameterized *Actor* and *Critic* function as $\mu(\boldsymbol{s}|\theta^{\mu})$ and $Q(\boldsymbol{s},\boldsymbol{a}|\theta^{Q})$ with parameters θ^{μ} and θ^{Q} respectively. At every time slot, we can observe a state $\boldsymbol{s}(t)$ collected by the HEMS system. The *Actor* selects an action based on the current optimal policy. Then, the system would get a reward $u(t)$ and the transit to next state $\boldsymbol{s}(t+1)$. The *Critic* network can be optimized after getting the reward $u(t)$ by minimizing the Mean Square Error (MSE) loss:

$$L(\theta^{Q}) = E_{a(t)\sim\pi,\boldsymbol{s}(t),u(t)}[(Q(\boldsymbol{s}(t),\boldsymbol{a}(t)|\theta^{Q}) - y(t))^2] \tag{20}$$

where $y(t) = u(\boldsymbol{s}(t),\boldsymbol{a}(t) + \gamma Q(\boldsymbol{s}(t+1),\boldsymbol{a}(t+1)|\theta^{Q})$. Finally, the *Actor* network updates the policy with respect to the direction of the Q-value gradient in following way:

$$\begin{aligned} &E_{s(t)}[\nabla_{\theta^{\mu}} Q(\boldsymbol{s},\boldsymbol{a}|\theta^{Q})|_{\boldsymbol{s}=\boldsymbol{s}(i),a=\mu(\boldsymbol{s}(i)|\theta^{\mu})}] \\ &= E_{\boldsymbol{s}(t)}[\nabla_a Q(\boldsymbol{s}(i),\boldsymbol{a}(i)|\theta^{Q})\nabla_{\theta^{\mu}}\mu(\boldsymbol{s}(i)|\theta^{\mu}))] \end{aligned} \tag{21}$$

Specifically, for the expectation parts in (20) and (21), we can form Monte Carlo estimates to obtain the estimation values as follows.

$$E_{z\sim q(z|x_i)}[f(z)] \approx \frac{1}{M}\sum_{m=1}^{M} f(z), z\sim q(z|x_i) \tag{22}$$

The training process of the whole model is shown in Fig. 2. In order to train the whole network in a stable and robust way, we do not update the model every time we collect a state transition sample, instead, we set a replay buffer with size R. When every training episode starts, state transitions follow the current policy and the tuple $(\boldsymbol{s}(t),\boldsymbol{a}(t),u(t),\boldsymbol{s}(t+1))$ is stored in the replay buffer. When the replay buffer is up to its capacity limitation, the *Actor* and *Critic* Network are updated by sampling the transition tuples from the buffer. Moreover, since even small update of θ^{Q},θ^{μ} will lead to a great change in action-value and strategy, we set two target networks with parameters $Q'(\boldsymbol{s},\boldsymbol{a}|\theta^{Q'})$ and $\mu'(\boldsymbol{s}|\theta^{\mu'})$ to give consistent targets during training process. The weights of these target networks are updated slowly by following the learned networks in the way: $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$. As $\tau \ll 1$, the

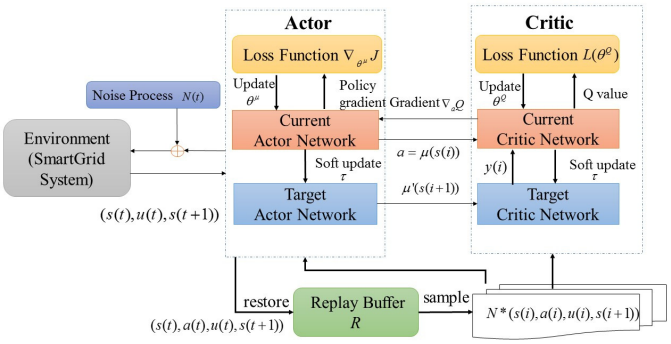


Fig. 2. Deep deterministic policy gradient framework

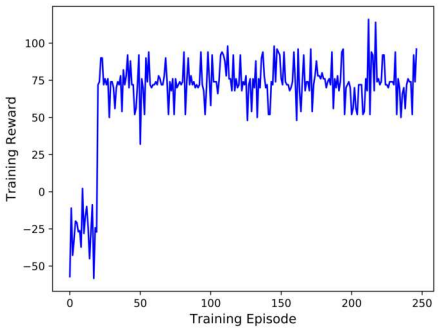


Fig. 3. Training rewards

target values are constrained to change slowly, which is called the "soft update".

Algorithm 1 Deep Deterministic Policy Gradient Algorithm

Randomly initialize critic network $Q(\boldsymbol{s}, \boldsymbol{a}|\theta^Q)$ and actor network $\mu(\boldsymbol{s}|\theta^\mu)$ with weight θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$.
Initialize replay buffer R
for $episode = 1$ to m **do**
  Initialize a random process for action exploration;
  Receive initial state: $\boldsymbol{s}(1) = (p_E(1), p_C(1), l(1), d(1), n(1), \boldsymbol{c}(1), \boldsymbol{e}(1))$
  for $t = 1$ to T **do**
    Select task resource allocation action: $\boldsymbol{a}(t) = \mu(s(t)|\theta^\mu) + N(t)$ according to the current policy and exploration noise.
    Execute action $\boldsymbol{a}(t)$ and observe reward $u(t)$ and observe next state $\boldsymbol{s}(t+1)$.
    Store transition $(\boldsymbol{s}, \boldsymbol{a}(t), u(t), \boldsymbol{s}(t+1))$ in replay buffer R
    Sample a random batch of k transitions from R
    Update *Critic* Network by minimizing the sampled MSE loss by (20)
    Update *Actor* policy using the sampled policy gradient by (21)
    Update the target networks Q' and μ':
    $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
    $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
  end for
end for

IV. Experimental Results

We present simulation results to show the performance of our proposed method. The environmental parameters are set in Table I according to the best practice [10], [11].

We compare our proposed DDPG method with three other approaches. 1) *Random Action* stands for a random policy, where the agent selects two values between (0, 50) randomly and assign them as the action $a_{E,i}, a_{C,i}$. 2) *Full Edge* means that all the computation tasks are executed locally; in other words, $a_{C,i} = 0$. 3) *Full Cloud* scheme offloads all tasks to the cloud server, i.e., $a_{E,i} = 0$.

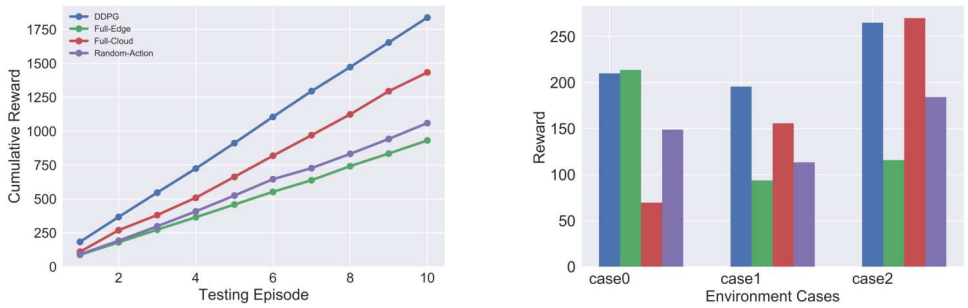
First, we present the training process of our DDPG method with a number of training episodes in Fig. 3. At the beginning of every episode, we reset the environment and then compute the total reward and update the model parameters for 500 steps. That means the task and DSM states are constantly changing during each training process, so our model needs to learn the optimal strategy under different environment. The curve shows that DDPG algorithm has a good convergence for our residential demand side management problem. As the number of training episodes increases, there are still fluctuations in the training rewards due to the randomness in the sampled environments.

Second, we fix the environmental parameters (i.e., the electric price) to compare rewards obtained by different algorithms. Fig. 4(a) shows the cumulative rewards of the aforementioned four methods as the number of testing episodes increases. During the training process, in order to make the agent fully trained, the environment settings in each episode are random. However, the testing episodes have periodicity, so the reward grows linearly. Taken as a whole, our DDPG achieves the highest reward, and the *Full Edge* policy obtains the lowest one. Specifically, after 10 testing episodes, our DDPG method yields 28%, 76%, and 96% more than the other three methods, respectively. This is because if all the tasks are computed locally, $l(t)$ will go down fast and become negative before $d(t) = 0$. Thus, the user will violate the demand response requirement and gain a penalty easily. On the contrary, if the user offloads all tasks to the cloud server, the cost of running the computation in the cloud will be higher than the edge computing electrical cost.

We also carry out experiments on the impact of environmental parameters on the rewards. Fig. 4(b) shows the rewards of the four algorithms in three different cases. In case 0, we set the price of electricity P_E as 0, then the cost of electricity consumption will be 0. Under this situation, *Full Edge* should be the optimal strategy while *Full Cloud* is the worst one. Experimental results show that the strategy chosen by DDPG

TABLE I
EXPERIMENTAL PARAMETERS

Category	Parameters	Value
Price parameters	Price of electricity P_E Price of cloud computing P_C	$\$0.5 \sim \$0.8/kWh$ [10] $\$0.002 - \$0.004$ per cpu unit per hour [11]
Task state	Task number $n(t)$ Required computation resources $c_i(t)$ Maximum elapsed time $E_i(t)$	New arriving task obeys the Poisson distribution Uniform distribution between (50, 800) cpu units Uniform distribution between (1, 600) seconds
Demand-side management state	Constraint electricity load $l(t)$ Constraint demand response time $d(t)$	Uniform distribution between (1000, 3000) kWh Uniform distribution between (1, 6) hours
Action	$a_{E,i}, a_{C,i}$	Continuous values between 0 and 50 cpu units
Reward	Completing the tasks Violating the demand response requirement	20 per task -100



(a) Cumulative reward

(b) Rewards under different environmental cases

Fig. 4. Simulation results

achieves almost the same reward as *Full Edge*. Similarly, we set P_C as 0 in case 2, so the optimal action tends to offload all tasks. In Fig. 4(b), we can see that our method obtains the same reward as the theoretically optimal strategy *Full Cloud*. Note that as we set a relatively large value for DSM penalty, the rewards in case 2 are relative higher than others. Case 0 and case 2 show that our strategy can achieve nearly optimal performance at the extreme situation when electricity fee (or zero cloud server rental) is 0. In addition, we describe a more practical scenario in case 1 that the electricity and cloud computing costs are basically equal. In this case, our proposed strategy gets twice as much reward as the *Full Edge* and outperforms *Full Cloud* and *Random Action* algorithms by 33% and 67%, respectively. These experiments have shown that our method can output different actions according to the change of environmental parameters and remain optimal in all kinds of situations.

V. CONCLUSION

In this paper, we studied an integrated smart grid system model for demand side management with edge computing. We derived the deep reinforcement learning-based algorithm (Deep Deterministic Policy Gradient) to solve this problem where we use neural networks to approximate the action-value function and the optimal action. Experimental results show that our proposed scheme works well and could achieve significant performance gains over other baselines under various environmental parameters.

REFERENCES

[1] P. Palensky and D. Dietrich, "Demand side management: Demand response, intelligent energy systems, and smart loads," *IEEE Transactions on Industrial Informatics*, vol. 7, no. 3, pp. 381–388, Aug 2011.
[2] E. Ahvar, A.-C. Orgerie, and A. Lebre, "Estimating energy consumption of cloud, fog and edge computing infrastructures," *IEEE Transactions on Sustainable Computing*, 2019.
[3] R. K. Barik, S. K. Gudey, G. G. Reddy, M. Pant, H. Dubey, K. Mankodiya, and V. Kumar, "Foggrid: Leveraging fog computing for enhanced smart grid network," in *2017 14th IEEE India Council International Conference (INDICON)*, Dec 2017, pp. 1–6.
[4] S. Zahoor, N. Javaid, A. Khan, B. Ruqia, F. J. Muhammad, and M. Zahid, "A cloud-fog-based smart grid model for efficient resource utilization," in *2018 14th International Wireless Communications Mobile Computing Conference (IWCMC)*, June 2018, pp. 1154–1160.
[5] K. Shahryari and A. Anvari-Moghaddam, "Demand side management using the internet of energy based on fog and cloud computing," in *2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, June 2017, pp. 931–936.
[6] J. Wang, M. Biviji, and W. M. Wang, "Case studies of smart grid demand response programs in North America," in *ISGT 2011*, Jan 2011, pp. 1–5.
[7] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.
[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.
[9] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.
[10] J. He, "The development and utilization of microgrid technologies in China," *Energy Sources, Part A: Recovery, Utilization, and Environmental Effects*, pp. 1–22, 2018.
[11] https://www.huaweicloud.com/en-us/product/ecs.html.