Deep Reinforcement Learning for Autonomous Internet of Things: Model, Applications and Challenges

Lei Lei, *Senior Member, IEEE*, Yue Tan, *Graduate Student Member, IEEE*, Kan Zheng, *Senior Member, IEEE*, Shiwen Liu, Kuan Zhang, *Senior Member, IEEE*, and Xuemin Shen, *Fellow, IEEE*

***Abstract*—The Internet of Things (IoT) extends the Internet connectivity into billions of IoT devices around the world, where the IoT devices collect and share information to reflect status of the physical world. The Autonomous Control System (ACS), on the other hand, performs control functions on the physical systems without external intervention over an extended period of time. The integration of IoT and ACS results in a new concept - autonomous IoT (AIoT). The sensors collect information on the system status, based on which the intelligent agents in the IoT devices as well as the Edge/Fog/Cloud servers make control decisions for the actuators to react. In order to achieve autonomy, a promising method is for the intelligent agents to leverage the techniques in the field of artificial intelligence, especially reinforcement learning (RL) and deep reinforcement learning (DRL) for decision making. In this paper, we first provide a tutorial of DRL, and then propose a general model for the applications of RL/DRL in AIoT. Next, a comprehensive survey of the state-of-art research on DRL for AIoT is presented, where the existing works are classified and summarized under the umbrella of the proposed general DRL model. Finally, the challenges and open issues for future research are identified.**

***Index Terms*—Autonomous Internet of Things, deep reinforcement learning.**

I. Introduction

A. Autonomous Internet of Things

THE INTERNET of Things (IoT) connects a huge number of IoT devices to the Internet, where the IoT devices generate massive amount of sensory data to reflect status of the physical world. These data could be processed and analyzed by leveraging machine learning (ML) techniques, with the objective of making informed decisions to control the reactions of IoT devices to the physical world. In other words, IoT devices become autonomous with ambient intelligence by integrating IoT, ML and autonomous control. For example, smart thermostats can learn to autonomously control central heating systems based on the presence of users and their routine. IoT and autonomous control system (ACS) [1] are originally independent concepts, and the realization of one does not necessarily require the other. The concept of autonomous IoT (AIoT) was proposed as the next wave of IoT that can explore its future potential [2].

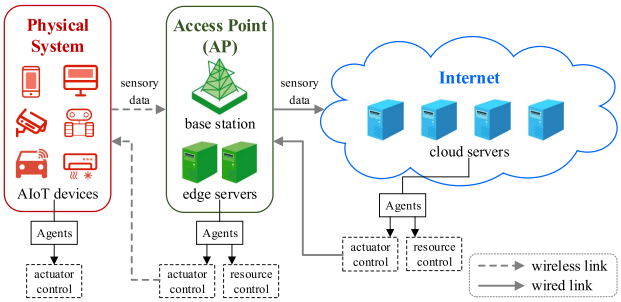


Fig. 1. Autonomous IoT system.

The AIoT systems provide a dynamic and interactive environment with a number of AIoT devices, which sense the environment and make control decisions to react. As shown in Fig. 1, an AIoT system typically includes a physical system where the AIoT devices with sensors and actuators are deployed. The IoT devices are usually connected by wireless networks to an access point (AP) such as a mobile base station (BS), which acts as a gateway to the Internet where the cloud servers are deployed. Moreover, the edge/fog servers with limited data processing and storage capabilities as compared to the cloud servers may be deployed at the APs [3]. After the IoT devices acquire the sensory data that represent full or partial status of the physical system, they need to process the data and make control decisions for the actuators to react. The data processing tasks can be executed locally at the IoT devices, or remotely at the edge/fog/cloud servers.

Manuscript received June 18, 2019; revised December 15, 2019; accepted April 13, 2020. Date of publication April 16, 2020; date of current version August 21, 2020. This work was supported by the National Natural Science Foundation of China under Grant 61671089. *(Corresponding author: Lei Lei.)*

Lei Lei is with the College of Engineering and Physical Sciences, University of Guelph, Guelph, ON N1G 2W1, Canada (e-mail: leil@uoguelph.ca).

Yue Tan, Kan Zheng, and Shiwen Liu are with the Intelligent Computing and Communication Laboratory, Key Laboratory of Universal Wireless Communications, Ministry of Education, Beijing University of Posts and Telecommunications, Beijing 100876, China.

Kuan Zhang is with the Department of Electrical and Computer Engineering, University of Nebraska–Lincoln, Lincoln, NE 68182 USA.

Xuemin Shen is with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada.

Digital Object Identifier 10.1109/COMST.2020.2988367

TABLE I
SUMMARY OF EXISTING SURVEY/OVERVIEW WORKS IN THE AREA OF IOT AND ML/DL/RL/DRL

		ML	DL	RL	DRL
General-purpose IoT System		[6]–[9]	[3]		
Specific IoT Application Areas	Intelligent Transportation Systems	[10]			
	smart city		[11]		
	smart building	[12]			
	smart grid	[13]	[14]		
Wireless Communications and Networks	IoT specific	[15], [16]			
	General-purpose	[17]	[18], [19]		[20]
Cloud/Fog/Edge Computing		[21], [22]	[23]		[24]

B. Deep Reinforcement Learning

Reinforcement learning (RL) introduces ambient intelligence into the AIoT systems by providing a class of solution methods to the closed-loop problem of processing the sensory data to generate control decisions to react. Specifically, the agents interact with the environment to learn optimal policies that map status or states to actions [4]. The learning agent must be able to sense the current state of the environment to some extent (e.g., sensing room temperature) and take the corresponding action (e.g., turn thermostat on or off) to affect the new state and the immediate reward so that a long-term reward over extended time period is maximized (e.g., keeping room temperature at a target value). Different from most forms of ML, e.g., supervised learning, the learner is not told which actions to take but must discover which actions yield the most long-term reward by trying them out.

While RL has been successfully applied to a variety of domains, it confronts a main challenge when tackling problems with real-world complexity, i.e., the agents must efficiently represent the state of the environment from high-dimensional sensory data, and use these information to learn optimal policies. Therefore, deep reinforcement learning (DRL), in which RL is assisted with deep learning (DL), has been developed to overcome the challenge [5]. One of the most famous applications of DRL is AlphaGo, the first computer program which can beat a human professional on a full-sized 19×19 board.

C. Application of DRL in AIoT Systems

It turns out that the formulation of RL/DRL models for the real-world AIoT systems is not as straightforward as it may appear to be. There are two types of entities in an RL/DRL model as discussed above - environment and agent. Firstly, the **environment** in RL/DRL can be restricted to reflect only the physical system, or be extended to include the wireless networks, the edge/fog servers and cloud servers as well. This is because that the network and computation performance, such as communication/computation delay, power consumption and network reliability, will have important impacts on the control performance of the physical system. Therefore, the control actions in RL/DRL can be divided into two levels: (physical system) actuator control and (communications/computation) resources control, as shown in Fig. 1. The two levels of control can be separated or jointly learned and optimized. Secondly, the **agent** in RL is a logical concept that makes decisions on action selection. In the AIoT systems, the agent with ambient intelligence can reside in the IoT devices, the edge/fog servers, and/or the cloud servers as shown in Fig. 1. The time sensitiveness of the IoT application is an important factor to determine the location of the agents. For example in autonomous driving, images from an autonomous vehicle's camera needs to be processed in real-time to avoid an accident. In this case, the agent should reside locally in the vehicle to make fast decisions, instead of transmitting the sensory data to the cloud and return the predictions back to the vehicle. However, there are many scenarios that it is not easy to determine the optimal locations for the agents, which may involve solving an RL problem in itself. Moreover, when there are multiple agents distributed in the IoT devices, the cooperation of the agents is also an important and challenging issue.

D. Related Overview/Survey Articles

Although AIoT is a relatively new concept, related research works already exist in IoT and ACS, respectively. In this paper, we will review the state-of-art research, and identify the model and challenges for the application of DRL in AIoT.

The existing overview/survey articles related to this paper are summarized and classified in Table I. There are several recent survey articles discussing on the applications of ML/DL in general-purpose IoT systems for data analysis [3], [6]–[9]. In addition, the overview of ML/DL/RL/DRL applications in some specific physical autonomous systems or IoT application areas are provided in [10]–[13]. As wireless communications and networks are essential parts of IoT systems, the survey and overview on ML/DL/RL/DRL applications in IoT specific [15], [16] or general-purpose wireless networks [18]–[20] are also listed in Table I. Finally, there are also a few overview articles on applying ML/DL/RL/DRL techniques to cloud/edge/fog computing systems, which are important subsystems in the IoT ecosystem.

E. Contributions

This paper focuses on a specific type of ML, i.e., DRL, and its application on a promising type of IoT system, i.e., AIoT. To the best of our knowledge, there are currently no

survey/overview articles focusing specifically on the application of DRL in IoT system as shown in Table I. Moreover, the concept of AIoT as the future IoT system is relatively new and not adequately addressed in existing literature. The main contributions of this paper lie in the following aspects:

- A comprehensive tutorial of DRL is provided. We first explain the relationship between DRL and its two fundamental building blocks, i.e., RL and DL. Then, a tutorial and review on the basic DRL algorithms is given, where the DRL algorithms are classified into two broad categories, i.e., value-based and policy gradient. Different from existing surveys on DRL [25], [26], we explain the various DRL algorithms from a unified perspective - the input, output, and loss functions of neural networks (NNs) which are used to approximate the different functions in RL algorithms. Moreover, we discuss the pros and cons of each category of DRL algorithms. Finally, we introduce two types of advanced DRL models and related DRL algorithms that are extremely important for AIoT systems, i.e., partially observable Markov decision process (POMDP)-based DRL and multi-agent (MA) DRL.
- We propose a general DRL model for AIoT systems, where the environment is divided into perception layer, network layer, and application layer according to the IoT architecture. The RL elements including state, action, and reward for each layer as well as the integration of three layers are defined. The relationship between the logical layer and physical locations of an agent in the DRL model is discussed. The general DRL model not only creates a taxonomy to summarize and classify existing works, but also provide a framework to formulate DRL models for future works.
- The emerging research contributions on the applications of DRL in the AIoT systems are reviewed under the umbrella of the proposed general DRL model. First, the general procedure to tackle research problems in this area is introduced. Then, we review and compare the different research works according to (1) whether a basic DRL model or an advanced DRL model such as MA or POMDP is considered; (2) the elements of DRL model as well as the adopted DRL algorithms; (3) the physical locations of the agents and whether centralized or distributed implementation is considered. Finally, we compare between the proposed general DRL model in AIoT and the DRL models in existing literature to derive useful insights for future works.
- As a new and emerging research field, there are many challenges and open issues in applying DRL to provide autonomous control in AIoT systems. Four main challenges are identified and discussed, such as incomplete perception problem and delayed control problem. These discussions provide useful information for those readers who seek promising future research directions.

The remainder of the paper is organized as follows. In Section II, we review the RL/DRL methodologies. Section III introduces a general model for RL/DRL in AIoT with a detailed discussion on the key elements. In Section IV, the existing works are surveyed and compared. The challenges and open issues are identified and highlighted in Section V. Finally, the conclusion is given in Section VI.

II. Overview of Deep Reinforcement Learning

DRL has two fundamental building blocks - RL and DL, the basic concepts of which are introduced in Appendix A and B, respectively. In RL, a large amount of memory is usually required to store the value functions and Q-functions. In most of the real-world problems, the state sets are large, sometimes infinite, which makes it impossible to store the value functions or Q-functions in the form of tables. Therefore, the trial-and-error interaction with the environment is hard to be learned due to the formidable computation complexity and storage capacity requirements. This is where DL comes into the picture - some functions of RL such as value/Q-functions or policy functions are approximated with a smaller set of parameters by the application of DL. The combination of RL and DL results in the more powerful DRL.

In this section, we first classify the basic DRL algorithms into two broad categories, i.e., value-based and policy gradient methods, according to whether value/Q-functions or policy functions are approximated by NN as shown in Fig. 2. The policy gradient methods are further discussed from three aspects:

- Based on the different *natures of the approximated policy functions*, we introduce stochastic policy gradient (SPG) versus deterministic policy gradient (DPG) methods;
- Based on the different *ways of policy evaluation*, we introduce Monte Carlo policy gradient versus actor-critic methods;
- Based on the different *learning or parameter update techniques*, we introduce simple policy gradient versus natural policy gradient (NPG) methods.

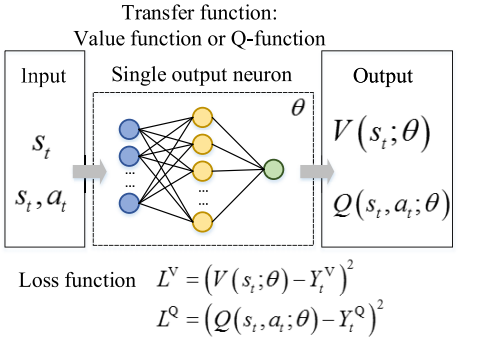
Then, we introduce two types of advanced DRL algorithms, i.e., POMDP-based DRL and MA-based DRL, that are envisioned to be extremely useful in addressing the open issues in AIoT. The organization of Section II is illustrated in Fig. 3.

A. Basic DRL Algorithms

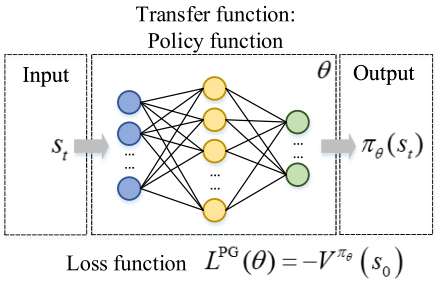
1) Value-Based Methods: In value-based methods for DRL as illustrated in Fig. 2(a), the states $s_t \in \mathcal{S}$ or state-action pairs $(s_t, a_t) \in \mathcal{S} \times \mathcal{A}$ are used as inputs to NNs, while Q-functions $Q^\pi(s_t, a_t)$ or value functions $V^\pi(s_t)$ are approximated by parameters θ of NNs. An NN returns the approximated Q-functions or value functions for the input states or state-action pairs. There can be a single output neuron or multiple output neurons as shown in Fig. 2(a). For the former case, the output can be either $V^\pi(s_t)$ or $Q^\pi(s_t, a_t)$ corresponding to the input s_t or (s_t, a_t). For the latter case, the outputs are the Q-functions for state s_t combined with every action, i.e., $Q^\pi(s_t, a^1), \ldots, Q^\pi(s_t, a^{|\mathcal{A}|})$.

To derive the loss functions, Y_t^{Q} and Y_t^{V} are defined as the target values of Q-functions and value functions, respectively. The regression loss

$$L^{\mathrm{Q}} = \left(Q(s_t, a_t; \theta) - Y_t^{\mathrm{Q}} \right)^2, \tag{1}$$



(a) General value-based methods for DRL.



(b) General policy gradient methods for DRL.

Fig. 2. General methods for DRL.

or

$$L^{\mathrm{V}} = \left(V(s_t;\theta) - Y_t^{\mathrm{V}}\right)^2, \tag{2}$$

can be used to evaluate how well the NN approximate Q-functions or value functions in value-based methods.

a) Deep Q-networks: Based on the idea of NN fitted Q-functions, the Deep Q-networks (DQN) algorithm is introduced by Mnih *et al.* in 2015 to obtain strong ability in ATARI games [5]. The illustration of DQN is shown in Fig. 4(a). The NN in DQN takes a state as input, and returns approximated Q-functions for every action under the input state.

In DQN, the algorithm first randomly initialize the parameters of networks as θ_0. The target Q-function Y_t^{DQN} is given by (3) according to Bellman equation as

$$Y_t^{\mathrm{DQN}} = r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta), \tag{3}$$

where the subscripts t or $t+1$ refer to the values of corresponding variables at the t^{th} or $(t+1)^{th}$ iteration.

The parameters in DQN are updated by minimizing the loss function L^{DQN}, which can be derived from (1) by replacing Y_t^{Q} with Y_t^{DQN}.

By applying stochastic gradient descent, the parameters are updated as

$$\theta \leftarrow \theta + \alpha\left(Y_t^{\mathrm{DQN}} - Q(s_t, a_t; \theta)\right)\nabla_\theta Q(s_t, a_t; \theta), \tag{4}$$

where α is the learning rate.

In order to deal with the limitations of DRL, two important techniques, freezing target networks and experience replay, are applied in DQN. To make the training process more stable and controllable, the target networks, whose parameters θ_t^- are kept fixed in a time period, are used to evaluate the Q-function of the next state, i.e., instead of (3), we have

$$Y_t^{\mathrm{DQN}} = r_{t+1} + \gamma \max_{a_{t+1}} Q\left(s_{t+1}, a_{t+1}; \theta^-\right). \tag{5}$$

The parameters of online network θ_t are updated after each iteration. After a certain number of iterations, the online network shares its parameters to the target network. This reduces the risk of divergence and prevents the instabilities resulted from the too quick propagation.

To perform experience replay, the experience of the agent at each time step is stored in a data set. Then, the updates are made on this data set, which removes correlations in the observation sequence and smooths over changes in the data distribution. This technique allows the updates to cover a wide range state-action space and provides more possibility to make larger updates of the parameters.

b) Double DQN: In DQN, the Q-function evaluated by target networks is used both to select and evaluate an action, which makes it more likely to overestimate the Q-function of an action. The estimating error will become larger if there are more actions. To overcome this problem, Hasselt *et al.* proposed a Double DQN (DDQN) method in 2016, where two sets of parameters are used to derive the target value Y_t^{DDQN} as shown in Fig. 4(b) [27]. Compared with (3), the target Q-value in DDQN can be rewritten as

$$Y_t^{\mathrm{DDQN}} = r_{t+1} + \gamma Q\left(s_{t+1}, \arg\max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta); \theta^-\right), \tag{6}$$

where the selection of the action is due to the parameters θ in online network and the evaluation of the current action is due to the parameters θ^- in target network. This means there will be less overestimation of the Q-Learning values and more stability to improve the performance of the DRL methods [26]. The loss function L^{DDQN} can be derived from (1) by replacing Y_t^{Q} with Y_t^{DDQN} and the parameters can be updated accordingly. DDQN algorithm gets the benefit of double Q-Learning and keeps the rest of DQN algorithm.

Apart from DQN and DDQN, there are also other value-based methods, some of which are developed based on DQN and DDQN with some further improvement, such as DDQN with Proportional Prioritization [28], and DDQN with duel architecture [29].

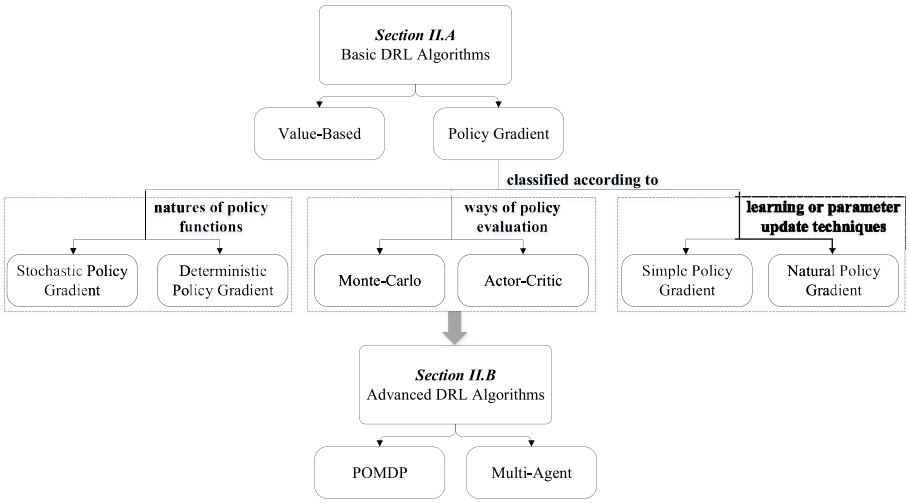


Fig. 3. Organization of Section II.

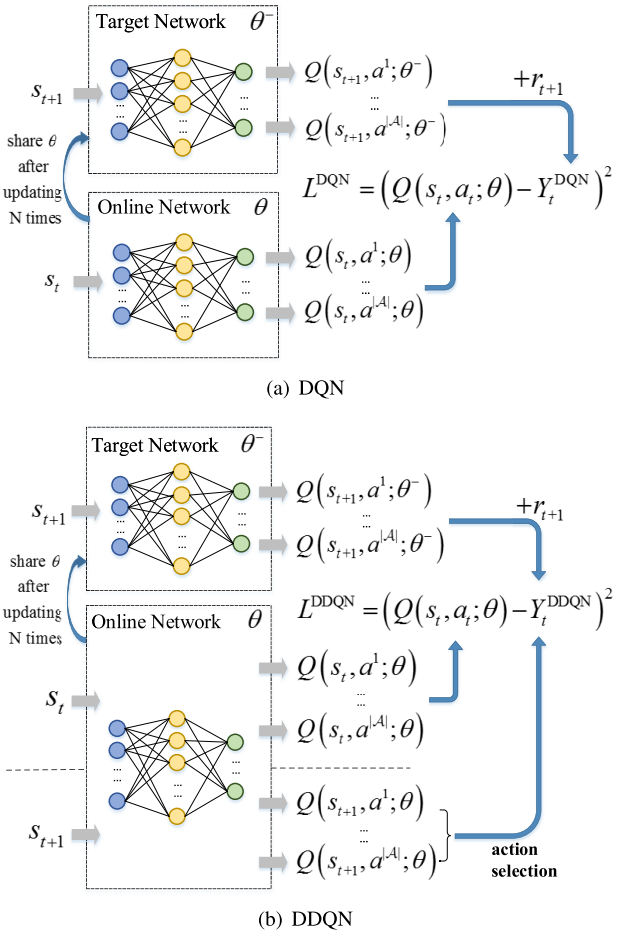


Fig. 4. DQN and DDQN with target networks.

Remark 1 (Pros and Cons of Value-Based DRL Methods): Although DQN and its improved versions have been widely adopted in existing literature as discussed in Section IV - mainly due to their relative simplicity and good performance, there are some limitations with value-based DRL methods. First, it cannot solve RL problems with large or continuous action space. Second, it cannot solve RL problems where the optimal policy is stochastic requiring specific probabilities. Since value-based method can only learn deterministic policies, the majority of the algorithms are off-policy, such as DQN.

2) Policy Gradient Methods: According to a policy π, action a is selected when the environment is in state s. In policy gradient methods, NNs can be applied to directly approximate a policy as a function of state, i.e., $\pi_\theta(s)$. As shown in Fig. 2(b), the states are used as inputs to the NNs, while policy π is approximated by parameters θ of NNs as π_θ.

To evaluate the performance of the current policy, the objective function is defined as

$$J(\theta) = V^{\pi_\theta}(s_0) = \mathrm{E}_{\tau_{s_0} \sim \pi_\theta}[G(\tau_{s_0})], \forall s_0 \in \mathcal{S}, \tag{7}$$

where $V^{\pi_\theta}(s_0)$ is the value function of policy π_θ as shown in (24), and τ_{s_0} refers to the sampling trajectory with an initial state s_0. If we can find the parameters θ for policy π_θ so that the objective function $J(\theta)$ is maximized, we can solve the problem. The basic idea of policy gradient methods are to adjust the parameters in the direction of greater expected reward [30]. For this purpose, we can set the loss function of NN to be

$$L^{\mathrm{PG}}(\theta) = -J(\theta) = -V^{\pi_\theta}(s_0). \tag{8}$$

In order to update the parameters, we need to express the gradient of $J(\theta)$ with respect to parameter θ as an expectation of stochastic estimates based on (7). As mentioned in Section II-A, the policy in RL can be classified into two categories, i.e., the stochastic policy and the deterministic policy. Hence, the SPG method and DPG method are correspondingly discussed below.

a) Stochastic policy gradients vs. deterministic policy gradient: By applying DRL, a stochastic policy is approximated as $\pi_\theta = \pi(a_t|s_t;\theta)$, which gives the probability of a specific action a is taken in a specific state s, when the agent follows the policy parameterized by θ. The policy parameters are usually the weights and bias of a NN [26]. For a DRL model with discrete state/action spaces, Softmax function is a

typical probability density function. In the cases of continuous state/action spaces, Gaussian distribution is generally used to characterize the policy. An NN is applied to approximate the mean, and a set of parameters specifies the standard deviation of the Gaussian distribution [31], [32].

According to the policy gradient theorem, we have

$$\nabla \mathrm{E}_{\tau_{s_0}\sim\pi_\theta}\quad [G(\tau_{s_0})] = \mathrm{E}_{\tau_{s_t}\sim\pi_\theta}[G(\tau_{s_t})\nabla_\theta \log \pi(a_t|s_t;\quad \theta)]. \tag{9}$$

By applying stochastic gradient descent, the parameters are updated as

$$\theta \leftarrow \theta + \alpha G(\tau_{s_t})\nabla_\theta \log \pi(a_t|s_t;\theta), \tag{10}$$

where α is the learning rate. In this way, θ is adjusted to enlarge the probability of trajectory $G(\tau_{s_t})$ with higher total reward.

From the perspective of NN, we give the loss function of SPG algorithm as

$$L^{\mathrm{SPG}}(\theta) = -G(\tau_{s_t}) \log \pi(a_t|s_t;\quad \theta). \tag{11}$$

Different from SPG where the policy is modeled as a probability distribution over actions, DPG models the policy as a deterministic decision, i.e., $\pi_\theta = \pi(s_t;\theta)$. According to the objective function given in (7) and the DPG theorem, we have

$$\nabla_\theta J(\theta) = \mathrm{E}_{s\sim\rho^{\pi_\theta}}\left[\nabla_\theta \pi(s_t;\ \theta)\nabla_a Q^{\pi_\theta}(s_t, a_t;\ \phi)|_{a=\pi(s_t;\theta)}\right], \tag{12}$$

where the policy improvement is decomposed into the gradient of the Q-function with respect to actions, and the gradient of the policy with respect to the policy parameters. ρ^{π_θ} is the state distribution following policy π_θ. Thus, the parameters are updated as

$$\theta \leftarrow \theta + \alpha\left[\nabla_\theta \pi(s_t;\theta)\nabla_a Q^{\pi_\theta}(s_t, a_t;\phi)|_{a=\pi(s_t;\theta)}\right]. \tag{13}$$

A differentiable function $Q^w(s_t, a_t;\phi)$ can be used as an approximator of $Q^{\pi_\theta}(s_t, a_t;\phi)$, and then the gradient $\nabla_a Q^{\pi_\theta}(s_t, a_t;\phi)$ can be replaced by $\nabla_a Q^w(s_t, a_t;\phi)$. The approximator is compatible with the deterministic policy, and $\nabla_a Q^w.(s_t, a_t;\phi)|_{a=\pi(s_t;\theta)}$ is achieved as $\nabla_\theta \pi(s_t;\theta)^\top w$ [33].

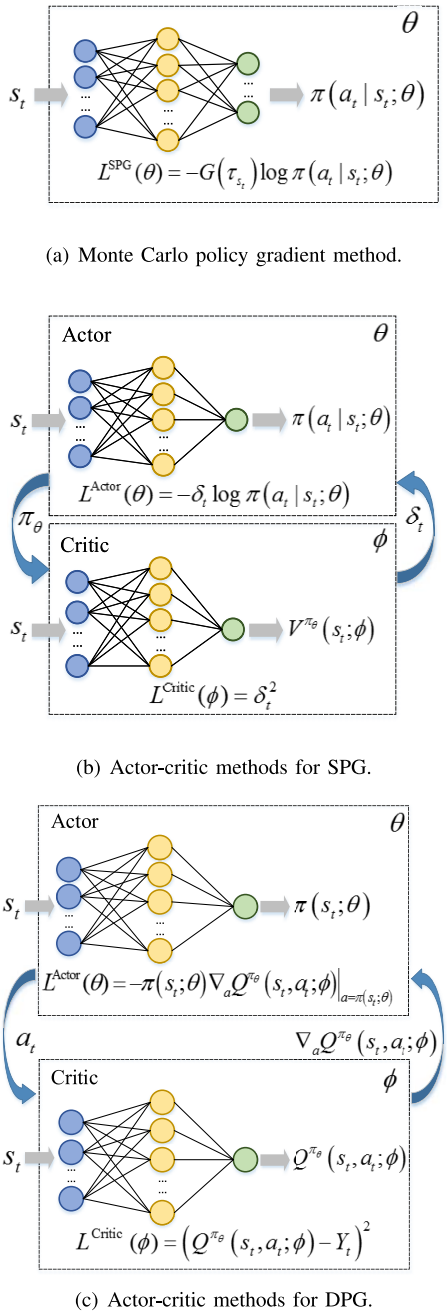
From the perspective of NN, the loss function of DPG algorithm is set as

$$L^{\mathrm{DPG}}(\theta) = -\pi(s_t;\theta)\nabla_a Q^{\pi_\theta}(s_t, a_t;\phi)|_{a=\pi(s_t;\theta)}. \tag{14}$$

b) Monte Carlo policy gradient vs. actor-critic: In (11) and (14), the value of $G(\tau_{s_t})$ and $\nabla_a Q^{\pi_\theta}$ need to be derived to update the policy parameters θ in SPG and DPG, respectively. For SPG, this can be achieved either by Monte Carlo policy gradient method or actor-critic method, as is illustrated in Fig. 5(a) and Fig. 5(b), respectively. For DPG, this is normally achieved by actor-critic method as is shown in Fig. 5(c).

The Monte Carlo policy gradient method tries to evaluate $G(\tau_{s_t})$ through Monte Carlo simulation. A typical Monte Carlo algorithm of the SPG methods is the REINFORCE algorithm proposed in [34]. Based on the Monte Carlo approach, a trajectory τ_{s_0} is firstly sampled by running the current policy from an initial state s_0. Then for each time step $t = 0, 1, \ldots, T$, the total reward $G(\tau_{s_t})$ starting from time step t is calculated, which is multiplied with the policy gradient $\nabla_\theta \log \pi(a_t|s_t;\theta)$ to update the parameters θ according to (10). The above procedure is repeated over multiple runs, while in each run a different trajectory is sampled.

Moreover, in order to reduce the variance of the policy gradient, a baseline function $b(s_t)$ which is independent of a_t is introduced. Based on this, the REINFORCE algorithm with baseline is introduced, and the loss function of it can be



(a) Monte Carlo policy gradient method.

(b) Actor-critic methods for SPG.

(c) Actor-critic methods for DPG.

Fig. 5. Monte Carlo policy gradient versus actor-critic methods.

formulated as

$$L^{\text{SPG\_BASE}}(\theta) = -(G(\tau_{s_t}) - b(s_t)) \log \pi(a_t|s_t;\theta). \quad (15)$$

Remark 2 (Pros and Cons of Monte Carlo Policy Gradient DRL Methods): In contrast to value-based DRL methods, the policy gradient methods for DRL is a direct mapping from state to action, which leads to better convergence properties and higher efficiency in high-dimensional or continuous action spaces [26]. Moreover, it can learn stochastic policies, which have better performance than deterministic policies in some situations. However, Monte Carlo policy gradient methods suffer from high variance of estimations. As on-policy methods, they require on-policy samples, which made them very sample intensive.

Actor-critic methods are well-known for combining the advantages of both Monte Carlo policy gradient and value-based methods, and they have been widely studied in DRL. As illustrated in Fig. 5(b) and Fig. 5(c), an actor-critic method is generally realized by two NNs, i.e., the actor network and the critic network, which share parameters with each other. The actor network is similar to the NN of the policy gradient method, while the critic network is similar to the NN of the value-based method. During the learning process, the critic updates the parameters of value functions, i.e., ϕ, according to the policy given by the actor. Meanwhile, the actor updates the parameters of policy, i.e., θ, according to the value functions evaluated by the critic. Generally, two learning rates are required to be predefined respectively for the updates of ϕ and θ [35].

In the actor-critic method for SPG, the critic network is used obtain the value of $G(\tau_{s_t})$ in (11). Specifically, the baseline $b(s_t)$ in (15) is set to the value function $V^{\pi_\theta}(s_t;\phi)$, which is approximated by the critic network with a loss function as given in (2). In a state s_t, the agent selects an action a_t according to the current policy π_θ given by the actor network, receives a reward r_{t+1}, and the state transits to s_{t+1}. Similar to (2) in value-based method, the loss function for the critic network can be expressed as

$$L^{\text{Critic}}(\phi) = {\delta_t}^2, \quad (16)$$

where

$$\delta_t = r_{t+1} + \gamma V^{\pi_\theta}(s_{t+1};\phi) - V^{\pi_\theta}(s_t;\phi), \quad (17)$$

Similar to (4) in DQN, the parameters of the critic network are updated as

$$\phi \leftarrow \phi + \beta \delta_t \nabla_\phi V^{\pi_\theta}(s_t;\phi), \quad (18)$$

where β is the learning rate for the critic.

Note that $G(\tau_{s_t})$ is an estimate of $Q^{\pi_\theta}(s_t, a_t;\phi) = r_{t+1} + \gamma V^{\pi_\theta}(s_{t+1};\phi)$. Therefore, given the value functions evaluated by the critic network, the value of $G(\tau_{s_t}) - b(s_t)$ in (15) can be replaced by δ_t in (17), which can be seen as an estimate of the advantage of action a_t in state s_t [36]. The loss function of the actor network can be defined similar to (15), i.e.,

$$L^{\text{Actor}}(\theta) = -\delta_t \log \pi(a_t|s_t;\theta). \quad (19)$$

Similar to (10) in the policy gradient method, the parameters of the actor network are updated as

$$\theta \leftarrow \theta + \alpha \delta_t \nabla_\theta \log \pi(a_t|s_t;\theta), \quad (20)$$

where α is the learning rate for the actor.

Through the update processes in the actor-critic algorithm, the critic can make the approximation of value functions more accurately, while the actor can choose better action to get higher reward.

Typical actor-critic methods for SPG include the asynchronous advantage actor-critic (A3C) algorithm and soft actor-critic (SAC). The former mainly focuses on the parallel training of multiple actors that share global parameters [37]. The latter involves a soft Q-function, a tractable stochastic policy and off-policy updates [38]. SAC achieves good performance on a range of continuous control tasks.

One typical actor-critic method for DPG is the deep deterministic policy gradient (DDPG) algorithm. The DDPG algorithm is a model-free off-policy actor-critic algorithm, which combines the ideas of DPG and DQN. It is first proposed by Lillicrap *et al.* in 2015 [39]. Besides the online critic network Q with parameters ϕ, and the online actor network π with parameters θ, the target networks Q' and π' in the DDPG algorithm are specified with ϕ' and θ', respectively. The parameters of these four NNs are required to be updated in the learning process. The gradient $\nabla_a Q^{\pi_\theta}(s_t, a_t;\phi)$ is obtained by the critic network.

Based on DDPG, several algorithms are proposed in recent years, such as Distributed Distributional Deep Deterministic Policy Gradients (D4PG) [40], Twin Delayed Deep Deterministic (TD3) [41], Multi-Agent DDPG (MADDPG) [42], and Recurrent Deterministic Policy Gradients (RDPG) [43].

Remark 3 (Pros and Cons of Actor-Critic DRL Methods): Actor-critic methods combine the advantages of both value-based and Monte Carlo policy gradient methods. They can be either on-policy or off-policy. Compared with Monte Carlo methods, they require far less samples to learn from and less computational resources to select an action, especially when the action space is continuous. Compared with value-based methods, they can learn stochastic policies and solve RL problems with continuous actions. However, it is prone to be unstable due to the recursive use of value estimates.

From the above discussion, we know that in Monte Carlo methods, the policy gradient is unbiased but with high variance; while in actor-critic methods, it is deterministic but biased. Therefore, an effective way is to combine these two types of methods together. Q-prop is such an efficient and stable algorithm proposed by Gu *et al.* in 2016 [44]. It constructs a new estimator that provides a solution to high sample complexity and combines the advantages of on-policy and off-policy methods.

Q-prop can be directly combined with a number of prior policy gradient DRL methods, such as DDPG and TRPO. Compared with actor-critic methods such as DDPG algorithms, Q-prop has achieved higher stability in DRL tasks in real-world problems. One limitation with Q-prop is that the

computation speed will be slowed down by the critic training when the speed of data collection is fast.

c) Simple policy gradient vs. natural policy gradient: The policy gradient methods discussed above all use a simple gradient of loss function $\nabla_\theta L(\theta)$ to update the parameters of NN. On the other hand, NPG method updates the parameters in NN using the natural gradient $\nabla_\theta^{\mathrm{N}} L(\theta)$ as discussed in Section II-B instead of simple gradient to provide a more efficient solution [31].

The loss function of NPG is the same as that of SPG, whose general expression is given in (8). The parameters are updated as

$$\theta \leftarrow \theta + F_\theta^{-1} \nabla_\theta V^{\pi_\theta}(s), \tag{21}$$

where

$$F_\theta = \mathrm{E}_{\pi_\theta}\left[\nabla_\theta \log \pi(a_t|s_t;\theta)(\nabla_\theta \log \pi(a_t|s_t;\theta))^T\right] \tag{22}$$

is the Fisher information matrix used to measure the step size for update [32].

NPG method defines a new form of step size that specifies how much those parameters should be adjusted, and therefore provides a more stable and effective update. However, the drawback of NPG is that when complicated NN is used to approximate the policy where the number of parameters is large, it is impractical to calculate the Fisher information matrix or store them appropriately [26]. Methods originated from NPG, such as Trust Region Policy Optimization (TRPO) [32] and Proximal Policy Optimization (PPO) [45] solve the above problem to some extent and are widely used for DRL in practice. Moreover, there are algorithms applying NPG to actor-critic methods, such as Actor Critic using Kronecker-Factored Trust Region (ACKTR) [46] and Actor-Critic with Experience Replay (ACER) [36].

B. Advanced DRL Algorithms

1) POMDP-Based DRL: In the previous sections, we consider RL in a Markovian environment, which implies that knowledge of the current state is always sufficient for optimal control. However in many real-world problems, total environment information cannot be observed by the agent accurately, usually due to the limitations in sensing and communications capabilities. An agent acting under situation with partial observability can model the environment as a POMDP [47]. RL tasks in realistic environments need to deal with those incomplete and noisy state information resulting from POMDP.

POMDP can be seen as an extension of MDP by adding a finite set of observations and a corresponding observation model [48]. A POMDP is usually defined as a six-tuple $<\mathcal{S}, \mathcal{A}, P, r, \Omega, \mathcal{O}>$, where state space $\mathcal{S}$, action space $\mathcal{A}$, transition probability *P*, and reward *r* are defined previously as elements in MDP,

- Ω is the observation space, where $o \in \Omega$ is a possible observation.
- $\mathcal{O}(o|s', a)$ is the conditional probability that taking an action a leading to a new state s' will result in an observation o.

Similar to MDP, an agent chooses an action $a \in \mathcal{A}$ according to policy $\pi(a|s)$ which results in the environment transiting to a new state $s' \in \mathcal{S}$ with probability $P(s'|s, a)$ and the agent receives a reward *r*(*s, a*). Different from MDP, the agent cannot directly observe system states, but instead receives an observation $o \in \Omega$ which depends on the new state of the environment with probability $\mathcal{O}(o|s', a)$. Also, the policy and Q-function are modified as $\pi(a|o)$ and *Q*(*o, a*) respectively.

Since the agent cannot directly observe the underlying state, it needs to exploit history information to reduce uncertainty about the current state [49]. The observation history at time step t can be defined as $h_t = \{(o_1, a_1), \ldots, (o_{t-1}, a_{t-1}), (o_t, -)\}$.

Several typical existing methods of solving POMDP problems are listed as follows.

a) Deep recurrent Q-network (DRQN): To address the partial observable problem, Hausknecht and Stone proposed Deep Recurrent Q-Network (DRQN) in 2015 to integrate information through time and enhance DQN's performance [50]. DRQN adds recurrency to DQN by replacing DQN's first fully-connected layer with a LSTM layer.

In the partially observed cases, the agent does not have access to state s_t. So Q-function in terms of history h_t is defined as $Q(h_t, a_t)$, which is the output of NN [43]. The input to NN is o_t, while the rest of the information in h_t apart from o_t, i.e., $\{(o_1, a_1), \ldots, (o_{t-1}, a_{t-1})\}$ is captured by the hidden states in RNN.

b) Recurrent policy gradients (RPG): RPG methods belong to policy gradient methods where NNs are used to approximate policies [49]. As mentioned in Section IV-D, in policy gradient methods, $\pi(s)$ or $\pi(a|s)$ is a direct mapping from state *s* to action *a*. But in RPG, the goal of the agent is to learn a policy that maps history *h* to action *a*, which is denoted as $\pi(h)$ or $\pi(a|h)$.

RPG methods are applied to many partially observed physical control problems, i.e., system identification with variable and unknown information, short-term integration of sensor information to estimate the system state, as well as long-term memory problems. A typical algorithm, Recurrent Deterministic Policy Gradient (RDPG), is proposed by Heess *et al.* based on RPG methods [43].

c) Memory, RL, and inference network (MERLIN): MERLIN algorithm focuses on memory-dependent policies which output the action distribution based on the entire observation sequence in the past [51]. The ideas for MERLIN, including predictive sensory coding, hippocampal representation theory and temporal context model, mainly originate in neuroscience and psychology. It is mainly composed of two basic components: a memory-based predictor and a policy.

d) Deep belief Q-network (DBQN): DBQN is a model-based method that uses DQN to map a belief b_t to an action. When *P*, *o* and *r* in a POMDP model are known, b_t can be estimated accurately with Bayes' theorem and sent to NN as input [52]. During updating, this approach usually leads to divergence. To stabilize the learning, techniques like experience replay, target network and an adaptive learning method are used.

Besides, there are also other methods of solving POMDP problems, some of which are developed based on RNN and typical methods for DRL, such as Action-specific Deep Recurrent Q-Network (ADRQN) [53], and Deep Distributed Recurrent Q-Networks (DDRQN) [54].

2) Multi-Agent DRL: In the previous sections, we mainly discuss the DRL methods for single-agent cases. In practice, there are situations where multiple agents need to work together, e.g., the manipulation in multi-robot systems, the cooperative driving of multiple vehicles. In these cases, DRL methods for MA systems are designed.

An MA system consists of a group of autonomous, interacting agents sharing a common environment, and has a good degree of robustness and scalability [55]. The multiple agents in the system can interact with each other in cooperative or competitive settings, and hence the concept of stochastic game is introduced to extend MDP into the MA setting. A stochastic game or MA-MDP with N agents is defined as a tuple $<\mathcal{S}, \mathcal{A}_1, \ldots, \mathcal{A}_N, P, r_1, \ldots, r_N>$, where

- $\mathcal{S}$ is the discrete set of states,
- $\mathcal{A}_i$, $i = 1, \ldots, N$ are the discrete sets of actions available to the agents, yielding the joint action set $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_N$,
- $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state transition probability function,
- $r_i : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, $i = 1, \ldots, N$ are the reward functions for the agents.

In MA-MDP, the state transitions depend on the joint action $a = [a_1, \ldots, a_n]$ of all the agents, where $a \in \mathcal{A}$ and $a_i \in \mathcal{A}_i$. In the fully-collaborative problems, all the agents share the same reward, i.e., $r_1 = \cdots = r_N$. In the fully-competitive problems, the agents have opposite rewards with $r_1 + \cdots + r_N = 0$. Therefore, $r_1 = -r_2$ in the typical scenario with two agents [55]. MA-MDP problems that are neither fully collaborative nor fully competitive are mixed games.

In MA RL, each agent learns to improve its own policy by interacting with the environment to obtain rewards. For each agent, the environment is usually complex and dynamic, and the system may encounter the action space explosion problem. Since multiple agents are learning at the same time, for a particular agent, when the policies of other agents change, the optimal policy of itself may also change. This may affect the convergence of the learning algorithm and cause instability.

In recent years, the DRL methods for single-agent cases have been extended to the MAs cases as discussed below.

a) Multi-agent value-based methods: The experience replay mechanism in DQN algorithm is not designed for the non-stationary environment in MA systems. Several variants of DQN have been proposed to deal with this problem.

Foerster *et al.* [56] introduced two methods for stabilizing experience replay of DQN in MA DRL. In the MA importance sampling (MAIS) algorithm, off-environment importance sampling is introduced to stabilize experience replay, where obsolete data is supposed to decay naturally. In the MA fingerprints (MAF) algorithm, each agent needs to be able to condition on only those values that actually occur in its replay memory to stabilize experience replay.

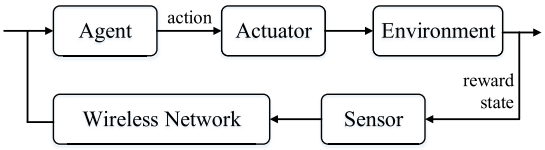


Fig. 6. The RL/DRL model for WSAN.

In [57], a coordinated MA DRL method is designed based on DQN. Faster and more scalable learning is realized by using transfer planning. To coordinate between multiple agents, the global Q-function is factorized as a linear combination of local sub-problems. Then, the max-plus coordination algorithm is applied to optimize the joint global action over the entire coordination graph.

b) Multi-agent policy gradient methods: Policy gradient methods usually exhibit very high variance when coordination of multiple agents is required. In order to overcome this challenge, several algorithms adopt the framework of centralized training with decentralized execution.

In the counterfactual MA policy gradient (COMAPG) algorithm [58], a centralized critic is used to estimate the Q-function, and decentralized actors are used to optimize the policies of multiple agents. The core idea of the COMAPG algorithm is to apply a counterfactual baseline, which can marginalize out a single agents action and keep the other agents actions fixed. Moreover, a critic representation is introduced for efficiently evaluating the counterfactual baseline in a single forward pass.

MA deep deterministic policy gradient (MADDPG) [42] is essentially a DPG algorithm that trains each agent with a critic that requires global information and an actor that requires local information. It allows each agent to have its own reward function, so that it can be used for cooperative or competitive tasks.

Based on the above tutorial, we list the classical algorithms for each type of DRL methods in Table II and summarize their pros and cons.

III. General RL/DRL Model for Autonomous IoT

Before we discuss the RL/DRL model for the AIoT system, we first examine that of a wireless sensor and actuator network (WSAN), which can be considered as an element or a simplified version of AIoT. A WSAN consists of a group of sensors that gather information about their environment, and a group of actuators that interact with and act on the environment. All elements communicate wirelessly. In the RL/DRL model for WSAN as illustrated in Fig. 6, an agent obtains aspects of its environment through sensors, and chooses control actions that are implemented by the actuators. The chosen action determines the value of the immediate reward as well as influences the dynamics of its environment. The agent communicates with the sensors and actuators to receive state information and send control commands.

Compared with WSAN, the AIoT has a more complex ecosystem that encompasses identification, sensing, communication, computation, and services. A typical AIoT architecture

TABLE II
CLASSICAL ALGORITHMS FOR DRL

Feature	Type			Classical Algorithms	Pros & Cons
basic	Value-Based			Deep Q-network (DQN) [5], Double Deep Q-network (DDQN) [26], DDQN with duel architecture [29], DDQN with Proportional Prioritization [28]	simplicity and good performance; only suitable for discrete action space
	Policy Gradients	classified according to natures of policy functions	Stochastic Policy Gradient (SPG)	REINFORCE [34], Soft Actor-Critic (SAC) [38], Asynchronous Advantage Actor Critic (A3C) [37]	/
			Deterministic Policy Gradient (DPG)	Deep Deterministic Policy Gradient (DDPG) [39], Distributed distributional deep deterministic policy gradients (D4PG), Twin Delayed Deep Deterministic (TD3) [41]	requiring less samples; only suitable for continuous action space
		classified according to ways of policy evaluation	Monte Carlo	REINFORCE [34], Trust Region Policy Optimization (TRPO) [32], Proximal Policy Optimization (PPO) [45], Trust Region Policy Optimization (TRPO) [32], Proximal Policy Optimization (PPO) [45]	better convergence properties, higher efficiency in high-dimensional or continuous action spaces; high variance of estimations, sample intensive
			Actor-Critic	Soft Actor-Critic (SAC) [38], Asynchronous Advantage Actor Critic (A3C) [37], Deep Deterministic Policy Gradient (DDPG) [39], Distributed distributional deep deterministic policy gradients (D4PG), Twin Delayed Deep Deterministic (TD3) [41], Trust Region Policy Optimization (TRPO) [3], Proximal Policy Optimization (PPO) [45]	requiring less samples and less computational resources; unstable due to the recursive use of value estimates
			Monte Carlo & Actor-Critic	Q-Prop [44]	higher stability; slow computation speed when the speed of data collection is fast
		classified according to learning or parameter update techniques	Simple Policy Gradient	REINFORCE [34], Soft Actor-Critic (SAC) [38], Asynchronous Advantage Actor Critic (A3C) [37], Deep Deterministic Policy Gradient (DDPG) [39], Distributed distributional deep deterministic policy gradients (D4PG), Twin Delayed Deep Deterministic (TD3) [41]	/
			Natural Policy Gradient (NPG)	Trust Region Policy Optimization (TRPO) [32], Proximal Policy Optimization (PPO) [45], Actor Critic using Kronecker-Factored Trust Region (ACKTR) [46], Actor-Critic with Experience Replay (ACER) [36]	more stable and effective update; impractical to calculate the Fisher information matrix when complicated NN is used
advanced	POMDP			Deep Belief Q-network (DBQN) [53], Deep Recurrent Q-network (DRQN) [50], Recurrent Deterministic Policy Gradients (RDPG) [43]	/
	MA			Multi-agent Importance Sampling (MAIS) [56], Coordinated Multi-agent DQN [57], Multi-agent Fingerprints (MAF) [56], Counterfactual Multi-agent Policy Gradient (COMAPG) [59], Multi-agent DDPG (MADDPG) [42]	

consists of three fundamental building blocks as shown in Fig. 7:

- *Perception layer:* corresponds to the **physical autonomous systems** in which IoT devices with sensors and actuators interact with the environment to acquire data and exert control actions;
- *Network layer:* corresponds to the **IoT communication networks** including wireless access networks and the Internet that discover and connect the IoT devices to the edge/fog servers and cloud servers for data and control command transmission;
- *Application layer:* corresponds to the **IoT edge/fog/cloud computing systems** for data processing/storage and control actions determination.

Due to the more sophisticated system architecture, the RL/DRL models for AIoT systems are more complex than those of WSAN as illustrated in Fig. 6. The environment can include one or more layers in the AIoT architecture. The agent(s) can locate at the IoT devices, the edge/fog/cloud servers, and wireless APs. In the following, we first define the basic RL/DRL elements such as state, action, and reward for each layer, respectively. Then, we define the RL/DRL elements when the environment includes all the three layers as an integrated part.

A. Perception Layer

When the environment only includes the perception layer, the physical system dynamics are modeled by a controlled stochastic process with the following state, action, and reward.

- **Physical system state** (s_{phy}), e.g., the on-off status of the actuators, the RGB images of the system, the locations of the agents;
- **Actuator control action** (a_{actu}), e.g., controlling the movement of a robot, adjusting the driving speed and direction of a vehicle, turning on/off a device;

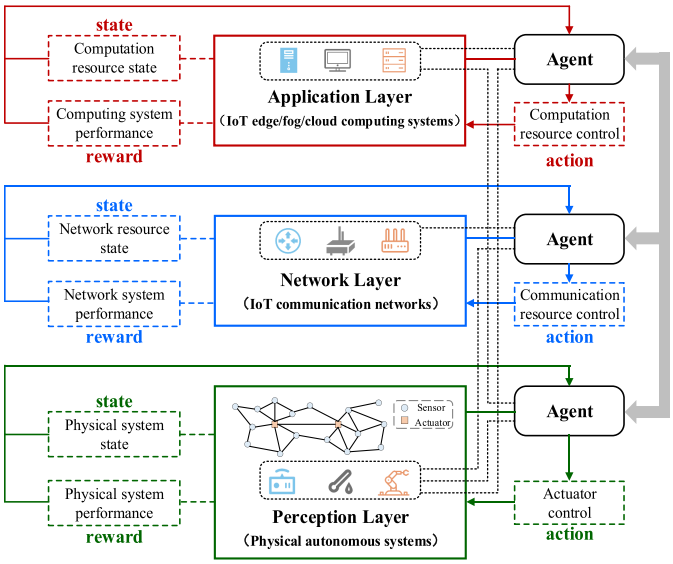


Fig. 7. General RL/DRL model for autonomous IoT.

- **Physical system performance** (r_{phy}), e.g., energy consumption in a power grid, how fast a mobile agent such as a robot or a vehicle can move, or whether it is away from obstacles.

B. Network Layer

When the environment only includes the network layer, the network dynamics are modeled by a controlled stochastic process with the following state, action, and reward.

- **Communication network state** (s_{net}), e.g., the amount of allocated bandwidth, the signal to interference plus noise ratio, the channel vector of a finite state Markov channel model;
- **Communication resource control action** ($a_{\text{comm\_re}}$), e.g., the power allocation, the multi-user scheduling, the subchannel allocation in OFDM system;
- **Communication network performance** (r_{net}), e.g., the transmission delay, the transmission error probability, the transmission power consumption.

C. Application Layer

When the environment only includes the application layer, the edge/fog/cloud computing system dynamics are modeled by a controlled stochastic process with the following state, action, and reward.

- **Computing system state** (s_{comp}), e.g., the number of virtual machines (VMs) that currently run, the number of tasks buffered in the queue for processing;
- **Computing resource control action** ($a_{\text{comp\_re}}$), e.g., the caching selection, the task offloading decisions, the virtual machine allocation;
- **Computing system performance** (r_{comp}), e.g., utilization rate of the computing resources, the processing delay of the offloading tasks.

D. Integration of Three Layers

When the environment includes all the three layers of AIoT architecture, the RL/DRL models generally include elements defined as follows.

- **AIoT state** (s_{aIoT}) includes the aggregation of physical system state, network resource state, and computation resource state, i.e., $s_{\text{aIoT}} = \{s_{\text{phy}}, s_{\text{net}}, s_{\text{comp}}\}$;
- **AIoT action** (a_{aIoT}) includes the aggregation of actuator control action, communication resource control action, and computing resource control action, i.e., $a_{\text{aIoT}} = \{a_{\text{actu}}, a_{\text{comm\_re}}, a_{\text{comp\_re}}\}$;
- **AIoT reward** (r_{aIoT}) is normally set to optimize the physical system performance, which can be expressed as a function of the network performance and computing system performance, i.e., $r_{\text{aIoT}} = r_{\text{phy}}(r_{\text{net}}, r_{\text{comp}})$.

As the agent in RL/DRL is a logical concept, the RL/DRL problem in each layer can be solved by the agent in its respective layer - observing the states and rewards from its environment and learning polices to determine corresponding actions as shown in Fig. 7. However, the physical location of an agent can be different from its logical layer. We classify the devices that an agent may locate in according to the physical locations of the devices as

- perception layer devices, i.e., IoT devices;
- network layer devices, i.e., wireless APs;
- application layer devices, i.e., edge/fog/cloud servers.

As shown in Fig. 7, the mapping of the logical layer of an agent and its physical locations are given. A perception layer agent may locate in IoT devices and/or edge/fog/cloud servers. A network layer agent may locate in wireless APs and/or IoT devices (e.g., for Device-to-Device (D2D) communications). An application layer agent may locate in edge/fog/cloud servers and/or even IoT devices (e.g., to perform task offloading).

When the environment of an RL/DRL problem includes more than one layer, the agents of different layers need to share information and jointly optimize their polices. For example, the network layer may provide transmission delay information to the perception layer to be included as part of the system state; or, the perception layer may provide its optimization objective to the network layer to formulate the reward function. When the physical locations of the agents of different layers are the same, e.g., when both perception layer agent and application layer agent locate at the cloud servers, a single logical agent combining agents of different layers can be considered for the RL/DRL problem.

IV. Applications of Deep Reinforcement Learning in Autonomous IoT

Although AIoT is a new trend in IoT that has not been adequately studied by existing research works, the respective applications of DRL in each of the three layers of AIoT architecture have been widely studied by recent works. Therefore, we provide a literature review of the applications of DRL in the perception layer (physical autonomous systems), the network layer (IoT communication networks), and the application layer (IoT edge/fog/cloud computing systems) in this section. Most

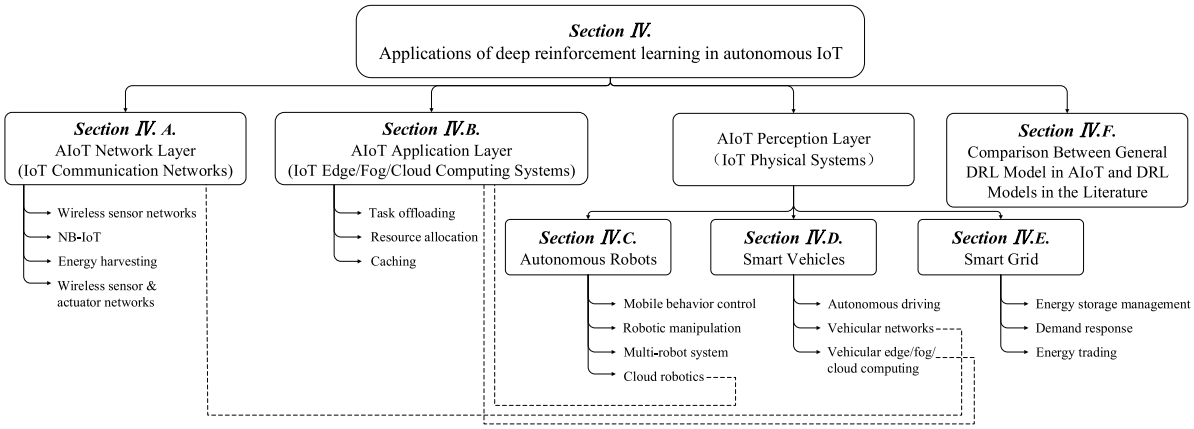


Fig. 8. Applications of deep reinforcement learning in autonomous IoT.

literature discussed in this section was published in the past decade, and was collected by searching in the IEEE Xplore using the words of related application and "deep reinforcement learning" as keywords. Moreover, we also found some other works in Google Scholar as a supplement. As there are a great variety of physical autonomous systems, we focus on three types of systems that have received most attention in DRL research for the perception layer, i.e., autonomous robots, smart vehicles, and smart grid.

In this section, we first discuss the applications of DRL in IoT communication networks and IoT edge/fog/cloud computing systems for general autonomous physical systems in Section IV-A and Section IV-B, respectively. Then, we focus on the three types of physical autonomous systems, i.e., autonomous robots, smart vehicles, and smart grid, in Section IV-C, IV-D, and IV-E, respectively. Note that some IoT communication technologies and IoT edge/fog/cloud computing technologies are designed specifically for a particular physical autonomous system, e.g., vehicular edge/fog/cloud computing and vehicular networks for smart vehicles, and cloud robotics for autonomous robots. These technologies are discussed in the respective physical autonomous system subsections. Finally, in Section IV-F, we compare the general DRL model in AIoT proposed in Section III with the reviewed DRL models in the existing literature. The framework of the literature review is given in Fig. 8.

Before we review the existing literature, we first provide the general procedure for solving optimal control problems in AIoT by DRL theory as shown in Fig. 9. Firstly, a system model needs to be given, which defines the dynamic behavior, control action, and performance criteria for a sequential decision problem in an AIoT system. Secondly, a DRL model is formulated based on the system model. The most essential part is to define the DRL elements including state, action, and reward functions. The transition probability of the states can also be given whenever available. Moreover, it is important to identify the features of the DRL model. Generally speaking, the DRL model can be either a basic MDP model, a POMDP model or an MA-MDP model as introduced in Section II. Finally, DRL algorithms to solve the DRL model need to be developed and implemented.

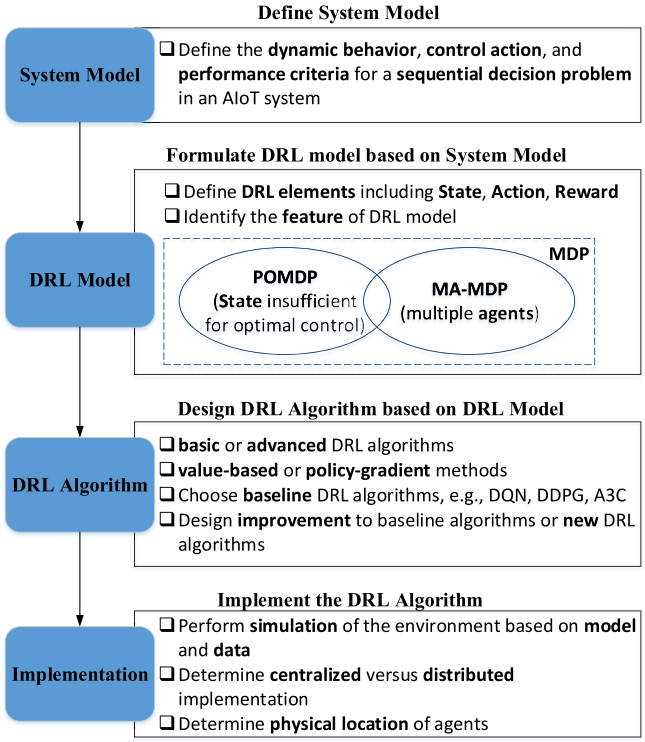


Fig. 9. General procedure for solving optimal control problems in AIoT by DRL theory.

Some typical examples of states, actions, and rewards in DRL models for AIoT systems in existing literature are summarized and classified according to the AIoT layers in Table III, Table IV, and Table V, respectively. In the rest of this section, we will review, compare and summarize the related research works from the following perspectives:

- review and compare **system models** considered for AIoT and identify promising system models for future research;
- review and compare **DRL models** according to the DRL elements and features. Some guidelines for the formulation of DRL models are provided with respect to different system models;

TABLE III
SUMMARY OF STATES IN RL/DRL MODEL FOR AIOT SYSTEMS

AIoT Layer		Examples	Details
Physical system state	Autonomous Robots	kinematic state	position, heading direction of the robot(s)
		manipulation state	angle of end-effector, opening status of the gripper, day-time or night-time mode, object grasping state
		surrounding environment	camera image and/or laser measurements of environment
	Smart Vehicles	driving environment	camera image of environment, relative position or distance to other vehicles, traffic signal state
		kinematic state	velocity and/or position of the agent vehicle, distance between multiple agent vehicles, state of the reservoir of UAV
	Smart Grid	battery SoC	amount of energy stored in the ESS
		time state	information on the time period relevant for the dynamics of the system, e.g. quarter of the day, day of the week and season of the year
		RE generation state	amount of renewable energy produced by PV panels, wind turbines, etc
		energy demand state	energy demand in smart grid by critical load
		price state	real-time electricity price
		DR device on/off state	on/off state of the DR device
Communication network state		channel state	SINR, pathloss, channel gain, data transmission rate, transmission successful indicator of wireless channel
		topology state	number of nodes in the network, locations of moving sensors in the field, whether an area is covered by a sensor
		sensor state	sleep, active, idle, process, TxRx, state estimation error
		queue state	number of tasks/packets/bits waiting to be transmitted or processed
		energy queue state	amount of available energy for tasks/packets/bits transmission or processing
		energy consumption state	the amount of energy consumed by the system
Computing system state		task state	remaining time to finish, waiting time, data size, CPU cycles, deadline, completion reward
		edge/fog/cloud server state	remaining computation resources; prices and CPU frequencies of different virtual machines (VMs) levels; number of VMs run in physical machines (PMs); whether a content is stored
		content state	popularity of requested content; number of requests for the content

- review and compare **DRL algorithms** used for AIoT with special attention on how to select different methods, e.g., value-based and policy gradient, according to the different characteristics of DRL models. As the baseline DRL algorithms such as DQN and DDPG are already introduced in Section II, we focus more on the new DRL algorithms that are different from the baseline algorithms or with proposed improvement over baseline algorithms;
- discuss the **implementation** considerations for the DRL algorithms, especially the physical location of the agent(s) to implement the algorithm and whether the centralized or distributed implementation is considered.

A. AIoT Network Layer-IoT Communication Networks

A reliable and efficient wireless communication network is an essential part of the IoT ecosystem. Such wireless networks range from short range local area networks such as Bluetooth, ZigBee/IEEE 802.15.4, and IEEE 802.11 to long range wide area networks such as Narrowband Internet of Things (NB-IoT) and LoRaWAN. When designing resource control mechanisms to efficiently utilize the scarce radio resources in transmitting the huge amount of IoT data, the IoT networks need to consider the characteristics of IoT devices such as massive in number, limited in energy, memory and computation resources. Moreover, the requirements of IoT applications such as low latency and high reliability have to be taken into account as well. One of the promising approaches to develop resource control mechanisms tailored for IoT is to enable IoT devices to operate autonomously in a dynamic environment by using learning frameworks such as DRL [59]. The existing works in this area mainly include studies on wireless sensor networks (WSNs), wireless sensor and actuator networks (WSAN), NB-IoT, and energy harvesting (EH). The system models of existing works are illustrated in Fig. 10. Table VI lists the related research works and their DRL models and DRL algorithms.

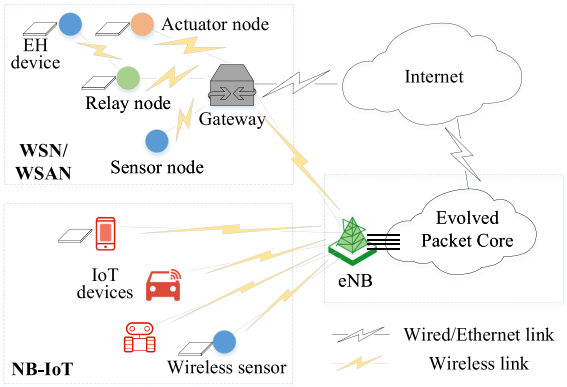


Fig. 10. IoT communication networks system models.

1) Wireless Sensor Networks: Wireless sensor network (WSN) is a wireless network of many tiny disposable low power sensors. WSNs are expected to be integrated into the

TABLE IV
SUMMARY OF ACTIONS IN RL/DRL MODEL FOR AIOT SYSTEMS

AIoT Layer		Examples	Details
Actuator control action	Autonomous Robots	kinematic control	turn left/right, go straight/back, rotating left/right, reach specific object, steering angle, velocity
		task manipulation	position of end-effector, change in position, change in azimuthal angle, gripper open and close, termination, sweep/pick up/put down objects
		charging	wander, recharge at home
	Smart Vehicles	velocity control	brake, accelerate, maintain (discrete) velocity (continuous)
		direction control	change lane (discrete); turn left, turn right, maintain (discrete) steering angle (continuous)
	Smart Grid	ESS management	amount of charging/discharging power of the ESS.
		DG energy dispatch	energy dispatch decision of DGs
		energy trading amount	amount of energy trading with main grid or other MGs
		energy trading price	the price to sell/buy.
		DR devices on/off change	whether to change the on/off state of the DR devices
	WSAN	actuator node movement	forward, back, left, right, and stop
		power mode control	turn of/off the sensor, select the high/low power mode of the sensor
Communication resource control action		communication resource allocation	bandwidth/subchannel allocation, energy allocation, IoT device scheduling, BS selection for offloading, route selection, relay selection, relay activation
Computation resource control action		offloading decision	offload or not (binary); the proportion of data to be offloaded (discrete or continuous); the number of offloaded bits (continuous).
		caching decision	whether to cache a content, which existing content to replace with
		computation resource allocation	number of CPUs cores (per second), VM level, edge/cloud server selection for offloading, serve or reject a request

TABLE V
SUMMARY OF REWARDS IN RL/DRL MODEL FOR AIOT SYSTEMS

AIoT Layer		Examples	Details
Physical system performance	Autonomous Robots	task completion	reach correct destination, correct operation (correct pushing/pulling, putting down/picking up of objects)
		task completion efficiency	backward penalty, cumulative distance traveled, overall completion time
	Smart Vehicles	driving safety	collision avoidance
		driving smoothness	keeping on the same driving lane, avoiding unnecessary velocity adjustment
		driving efficiency	driving speed, moving direction, junction waiting time, flow rate
		environmental benefits	vehicle fuel consumption, power consumption
	Smart Grid	energy balance	power balance within the MG, taking into account the charge/discharge of ESS
		DG generation cost	the cost of energy generation by DG
		ESS operation cost	losses for batteries based on charge/discharge operations
		energy trading cost/profit	the cost/profit caused by energy transaction
		load shedding cost	the cost to meet the requirement of controllable load
		consumer's satisfaction	level of consumer's satisfaction about the DR action
Communication network performance		reliability	decoding error probability, quality of the selected channel, packet loss rate, the expected correct bits per packet
		throughput	sum data rate of all the transmissions
		connectivity	number of connected nodes
		sensing coverage	covered area, number of sensing events, field estimation error, information gain
		energy consumption	energy consumption related to task/content transmission and task processing
		delay	task/content transmission delay and task processing delay
		task drop rate	probability that a task is dropped due to task queue being saturated
		load balance	efficient distribution of network or application traffic across multiple servers
Computing system performance		edge/cloud service cost	payment for purchasing edge/cloud service or PMs/VMs
		server utilization	utilization rate of edge/fog/cloud server or PMs/VMs
		content freshness	popularity of stored content

IoT systems, where sensor nodes join the Internet dynamically and use it to collaborate and accomplish their tasks.

A WSN mainly consists of three types of nodes, i.e., sensor, relay and gateway, which can be organized into three types of network topology as shown in Fig. 11. In the star topology, each sensor node connects to the gateway, which means the network is highly dependent on the central gateway. In the tree topology, the system is arranged in a top-down structure.

TABLE VI
SUMMARY OF DRL MODELS AND ALGORITHMS IN RESEARCH WORKS FOR IOT COMMUNICATION NETWORKS

Theme	Ref.	DRL Model				DRL Algorithm	Agent Location
		State	Action	Reward	Feature		
WSN	[60]	topology state	communication resource allocation	throughput, energy consumption	basic	DDQN	IoT device (centralized)
	[61]	topology state	power mode control	energy consumption, sensing coverage	MA	fully distributed Q-Learning, etc.	sensor controller (distributed)
	[62]	topology state	sensor node control	sensing coverage	basic	deep reinforced learning tree (DRLT)	sensor controller (centralized)
	[63]	channel state, energy consumption state	communication resource allocation	energy consumption, sensing coverage	basic	DQN	sensor controller (centralized)
	[64]	task state, channel state	communication resource allocation	throughput	basic	Deep Q-Learning	relay node (centralized)
WSAN	[65]	topology state	actuator node movement	sensing coverage	basic	DQN	actuator node (centralized)
	[66]	topology state	actuator node movement	reliability, delay	basic	Q-Learning	routing agent (centralized)
	[67]	sensor state, channel state	communication resource allocation	reliability	basic	DQN	network controller (centralized)
NB-IoT	[68]	channel state	communication resource allocation	reliabilty	MA, POMDP	CMA-DQN	NB-IoT devices (distributed)
	[69]	channel state	communication resource allocation	reliability	basic	upper confidence band (UCB)	NB-IoT device (centralized)
EH	[70]	task queue state, channel state, energy queue state	communication resource allocation	reliability, delay	basic	AMDP+OSL	fusion center (centralized)
	[71]	channel state, energy queue state	communication resource allocation	reliability	basic	LSTM-based DQN	BS (centralized)
	[72]	energy queue state	communication resource allocation	energy consumption	POMDP	DDQN	BS (centralized)
	[73]	energy consumption state, sensor state	communication resource allocation	reliability	basic	DDPG	sensor controller (centralized)

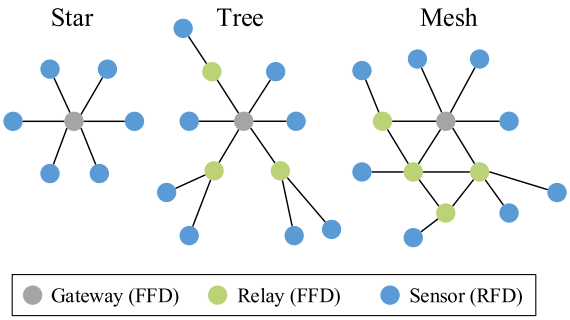


Fig. 11. Three types of network topology types for WSN.

When a parent node fails to work effectively, all its child nodes will be affected. In the mesh topology, the network nodes are interconnected with one another forming a mesh structure, where the data from sensor nodes can be forwarded by the relay nodes in a multi-hop fashion to the gateways.

In a mesh topology, network connectivity has an important effect on network performance such as throughput. Generally speaking, better connectivity can be achieved by activating more relay nodes in the network at the cost of larger energy consumption. Therefore, it is important to adjust the wireless transmission range of relay nodes to maximize the throughput while minimizing the energy consumption for the network. Reference [60] proposes an autonomous network formation solution that enables an IoT device to make the decision on whether to activate its transmission or not based on the multi-hop ad hoc network topology state. Distributed implementation is considered where each IoT device is an agent that applies the DDQN algorithm to make decisions based only on the local system state, i.e., the number of nodes in its transmission range.

Cooperative communication is recognized as an important technology to improve the WSNs' performance in data transmission rate and node coverage. A typical WSN system model with cooperative communication consists of a sensor node, multiple relay nodes, and a gateway, where the best relay node to forward data from sensor to gateway needs to be determined. In [63], the process of cooperative communication with relay selection is modeled as an MDP, where the channel state and energy consumption for signal broadcasting are used as state and an appropriate relay is selected to participate in the cooperative communication by a DQN type algorithm. The reward is determined by energy consumption and information gain.

The cognitive network is envisioned as one of the key enablers for the IoT, which can tackle the problem of the crowded spectrum for the rapidly increasing amount of IoT applications. A cognitive network consists of cognitive nodes that can sense the environment and make intelligent decisions to seize the opportunities to transmit. Reference [64] leverage the DRL technique to enhance the packet transmission efficiency in cognitive IoT. The system model considers one relay

node which gathers packets from multiple sensor nodes and sends to the gateway via M channels. According to the queue state and channel state, the relay makes channel selection decisions for packet transmission. The goal of the policy is to maximize the throughput and minimize the energy consumption, and a deep Q-Learning based algorithm is proposed to solve the problem.

One important application of the sensor nodes is to properly cover an area in order to make sure that all important events which occur in that area can be accurately detected by at least one sensor. This is referred to as the sensor coverage problem. An MA system approach on WSNs is able to tackle the resource constraints in these networks by efficiently coordinating the activities among the nodes. Reference [61] tackles the coordinated sensing coverage problem in an MA system, where each sensor acts as an agent. The state obtained by the sensor is the binary sensing status of its area. There are three discrete actions in the action space: turn off the sensor, turn on the sensor in low power mode and turn on the sensor in high power mode. The reward function for each agent is based on both the gain of sensing coverage and energy consumption resulting from the action. The authors study the behavior and performance of four distributed DRL algorithms, i.e., fully distributed Q-Learning, distributed value function (DVF), optimistic DRL, and frequency maximum Q-Learning (FMQ).

Reference [62] also studies the sensing coverage problem in WSNs with a focus on mobile robotic sensor nodes. The position of the moving sensor is taken as the state and an action corresponds to the movement of the mobile sensor. Higher information gain according to the location of the sensors can lead to higher rewards in the DRL model. To accelerate the exploration process and find near-optimal sampling locations for the mobile sensors, deep reinforced exploring learning tree (DRLT) is designed and outperforms other field exploration algorithms, such as rapidly exploring random tree (RRT) and rapidly exploring tree with linear reduction (RRLR).

2) Wireless Sensor and Actuator Networks: Wireless sensor and actuator networks (WSANs) are composed of a large number of sensor nodes with low power, one or more actuators, and a processing unit. They are envisioned as an important part of the IoT ecosystem and have a panoply of applications, ranging from industrial automation to homeland security. Unlike conventional WSNs, sensor and actuator nodes must work together closely to collect and forward data, and act on any sensed data collaboratively, promptly and reliably to react to the physical world.

Scheduling transmissions is one of the challenges in WSAN from a networking perspective because of the volatile nature of wireless channels. Wireless transmission is scheduled from sensors to the gateway and from the gateway to the actuators over a shared medium. To solve this problem, [67] formulates a DRL-based sensor scheduling problem for allocating wireless channels to sensors for remote state estimation of dynamical systems. The problem is formalized as an MDP and solved by the DQN algorithm. In this work, the network controller allocates communication resources based on the sensor state and channel state. The reward function is calculated by the reliability.

WSANs, e.g., ISA SP100.11a and WirelessHART, have special devices known as network managers that perform tasks such as admission control of devices, the definition of routes, and allocation of communication resources. The state-of-art routing algorithms used in these protocols usually have different weights for different route preferences. Weight adjustment can be challenging because of the dynamicity of wireless networks. RL/DRL models can be used for weight adjustment with a consideration of current application requirements and communication conditions. In [66], a global routing agent with Q-Learning is proposed for weight adjustment of the state-of-the-art routing algorithm, aiming at achieving a balance between the overall delay and the lifetime of the network. The routing agent receives network topology state information including the weight of the number of hops and the weight of the energy source. It makes decisions on whether to change into a neighbor state or to keep the current state. The reward is determined by the expected network lifetime and average network latency.

Similar to the mobile sensor movement control in WSNs [62], automatic control of node mobility is also essential in WSANs. Several performance metrics, such as connectivity, coverage, energy consumption and accuracy, can be improved by moving the nodes in the networks. Reference [65] focuses on the connectivity performance which is essential to conducting collaborative tasks among the actuator nodes. Specifically, the authors present the design and implementation of a simulation system based on DQN for mobile actor node control in a WSAN. The actuator node takes the network topology state as the input state and makes the decision on mobile actor node movement from five discrete patterns, including stop, forward, back, left and right. The reward is measured by sensing coverage for each action.

3) NB-IoT: NB-IoT is a technology proposed by 3GPP in Release-13. It offers low energy consumption and extensive coverage to meet the requirements of a variety of social, industrial and environmental IoT applications. Compared to legacy LTE technologies, NB-IoT chooses to increase the number of repetitions of transmission to serve users in deep coverage. However, large repetitions can reduce system throughput and increase the energy consumption of IoT devices, which can shorten their battery life and increase their maintenance costs.

Radio resource allocation in NB-IoT specifies the number of radio resources allocated to each group of devices in a Transmission Time Interval (TTI). NB-IoT includes two types of uplink channels, namely, Narrowband Physical Random Access Channel (NPRACH) and Narrowband Physical Uplink Shared CHannel (NPUSCH). At the beginning of each uplink TTI, the evolved Node B (eNB) selects a configuration that specifies the radio resource allocation in order to accommodate the NPRACH procedure with the remaining resources used for data transmission. However, it is a challenge to balance the channel resource allocation between the NPRACH procedure and data transmission. To solve this uplink resource configuration problem, a Cooperative MA DQN (CMA-DQN)

approach is developed in [68], in which each DQN agent independently controls a configuration variable for each group, in order to maximize the long-term average number of working IoT devices in NB-IoT. As the eNB can only observe the channel state at the end of each TTI, the problem is formalized as POMDP and historical information is used for current state prediction. The state is represented by channel state information of the last M TTIs. Each agent receives the same common reward at the end of the current TTI. The common reward can ensure that all the agents are aiming at maximizing the number of devices that transmit data successfully in NB-IoT. Multiple agents are trained in parallel in CMA-DQN and the weight matrix is updated by using DDQN.

Enhancing the coverage and reducing energy consumption are the key targets for NB-IoT. The major state-of-art solutions are repeating transmission data and control signals, which lead to system throughput reduction as well as spectral efficiency loss. In [69], the authors propose a new method based on the RL algorithm to enhance NB-IoT coverage. Instead of employing a random spectrum access procedure, dynamic spectrum access can reduce the number of required repetitions, increase the coverage, and reduce energy consumption. The agent receives two values 0 or 1 as the reward, which means that the selected channel is occupied or vacant.

4) Energy Harvesting: Energy Harvesting (EH) is a promising technology for the long-term and self-sustainable operation of the IoT devices. While EH is a promising technique to extend the lifetime of IoT devices, it also brings new challenges to resource control due to the stochastic nature of the harvested energy.

The uncertainty of the harvested energy poses challenges to the reliability of EH systems, which is essential for a number of industrial applications. In [70], the energy management policy in an industrial WSN is investigated to minimize the weighted packet loss rate under the delay constraint, where the packet loss rate considers the lost packets both during the sensing and transmission processes. A centralized fusion center (FC) takes task queue state, channel state and energy queue state as the input state. At each time slot, a sensor scheduling action and a transmission energy allocation action are chosen from the discrete action space. The problem is formulated into an MDP model, and stochastic online learning is applied to derive a distributed energy allocation algorithm with a water-filling structure and a scheduling algorithm by an auction mechanism.

One way to deal with the uncertainty of harvested energy is through battery level prediction in EH-based systems. Reference [71] considers an uplink transmission scenario with multiple EH user equipment (UEs) and a BS with limited access channels. The authors model the access control based on battery prediction as an MDP. The channel state and energy queue state are employed as the input state to the BS, which then outputs the action according to the scheduling policy. As the performance of the model relies on both the battery prediction result and the access control policy, the reward takes the sum rate of the transmissions into account. A two-layer LSTM-based DQN control network is proposed to solve the problem. The first layer is an LSTM-based network to perform the battery prediction. The second layer takes the battery prediction result, channel state and energy queue state as the input and outputs the action for producing the access control policy.

The uncertainty of harvested energy can also be captured by formulating a POMDP problem for the EH-based systems. In [72], BS is considered as an agent to schedule the IoT devices after receiving the energy queue states of some of the nodes. The amount of energy consumption is considered as the reward. DDQN algorithm is adopted to solve this POMDP-based problem.

While most algorithms for EH are value-based methods, [73] proposes an algorithm based on DDPG which can tackle the energy management problem in a continuous space. The sensor controller receives the energy consumption state and bit error rate of the sensor and determines the transmission energy allocation for the sensor. The reward is measured by the net bit rate.

5) Comparison and Insights: By summarizing and comparing the above literature, the following insights can be obtained.

- *System model:* Most of the research work focus on star topology since the agent control in a single-hop network is relatively simple. On the other hand, DRL-based solutions for mesh topology including cellular networks with D2D communications can be studied more.
- *DRL model:* In terms of the system state, the channel state is usually included due to the time-varying nature of the wireless channel. Examples of typical channel state include SINR, pathloss, channel gain, data transmission rate as given in Table III. Another common system state is the transmission queue state, especially when the data packets arrive according to a dynamic process. When EH is considered, the energy queue state is normally an important component of the system state. In routing problems for mesh networks, the topology state usually has overall information about the nodes in the network which can be helpful for the routing agent [66]. In terms of action, most research works focus on communication resource allocation. Some literature in WSAN takes actuator control as action. Typical reward functions include throughput, reliability, energy consumption, sensing coverage.
- *DRL algorithm:* DQN and novel DQN-based DRL algorithms are most frequently adopted in existing literature as can be observed from Table VI. This is partly due to the fact that the action space in most existing works is discrete. As discussed in Section II, value-based methods are simpler than actor-critic methods and easier to converge. However, value-based methods cannot be applied to continuous action space unless the continuous actions are discretized, which results in loss of performance and curse-of-dimensionality problem.
- *Implementation:* As shown in Table VI, most DRL algorithms are centrally implemented at the BS, gateway, etc., while a few MA-based DRL algorithms are distributively implemented at sensors or IoT devices. As sensors and

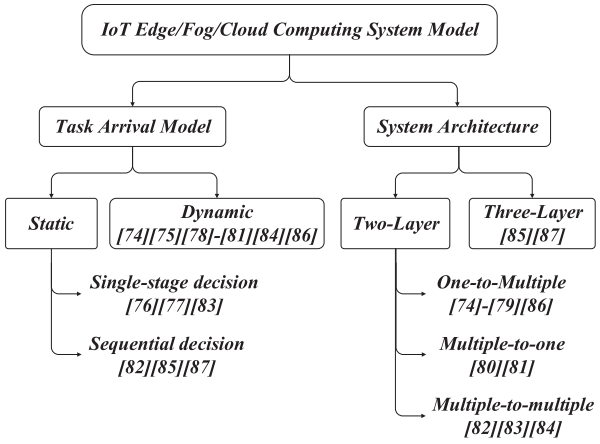


Fig. 12. Classification of IoT edge/fog/cloud computing system models.

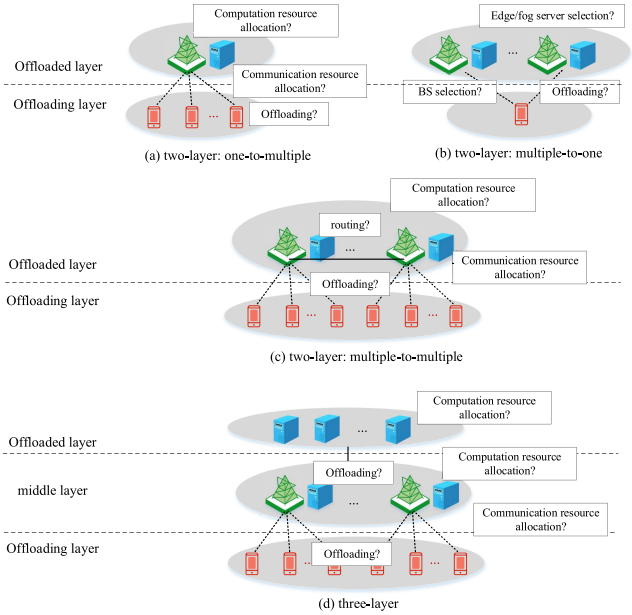


Fig. 13. IoT edge/fog/cloud computing system architecture.

IoT devices normally have limited energy, computation, and storage resources, the algorithm design needs to consider the trade-off between performance and complexity. The energy spent on communication and computation by the five agents on Crossbow Mica2 motes during the learning phase (first 20,000 iterations) is compared between different DRL algorithms in [61]. It is shown that although DRL algorithms based on the global system state can achieve the best performance, the incurred communication and computation overhead is too large. It is suggested that performing DRL algorithms based on local system state in resource constraint IoT devices is a more viable option.

B. AIoT Application Layer-IoT Edge/Fog/Cloud Computing Systems

Edge/fog/cloud computing is a crucial technique to process and analyze the huge amount of sensory data in IoT. In such systems, IoT devices can offload the computationally intensive tasks to the edge/fog/cloud servers. Moreover, caching IoT data at the network edge is considered to be able to alleviate the congestion and delay in transmitting IoT data through wireless networks. The above problems have been widely studied by applying DRL techniques. We classify the existing research based on the different considerations in system models as shown in Fig. 12. Table VII lists the related research works and their respective DRL models and algorithms.

1) Task Offloading and Resource Allocation: Reasonable decisions are required to be made on whether to offload the computation tasks to the edge/fog/cloud servers or perform them locally at the IoT devices. Moreover, a proper amount of communication and computation resources need to be allocated for the transmission and processing of each task. In the following, we will identify several important system model considerations that have an important impact on the formulation of the corresponding DRL models.

a) System architecture: The system architectures considered in existing works can be classified into two-layer and three-layer as illustrated in Fig. 13. A two-layer architecture has an offloading layer and an offloaded layer. Some examples of offloading vs. offloaded layers are IoT device layer vs. edge/fog/cloud layer and edge/fog layer vs. cloud layer. Fig. 13 uses the IoT device layer vs. edge/fog layer as an example. On the other hand, a three-layer architecture has a middle layer (i.e., edge/fog layer) in addition to the offloading layer (i.e., IoT device layer) and offloaded layer (i.e., cloud layer). An example of three-layer architecture is considered in [85], where a DRL-based computing offloading approach is proposed to learn the optimal offloading policy in a space-air-ground integrated network (SAGIN). The flying unmanned aerial vehicles (UAVs) and satellites provide access to edge computing and cloud computing, respectively. The offloading decisions need to determine whether the IoT devices should offload data to the edge server, and also whether the edge server should further offload data to the cloud.

The two-layer architecture can be further classified according to the number of nodes in the offloaded and offloading layers, namely, one-to-multiple, multiple-to-one, and multiple-to-multiple as shown in Fig. 13. In one-to-multiple architecture, resource allocation is usually determined jointly with the offloading decisions for each node in the offloading layer (e.g., IoT device). A joint task offloading decision and bandwidth allocation optimization method based on DQN is designed for the edge computing system in [76]. The overall offloading cost is evaluated in terms of energy cost, computation cost, and delay cost. The authors in [77] consider wireless powered edge computing system that optimally adapts offloading decisions and communication resource allocations to the time-varying wireless channels. Only the offloading decisions are determined by DRL algorithms, while the wireless resource allocations are derived by convex optimization. Some works focus only on offloading decisions. In [75], the offloading decision is determined by each IoT device as an agent based on the channel state, task queue state, and its remaining computation resources. A ϵ-greedy Q-Learning algorithm is adopted to optimize the delay and energy consumption. The computing

TABLE VII
SUMMARY OF DRL MODELS AND ALGORITHMS IN RESEARCH WORKS FOR IOT EDGE/FOG/CLOUD COMPUTING SYSTEMS

Theme	Ref.	DRL Model				DRL Algorithm	Agent Location
		State	Action	Reward	Environment		
Task Offloading and Resource Allocation	[74]	task state, queue state	offloading decision (discrete)	delay	application layer	Deep Q-Learning	edge server (centralized)
	[75]	channel state, task queue state, edge server state	offloading decision (discrete)	delay, energy consumption	application layer	Deep Q-Learning	IoT devices (distributed)
	[76]	offloading decision, communication resource allocation	movement among two neighboring states (discrete)	delay, energy consumption	application layer, communication layer	DQN	edge server (centralized)
	[77]	channel state	offloading decision (discrete)	delay	application layer	deep actor network with supervised learning	edge server (centralized)
	[78]	queue state, task state	offloading decision, communication resource allocation (discrete)	delay, energy consumption	application layer, communication layer	value-based DRL with value function decomposition	edge server, IoT device (semi-distributed)
	[79]	channel state, queue state, topology state, task state, edge server state	offloading decision, computation resource allocation (discrete)	delay, energy consumption, cloud service cost	application layer	actor-critic with ACA for exploration	edge server, centralized
	[80]	channel state, queue state, energy queue state	offloading decision, computation resource allocation, energy allocation (discrete)	delay, task drop rate, edge service cost	application layer	DDQN with Q-function decomposition	centralized network controller (centralized)
	[81]	channel state, energy queue state	offloading decision (discrete)	delay, energy consumption	application layer	DQN	edge server, centralized
	[82]	edge server state, task state, network state	offloading decision, communication resource allocation, computation resource allocation (discrete),	delay, energy consumption	application layer, communication layer, application layer	MCTS+MLT	centralized network controller (centralized)
	[83]	computation resource allocation	adjustment of states (discrete)	delay, load balance	application layer, communication layer	DQN	SDN controller (centralized)
	[84]	channel state, queue state, energy queue state	offloading decision, communication resource allocation (discrete),	delay, energy consumption	application layer, communication layer	DDQN+FL	edge servers (distributed)
	[85]	task state, channel state	offloading decision (discrete)	delay, energy consumption, server utilization	application layer	deep actor-critic	cloud server (centralized)
Caching	[86]	content state	caching decision, discrete	delay	application layer	deep actor-critic	edge server (centralized)
	[87]	task state, content state, channel state, edge router state	offloading decision, caching decision, communication resource allocation, computation resource allocation (discrete)	delay, content freshness	application layer, communication layer	natural actor-critic	edge router (centralized)

offloading problem in an LTE-U-enabled network is considered in [74], which determines whether the task on an IoT device is carried out locally or is offloaded to the LTE-U BS based on queue state and task priority. A Deep Q-Learning algorithm is adopted to solve the problem. A blockchain-empowered edge computing system is considered in [79], where the offloading decisions are made for both mining tasks as well as data processing tasks.

In multiple-to-one architecture, the main objectives are usually to determine for a representative IoT device whether and to which edge/cloud servers to offload its tasks. In [81], a DQN-based offloading scheme is proposed to select the edge server and proportion of data to be offloaded for an IoT device with EH. On the other hand, [80] considers a representative mobile user served by multiple BSs connected to a single edge server in an ultra-dense RAN. The problem of selecting the proper BS via which to offload data from mobile users to the edge server is tackled by a DQN-type algorithm.

In multiple-to-multiple architecture, routing and load balance between different nodes in the offloaded layer need to be considered. A collaborative edge computing system is considered in [82], where multiple edge computing servers collaboratively perform distributed computing. Each edge computing server serves multiple mobile devices (MDs), and when it received an offloaded task form an MD, it can choose to further offload it to other collaborative edge computing servers. The state of the DRL model includes network state and task characteristics. The action involves determining MD offloading

rate, edge computing server offloading rate, communication resource allocation, and computation resource allocation. The reward is designed to optimize the performance over delay and power consumption. Instead of using baseline DRL algorithms, a DNN is trained to predict the resource allocation action in a self-supervised learning manner, where the training data is generated from the searching process of Monte Carlo tree search (MCTS) algorithm. Moreover, Multitask Learning (MTL)-Based Action Prediction method is adopted, which improves the traditional DNN by splitting the last layers of DNN to construct a sub NN for supporting higher action dimensions. A similar system model is considered in [83], where a DQN-based scheme is proposed to allocate computation and communication resources for an edge computing system with multiple edge servers and mobile users, in order to reduce the delay and achieve load balance. The authors in [84] study an edge computing system where there are multiple IoT devices with EH capabilities. Each IoT device determines where to perform a task, i.e., whether and which edge computing server to offload; and how many energy resources should be allocated based on the channel state, task queue state, and energy queue state. In order to reduce the transmission costs between the IoT devices and edge nodes, federated learning (FL) is used to train DRL agents in a distributed fashion.

b) Task arrival model: The computation task arrival models considered in the existing literature can be divided into static versus dynamic. The static task arrival models consider that the number of tasks in the system is a fixed value, while the more practical dynamic task arrival models [74], [75], [78]–[81], [84] consider that tasks arrive according to a stochastic/deterministic process and are buffered in a queue if cannot be processed immediately upon arrival. The static task arrival model can be further divided into two types. In single-stage decision static models [76], [77], [83], single-stage decisions for all the tasks are determined in one shot simultaneously. In sequential decision static model [82], [85], offloading and resource allocation decisions for the fixed number of tasks in the system are determined sequentially until all the tasks are processed or all the resources are occupied.

For dynamic and sequential decision static models, the control decisions for a particular task need to consider their impacts on the future tasks in terms of the long-term average performance of the system. They belong to the sequential decision problems, where DRL techniques provide a powerful tool in dealing with them. On the other hand, the single-stage optimization problems in a single-stage static model are usually formulated as the mixed linear programming problems, where DRL is considered as a better tool than the heuristic algorithms.

c) Centralized vs. distributed implementation: Most of the above DRL algorithms are considered to be centrally implemented in the edge/fog/cloud server or central control unit. This means that the IoT devices need to report their local states to the central control unit so that the latter can perceive the global system state. As the state space of the DRL model grows exponentially with the number of IoT devices, the computation complexity and communication overhead can be overwhelming when the number of IoT devices are large.

From this perspective, it seems that a distributed DRL algorithm where each IoT device makes independent decisions based on its local system state is a promising solution as in [75]. However, the mutual exclusion nature in multi-user resource allocation makes it hard to design a fully distributed solution in general. Moreover, as DRL algorithms normally take tens of thousands or even millions of time steps to train, the computation complexity and energy consumption are likely to forbid their implementation on the resource-constrained IoT devices.

In contrast to a fully centralized or distributed DRL algorithm, a semi-distributed implementation where the edge server and IoT devices cooperate to determine the optimal action is proposed in [78]. A multi-user edge computing system with NB-IoT wireless network is considered, where the offloading decision and user scheduling are optimized to minimize delay and energy consumption. A value function approximation architecture as illustrated in Fig. 14 for DRL algorithm is proposed. The global system state is first decomposed into local system states, which are used as input to the NN to approximate the value functions. The output to the NN are value functions for the global system states, which are derived from the NN as the sum of per-node value functions of local system states. Specifically, the value function for the *i*-th global system state $\mathbf{s}^{(i)}$ can be derived as

$$V\left(\tilde{\mathbf{s}}^{(i)}\right) \cong \sum_{n=1}^{N} \sum_{j=1}^{D} \phi_{\tilde{s}_n^{(j)}}\left(\tilde{\mathbf{s}}^{(i)}\right) V_n\left(\tilde{s}_n^{(j)}\right), \tag{23}$$

where D is the cardinality of the local system space of any device $n \in \{1, 2, \ldots, N\}$, and $V_n(\tilde{s}_n^{(j)})$ is the per-node value function of IoT device n for its local system state $\tilde{s}_n^{(j)}$. $\{\phi_{\tilde{s}_n^{(j)}}(\tilde{\mathbf{s}}^{(i)})\}_{n=1}^{N}$ is the feature vector of the global system state $\tilde{\mathbf{s}}^{(i)}$.

The above proposed NN architecture can facilitate semi-distributed auction-based implementation as shown in Fig. 15. Specifically, each IoT device maintains its per-node value functions, based on which it can distributively calculate and submit bids to the BS and edge server. The BS centrally determines the optimal action based on the bids submitted by all the IoT devices. In this way, IoT devices help to alleviate the computational and storage burdens from the BS, while BS makes control decisions to control the scarce spectrum resources in the license band.

2) Caching: Caching decision mainly involves content placement, such as whether to cache content at the edge server, and which existing content should be replaced when new content is stored as listed in Table table_edge2. The research in [86] solves the problem of caching IoT data at the edge with the help of DRL. The proposed data caching policy aims to strike a balance between the communication cost and the loss of data freshness. In [87], the issue of caching strategy is tackled together with the offloading policy and resource allocation. The natural actor-critic algorithm is adopted for this purpose.

3) Comparison and Insights: By summarizing and comparing the above literature, the following insights can be obtained.

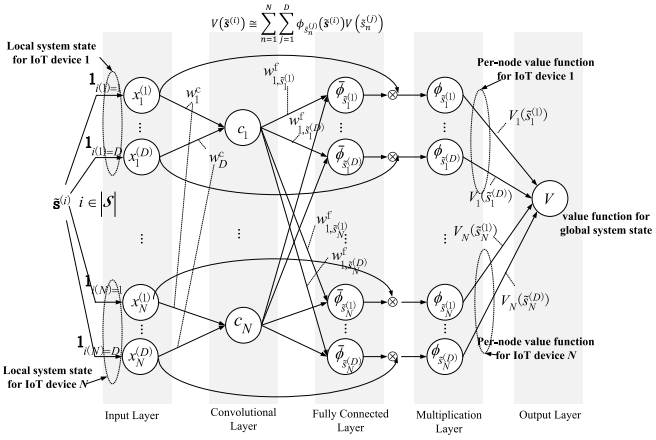


Fig. 14. A value function approximation architecture of DRL Algorithm for IoT edge computing that facilitates semi-distributed implementation.

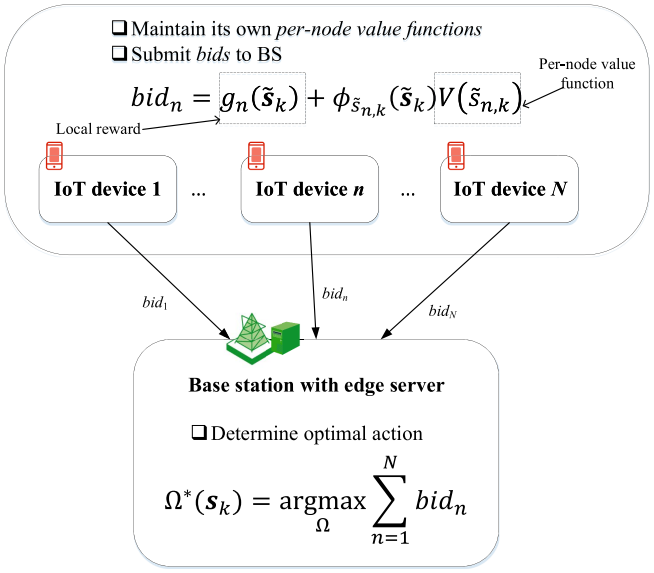


Fig. 15. A semi-distributed implementation of DRL algorithm for IoT edge computing.

- *System model:* Most of the research works focus on two-layer one-to-multiple and two-layer multiple-to-one system architectures. On the other hand, only a few literatures are devoted to two-layer multiple-to-multiple or three-layer architectures due to their complexities. Therefore, it is worthwhile to devote more attention to these system architectures in the future.
- *DRL model:* A general rule of thumb when defining the system state is that if the wireless channel fading is considered, the channel state usually needs to be included in the system state. On the other hand, if the dynamic task arrival model is considered, the queue state is usually necessary. Also note that when DRL is used to solve mixed integer programming problems under single-stage static task arrival models [76], [77], [83], the system states are usually defined as the control decisions to be optimized (e.g., offloading decision, resource allocation), which are usually formulated as actions in DRL models for dynamic and sequential-decision static task arrival models. It can be observed from Table VII that the mostly considered rewards are delay and energy consumption.
- *DRL algorithm:* DQN, deep Q-Learning or similar value-based DRL methods are adopted most frequently as can be observed from Table VII. Actor-critic algorithms are adopted for those few works with continuous action space. Moreover, some works with discrete action spaces also adopt the actor-critic algorithms, as they promise better performance than their value-based counterparts. It should also be pointed out that most existing works usually directly apply the well-known DRL algorithms such as DQN and DDPG. However, there are also some research works that propose to improve the existing DRL algorithms considering the specific characteristics of studied problems. The authors in [79] use the adaptive genetic algorithm (ACA) for exploration. Moreover, recent works in [78], [80] designed the Q or value function approximation architectures according to the structures of the specific problems. More research efforts can be devoted to this aspect in future works.
- *Implementation:* Most DRL algorithms are considered to be centrally implemented in the edge/fog/cloud server or central control unit, where the computation complexity and communication overhead increase with the increasing number of IoT devices. On the other hand, fully distributed implementation of DRL algorithms on the resource-constrained IoT devices may result in large processing delay and energy consumption. It is interesting to explore efficient collaboration between edge/fog/cloud server and IoT devices to achieve the most efficient implementation of the DRL algorithms.

C. AIoT Perception Layer-Autonomous Robots

The applications of DRL methods in autonomous robots have been widely discussed. Researches include mobile behavior control of robots, robotic manipulation, management in multi-robot systems, and cloud robotics issues. Table VIII lists the related research works and their respective DRL models and DRL algorithms.

1) Mobile Behavior Control: The mobile behavior control mainly refers to the path planning, navigation, and general movement control of robots. DRL approaches have been applied in many existing works for this purpose. In DRL models for mobile behavior control of robots, the actions mainly focus on the kinematic control of the autonomous robots, i.e., defining the moving velocity and direction in discrete or continuous form. The states are related to the surrounding environment and the kinematic state of robots. The surrounding environment can be obtained by sensors on the robot, for example, taking images of environment by camera, or measuring distance to obstacles by lasers. As for the reward, collision avoidance in robot movement is often considered, as collisions may cause damage of the robot and failure in the task of arriving the destination. Moreover, in some studies, the efficiency and effectiveness of robot movement are also taken into account. For example, in [88], a penalty is given to the robot when it moves backward. The authors apply DQN to the

TABLE VIII
SUMMARY OF DRL MODELS AND ALGORITHMS IN RESEARCH WORKS FOR IOT AUTONOMOUS ROBOTS

Theme	Ref.	DRL Model					DRL Algorithm	Agent Location
		State	Action	Reward	Environment	Feature		
Mobile Behavior Control	[88]	surrounding environment	kinematic control	collision avoidance, task completion efficiency	perception layer	basic	DQN	robot (centralized)
	[89], [90]	surrounding environment	kinematic control	task completion	perception layer	basic	DQN	robot (centralized)
	[91]	kinematic state	kinematic control	collision avoidance	perception layer	basic	DDPG	robot (centralized)
	[92]	surrounding environment, kinematic state	kinematic control	collision avoidance, task completion efficiency	perception layer	basic	A3C	robot (centralized)
Robotic Manipulation	[93]	surrounding environment	kinematic control	collision avoidance, task completion efficiency	perception layer	basic	DDPG	robot (centralized)
	[94]	manipulation state, kinematic state	task manipulation action	task completion	perception layer	basic	DDPG	robot (centralized)
	[95]	surrounding environment, manipulation state	task manipulation action	task completion	perception layer	basic	deep Q-Learning	robot (centralized)
	[96]	surrounding environment	task manipulation action	task completion	perception layer	basic	deep P-network	robot (centralized)
Multi-Robot System	[97]	surrounding environment	kinematic control	task completion	perception layer	basic	DQN	robot (centralized)
	[98]	task completion state	task manipulation action	task completion efficiency	perception layer	MA	deep Q-Learning	robot (distributed)
	[99]	surrounding environment	task manipulation action	task completion efficiency, task completion	perception layer	MA	actor-critic	robot (distributed)
	[100]	surrounding environment, kinematic state, targeted positions	kinematic control	collision avoidance, task completion efficiency	perception layer	POMDP, MA	policy gradient	robot (distributed)
	[101]	kinematic state, manipulation state	task manipulation action, charging action	task completion	perception layer	MA	deep Q-Learning	robot (distributed)
Cloud Robotics	[102]	content state	kinematic control, resource allocation	collision avoidance, task completion	perception layer, application layer	basic	DQN	cloud server (centralized)
	[103]	server state, task state	offloading decision, resource allocation	server utilization	perception layer, application layer	basic	Q-Learning	cloud server (centralized)

robot behavior learning simulation environment, so that mobile robots can learn to obtain good mobile behavior by using high-dimensional visual information as input data. Moreover, the authors incorporate profit sharing methods into DQN to speed up learning, and the method reuses the best target network in the case of a sudden drop in learning performance. In [92], the cumulative distance traveled is involved when defining the reward in the DRL model of solving a mobile robot navigation problem. A hybrid A3C method is applied with the aid of convolution neural network (CNN) and LSTM. Mobile robot path planning is a common problem in mobile behavior control of autonomous robot. DQN is designed in [89] and DDPG is applied in [91] to solve the path planning issues. In these cases, whether the robot reaches targeted destinations is an essential concern when defining the reward function. The study in [90] combines Q-Learning with CNN for IoT enabled mobile robot with an arm which can reach the destination autonomously and perform suitable actions.

2) Robotic Manipulation: Since intelligent robots usually help to perform some operation tasks in practice, appropriate controlling schemes for them are necessary for successful manipulations. DRL theories are widely applied in solving the robotic manipulation problems. For example, the problem of controlling robots to accomplish compound tasks

is solved by a hierarchical DRL algorithm in [93]. In [94], the authors demonstrate that the DRL algorithm based on off-policy training of deep Q-functions can be applied to complex three-dimensional (3D) operation tasks, and can effectively learn DNN strategies to train real physical robots. The policy updates are pooled asynchronously to decrease the training time. Similarly, the problem of learning vision-based dynamic manipulation skills is solved by using a scalable DQN approach in [95].

In the studies related to robotic manipulation control by DRL theories, the manipulation state is usually involved in the state of the DRL model, in addition to the kinematic state and the surrounding environment. The specific form of the manipulation state is determined by the type of the task for the robot. For example, the current status of the gripper on the robot, i.e., whether it is open and the height it has reached are a part of action in [95]. The action in the model is also related to the specific manipulation task. For example, in [96], the action is defined as picking up the handkerchief in a real robotic cloth manipulation task. As for the reward in robotic manipulation problems, the success of target achievement is a necessary criteria.

3) Multi-Robot System: In some cases, multiple robots are required to collaborate properly to fulfil some tasks that are difficult to be accomplished by an individual robot. A review on MA RL in multi-robot systems is provided in [104]. In multi-robot system, the action is related to the kinematic control and manipulation control of multiple robots. The research in [97] investigates a DRL approach to the collective behavior acquisition of swarm robotics systems. The multiple robots are expected to collect information in parallel and share their experience for accelerating the learning. In [101], the charging action is involved, where the robot is supposed to recharge in time. The task completion efficiency is an important evaluation criteria of effectiveness of the cooperation of the multiple robots. Thus, in multi-robot system, the task completion efficiency usually contributes to the reward in the DRL model. For autonomous robots, the task completion efficiency can be evaluated by overall completion time of tasks [98].

Distributed control schemes are introduced in some existing works. In [98], the authors propose a collaborative multi-robot RL method, which realizes task learning and the emergence of heterogeneous roles under a unified framework. The method interleaves online execution and relearning to accommodate environmental uncertainty and improve performance. The multi-robot framework in Fig. 16 provides an architecture for robots to collaborate in a joint task space with environmental uncertainties towards maximizing global team utility, where the components follow the environmental sensation, neural perception, decision, execution, reward feedback cycle. The study in [99] extends the A3C algorithm in single agent problems to a multi-robot scenario, where the robots work together toward a common goal. The policy and critic learning are centralized, while the policy execution is decentralized. A distributed sensor-level collision avoidance policy for multi-robot systems is proposed in [100]. A multi-scenario multi-stage training framework based on policy gradient methods is used

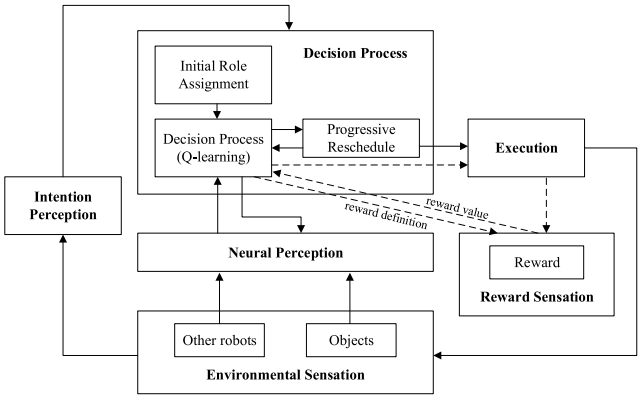


Fig. 16. An MA framework for multiple robots to collaborate in a joint task space.

to learn the optimal policy for a large number of robots in a rich, complex environment.

4) Cloud Robotics: The concept of cloud robotics allows the robotic system to offload computing-intensive tasks from the robots to the cloud [105]. Generally, cloud robotics applications include perception and computer vision applications, navigation, grasping or manipulation, manufacturing or service robotics, etc. Similar to discussions on the AIoT edge/fog/cloud computing system, the cloud robotics issues involve solving the offloading decision and resource allocation problems via DRL algorithms. For example, the authors in [103] propose an RL-based resource allocation scheme, which can help the cloud to decide whether a request should be accepted and how many resources are supposed to be allocated. The scheme realizes an autonomous management of computing resources through online learning, reduces human participation in scheme planning, and improves the overall utility.

Moreover, the resource allocation and kinematic control issues can be considered together in cloud robotics. For example, in [102], an effective transfer learning scheme based on lifelong federated RL (LFRL) is proposed for the navigation in cloud robotic systems, where the robots can effectively use prior knowledge and quickly adapt to new environments of the system in the long run.

5) Comparison and Insights: By summarizing and comparing across the above literature, the following insights can be obtained.

- *System model:* The autonomous robot problems include models with single robot or multiple robots. The multi-robot problem has been widely studied, as it is a typical issue in MA RL.
- *DRL model:* The state in DRL models for autonomous robot usually involves perceiving the environment of robots. The camera image is mostly adopted to represent the surrounding environment. No matter in a single robot or a system of multiple robots, if the problem involves the motion of the robot, then the kinematic state is generally a part of the state. When robot manipulation is studied, the manipulation state is an important part of the state. When intelligent robots are combined with cloud

computing techniques, that is, when the cloud robotics problem is studied, the state in the DRL model includes the server computing resource state and the task execution state at cloud. The action in the DRL model for autonomous robot is mainly related to what the robot does. For example, when the main job of a robot is to move, the action in the model is related to kinematic control. For problems related to robot manipulation, the task manipulation action is considered in the DRL model. For example, when a robot wants to use its mechanical arm to pick up or drop down some objects, the possible angle, height, and switch state of the arm constitute the action set. In cloud robotics, the actions in the model may involve offloading decision and resource allocation. As for the reward in the DRL model, it is often related to whether the autonomous robot can successfully complete the task.

- *DRL algorithm:* DQN and DDPG algorithms are widely applied in autonomous robots. For continuous action space, DDPG is mostly adopted [91], [93]. In the research of autonomous robots, the multi-robot problem involves MA DRL theory. In this case, MA DRL algorithms are adopted. For example, in [100], a policy gradient based RL algorithm is applied among multiple robots. In the system, each robot receives its own observation at each time step and executes the action generated from the shared policy. The policy is trained with experiences collected by all robots simultaneously, which allows the multiple robots to cooperate well.
- *Implementation:* For cloud robotics, DRL models are implemented on the cloud servers. The servers make decisions for the task execution in the robot system. On the other hand, DRL algorithms can also be implemented on the robots, which can be considered as a type of powerful IoT devices. For example, the implementation of the algorithm in [100] involves a large-scale robot group with laser scanners. The multi-robot collision avoidance policy among them is trained on each robot with a computer of an i7-7700 CPU and a Nvidia GTX 1080 GPU.

D. AIoT Perception Layer-Smart Vehicles

The development of IoT technology has promoted the development of intelligent transportation systems (ITS). In the Internet of Vehicles (IoV), smart vehicles with IoT capabilities including sensing, communications, and data processing can possess artificial intelligence to enhance driving aid. The existing works on the applications of DRL in smart vehicles mainly include the studies on autonomous driving, vehicular networks, and vehicular edge/fog computing (VEC/VFC). Table IX lists the related research works and their respective DRL models and DRL algorithms.

1) Autonomous Driving: The application of DRL methods for the control of autonomous vehicles is addressed in many existing works. The authors in [125] review the applications and address the challenges of real-world deployment of DRL in autonomous driving. The existing works mainly discuss the autonomous driving problem of a single vehicle or multiple vehicles. Generally, the autonomous driving problem is formulated as an MDP. In the DRL model, the state may include both the kinematic state and driving environment of vehicles. Generally, the kinematic state refers to position, velocity and other kinematic features of the vehicle whose driving behavior is required to be determined in the model. The driving environment is mainly related to other vehicles nearby, as well as the traffic facilities and obstacles on the road. The information of the driving environment can be obtained by sensing, i.e., using sensors such as radar and camera, or by communicating, i.e., communication in IoV such as vehicle-to-vehicle (V2V) and vehicle-to-Internet (V2I). Moreover, the driving environment can be in the form of an image, i.e., take an image of the surrounding traffic by the on-board camera, or be characterized in the form of vectors, i.e., use variables to represent the state of traffic signals and positions of obstacles. The velocity control and direction control are usually characterized as actions. The rewards are mostly related to assessment criteria of the driving operations, such as driving safety, driving smoothness and driving efficiency.

In the problem of a single vehicle, the driving behavior of one autonomous vehicle is studied. Since the driving environment may affect the driving behavior of the vehicle, the camera image of the environment is considered as the state in the DRL model in [106], and the deep Q-Learning method is applied to control simulated cars. In [107], [108], [112], the proposed methods are also based on deep Q-Learning, the DRL models become more complex, since the driving environment as well as the kinematic state of the vehicle are taken into account.

The actions of the single vehicle include the velocity control and direction control. The velocity control can be in a discrete form as a decision of acceleration or deceleration, or in a continuous form as a decision of a specific value of velocity. The direction control can be in a discrete form, such as lane-changing, turning left or right, or maintaining in the current direction. The direction control can also be in a continuous form, where the specific steering angle is required to be obtained. In most existing works, the velocity or direction control action is in discrete form, where rough decisions on velocity and direction adjustment are made [107], [111], [112], [115]. For examples, in [111], road geometry is taken into account in the MDP model in order to be applicable for more diverse driving styles, and discrete velocity and direction control are involved. The study in [112] aims to optimize the driving utility of the autonomous vehicle, and enables the autonomous vehicle to jointly select the discrete motion planning action performed on the road and the communication action of querying the sensed information from the infrastructure. In some existing works, the velocity or direction control is in continuous form. For instance, the problem of ramp merging in autonomous driving is tackled in [113], and the specific velocity and steering angle of the vehicle is decided.

As for the reward in DRL model for autonomous driving of a single vehicle, driving safety is usually concerned, and the reward function is related to the collision avoidance performance of the vehicle. For example, in [108], the authors address the autonomous driving issues by presenting an RL-based approach, which is combined with formal

TABLE IX
SUMMARY OF DRL MODELS AND ALGORITHMS IN RESEARCH WORKS FOR IOT SMART VEHICLES

Theme	Ref.	DRL Model					DRL Algorithm	Agent Location
		State	Action	Reward	Environment	Feature		
Autonomous Driving	[106]	driving environment	velocity control, direction control	driving safety, driving smoothness, driving efficiency	perception layer	basic	deep Q-Learning	on board (centralized)
	[107]	driving environment, kinematic state	velocity control, direction control	driving efficiency	perception layer	basic	deep Q-Learning	on board (centralized)
	[108]	driving environment, kinematic state	direction control	driving efficiency	perception layer	basic	deep Q-Learning	on board (centralized)
	[109]	driving environment, kinematic state	velocity control, direction control	driving smoothness, driving efficiency	perception layer	basic	policy gradient	road-side unit (centralized)
	[110]	driving environment, kinematic state	velocity control, direction control	environmental benefits	perception layer	basic	deep Q-Learning	road-side unit (centralized)
	[111]	driving environment	velocity control, direction control	driving safty	perception layer	basic	deep Q-Learning	on board (centralized)
	[112]	driving environment, kinematic state	velocity control, direction control	driving efficiency	perception layer, network layer	basic	deep Q-Learning	on board (centralized)
	[113]	driving environment	velocity control, direction control	driving smoothness, driving efficiency	perception layer	basic	actor-critic	on board (centralized)
	[114]	driving environment, kinematic state	velocity control, direction control	driving safety	perception layer	basic	deep Q-Learning	on board (centralized)
	[115]	driving environment	velocity control, direction control	driving safety, driving efficiency, environmental benefits	perception layer	MA	coordinated MA DQN	road-side unit (distributed)
Vehicular Network	[116]	channel state, vehicle state, performance requirements	transmission control	reliability, transmission efficiency	network layer	basic	deep Q-Learning	BS (centralized)
	[117]	channel state, vehicle state	transmission control, velocity control, direction control	reliability, transmission efficiency, driving efficiency	perception layer, network layer	basic	deep echo state network	on board (centralized)
	[118]	resource state	transmission control	reliability, resource utilization	network layer, appication layer	basic	DDQN	edge server (centralized)
	[119]	channel state, vehicle state, resource state	transmission control	reliability, driving efficiency, environmental benefits	network layer	basic	DQN	on board (centralized)
Vehicular Edge/Fog/Cloud Computing	[120], [121]	kinematic state, resource state	offloading decision	task handling efficiency	application layer	basic	A3C	edge server (centralized)
	[122]	resource state, request state	offloading decision	task handling efficiency, environmental benefits	application layer	basic	deep Q-Learning	cloud server (centralized)
	[123]	resource state	offloading decision, resource allocation	reliability, task handling efficiency	network layer, application layer	basic	deep Q-Learning	cloud server (centralized)
	[124]	request state	offloading decision, caching strategy	task handling efficiency	application layer	basic	deep Q-Learning	edge server (centralized)

safety verification to ensure that only safe actions are chosen at any time. A DRL agent learns to drive as close as possible to the desired velocity by executing reasonable lane changes on simulated highways with an arbitrary number of lanes. Besides, the driving efficiency and driving smoothness are also discussed in some existing works. For examples, the authors in [109] use Flow to develop reliable controllers in mixed-autonomy traffic scenarios, and the reward function is related to the average velocity of vehicles, with an added penalty to discourage accelerations or excessive lane-changes. The authors in [126] apply a continuous, model-free DRL algorithm for autonomous driving. The distance traveled by the autonomous vehicle is used to evaluate the reward in the model. Moreover, some researchers also consider the

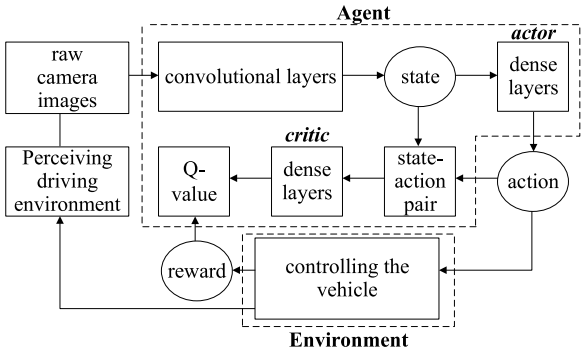


Fig. 17. Actor-critic method for vehicle control problems in autonomous driving.

environmental benefits in autonomous driving, and vehicle fuel consumption or power consumption is taken into account.

There are also studies on the cooperative driving of multiple vehicles. The types of state, action and reward in the DRL model are similar to those in the single-vehicle cases. However, in multi-vehicle cases, the DRL algorithms may involve the MA methods, and the implementation of the system may be in distributed form. In [114], the authors present a novel method of cooperative movement planning. RL is applied to solve this decision-making task of how two cars coordinate their movements to avoid collisions and then return to their intended path. An MA multi-objective RL traffic signal control framework is proposed in [115], which simulates the driver's behavior, e.g., acceleration or deceleration, continuously in space and time dimensions.

When evaluating the performance of proposed DRL-based methods for autonomous driving, the researchers mostly test their methods in simulated traffic environment. A dataset can be obtained by using simulators such as JavaScript Racer [106], Vdrift Ruddskogen track [107], and Auto Drive Simulator [112], etc. Different from testing in a simulated environment, some works use real traffic dataset in their experiments, which can better reflect the real traffic environment [108], [127].

Fig. 17 shows an example of applying DRL algorithms in an autonomous driving problem, where actor-critic algorithm is adopted. The driving environment and kinematic state is characterized as the state vector in the DRL model. An action is selected according to the policy, and the Q-value of each possible state-action pair is estimated. The Q-values act as the critic in the model, and give a guideline for action selections. The selected action directly determines the controlling of the smart vehicle. After executing the control command, the vehicle obtains a reward which can influence the critic in the model.

2) Vehicular Networks: The concept of vehicular networking brings a new level of connectivity to vehicles, and has become a key driver of ITS. The control functionalities in the vehicular network can be divided into three parts according to their usages, including communication control, computing control and storage control [128]. In [129] and [130], the applications of ML in studying the dynamics of vehicular networks and making informed decisions to optimize network performance are discussed. In vehicular networks, problems such as resource allocation, caching, and networking can be formulated and solved via DRL.

In the existing works, the states include the transmission channel status, vehicle status, resource status, and performance requirements. The actions mainly focus on message transmission control in the network. The reliability and transmission efficiency are mostly concerned when defining the reward function. For example, in [117], a DRL algorithm based on echo state network (ESN) cells is proposed in order to provide an interference-aware path planning scheme for a network of cellular-connected UAVs. The state focuses on the external input of UAV and state of the reservoir of UAV. In [116], the performance requirement, i.e., the delay constraint, is further considered when defining the state in the DRL model. The authors use a DRL approach to perform joint resource allocation and scheduling in V2V broadcast communications. In the system, each vehicle makes a decision based on its local observations without the need of waiting for global information. The resource state, i.e., the available BS and the available cache, is considered in [118], where the authors develop an integration framework that enables dynamic orchestration of networking, caching, and computing resources to improve the performance of vehicular networks. The resource allocation strategy is formulated as a joint optimization problem, in which the gains of networking, caching and computing are all taken into consideration. To solve the problem, a double-dueling-deep Q-network algorithm is proposed. Similarly, deep Q-Learning is applied in [119] to learn a scheduling policy, which can guarantee both safety and quality-of-service (QoS) concern in an efficient vehicular network.

3) Vehicular Edge/Fog/Cloud Computing: Emerging vehicular applications require more computing and communication capabilities to perform well in computing-intensive and latency-sensitive tasks. Vehicular Cloud Computing (VCC) provides a new paradigm in which vehicles interact and collaborate to sense the environment, process the data, propagate the results and more generally share resources [131]. Moreover, VEC/VFC focuses on moving computing resources to the edge of the network to resolve latency constraints and reduce cloud ingress traffic [132]–[134].

As studied in [120] and [121], the vehicular edge, fog or cloud computing problems focus on the service offloading issues in the IoV. The state in the DRL model focuses on the resource and request status in the system. The determination of offloading decisions for the multiple tasks is considered as a long-term planning problem. In the existing works, service offloading decision frameworks are proposed, which can provide the optimal policy via DRL. For examples, the authors in [122] propose an optimal computing resource allocation scheme to maximize the total long-term expected return of the VCC system.

With multiple access edge computing techniques, roadside units (RSUs) can provide fast caching services to moving vehicles for content providers. In [124], the authors apply the MDP to model the caching strategy, and propose a

heuristic Q-Learning solution together with vehicle movement predictions based on an LSTM network.

4) Comparison and Insights: By summarizing and comparing the above literature, the following insights can be obtained.

- *System model:* The DRL problems for smart vehicles may be designed for a single vehicle or multiple vehicles. In autonomous driving, most existing works are single-vehicle problems, which can be solved by basic DRL algorithms such as deep Q-Learning and policy gradient methods. Some problems are multi-vehicle problems, where MA DRL algorithms are needed. In the vehicular network and vehicular edge/fog/cloud computing problems, the system usually involves multiple vehicles.
- *DRL model:* The DRL model states, actions and rewards of related works are summarized in Table IX. When defining the state in DRL models in autonomous driving problems, the kinematic state of the single or multiple vehicle(s) as well as the driving environment can be involved. The kinematic state is related to the kinematic features of the smart vehicle(s), and is usually be defined in the form of vectors or matrices. The driving environment refers to the traffic surrounding the smart vehicle(s), and can be characterized as images or vectors. The actions can be in discrete or continuous form, with velocity and moving direction concerned. The typical rewards in existing works are based on some driving criteria, including safety, smoothness, efficiency and environmental benefits. For the sake of feasibility and simplicity, few works consider these criteria at the same time. In future works, researchers can make efforts on an overall consideration of these criteria, and design reasonable priorities among them.
- *DRL algorithm:* As shown in Table IX, when the state and action are in discrete form, deep Q-Learning is mostly adopted. When the state or action space is continuous, the actor-critic method, e.g., A3C or DQN is applied [113], [115], [120], [121].
- *Implementation:* A powerful IoT device like a smart vehicle can act as an agent in the DRL model. The vehicle-mounted servers are able to perform DRL algorithms on board. For example, in [107], the authors trained their agent using CUDA-based GPU acceleration on commodity hardware and run the experiments on a Macbook Pro with an Nvidia GeForce GT 750M GPU. The hardware is relatively modest by most standards, and this indicates that such a number of computational resources is enough for implementing DQNs.

E. AIoT Perception Layer-Smart Grid

The integration of distributed renewable energy sources (DRES) into the power grid introduces the need for autonomous and smart energy management capabilities in the smart grid due to the intermittent and stochastic nature of renewable energy sources (RES). With advanced metering infrastructure (AMI) and various types of sensors in the power grid to collect real-time power generation and demand data,

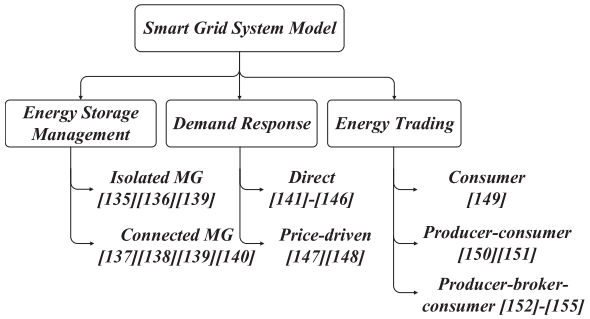


Fig. 18. Classification of smart grid system models.

RL and DRL provide promising methods to learn efficient energy management policies autonomously in such a complex environment with uncertainty. Specifically, the historical data can be leveraged by powerful DRL algorithms in learning optimal decisions to cope with the high uncertainty of the electrical patterns. The existing works mainly include studies on energy storage management (ESM), demand response (DR) and energy trading. We classify the existing research based on the different considerations in system models as shown in Fig. 18. Table X lists the related research works and their DRL models and DRL algorithms.

1) Energy Storage Management: Microgrids (MGs) are small-scale, self-supporting power networks driven by on-site generation sources. They normally integrate RESs such as solar and wind, as well as energy storage elements such as the electrochemical battery. An MG can either connect or disconnect from the external main grid to operate in a grid-connected or isolated-mode. The most essential task in energy management of MG is to fully exploit the renewable energy power, while satisfying the critical requirement that the power generation and consumption are balanced. However, the main challenge lies in the lack of knowledge of future electricity generation and consumption. One promising method to deal with this challenge is through energy storage. Direct energy storage such as the battery is one of the energy storage options. However, the battery also brings new challenges in energy management. As the maximum amount of energy that can be charged or discharged at a certain point of time is limited by the energy storage capability and the current state-of-charge (SoC) of the battery, while the current SoC is in turn determined by the previous charging/discharging behavior, energy management becomes a sequential decision problem for a dynamic system where earlier decisions influence future available choices.

The techniques using RL/DRL to determine the optimal charging/discharging policy for ESS in MG have been studied in some recent literature [135]–[137]. Specifically, [135] and [136] model the system as an isolated MG and only consider battery SoC as the state of the DRL model. The reward in [135] uses the energy balance within the MG, taking into account the energy demand and the battery SoC level. In [136], on the other hand, the reward is defined by the ESS operation cost, which can be represented by the battery loss based on charge/discharge operation. Grid-connected mode MG is considered in [137] where MG is connected to the main grid.

TABLE X
SUMMARY OF DRL MODELS AND ALGORITHMS IN RESEARCH WORKS FOR SMART GRID

Theme	Ref.	DRL Model				DRL Algorithm	Agent Location
		State	Action	Reward	Feature		
Energy Storage Management	[135]	battery SoC	ESS management	energy balance	POMDP	DQN	EMS (centralized)
	[136]	battery SoC	ESS management	ESS operation cost	basic	Q-Learning	EMS (centralized)
	[137]	time state, RE generation state, battery SoC, energy demand state	ESS management	energy trading cost	basic	Q-Learning	EMS (centralized)
	[138]	Energy demand state, RE generation state, price state, battery SoC	DG energy dispatch, ESS management, energy trading amount	MG operational cost	basic	ADP and RNN	EMS (centralized)
	[139]	energy demand state, battery SoC, RE generation state	DG energy dispatch, ESS management, energy trading amount	energy balance;	basic	ADP and actor-critic	EMS (centralized)
	[140]	energy demand state, price state, battery SoC	DG energy dispatch, ESS management, energy trading amount	ESS operational cost	basic	DQN	MG controller (centralized)
Demand Response	[141]	price state, energy demand state	DR devices on/off change	load shedding cost	basic	DQN and DPG	EMS (centralized)
	[142]	time state, energy demand state, price state	DR devices on/off change	load shedding cost, consumer's satisfaction	basic	Q-Learning	EMS (centralized)
	[143]	price state, energy demand state	DR devices on/off change	load shedding cost, consumer's satisfaction	basic	Q-Learning with fixed step size	EMS (centralized)
	[144]	time state, DR device on/off state	DR devices on/off change	load shedding cost	POMDP	DQN with CNN architecture	broker (centralized)
	[145]	battery SoC	DR devices on/off change	load shedding cost	basic	fitted Q-iteration	device controller (centralized)
	[146]	time state, DR device on/off state	DR devices on/off change	load shedding cost	basic	fitted Q-iteration	device controller (centralized)
	[147]	energy demand state, price state	energy trading price	load shedding cost, energy trading cost	basic	Q-Learning	broker (centralized)
	[148]	energy demand state, price state	energy trading price	load shedding cost, energy trading cost	basic	Q-Learning	broker (centralized)
Energy Trading	[149]	price state	energy trading price	energy trading profit	MA	Q-Learning	consumer (distributed)
	[150]	energy demand state, RE generation state, price state, battery SoC	energy trading amount	energy trading profit	basic	Q-Learning	MG (centralized)
	[151]	energy demand state, battery SoC	energy trading amount	energy trading profit	POMDP	DQN+CNN	MG (centralized)
	[152]	time state, energy demand state, price state	energy trading price	energy trading profit	basic	Q-Learning with virtual experience	broker (centralized)
	[153]	Energy demand state, price state	energy trading price	energy trading profit	MA	Q-Learning with virtual experience	broker and consumer (distributed)
	[154]	energy demand state, price state	energy trading price	energy trading profit	basic	Q-Learning	broker (centralized)
	[155]	energy demand state, battery SoC, price state, time state	energy trading amount and price	energy trading profit	MA	Q-Learning	producer and consumer (distributed)

The state space consists of time state, renewable energy generation state, battery SoC and energy demand state. The reward is defined as the negative of the energy transaction cost.

In [138], [139], [140], except for ESS management, diesel generators (DGs) energy dispatch and energy trading are taken into account as part of the action space. There are two types of energy trading actions. One is the amount of energy that is traded with the main grid or other MGs, and the other refers to the price for energy transaction. In [138], the reward function is the negative of the MG operation cost which consists of DG cost, load shedding cost, energy transaction cost and ancillary services cost. To derive the optimal policy, approximate dynamic programming (ADP) and RNN learning are employed to solve the finite-horizon MDP problem. In [139], the system is modeled in both isolated and connected modes and the reward functions represented by energy balance for both modes are given respectively. This work combines ADP and actor-critic framework to solve the dynamic optimization problem in EMS.

a) Leverage historical information to deal with uncertainty: Due to the uncertainty of future renewable energy generation and load demand, MG system dynamics are often

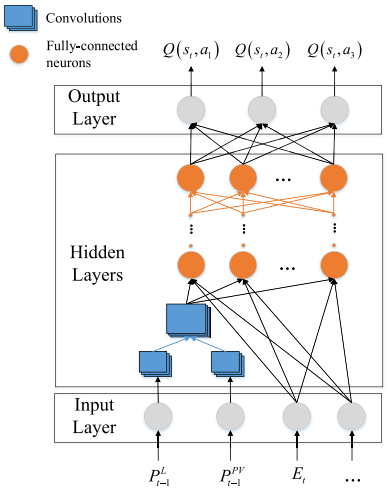
TABLE XI
A COLLECTION OF DATASETS FOR RESEARCH ON SMART GRID

Dataset	Description
PJM	public data published by regional transmission organization PJM, such as renewable energy generation data, real-Time locational marginal pricing data.
CAISO	real-time data related to the ISO transmission system and its Market, such as system demand forecasts, transmission outage and capacity status, market prices and market result data.
MISO	all aspects of real-time and day-ahead energy and ancillary services markets and reliability coordination for the region.
MIDC	an access to Oahu Solar Measurement Grid data by Measurement and Instrumentation Data Center (MIDC).

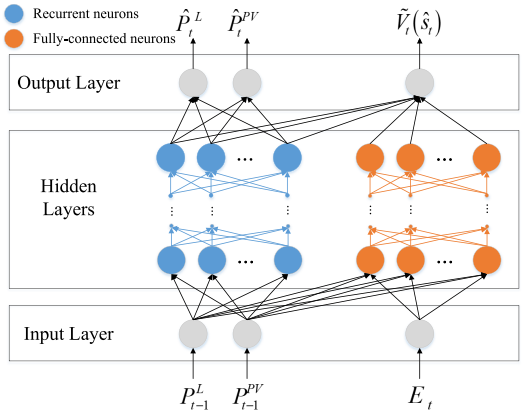
formalized as a POMDP in existing research [135], [138]. The observation of the POMDP model is normally made up of time series of historical data for renewable energy generation and load demand, along with other observable current system states such as the battery SoC level. The agent takes action according to the historical observation and obtains new observation which is then used for historical sequence updates. In [135], three discrete actions are considered, i.e., charge, discharge, or idle. A DQN-type algorithm is used to solve this POMDP-based DRL model. The authors tried both CNN and RNN to process the time series of historical data as shown in Fig. 19(a), where the historical data is first processed by a set of CNN or LSTM layers, and then the output from the CNN/LSTM layers as well as other inputs are processed by the fully connected layers and output layer. The authors conclude that both NN architectures obtain close results. Compared with CNN, RNN is more widely adopted by other literature due to its capability to better capture the information in historical features. Fig. 19(b) shows a designed RNN architecture for a POMDP-based ESM problem in [138], where RNN and approximate policy iteration algorithm are used to solve the problem. The NN consists of two parts as shown in Fig. 19(b), where the left part is an RNN that takes the renewable energy generation and load demand at previous time steps as input, and output the estimated energy generation and consumption at the current time step to derive the immediate reward. On the other hand, the right part is a feed-forward network whose input includes the historical data after it is processed by the RNN input layer of the left part, as well as other state information such as the battery SoC. The output of the right part is the approximated value functions of the input observations.

In the above research on MG, a mass of historical data is essential when solving a POMDP-based DRL model. Table XI lists a collection of datasets for research on smart grid. The time series data provided by these datasets include historical real-time price, energy load, renewable energy generation, etc, which can be used to simulate the real-time smart grid system.

2) Demand Response: Another method to support the integration of DRES is through DR systems, which dynamically adjust electrical demand in response to changing electrical energy prices or other grid signals. Thermostatically controlled



(a) A CNN architecture for POMDP-based ESM problem.



(b) A RNN architecture for POMDP-based ESM problem.

Fig. 19. DRL algorithms for POMDP-based ESM problem.

loads (TCLs) such as electric water heaters are a prominent example of loads that offer flexibility at the residential level. In fact, TCLs can be seen as a type of energy storage entity through the power to heat conversion, which is in contract to the direct energy storage entity such as a battery. DR can be divided into direct DR and price-driven DR, where the energy consumption profile of a user is adjusted according to a utility in the former while according to the price in the latter. In any case, the energy consumers need to make a continuing sequence of decisions as to either consumes energy at current (known) utility/price or to defer power consumption until later at possibly unknown utility/prices.

Research works in [141], [142], [143], [144], [145], [146], belong to direct DR where energy consumption profile is adjusted according the utility. Among them, [141], [142], [143], perform optimization of schedules for residential or commercial buildings, while [144], [145], [146], perform optimization of schedules for a cluster by using the thermal storage.

In [141], [142], [143], the state space is made up of price state, energy demand state and/or time state, and the action space is DR devices on/off change. Two components are normally being considered for the rewards: one is the controllable load cost which is an important part for demand side management, while the other is the consumer satisfaction which measures the consumer's preference over the DR.

In [144] and [146], the on/off state of DR device is taken into account in state space. Reference [144] formalizes the direct load control problem as a high-dimensional POMDP problem, adding history of observations to the state. The researchers in [144] also proposes a novel DQN with CNN architecture to solve the problem.

Research works in [147] and [148] belong to price-driven DR where energy consumption profile is adjusted according to the price. In these works, the price state is considered as an important component of the state space. The action in [147] is the selection of retail price while in [148] is the incentive rate for consumers. Both of these two research works choose broker as the agent, who provides services between the utility company and consumer by purchasing energy from the utility company and selling it to the consumer. In this way, broker's cost is considered in the reward function.

3) Energy Trading: The integration of the DRES into the power grid blurs the distinction between an energy provider and a consumer. This is especially true for an MG, which may constantly switch its role between a provider or consumer depending on whether its generated energy exceeds or falls short of its demanded energy. In fact, a key goal of smart grid design is to facilitate the two-way flow of electricity by enhancing the ability of distributed small-scale electricity producers, such as small wind farms or households with solar panels, to sell energy into the power grid. Due to the unpredictability of the DRES, an autonomous control mechanism to ensure power supply/demand balance is essential. One promising method is through the introduction of Broker Agents, who buy electricity from distributed producers and also sell electricity to consumers. RL/DRL can be applied for the Broker Agents to learn pricing decisions to effectively maintain that balance, and earn profits while doing so, contribute to the stability of the grid through their continued participation.

There are three types of entities in the scenario of energy trading where they play different roles in a smart grid system, i.e., producer, broker and consumer. Consumers' energy consumption may influence the producer's decision on the wholesale energy market price. Broker buys energy from the producer through a wholesale energy market and provides it to the consumers through a retail electricity market.

In [149], only consumer is considered in energy trading, where an MA system is presented for an isolated MG. Four types of agents are associated with energy generation, energy storage, energy consumption and energy bidding, respectively. References [150] and [151] consider producer and consumer as two entities in energy trading. N MGs are connected with each other and the main grid, where each MG can receive energy from other MGs, main grid, local DGs, RESs and **batteries**. In [150], the amount of energy that MG i intends to trade with MG j is decided by MG i according to its energy demand, energy generation, energy storage as well as price state. Researchers proposed a Hotbooting Q-Learning based energy trading algorithm to improve the utility of the MG and reduce the total energy purchased from the main grid. In [151], MG chooses an energy trading amount by estimating the local energy demand, renewable energy generation, battery level in an experience sequence which consists of the current state and previous state-action pairs. A DQN-based energy trading scheme that uses CNN as the nonlinear function approximator is proposed to compress the state space of MG and make it easier to estimate the Q-value of each energy trading policy.

In [152], [153], [154], [155], multiple energy trading processes among producer, broker and consumer are discussed. References [152] and [154] set broker as the agent who takes energy demand state, price state and/or time state as state input, and chooses action to sell/buy in energy trading. Reference [153] extends the above simple system model to a MA MG system where not only the broker but also each consumer can learn an optimal policy for its decision making. The authors also propose two improvements to allow the DRL model to be conducted online in a fully distributed manner with faster speed.

Reference [155] also presents an MA-based distributed energy and load management approach that involves energy producer, broker and consumer. Each agent is located in either the supply or the demand side. The aim of these agents is to optimize its utility in the auction-based market. Firstly, each agent initializes its learning parameters. Next, in each iteration of the algorithm, the producer agent predicts its production and the consumer agent selects an energy trading amount and/or price and submits it to the market. Then, the amount and price of energy to be traded are determined by the broker and each agent can obtain its reward and update the parameter.

4) Comparison and Insights: By summarizing and comparing the above literature, the following insights can be obtained.

- *System model:* For DR, most of the research works focus on direct DR, while price-driven DR deserves more detailed studies. For Energy Trading, most of the works focus on system models in producer-broker-consumer or producer-consumer mode, few works put the emphasis on consumer-broker or producer-broker mode.
- *DRL model:* For ESM problem, battery SoC is usually considered as part of the input state. When facing partial observability problems, historical features can be used as input states and POMDP can be formalized. As the MA problem is widely considered in Energy Trading, DRL models for different agents are usually built in a distributed way.
- *DRL algorithm:* As is shown in Table X, Q-Learning and DQN have been widely used and developed to solve different kinds of problems due to their simplicity, good performance and the fact that the action space in smart grid problem is not relatively large and is always discrete. So, it is convenient to use these value-based methods of DRL algorithms. Classical NN architecture can be used to improve the performance of DRL methods. The work in [138] and [144] have taken advantage of RNN and

CNN respectively to extract more features in the input state. Actor-critic methods can be used when it requires fewer samples and less computational resources to train the DRL models. Moreover, compared with value-based methods like DQN, it is possible to learn stochastic policies and solve RL problems with continuous actions by using actor-critic methods.

- *Implementation:* In ESM, most DRL algorithms are centralized implemented in EMS. As for DR, most research concentrate on centralized rather than distributed implementation. It is worthwhile to devote more attention to the cooperation between the DR devices.

F. Comparison Between General DRL Model in AIoT and DRL Models in the Literature

Comparing the general DRL model in AIoT proposed in Section III, and the DRL models in existing works reviewed in Section IV, the following facts are summarized:

- *IoT Communication Networks:* Table VI shows that most existing research has focused only on the network layer, while there are a few studies in the literature that consider both the perception layer and the network layer in WSAN.
- *IoT Edge/Fog/Cloud Computing Systems:* Table VII shows that most existing research has focused on the application layer, while there are a few studies in the literature that consider both the application layer and the communication layer, i.e., joint communication and computation resource control.
- *Autonomous Robots:* Table VIII shows that most existing research has focused on the perception layer. In cloud robotics, both the perception layer and the application layer have been involved.
- *Smart Vehicles:* Table IX shows that in most existing works, the autonomous driving problems have focused on the perception layer of AIoT. In vehicular networking problems, most works have only been related to the network layer. There are a few studies in the literature that are related to both the perception layer and the network layer where the vehicle control and vehicular networking have been simultaneously considered [112], [117]. As for the vehicular edge/fog/cloud computing problems, the majority of the studies have focused on the application layer, while only a few research works are related to both the network layer and the application layer [118] where the performance of not only caching and computing but also networking have been taken into consideration.
- *Smart Grid:* Table X shows that current works have mainly focused on the perception layer of AIoT systems.

While the majority of existing research has focused on only one of the three layers of AIoT systems, it will be interesting to explore the joint and integrated control of multiple layers of AIoT systems. For example, sensors are most widely used for perceiving the environment in autonomous driving problems. The cameras on the smart vehicles are used for taking pictures of the traffic environment, and the on-board radars are used to sense the nearby vehicles or obstacles. Based on the information obtained by sensors, the agents make decisions on the controlling of the smart vehicles. On the other hand, communications in IoV enable information exchange between smart vehicles. The communications can help the smart vehicles better perceive the traffic environment, but may also introduce extra delay and affect the reliability of decision making in vehicle control. In other words, the performance of the communication network may influence the effectiveness of message transmission and thereby influence the vehicle control. In current vehicular networking problems, most works are only related to the network layer and focus on optimizing network performance such as delay and reliability. In the future works, researchers can make effort on an effective combination of autonomous driving and IoV communications issues, where the communication resource control and vehicle control are jointly optimized, while the reward can be determined by the ultimate objective - driving performance of the controlled vehicle.

Similarly in most existing works on smart grid, the system state and observation information are considered to be readily available to the agents for optimal decision. However in reality, as the state information needs to be obtained by sensors distributed throughout the smart grid, and transmitted to the EMS possibly from remote areas over long distance via the IoT communications network [156], the communication layer can be included in the DRL model as well to reflect the real-world problem more accurately. Although the problem of smart grid state estimation over IoT networks has been addressed in some recent research works [**?**], [157], there have not been studies on incorporating the delay and inaccuracy in state estimation in the DRL models for optimal control of smart grid to the best of our knowledge.

V. Challenges, Open Issues, and Future Research Directions

Although DRL is a powerful theoretical tool that is well-suited to the task of introducing artificial intelligence to AIoT systems, there are still a lot of challenges and open issues to be overcome and addressed. The following lists some of the future research directions in this area.

A. Incomplete Perception Problem

In AIoT systems, it might not be possible for the agent to have perfect and complete perception of the state of the environment. This could be due to

- limited sensing capabilities of sensors in the perception layer;
- information loss due to limited transmission capability in the network layer;

An important challenge in applying DRL to AIoT system is to learn with incomplete perception or partially observable states. The MDP model is no longer valid, as the state information is no longer sufficient to support the decision on optimal action. The action can be improved if more information is available to the agent in addition to the state information. Although the DRL algorithms and methods introduced in Section III-E can be applied, there are still some open

issues with the POMDP-based DRL algorithms. Firstly, an agent in POMDP needs to select an action based on the observation history space which grows exponentially. Approaches proposed for this problem require large memory and can only work well for small discrete observation spaces [158]. Secondly, when introducing belief state to POMDP problems, the belief space will not grow exponentially but the knowledge of the model becomes essential for the agent, which is not suitable for many complicated scenarios. Finally, nearly all these algorithms in POMDP problems need to face a challenge referred to as information gathering and exploitation dilemma. In a POMDP, the agent does not know what the current state is exactly. It needs to decide whether to gather more information about the true state first or to exploit its current knowledge first. Obviously, in order to find the optimal policy, an agent in POMDP needs to have more interactions with the environment. Apart from the above challenges associated with POMDP-based DRL problems, the DRL model formulation and parameter optimization for various AIoT systems are different case by case. Moreover, more efficient algorithms could be designed according to the specific characteristics of AIoT systems.

B. Delayed Control Problem

In DRL problems, we normally consider that an action is exerted as soon as it is selected by the agent, and a corresponding reward is immediately available at the agent. However, a challenge in applying DRL to real-world AIoT system is to learn despite the existence of control delay, i.e., the delay between measuring a system's state and acting upon it. Control delay is always present in real systems due to transporting measurement data to the learning agent, computing the next action, and changing the state of the actuator. Therefore, it is important to design RL/DRL algorithms which take the control delay into account.

Most of the existing RL algorithms don't consider the control delay. At each time step *t*, the state s_t of the environment is observed, and an action a_t is immediately determined by the agent. However in practice, the actual action $\hat{a}_t$ executed at time step *t* might be the action generated τ time steps before, i.e., $\hat{a}_t = a_{t-\tau}$. In this case, the next state s_{t+1} depends on the current state and a previously determined action, i.e., $(s_t, a_{t-\tau})$, instead of the current state and currently determined action pair (s_t, a_t), which makes the state transition violating the Markov property. Therefore, the MDP model based on which RL/DRL algorithms are developed are no longer valid and a POMDP model is more appropriate.

In order to deal with the delayed control problem, existing works in RL developed several methods [159]–[161]. The first method [159] incorporates the past actions taken during the length of the delay into the current state in formulating an MDP model, so that the classical RL methods such as TD-learning and Q-Learning can be applied. However, this method results in larger state space with the state dimensionality depending on the number of time steps for the delay. The second method [160] learns a state transition model so that it can predict the state at which the currently selected action is actually going to be executed. Then, a model-based RL algorithm can be applied. However, the learning process of the underlying model is usually time-consuming and will incur additional delay itself. Finally in the third method [161], the classical model-free RL algorithms such as TD-learning and Q-Learning are applied, except that at each time step *t*, the Q-function $Q(s_t, \hat{a}_t)$ with respect to current state s_t and actually executed action $\hat{a}_t$ is updated, instead of the normal $Q(s_t, a_t)$ with respect to current state s_t and currently generated action a_t.

The above methods mostly focus on the constant delay problem. However, the actual delay in an AIoT system is likely to be stochastic. Moreover, the delay can depend on the communication and computation resource control actions in the IoT communications networks and edge/fog/cloud servers. Therefore, developing RL algorithms to consider stochastic control delay or control delay that depends on other parameters is an open issue. Another important challenge is how to extend the above algorithms from RL to DRL leveraging the powerful NNs while dealing with the intrinsic complexities.

C. Multi-Agent Control Problem

The agent in RL is a virtual concept that learns the optimal policy by interacting with the environment. In AIoT system, agents can be implemented in IoT devices, edge/fog servers, and cloud servers as discussed previously. For a single RL task, there are some typical scenarios for the implementation of agents:

- *centralized architecture:* a single agent in a cloud server, edge/fog node, or an IoT device;
- *distributed architecture:* multiple agents with each agent implemented in an IoT device or edge/fog server;
- *semi-distributed architecture:* one centralized agent in a cloud server or edge/fog server and multiple distributive agents in edge/fog servers or IoT devices.

For distributed and semi-distributed architecture, it is an important challenge to enable efficient collaboration and fair competition among multiple agents in a single RL task. The tasks of each agent in a MA system may be different, and they are coupled to each other. Therefore, the design of a reasonable joint reward function becomes a challenge, which may directly affect the performance of the learning policy. Compared to the stable environment in the single-agent RL problem, the environment in the MA RL is complex and dynamic, which brings challenges to design of MA DRL approaches.

In most existing MA DRL methods, the agents are assumed to have same capability. For examples, the robots in a multi-robot system have the same manipulation ability, or the multiple vehicles in a cooperative driving scenario have the same kinematic performance. Thus, the application of DRL in heterogeneous MA systems remains to be further studied. The heterogeneity makes cooperative decision more complex, since each agent needs to model other agents when their capabilities are unknown. Although the MA DRL algorithms and methods introduced in Section III-F can be applied to solve the problem of space explosion and guarantee the convergence of the algorithm, the DRL model formulation, parameter optimization,

as well as algorithm adaptation and improvement remain to be open issues. Moreover, significant progress in the field of MA RL can be achieved by a more intensive cross-domain research between the fields of ML, game theory, and control theory.

D. Joint Resource and Actuator Control Problem

In AIoT systems, there are two levels of control, i.e., resource control and actuator control as discussed previously. Although the ultimate objective is to optimize the long-term reward of the physical system by selecting appropriate actuator control actions, the computation and network resource control actions will impact the physical system performance through their effects on the network and computation system performances. For example, an efficient network resource control policy can result in larger data transmission rates for the sensory data, and thus allow more information to be available at the cloud server for the agent to derive an improved policy. Currently, most existing research works either optimize the computation and/or network performances for IoT systems, or optimize the physical system performance considering an ideal communication and computation environment. Therefore, how to jointly optimize the two levels of control actions to achieve an optimized physical system performance is an important open issue for applying DRL in AIoT system.

When the RL/DRL environment includes more than one layer in AIoT architecture, the corresponding RL/DRL model will be more complex as discussed in Section III. For example, instead of optimizing normal network performance such as transmission delay, transmission power, and packet loss rate in the network layer, the communication resource control actions need to be selected to optimize the control performance of a physical autonomous system, which may be a function of the network performance. In order to optimize the control performance, the best trade-off between several network performance metrics may need to be considered. For example, larger amount of sensory data may be transmitted at the cost of larger transmission delay, which relieves the incomplete perception problem but deteriorates the delayed control problem as discussed above.

There are many challenges to model and solve such complex RL/DRL problems. Firstly, feature selection is an crucial task. An appropriate feature selection can lead to better generalization which is helpful for the bias-overfitting tradeoff. When too many features are taken into consideration, it is hard for the agent to determine which features are more indispensable. Although some features may play a key role in reconstruction of the observation, they may be discarded because they are not related to the current task directly. Secondly, the selection of algorithm and function approximator is also a tough task. The function approximator used for value function or policy converts the features into abstraction in higher level. Sometimes the approximator is too simple to avoid the bias, while sometimes the approximator is too complex to obtain a good generalization result from the limited dataset, i.e., overfitting. Errors resulted from this bias/overfitting problem need to be overcome, so an appropriate approximator needs to be used according to the current task. Thirdly, in such complex RL/DRL problems, the objective function needs to be modified. Typical approaches include reward shaping and discount factor tuning. Reward shaping adds an additional function $F(s_t, a_t)$ to the original reward function $r(s_t, a_t)$. It is mainly used for DRL problems with sparse and delayed rewards [162]. Discount factor tuning helps to adjust the impact of temporally distant rewards. When the discount factor is high, the training process tends to be instable in convergence and when the discount factor is low, some potential rewards will be discarded [26]. Hence, modifying the objective function can help to tackle the above problems to some extent.

VI. Conclusion

This paper has presented the model, applications and challenges of DRL in AIoT systems. Firstly, a summary of the existing RL/DRL methods has been provided. Then, the general model of AIoT system has been proposed, including the DRL framework for AIoT based on the three-layer structure of IoT. The applications of DRL in AIoT have been classified into several categories, and the applied methods and the typical state/action/reward in the models have been summarized. Finally, the challenges and open issues for future research have been identified.

Appendix

A. Building Blocks of DRL-Reinforcement Learning

Generally, reinforcement learning (RL) is a type of algorithms in machine learning (ML) that can achieve optimal control of a Markov Decision Process (MDP) [4]. As discussed in Section I, there are generally two entities in RL as shown in Fig. 20 - an agent and an environment. The environment evolves over time in a stochastic manner and may be in one of the states within a state space at any point in time. The agent performs as the action executor and interacts with the environment. When it performs an action under a certain state, the environment will generate a reward function as signals for positive or negative behaviour. Moreover, the action will also impact on the next state that the environment will transit to. The stochastic evolution of the state-action pair over time forms an MDP, which consists of the following elements.

- state s, which is used to represent a specific status of environment in a possible state space $\mathcal{S}$. In MDP, the state comprises all the necessary information of the environment for the agent to choose the optimal action from the action space.
- action a, which is chosen by the agent from an action space $\mathcal{A}$ in a specific state s. An RL agent interacts with the environment and learn how to behave in different states by observing the consequences of its actions.
- reward $r(s, a)$, which is generated when the agent takes a certain action a in a state s. Reward indicts the intrinsic desirability of an action in a certain state.
- transition probability $P(s'|s, a)$, which is the conditional probability that the next state of system will be $s' \in \mathcal{S}$ given the current state s and action a. In model-based RL, this transition probability is considered to be known by

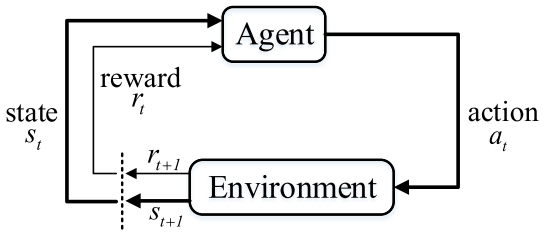


Fig. 20. Reinforcement learning process.

the agent, while agent does not require this information in model-free RL.

A policy determines how an agent selects actions in different states, which can be categorized into either a stochastic policy or a deterministic policy [26]. In stochastic case, the policy is described by $\pi(s, a) \coloneqq P(a|s)$, which denotes the probability that an action a may be chosen in state s. In deterministic case, the policy is described by $\pi(s) \coloneqq a$, which denotes the action a that must be chosen in state s.

For simplicity of introduction, we focus on the discrete time model, where the agent interacts with the environment at each of a sequence of discrete time steps $t = 0, 1, 2, 3, \ldots$ The goal of agent is to learn how to map states to actions, i.e., to find a policy π to optimize the value function $V^{\pi}(s_0)$ for any state $s_0 \in \mathcal{S}$. The value function $V^{\pi}(s_0)$ is the expected reward when a policy π is taken with an initial state s_0, i.e.,

$$V^{\pi}(s_0) = \mathrm{E}_{\tau_{s_0} \sim \pi}[G(\tau_{s_0})], \tag{24}$$

where E stands for expectation, τ_{s_0} is a trajectory or sequence of triplets (s_t, a_t, r_{t+1}), $t \in \{0, 1, 2, \ldots\}$ with $r_{t+1} = r(s_t, a_t)$, $a_t \sim \pi(s_t, a_t)$ or $a_t = \pi(s_t)$ and $s_{t+1} \sim P(s_{t+1}|s_t, a_t)$. $G(\tau_{s_0}) = \sum_{t=1}^{T} f(r_t)$ can be the total reward, discounted total reward, or average reward of trajectory τ_{s_0}, where T is the terminal time step that can be ∞.

Apart from value function, another important function is Q-function $Q^{\pi}(s_0, a_0)$, which is the expected reward for taking action a_0 in state s_0 and thereafter following a policy π. When policy π is the optimal policy π^*, value function and Q-function are denoted by $V^*(s)$ and $Q^*(s, a)$, respectively. Note that $V^*(s) = \max_a Q^*(s, a)$. If the Q-functions $Q^*(s, a)$, $a \in \mathcal{A}$ are given, the optimal policy can be easily found by $\pi^* = \arg\max_a Q^*(s, a)$.

In order to learn the value functions or Q-functions, the Bellman optimality equations are usually used. Taking the discounted MDP with a discount factor of γ for example, the Bellman optimality equations for value function and Q-function are

$$V^*(s_t) = \max_{a_t} \left[r_{t+1} + \gamma \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t) V^*(s_{t+1}) \right], \tag{25}$$

and

$$Q^*(s_t, a_t) = r_{t+1} + \gamma \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t) \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}), \tag{26}$$

respectively.

Bellman equations represent the relation between the value/Q-functions of the current state and next state. For example, it can be inferred from (26) that the expected reward equals to the sum of the immediate reward and the maximum expected reward thereafter. When the future expected reward is obtained, the expected reward since current state can be calculated. Bellman equations are the basis of an important class of RL algorithms using the "bootstrap" method, such as Q-Learning and Temporal-Difference (TD)-learning. During the learning process, the agent first initializes the value/Q-functions to some random values. Then, it iteratively repeats the policy prediction and policy evaluation phases until the convergence of the value/Q-functions. In the policy prediction phase, the agent chooses an action according to the current value/Q-functions, which results in an immediate reward and a new state. In the policy evaluation phase, it updates the value/Q-functions according to the Bellman equations (25) or (26) given the immediate reward and the new state.

In the policy prediction phase, instead of always selecting the greedy action that maximizes the current value/Q-functions, a "soft" policy such as ϵ-greedy, ϵ-soft, and softmax is usually used to explore the environment seeking the potential to learn a better policy. Moreover, according to the different methods adopted in policy evaluation phase, RL algorithms can be either *on-policy* or *off-policy*, depending on whether the value/Q-functions of the predicted policy or an hypothetical (e.g., greedy) policy is estimated.

B. Building Blocks of DRL-Deep Learning

Deep learning (DL) refers to a subset of ML algorithms and techniques that leverage artificial neural networks (ANN) to learn from large amount of data in an autonomous way. It is able to perform well in tasks like regression and classification. Regression task deals with predicting a continuous value, while classification task predicts the output from a set of finite categorical values. Given input data *X* and output data *Y*, NN models can be viewed as mathematical models defining a function $f : X \to Y$ or a distribution over *X* or both *X* and *Y*. The learning rule of NN modifies its parameters in order for a given input *X*, the network can produce a favored output $\hat{Y}$ that best approximates the target output data *Y*.

A general feedforward NN, as shown in Fig. 21, is constructed by an input layer, one or more hidden layers and an output layer. Each layer consists of one or multiple neurons which represent different non-linear nodes in the model. As illustrated in Fig. 21, neuron *i* in layer *j* has a vector of weights $\mathbf{w}_i^{(j)}$ for the connections from layer *j*–1 to itself and a bias value $b_i^{(j)}$. It also has an activation function *h* such as Sigmoid, Logistic, Tanh and ReLU. The output of neuron *i* in layer *j* equals to $a_i^{(j)} = h(\mathbf{w}_i^{(j)} \mathbf{a}_{j-1}^{\mathrm{T}} + b_i^{(j)})$, where $\mathbf{a}_{j-1}$ is the vector of outputs from neurons in layer *j*–1. The typical parameters of NN are the weights and bias of every node. Any NN with two or more hidden layers can be called Deep Neural Network (DNN).

A feedforward network has no notion of order in time, and the only input it considers is the current input data it has been exposed to. Recurrent neural network (RNN) refers to a special

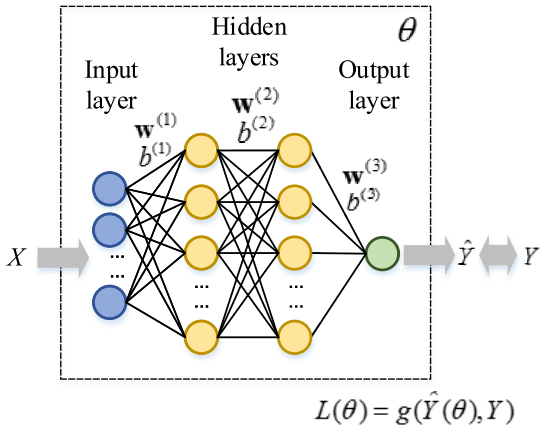


Fig. 21. Feedforward neural network.

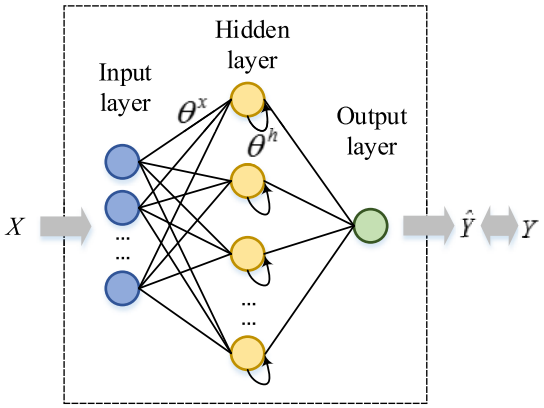


Fig. 22. Recurrent neural network.

type of NNs which can process sequences of inputs by using internal memory. In RNN, the prior output of neurons in hidden state can be used as input along with the current input data, which enables the network to learn from history. The basic architecture of RNN is illustrated in Fig. 22. At each time step, the input of RNN is propagated in the feedforward NN. First, it is modified by the weight matrix θ^x. Meanwhile, the hidden state of the previous time step is multiplied by another weight matrix θ^h. Then, those two parts are added together and activated by the neuron. A special RNN architecture called Long Short-Term Memory (LSTM) is widely used [163]. LSTM is able to solve shortcomings in RNN, i.e., vanishing gradient, exploding gradient and long term dependencies [164].

Usually, a loss function $L(\theta) = g(\hat{Y}(\theta), Y)$ is used in DL, which is a function of the output $\hat{Y}(\theta)$ from NN and the target output Y. The loss function evaluates how well a specific NN along with current learned parameter values θ models the given data $Y = f(X)$.

The objective of the NN is to minimize the loss function, i.e., $\min_\theta L(\theta)$. For this purpose, the parameters θ in NNs are updated by a method called gradient descent. Given a function $L(\theta)$, the simple gradient $\nabla_\theta L(\theta) = \frac{\partial L(\theta)}{\partial \theta}$ is usually used to update the parameters. The gradient descent method starts from an initial point θ_0. As a mini-batch of input data is fed to NN, the average loss function over all input data in the mini-batch is derived, and used to find the minimum of $L(\theta)$ by taking a step along the descent direction, i.e.,

$$\theta \leftarrow \theta - \alpha \nabla_\theta L(\theta), \tag{27}$$

where α is a hyper-parameter named step size. It is set to determine how fast the parameter values move towards the optimal direction. The above process is repeated iteratively as more mini-batches of input data are fed to NN until convergence.

The simple gradient $\nabla_\theta L(\theta)$ is easy to derive, but simple gradient descend is often not the most efficient method to optimize the loss function. During training, an appropriate value of step size α should be set because if the value is too big, it may not be able to reach the local minimum and if the value is too small, it may take too much time to reach the local optimal point. On the other hand, natural gradients $\nabla_\theta^{\mathrm{N}} L(\theta)$ do not follow the usual steepest direction in the parameter space, but along the steepest descent direction with respect to the Fisher metric in the space of distributions. Specifically, the Fisher information metric F is usually used to determine the step size, so that $\nabla_\theta^{\mathrm{N}} L(\theta) = \nabla_\theta L(\theta) F^{-1}$. Then, (27) can be used to update the parameters by replacing $\alpha \nabla_\theta L(\theta)$ with $\nabla_\theta^{\mathrm{N}} L(\theta)$.

REFERENCES

[1] P. J. Antsaklis, K. M. Passino, and S. J. Wang, "An introduction to autonomous control systems," *IEEE Control Syst. Mag.*, vol. 11, no. 4, pp. 5–13, Jun. 1991.

[2] (2018). *Smarter Things: The Autonomous IoT*. [Online]. Available: http://gdruk.com/smarter-things-autonomous-iot/

[3] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, "Deep learning for IoT big data and streaming analytics: A survey," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 2923–2960, 4th Quart., 2018.

[4] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[5] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[6] O. B. Sezer, E. Dogdu, and A. M. Ozbayoglu, "Context-aware computing, learning, and big data in Internet of Things: A survey," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 1–27, Feb. 2018.

[7] F. Samie, L. Bauer, and J. Henkel, "From cloud down to things: An overview of machine learning in Internet of Things," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4921–4934, Jun. 2019.

[8] M. S. Mahdavinejad, M. Rezvan, M. Barekatain, P. Adibi, P. Barnaghi, and A. P. Sheth, "Machine learning for Internet of Things data analysis: A survey," *Digit. Commun. Netw.*, vol. 4, no. 3, pp. 161–175, 2018.

[9] L. Cui, S. Yang, F. Chen, Z. Ming, N. Lu, and J. Qin, "A survey on application of machine learning for Internet of Things," *Int. J. Mach. Learn. Cybern.*, vol. 9, no. 8, pp. 1399–1417, 2018.

[10] F. Zantalis, G. Koulouras, S. Karabetsos, and D. Kandris, "A review of machine learning and IoT in smart transportation," *Future Internet*, vol. 11, no. 4, p. 94, 2019.

[11] Q. Chen *et al.*, "A survey on an emerging area: Deep learning for smart city data," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 3, no. 5, pp. 392–410, Oct. 2019.

[12] B. Qolomany *et al.*, "Leveraging machine learning and big data for smart buildings: A comprehensive survey," *IEEE Access*, vol. 7, pp. 90316–90356, 2019.

[13] E. Hossain, I. Khan, F. Un-Noor, S. S. Sikander, and M. S. H. Sunny, "Application of big data and machine learning in smart grid, and associated security concerns: A review," *IEEE Access*, vol. 7, pp. 13960–13988, 2019.

[14] D. Zhang, X. Han, and C. Deng, "Review on the research and practice of deep learning and reinforcement learning in smart grids," *CSEE J. Power Energy Syst.*, vol. 4, no. 3, pp. 362–370, 2018.

[15] S. K. Sharma and X. Wang, "Towards massive machine type communications in ultra-dense cellular IoT networks: Current issues and machine learning-assisted solutions," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 1, pp. 426–471, 1st Quart., 2020.

[16] M. A. Alsheikh, S. Lin, D. Niyato, and H.-P. Tan, "Machine learning in wireless sensor networks: Algorithms, strategies, and applications," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 4, pp. 1996–2018, 4th Quart., 2014.

[17] M. Chen, U. Challita, W. Saad, C. Yin, and M. Debbah, "Artificial neural networks-based machine learning for wireless networks: A tutorial," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 4, pp. 3039–3071, 4th Quart., 2019.

[18] A. Zappone, M. Di Renzo, and M. Debbah, "Wireless networks design in the era of deep learning: Model-based, AI-based, or both," 2019. [Online]. Available: arXiv:1902.02647.

[19] Q. Mao, F. Hu, and Q. Hao, "Deep learning for intelligent wireless networks: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 4, pp. 2595–2621, 1st Quart., 2018.

[20] N. C. Luong *et al.*, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 4, pp. 3133–3174, 4th Quart., 2019.

[21] K. H. Abdulkareem *et al.*, "A review of fog computing and machine learning: Concepts, applications, challenges, and open issues," *IEEE Access*, vol. 71, pp. 153123–153140, 2019.

[22] T. K. Rodrigues, K. Suto, H. Nishiyama, J. Liu, and N. Kato, "Machine learning meets computation and communication control in evolving edge and cloud: Challenges and future perspective," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 1, pp. 38–67, 1st Quart., 2020.

[23] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proc. IEEE*, vol. 107, no. 8, pp. 1655–1674, Aug. 2019.

[24] H. Zhu, Y. Cao, W. Wang, T. Jiang, and S. Jin, "Deep reinforcement learning for mobile edge caching: Review, new features, and open issues," *IEEE Netw.*, vol. 32, no. 6, pp. 50–57, Nov./Dec. 2018.

[25] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Process. Mag.*, vol. 34, no. 6, pp. 26–38, Nov. 2017.

[26] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An introduction to deep reinforcement learning," *Found. Trends Mach. Learn.*, vol. 11, nos. 3–4, pp. 219–354, 2018.

[27] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double *Q*-learning," in *Proc. 13th AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.

[28] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2015. [Online]. Available: arXiv:1511.05952.

[29] Z. Wang *et al.*, "Dueling network architectures for deep reinforcement learning," 2015. [Online]. Available: arXiv:1511.06581.

[30] S.-I. Amari, "Natural gradient works efficiently in learning," *Neural Comput.*, vol. 10, no. 2, pp. 251–276, 1998.

[31] S. M. Kakade, "A natural policy gradient," in *Proc. Adv. Neural Inf. Process. Syst.*, 2002, pp. 1531–1538.

[32] J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1889–1897.

[33] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. A. Riedmiller, "Deterministic policy gradient algorithms," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2014, pp. 387–395.

[34] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 229–256, 1992.

[35] V. R. Konda and J. N. Tsitsiklis, "Actor–critic algorithms," in *Proc. Adv. Neural Inf. Process. Syst.*, 2000, pp. 1008–1014.

[36] Z. Wang *et al.*, "Sample efficient actor–critic with experience replay," 2016. [Online]. Available: arXiv:1611.01224.

[37] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.

[38] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor–critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018. [Online]. Available: arXiv:1801.01290.

[39] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," 2015. [Online]. Available: arXiv:1509.02971.

[40] G. Barth-Maron *et al.*, "Distributed distributional deterministic policy gradients," 2018. [Online]. Available: arXiv:1804.08617.

[41] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor–critic methods," 2018. [Online]. Available: arXiv:1802.09477.

[42] R. Lowe, Y. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, "Multi-agent actor–critic for mixed cooperative-competitive environments," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6379–6390.

[43] N. Heess, J. J. Hunt, T. P. Lillicrap, and D. Silver, "Memory-based control with recurrent neural networks," 2015. [Online]. Available: arXiv:1512.04455.

[44] S. Gu, T. Lillicrap, Z. Ghahramani, R. E. Turner, and S. Levine, "*Q*-Prop: Sample-efficient policy gradient with an off-policy critic," 2016. [Online]. Available: arXiv:1611.02247.

[45] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017. [Online]. Available: arXiv:1707.06347.

[46] Y. Wu, E. Mansimov, R. B. Grosse, S. Liao, and J. Ba, "Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5279–5288.

[47] G. Shani, J. Pineau, and R. Kaplow, "A survey of point-based POMDP solvers," *Auton. Agents Multiagent Syst.*, vol. 27, no. 1, pp. 1–51, 2013.

[48] P. Dai, C. H. Lin, Mausam, and D. S. Weld, "POMDP-based control of workflows for crowdsourcing," *Artif. Intell.*, vol. 202, pp. 52–85, Sep. 2013.

[49] D. Wierstra, A. Foerster, J. Peters, and J. Schmidhuber, "Solving deep memory POMDPs with recurrent policy gradients," in *Proc. Int. Conf. Artif. Neural Netw.*, 2007, pp. 697–706.

[50] M. J. Hausknecht and P. Stone, "Deep recurrent *Q*-learning for partially observable MDPs," in *Proc. AAAI Fall Symp. Series*, 2015, pp. 29–37.

[51] G. Wayne *et al.*, "Unsupervised predictive memory in a goal-directed agent," 2018. [Online]. Available: arXiv:1803.10760.

[52] M. Egorov, "Deep reinforcement learning with POMDPs," Stanford University, Rep., 2015. [Online]. Available: http://cs229.stanford.edu/proj2015/363_report.pdf

[53] P. Zhu, X. Li, P. Poupart, and G. Miao, "On improving deep reinforcement learning for POMDPs," 2018. [Online]. Available: arXiv:1804.06309.

[54] J. N. Foerster, Y. M. Assael, N. de Freitas, and S. Whiteson, "Learning to communicate to solve riddles with deep distributed recurrent *Q*-networks," 2016. [Online]. Available: arXiv:1602.02672.

[55] L. Bu, R. Babu, and B. De Schutter, "A comprehensive survey of multiagent reinforcement learning," *IEEE Trans. Syst., Man, Cybern. C, Appl., Rev.*, vol. 38, no. 2, pp. 156–172, Mar. 2008.

[56] J. Foerster *et al.*, "Stabilising experience replay for deep multi-agent reinforcement learning," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, 2017, pp. 1146–1155.

[57] E. Van der Pol and F. A. Oliehoek, "Coordinated deep reinforcement learners for traffic light control," in *Proc. Learn. Inference Multiagent Syst. NIPS*, 2016, pp. 1–8.

[58] J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 2974–2982.

[59] T. Park, N. Abuzainab, and W. Saad, "Learning how to communicate in the Internet of Things: Finite resources and heterogeneity," *IEEE Access*, vol. 4, pp. 7063–7073, 2016.

[60] M. Kwon, J. Lee, and H. Park, "Intelligent IoT connectivity: Deep reinforcement learning approach," *IEEE Sensors J.*, vol. 20, no. 5, pp. 2782–2791, Mar. 2020.

[61] J.-C. Renaud and C.-K. Tham, "Coordinated sensing coverage in sensor networks using distributed reinforcement learning," in *Proc. 14th IEEE Int. Conf. Netw.*, vol. 1, 2006, pp. 1–6.

[62] J. Chen, T. Shu, T. Li, and C. W. de Silva, "Deep reinforced learning tree for spatiotemporal monitoring with mobile robotic wireless sensor networks," *IEEE Trans. Syst., Man, Cybern., Syst.*, early access, Jun. 17, 2019, doi: 10.1109/TSMC.2019.2920390.

[63] Y. Su, X. Lu, Y. Zhao, L. Huang, and X. Du, "Cooperative communications with relay selection based on deep reinforcement learning in wireless sensor networks," *IEEE Sens. J.*, vol. 19, no. 20, pp. 9561–9569, Oct. 2019.

[64] J. Zhu, Y. Song, D. Jiang, and H. Song, "A new deep-*Q*-learning-based transmission scheduling mechanism for the cognitive Internet of Things," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 2375–2385, Aug. 2018.

[65] T. Oda, R. Obukata, M. Ikeda, L. Barolli, and M. Takizawa, "Design and implementation of a simulation system based on deep *Q*-network for mobile actor node control in wireless sensor and actor networks," in *Proc. IEEE 31st Int. Conf. Adv. Inf. Netw. Appl. Workshops (WAINA)*, 2017, pp. 195–200.

[66] G. Künzel, G. P. Cainelli, I. Müller, and C. E. Pereira, "Weight adjustments in a routing algorithm for wireless sensor and actuator networks using *Q*-learning," *IFAC PapersOnLine*, vol. 51, no. 10, pp. 58–63, 2018.

[67] A. S. Leong, A. Ramaswamy, D. E. Quevedo, H. Karl, and L. Shi, "Deep reinforcement learning for wireless sensor scheduling in cyber-physical systems," 2018. [Online]. Available: arXiv:1809.05149.

[68] N. Jiang, Y. Deng, O. Simeone, and A. Nallanathan, "Cooperative deep reinforcement learning for multiple-group NB-IoT networks optimization," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, 2019, pp. 8424–8428.

[69] M. Chafii, F. Bader, and J. Palicot, "Enhancing coverage in narrow band-IoT using machine learning," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, 2018, pp. 1–6.

[70] L. Lei, Y. Kuang, X. S. Shen, K. Yang, J. Qiao, and Z. Zhong, "Optimal reliability in energy harvesting industrial wireless sensor networks," *IEEE Trans. Wireless Commun.*, vol. 15, no. 8, pp. 5399–5413, Aug. 2016.

[71] M. Chu, H. Li, X. Liao, and S. Cui, "Reinforcement learning based multi-access control and battery prediction with energy harvesting in IoT systems," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2009–2020, Apr. 2019.

[72] D. Li, S. Xu, and J. Zhao, "Partially observable double DQN based IoT scheduling for energy harvesting," in *Proc. IEEE Int. Conf. Commun. Workshops (ICC Workshops)*, 2019, pp. 1–6.

[73] C. Qiu, Y. Hu, Y. Chen, and B. Zeng, "Deep deterministic policy gradient (DDPG) based energy harvesting wireless communications," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8577–8588, Oct. 2019.

[74] H. He, H. Shan, A. Huang, Q. Ye, and W. Zhuang, "Reinforcement learning-based computing and transmission scheduling for LTE-U-enabled IoT," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2018, pp. 1–6.

[75] X. Liu, Z. Qin, and Y. Gao, "Resource allocation for edge computing in IoT networks via reinforcement learning," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2019, pp. 1–6.

[76] L. Huang, X. Feng, C. Zhang, L. Qian, and Y. Wu, "Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing," *Digit. Commun. Netw.*, vol. 5, no. 1, pp. 10–17, 2019.

[77] L. Huang, S. Bi, and Y. J. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," 2019. [Online]. Available: arXiv:1808.01977v5.

[78] L. Lei, H. Xu, X. Xiong, K. Zheng, W. Xiang, and X. Wang, "Multi-user resource control with deep reinforcement learning in IoT edge computing," *IEEE Internet Things J.*, vol. 6, no. 6, pp. 10119–10133, Dec. 2019.

[79] X. Qiu, L. Liu, W. Chen, Z. Hong, and Z. Zheng, "Online deep reinforcement learning for computation offloading in blockchain-empowered mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 68, no. 8, pp. 8050–8062, Aug. 2019.

[80] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4005–4018, Jun. 2019.

[81] M. Min, L. Xiao, Y. Chen, P. Cheng, D. Wu, and W. Zhuang, "Learning-based computation offloading for IoT devices with energy harvesting," *IEEE Trans. Veh. Technol.*, vol. 68, no. 2, pp. 1930–1941, Feb. 2019.

[82] J. Chen, S. Chen, Q. Wang, B. Cao, G. Feng, and J. Hu, "iRAF: A deep reinforcement learning approach for collaborative mobile edge computing IoT networks," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 7011–7024, Aug. 2019.

[83] J. Wang, L. Zhao, J. Liu, and N. Kato, "Smart resource allocation for mobile edge computing: A deep reinforcement learning approach," *IEEE Trans. Emerg. Topics Comput.*, early access, Mar. 4, 2019, doi: 10.1109/TETC.2019.2902661.

[84] J. Ren, H. Wang, T. Hou, S. Zheng, and C. Tang, "Federated learning-based computation offloading optimization in edge computing-supported Internet of Things," *IEEE Access*, vol. 7, pp. 69194–69201, 2019.

[85] N. Cheng *et al.*, "Space/aerial-assisted computing offloading for IoT applications: A learning-based approach," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 5, pp. 1117–1129, May 2019.

[86] H. Zhu, Y. Cao, X. Wei, W. Wang, T. Jiang, and S. Jin, "Caching transient data for Internet of Things: A deep reinforcement learning approach," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2074–2083, Apr. 2019.

[87] Y. Wei, F. R. Yu, M. Song, and Z. Han, "Joint optimization of caching, computing, and radio resources for fog-enabled IoT using natural actor–critic deep reinforcement learning," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2061–2073, Apr. 2019.

[88] H. Sasaki, T. Horiuchi, and S. Kato, "A study on vision-based mobile robot learning by deep *Q*-network," in *Proc. 56th Annu. Conf. Soc. Instrum. Control Eng. Japan (SICE)*, 2017, pp. 799–804.

[89] J. Xin, H. Zhao, D. Liu, and M. Li, "Application of deep reinforcement learning in mobile robot path planning," in *Proc. Chin. Autom. Congr. (CAC)*, 2017, pp. 7112–7116.

[90] M. Saravanan, P. Kumar, and A. Sharma, "IoT enabled indoor autonomous mobile robot using CNN and *Q*-learning," in *Proc. IEEE Int. Conf. Ind. Artif. Intell. Commun. Technol. (IAICT)*, 2019, pp. 7–13.

[91] T. Yan, Y. Zhang, and B. Wang, "Path planning for mobile robot's continuous action space based on deep reinforcement learning," in *Proc. Int. Conf. Big Data Artif. Intell. (BDAI)*, 2018, pp. 42–46.

[92] T. Tongloy, S. Chuwongin, K. Jaksukam, C. Chousangsuntorn, and S. Boonsang, "Asynchronous deep reinforcement learning for the mobile robot navigation with supervised auxiliary tasks," in *Proc. 2nd Int. Conf. Robot. Autom. Eng. (ICRAE)*, 2017, pp. 68–72.

[93] Z. Yang, K. Merrick, L. Jin, and H. A. Abbass, "Hierarchical deep reinforcement learning for continuous action control," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 11, pp. 5174–5184, Nov. 2018.

[94] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2017, pp. 3389–3396.

[95] D. Kalashnikov *et al.*, "Scalable deep reinforcement learning for vision-based robotic manipulation," in *Proc. Conf. Robot Learn.*, 2018, pp. 651–673.

[96] Y. Tsurumine, Y. Cui, E. Uchibe, and T. Matsubara, "Deep reinforcement learning with smooth policy update: Application to robotic cloth manipulation," *Robot. Auton. Syst.*, vol. 112, pp. 72–83, Feb. 2019.

[97] T. Yasuda and K. Ohkura, "Collective behavior acquisition of real robotic swarms using deep reinforcement learning," in *Proc. 2nd IEEE Int. Conf. Robot. Comput. (IRC)*, 2018, pp. 179–180.

[98] X. Sun, T. Mao, J. D. Kralik, and L. E. Ray, "Cooperative multi-robot reinforcement learning: A framework in hybrid state space," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2009, pp. 1190–1196.

[99] G. Sartoretti, Y. Wu, W. Paivine, T. S. Kumar, S. Koenig, and H. Choset, "Distributed reinforcement learning for multi-robot decentralized collective construction," in *Distributed Autonomous Robotic Systems*, Springer, 2019, pp. 35–49.

[100] P. Long, T. Fanl, X. Liao, W. Liu, H. Zhang, and J. Pan, "Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2018, pp. 6252–6259.

[101] M. J. Matarić, "Reinforcement learning in the multi-robot domain," in *Robot Colonies*, Springer, 1997, pp. 73–83.

[102] B. Liu, L. Wang, M. Liu, and C. Xu, "Lifelong federated reinforcement learning: A learning architecture for navigation in cloud robotic systems," 2019. [Online]. Available: arXiv:1901.06455.

[103] H. Liu, S. Liu, and K. Zheng, "A reinforcement learning-based resource allocation scheme for cloud robotics," *IEEE Access*, vol. 6, pp. 17215–17222, 2018.

[104] E. Yang and D. Gu, "A survey on multiagent reinforcement learning towards multi-robot systems," in *Proc. IEEE Conf. Comput. Intell. Games (CIG)*, 2005, pp. 292–299.

[105] O. Saha and P. Dasgupta, "A comprehensive survey of recent trends in cloud robotics architectures and applications," *Robotics*, vol. 7, no. 3, p. 47, 2018.

[106] A. Yu, R. Palefsky-Smith, and R. Bedi, "Deep reinforcement learning for simulated autonomous vehicle control," Stanford University, Rep., 2016. [Online]. Available: http://cs231n.stanford.edu/reports/2016/pdfs/112_Report.pdf

[107] M. Vitelli and A. Nayebi, "CARMA: A deep reinforcement learning approach to autonomous driving," Stanford University, Rep., 2016. [Online]. Available: https://web.stanford.edu/∼anayebi/projects/CS_239_Final_Project_Writeup.pdf

[108] B. Mirchevska, C. Pek, M. Werling, M. Althoff, and J. Boedecker, "High-level decision making for safe and reasonable autonomous lane changing using reinforcement learning," in *Proc. 21st Int. Conf. Intell. Transp. Syst. (ITSC)*, 2018, pp. 2156–2162.

[109] C. Wu, A. Kreidieh, K. Parvate, E. Vinitsky, and A. M. Bayen, "FLOW: Architecture and benchmarking for reinforcement learning in traffic control," 2017. [Online]. Available: arXiv:1710.05465.

[110] H. D. Gamage and J. B. Lee, "Reinforcement learning based driving speed control for two vehicle scenario," in *Proc. 39th Aust. Transp. Res. Forum (ATRF)*, Auckland, New Zealand, 2017, pp. 1–15.

[111] C. You, J. Lu, D. Filev, and P. Tsiotras, "Highway traffic modeling and decision making for autonomous vehicle using reinforcement learning," in *Proc. IEEE Intell. Veh. Symp. (IV)*, 2018, pp. 1227–1232.

[112] M. K. Pal, R. Bhati, A. Sharma, S. K. Kaul, S. Anand, and P. Sujit, "A reinforcement learning approach to jointly adapt vehicular communications and planning for optimized driving," in *Proc. IEEE 21st Int. Conf. Intell. Transp. Syst. (ITSC)*, 2018, pp. 3287–3293.

[113] P. Wang and C.-Y. Chan, "Formulation of deep reinforcement learning architecture toward autonomous driving for on-ramp merge," in *Proc. IEEE 20th Int. Conf. Intell. Transp. Syst. (ITSC)*, 2017, pp. 1–6.

[114] Q. Wang and C. Phillips, "Cooperative collision avoidance for multi-vehicle systems using reinforcement learning," in *Proc. IEEE 18th Int. Conf. Methods Models Autom. Robot. (MMAR)*, 2013, pp. 98–102.

[115] M. A. Khamis and W. Gomaa, "Adaptive multi-objective reinforcement learning with hybrid exploration for traffic signal control based on cooperative multi-agent framework," *Eng. Appl. Artif. Intell.*, vol. 29, pp. 134–151, Mar. 2014.

[116] H. Ye and G. Y. Li, "Deep reinforcement learning based distributed resource allocation for V2V broadcasting," in *Proc. IEEE 14th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, 2018, pp. 440–445.

[117] U. Challita, W. Saad, and C. Bettstetter, "Interference management for cellular-connected UAVs: A deep reinforcement learning approach," *IEEE Trans. Wireless Commun.*, vol. 18, no. 4, pp. 2125–2140, Apr. 2019.

[118] Y. He, N. Zhao, and H. Yin, "Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach," *IEEE Trans. Veh. Technol.*, vol. 67, no. 1, pp. 44–55, Jan. 2018.

[119] R. F. Atallah, C. M. Assi, and M. J. Khabbaz, "Scheduling the operation of a connected vehicular network using deep reinforcement learning," *IEEE Trans. Intell. Transp. Syst.*, vol. 20, no. 5, pp. 1669–1682, May 2019.

[120] Q. Qi *et al.*, "Knowledge-driven service offloading decision for vehicular edge computing: A deep reinforcement learning approach," *IEEE Trans. Veh. Technol.*, vol. 68, no. 5, pp. 4192–4203, May 2019.

[121] Q. Qi and Z. Ma, "Vehicular edge computing via deep reinforcement learning," 2018. [Online]. Available: arXiv:1901.04290.

[122] K. Zheng, H. Meng, P. Chatzimisios, L. Lei, and X. Shen, "An SMDP-based resource allocation in vehicular cloud computing systems," *IEEE Trans. Ind. Electron.*, vol. 62, no. 12, pp. 7920–7928, Dec. 2015.

[123] Y. Liu, H. Yu, S. Xie, and Y. Zhang, "Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 11, pp. 11158–11168, Nov. 2019.

[124] L. Hou, L. Lei, K. Zheng, and X. Wang, "A Q-learning based proactive caching strategy for non-safety related services in vehicular networks," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4512–4520, Jun. 2019.

[125] V. Talpaert *et al.*, "Exploring applications of deep reinforcement learning for real-world autonomous driving systems," 2019. [Online]. Available: arXiv:1901.01536.

[126] A. Kendall *et al.*, "Learning to drive in a day," 2018. [Online]. Available: arXiv:1807.00412.

[127] H. Xu, Y. Gao, F. Yu, and T. Darrell, "End-to-end learning of driving models from large-scale video datasets," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 2174–2182.

[128] K. Zheng, L. Hou, H. Meng, Q. Zheng, N. Lu, and L. Lei, "Soft-defined heterogeneous vehicular network: Architecture and challenges," *IEEE Netw.*, vol. 30, no. 4, pp. 72–80, Jul./Aug. 2016.

[129] L. Liang, H. Ye, and G. Y. Li, "Toward intelligent vehicular networks: A machine learning framework," *IEEE Internet Things J.*, vol. 6, no. 1, pp. 124–135, Feb. 2019.

[130] H. Ye, L. Liang, G. Y. Li, J. Kim, L. Lu, and M. Wu, "Machine learning for vehicular networks," 2017. [Online]. Available: arXiv:1712.07143.

[131] A. Mehmood, S. H. Ahmed, and M. Sarkar, "Cyber-physical systems in vehicular communications," in *Handbook of Research on Advanced Trends in Microwave and Communication Engineering*. Hershey, PA, USA: IGI Global, 2017, pp. 477–497.

[132] Y. Xiao and C. Zhu, "Vehicular fog computing: Vision and challenges," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops (PerCom Workshops)*, 2017, pp. 6–9.

[133] J. C. Nobre *et al.*, "Vehicular software-defined networking and fog computing: Integration and design principles," *Ad Hoc Netw.*, vol. 82, pp. 172–181, Jan. 2019.

[134] Z. Ning, J. Huang, and X. Wang, "Vehicular fog computing: Enabling real-time traffic management for smart cities," *IEEE Wireless Commun.*, vol. 26, no. 1, pp. 87–93, Feb. 2019.

[135] V. François-Lavet, D. Taralla, D. Ernst, and R. Fonteneau, "Deep reinforcement learning solutions for energy microgrids management," in *Proc. Eur. Workshop Reinforcement Learn. (EWRL)*, 2016, pp. 1–7.

[136] X. Qiu, T. A. Nguyen, and M. L. Crow, "Heterogeneous energy storage optimization for microgrids," *IEEE Trans. Smart Grid*, vol. 7, no. 3, pp. 1453–1461, May 2016.

[137] B. Mbuwir, F. Ruelens, F. Spiessens, and G. Deconinck, "Reinforcement learning-based battery energy management in a solar microgrid," *Energy Open*, vol. 2, no. 4, p. 36, 2017.

[138] P. Zeng, H. Li, H. He, and S. Li, "Dynamic energy management of a microgrid using approximate dynamic programming and deep recurrent neural network learning," *IEEE Trans. Smart Grid*, vol. 10, no. 4, pp. 4435–4445, Jul. 2019.

[139] G. K. Venayagamoorthy, R. K. Sharma, P. K. Gautam, and A. Ahmadi, "Dynamic energy management system for a smart microgrid," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 8, pp. 1643–1656, Aug. 2016.

[140] Y. Ji, J. Wang, J. Xu, X. Fang, and H. Zhang, "Real-time energy management of a microgrid using deep reinforcement learning," *Energies*, vol. 12, no. 12, p. 2291, 2019.

[141] E. Mocanu *et al.*, "On-line building energy optimization using deep reinforcement learning," *IEEE Trans. Smart Grid*, vol. 10, no. 4, pp. 3698–3708, Jul. 2019.

[142] Z. Wen, D. O'Neill, and H. Maei, "Optimal demand response using device-based reinforcement learning," *IEEE Trans. Smart Grid*, vol. 6, no. 5, pp. 2312–2324, Sep. 2015.

[143] D. O'Neill, M. Levorato, A. Goldsmith, and U. Mitra, "Residential demand response using reinforcement learning," in *Proc. 1st IEEE Int. Conf. Smart Grid Commun.*, 2010, pp. 409–414.

[144] B. J. Claessens, P. Vrancx, and F. Ruelens, "Convolutional neural networks for automatic state-time feature extraction in reinforcement learning applied to residential load control," 2016. [Online]. Available: arXiv:1604.08382.

[145] F. Ruelens, B. J. Claessens, S. Vandael, S. Iacovella, P. Vingerhoets, and R. Belmans, "Demand response of a heterogeneous cluster of electric water heaters using batch reinforcement learning," in *Proc. IEEE Power Syst. Comput. Conf.*, 2014, pp. 1–7.

[146] F. Ruelens, B. J. Claessens, S. Quaiyum, B. De Schutter, R. Babuška, and R. Belmans, "Reinforcement learning applied to an electric water heater: From theory to practice," *IEEE Trans. Smart Grid*, vol. 9, no. 4, pp. 3792–3800, Jul. 2018.

[147] R. Lu, S. H. Hong, and X. Zhang, "A dynamic pricing demand response algorithm for smart grid: Reinforcement learning approach," *Appl. Energy*, vol. 220, pp. 220–230, Jun. 2018.

[148] R. Lu and S. H. Hong, "Incentive-based demand response for smart grid with reinforcement learning and deep neural network," *Appl. Energy*, vol. 236, pp. 937–949, Feb. 2019.

[149] Y. Lim and H.-M. Kim, "Strategic bidding using reinforcement learning for load shedding in microgrids," *Comput. Elect. Eng.*, vol. 40, no. 5, pp. 1439–1446, 2014.

[150] X. Xiao, C. Dai, Y. Li, C. Zhou, and L. Xiao, "Energy trading game for microgrids using reinforcement learning," in *Proc. Int. Conf. Game Theory Netw.*, 2017, pp. 131–140.

[151] L. Xiao, X. Xiao, C. Dai, M. Pengy, L. Wang, and H. V. Poor, "Reinforcement learning-based energy trading for microgrids," 2018. [Online]. Available: arXiv:1801.06285.

[152] B.-G. Kim, Y. Zhang, M. Van Der Schaar, and J.-W. Lee, "Dynamic pricing for smart grid with reinforcement learning," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, 2014, pp. 640–645.

[153] B. G. Kim, Y. Zhang, M. Van Der Schaar, and J.-W. Lee, "Dynamic pricing and energy consumption scheduling with reinforcement learning," *IEEE Trans. Smart Grid*, vol. 7, no. 5, pp. 2187–2198, Sep. 2016.

[154] P. P. Reddy and M. M. Veloso, "Strategy learning for autonomous agents in smart grid markets," in *Proc. 22nd Int. Joint Conf. Artif. Intell.*, 2011, pp. 1446–1451.

[155] E. Foruzan, L.-K. Soh, and S. Asgarpoor, "Reinforcement learning approach for optimal distributed energy management in a microgrid," *IEEE Trans. Power Syst.*, vol. 33, no. 5, pp. 5749–5758, Sep. 2018.

[156] G. Bedi, G. K. Venayagamoorthy, R. Singh, R. R. Brooks, and K. Wang, "Review of Internet of Things (IoT) in electric power and energy systems," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 847–870, Apr. 2018.

[157] M. Rana, L. Li, and S. W. Su, "Distributed state estimation over unreliable communication networks with an application to smart grids," *IEEE Trans. Green Commun. Netw.*, vol. 1, no. 1, pp. 89–96, Mar. 2017.

[158] M. Igl, L. Zintgraf, T. A. Le, F. Wood, and S. Whiteson, "Deep variational reinforcement learning for POMDPs," 2018. [Online]. Available: arXiv:1806.02426.

[159] K. V. Katsikopoulos and S. E. Engelbrecht, "Markov decision processes with delays and asynchronous cost collection," *IEEE Trans. Autom. Control*, vol. 48, no. 4, pp. 568–574, Apr. 2003.
[160] T. J. Walsh, A. Nouri, L. Li, and M. L. Littman, "Learning and planning in environments with delayed feedback," *Auton. Agent Multiagent*, vol. 18, no. 1, p. 83, 2009.
[161] E. Schuitema, L. Buşoniu, R. Babuška, and P. Jonker, "Control delay in reinforcement learning for real-time dynamic systems: A memoryless approach," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2010, pp. 3226–3231.
[162] G. Lample and D. S. Chaplot, "Playing FPS games with deep reinforcement learning," in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 2140–2146.
[163] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[164] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient flow in recurrent nets: The difficulty of learning long-term dependencies," in *A Field Guide to Dynamical Recurrent Neural Networks*, IEEE Press, 2001, pp. 237–243.

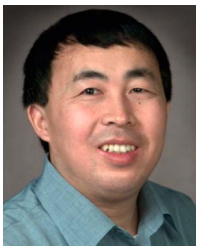
Lei Lei (Senior Member, IEEE) received the B.S. and Ph.D. degrees in telecommunications engineering from the Beijing University of Posts and Telecommunications, Beijing, China, in 2001 and 2006, respectively. She is currently an Associate Professor with the College of Engineering and Physical Sciences, University of Guelph, Canada. Her research interests mainly lie in machine learning/deep reinforcement learning, Internet of Things/Internet of Vehicles, mobile edge computing, and smart grid.

Yue Tan (Graduate Student Member, IEEE) received the B.S. degree from the Beijing University of Posts and Telecommunications, Beijing, China, in 2017, where she is currently pursuing the master's degree. Her current research interests include deep reinforcement learning and its applications in Internet of Things and smart grid.

Kan Zheng (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees from the Beijing University of Posts and Telecommunications, China, in 1996, 2000, and 2005, respectively, where he is currently a Full Professor. He has rich experience in research and standardization of new emerging technologies. He has authored over 200 journal articles and conference papers in the field of wireless communications, vehicular networks, IoT, and security. He holds editorial board positions with several journals. He has organized several special issues in the journals, including the IEEE COMMUNICATIONS SURVEYS AND TUTORIALS, IEEE COMMUNICATIONS MAGAZINE, and IEEE SYSTEMS JOURNAL. He has also served in the organizing/TPC committees for more than ten conferences.

Shiwen Liu received the B.S. degree from the Beijing University of Posts and Telecommunications, China, in 2017, where she is currently pursuing the master's degree with the Intelligent Computing and Communication Laboratory, Key Laboratory of Universal Wireless Communications, Ministry of Education. Her research interests include wireless communications, cloud robotics, and Internet of Vehicle.

Kuan Zhang (Senior Member, IEEE) received the B.Sc. degree in communication engineering and the M.Sc. degree in computer applied technology from Northeastern University, Shenyang, China, in 2009 and 2011, respectively, and the Ph.D. degree in electrical and computer engineering from the University of Waterloo, Waterloo, ON, Canada, in 2016. He has been an Assistant Professor with the Department of Electrical and Computer Engineering, University of Nebraska–Lincoln, Omaha, NE, USA, since 2017. He was also a Postdoctoral Fellow with the Broadband Communications Research Group, Department of Electrical and Computer Engineering, University of Waterloo, from 2016 to 2017. His current research interests include security and privacy for mobile social networks, e-healthcare systems, cloud/edge computing, and cyber physical systems. He was the recipient of Best Paper Award in IEEE WCNC 2013, Securecomm 2016, and BigDataSE 2019.



Xuemin (Sherman) Shen (Fellow, IEEE) received the B.Sc. degree in electrical engineering from Dalian Maritime University, China, in 1982, and the M.Sc. and Ph.D. degrees in electrical engineering from Rutgers University, New Jersey, NJ, USA, in 1987 and 1990, respectively. He is a Professor and a University Research Chair with the Department of Electrical and Computer Engineering, University of Waterloo, Canada. He was the Associate Chair for Graduate Studies from 2004 to 2008. He has coauthored/edited six books, and has published more than 600 papers and book chapters in wireless communications and networks, control and filtering. His research focuses on resource management in interconnected wireless/wired networks, wireless network security, social networks, smart grid, and vehicular ad hoc and sensor networks. He received the Excellent Graduate Supervision Award in 2006, and the Outstanding Performance Award in 2004, 2007, and 2010 from the University of Waterloo, the Premier's Research Excellence Award in 2003 from the Province of Ontario, Canada, and the Distinguished Performance Award in 2002 and 2007 from the Faculty of Engineering, University of Waterloo. He is an elected member of IEEE ComSoc Board of Governor, and the Chair of Distinguished Lecturers Selection Committee. He served as the Technical Program Committee Chair/Co-Chair for IEEE Infocom'14, IEEE VTC'10 Fall, the Symposia Chair for IEEE ICC'10, the Tutorial Chair for IEEE VTC'11 Spring and IEEE ICC'08, the Technical Program Committee Chair for IEEE Globecom'07, the General Co-Chair for Chinacom'07 and QShine'06, the Chair for IEEE Communications Society Technical Committee on Wireless Communications, and P2P Communications and Networking. He also serves/served as the Editor-in-Chief for IEEE NETWORK, *Peer-to-Peer Networking and Applications*, and *IET Communications*; a Founding Area Editor for the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS; an Associate Editor for the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, *Computer Networks*, and *ACM/Wireless Networks*; and the Guest Editor for the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, IEEE WIRELESS COMMUNICATIONS, *IEEE Communications Magazine*, and *ACM Mobile Networks and Applications*. He is a Registered Professional Engineer of Ontario, Canada, an fellow of the Engineering Institute of Canada, the Canadian Academy of Engineering, and a Distinguished Lecturer of IEEE Vehicular Technology Society and Communications Society.