applied sciences

MDPI

Article

Adaptive Real-Time Offloading Decision-Making for Mobile Edges: Deep Reinforcement Learning Framework and Simulation Results

Soohyun Park [1], Dohyun Kwon [1], Joongheon Kim [2,*], Youn Kyu Lee [3,*] and Sungrae Cho [1,*]

[1] School of Computer Science and Engineering, Chung-Ang University, Seoul 06974, Korea; shpark.cau@gmail.com (S.P.); kdh1102@cau.ac.kr (D.K.)
[2] School of Electrical Engineering, Korea University, Seoul 02841, Korea
[3] Multimedia Processing Lab., Samsung Advanced Institute of Technology, Suwon 16677, Korea
* Correspondence: joongheon@korea.ac.kr (J.K.); younkyu.lee@samsung.com (Y.K.L.); srcho@cau.ac.kr (S.C.); Tel.: +82-2-3290-3223 (J.K.)

Received: 29 January 2020; Accepted: 27 February 2020; Published: 1 March 2020

Abstract: This paper proposes a novel dynamic offloading decision method which is inspired by deep reinforcement learning (DRL). In order to realize real-time communications in mobile edge computing systems, an efficient task offloading algorithm is required. When the decision of actions (offloading enabled, i.e., computing in clouds or offloading disabled, i.e., computing in local edges) is made by the proposed DRL-based dynamic algorithm in each unit time, it is required to consider real-time/seamless data transmission and energy-efficiency in mobile edge devices. Therefore, our proposed dynamic offloading decision algorithm is designed for the joint optimization of delay and energy-efficient communications based on DRL framework. According to the performance evaluation via data-intensive simulations, this paper verifies that the proposed dynamic algorithm achieves desired performance.

Keywords: mobile edge computing; offloading; real-time; deep reinforcement learning; deep Q-network

1. Introduction

According to the fact that the 5G era has been realized based on networking technology innovation, many new computing and communications paradigms are introduced such as millimeter-wave communications, hyper-dense networks, and device-to-device proximal networking [1–3]. Among them, mobile edge computing (MEC) is one of the major technologies for realizing data/computing distribution in order to improve cellular network performance [4]. Based on the benefits of MEC technologies such as data rate improvement and quality enhancement, mobile cellular users can enjoy high quality, seamless, and real-time communication networking services.

Together with the MEC technologies, many networked components are also of interest such as cloud servers and connected devices/vehicles. As the number of connected devices/vehicles increases, the amount of data transmitted to the MEC is also rapidly increasing. This obviously introduces serious network limitations such as data processing performance limits, storage capacity limits, and the battery use of terminal devices. Under the consideration of these limitations and problems, the use of cloud computing server is more efficient to deal with big data (gathered from connected vehicles/devices via MEC edges) than the local computing on terminals. On the other hand, for any cases where the data cannot be handled in the cloud servers due to delay requirements and security reasons, MEC devices should be able to handle or process the data from the connected vehicles/devices. Therefore, we can observe the trade-off between cloud computing servers and MEC servers (local edge computing servers). In summary, cloud servers have more power and centralized computing benefits

comparing to MEC servers, whereas, it may have limitations when we have delay requirements and security requirements/regulations. Figure 1 shows the combined architecture of local edge computing and cloud computing. As the transmission from connected vehicles/devices (i.e., data upload and download) delay goes down, it is much more suitable for real-time processing. It enables real-time data processing and transmission using high-quality data in real-time broadcasting or video streaming. Therefore, it is essential to design an algorithm which can handle the data transmission scheduling (i.e., offloading decision-making, i.e., centralized vs. local computing). Note that the data from connected vehicles/devices will be handled in cloud computing servers when offloading decision is made, whereas the MEC servers will handle the data when the offloading is not preferred.

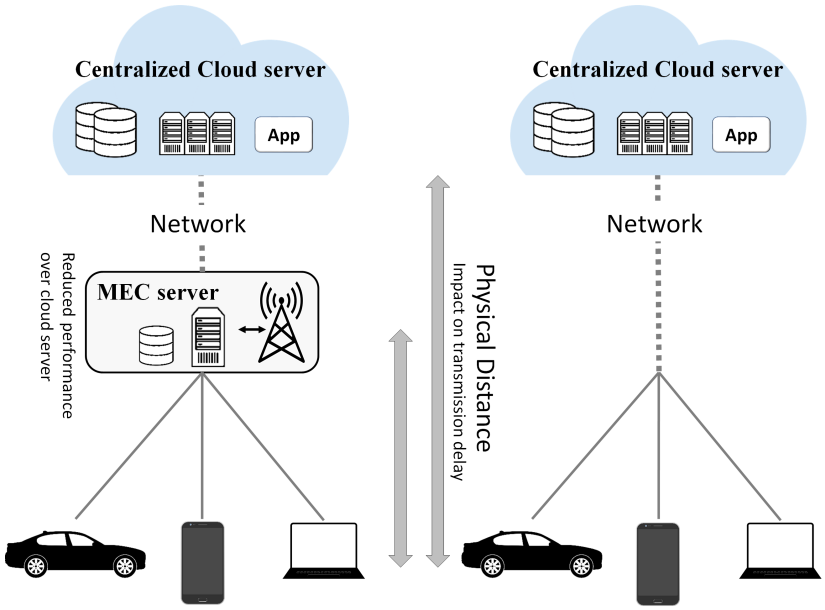


Figure 1. Edge computing and cloud computing architecture.

In order to handle this problem, many research contributions nowadays have been considering deep learning and machine learning based methodologies. Among them, this paper considers reinforcement learning based approaches because this given problem is for stochastic sequential *offloading* decision-making. Among a lot of deep reinforcement learning (DRL) methodologies such as Q-learning, Markov decision process (MDP) [5], deep Q-network (DQN), and deep deterministic policy gradient (DDPG) [6,7], this paper designs a sequential offloading decision-making algorithm based on DQN. The reason why this paper considers DQN is that it is the function approximation of Q-learning using deep neural network (DNN) in order to take care of large-scale problem setting. The proposed sequential offloading decision-making algorithm inspired by DQN aims at the maximization the cumulative summation of the rewards for sequential offloading actions (centralized vs. local computing). Therefore, it is obvious that the reinforcement learning is a way to proceed optimal learning even in a dynamic environment under uncertainty. The reinforcement learning is applied to a variety of environments such as artificial intelligence robotics development, automatic systems, and self-driving technology development. This paper utilizes this DQN technology for offloading decision-making (i.e., processing the transmitted data at centralized computing, i.e., *cloud* or local computing, i.e., *MEC*) for real-time seamless high quality data communications under uncertainty conditions, e.g., channel dynamics. The reason why DQN is selected among the given Q-learning, MDP, DQN, and DDPG is as follows. The Q-learning and MDP are not related to DNN based function approximation; and their solutions can be computed using dynamic programming. Therefore, the computational complexity of Q-learning and MDP solution computation can be very high when the search space becomes large. The DDPG based formulation is for continuous action domains.

In this paper, our formulation is for offloading decision-making whether the offloading should be conducted or not. Thus, our action space is discrete (0 or 1), i.e., continuous domain DDPG is not suitable; and finally DQN is the best solution among given options.

In mobile edge computing research, several algorithms were proposed in order to optimize their own objectives [8–12]. However, none of them are considering offloading decision for determining whether it has to conduct offloading or not. The proposed algorithm in this paper is the improvement of our previous contributions in [13]. Comparing to our previous work in [13], the proposed algorithm in this paper includes more detailed mathematical formulations and descriptions. Furthermore, various data-intensive performance evaluation results are included in this paper.

These kinds of DRL-based algorithms including our proposed DQN-based method can be widely used and are practical due to the function approximation nature of DNN. With DQN, the states and actions are used as input and output values in DNN training. Once the training is done, we can simply infer our approximated optimal actions by putting current states into the trained DNN. Thus, the computation itself during the inference is light-weight (even though training itself is not) and thus using a DQN-based method is realistic in real-time decision-making systems, as also discussed in [14].

The rest of this paper is constituted as follows. Section 2 introduces preliminaries, i.e., deep learning foundations and related work, i.e., reinforcement learning algorithms for mobile edge computing. Section 3 proposes our system model. Section 4 describes the details of our algorithm with explanation about DQN. In Section 5, we describe the details of simulations and results. Finally, Section 6 concludes this paper and presents future research directions.

2. Background and Related Work

2.1. Deep Learning and Deep Reinforcement Learning Foundations

Nowadays, there are huge interests on deep learning algorithms and applications by industry and academia. The deep learning is essentially for conducting artificial intelligence (AI) tasks using neural network framework. The neural network is for enabling nonlinear computation (so called activation function) in terms of its own AI tasks. If the neural network has more and more hidden layers, the nonlinearity increases. Previously, having many hidden layers was not possible due to a gradient vanishing problem (i.e., the problem where the parameters in hidden layers are not trained and converge to zero). However, the problem was solved by utilizing other types of activation functions, e.g., rectified linear unit (ReLU). Finally, the neural network starts to have many hidden layers; thus, the nonlinearity increases a lot. Eventually, the performance is dramatically increased, even better than human beings in some areas. This neural network is called deep learning because it has deep many hidden layers. Once the neural network is trained with many data, the parameters within the neural network are trained. Thus, we can move to the inference procedure which takes inputs, then the input will be calculated with the trained parameters, and then the outputs can be derived.

Among many deep learning frameworks, if the inputs and outputs of neural network are states and their corresponding actions, respectively, it means we can train the neural network with states and their corresponding actions. Once the training is done, the neural network is now capable of deriving approximated actions depending on input states. Semantically, it means that the neural network can sequentially derive approximated optimal actions in current input states. Thus, this neural network is capable of sequential decision-making for time-series states inputs. This is deep learning based reinforcement learning, and thus it is called deep reinforcement learning, which is mainly discussed in this paper.

2.2. Related Work: Reinforcement Learning for Mobile Edge Computing

There are remarkable research results that are based on reinforcement learning techniques for sequential stochastic decision-making in various computing research domains. For the application

of deep reinforcement learning to mobile edge computing, the research contributions in [8–11] had been discussed about the optimization for their own objective functions. Even if they considered many criteria for the offloading, there are not contributions that aim at the optimal sequential decision-making for offloading decisions, i.e., whether it has to conduct offloading (i.e., centralized computing) or not (i.e., local edge computing). For the design and implementation of intelligent and efficient systems, Feng et al. proposed a hybrid intelligent control mechanism which combines high-level time PetriNet and reinforcement learning frameworks [15]. The control system of the model is mathematically designed by hybrid time PetriNet and additional methods are utilized based on Q-learning, which is a special type of deep reinforcement learning for minimizing transitions delay time. Even though the paper is successful for delay minimization, the other important factors, e.g., energy consumption optimization and offloading, are not discussed in this paper. Furthermore, Bhagat et al. designed a self-sufficient intelligent agent that can be learned based on the information collected from the agent's environment by combining DRL algorithms and soft bio-inspired structures [16]. They also presented several examples in various real-world scenarios. Furthermore, the paper described various research results that conduct DRL methods in soft robotics research domains. Even though the algorithm in this paper improves the performance by utilizing bio-inspired methodologies, it is not discussing about communication-related performance improvements, e.g., delay and energy optimization. In autonomous driving research, Zhang et al. improved the instability seen by Q-learning which is using a double deep Q-learning network as an automatic stochastic sequential decision-making approach for vehicle controls [17]. This paper is novel and conducts efficient tasks in vehicle controls; the algorithm is not scalable for conventional mobile edge computing environment because the parameter setting and formulations are all based on application-specific vehicle controls. Xu et al. designed a sequential intelligent decision-making scheme for highway autonomous driving [18]. This decision-making policy is learned via Q-learning, which is one of the well-known reinforcement learning algorithms based on the series of simulated driving scenarios data. In addition, there are a number of control-related studies for safe autonomous driving in highway environments [19,20]. Shin et al. proposed a novel inverse-reinforcement and imitation learning algorithm, which is one of major reinforcement learning algorithms nowadays, for autonomous driving control under the benefits of augmented random search [21,22]. In addition, there exists RL applications to game playing. Silver et al. designed and implemented AlphaGo-Zero, which evaluates the location decision using adaptive tree search techniques and determines the movement using trained deep learning based on reinforcement learning [23]. The AlphaGo Zero conducted the learning procedures through its own self without a guide or knowledge of human data and game rules, and it improved the performance of the tree search techniques. Finally, the AlphaGo Zero has achieved more than 100 wins against AlphaGo, which won against world-champion human Go players. The algorithms in [18–23] conduct engineering for their specific applications well such as autonomous driving and game AI. Therefore, they are all independent of the communications and networks specific key consideration factors such as energy and delay optimization. Therefore, none of our related research literature papers are directly associated with ours. If we directly utilize their algorithms to our mobile edge computing systems without any modifications, the algorithm definitely presents poor performances due to the lack of mathematical considerations in terms of energy and delay optimization.

3. System Model

3.1. Reference System Model and Formulation

In this scenario, terminal devices (i.e., CCTV camera, drones equipped camera, and vehicle black-box) are wirelessly connected to an edge server which is located nearby as shown in Figure 2. When the terminal device collects video continuously/seamlessly, there exists the limits of battery and storage capacity depending on the size of devices. Based on this limitation, it is not possible to store or compute all of the data on the device itself in any time. Furthermore, even if all data can be processed, a delay which is caused by the processing limit of the device occurs. This can make serious problems in environments where real-time communications is essentially necessary, e.g., mission-critical communications. For this reason, it is necessary to use virtual servers that have better performance than terminal devices. Moreover, using the edge server can be a way to overcome these problems related to the limitation of the terminal devices. However, using edge computing technique (i.e., data offloading) is always not the best solution. We have to consider the delay time of wireless data transmission between the terminal device and the edge server.

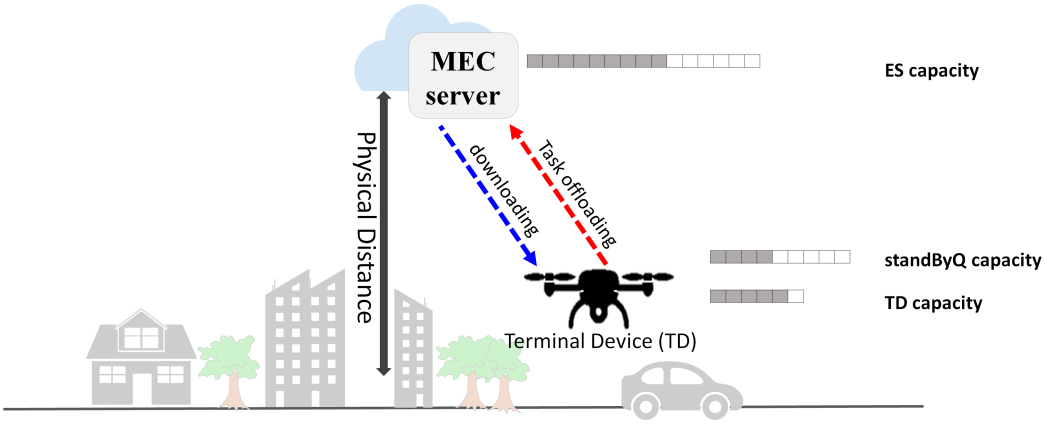


Figure 2. Reference system model.

For this reason, we propose a novel offloading decision algorithm that minimizes the combination of energy consumption and delay at each terminal device. The algorithm we proposed in this paper is based on DQN, which is one of reinforcement learning algorithms that maximizes cumulative summation of rewards. Using this DQN-based algorithm, each terminal device can decide whether it will do offload or not. For this purpose, the corresponding actions and rewards values in DQN-based formulation are as follows:

- *State* $S \triangleq$ {TD capacity, StandBy Q capacity, ES capacity}
- *Action* $A \triangleq \{0, 1\}$
- *Reward* $R_{combi} \triangleq \beta \times (\gamma \times R_{delay} + (1-\gamma) \times R_{energy}) + (1-\beta) \times R_{capacity}$ (Refer to (6), in Section 4)

where TD capacity, StandBy Q capacity, and ES capacity stand for the capacity of terminal device, stand-by Q capacity, and the capacity of edge server, respectively. Here, the standByQ means the waiting buffer of each terminal device. The data/information that is decided to be offloaded is stored the standByQ. Next, each action means whether it needs to be offloaded or not. Here, the value 1 means offloading happens at time step t, whereas the value 0 means processing the data locally at time step t (i.e., offloading does not happen). The details of the proposed algorithm are explained in Section 4.

3.2. Offloading Decision: Trade-Off between Energy Consumption and Delay

In mobile edge computing (MEC) scenarios, each terminal device has limitations in terms of storage capacity, computing performance, and battery lifetime. Each terminal can offload its data

(or task) to its associated nearby edge server and also does not need to operated by itself. The MEC can (i) save the energy of each device, (ii) prevent data loss due to insufficient storage capacity, and (iii) reduce delay due to the lack of computing performance. To make good use of the advantages, it is necessary to decide what to offload and when to offload. The execution delay and energy consumption of each terminal device can be distinguished into two parts, i.e., local computing and edge computing as explained in (1) and (2). In edge computing architectures, the values include the transmission to edge server, processing on the edge server, and downloading from the edge server.

$$\begin{aligned} Delay_{offload} &= Uploadingdelay + ESprocessingdelay + Downloadingdelay, \\ Delay_{local} &= TDprocessingdelay \end{aligned} \tag{1}$$

where $Delay_{offload}$ and $Delay_{local}$ are the delays when the offloading happens or not, respectively.

$$\begin{aligned} Energy_{offload} &= Uploadingenergy + ESprocessingenergy + Downloadingenergy, \\ Energy_{local} &= TDprocessingenergy \end{aligned} \tag{2}$$

where $Energy_{offload}$ and $Energy_{local}$ are the energy consumption when the offloading happens or not (based on offloading decision-making by DQN), respectively.

As discussed in the literature, offloading decisions are made depending on the following three criteria, i.e., (i) minimization of execution delay, (ii) minimization of energy consumption while satisfying tolerance delay, and (iii) trade-off between energy consumption and execution delay depending on the environment of the applications of offloading in each terminal device [24].

When the goal is the minimization of total execution delay regardless of energy consumption, each device will conduct the internal offloading policy and it decides whether offloading should happen or not, i.e., local computing or offloading computing [25]. If the goal is the minimization of the energy consumption of each device, it uses the smallest energy resources even if the delay increases. This paper considers both execution delay and energy consumption of each terminal. Furthermore, for real-time communications, more weights are allocated to the minimization of total delays than energy consumption. In this postulated environment, each terminal determines whether it has to offload or not, according to DQN-based offloading decision-making algorithm which is based on reinforcement learning instead of general offloading determination [26,27].

4. Proposed DQN-Based Offloading Decision Algorithm

In this section, a DQN-based novel offloading compression decision (OCD) algorithm is proposed. The brief introduction to the algorithm is explained in Figure 3.

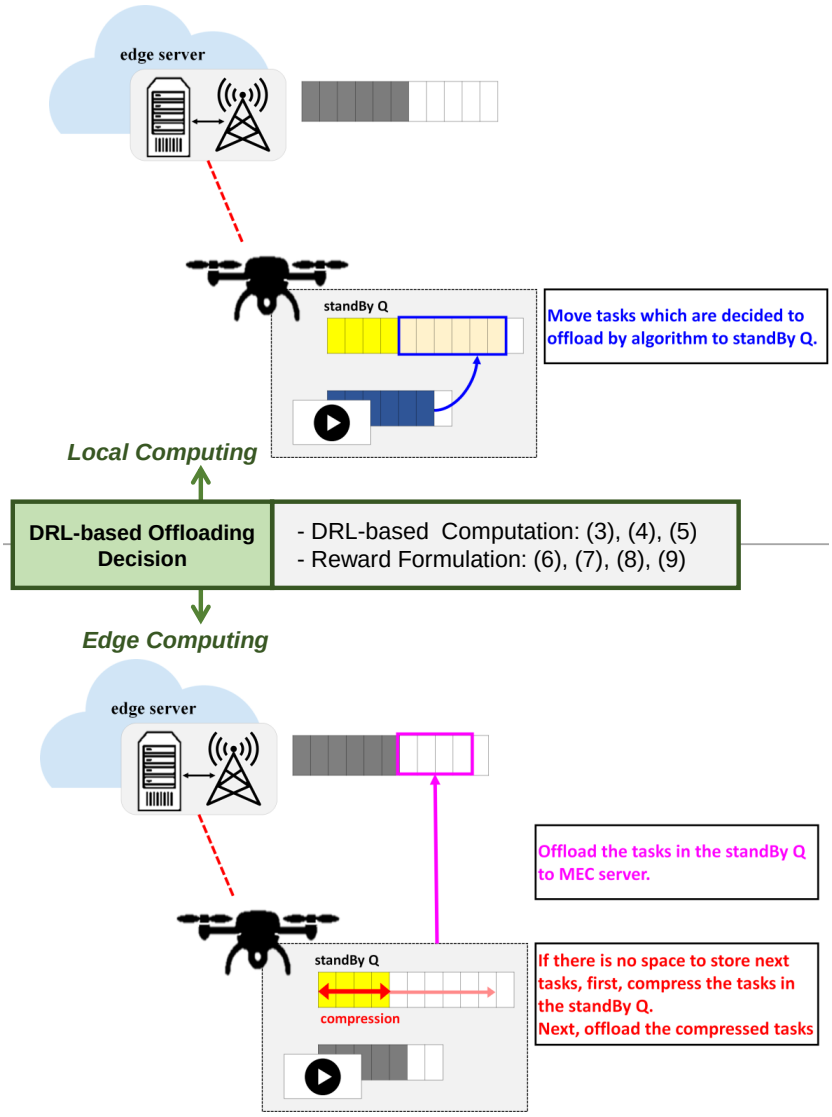


Figure 3. Execution process for the proposed DQN-based novel offloading compression decision algorithm.

4.1. Deep Reinforcement Learning

In theory, the reinforcement learning is a method where an agent learns the set of actions which purposes maximizing cumulative summation of rewards. The reward is defined as a return value that the agent can get. The agent receives the reward and information about the changed state at time step $t+1$ from the environment. Using this reward and observation, the agent learns a policy π, and we say the policy as optimal policy π^* when the reward is always optimal through the policy [28,29]. The reward value can be calculated using Q-function. $\hat{Q}(s,a)$ is a reward of current state and it is a summation of return value r and $\hat{Q}(s',a')$ which is a maximum reward value expected to be received in the future [30]. The corresponding mathematical formulation is as follows:

$$\hat{Q}(s,a) \leftarrow r + \max \hat{Q}(s',a') \tag{3}$$

$$\hat{Q}(s,a) \leftarrow (1-\alpha)\hat{Q}(s,a) + \alpha \left[r + \gamma \max_{a'} \hat{Q}(s',a') \right] \tag{4}$$

where (4) is an equation that contains a learning rate α and a discounting rate γ to (3).

Here, DQN is a way to learn the Q-function using a deep neural network. Through layers, the output informs all or the possible actions and the corresponding reward of each action [31]. The application of deep neural network overcomes the size of memory that has been the limitation of Q-table in Q-learning; and an efficient function output is possible for large amounts of data. The expressions (5) show that it minimizes a difference between the optimal value and real value (i.e., predicted value). The optimal value is a target value that the agent aims to get. The equation form is similar to the *costfunction* of linear regression problem:

$$\min_{\theta} \sum_{t=0}^{T} \left[\hat{Q}(s_t, a_t|\theta) - \left(r_t + \gamma \max_{a'} \hat{Q}(s_{t+1}, a'|\theta) \right) \right]^2 \tag{5}$$

However, since both the target and predicted values use the same network and similar learning data, DQN has a problem with correlation between samples and non-stationary target. The change of target value when performing network update in Q_{pre} is the cause of the non-stationary target problem. Furthermore, because of that problem, it is impossible to make the predicted value similar to the target value, and the agent will learn policies in a different direction than the target. To solve this problem, we can use DQN. It can store data collected by agent in reply buffer and use only data that is extracted from the buffer. In addition, in DQN, the pretend network and the target network are separated. Therefore, the updates of the pretend network θ cannot make an impact to the target network. According to these characteristics of DQN, the agent can get appropriate policies [32–34].

4.2. DQN-Based Offloading and Compression Decision

For the DQN-based offloading decision algorithm, the states, actions, and rewards are defined as mentioned in Section 3.1. This proposed algorithm is formally described in Algorithm 1. In this subsection, we explain the details of the proposed algorithm. Each state is the observation that the agent gets after taking an action from the environment. We set the capacity of TD, StandBy Q, and ES as state, and it represents the capacities' changes over time. Likewise, the agent will get a reward (R_{combi}) in each step. The reward we set is the combination of three aspects of the reward, i.e., the reward of execution delay, the reward of energy consumption, and the reward of terminal device's capacity. The agent uses the state information and reward to make a better action decision:

$$R_{combi} = (\gamma \times R_{delay} + (1-\gamma) \times R_{energy}) \times \beta + R_{capacity} \times (1-\beta) \tag{6}$$

where R_{combi} stands for the reward function definition which is the combination of R_{delay} (reward under the consideration of execution delays), R_{energy} (reward under the consideration of energy consumption), and $R_{capacity}$ (reward under the consideration of capacity) where the three reward components are defined as follows:

$$R_{delay} = 1 - D_{dqn}/D_{local}, \tag{7}$$

$$R_{energy} = 1 - E_{dqn}/E_{local}, \tag{8}$$

$$R_{capacity} = -1 \times tasktodo \times 0.01, \tag{9}$$

where D_{dqn} and E_{dqn} are

$$D_{dqn} = \sum_{n=1}^{N} \alpha_n * D_n^{offload} + (1-\alpha_n) * D_n^{local}, \tag{10}$$

$$E_{dqn} = \sum_{n=1}^{N} \alpha_n * E_n^{offload} + (1-\alpha_n) * E_n^{local}, \tag{11}$$

When offloading computing is decided in each step based on DQN, D_{local} and E_{local} stand for the execution delay and energy consumption when the tasks are processed in the local at time step t. Similarly, $D_{offload}$ and $E_{offload}$ are the values when the tasks are processed in the edge server. Finally, the total execution delay and energy consumption for all the tasks are represented as (10) and (11). Thus, semantically, the (10) and (11) are the linear combinations of execution delay and energy consumption depending on offloading decisions.

Based on the definitions of (10) and (11), the rewards R_{delay} in (7) and R_{energy} in (8) are defined as the difference between the value when the tasks are performed locally and the value by the decision algorithm based on DQN. As a result, the learning is carried out to minimize D_{dqn} and E_{dqn} while it maximizes R_{delay} and R_{energy}. Furthermore, the reward $R_{capacity}$ in (9) is a value to represent an overflow the terminal's capacity. Note that $R_{capacity}$ is meaningful only under certain conditions according to Algorithm 2. If local computing (i.e., no offloading) is decided by the algorithm and the capacity of terminal device is not sufficient to process task at time step t, the negative reward, $R_{capacity}$, is applied.

Algorithm 1 DQN-based offloading decision

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\bar{\theta} = \theta$
for episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  for t = 1, T **do**
    With probability ϵ selects a random action a_t
    Otherwise select $a_t = argmax_a Q(\phi(s_t), a; \theta)$
    Execute action a_T in emulator and observe reward r_t^e, r_t^d and image x_{t+1}
    Set $r_t = \alpha * r_t^e + (1 - \alpha) * r_t^d + r_t^c$ with weight parameter α
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and process $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+'})$ from D
    if Episode terminates at step $j + 1$ **then**
      $y_j = r_j$
    else
      $y_j = r_j + \gamma \max_{a\prime} \hat{Q}(\phi_{j+1}, a\prime; \bar{\theta})$
    end if
    Compute a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q}$=Q
  end for
end for

Algorithm 2 Threshold-based compression decision

```
if task is decided to offload then
    if StandByQcap >= tasktodo then
        compression is not necessary, store the data in the Q
    else
        compress and offload the compressed task to edge server
    end if
else if task is decided to process locally then
    if TDcap >= tasktodo then
        perform the task at terminal device without any negative reward
    else
        perform the task at terminal device
        a negative reward R_capacity(r_t^c) is added
    end if
end if
```

5. Performance Evaluation

This section presents the simulation setting and the performance evaluation results of the proposed DRL-based dynamic offloading decision-making algorithm. We simulate the values of execution delay and energy consumption for the offloading decision tasks computed by the proposed algorithm in each time step. We calculate the values for the processing based on (10) and (11), and we compare the values with the cases of fully local computing and fully offloading.

5.1. Simulation Setting

We assume our simulation setting parameters as follows. Each terminal device has the maximal capacity of 200 (normalized); and also has an internal queue (i.e., standBy Q) which has the maximal capacity of 500. In addition, the capacity of edge server is 10,000. This assumption is reasonable because edge servers are generally powerful comparing to user terminal devices, i.e., order of 100 as used in [35]. The capacities of terminal device queue, stanBy Q, and the edge server are set between maximum capacities at the start of the episode. The episode is terminated when the agent performs 30,000 tasks. In addition, the task which the agent should do is selected between 50 and 350 continuously (i.e., uniform distribution) and, if there are any tasks left at the time step, the remaining tasks are accumulated in each time step. We also set the parameters to calculate the execution delay and energy consumption of the terminal device and edge server, as presented in Table 1. The values of each parameter are based on the 100 task executions and they are the representations of ratios for the local computing values when the execution time and energy consumption are set to 1. We also assume that the terminal device can process 100 tasks at one-time step and also the edge server can process 1250 tasks at the same time. In our DQN-based learning architecture, the input size of the model is set to 4. The input size stands for the number of dimension of observation (TD capacity, Q capacity, ES server capacity, and remaining tasks). The output size is 2, which stands for the number of dimensions of actions (offloading or not).

The assumed values can be updated depending on the development of hardware and computing systems. Despite these parameter updates, our scheme can be superior to the other static schemes. Furthermore, we can conduct more data-intensive simulations with various settings or real-world implementation as one of future research directions.

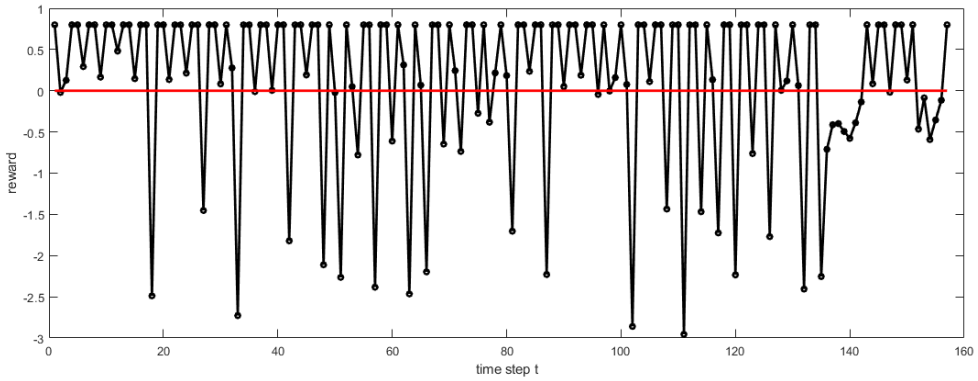
Table 1. Simulation parameters.

Parameter	Value
Processing delay at terminal	1.0
Processing delay at edge server	0.08
Transmission delay (uploading)	0.25
Transmission delay (downloading)	0.05
Processing energy at terminal	1.0
Processing energy at edge server	0.05
Transmission energy (uploading)	0.15
Transmission energy (downloading)	0.05

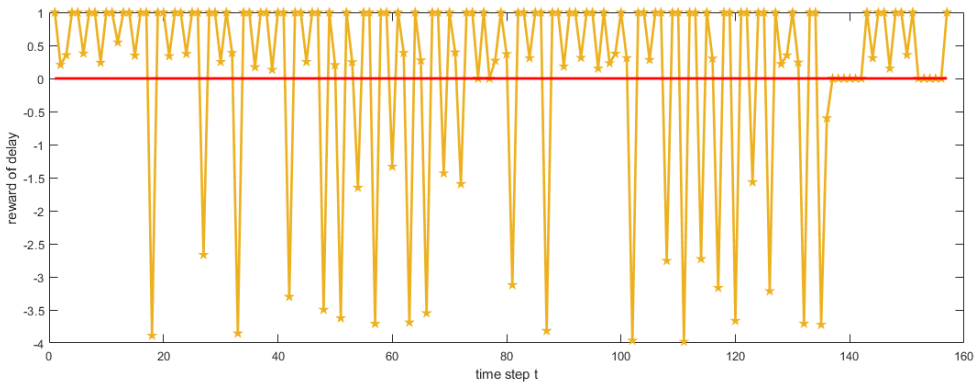
5.2. Simulation Results and Discussions

In this paper, the values of R_{delay} and R_{energy} are no more than 1. In addition, we set the ratio of results by local computing and the OCD algorithm as an indicator of the reward. Furthermore, we set $R_{capacity}$ as a negative reward in order to have 20% of R_{combi} in (6). According to this setting, the maximum value of R_{combi} is 0.8. We can see the rewards per time step in Figure 4. The red graph presents the case where $y = 0$. In the reward of delay and the reward of energy graph, D_{dqn} and E_{dqn} have the same values with D_{local} and E_{local} when the proposed OCD algorithm decides to take an action for non-offloading, i.e., local computing. As a result, the reward values are to be 0, when the OCD algorithm decides offloading D_{dqn}; and E_{dqn} has a value other than zero. If the values of delay and energy consumption are less than D_{local} and E_{local}, the rewards are above the $y = 0$. On the other hand, they are below $y = 0$ if D_{dqn} and E_{dqn} are greater than the values by local computing. In Figure 4, the first graph is the combination of the below two graphs. It is as shown in Figure 5, and it stands for the reward that the agent receives at every time step. In Figure 5, the red line presents the accumulated reward. Although the OCD algorithm always makes optimal offloading decisions, the reward is not always positive. As a result, the total value of the reward can decrease when the negative reward is accumulated. In particular, there is a constant return of negative rewards from 130 to 150 intervals.

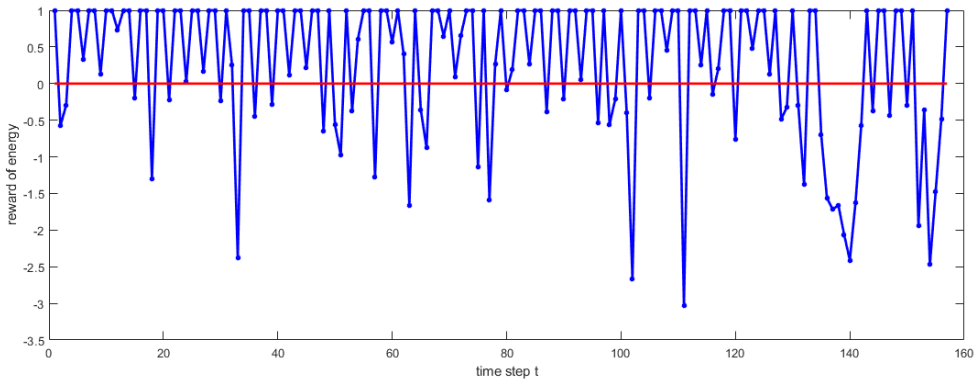
We compare the result of 30,000 tasks performed by OCD algorithm with the results performed by fully local computing (i.e., no offloading) and edge computing. The executed simulation gets results as shown in Figure 6 and Table 2. If there are tasks that are not processed at time step t due to capacity limit, the tasks are accumulated as the tasks to be handled at the next time step. Thus, in the execution delay and energy consumption side, local computing continues to increase the values. Furthermore, from the time step 30, the time to execute tasks is the longest and it reduces TD's energy as maximum. The full offloading has the biggest values at the first time, and it has the smallest growth rate until all the tasks are executed. The reason why the initial values of full offloading is a longer transmission time. The proposed OCD algorithm (DQN-based) has significantly better performance than the local computing by TD. In terms of execution delay, the value of the proposed OCD algorithm (DQN-based) is the lowest until time step 110. However, as the time step goes by, and the task size increases, the results of the proposed OCD algorithm (DQN-based) are better than the results of full offloading. In the energy consumption side, the proposed DQN-based algorithm always has the smallest values. These results show that the proposed OCD algorithm (DQN-based) is the most suitable in real-time wireless communication applications.



(**a**)



(**b**)



(**c**)

Figure 4. Reward of each time step t. (**a**) represents the figure of R_{combi} values that is a combination of R_{delay}, R_{energy}, and $R_{capacity}$. (**b**) is a graph of R_{delay} per time step, (**c**) is about R_{energy}.

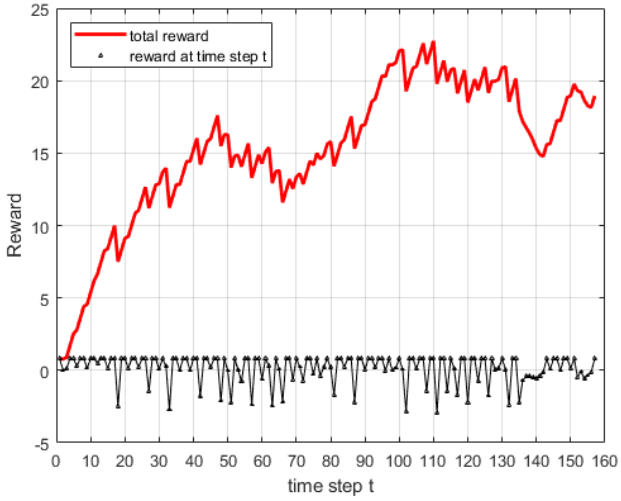


Figure 5. Accumulated reward.

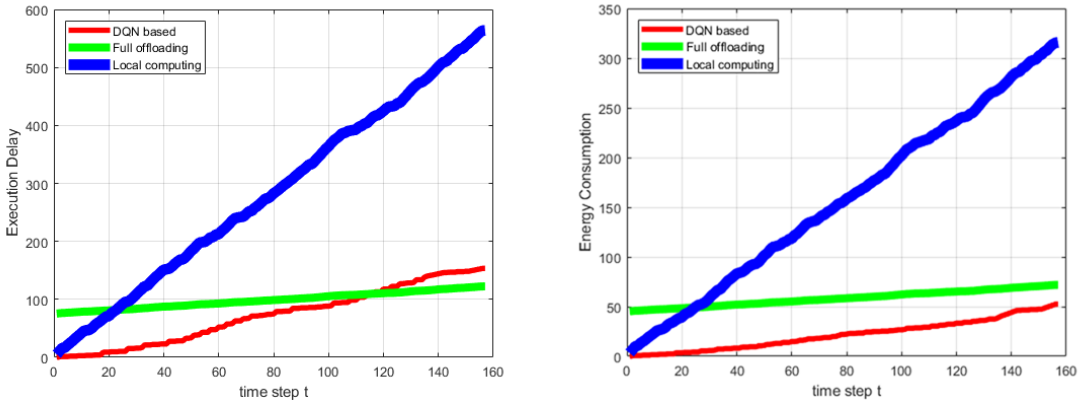


Figure 6. Performance evaluation results (comparison among DQN-based, full offloading, and local offloading): execution delay (**left**) and energy consumption (**right**).

Table 2. Energy consumption and latency values with 10,000 tasks.

	DQN Based	Local Computing	Full Offloading
Energy consumption	52.448	315.830	71.845
Execution delay	153.320	563.830	122.374

Lastly, we conduct the simulations in order to present the novelty of DQN-based algorithm than the other methods. One of the offloading decision algorithms is designed based on well-known greedy algorithms, as discussed in [36]. We remind readers that our proposed BRL-based offloading decisions will be made based on the DQN-based framework under the consideration of our rewards (i.e., combination of delay, energy, and capacity). On the other hand, the greedy algorithm can be used for offloading decision-making based on uploading data sizes (i.e., related to delays), i.e., this algorithm firstly sorts its own data to transmit at first. Then, it transmits the data sequentially.

As shown in Figure 7, this greedy-based algorithm introduces more average delays than our proposed DQN-based algorithm. Note that the average delay is defined as follows when N number of packets are simulated for the average delay calculation:

$$\frac{1}{N}\sum_{i=1}^{N}\left(t_i^{\text{Departure}} - t_i^{\text{Arrival}}\right) \quad (12)$$

where $t_i^{\text{Departure}}$ and t_i^{Arrival} are the times when the i-th packet is left from the queue and when the i-th packet is arrived at the queue, respectively. As the average data size for offloading increases, the delays increase. In the DQN-based algorithm, optimal control will be made which considers delay, energy, and capacity, during offloading decisions. Thus, the delays are smaller than the greedy-based decision-making. As shown on the right-hand side of Figure 7, our proposed DQN-based decision-making algorithm outperforms the greedy; and the gaps become larger when the average data sizes increase.

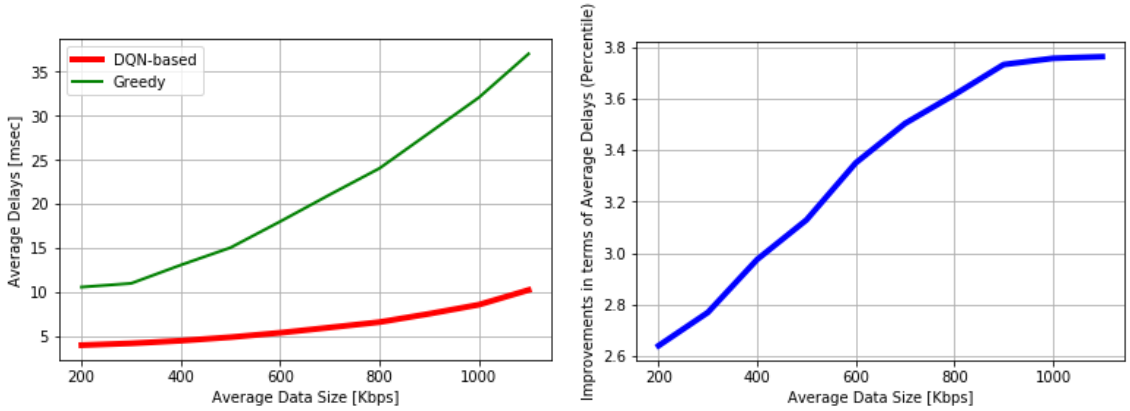


Figure 7. Performance evaluation results (Proposed DQN-based vs. Greedy): Average delays (**left**) and performance gap (**right**).

6. Concluding Remarks and Future Work

This paper proposes a novel deep Q-network based offloading decision-making method in mobile edge computing systems. For utilizing real-time and high-quality communications in mobile edge networks, efficient offloading algorithms and task scheduling in MEC servers are essentially required. In this paper, therefore, a new dynamic offloading algorithm is proposed, and the algorithm improves its performance based on DQN. For performance evaluation, the results of our simulation can be changed due to environment settings or reward decision parameters. According to the simulation-based evaluation, we verify that the proposed DQN-based algorithm achieves desired performance.

As future research directions, various factors can be considered in order to conduct more realistic performance evaluation (e.g., channel allocation, channel quality, and mobility of devices) and real-world implementation (e.g., with Raspberry Pi). In addition, more detailed discussions about energy consumption and energy-efficiency are desired. Therefore, we will include this in our future investigation. Moreover, the computational overhead is definitely considerable for our future work because considering the overhead is meaningful in mobile edge computing research. Lastly, the proposed algorithm in this paper runs in a centralized manner. It would be much scalable and practical if the computation can be done in a distributed manner. Therefore, our next work should be dedicated to the design and implementation of fully distributed multi-agent reinforcement learning algorithm which does not require centralized computation.

Author Contributions: S.P., J.K., and S.C. were the main researchers who initiated and organized research reported in the paper, and all authors including S.P., D.K., and J.K. were responsible for analyzing the simulation results and writing the paper. Y.K.L. provided more realistic network setting and its related most suitable algorithm modification. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Chung-Ang University Graduate Research Scholarship in 2019 (for Soohyun Park) and the National Research Foundation of Korea (2019M3E4A1080391).

Acknowledgments: J.K., Y.K.L., and S.C. are the corresponding authors of this paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Kim, J.; Molisch, A.F. Fast Millimeter-Wave Beam Training with Receive Beamforming. *J. Commun. Netw.* **2014**, *16*, 512–522. [CrossRef]
2. Kwon, D.; Kim, S.-W.; Kim, J.; Mohaisen, A. Interference-Aware Adaptive Beam Alignment for Hyper-Dense IEEE 802.11ax Internet-of-Things Networks. *Sensors* **2018**, *18*, 3364. [CrossRef] [PubMed]
3. Kim, J.; Caire, G.; Molisch, A.F. Quality-Aware Streaming and Scheduling for Device-to-Device Video Delivery. *IEEE/ACM Trans. Netw.* **2016**, *24*, 2319–2331. [CrossRef]
4. Dao, N.-N.; Vu, D.-N.; Na, W.; Kim, J.; Cho, S. SGCO: Stabilized Green Crosshaul Orchestration for Dense IoT Offloading Services. *IEEE J. Sel. Areas Commun.* **2018**, *36*, 2538–2548. [CrossRef]
5. Choi, M.; No, A.; Ji, M.; Kim, J. Markov Decision Policies for Dynamic Video Delivery in Wireless Caching Networks. *IEEE Trans. Wirel. Commun.* **2019**, *18*, 5705–5718. [CrossRef]
6. Kwon, D.; Kim, J. Multi-Agent Deep Reinforcement Learning for Cooperative Connected Vehicles. In Proceedings of the IEEE Global Communications Conference (GLOBECOM), Waikoloa, HI, USA, 9–13 December 2019.
7. Kwon, D.; Kim, J. Optimal Trajectory Learning for UAV-BS Video Provisioning System: A Deep Reinforcement Learning Approach. In Proceedings of the IEEE International Conference on Information Networking (ICOIN), Kuala Lumpur, Malaysia, 9–11 January 2019.
8. Wu, H. Multi-Objective Decision-Making for Mobile Cloud Offloading: A Survey. *IEEE Access* **2018**, *6*, 3962–3976. [CrossRef]
9. Wu, H.; Sun, Y.; Wolter, K. Energy-Efficient Decision Making for Mobile Cloud Offloading. *IEEE Trans. Cloud Comput.* **2020**, *2020*, 1–15. [CrossRef]
10. Kim, B.; Min, H.; Heo, J.; Jung, J. Dynamic Computation Offloading Scheme for Drone-based Surveillance Systems. *Sensors* **2018**, *18*, 2982. [CrossRef]
11. Huang, L.; Feng, X.; Zhang, C.; Qian, L.; Wu, Y. Deep Reinforcement Learning-based Joint Task Offloading and Bandwidth Allocation for Multi-User Mobile Edge Computing. *Digit. Commun. Netw.* **2019**, *5*, 10–17. [CrossRef]
12. Akherfi, K.; Gerndt, M.; Harroud, H. Mobile Cloud Computing for Computation Offloading: Issues and Challenges. *Appl. Comput. Inform.* **2018**, *14*, 1–16. [CrossRef]
13. Park, S.; Kim, J.; Kwon, D.; Shin, M.; Kim, J. Joint Offloading and Streaming in Mobile Edges: A Deep Reinforcement Learning Approach. In Proceedings of the IEEE VTS Asia Pacific Wireless Communications Symposium (APWCS), Singapore, 28–30 August 2019.
14. Shin, M.; Choi, D.-H.; Kim, J. Cooperative Management for PV/ESS-Enabled Electric-Vehicle Charging Stations: A Multi-Agent Deep Reinforcement Learning Approach. *IEEE Trans. Ind. Inform.* **2020**, *16*, 3493–3503. [CrossRef]
15. Feng, L.; Obayshi, M.; Kuremoto, T.; Kobayashi, K. An Intelligent Control System Construction using High-Level Time Petri Net and Reinforcement Learning. In Proceedings of the IEEE International Conference on Control, Automation and Systems (ICCAS), Gyeonggi-do, Korea, 27–30 October 2010.
16. Bhagat, S.; Banerjee, H.; Tse, Z.T.H.; Ren, H. Deep Reinforcement Learning for Soft, Flexible Robots: Brief Review with Impending Challenges. *Robotics* **2019**, *8*, 4. [CrossRef]
17. Zhang, Y.; Sun, P.; Yin, Y.; Lin, L.; Wang, X. Human-Like Autonomous Vehicle Speed Control by Deep Reinforcement Learning with Double Q-Learning. In Proceedings of the IEEE Intelligent Vehicles Symposium (IV), Changshu, China, 26–30 June 2018.
18. Xu, X.; Zuo, L.; Li, X.; Qian, L.; Ren, J.; Sunm, Z. A Reinforcement Learning Approach to Autonomous Decision Making of Intelligent Vehicles on Highways. *IEEE Trans. Syst. Man Cybern. Syst.* **2018**. [CrossRef]
19. Zheng, R.; Liu, C.; Guo, Q. A Decision-Making Method for Autonomous Vehicles based on Simulation and Reinforcement Learning. In Proceedings of the International Conference on Machine Learning and Cybernetics (ICMLC), Tianjin, China, 14–17 July 2013.
20. Sqryn, M.; Sharma, A.; Parkar, D.; Shrimal, M. Distributed Deep Reinforcement Learning on the Cloud for Autonomous Driving. In Proceedings of the IEEE/ACM International Workshop on Software Engineering for AI in Autonomous Systems (SEFAIAS), Gothenburg, Sweden, 28 May 2018.
21. Shin, M.; Kim, J. Randomized Adversarial Imitation Learning for Autonomous Driving. In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), Macau, China, 10–16 August 2019.

22. Shin, M.; Kim, J. Adversarial Imitation Learning via Random Search. In Proceedings of the International Joint Conference on Meural Networks (IJCNN), Budapest, Hungary, 14–19 July 2019; pp. 1–8.
23. Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. Mastering the Game of Go without Human Knowledge. *Nature* **2017**, *550*, 354–359. [CrossRef] [PubMed]
24. Shi, W.; Cao, J.; Zhang, Q.; Li, Y.; Xu, L. Edge Computing: Vision and Challenges. *IEEE Internet Things J.* **2016**, *3*, 637–646. [CrossRef]
25. Chen, X.; Jiao, L.; Li, X.; Fu, X. Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing. *IEEE/ACM Trans. Netw.* **2016**, *24*, 2795–2808. [CrossRef]
26. Munoz, O.; Iserte, A.P.; Vidal, J.; Molina, M. Energy-Latency Trande-off for Multiuser Wireless Computation Offloading. In Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC) Workshop on Cloud Technologies and Energy Efficiency in Mobile Communication Networks, Istanbul, Turkey, 6–9 April 2014.
27. Wang, J.; Zhao, L.; Liu, J.; Kato, N. Smart Resource Allocation for Mobile Edge Computing: A Deep Reinforcement Learning Approach. *IEEE Trans. Emerg. Top. Comput.* **2020**. [CrossRef]
28. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 1998.
29. Sutton, R.S.; McAllester, D.A.; Singh, S.P.; Mansour, Y. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In Proceedings of the Advances in Neural Information Processing Systems (NIPS), Denver, CO, USA, 1 January 2000; pp. 1057–1063.
30. Zhu, L.; He, Y.; Yu, F.R.; Ning, B.; Tang, T.; Zhao, N. Communication-based Train Control System Performance Optimization using Deep Reinforcement Learning. *IEEE Trans. Veh. Technol.* **2017**, *66*, 10705–10717. [CrossRef]
31. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Petersen, S. Human-Level Control through Deep Reinforcement Learning. *Nature* **2015**, *518*, 529–533. [CrossRef]
32. Nair, A.; Srinivasan, P.; Blackwell, S.; Alcicek, C.; Fearon, R.; Maria, A.; Panneershelvam, V.; Suleyman, M.; Beattie, C.; Petersen, S.; et al. Massively Parallel Methods for Deep Reinforcement Learning. *arXiv* **2015**, arXiv:preprint/1507.04296.
33. Kober, J.; Bagnell, J.A.; Peters, J. Reinforcement Learning in Robotics: A Survey. *Int. J. Robot. Res.* **2013**, *32*, 1238–1274. [CrossRef]
34. Arulkumaran, K.; Deisenroth, M.P.; Brundage, M.; Bharath, A.A. Deep Reinforcement Learning: A Brief Survey. *IEEE Signal Process. Mag.* **2017**, *34*, 26–38. [CrossRef]
35. Bozorgchenani, A.; Tarchi, D.; Corazza, G.E. An Energy and Delay-Efficient Partial Offloading Technique for Fog Computing Architectures. In Proceedings of the IEEE Global Communications Conference (GLOBECOM), Singapore, 4–8 December 2017; pp. 1–6.
36. Feng, W.; Yang, C.; Zhou, X. Multi-User and Multi-Task Offloading Decision Algorithms Based on Imbalanced Edge Cloud. *IEEE Access* **2019**, *7*, 95970–95977. [CrossRef]