


Article

Deep Reinforcement Learning by Balancing Offline Monte Carlo and Online Temporal Difference Use Based on Environment Experiences

Chayoung Kim 

College of Liberal Arts and Interdisciplinary Studies, Kyonggi University, 154-42 Gwanggyosan-ro, Yeongtong-gu, Suwon-si, Gyeonggi-do 16227, Korea; kimcha0@kgu.ac.kr; Tel.: +82-31-249-9509

Received: 7 September 2020; Accepted: 13 October 2020; Published: 14 October 2020



Abstract: Owing to the complexity involved in training an agent in a real-time environment, e.g., using the Internet of Things (IoT), reinforcement learning (RL) using a deep neural network, i.e., deep reinforcement learning (DRL) has been widely adopted on an online basis without prior knowledge and complicated reward functions. DRL can handle a symmetrical balance between bias and variance—this indicates that the RL agents are competently trained in real-world applications. The approach of the proposed model considers the combinations of basic RL algorithms with online and offline use based on the empirical balances of bias–variance. Therefore, we exploited the balance between the offline Monte Carlo (MC) technique and online temporal difference (TD) with on-policy (state-action–reward-state-action, Sarsa) and an off-policy (Q-learning) in terms of a DRL. The proposed balance of MC (offline) and TD (online) use, which is simple and applicable without a well-designed reward, is suitable for real-time online learning. We demonstrated that, for a simple control task, the balance between online and offline use without an on- and off-policy shows satisfactory results. However, in complex tasks, the results clearly indicate the effectiveness of the combined method in improving the convergence speed and performance in a deep Q-network.

Keywords: Q-learning (off-policy); sarsa (on-policy); reinforcement learning (RL); internet of things (IoT); monte carlo (offline); Q-learning (online); deep learning

1. Introduction

Reinforcement learning (RL) and deep reinforcement learning (DRL) are incredibly autonomous and interoperable, i.e., they have many real-time Internet of Things (IoT) applications. RL pertains to a machine learning method based on trial-and-error, which improves the performance by accepting feedback from the environment [1,2]. There have been many studies on applying RL or DRL in IoT, which are relevant to a variety of applications, such as energy demand based on the critical load or real-time electricity prices in a smart grid. Robots or smart vehicles using IoT are autonomous in their working environment, wherein they attempt to find a collision-free path from the current location to the target. Regarding the applications of RL or DRL in autonomous IoT, a broad spectrum of technology exists, such as fast real-time decisions made locally in a vehicle or the transmission of data to and from the cloud [1,2]. In particular, one of the large issues regarding a real-time fast decision is online learning and decision making based on approximate results from the learning [1–3] similar to near-optimal path-planning with respect to real-time criteria. However, the real-time environment might be imprecise, dynamic, and partially non-structured [1–4]. Owing to the complexity of online learning and real-time decisions in RL or DRL, Q-learning or a deep Q-network (DQN) has been widely adopted along with some pre-trained [5] or prior knowledge, such as environmental maps or environmental dynamics [6]. In [5], the authors showed that deep Q-learning with transfer learning is

significantly applicable in emergency situations such as fire evacuation planning. Their models have shown that an emergency evacuation can benefit from RL because it is highly dynamic, with a lot of changing variables and complex constraints. Prior knowledge can help a smart robot with navigation planning and even obstacle avoidance. Fundamentally, however, little prior knowledge of RL has been presumed.

The approach of the proposed model is similar to that of recent studies [7,8], which have applied RL techniques to real-time applications on an online basis and a combination of different algorithms, without prior knowledge. These environments pose more challenges than a simulated environment, such as enlarged state spaces and an increased computational complexity. The primary advantage of RL lies in its inherent power of automatic learning even in the presence of small changes in the real-time environment. Regardless of whether real-time applications with self-learning have become a significant research topic, there have already been some studies based on improved Q-learning or a DQN on the replacement of the complicated parameters, although in approximations and not in actual results [1,2]. In [7], the authors investigated when an optimization is necessary through an online method. In the era of big data, in [7], a DQN and deep policy gradient (DPG) are proposed to overcome the time consumption required for all possible solutions and to allow the best solution to be chosen. Moreover, the authors showed that their models are capable of generalization and exploitation and minimize the energy consumption or cost in newly encountered situations [7]. In [8], the authors demonstrated how a different mixing of off-policy gradient estimates with on-policy samples contribute to improvements in the empirical performance of a DRL. However, recent research [1] has shown that a DRL is confronted with situations that differ in minor ways from those upon which the DRL was trained, indicating that solutions to DRLs are often extremely superficial. Most of these studies require the state spaces of the smart grid to be well-organized, notwithstanding that real applications take place in real value vectors of the state spaces, which are continuous and large in scale [1]. Thus, using only Q-learning with complicated reward functions in real applications can lead to larger issues, such as the curse of dimensionality [1,7]. In terms of the approximation of a value function, a generalization in RL can cause a divergence in the case of bootstrapped value updates [9]. Therefore, we are devoted to determining and thoroughly understanding why and when an RL is able to work well. The design is also inspired by recent studies [2,9–11] that provide a better understanding on the role of online and offline environments in RL, which are characterized by an extremely high observation dimensionality and infinite-dimensional state spaces. Hausknecht and Stone [9] indicated that mixing an off-policy Q-learning of online TD updates with offline MC updates provides an increased performance and stability for a deep deterministic policy gradient. However, Hausknecht and Stone [9] showed slow learning for deep Q-learning. In addition, Xu et al. [10] proposed deep Sarsa and Q-networks, which are combined with both on-policy Sarsa and off-policy Q-learning in online learning. Off-policy Q-learning is considered better than on-policy Sarsa in terms of the local minima in online learning. However, Q-learning also suffers from a high bias estimate because the estimate is never completely accurate [3]. Wang et al. [11] also combined Sarsa and Q-learning but utilized some information regarding a well-designed reward until the maximum time is reached. As such, their study [11] necessitated the full storage for the entire episode as a worst case. By contrast, the proposed model does not consider a well-designed reward for optimization, thereby making it different from the research by Wang et al. Remarkably, Amiranashvili et al. [2] demonstrated that, in a representative online learning, the temporal difference (TD), is not always superior to a representative offline learning, i.e., a Monte Carlo (MC) approach.

In this study, we exploit a combination of an offline learning MC approach and off-policy Q-learning and an on-policy state-action-reward-state-action (Sarsa) of online learning TD. The purpose of this combination is to achieve a reasonable performance and stability with reply memories and updates to a target network [12] while ensuring a real-time online learning criterion in the environments. By using the DQN, which is the de facto standard, Q-estimates are computed using the target network, which can provide older Q-estimates, with a specific bias limiting the generalization but achieving a

more stable network instead. In RL, bias and variance indicates how well the RL signals reflect the true rewards in the environment. A combination of online and offline environments is applied for a bias-variance tradeoff [13]. Previous studies [8,14] have handled the issues involved in balancing bias and variance. The most common approaches are to reduce the variance of an estimate while keeping the bias unchanged. The baselines of such studies [8,14] are of the policy gradient [15], which utilizes an actor, who defines the policy, and a critic, who provides a more reduced variance reward structure to update the actor.

Herein, we consider DQN, the de facto standard, and Q-learning with a ϵ -greedy algorithm for the balance of exploration and exploitation, which is easily applicable to more general cases and is a simple but powerful strategy for the challenge of bias–variance tradeoff—a crucial element in several RL components. This study aims to bridge the gap between theory and practice using a simple RL strategy. Therefore, we rely on the “baseline” of the balance in offline MC and online TD with off-policy Q-learning and on-policy Sarsa in a real-time environment. Based on these considerations, we propose a random probability that determines whether to use an offline or online environment during the learning process. The initial value of probability δ is δ_{ini} . The value of δ_{ini} is set to 1.0; therefore, the proposed algorithm will use offline learning with higher probability during the initial stage. For random probability δ , each step is decreased by $\Delta\delta$ until $\delta_{fin} = 0.01$. Consequently, the proposed algorithm will more likely utilize online learning with an on- and off-policy during the late stage. The algorithm is based on a simple and random method with probability δ , and it has facilitated better prediction by the agent in several cases. Therefore, as training of the agent progresses, a complete episode for offline learning is not required.

Through this study, we show that merely using only a DQN by balancing offline and online environments with an on- and off-policy achieves a satisfactory result. We demonstrate the capability of the proposed model and its suitability for RL in an autonomous IoT for achieving a bias–variance trade-off. The aforementioned contributions are significant because several researchers have based their studies on complex function approximations with deep neural networks. In the simulations using control problems such as a cart-pole balancing, mountain-car, and lunar-lander from the OpenAI Gym [16], we demonstrated that in simple control task such as a cart-pole and mountain-car, merely the balance of online and offline environments without an on- and off-policy achieves satisfactory results. However, in complex tasks such as a lunar-lander or the cart-pole and mountain-car with qualified upper bound, the results provide direct evidence of the effectiveness of the combined method for improving the convergence speed and performance. The proposed algorithm initially chooses a gradual balance from an offline environment, followed by online with on- and off-policy towards the end of the application, with a simple and random probability. Furthermore, we attempt to demonstrate the superiority of this algorithm over other technique by comparing it with the classic DQN, DQN with MC, and DQN with Sarsa. The proposed algorithm aims to achieve a significantly lightweight version of a random balance of the probability, and is worth consideration in most real-time environments.

2. Background

The standard structure of the RL [17–19] is given in Figure 1. The environment and agent of the learning system interact continuously. The agent, based on the policy, selects an action a_t in the current state s_t . The environment will then supply a reward to the agent based on the action a_t , and create a new situation, s_{t+1} . In the environment, the state s_t , the action a_t , the reward r_{t+1} , the new state s_{t+1} , and the new action a_{t+1} are presented in a circular form. RL attempts to teach the agent how to improve the action, when placed in an unknown environment, by acquiring the near-optimal Q-values that achieve the best results for all states. The agent takes advantage of the rewards given by the environment after selecting an action in every state to update the Q-values for a convergence of optimality. The constant issue of a trade-off between exploration and exploitation in the unknown environment in an RL algorithm has yet to be addressed. On the one hand, choosing the action with the best-estimated value implies that the agent exploits its current knowledge. On the other hand,

choosing one of the other actions implies that the agent explores how to improve its estimate of the values of such actions. The exploitation maximizes the reward system in the short term. However, it does not guarantee a maximization of the accumulated reward in the long run. As such, although the exploration reduces the short-term benefits of the total rewards, it produces the maximum reward in the long run. This is because, after the agent has explored the actions at random, allowing the agent to check for better alternatives, it can begin to exploit them. It must be noted that exploitation and exploration are mutually exclusive. Hence, the agent cannot perform both exploration and exploitation in one selection. Therefore, it is fundamentally essential to balance the exploration and exploitation for the convergence of the near-optimal value functions. The most common algorithm for balancing this trade-off of exploration and exploitation is the ϵ -greedy algorithm. In this algorithm, the action with the maximally estimated value is called the “greedy action,” and the agent usually exploits its current knowledge by choosing this so-called greedy action. However, there are other chances of probability ϵ for the agent to explore under a random selection, i.e., “non-greedy actions.” This type of action selection is called a ϵ -greedy algorithm [17–19].

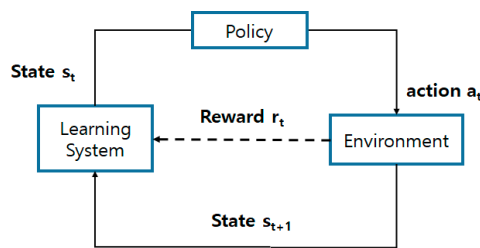


Figure 1. Reinforcement learning (RL) structure.

Furthermore, two different approaches, MC and TD, are applied when dealing with the trade-off between online and environments offline for determining the Q-value functions in RL [17]. TD can learn before knowing the final outcome. Thus, TD can learn online after every step. However, MC can only learn from complete sequences. Thus, MC learning is offline. Figure 2 shows a significant difference between MC and TD. In Figure 2, $V(S_t)$ is the value function at S_t and G_t is the total discounted reward. α is the learning rate and γ is the discount factor. $R_{t+1} + \gamma V(S_t)$ is the estimated return, also known as the TD-target, and $[R_{t+1} + \gamma V(S_t) - V(S_t)]$ is TD-error. In addition, Figure 3 shows a first-visit MC policy evaluation [17].

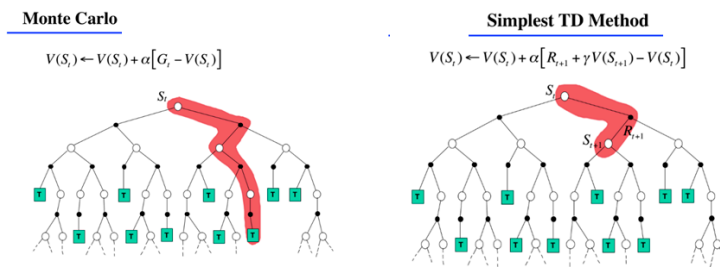


Figure 2. Differences between Monte Carlo (MC) and temporal difference (TD) [17].

- To evaluate state s
- The first time-step t that state s is visited in an episode,
- Increment counter $N(s) \leftarrow N(s) + 1$
- Increment total return $S(s) \leftarrow S(s) + G_t$
- Value is estimated by mean return $V(s) = S(s)/N(s)$
- By law of large numbers, $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$

Figure 3. First-visit MC policy evaluation [17].

Among the most common TD algorithms are Sarsa, which is an on-policy, and Q-learning, an off-policy. There are two types of policy learning methods in RL: on-policy and off-policy. On-policy learns on the job, which means it evaluates or improves the policy that is used to make the decisions. By contrast, off-policy evaluates one target policy, while following another behavior policy. TD methods allow learning directly from the previous experience, do not require any model of the environment, ascertain convergence for near-optimal performance, and are easy to implement. For these reasons, TD methods have been widely adopted since researchers first started using RL algorithms. The Sarsa algorithm was proposed by Rummery and Niranjana [20]. The Sarsa algorithm estimates the value of $Q(s_t, a_t)$ by applying a_t in state s_t according to the updated formula

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + [r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \quad (1)$$

This update can be performed after every transition from a non-terminal state s_t . Here, $Q(s_{t+1}, a_{t+1})$ is determined as zero if s_{t+1} is terminal. Every element of the quintuple event $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ is used in the updated Formula (1), which is a transition from a pair of one state s_t and action a_t to the next. Thus, this quintuple leads to the name Sarsa. The Sarsa algorithm is shown in Figure 4.

```

Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$ 
for each episode:
  Initialize  $S$ 
  Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.  $\epsilon$ -greedy)
  for each step of episode:
    Take action  $A$ , observe  $R, S'$ 
    Choose  $A'$  from  $S'$  using policy derived from  $Q$  (e.g.  $\epsilon$ -greedy)
     $Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$  # on-policy
     $S \leftarrow S'$ 
     $A \leftarrow A'$ 
  do  $S$  is not terminal
do the episode is not done

```

Figure 4. Sarsa algorithm [20].

Moreover, the most popular TD, Q-learning, which is an off-policy proposed by Watkins and Dayan, is one of the most important RL algorithms [21]. Q-learning is determined by

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + [r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (2)$$

where α is the learning rate, γ is the discount factor, and r_{t+1} is the immediate reward received from the environment by taking action a_t in state s_t at the moment of time t . The Q-learning algorithm is given in Figure 5. The difference between Sarsa and Q-learning is the TD-target, which is mentioned above. The TD-target, $R + \gamma Q(S', A')$ in Sarsa means “Update the current Q value with the immediate reward and the Q value of the next action”. However, the TD-target, $R + \gamma \max_a Q(S', a)$ in Q-learning means “an next action is chosen using behavior policy. But, the alternative successor action is considered.” Therefore, Q-learning is off-policy, which means it will “Evaluate target policy while following behavior policy”.

Sarsa is actually an enhancement of Q-learning in terms of fast convergence. In other words, Sarsa allows the agent to learn faster than normal. Apart from Sarsa, other studies have focused on improving the learning performance in Q-learning [17–19].

```

Initialize  $Q(s, a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ , arbitrarily, and  $Q(\text{terminal-state}, \cdot) = 0$ 
for each episode:
  Initialize  $S$ 
  for each step episode:
    Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.  $\epsilon$ -greedy)
    Take action  $A$ , observe  $R, S'$ 
     $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$  # off-policy
     $S \leftarrow S'$ 
  do  $S$  is not terminal
do the episode is not done

```

Figure 5. Q-learning algorithm [21].

DQN is a well-known, model-free RL algorithm attributed to discrete action spaces. In DQN [18,19], we construct a DNN, Q , which approximates Q^* and is greedily defined as $\pi_Q(s) = \operatorname{argmax}_{a \in \mathcal{A}} Q(s, a)$ [18,19]. This is a ϵ -greedy policy with probability ϵ that takes the action $\pi_Q(s)$ with probability $1 - \epsilon$. Each episode uses the ϵ -greedy policy following Q as an approximation of a DNN. The tuples (s_t, a_t, r_t, s_{t+1}) are stored in the replay buffer, and each new episode is configured to neural network training [18,19]. The DNN is trained using the gradient descent of random episodes on a loss function, encouraging Q to follow the Bellman equation [17]. The tuples are sampled from the replay buffer of random episodes. The target network y_t is computed using a separate neural network that changes more slowly than the main deep neural network to optimize the process stability. The weights of the target network are set to the current weights of the main deep neural network. The DQN algorithm [18,19] is presented in Figure 6. The maximum action a_t is selected by $Q^*(s_t, a; \theta)$ of DNN with the probability $1 - \epsilon$. The TD-target, y_j is $r_j + \gamma \max_{a'} Q(\Phi_{j+1}, a'; \theta)$ and TD-error is $y_j - Q(\Phi_j, a_j; \theta)$. The DNN performs a gradient descent on the TD-error.

Algorithm 1 Deep Q-learning with Experience Replay

```

Initialize replay memory  $\mathcal{D}$  to capacity  $N$ 
Initialize action-value function  $Q$  with random weights
for episode = 1,  $M$  do
  Initialise sequence  $s_1 = \{x_1\}$  and preprocessed sequenced  $\phi_1 = \phi(s_1)$ 
  for  $t = 1, T$  do
    With probability  $\epsilon$  select a random action  $a_t$ 
    otherwise select  $a_t = \max_a Q^*(\phi(s_t), a; \theta)$ 
    Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$ 
    Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$ 
    Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $\mathcal{D}$ 
    Sample random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $\mathcal{D}$ 
    Set  $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$ 
    Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j; \theta))^2$  according to equation 3
  end for
end for

```

Figure 6. Deep Q-network (DQN) algorithm [18,19].

3. Proposed Algorithm

3.1. Balancing Offline and Online in DQN

RL has significant applications in real-world online learning. However, owing to the complexity of balancing exploration and exploitation, there are some considerations when adapting it to online learning for real-world applications. The performance of RL is significantly influenced by two important factors used in the algorithm: “exploration” and “exploitation”. Exploration usually refers to the selection of any action with a non-zero probability by the agent, whereas exploitation refers to a situation wherein the agent uses its current knowledge during the selection process [17–19]. The proposed approach, RL with an online and offline environment, combines offline MC, which is unbiased but has high-variance, and the online approach of off-policy Q-learning and on-policy Sarsa, which is low-variance but biased. The approach of the proposed model considers the combinations of basic algorithms in RL with online and offline environments for an exploration–exploitation tradeoff that

considers the empirical balances of bias–variance, similar to [22], which uses variance estimates in multi-armed bandits for an exploration–exploitation trade-off. Owing to the integration of various RL methods, several computational parameters should be tuned, which leads to prolonged computational times. Moreover, these factors may worsen the online learning in terms of convergence. Therefore, RL in combination with heuristic methods, such as human interactions, has been proposed to accelerate the convergence. Most of these approaches are related to the management balance in Q-learning, which is the most well-known RL method. In [11,23], complicated reward functions were studied with respect to balancing the exploration and exploitation in Q-learning, e.g., tuned action-selection policies or the adaptive learning rate and discount rate parameters with applied artificial intelligence (AI) techniques with fuzzy logic. Moreover, the two policies Q-learning and Sarsa are merged for bias and variance balance by employing baseline strategies such as actor–critic methods [8,14]. Q-learning has an off-policy TD, whereas Sarsa, an alternative to Q-learning, has an on-policy TD [2]. Overall, Q-learning provides a better final performance, whereas Sarsa provides faster convergence [11]. However, Q-learning with the abovementioned strategies remains a challenge in terms of balancing the exploitation and exploration, because the estimate is never completely accurate [3] and involves an enormous number of computations. Therefore, in this study, a combination of offline MC and online TD with off-policy Q-learning and on-policy Sarsa is suggested as a deep learning approach without complicated reward functions, AI techniques, integrated ensemble algorithms, or actor–critic baselines. We merely consider how to combine online and offline environments in a simple manner to allow quick convergence and improve the final performance. Furthermore, the proposed algorithm is inspired by the recent research conducted on open-review-net [24]. Some reviews on open-review-net [24] indicate that it is not easy to design a fundamentally proper reward function. Moreover, despite a well-designed reward, it is not easy to avoid local optimization. Occasionally, the result may become unstable and difficult to reproduce [3].

We simply rely on the “baseline” of the probability δ_1 in terms of balancing the online and offline environments and the probability δ_2 for off-policy Q-learning and on-policy Sarsa in an online real-time environment for a deep learning structure. For probability, δ_1 , $\delta_{1_{ini}} = 0.99$, and $\delta_{1_{fin}} = 0.01$, for each step, $\Delta\delta_1$, $\Delta\delta_1 = \delta_{1_{ini}} - (\delta_{1_{ini}} - \delta_{1_{fin}})/N$, where N is the number of total episodes, we continue to use the offline MC in the initial learning stage, where agents know little about the environment. As the learning process progresses, we are more likely to use online TD of off-policy Q-learning and on-policy Sarsa. While achieving a more accurate expected value approximation, the following remains true: the larger the number of samples, the more accurate the value function that can be found. There is a small number of samples in the early stages; therefore, an agent can wait until the end of an episode before a return is known. Consequently, MC works satisfactorily for an episodic environment and learns from complete sequences. During the progression of learning, deep learning operates satisfactorily as a value approximation, and TD is more likely to be used during the late stages because an AI agent expects the approximation better during such stages, and the lengths of the episodes increase over time. For the on- and off-policy, we set the probability δ_2 similar to the probability δ_1 . For probability δ_2 , $\delta_{2_{ini}} = 0.99$ and $\delta_{2_{fin}} = 0.01$, for each step $\Delta\delta_2$, $\Delta\delta_2 = \delta_{2_{ini}} - (\delta_{2_{ini}} - \lambda_{fin})/N$, where N is the number of total training steps per episode, we continue to use off-policy Q-learning in the initial learning stage. Based on the δ_2 -greedy algorithm, the agent continues with Q-learning in the beginning. Over time, the agent can employ Sarsa for faster convergence. The offline MC is an on-policy control method. The proposed approach starts with on-policy (MC) in small episodes and then moves to both on-policy and off-policy (TD). This is advantageous because the agent makes the best use of an on-policy towards the achievement of an on- and off-policy for breaking the local optimum. Moreover, the ϵ -greedy algorithm can be adaptable for both a random policy and the balance between Q-learning and Sarsa, such as $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$ [20] or $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$ [21]. Based on the ϵ -greedy algorithm [18,19], the agent can achieve the balance between exploration and exploitation. The proposed structure of a DQN integrated with balancing both an

offline environment with probability δ_1 and an on- and off-policy of an online environment with the probability δ_2 is presented in Section 3.2.

3.2. Algorithm Description

In the proposed Algorithm 1, Offline–Online in DQN, the agent follows a ϵ -greedy policy for selecting actions according to the Q-value. The ϵ -greedy algorithm was originally applied to an exploration for which it can be established to guarantee a minimization of the local optima [1,18,19]. The Offline–Online in DQN exploits well-known technologies known as experience replay [18,19] in a dataset D and target network [18,19] with different weights θ^* . Moreover, we employ another experience replay Ω for offline MC, where the tuples of experience are pooled in the form of $\{s_t, r_t, done\}$. By utilizing both D and Ω , the behavior distribution is utilized over numerous actions, smoothing out from offline learning to online learning, or actually, from an on-policy (MC) to an on- and off-policy (Q-learning and Sarsa) to avoid divergence in the parameters. In this research [9], an MC based on the control of an on-policy can assist the removal of the target network because an MC update cannot diverge because the target is computed directly from the trajectories rather than the bootstraps. However, an on-policy-based MC which suffers from an exploration might negatively skew the Q-value estimates. Such an MC update is reasonable when the estimates of the next state in a neural network are inaccurate, particularly when an agent begins learning. We use the target network for off-policy Q-learning and on-policy Sarsa to address the issue, because on-policy MC updates are computed. Similar to [18,19], for every C step, we copy the weights of the target network. We can exploit the target network to overcome oscillations in the update of Q-learning for more stability. The Offline–Online in DQN takes advantage of Sarsa to reduce overestimations owing to the fact that Sarsa follows a certain strategy rather than making an exploration toward the end of the learning [10]. The Offline–Online in DQN utilizes the probability δ_1 for combining offline MC with online TD and the probability δ_2 for off-policy Q-learning and on-policy Sarsa of TD. The approach introduces combinations of online and offline environments for an exploration–exploitation trade-off with a balance between the bias and variance. The purpose is to take advantage of the basic and lighter version of the balance on DNNs as a function approximates in a real-time environment. We adopt the probability δ_1 and the probability δ_2 strategies without taking advantage of the actor–critic [8,14] or pre-trained models [5]. To provide a simple yet powerful strategy, we consider only the fact that probability δ_1 and probability δ_2 are applicable to cases that are more general. Although previous studies [5,8,14] have been mostly suitable theoretically and empirically, they are not always generally suitable for real-world online learning with dynamic and partially observable settings.

More specifically, in line 4, every episode has many training steps. In line B, in every training step, the agent is trained through both online learning and offline learning with probability δ_1 . In line iv of online learning, the transition tuples $(\Phi_t, a_t, r_t, \Phi_{t+1})$ are stored in D , and in line iv of offline learning, the transition tuples $(s_t, r_t, done)$ are stored in Ω . In line ①, based on the probability δ_2 , the agent is trained by both off-policy Q-learning and on-policy Sarsa of TD. In line C, based on the probability δ_1 , the agent is trained using MC.

Algorithm 1 Offline-Online in DQN

-
1. Initialize replay memories D for online learning and Ω for offline learning
 2. Initialize action-value function Q with random weights θ
 3. Initialize target action-value function \hat{Q} with weight $\hat{Q} = \theta$
 4. For every episode do
 - A Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\Phi_1 = \Phi(s_1)$
 - B For every training-step do
 - I If the probability δ_1 (online),
 - i. If ϵ -greedy, select a random action a_t Else, select $a_t = \operatorname{argmax}_a ((s_t), a; \theta)$
 - ii. Execute an action a_t in an emulator and observe reward r_t and image x_{t+1}
 - iii. Set $s_{t+1} = s_t, a_t, x_{t+1}$, and preprocess $\Phi_{t+1} = \Phi(s_{t+1})$
 - iv. Store transition $(\Phi_t, a_t, r_t, \Phi_{t+1})$ in D
 - v. Sample random mini-batch of transitions $(\Phi_j, a_j, r_j, \Phi_{j+1})$ in D
 - vi. If terminal Φ_{j+1} , Set $y_j = r_j$
Else, if non-terminal Φ_{j+1} ,
 - Ⓛ If the probability δ_2 (off-policy), Set $y_j = r_j + \gamma Q(\Phi_{j+1}, \operatorname{argmax}_a Q(\Phi_{j+1}, a; \theta_t); \theta_t)$
Else (on-policy), set $y_j = r_j + (\Phi_{j+1}, a_{j+1}, \theta_t)$
 - vii. Perform a gradient descent step on $(y_j - Q(\Phi_j, a_j; \theta_j))^2$ with respect to θ_t
 - viii. Every C steps, reset $\hat{\theta} = \theta$
 - II Else (offline),
 - i. Select $a_t = \operatorname{argmax}_a Q(\Phi(s_t), a; \theta)$
 - ii. Execute action a_t in an emulator and observe reward r_t and image x_{t+1}
 - iii. Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\Phi_{t+1} = \Phi(s_{t+1})$
 - iv. Store transition $(s_t, r_t, done)$ in Ω
 - C If s is not visited in all samples in Ω based on the probability δ_1
 - I. $G_t = \gamma^*(r + G_t)$
 - II. $\hat{v}(s) = v(s) + \alpha(G_t - v(s))$
 - D Clear all samples in Ω
-

4. Evaluation and Results

We augmented a full episode of transition tuples into the replay memory Ω and compute backward the tuples for online MC and in D for TD, where all samples are bootstrapped. With the probability δ_1 , the targets present a way for a bias–variance tradeoff between online MC and offline TD. Moreover, with the probability δ_2 , the proposed algorithm makes the Q -value estimates reduce the overestimates and not diverge to infinity as the updates continually grow. We selected a few classic control tasks in OpenAI Gym [16] for the comparisons of the proposed offline-online in DQN algorithm with DQN, DQN with MC, and DQN with Sarsa. We implemented these four algorithms with PYTHON Tensorflow and Keras [25,26]. First, we compared the offline–online approaches in the DQN algorithm with DQN, DQN with MC, and DQN with Sarsa on a Cart-Pole [27], the most used control task for RL algorithms. Next, we conducted experiments on MountainCar [28]. Finally, we tested the offline–online approach using the DQN algorithm on LunarLander [29], a more complex task than Cart-Pole and MountainCar. We exploited a function approximation, such as an artificial neural network, for the four algorithms. For the experimental setup, the first dense layer has four inputs, 24 outputs, and a ReLU activation function [30]. The second dense layer has 24 inputs, 24 outputs, and ReLU. The third has 24 inputs, 24 outputs, and a linear function. The loss of the model is

a mean square error, and the optimizer is an adaptive moment estimation (Adam) [31]. For the hyper-parameters, the discount factor, $\gamma = 0.95$, the learning rate, $\alpha = 0.001$, $\varepsilon_{\max} = 1.0$, $\varepsilon_{\min} = 0.01$, $\varepsilon_{\text{decay}} = 0.995$, the bootstrapped Mini-Batch = 64, and the target network parameter update, $C = 300$. For the offline-online in DQN, the probability $\delta 1$ for offline and online environments are as follows: The initial value of probability $\delta 1$ is $\delta 1_{\text{ini}} = 0.99$. For probability $\delta 1$, each step is decreased by $\Delta \delta 1$ until it equals $\delta 1_{\text{fin}} = 0.01$, where $\Delta \delta 1 = \delta 1_{\text{ini}} - (\delta 1_{\text{ini}} - \delta 1_{\text{fin}})/N$, and N refers to the total number of episodes. Likewise, the probability $\delta 2$ for an on- and off-policy are as follows: The initial value of probability $\delta 2$ is $\delta 2_{\text{ini}} = 0.99$. For probability $\delta 2$, each step is decreased by $\Delta \delta 2$ until it equals $\delta 2_{\text{fin}} = 0.01$, where $\Delta \delta 2 = \delta 2_{\text{ini}} - (\delta 2_{\text{ini}} - \delta 2_{\text{fin}})/N$, where N refers to the training steps.

4.1. Cart-Pole Balancing

In Cart-Pole [27], there are four observations and two discrete actions, as shown in Figure 7.

(a) Environment Observation

Number	Observation	Minimum	Maximum
0	Cart Position	-2.4	+2.4
1	Cart Velocity	-Inf	+Inf
2	Pole Angle	$\sim -41.8^\circ$	$\sim +41.8^\circ$
3	Pole Velocity At Tip	-Inf	+Inf

(b) Actions

Number	Action
0	Push Cart to the Left
1	Push Cart to the Right

Figure 7. Environment of Cart-Pole [27].

A pole is attached to a cart, which goes back and forth from left to right. The pole starts upright. The goal is to not fall over when increasing and decreasing the velocity of the cart. A reward of +1 is considered by the environment for every step when the pole remains upright until the next action is terminated. When the episode is terminated, the angle of the pole is between -12° and $+12^\circ$, the cart position is between -2.4 and $+2.4$, or the length of the episode is greater than 200. It is considered solved when the average reward is greater than or equal to 195.0 over 100 consecutive runs [27].

Figure 8 shows the average rewards of the four algorithms, DQN, DQN with Sarsa, DQN with MC, and the proposed DQN with MC and Sarsa on Cart-Pole [27]. From these results, we can observe that DQN with only offline MC and the proposed DQN with MC and Sarsa have guaranteed better rewards and converged sooner in the final goal than the other two algorithms. This might indicate that the proposed algorithm can work without an on-policy TD or Sarsa. However, Cart-Pole balancing is simpler than the other tasks.

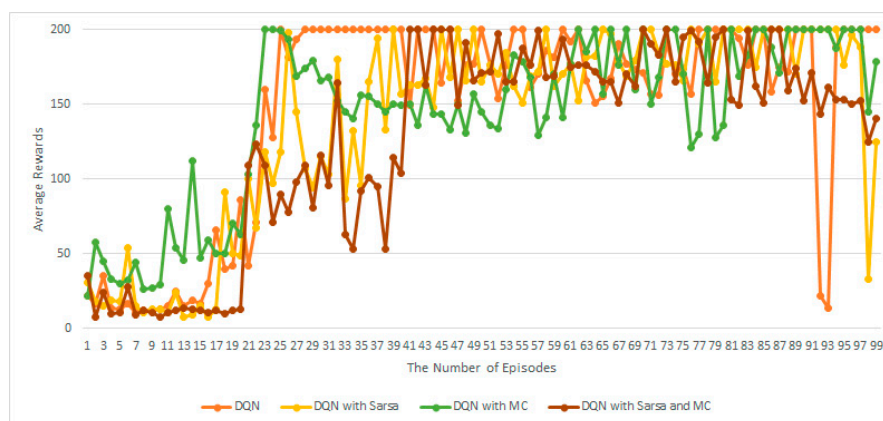


Figure 8. Average rewards of four algorithms.

4.2. Mountain Car

For the MountainCar [28], there are two observations and three discrete actions, as shown in Figure 9.

(a) Environment Observation

Number	Observation	Minimum	Maximum
0	Position	-1.2	+0.6
1	Velocity	-0.07	+0.07

(b) Actions

Number	Action
0	Push Left
1	No Push
2	Push Right

Figure 9. Environment of MountainCar [28].

A car goes back and forth between a “left” mountain and a “right” mountain. The goal is to drive up the “right” mountain. The car is not strong enough to go up the “right” mountain without building up momentum, however. Thus, it goes back and forth without a break to create the momentum. A reward of -1 is given for every step when the position is reached at the half-point between the “left” and “right” mountains. The episode is terminated if a 0.5 position of the height of the mountain is reached, or the number of iterations is more than 200. It is considered solved when it obtains an average reward of $+110.0$ over 100 consecutive runs.

Figure 10 shows the average rewards of the four algorithms, DQN, DQN with Sarsa, DQN with MC, and the proposed DQN with MC and Sarsa on MountainCar [28]. From these results, we can observe that DQN with only offline MC and the proposed DQN with MC and Sarsa have guaranteed improved rewards similar to the results of Cart-Pole balancing in Figure 8. The results show that only the balance in offline MC and online Q-learning in DQN, without the balance in off-policy Q-learning and on-policy Sarsa, might work in both Cart-Pole and MountainCar.

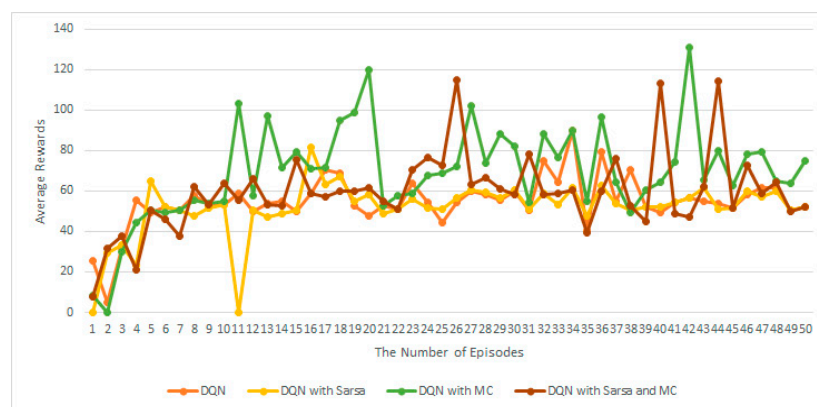


Figure 10. Average rewards of four algorithms.

4.3. LunarLander

In the LunarLander [29], there are three observations and four discrete actions, as indicated in Figure 11. LunarLander as a Box2D-based game was developed by Atari. The landing pad is at the coordinates $(0, 0)$. An approximately 100 to 400 point reward is given by the environment from the top to the landing pad and at zero speed. If the lander moves away from landing pad, the reward is taken away. When the episode is terminated if the lander crashes, then the agent receives -100 or sits down,

and then it receives +100. If each leg contacts the ground, it receives +10. In addition, it is possible to land outside the landing pad. It is considered solved when it obtains an average reward of 200 over 100 consecutive runs.

(a) Environment Observation

Number	Observation	Direction	
0	Throttle(Down)	From the top	To landing pad
1	Throttle(Left)	Between the two flags	
2	Throttle(Right)	Between the two flags	

(b) Actions

Number	Action(Discrete)
0	Do nothing
1	Fire left engine
2	Fire down engine
3	Fire right engine

Figure 11. Environment of LunarLander [29].

Figure 12 shows the average rewards of four algorithms: DQN, DQN with Sarsa, DQN with MC, and the proposed DQN with MC and Sarsa on LunarLander [29]. From these results, we can see that the proposed DQN with MC and Sarsa achieves the most stable performance for LunarLander, which is more complex than Cart-Pole or MountainCar. However, the results show that our proposed algorithm and DQN with MC achieved a better performance. Thus, we attempt to make more comparisons with only the proposed algorithm and DQN with an MC based on the environments with some time-constraints. The results are described in Section 4.4.

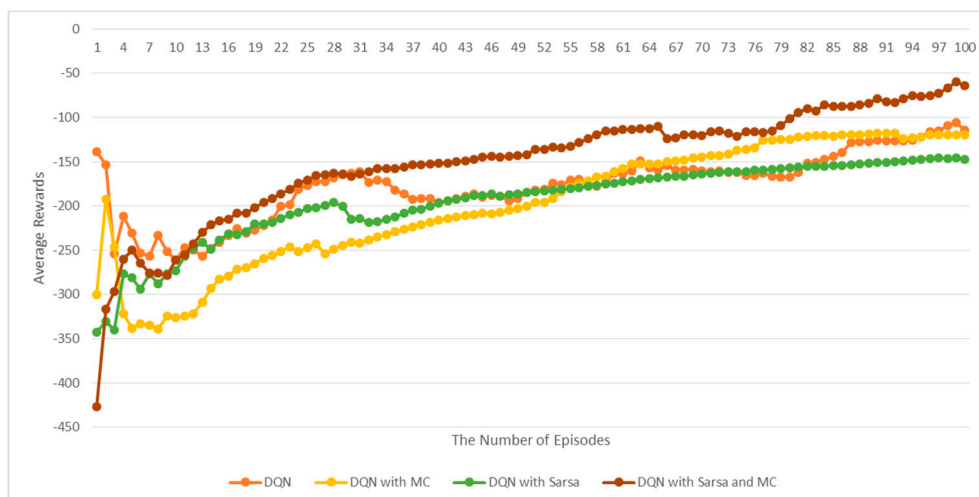


Figure 12. Average rewards of four algorithms.

4.4. Cart-Pole and MountainCar with Qualified Upper Bound

The maximum episode length of Cart-Pole is 500. For a better comparison of the solutions, we consider a few requirements between only DQN with MC and the proposed offline-online in DQN because the two algorithms show similar performances in Cart-Pole and MountainCar but not in LunarLander. The first reason for this is that they receive a reward of -100 when they fall down prior to the maximum length of the episode. The other is that, if the average reward is more than 490 or equal to 500 over 100 consecutive runs, then the loop is terminated. These time constraints in these comparisons are based on the previous implementation [32], which suggests more restrictive time

conditions. Moreover, based on empirical results, we adjust a small parameter between off-policy Q-learning and on-policy Sarsa. We followed the strategies of [10] between balancing Q-learning and Sarsa. However, in the experiment in this section, we change the parameter slightly, i.e., if $r + 1$ is between zero and a small constraint, which is selected based on many empirical trials, then we follow the immediate policy. The results of offline–online balance use in DQN show that Q-value divergence in the constraints is preventable. Figures 13–15 indicate that offline–online balance use in DQN is worth considering for simple control tasks, such as Cart-Pole. We evaluated only 100 rounds owing to the simple environment.

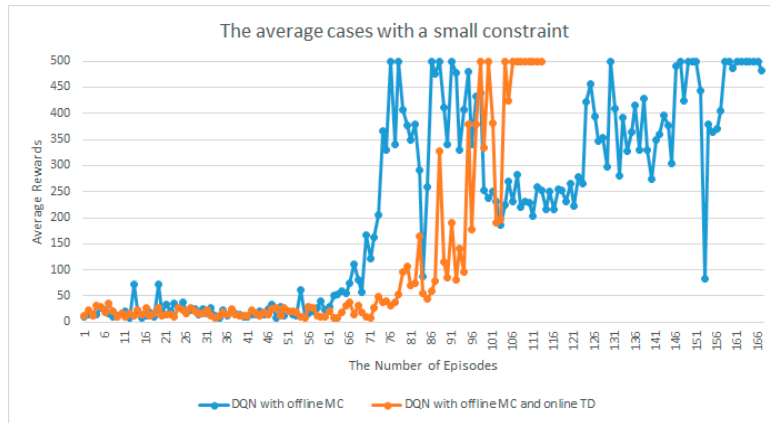


Figure 13. Average comparisons with DQN with MC and the proposed DQN with MC and TD.

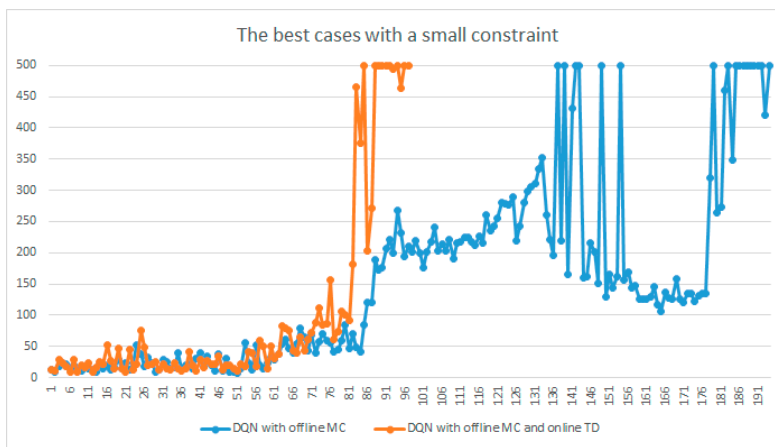


Figure 14. Best-case comparisons with DQN with MC and the proposed DQN with MC and TD.

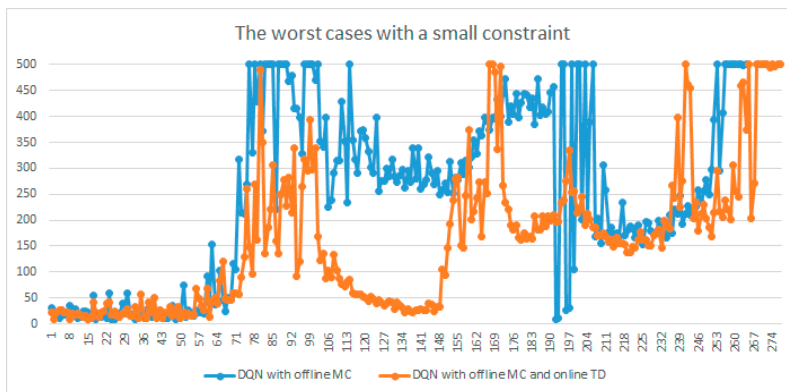


Figure 15. Worst-case comparisons with DQN with MC and the proposed DQN with MC and TD.

In MountainCar, we used the phrase “Car has reached the goal” in each round for better comparisons between the proposed algorithm and DQN with MC. It is considered solved when it obtains an average reward of +110.0 over 100 consecutive runs. In addition, we set the reward +150 to “Car has reached its goal.” We check how many times we can use “Car has reached the goal” in a limited number of runs. These qualified constraints in this comparisons are based on the previous implementation [33]. Figure 16 shows the proposed algorithm uses “Car has reached the goal” 42 times, while DQN with MD uses it 23. They show that the number of “Car has reached the goal” outputs of the proposed algorithm is higher than that obtained by DQN with MC. Under the strict constraints for comparisons of qualification, we demonstrated that the algorithm learns slightly faster than DQN with MC and can balance out the bias–variance trade-off during the training process. It is natural that if we use a transferred model, such as in [5], the proposed model could become faster during the training. However, the pre-trained model in [5] could fail in a variety of minor perturbations [3], specifically in a real-time environment. These results might indicate that the offline–online in DQN can check the performance in terms of the balance in offline MC and online TD. Moreover, it can provide better results when it comes to constrains such as real-world settings.

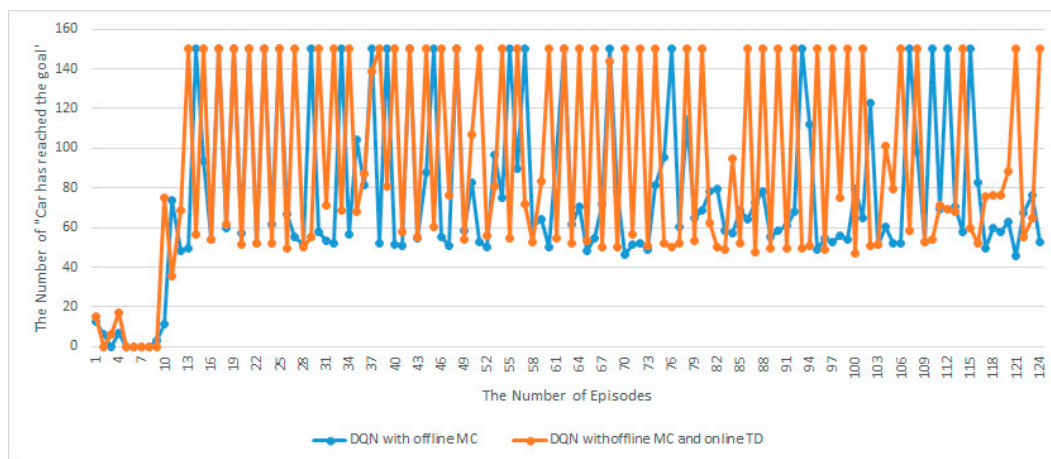


Figure 16. Multiple “the car has reached the goal” comparisons with DQN with MC and the proposed DQN with MC and TD.

5. Conclusions

We proposed balancing offline MC and online TD with on-policy Sarsa and off-policy Q-learning based on the probabilities of empirical experience to achieve reasonable performance and stability in a DQN, while pursuing real-time online learning criteria in the environment. The balance of use between online and offline is intended to see how well the RL can handle the issues of the tradeoff of bias–variance. We only considered a DQN because it is the de facto standard. Q-learning in a DQN with the ϵ -greedy algorithm can also balance the exploration and exploitation, which provides another powerful strategy for achieving the bias–variance tradeoff, a crucial element in several components of RL. This study aims to bridge the gap between theory and practice with a simple strategy of the “baseline” of balancing offline MC and online TD with off-policy Q-learning and on-policy Sarsa in a real-time environment. Based on these considerations, we propose a random probability that decides whether to use an offline or online environment during the learning process. Based on the random probability, we are more likely to utilize online learning with an on- and off-policy during the later stage. As the training of the agent progresses, the entire episode is not needed for offline learning. In the simulations on OpenAI Gym, we demonstrated that, in a simple control task, the balance in online and offline without on- and off-policy shows satisfactory results. However, in a complex task, the proposed algorithm shows a direct evidence of an improvement in the convergence speed and

performance. Therefore, we suggest that the proposed algorithm is worthy of consideration in most real-time environments with time constraints.

Funding: This research was funded by expert fee of SSiS.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Lei, L.; Tan, Y.; Zheng, K.; Liu, S.; Zhang, K.; Shen, X. Deep Reinforcement Learning for Autonomous Internet of Things: Model, Applications and Challenges. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 1722–1760. [CrossRef]
2. Amiranashvili, A.; Dosovitskiy, A.; Koltun, V.; Brox, T. TD OR NOT TD: Analyzing the Role of Temporal Differencing in Deep Reinforcement Learning. *arXiv* **2018**, arXiv:1806.01175.
3. Marcus, G. Deep Learning: A Critical Appraisal. *arXiv* **2019**, arXiv:1801.00631.
4. Naveed, M.; Kitchin, D.; Crampton, A.; Chrapa, L.; Gregory, P. A Monte-Carlo path planner for dynamic and partially observable environments. In Proceedings of the 2012 IEEE Conference on Computational Intelligence and Games (CIG), Granada, Spain, 11–14 September 2012; pp. 211–218. [CrossRef]
5. Sharma, J.; Andersen, P.; Granmo, O.; Goodwin, M. Deep Q-Learning with Q-Matrix Transfer Learning for Novel Fire Evacuation Environment. In *IEEE Transactions on Systems, Man, and Cybernetics: Systems*; IEEE: New York, NY, USA, 2020; pp. 1–19. [CrossRef]
6. Badreddine, S.; Spranger, M. Injecting Prior Knowledge for Transfer Learning into Reinforcement Learning Algorithms using Logic Tensor Networks. *arXiv* **2019**, arXiv:1906.06576.
7. Mocanu, E. On-Line Building Energy Optimization Using Deep Reinforcement Learning. *IEEE Trans. Smart Grid* **2019**, *10*, 3698–3708. [CrossRef]
8. Gu, S.; Lillicrap, T.; Ghahramani, Z.; Turner, R.; Scholkopf, B.; Levine, S. Interpolated Policy Gradient: Merging On-Policy and Off-Policy Gradient Estimation for Deep Reinforcement Learning. In *Advances in Neural Information Processing Systems*; NIPS: San Diego, CA, USA, 2017.
9. Hausknecht, M.; Stone, P. On-Policy vs. Off-Policy Updates for Deep Reinforcement Learning. In *Deep Reinforcement Learning: Frontiers and Challenges, IJCAI 2016 Workshop*; AAAI Press: New York, NY, USA, 2016.
10. Xu, Z.; Cao, L.; Chen, X.; Li, C.; Zhang, Y.; Lai, J. Deep Reinforcement Learning with Sarsa and Q-Learning: A Hybrid Approach. *IEICE Trans. Inf. Syst.* **2018**, *E101.D*, 2315–2322. [CrossRef]
11. Wang, Y.-H.; Li, T.-H.; Lin, C.-J. Backward Q-learning: The combination of Sarsa algorithm and Q-learning. *Eng. Appl. Artif. Intell.* **2013**, *26*, 2184–2193. [CrossRef]
12. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.; Veness, J.; Bellemare, M.; Graves, A.; Riedmiller, M.; Fidjeland, A.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef] [PubMed]
13. Domingos, P. A few useful things to know about machine learning. *Commun. ACM* **2012**, *55*, 78–87. [CrossRef]
14. Chen, G. Merging Deterministic Policy Gradient Estimations with Varied Bias-Variance Tradeoff for Effective Deep Reinforcement Learning. *arXiv* **2019**, arXiv:1911.10527.
15. Mnih, V.; Badia, A.P.; Mirza, M.; Graves, A.; Lillicrap, T.P.; Harley, T.; Silver, D.; Kavukcuoglu, K. Asynchronous Methods for Deep Reinforcement Learning. In Proceedings of the International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016; pp. 1928–1937.
16. OpenAI Gym. Available online: <https://gym.openai.com/> (accessed on 19 May 2020).
17. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, UK, 1998; Volume 1.
18. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv* **2013**, arXiv:1312.5602.
19. Silver, D.; Huang, A.; Maddison, C.J.; Guez, A.; Sifre, L.; Driessche, G.V.D.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M. Mastering the game of go with deep neural networks and tree search. *Nature* **2016**, *529*, 484–489. [CrossRef] [PubMed]
20. Rummery, G.A.; Niranjan, M. *Online Q-Learning Using Connectionist Systems*; University of Cambridge, Department of Engineering: Cambridge, UK, 1994.
21. Watkins, C.J.C.H.; Dayan, P. Q-learning. *Mach. Learn.* **1992**, *8*, 279–292. [CrossRef]
22. Audibert, J.; Munos, R.; Szepesvári, C. Exploration–exploitation tradeoff using variance estimates in multi-armed bandits. *Theor. Comput. Sci.* **2009**, *410*, 1876–1902. [CrossRef]

23. Konar, A.; Chakraborty, I.G.; Singh, S.J.; Jain, L.C.; Nagar, A.K. A Deterministic Improved Q-Learning for Path Planning of a Mobile Robot. *IEEE Trans. Syst. Man Cybern. Syst.* **2013**, *43*, 1141–1153. [CrossRef]
24. Available online: <https://openreview.net/forum?id=ByBAI2eAZ> (accessed on 19 May 2020).
25. Tensorflow. Available online: <https://github.com/tensorflow/tensorflow> (accessed on 19 May 2020).
26. Keras. Available online: <https://keras.io/> (accessed on 19 May 2020).
27. Cart-Pole. Available online: <https://gym.openai.com/envs/CartPole-v1/> (accessed on 19 May 2020).
28. MountainCar. Available online: <https://gym.openai.com/envs/MountainCar-v0/> (accessed on 19 May 2020).
29. LunarLander. Available online: <https://gym.openai.com/envs/LunarLander-v2/> (accessed on 19 May 2020).
30. Hahnloser, R.; Sarpeshkar, R.; Mahowald, M.A.; Douglas, R.J.; Seung, H.S. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature* **2000**, *405*, 947–951. [CrossRef] [PubMed]
31. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2015**, arXiv:1412.6980.
32. Available online: https://github.com/rlcode/reinforcement-learning-kr/blob/master/2-cartpole/1-dqn/cartpole_dqn.py (accessed on 19 May 2020).
33. Available online: <https://github.com/shivaverma/OpenAIGym/blob/master/mountain-car/MountainCar-v0.py> (accessed on 19 May 2020).

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



© 2020 by the author. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).