frontiers in Neurorobotics

ORIGINAL RESEARCH
published: 10 December 2019
doi: 10.3389/fnbot.2019.00103

Constrained Deep Q-Learning Gradually Approaching Ordinary Q-Learning

Shota Ohnishi[1], *Eiji Uchibe*[2]*, *Yotaro Yamaguchi*[3], *Kosuke Nakanishi*[4], *Yuji Yasui*[4] and *Shin Ishii*[2,3]

[1] Department of Systems Science, Graduate School of Informatics, Kyoto University, Now Affiliated With Panasonic Co., Ltd., Kyoto, Japan, [2] ATR Computational Neuroscience Laboratories, Kyoto, Japan, [3] Department of Systems Science, Graduate School of Informatics, Kyoto University, Kyoto, Japan, [4] Honda R&D Co., Ltd., Saitama, Japan

A deep Q network (DQN) (Mnih et al., 2013) is an extension of Q learning, which is a typical deep reinforcement learning method. In DQN, a Q function expresses all action values under all states, and it is approximated using a convolutional neural network. Using the approximated Q function, an optimal policy can be derived. In DQN, a target network, which calculates a target value and is updated by the Q function at regular intervals, is introduced to stabilize the learning process. A less frequent updates of the target network would result in a more stable learning process. However, because the target value is not propagated unless the target network is updated, DQN usually requires a large number of samples. In this study, we proposed Constrained DQN that uses the difference between the outputs of the Q function and the target network as a constraint on the target value. Constrained DQN updates parameters conservatively when the difference between the outputs of the Q function and the target network is large, and it updates them aggressively when this difference is small. In the proposed method, as learning progresses, the number of times that the constraints are activated decreases. Consequently, the update method gradually approaches conventional Q learning. We found that Constrained DQN converges with a smaller training dataset than in the case of DQN and that it is robust against changes in the update frequency of the target network and settings of a certain parameter of the optimizer. Although Constrained DQN alone does not show better performance in comparison to integrated approaches nor distributed methods, experimental results show that Constrained DQN can be used as an additional components to those methods.

Keywords: deep reinforcement learning, deep Q network, regularization, learning stabilization, target network, constrained reinforcement learning

OPEN ACCESS

Edited by:
Hong Qiao,
University of Chinese Academy of Sciences, China

Reviewed by:
Jiwen Lu,
Tsinghua University, China
David Haim Silver,
Independent Researcher, Haifa, Israel
Timothy P. Lillicrap,
Google, United States

Correspondence:
Eiji Uchibe
uchibe@atr.jp

Received: *17 December 2018*
Accepted: *27 November 2019*
Published: *10 December 2019*

Citation:
Ohnishi S, Uchibe E, Yamaguchi Y, Nakanishi K, Yasui Y and Ishii S (2019) Constrained Deep Q-Learning Gradually Approaching Ordinary Q-Learning.
Front. Neurorobot. 13:103.
doi: 10.3389/fnbot.2019.00103

1. INTRODUCTION

In recent years, considerable research has focused on deep reinforcement learning (DRL), which combines reinforcement learning and deep learning. Reinforcement learning is a framework for obtaining a behavioral policy that maximizes value through trial and error, even under unknown conditions. Deep learning is a method of high-level pattern recognition, and it has demonstrated efficient performance in a variety of image-processing problems. DRL has been applied to various optimization problems, such as robot (Levine et al., 2016) and drone control (Kahn et al., 2017) and game learning (Mnih et al., 2013). Alpha Go (Silver et al., 2016) is one of the most well-known

applications of DRL. Alpha Go was created in 2016, and it later defeated Lee Sedol (9 dan, or 9th level) and Ke Jie (9 dan), the world's best Go players. The newest version of Alpha Go outplays the previous version without prior knowledge of the positions of the stones or historical records of human actions during play (Silver et al., 2017). However, use of DRL still faces several unresolved problems. These problems include the requirement for a very large number of samples, the inability to plan a long-term strategy, and the tendency to perform risky actions in actual experiments.

The original DRL is a Deep Q Network (DQN) (Mnih et al., 2013, 2015) proposed by Google Deep Mind, which learned to play 49 different Atari 2600 games simply through a game screen. Q learning (Watkins and Dayan, 1992; Sutton and Barto, 1998) is a typical reinforcement learning method. In Q learning, an optimal action policy is obtained after learning an action value function (a.k.a. Q function). DQN uses a convolutional neural network (CNN) to extract features from a screen and Q learning to learn game play. Considerable research has been conducted on expanded versions of DQN, such as double Q learning (van Hasselt, 2010; van Hasselt et al., 2016) that reduces overestimation of the action values, prioritized experience replay (Schaul et al., 2015), which gives priority to experience data used for learning, dueling network architecture (Wang et al., 2016), which outputs action values from state values and advantage values, and the asynchronous learning method with multiple agents (Mnih et al., 2016). Rainbow DDQN (Hessel et al., 2017) combines several DQN extensions: Double DQN, prioritized experience replay, dueling network, multi-step bootstrap targets, Noisy Net (Fortunato et al., 2018) that injects noise into the networks' weights for exploration, and Distributional DQN that models the distribution whose expectation is the action value. Ape-X DDQN (Horgan et al., 2018) is a distributed DQN architecture, as in which distributed actors are separated from the value leaner, and it employs Double DQN, dueling network architecture, distributed prioritized experience replay, and multi-step bootstrap targets. Recurrent Replay Distributed DQN (R2D2) is one of the state-of-the-art distributed architecture that proposes distributed prioritized experience replay when the value function is approximated by recurrent neural networks (Kapturowski et al., 2019).

When attempting to find the optimal Q function within a class of non-linear functions, such as neural networks, learning becomes unstable or in some cases does not converge (Tsitsiklis and Van Roy, 1997). In DQN, learning is stabilized through a heuristic called experience replay (Lin, 1993) and the use of a target network. Experience replay is a technique that saves time-series data in a buffer called replay memory. In experience replay, mini batch learning is performed using randomly sampled data from the buffer. Consequently, the correlations between the training samples are reduced, and thus the learning is stabilized. The target network is a neural network, which is updated with a slower cycle for the neural network representing the Q function. By using a fixed target network to calculate the target value, we can expect stabilization of the entire learning process. In general, a less frequent updates of the target network would result in a more stable learning process. For example, Hernandez-Garcia (2019) and Hernandez-Garcia and Sutton (2019) reported that decreasing the update frequency from 2,000 to 500 steadily reduced the instability of the algorithm. van Hasselt et al. (2016) increased the update frequency of the target network from 10,000 to 30,000 to reduce overestimation of the action values. It is known that using the target network technique disrupts online reinforcement learning and slows down learning because the value is not propagated unless the target network is updated (Lillicrap et al., 2016; Kim et al., 2019). Consequently, the number of samples required for learning becomes extremely large.

To stabilize the learning processes in DQN, Durugkar and Stone (2017) proposed Constrained Temporal Difference (CTD) algorithm to prevent the average target value from changing after an update by using the gradient projection technique. The CTD algorithm was validated by showing convergence on Baird's counterexample (Baird, 1995) and a grid-world navigation task, although the CTD algorithm did not require a target network. However, Pohlen et al. (2018) and Achiam et al. (2019) showed that the CTD algorithm did not work well in more complicated problems. Pohlen et al. (2018) proposed the Temporal Consistency loss (TC-loss) method, which tries to prevent the Q function at each target state-action pair from changing substantially by minimizing changes of the target network. Although the use of high discount factors usually leads to propagation of errors and instabilities, it has been shown that DQN with TC-loss can learn stably even with high discount factors.

In this study, we focus on this problem and propose a method with practically improved sample efficiency. In our proposed method, a standard TD error is adopted for bootstrapping, while the difference between the value of the best action of the learning network and that of the target network is used as a constraint for stabilizing the learning process. We call this method Constrained DQN. When the difference between the maximum value of the Q function and the corresponding value of the target network is large, Constrained DQN updates the Q function more conservatively, and when this difference is sufficiently small, Constrained DQN behaves in the manner of Q learning. As learning progresses, the number of times the constraints are activated decreases, and DQN gradually approaches conventional Q learning. Using this method, we expect an acceleration of convergence in the early stage of learning by reducing the delay in updating the Q function, since the target value is calculated without using the target network. In addition, we expect the results to be equivalent to those of Q learning without constraints when learning is completed. We applied our Constrained DQN to several tasks, such as some Atari games and a couple of control problems with a discrete state space and a continuous state space, respectively. Consequently, Constrained DQN converged with fewer samples than did DQN, and it was robust against fluctuations in the frequency of updates in the target network. Although Constrained DQN alone does not show better performance in comparison to integrated approaches, such as Rainbow nor distributed methods like R2D2, experimental results show that Constrained DQN can be used as an additional component to those methods.

2. BACKGROUND

In this section, we give an outline of Q learning, which is a typical method of value-based reinforcement learning and forms the basis of the proposed method. Variants of Q learning are also discussed here.

2.1. Reinforcement Learning

The agent is placed in an unknown environment, observes a state of the environment at each time step, and receives a reward following the selected action. Mapping a state to an action is called a policy. The objective of reinforcement learning is to determine the optimal policy that allows the agent to choose an action that maximizes the expected sum of rewards received in the future. The state of the environment is represented as s, and the agent probabilistically selects an action a based on stochastic policy $\pi(a \mid s) = \Pr(a \mid s)$. After the action selection, the state changes to s' according to the transition probability $\Pr(s' \mid s, a)$, and thus a reward $r(s, a, s')$ is obtained.

When the state transition has a Markov property, this problem setting is called a Markov Decision Process (MDP). On the other hand, the environmental state s may not be observed directly or may only be partially observed. In such a case, the agent must predict state s from observation o. When the mapping of s to o is stochastic, this problem setting is called a Partially Observable Markov Decision Process (POMDP). As a solution to POMDP, we use the history of past observations o as a pseudo state and then apply a reinforcement learning method as if the pseudo state constituted an ordinary MDP. The Atari game, to which DQN was first applied, is a POMDP problem because the current game screen alone cannot uniquely represent the game state. However, in Mnih et al. (2015), the most recent set of four consecutive observations was used as a pseudo state to approximately handle the Atari game play as an MDP.

2.2. Q Learning With Function Approximation

To illustrate the difference between Q learning and DQN, we briefly explain the basic algorithm here. We define the sum of discounted rewards obtained after time t as the return R_t as follows:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k},$$

where $0 \leq \gamma < 1$ is the discount rate for future rewards. The smaller the value of γ, the more emphasis is placed on the reward in the near future. The action value function (Q function) is then defined by

$$Q^\pi(s, a) = E_\pi\{R_t | s_t = s, a_t = a\},$$

where $E_\pi\{...\}$ represents the expectation under stochastic policy π. The Q function $Q^\pi(s, a)$ represents the expected sum of discounted rewards when the agent chooses action a under state s and then selects actions according to policy π. The Q function is described as the following recursive formula:

$$Q^\pi(s, a) = \sum_{s' \in S} \Pr(s' \mid s, a) \left(r(s, a, s') + \gamma \sum_{a' \in A} \pi(a' \mid s') Q^\pi(s', a') \right),$$

where S and A are the state set and the action set, respectively. From this formula, we can determine that the Q function under the optimal policy π^*, i.e., the optimal Q function, satisfies the following equation, which is known as the Bellman optimality equation:

$$Q^*(s, a) = E_{s'}\{r_t + \gamma \max_{a'} Q^*(s', a')\}. \tag{1}$$

In Q learning, by iteratively updating the Q function using (1) based on empirical data, the Q function can be stochastically converged to $Q^*(s, a)$, and so the optimal policy can be determined as the policy that is greedy with respect to Q^*: $a^* = \text{argmax}_a Q^*(s, a)$. In practice, a learning agent has to explore the environment because the Q function is not reliable, and ϵ-greedy action selection has been widely used as a stochastic policy to probabilistically select an action a for an input state s. More specifically, the ϵ greedy policy selects an action that maximizes the Q function at state s with a probability of $1 - \epsilon$, $\epsilon \in [0, 1]$ and chooses a random action with the remaining probability. ϵ was initially set to 1.0 and gradually reduced as learning progressed, and it was fixed after becoming a small value like 0.1. Consequently, at the beginning of learning, various actions were searched at random, and as learning progressed, good actions were selectively performed based on the action value function that had become more reliable.

When the states and actions are discrete and finite, a simple way to represent the Q function is to use a table of values for all pairs of states and actions. The table is arbitrarily initialized and updated with data on the agent experience as follows:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left(r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right),$$

where $0 < \alpha \leq 1$ is the learning rate, and the larger the learning rate, the stronger is the influence of new data for updating. With this learning algorithm, the Q table converges to the optimal Q function under the convergence condition of stochastic approximation. On the other hand, because this is based on the stochastic approximation method, a sufficient number of data for all pairs of (s, a) is required.

In tabular Q learning, when the number of elements in the state or action space is enormous or the state or action space is continuous, we often express the Q function as a parametric function $Q(s, a; \theta)$ using the parameters θ and then update the parameters according to the gradient method:

$$\theta \leftarrow \theta + \alpha \left(\text{target}_Q - Q(s, a; \theta) \right) \nabla_\theta Q(s, a; \theta). \tag{2}$$

Here, "target_Q" is a target value based on the optimal Bellman Equation (1), and it is calculated as follows:

$$\text{target}_Q = r(s, a, s') + \gamma \max_{a'} Q(s', a'; \theta).$$

The Q function is updated according to its self-consistent equation. Q-learning is a bootstrap method, where the Q function approximator is regressed toward the target value, which depends on itself. This implies that the target value changes automatically when the learning network is updated. Therefore, when a non-linear function, such as a neural network is used for approximation, this learning process becomes unstable due to dynamical changes in the target, and in the worst case, the Q function diverges (Sutton and Barto, 1998).

3. EXTENSION OF Q LEARNING

3.1. Deep Q Network

To prevent the Q function from diverging, DQN introduces a separate target network that is a copy of the Q function, and this is used to calculate the target value. In this case, the Q function is updated as follows:

$$\theta \leftarrow \theta + \alpha \left(\text{target}_{\text{DQN}} - Q(s, a; \theta)\right) \nabla_\theta Q(s, a; \theta).$$

Here, $\text{target}_{\text{DQN}}$ is a target value computed by

$$\text{target}_{\text{DQN}} = r(s, a, s') + \gamma \max_{a'} T(s', a'),$$

where $T(s, a)$ represents the target network. Three alternatives can be chosen to update the target network. The first one is periodic update, i.e., the target network is synchronized with the current Q function at every C learning step when the following condition is satisfied:

$$\text{total_steps} \mod C = 0, \quad T \leftarrow Q,$$

where total_steps represents the total number of updates applied to the Q function up to the present time. This method is based on Neural Fitted Q Iteration (Riedmiller, 2005), which is a batch Q learning method that employs a neural network. During the interval from the previous update to the next update of the target network, the learning is supervised wherein the target is given by the immutable network T. The second alternative is symmetric update, in which the target network is updated symmetrically as the learning network, and this is introduced in double Q learning (van Hasselt, 2010; van Hasselt et al., 2016). The third possible choice is Polyak averaging update, where the parameter of the target network is updated by the weighted average over the past parameters of the learning network, and this was used, for example, in Lillicrap et al. (2016). In our experiments, we examined DQN using a periodic update of the target network.

In addition to using the target network, DQN utilizes the previously proposed Experience Replay (ER) (Lin, 1993), which is a heuristic that temporarily stores to memory a record of state transitions during a certain number of steps and randomly selects a data point from memory for learning so that the correlations between samples are reduced and sample efficiency is increased through the reuse of data. Specifically, when the agent selects an action a at a state s and receives a reward r and the state then transits to s', this data point (s, a, r, s') is stored in replay memory D and used for mini-batch learning. In mini-batch learning, the parameters are updated based on a certain number of data points randomly selected from the replay memory D, and this procedure is repeated several times. This makes it possible to prevent stagnation of learning as a result of correlation between data points while maintaining the one-step Markov property. Therefore, the update rule of DQN with ER is given by

$$\theta \leftarrow \theta + \alpha \mathbb{E}_{(s,a,r,s') \sim U(D)} \left[\left(\text{target}_{\text{DQN}} - Q(s, a; \theta)\right) \nabla_\theta Q(s, a; \theta) \right], \tag{3}$$

where $(s, a, r, s') \sim U(D)$ indicates that an experienced sample (s, a, r, s') is drawn uniformly at random from the replay buffer D. The learning process of DQN is more stable than that of Q learning because the update rule of DQN introduces a delay between the time when $Q(s, a; \theta)$ is updated and the time when $T(s, a)$ is updated. Although the use of the target network is critical for stable learning, it hinders fast learning due to this delay.

3.2. Techniques Together With DQN

After the DQN work, there have been additional modifications and extensions such to enhance the speed or stability of DQN. One is the dueling network architecture (Wang et al., 2016) that has a neural network architecture with two parts to produce separate estimates of state-value function $V(s)$ and advantage function $A(s, a)$. More specifically, the action-value function is decomposed as

$$Q(s, a; \theta) = V(s; \theta_V) + \left(A(s, a; \theta_A) - \frac{1}{|A|} \sum_{a'} A(s, a; \theta_A) \right), \tag{4}$$

where θ_V and θ_A are respectively, the parameters of the state-value function and of the advantage function, and $\theta = \{\theta_v, \theta_A\}$. It is experimentally shown that the dueling network architecture converges faster than the conventional single-stream network architecture (Wang et al., 2016; Tsurumine et al., 2019).

Another important technological advance is entropy-based regularization (Ziebart et al., 2008; Mnih et al., 2016) that has been shown to improve both exploration and robustness, by adding the entropy of policy to the reward function. The role of the entropy is to discourage premature convergence and encourage policies to put probability mass on all actions. Soft Q-learning (Haarnoja et al., 2017) is one of off-policy algorithm that maximizes the entropy regularized expected reward objective, and its update rule is given by

$$\theta \leftarrow \theta + \alpha \mathbb{E}_{(s,a,r,s') \sim U(D)} \left[\left(\text{target}_{\text{SQL}} - Q(s, a; \theta)\right) \nabla_\theta Q(s, a; \theta) \right], \tag{5}$$

where $\text{target}_{\text{SQL}}$ is the target value of Soft Q-learning computed by

$$\text{target}_{\text{SQL}} = r(s, a, s') + \frac{\gamma}{\beta} \ln \sum_{a'} \exp\left(\beta T(s', a')\right). \tag{6}$$

Here, β is a predefined hyperparameter. Note that $\ln \sum \exp()$ is a smoothened alternative to the maximum function, and the

target value of Soft Q-learning (6) converges to that of DQN (3) as $\beta \to \infty$.

3.3. Temporal Consistency Loss

Pohlen et al. (2018) pointed out that DQN is still unstable as the discount factor γ approaches 1 because the temporal difference between non-rewarding subsequent states tends to be $\sim$0. This also makes the learning process unstable, especially in long-horizon MDPs, because unnecessary generalization happens between temporally adjacent target values. Fujimoto et al. (2018) pointed out that the variance of the target value can grow rapidly with each update when γ is large. To resolve this instability issue, Pohlen et al. (2018) added the following auxiliary loss function called Temporal Consistency (TC) loss:

$$\mathcal{L}_{\mathrm{TC}}(s', \tilde{a}^*, \theta) = \frac{1}{2}\left(T(s', \tilde{a}^*) - Q(s', \tilde{a}^*; \theta)\right)^2, \tag{7}$$

where $\tilde{a}^* = \mathrm{argmax}_{a'} T(s', a')$. Although Huber loss was adopted in the original paper, L2 loss is used in this paper to make a fair comparison between different methods. The update rule of DQN with TC-loss is given by

$$\theta \leftarrow \theta + \alpha \mathbb{E}_{(s,a,r,s') \sim U(D)}\left[\left(\mathrm{target}_{\mathrm{DQN}} - Q(s, a; \theta)\right) \nabla_\theta Q(s, a; \theta) - \lambda \nabla_\theta \mathcal{L}_{\mathrm{TC}}(\theta)\right],$$

where λ is a predefined positive hyperparameter. Note that Pohlen et al. (2018) used TC-loss together with the transformed Bellman operator, which reduces the target's scale and variance, instead of clipping the reward distribution to the interval $[-1, 1]$; however, we do not adopt the transformed Bellman operator because the hyperparameters of the learning algorithm should be tuned individually for each task.

4. CONSTRAINED DQN

DQN uses a target network to calculate the target required for updating the Q function. The target network T is synchronized with the Q function Q at every learning step. Although this heuristic successfully stabilized the learning process, it was often time-consuming for learning because the value was not propagated unless the target network was updated.

In this study, we present Constrained DQN, which not only calculates the target with the current Q, as in the case of conventional Q learning, but also constrains parameter updates based on the distance between the current Q and the target network as a way to stabilize the learning. When the difference between the outputs of the Q function and the target network is large, Constrained DQN updates its parameters conservatively, as in the case of DQN, but when the difference between the outputs is small, it updates the parameters aggressively, as in the case of conventional Q learning.

In Constrained DQN, the following loss function, which is similar to TC-loss, is considered:

$$\mathcal{L}_{\mathrm{CDQN}}(s', a^*, \theta) = \frac{1}{2}\left(T(s', a^*) - Q(s', a^*; \theta)\right)^2, \tag{8}$$

where $a^* = \mathrm{argmax}_{a'} Q(s', a')$. The difference between Equations (7) and (8) is that our loss function considers the maximum value of the learning network while TC-loss uses that of the target network. Constrained DQN updates the parameters by the standard Q learning algorithm (2) when the following constraint is satisfied:

$$\mathbb{E}_{(s,a,r,s') \sim U(D)}\left[\mathcal{L}_{\mathrm{CDQN}}(s', a^*, \theta)\right] \leq \eta, \tag{9}$$

where η is a positive threshold of the constraint. Otherwise, the loss function (8) is minimized. Consequently, Constrained DQN updates the parameter by

$$\theta \leftarrow \theta + \alpha \mathbb{E}_{(s,a,r,s') \sim U(D)}\left[\left(\mathrm{target}_Q - Q(s, a; \theta)\right) \nabla_\theta Q(s, a; \theta) - \lambda l(s', a^*; \theta, \eta)\right], \tag{10}$$

where $l(s', a^*; \theta, \eta)$ is the gradient of the regularization term, defined by

$$l(s', a^*; \theta, \eta) = \begin{cases} 0 & \text{if } \mathbb{E}_{(s,a,r,s') \sim U(D)}\left[\mathcal{L}_{\mathrm{CDQN}}(s', a^*, \theta)\right] \leq \eta \\ \nabla_\theta \mathcal{L}_{\mathrm{CDQN}} & \text{otherwise.} \end{cases}$$

If the constraint condition is satisfied, then $l(s', a^*; \theta, \eta) = 0$ and the update rule of Constrained DQN is identical to that of Q-learning with experience replay. Similar to DQN with periodic update, the target network is synchronized with the current Q function at every C learning step.

Constrained DQN can be used together with the techniques described before. When the target value is computed by

$$\mathrm{target}_{\mathrm{CSQL}} = r(s, a, s') + \frac{\gamma}{\beta} \ln \sum_{a'} \exp\left(\beta Q(s', a'; \theta)\right), \tag{11}$$

the update rule is interpreted as Constrained Soft Q learning, which can also be seen as Soft Q learning with the inequality constraint (9).

Table 1 gives the frequency of constraint activation for each learning phase and hyperparameter. As learning progresses and approaches convergence, the difference between the Q function and the target network tends to be small, and so the frequency of activating the constraint decreases and the update rule approaches that of ordinary Q learning. As the value of λ increases, the influence of the constraint term increases, and so the update rule becomes conservative as in the case of DQN. As the value of η increases, it becomes more difficult to activate the constraint, and so the update rule approaches that of ordinary Q learning.

TABLE 1 | Frequency of constraint activation for each learning phase and hyperparameter.

Learning phase	Initial phase	Final phase
λ	Big	Small
η	Small	Big
Frequency of activating constraint	High	Low
(update rule)	(like DQN)	(Q learning)

5. EXPERIMENTS RESULTS

In this section, we compare the proposed method and the conventional method using an MNIST maze task, a Mountain-Car task based on OpenAI Gym (Brockman et al., 2016), a robot navigation task, and two Atari games. The state spaces of the MNIST maze, the Mountain-Car, and the robot navigation are a grayscale image, a two-dimensional continuous real-valued vector, and a concatenation of an RGB image and a 360-dimensional LIDAR vector, respectively. The state space of the Atari games is explained later.

5.1. MNIST Maze

5.1.1. Task Setting

The MNIST maze is a 3×3 maze (Elfwing et al., 2016) tiled with handwritten number images taken from the MNIST dataset shown in **Figure 1**. The images in a maze are randomly taken from MNIST for each episode, but the number on each maze square is fixed for all learning episodes. The initial position of the agent is "1" (upper-left square of the maze). In each step, the agent observes the image in which it resides and then chooses an action of up, down, left, or right based on its observation according to the behavioral policy. The agent then moves in its chosen direction in a deterministic manner; however, it is impossible to pass through pink walls, so if the agent selects the direction of going through a pink wall, the movement is ignored and the agent position does not change. If the agent reaches "5" across the green line without going through any red line, a $+1$ reward is given, and if the agent reaches "5" across a red line, a -1 reward is given. The episode ends when the agent reaches "5" or when the agent has performed $1,000$ behavioral steps without reaching "5." In the latter case, a reward of 0 is given. This task requires both MNIST handwritten character recognition and maze search based on reinforcement learning.

FIGURE 1 | MNIST maze task. The agent aims to reach goal "5" on the 3×3 maze by selecting either an up, down, left, or right movement. The lines separating the squares are yellow, green, red, or pink, and they do not change over the episodes. It is impossible to pass through a pink wall, and if the agent selects the direction to a pink wall, the movement is canceled and that agent's position does not change. If the agent reaches "5" across the green line, a $+1$ reward is provided, and if the agent reaches "5" across the red line, a -1 reward is provided. The agent can observe an image of 24×24 pixels in which it resides. The number assignment is fixed for all episodes, but the image for each number is changed at the onset of each episode. For example, the upper left tile is always a "1," but the image of "1" is randomly selected from the training data set of MNIST handwritten digits at the onset of each episode.

We applied Q learning, DQN, DQN with TC-loss, and Constrained DQN to this task to make a comparison. We used a network architecture consisting of the three convolutional layers and two fully connected layers shown in **Figure 2A**. The input dimensionality is the same as that of the MNIST images, and the output dimensionality was four, which indicates the number of possible actions in this task. According to the ϵ greedy method, ϵ was reduced in the first $1,000$ steps. We set the other parameters, such as the size of replay memory and the parameters of the optimizer, RMSProp [more specifically, RMSPropGraves (Graves, 2013) described in **Appendix**], to those used in DQN (Mnih et al., 2015). As for the hyperparameters, λ was set to either 0.5, 1, 2, or 5, and η was set to either 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, or 0. Training takes about 4 h on a single NVIDIA Tesla K40 GPU for each setting.

5.1.2. Results

Figure 3 presents the learning curves of Q learning, DQN, DQN with TC-loss, and Constrained DQN. The best hyperparameters of Constrained DQN are $\lambda = 2, \eta = 10^{-5}$, and $C = 10$, those of DQN with TC-loss are $\lambda = 1$ and $C = 10,000$, and that of DQN is $C = 10,000$. We found that the number of samples required for convergence is smaller in Constrained DQN than in the baselines. We also found that DQN had an unstable learning curve after about 2,000 training episodes. DQN with TC-loss yielded a stable learning curve, but it learned much more slowly than did DQN or Constrained DQN. We did not find Q learning to work in the MNIST maze task. Note here that we consistently used the same network architecture, including CNN, even in the case of Q learning.

Figure 4 presents the average Q value after 50,000 learning steps and the true Q value for each state and action pair. We performed 100 episodes for evaluation, and the Q value averaged over those episodes is shown for each state and action pair. Here, we obtained the true Q values through complete dynamic programming. **Figure 5** illustrates the variance of the Q value after 50,000 learning steps. These figures show that the average Q value estimated by Constrained DQN is almost the same as the true Q value, and the variance of the Q value estimated by Constrained DQN is smaller than that of DQN. Although the learned policy obtained by DQN was optimal, the estimated Q function was far from the true one.

Figure 6 presents the results when we changed the update frequency of the target network. We examined DQN and our Constrained DQN with three hyperparameter settings. DQN did not converge when the update frequency of the target network

A grayscale image (24 x 24)
Conv | 32 | 8x8 | 1
Conv | 64 | 4x4 | 2
Conv | 64 | 3x3 | 1
FC | 512
FC | 4
Q values

B States of the Mountain-Car (2)
FC | 100
FC | 100
Q values

C RGB image (640 x 480 x 3 → 32 x 32 x 3)
Conv | 64 | 7x7 | 64
Conv | 64 | 7x7 | 64
Conv | 64 | 7x7 | 64
Maxpooling | 2x2 | 1x1
Flatten
FC | 256
360° LIDAR (360)
FC | 360
FC | 5
Q values

FIGURE 2 | Network structures for the Q function. **(A)** MNIST maze task. **(B)** Mountain-Car task. **(C)** robot navigation task. Every convolutional layer is represented by its type, channel size, kernel size, and stride size. Other layers are represented by their types and dimensions.

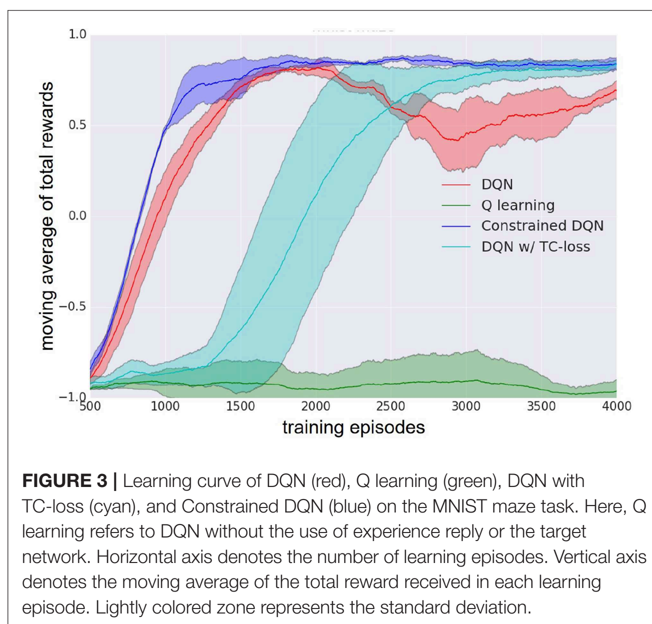

FIGURE 3 | Learning curve of DQN (red), Q learning (green), DQN with TC-loss (cyan), and Constrained DQN (blue) on the MNIST maze task. Here, Q learning refers to DQN without the use of experience reply or the target network. Horizontal axis denotes the number of learning episodes. Vertical axis denotes the moving average of the total reward received in each learning episode. Lightly colored zone represents the standard deviation.

was one learning step (that is, Q learning) or 10 learning steps, but Constrained DQN converged regardless of the setting of λ and η when the update of the target network was performed every 10 learning steps. Since the Q learning did not progress well, the average number of steps per episode was large; this is the reason why the number of episodes was <10,000 even in the later part of the 1,000,000 steps. From this result, we consider Constrained DQN to be more robust to changes in the update frequency of the target network than DQN.

Figure 7 shows the number of times the constraint was activated along the learning steps by Constrained DQN. The figure presents the results for two different settings of hyperparameters η. In both cases, the constraint was activated many times in the early stages of learning, but the number of activations decreased as learning progressed. From this result, it is clear that the constraint was effective in the early learning stages, and the learning was equivalent to the unbiased Q learning in later stages. When η is large, it is easy to satisfy the constraint: the smaller the η, the more the constraint was activated.

Figure 8 is a heatmap of the sum of rewards received throughout learning for each combination of λ, η, and the update

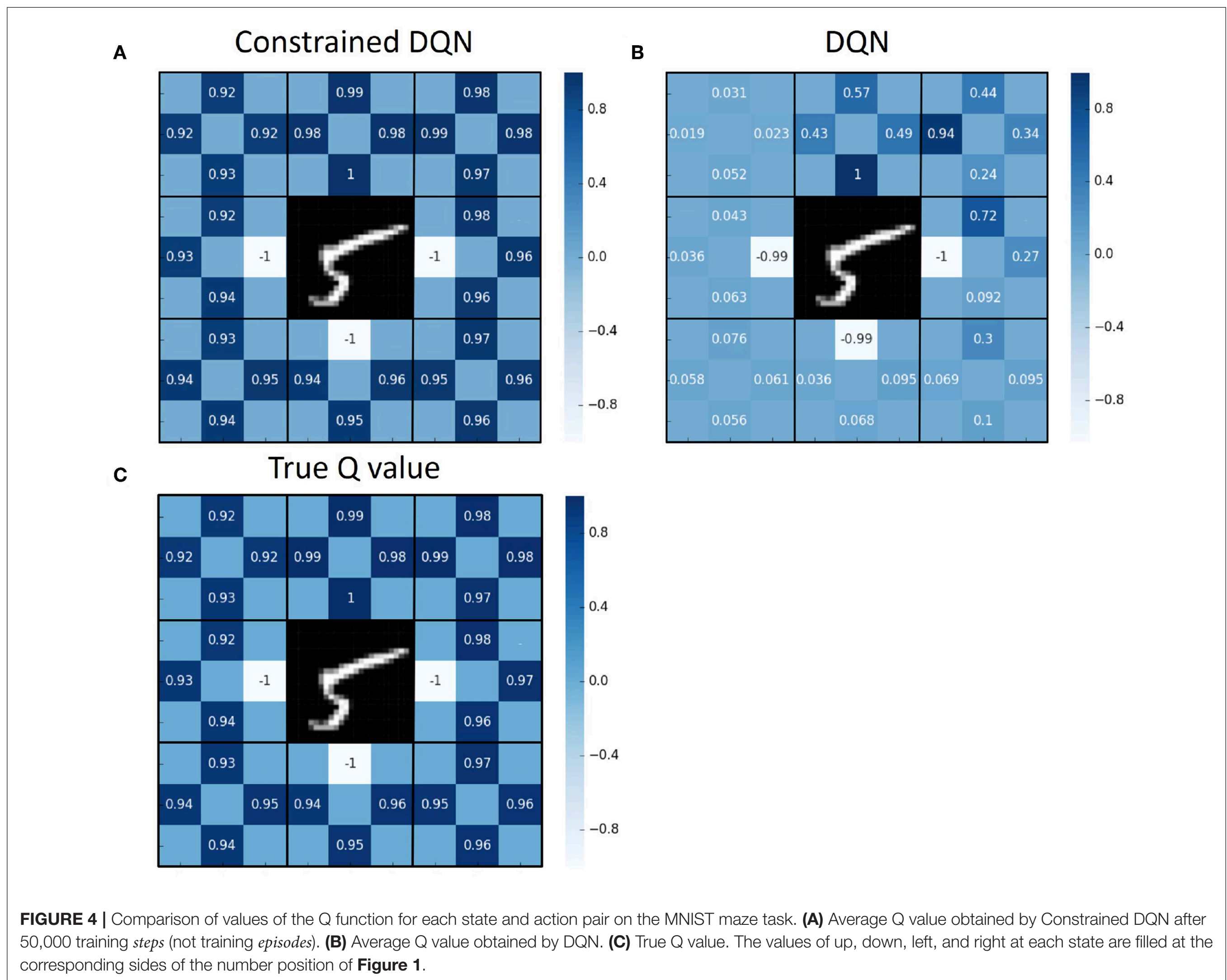

FIGURE 4 | Comparison of values of the Q function for each state and action pair on the MNIST maze task. **(A)** Average Q value obtained by Constrained DQN after 50,000 training *steps* (not training *episodes*). **(B)** Average Q value obtained by DQN. **(C)** True Q value. The values of up, down, left, and right at each state are filled at the corresponding sides of the number position of **Figure 1**.

frequency of the target network in Constrained DQN. When the update frequency of the target network was large, the square distance between the Q function and the target network was likely large. In this case, the constraint was frequently activated so that the convergence was delayed, especially when the threshold value η was small. When the update frequency was small, on the other hand, the square distance hardly increased, especially when η was large. In such a case, the update method was close to that of conventional Q learning even in the early stages of learning, which made learning unstable. However, when $\lambda = 1$, the results were reasonably good regardless of the hyperparameter setting.

5.2. Mountain-Car

5.2.1. Task Setting

Mountain-Car is a classical control task, and it has often been used for evaluating reinforcement learning algorithms. The agent (i.e., the car) aims to reach the fixed goal $x = 0.5$ (the top of the hill) from the fixed start position $x = -0.5$ (at almost the bottom of the valley). The agent can observe its current position and velocity. Position is limited to the range $[-1.2, 0.6]$, velocity is limited to the range $[-0.07, 0.07]$, and time is discretized. Available actions at each discretized time include "left acceleration" ($a = 0$), "no acceleration" ($a = 1$), and "right acceleration" ($a = 2$). After determining the action (a), velocity is calculated as follows:

$$v = v + 0.001(a - 1) - 0.0025 \cos 3x,$$

where v is velocity and x is position. Even if the agent continues to choose "right acceleration" from the start position, the agent cannot reach the goal because of insufficient motion energy. To reach the goal, the agent must choose "left acceleration" first and accumulate enough potential energy on the left-side slope, which is then transformed into motion energy to climb up the right-side hill. The agent is given a -1 reward at every discretized time step. If the agent reaches the goal, the episode ends; that is, the problem is defined as a stochastic shortest path problem.

We applied Q learning, DQN, DQN with TC-loss, and Constrained DQN to this task and made a comparison. We used a

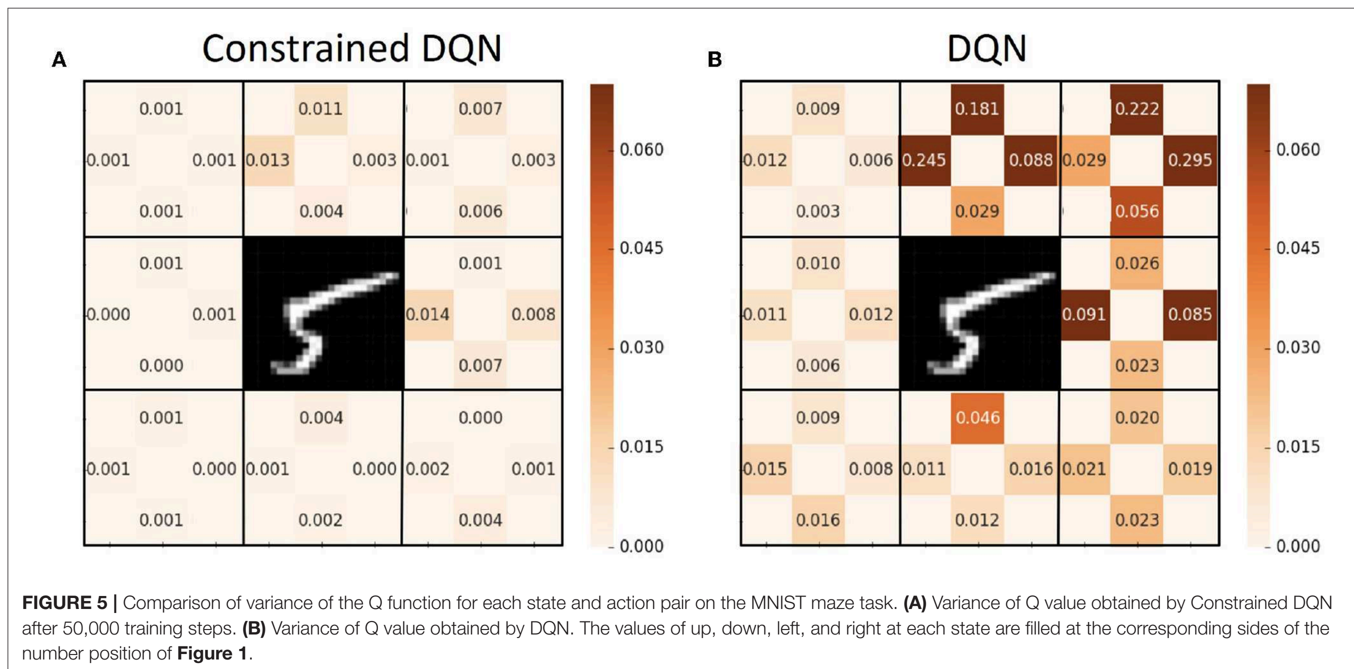

FIGURE 5 | Comparison of variance of the Q function for each state and action pair on the MNIST maze task. **(A)** Variance of Q value obtained by Constrained DQN after 50,000 training steps. **(B)** Variance of Q value obtained by DQN. The values of up, down, left, and right at each state are filled at the corresponding sides of the number position of **Figure 1**.

network architecture consisting of the two fully connected layers shown in **Figure 2B**. The input dimensionality is two, current position and velocity, and the output dimensionality is three, corresponding to the number of possible actions. According to the ϵ greedy method, ϵ was reduced in the first $10,000$ steps and then fixed at a constant value. We set the replay memory size to $10,000$, target network update frequency to 100, λ to 2, and η to 10^{-3}. We set the other parameters, such as the RMSProp parameters, except ξ, to those used in DQN (Mnih et al., 2015). It took about 6 h to train each method on a standard multi-core CPU (16-core/32-thread, 2.4 GHz, and 256 GB RAM).

5.2.2. Results

Figure 9 shows the learning curves of Q learning, DQN, DQN with TC-loss, and Constrained DQN. We conducted four independent runs for each algorithm. The lines excepting DQN indicate the moving average across four independent runs. One run of DQN was excluded because it failed to learn. We found that Constrained DQN performed better than DQN with TC-loss and Q learning, and learned faster than DQN although its performance was lower than that of DQN. DQN with TC-loss converged faster and achieved more stable learning than Q learning. Although DQN achieved the highest total rewards, its learning process was unstable. **Figure 10** shows the learning curves of DQN and Constrained DQN, with two different settings of ξ, which is one of the RMSProp parameters. We observed that the number of samples required for convergence was smaller in Constrained DQN than in DQN for both settings of ξ. We could also observe that when $\xi = 0.01$, the original DQN was unstable and even degraded after attaining a temporary convergence. On the other hand, our Constrained DQN demonstrated relatively good stability. The learning curves of DQN with TC-loss and Q-learning are shown in **Figure S1**.

Figure 11 shows how the L1-norms of gradients of the last fully-connected layer changed during learning. When $\xi = 1$, in the original DQN and Constrained DQN, the number of parameter updates was small. When $\xi = 0.01$, on the other hand, the number of updates by the original DQN was much larger than that of Constrained DQN. We consider that this is the cause of instability in the original DQN. Because the target value was fixed for C learning steps in the original DQN, the variance of gradient decreased rapidly within the C steps, making it very small, especially at the end of the C steps. On the other hand, because the target value changed every step in Constrained DQN, the variance of gradient smoothly became small through the entire learning process. ξ worked only when the variance of gradient was very small. In such situations, the smaller the ξ, the larger was the number of parameter updates. Consequently, it is considered that in the original DQN, ξ worked too frequently and thus the resulting large number of updates made learning unstable. Actually, with a small ξ ($\xi = 0.01$), the DQN learning was unstable. On the other hand, Constrained DQN was more robust regardless of the setting of ξ because the variance of gradient decreased smoothly throughout the entire learning process.

5.3. Robot Navigation Task

5.3.1. Task Setting

Constrained DQN is evaluated on the robot navigation task shown in **Figure 12**, in which the environmental model is provided by the task's official site (Robotis e-Manual, 2019). The environment consists of six rooms with six green trash cans, three tables, and three bookshelves. We use a Turtlebot 3 waffle pi platform equipped with a 360° LIDAR. The robot has five possible actions at every step: (1) move forward, (2) turn left, (3) turn right, (4) rotate left, and (5) rotate right. The objective is to

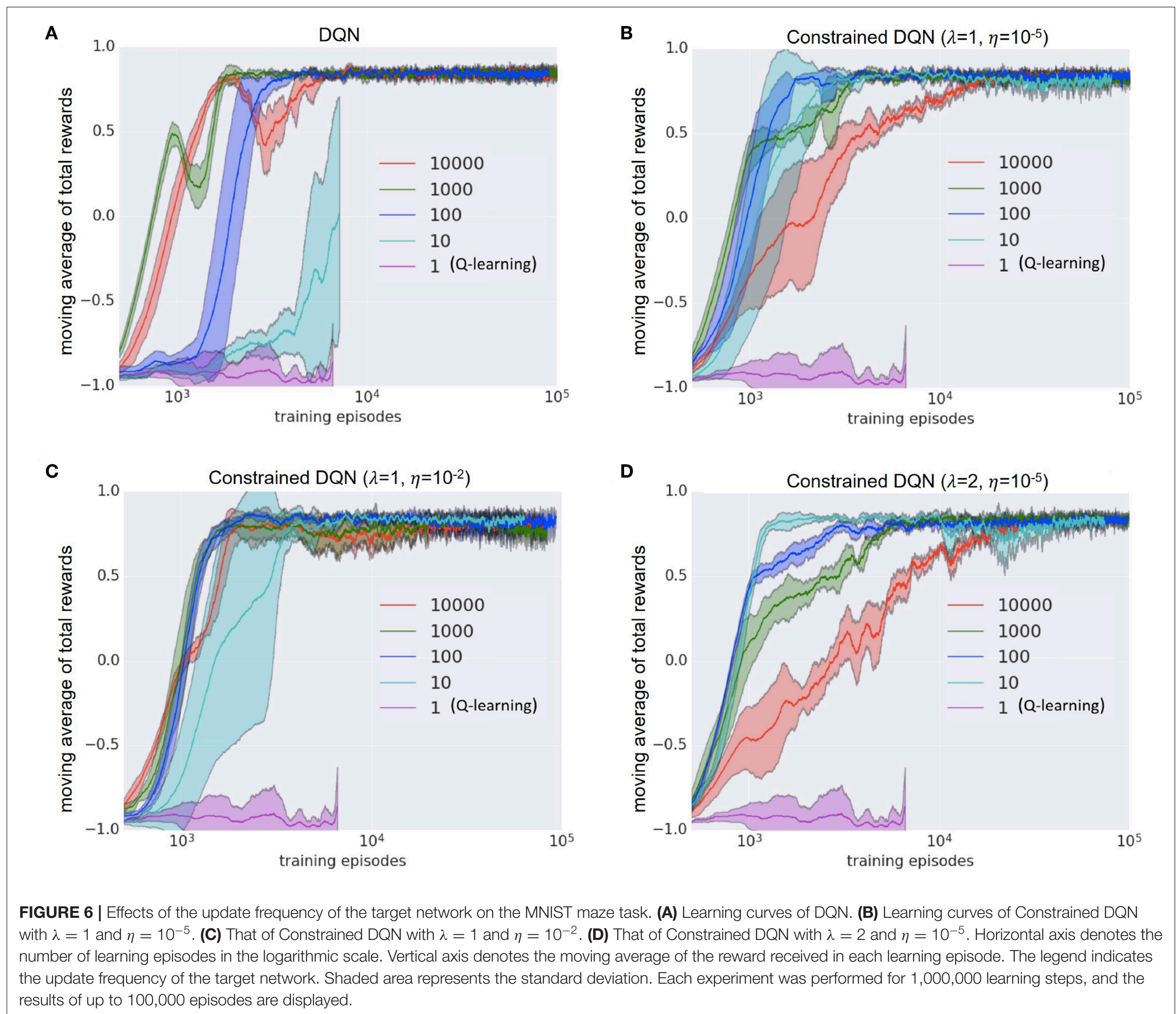

FIGURE 6 | Effects of the update frequency of the target network on the MNIST maze task. **(A)** Learning curves of DQN. **(B)** Learning curves of Constrained DQN with $\lambda = 1$ and $\eta = 10^{-5}$. **(C)** That of Constrained DQN with $\lambda = 1$ and $\eta = 10^{-2}$. **(D)** That of Constrained DQN with $\lambda = 2$ and $\eta = 10^{-5}$. Horizontal axis denotes the number of learning episodes in the logarithmic scale. Vertical axis denotes the moving average of the reward received in each learning episode. The legend indicates the update frequency of the target network. Shaded area represents the standard deviation. Each experiment was performed for 1,000,000 learning steps, and the results of up to 100,000 episodes are displayed.

navigate to one of the green trash cans placed in the environment. The robot receives a positive reward of $+10$ for reaching the trash can but a negative reward of -1 for hitting an obstacle. If the robot hits an obstacle, it remains in its current position. If the robot reaches the trash can, the position of the robot is re-initialized randomly in the environment.

We applied Q learning, DQN, DQN with TC-loss, Double DQN (DDQN), Soft Q learning (SQL), and Constrained DQN. The Q function is implemented by the two-stream neural network shown in **Figure 2C**, in which the input is the values measured by the RGB image and the LIDAR and the output is a five-dimensional vector representing action values. To collect experience efficiently, we use three robots that share the Q function, i.e., one ϵ greedy policy derived from the Q function controls three robots independently and they collect experiences that are sent to the replay buffer. In that sense, this is a simplified implementation of the large-scale distributed optimization that is known to accelerate the value learning (Levine et al., 2016). We conducted five independent runs for each method. Every run included 100 episodes, and each episode had at most 10,000 steps. The memory size of the replay buffer was 30,000 steps. The target network was updated every 3,000 steps. The training batch size was 32, uniformly sampled from the memory buffer. The hyperparameters were $\alpha = 0.01$, $\gamma = 0.99$, $\epsilon = 0.1$, $\beta = 10$. We used Ubuntu Linux 16.04 LTS as the operating system and version 375.66 of the NVIDIA proprietary drivers along with CUDA Toolkit 8.0 and cuDNN 6. Training takes about 1 day on a single NVIDIA Tesla P100 GPU for one setting.

5.3.2. Results

Since there are two sources of rewards and experience was collected by multiple robots asynchronously, we evaluated the

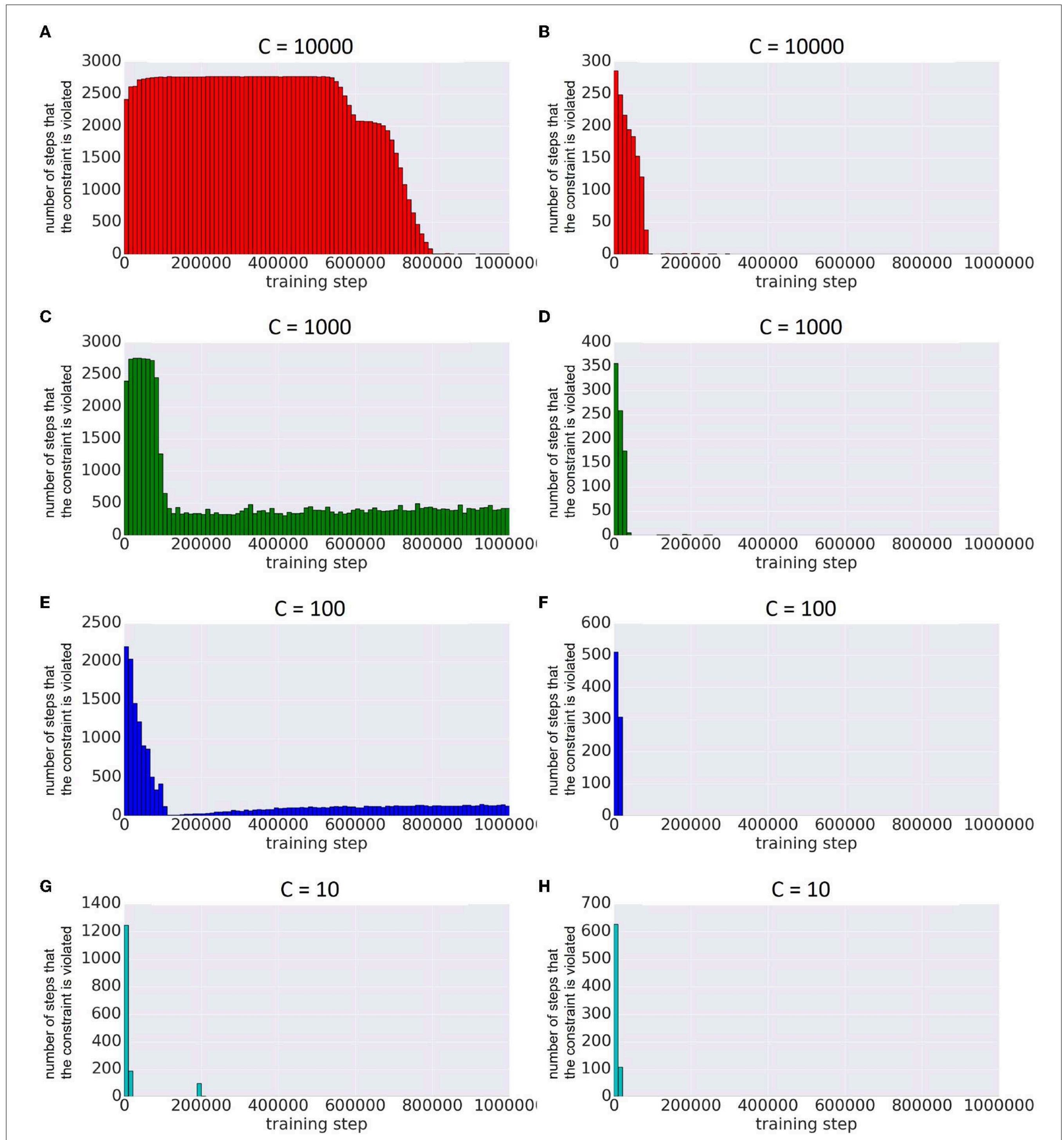

FIGURE 7 | Comparison of the number of steps by which the constraint is violated with different frequencies of updating the target network on the MNIST maze task. Horizontal axis represents the number of learning steps. Vertical axis represents the number of steps in which the constraints were activated within a fixed number of steps. The left column is the result when $\eta = 10^{-5}$, and the right column is when $\eta = 10^{-2}$ ($\lambda = 1$ in both columns). **(A)** The case of C = 10000 and $\eta = 10^{-5}$, **(B)** that of C = 10000 and $\eta = 10^{-2}$, **(C)** that of C = 1000 and $\eta = 10^{-5}$, **(D)** that of C = 1000 and $\eta = 10^{-2}$, **(E)** that of C = 100 and $\eta = 10^{-5}$, **(F)** that of C = 100 and $\eta = 10^{-2}$, **(G)** that of C = 10 and $\eta = 10^{-5}$, and **(H)** that of C = 10 and $\eta = 10^{-5}$.

learned Q function every five episodes by executing a greedy policy. **Figure 13** shows the number of steps used to go to one of the green trash cans and the number of collisions. This shows that Constrained DQN obtained nearly the fewest steps, the lowest collision rates, and the fastest convergence of learning among the methods compared here. Although there was not

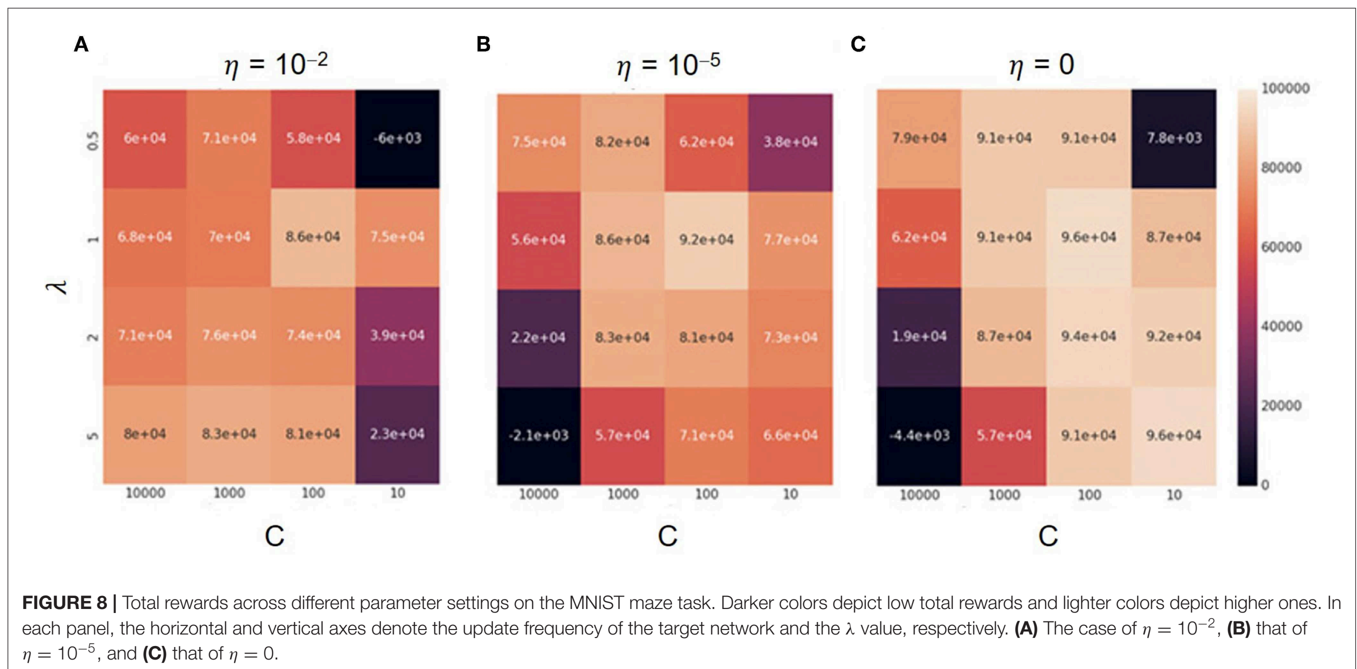

FIGURE 8 | Total rewards across different parameter settings on the MNIST maze task. Darker colors depict low total rewards and lighter colors depict higher ones. In each panel, the horizontal and vertical axes denote the update frequency of the target network and the λ value, respectively. **(A)** The case of $\eta = 10^{-2}$, **(B)** that of $\eta = 10^{-5}$, and **(C)** that of $\eta = 0$.

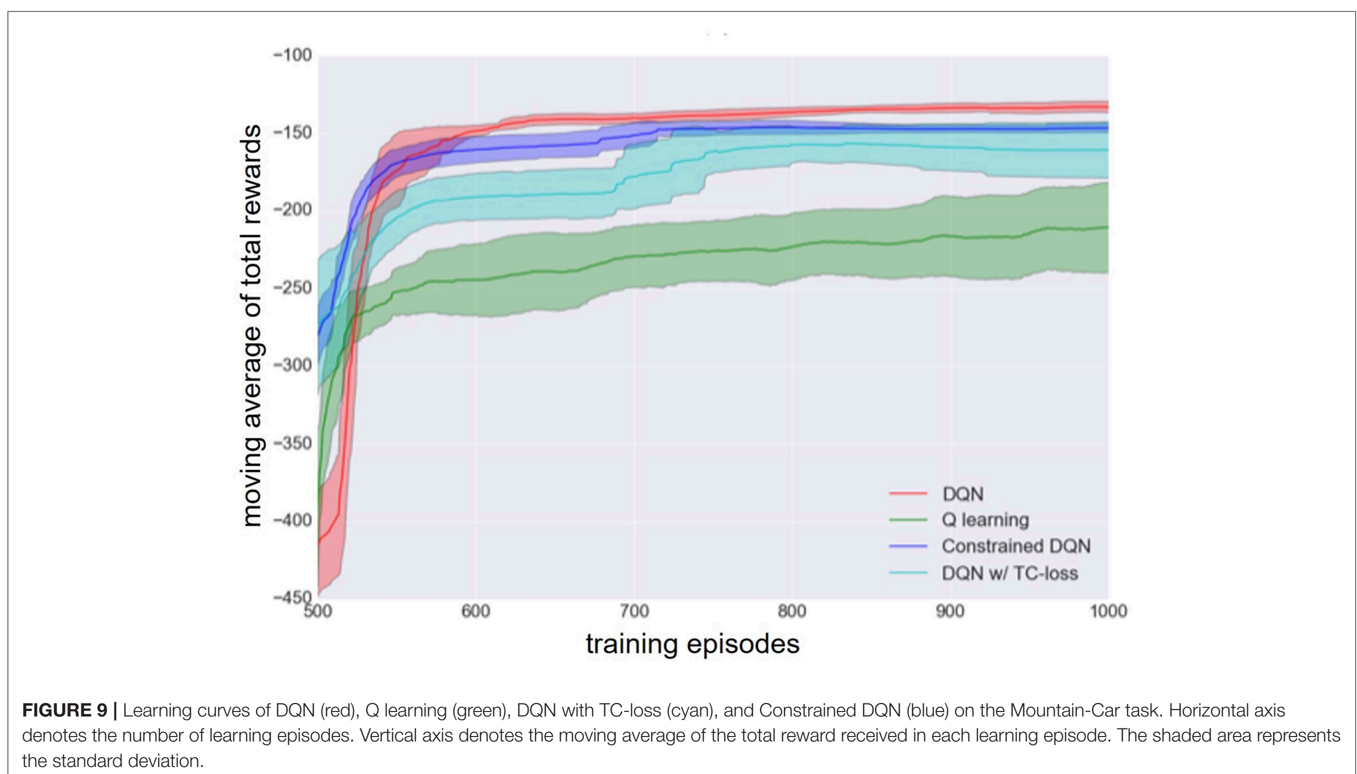

FIGURE 9 | Learning curves of DQN (red), Q learning (green), DQN with TC-loss (cyan), and Constrained DQN (blue) on the Mountain-Car task. Horizontal axis denotes the number of learning episodes. Vertical axis denotes the moving average of the total reward received in each learning episode. The shaded area represents the standard deviation.

much difference in the number of steps between DQN with and without TC-loss, the number of steps for DQN decreased faster than that for DQN with TC-loss. Q learning learned collision avoidance behaviors faster than DQN, but it completely failed to learn about approaching the trash cans. The performance of DDQN at the end of episode is comparable to that of Constrained DQN, but DDQN learned relatively slower than Constrained DQN.

5.4. Atari Games

Finally, we investigate whether Constrained DQN can be combined with useful and existing techniques to improve

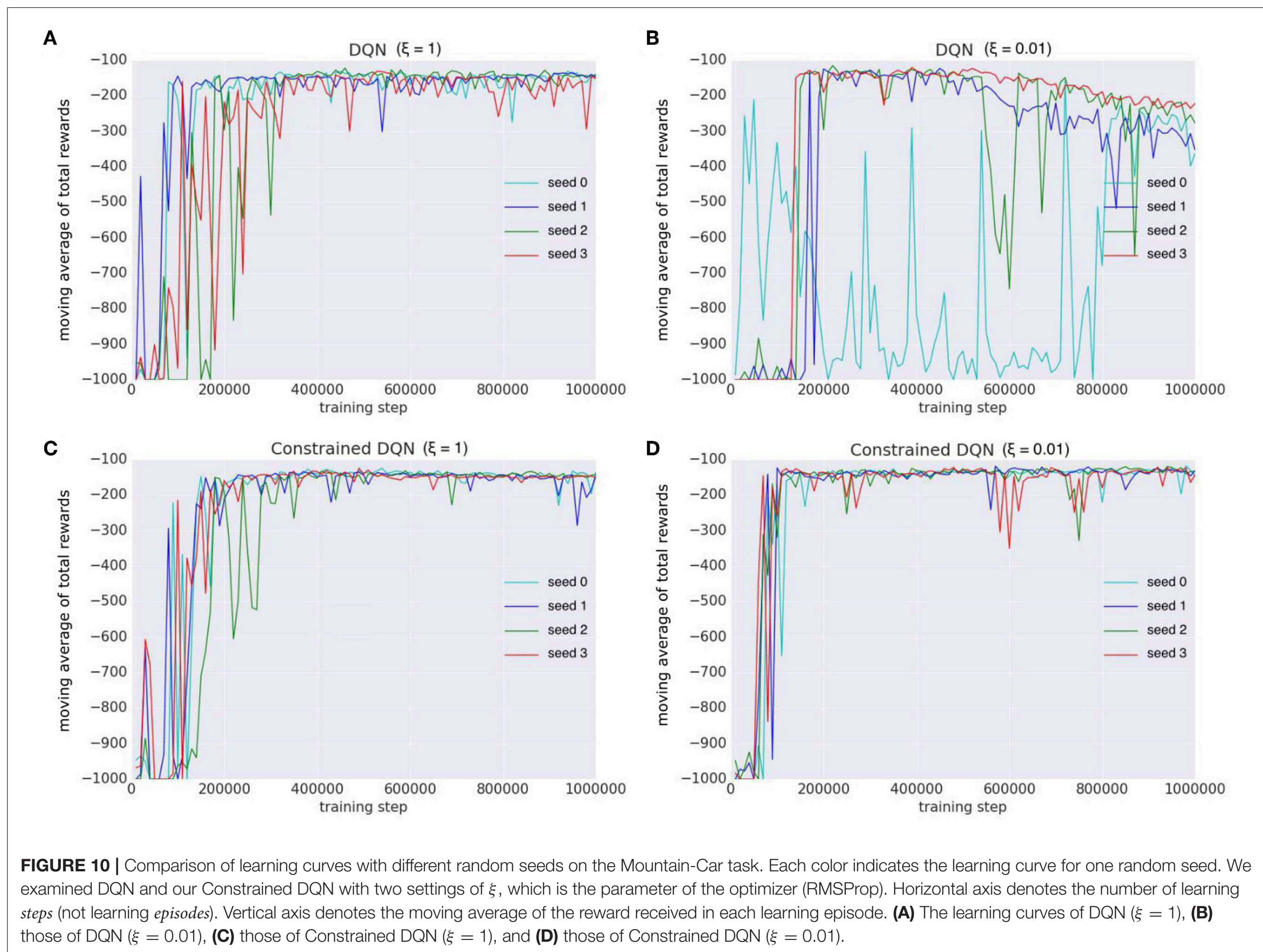

FIGURE 10 | Comparison of learning curves with different random seeds on the Mountain-Car task. Each color indicates the learning curve for one random seed. We examined DQN and our Constrained DQN with two settings of ξ, which is the parameter of the optimizer (RMSProp). Horizontal axis denotes the number of learning *steps* (not learning *episodes*). Vertical axis denotes the moving average of the reward received in each learning episode. **(A)** The learning curves of DQN ($\xi = 1$), **(B)** those of DQN ($\xi = 0.01$), **(C)** those of Constrained DQN ($\xi = 1$), and **(D)** those of Constrained DQN ($\xi = 0.01$).

sample efficiency. Here, we examined two techniques. One is the dueling network architecture, and the other is entropy-based regularization.

5.4.1. Task Setting

We evaluate Constrained DQN in two Atari 2600 games, namely Ms. Pac-Man and Seaquest, which were also evaluated by Pohlen et al. (2018). The goal of Ms. Pac-Man is to earn points by eating pellets while avoiding ghosts. Seaquest is an underwater shooter and the goal is to destroy sharks and enemy submarines to rescue divers in the sea. In these experiments, Constrained DQN and DQN with TC-loss were evaluated as the standard algorithm. For each algorithm, the dueling network architecture and the entropy regularization were selected to examine whether Constrained DQN can be used together with such useful techniques developed for improving learning processes. Consequently, we applied six algorithms on the Atari games. Note that TC-loss with the entropy regularization and that with the dueling network architecture are identical to Soft Q learning with TC-loss and Dueling DQN with TC-loss, respectively.

Although the original games are deterministic, randomness was added to introduce uncertainty to the starting state, by performing a random number of no-op actions on initialization. We used the same network architecture used by Mnih et al. (2015) for the standard algorithms and those with the entropy regularization, and the dueling network architecture used by Wang et al. (2016) for the algorithms with the dueling network architecture. The input at each time step was a preprocessed version of the current frame. Preprocessing consisted of gray-scaling, down-sampling by a factor of 2, cropping the game space to an 80×80 square and normalizing the values to $[0, 1]$. We stack four consecutive frames together as a state input to the network and clip the reward to the range of $[-1, 1]$. We adopted the optimizer and hyperparameters of Pohlen et al. (2018) and $\beta = 1$ for the entropy regularization.

5.4.2. Results

Figure 14 shows that Constrained DQN with the dueling network architecture achieved higher rewards faster than other methods on both games. The learning speed of Constrained DQN with entropy regularization was comparable to that of Constrained DQN with the dueling network architecture at the early stage of learning, but its learning process was less stable on Ms. PacMan. However, usage of the entropy regularization

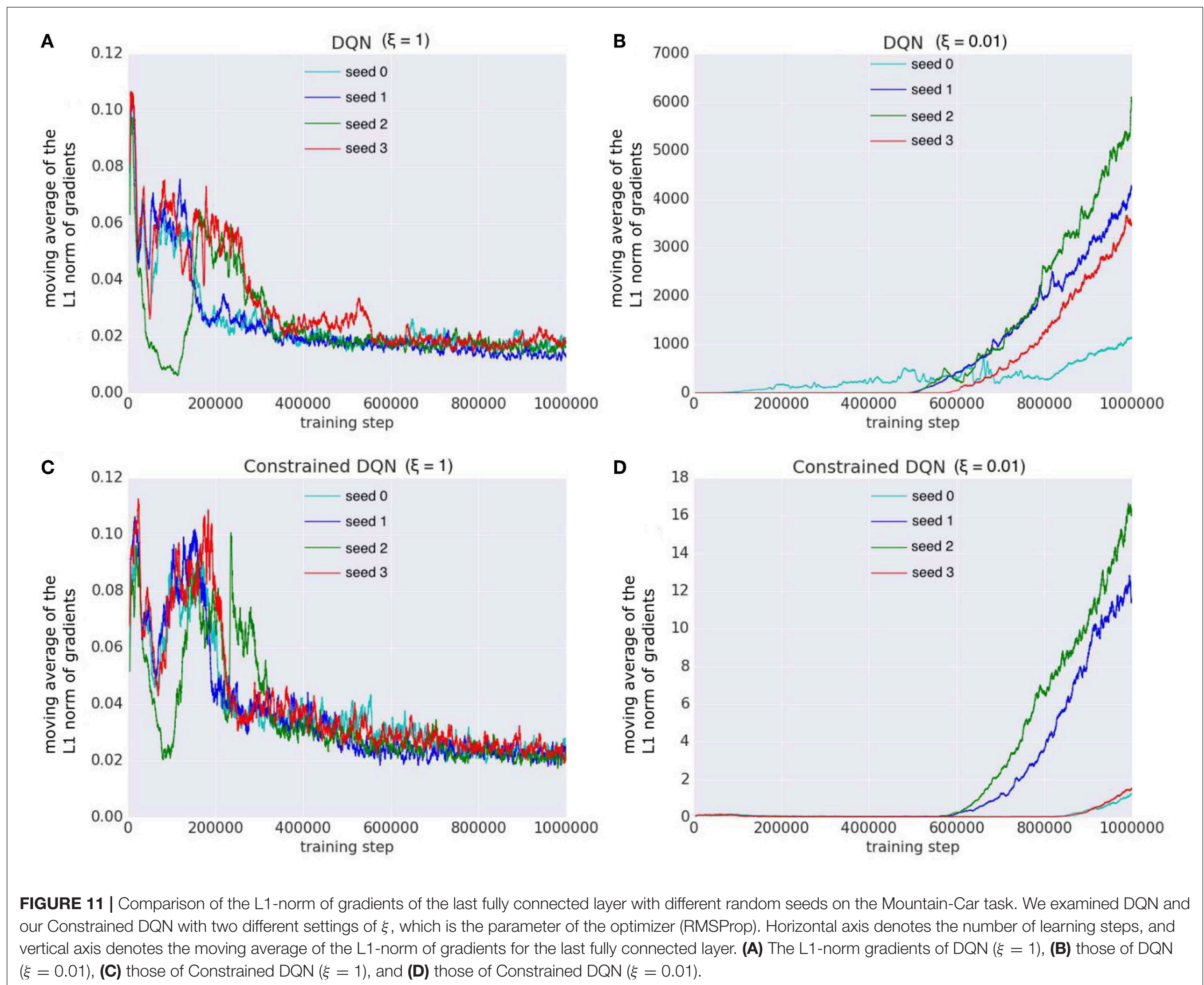

FIGURE 11 | Comparison of the L1-norm of gradients of the last fully connected layer with different random seeds on the Mountain-Car task. We examined DQN and our Constrained DQN with two different settings of ξ, which is the parameter of the optimizer (RMSProp). Horizontal axis denotes the number of learning steps, and vertical axis denotes the moving average of the L1-norm of gradients for the last fully connected layer. **(A)** The L1-norm gradients of DQN ($\xi = 1$), **(B)** those of DQN ($\xi = 0.01$), **(C)** those of Constrained DQN ($\xi = 1$), and **(D)** those of Constrained DQN ($\xi = 0.01$).

did not improve the performance on Seaquest, and it achieved almost the same performance as that of the normal Constrained DQN at the end of learning. This might have occurred, because the hyperparameter of the entropy regularization, β, was fixed during learning.

The experimental results also show that TC-loss and its variants performed worse than the corresponding Constrained DQNs. The performance of TC-loss with the dueling network architecture was comparable to that of Constrained DQN at the end of learning, but it took more epochs to converge.

6. DISCUSSION

We performed experiments involving four kinds of tasks, the MNIST maze, Mountain-Car, robot navigation, and two Atari games. Through these experiments, we demonstrated that Constrained DQN converges with fewer samples than does DQN with and without TC-loss and Q learning and, moreover, that Constrained DQN is more robust against changes in the target update frequency and the setting of important parameters of the optimizer, i.e., ξ.

Because the proposed method is a regularization method for DQN, it can more efficiently solve any problem to which DQN has been applied. The experimental results on the Atari games show that Constrained DQN can be used together with the dueling network architecture and the entropy regularization. We think that Constrained DQN can be combined with other techniques that employ a target network, such as improved experience replay techniques (Schaul et al., 2015; Andrychowicz et al., 2017; Karimpanal and Bouffanais, 2018), parametric function of the noise (Fortunato et al., 2018; Plappert et al., 2018), and modified Bellman operators (Bellemare et al., 2016; Pohlen et al., 2018). It suggests that Rainbow Constrained DQN and Ape-X Constrained DQN are respectively considered as an alternative of Rainbow DDQN (Hessel et al., 2017) and Ape-X DQN (Horgan et al., 2018), in which DDQN is replaced with Constrained DQN. Recently, van Hasselt et al. (2019) showed that data-efficient Rainbow outperformed model-based

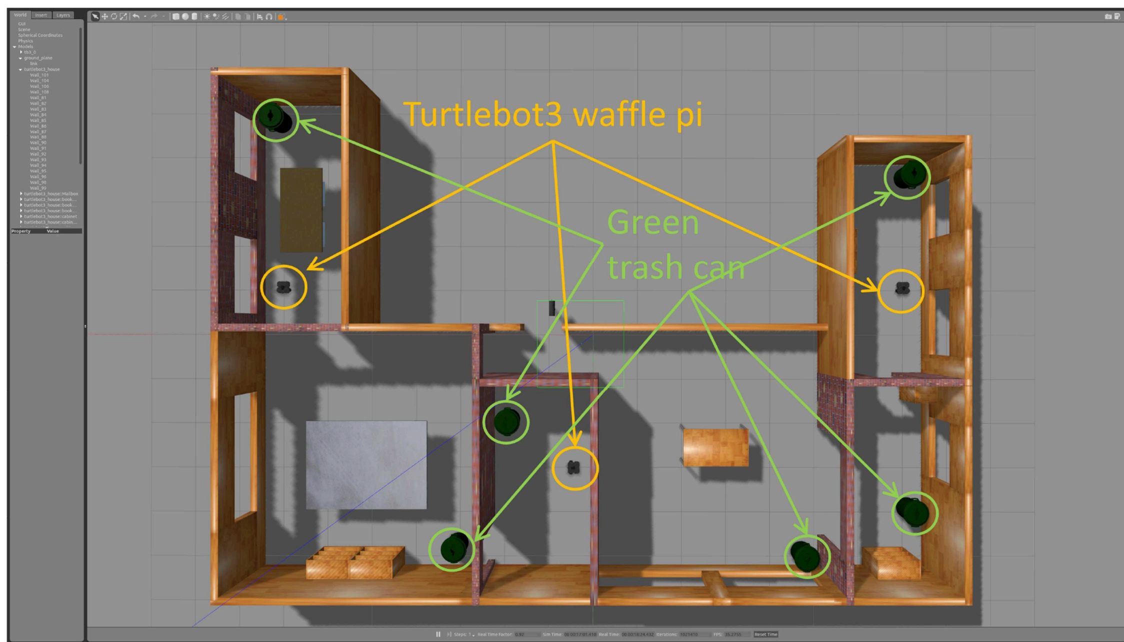

FIGURE 12 | Robot navigation task. Three mobile robots (Turtlebot3 waffle pi), six green trash cans, and various objects were placed in the environment. The objective of the robot is to move to one of the green trash cans without colliding against other objects, including obstacles.

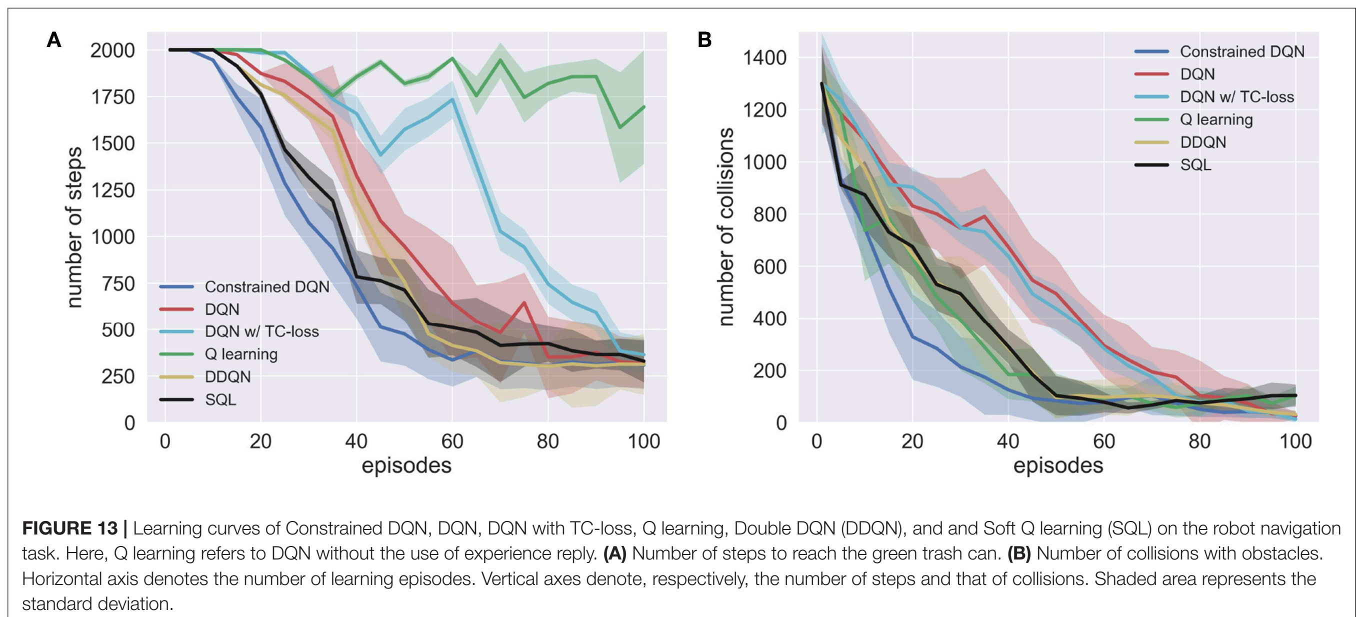

FIGURE 13 | Learning curves of Constrained DQN, DQN, DQN with TC-loss, Q learning, Double DQN (DDQN), and and Soft Q learning (SQL) on the robot navigation task. Here, Q learning refers to DQN without the use of experience reply. **(A)** Number of steps to reach the green trash can. **(B)** Number of collisions with obstacles. Horizontal axis denotes the number of learning episodes. Vertical axes denote, respectively, the number of steps and that of collisions. Shaded area represents the standard deviation.

reinforcement learning by extensively using the replay buffer to train and improve the reinforcement learning agent on the Atari games. Their study is impressive because no algorithmic changes were required in its implementation. It is promising to evaluate data-efficient Constrained DQN under the same setting of van Hasselt et al. (2019).

However, it is not trivial to integrate Constrained DQN with DDQN and its extension called Weighted Double Q learning (Zhang et al., 2017), because in these methods the target network was used to decompose the max operation into action selection and action evaluation. To reduce the problem of overestimation, the mellowmax operator (Kim et al., 2019) is promising, which is a variant of Soft Q learning.

Speedy Q learning (Azar et al., 2011) is based on a similar idea to that of our study. Speedy Q learning uses the difference between the output of the current Q function and the previous step's Q function as the constraint. Because the authors of that method only examined the task of discrete state space, one may

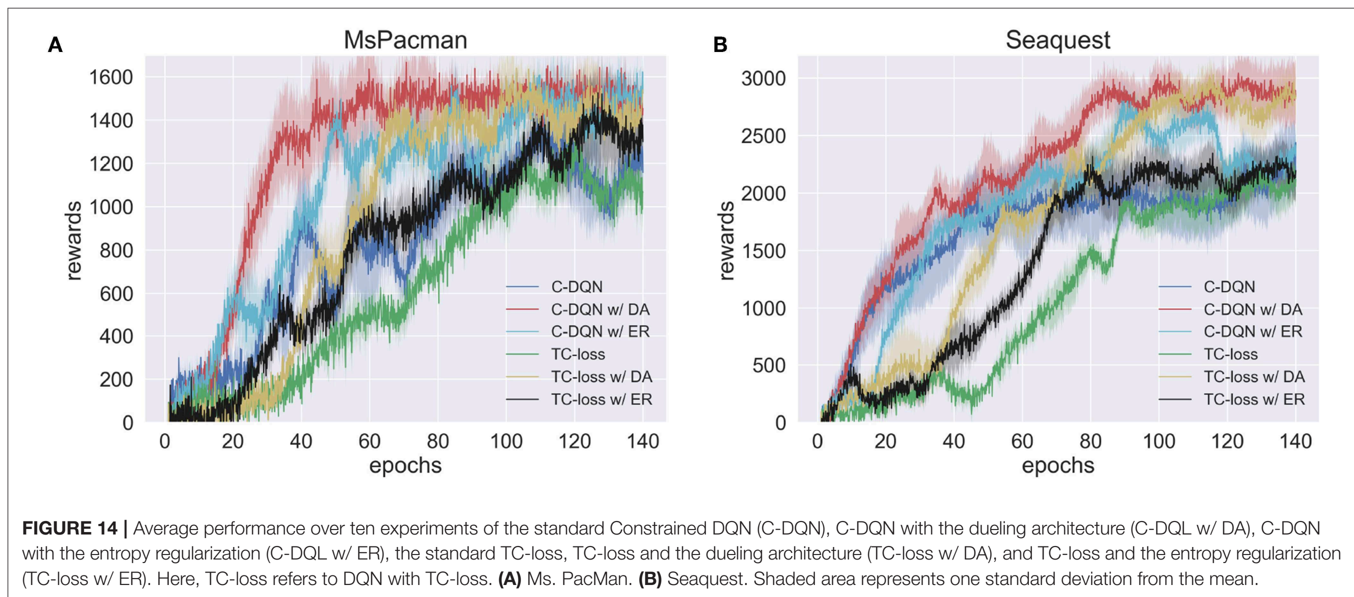

FIGURE 14 | Average performance over ten experiments of the standard Constrained DQN (C-DQN), C-DQN with the dueling architecture (C-DQL w/ DA), C-DQN with the entropy regularization (C-DQL w/ ER), the standard TC-loss, TC-loss and the dueling architecture (TC-loss w/ DA), and TC-loss and the entropy regularization (TC-loss w/ ER). Here, TC-loss refers to DQN with TC-loss. **(A)** Ms. PacMan. **(B)** Seaquest. Shaded area represents one standard deviation from the mean.

wonder whether it could be applied to learning with function approximation. In our study, deep/shallow neural networks were used for function approximation so that the results could be verified by tasks of both discrete state space and continuous state space. He et al. (2017) is another method of adding a constraint to Q learning. In this method, the accumulated reward is added to the data saved in the replay memory to allow the upper and lower limits of the Q function to be estimated at each learning step. Averaged DQN (Anschel et al., 2017) is similar to our method because both methods use past Q functions. Averaged DQN uses Q functions of the past few steps for calculating its output, i.e., action values, as the average of the outputs of the past Q functions. This averaging is effective in reducing the variance of the approximation error so that learning can be stabilized. One possible drawback of this method is the necessity of maintaining multiple Q functions, which are often represented as costly neural networks. On the other hand, our method requires only two networks, the current Q function and the target network, as in DQN, and so the number of parameters is not so large.

7. CONCLUDING REMARKS

In this study, we proposed Constrained DQN, which employs the difference between the outputs of the current Q function and the target network as a constraint. Based on several experiments that include the discrete state-space MNIST maze, the continuous state-space Mountain-Car, simulated robot navigation task, and two Atari games, we showed that Constrained DQN required fewer samples to converge than did the baselines. In addition, the proposed method was more robust against changes in the update frequency of the target network and the setting of important optimizer parameters (i.e., ξ of the RMSProp) than was DQN.

The several tasks used in this study have a discrete action space, but the proposed method can be combined with other methods that are applicable to problems with continuous action space, such as deep deterministic policy gradient (Lillicrap et al., 2016) and normalized advantage function (Gu et al., 2016). If the proposed method could also reduce the number of samples in continuous action space problems, it would be available for a wide range of applications, such as robot control and autonomous driving, since real-world applications involve complications in the collection of a sufficient number of samples for training deep reinforcement learners.

Possible future directions of this study include the following. Although we have shown that our proposed method was sample efficient experimentally, we have not yet established any theoretical reason for Constrained DQN to work properly. Recently, theoretical analyses are made for DQN (Yang et al., 2019) and conservative value iteration (Kozuno et al., 2019). For better understanding of Constrained DQN, we will establish the algorithmic and statistical rates of convergence. In addition, hyperparameter λ was fixed at a heuristic value in this study, but λ could also be optimized under the formulation of the constrained optimization; we can expect an improvement in performance by applying this extension.

AUTHOR CONTRIBUTIONS

SO and EU conceived the research. SO, EU, KN, YYas, and SI developed the algorithm. SO and YYam performed the computer simulations. SO, YYam, and EU analyzed the data. SO wrote the draft. SI and EU revised the manuscript. All authors prepared the submitted version.

FUNDING

This work was based on the results obtained from a project commissioned by the New Energy and Industrial Technology

Development Organization (NEDO) and was partially supported by JSPS KAKENHI Grant Numbers JP16K12504, JP17H06042, and 19H05001 (for EU), and JP17H06310 and JP19H04180 (for SI). Computer resources were provided by Honda R&D Co., Ltd.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fnbot.2019.00103/full#supplementary-material

REFERENCES

Achiam, J., Knight, E., and Abbeel, P. (2019). Towards characterizing divergence in deep Q-learning. *arXiv[Prepront].arXiv:1903.08894*.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., et al. (2017). "Hindsight experience replay," in *Advances in Neural Information Processing Systems, Vol. 30*, eds I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Long Beach, CA: Curran Associates, Inc.), 5048–5058.

Anschel, O., Baram, N., and Shimkin, N. (2017). "Averaged-DQN: variance reduction and stabilization for deep reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning* (Sydney, NSW), 176–185.

Azar, M. G., Munos, R., Ghavamzadeh, M., and Kappen, H. J. (2011). "Speedy Qlearning," in *Advances in Neural Information Processing Systems, Vol. 24*, eds J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger (Granada: Curran Associates, Inc.), 2411–2419.

Baird, L. (1995). "Residual algorithms: reinforcement learning with function approximation," in *Proceedings of the 12th International Conference on Machine Learning* (Montreal, QC), 30–37. doi: 10.1016/B978-1-55860-377-6.50013-X

Bellemare, M. G., Ostrovski, G., Guez, A., Thomas, P. S., and Munos, R. (2016). "Increasing the action gap: New operators for reinforcement learning," in *Proceedings of the 30th AAAI Conference on Artificial Intelligence* (Phoenix, AZ).

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., et al. (2016). OpenAI gym. *arXiv[Preprint].arXiv:1606.01540*.

Durugkar, I., and Stone, P. (2017). "TD learning with constrained gradients," in *Proceedings of the Deep Reinforcement Learning Symposium, NIPS 2017* (Long Beach, CA).

Elfwing, S., Uchibe, E., and Doya, K. (2016). From free energy to expected energy: improving energy-based value function approximation in reinforcement learning. *Neural Netw.* 84, 17–27. doi: 10.1016/j.neunet.2016.07.013

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., et al. (2018). "Noisy networks for exploration," in *Proceedings of the 6th Inernational Conference on Learning Representation* (Vancouver, BC).

Fujimoto, S., van Hoof, H., and Meger, D. (2018). "Addressing function approximation error in actor-critic methods," in *Proceedings of the 35th International Conference on Machine Learning* (Stockholm).

Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv[Preprint].arXiv:1308.0850*.

Gu, S., Lillicrap, T., Sutskever, I., and Levine, S. (2016). "Continuous deep Q-learning with model-based acceleration," in *Proceedings of the 33rd International Conference on Machine Learning* (New York, NY), 2829–2838.

Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. (2017). "Reinforcement learning with deep energy-based policies," in *Proceedings of the 34th International Conference on Machine Learning* (Sydney, NSW), 1352–1361.

He, F. S., Liu, Y., Schwing, A. G., and Peng, J. (2017). "Learning to play in a day: faster deep reinforcement learning by optimality tightening," in *Proceedings of the 5th Inernational Conference on Learning Representation* (Toulon).

Hernandez-Garcia, J. F. (2019). *Unifying n-step temporal-difference action-value methods* (Master's thesis), Master of Science in Statistical Machine Learning, University of Alberta, Edmonton, AB, Canada.

Hernandez-Garcia, J. F., and Sutton, R. S. (2019). Understanding multi-step deep reinforcement learning: a systematic study of the DQN target. *arXiv[Preprint].arXiv:1901.07510*.

Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., et al. (2017). "Rainbow: combining improvements in deep reinforcement learning," in *Proceedings of the 32nd AAAI Conference on Artificial Intelligence* (New Orleans, LA).

Horgan, D., Quan, J., Budden, D., Barth-Maron, G., Hessel, M., van Hasselt, H., et al. (2018). "Distributed prioritized experience replay," in *Proceedings of the 6th International Conference on Learning Representations* (Vancouver, BC).

Kahn, G., Zhang, T., Levine, S., and Abbeel, P. (2017). "PLATO: policy learning using adaptive trajectory optimization," in *Proceedings of IEEE International Conference on Robotics and Automation* (Singapore), 3342–3349.

Kapturowski, S., Ostrovski, G., Quan, J., Munos, R., and Dabney, W. (2019). "Recurrent experience replay in distributed reinforcement learning," in *Proceedings of the 7th International Conference on Learning Representations* (New Orleans, LA).

Karimpanal, T. G., and Bouffanais, R. (2018). Experience replay using transition sequences. *Front. Neurorobot.* 21:32. doi: 10.3389/fnbot.2018.00032

Kim, S., Asadi, K., Littman, M., and Konidaris, G. (2019). "Deepmellow: removing the need for a target network in deep Q-learning," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence* (Macao).

Kozuno, T., Uchibe, E., and Doya, K. (2019). "Theoretical analysis of efficiency and robustness of softmax and gap-increasing operators in reinforcement learning," in *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics* (Okinawa), 2995–3003.

Levine, S., Pastor, P., Krizhevsky, A., and Quillen, D. (2016). "Learning hand-eye coordination for robotic grasping with large-scale data collection," in *Proceedings of International Symposium on Experimental Robotics* (Tokyo: Springer), 173–184.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., et al. (2016). "Continuous control with deep reinforcement learning," in *Proceedings of the 4th International Conference on Learning Representations* (San Diego, CA).

Lin, L.-J. (1993). *Reinforcement Learning for Robots Using Neural Networks*. Technical report, School of Computer Science, Carnegie-Mellon University, Pittsburgh PA.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., et al. (2016). "Asynchronous methods for deep reinforcement learning," in *Proceedings of the 33rd International Conference on Machine Learning* (New York, NY), 1928–1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., et al. (2013). Playing Atari with deep reinforcement learning. *arXiv[Preprint].arXiv:1312.5602*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., et al. (2015). Human-level control through deep reinforcement learning. *Nature* 518, 529–533. doi: 10.1038/nature14236

Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., et al. (2018). "Parameter space noise for exploration," in *Proceedings of the 6th Inernational Conference on Learning Representation* (Vancouver, BC).

Pohlen, T., Piot, B., Hester, T., Azar, M. G., Horgan, D., Budden, D., et al. (2018). Observe and look further: achieving consistent performance on Atari. *arXiv[Preprint].arXiv:1805.11593*.

Riedmiller, M. (2005). "Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method," in *Proceedings of the 16th European Conference on Machine Learning* (Porto: Springer), 317–328.

Robotis e-Manual (2019). Available online at: http://emanual.robotis.com/docs/en/platform/turtlebot3/overview/ (accessed June 21, 2019).

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). "Prioritized experience replay," in *Proceedings of the 4th International Conference on Learning Representations* (San Diego, CA).

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 484–489. doi: 10.1038/nature 16961

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., et al. (2017). Mastering the game of Go without human knowledge. *Nature* 550, 354–359. doi: 10.1038/nature24270

Sutton, R. S., and Barto, A. G. (1998). *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning series)*. Cambridge, MA; London: The MIT Press.

Tsitsiklis, J. N., and Van Roy, B. (1997). Analysis of temporal-diffference learning with function approximation. *IEEE Trans. Autom. Control* 42, 674–690.

Tsurumine, Y., Cui, Y., Uchibe, E., and Matsubara, T. (2019). Deep reinforcement learning with smooth policy update: application to robotic cloth manipulation. *Robot. Auton. Syst.* 112, 72–83. doi: 10.1016/j.robot.2018. 11.004

van Hasselt, H. (2010). "Double Q-learning," in *Advances in Neural Information Processing Systems, Vol. 23*, eds J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta (Vancouver, BC: Curran Associates, Inc.), 2613–2621.

van Hasselt, H., Guez, A., and Silver, D. (2016). "Deep reinforcement learning with double Q-learning," in *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, Vol. 16 (Phoenix, AZ), 2094–2100.

van Hasselt, H., Hessel, M., and Aslanides, J. (2019). "When to use parametric models in reinforcement learning?" in *Advances in Neural Information Processing Systems*, Vol. 32, eds H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Vancouver, BC: Curran Associates, Inc.).

Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., and De Freitas, N. (2016). "Dueling network architectures for deep reinforcement learning," in *Proceedings of the 33rd International Conference on Machine Learning* (New York, NY).

Watkins, C. J. C. H., and Dayan, P. (1992). Q-learning. *Mach. Learn.* 8, 279–292.

Yang, Z., Xie, Y., and Wang, Z. (2019). A theoretical analysis of deep Q-learning. *arXiv[Preprint].arXiv:1901.00137*.

Zhang, Z., Pan, Z., and Kochenderfer, M. J. (2017). "Weighted double Q-learning," in *Proceedings of the 26th International Joint Conference on Artificial Intelligence* (Melbourne, VIC), 3455–3461.

Ziebart, B. D., Maas, A., Bagnell, J. A., and Dey, A. K. (2008). "Maximum entropy inverse reinforcement learning," in *Proceedings of the 23rd AAAI Conference on Artificial Intelligence* (Chicago, IL), 1433–1438.

Conflict of Interest: SO, EU, KN, and YYas are employed by company Panasonic Co. Ltd, ATR Computational Neuroscience Laboratories, Honda R&D Co. Ltd., and Honda R&D Co. Ltd., respectively. SI was partly employed by ATR Computational Neuroscience Laboratories. These companies are mutually independent commercially or financially.

The remaining author declares that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Copyright © 2019 Ohnishi, Uchibe, Yamaguchi, Nakanishi, Yasui and Ishii. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.

APPENDIX

Optimizer

In this study, RMSPropGraves (Graves, 2013) is used as an optimizer. When the gradient f_i with respect to θ_i is given, θ_i is updated as follows:

$$
\begin{aligned}
n_i &= \rho n_i + (1-\rho) f_i^2, \\
g_i &= \rho g_i + (1-\rho) f_i, \\
\Delta\theta_i &= \beta \Delta\theta_i - \frac{\alpha}{\sqrt{n_i - {g_i}^2 + \xi}} f_i, \\
\theta_i &= \theta_i + \Delta\theta_i,
\end{aligned}
\tag{A1}
$$

where n_i and g_i are the first- and second-order moments of the gradient and ρ and β give the exponential decay rate. α represents the learning rate, and $\Delta\theta_i$ is the update amount of θ_i. $n_i - {g_i}^2$ in (A1) approximately refers to the moving average of the variance of the gradient. When this term is small (large), the change in parameters is large (small). ξ is a small value that prevents Δ_i from becoming too large. In all of our experiments, we set $\alpha = 0.00025$, $\rho = 0.95$, $\beta = 0.95$, and $\xi = 0.01$, unless otherwise stated.