symmetry

Article

Temporal Consistency-Based Loss Function for Both Deep Q-Networks and Deep Deterministic Policy Gradients for Continuous Actions

Chayoung Kim

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

Abstract: Artificial intelligence (AI) techniques in power grid control and energy management in building automation require both deep Q-networks (DQNs) and deep deterministic policy gradients (DDPGs) in deep reinforcement learning (DRL) as off-policy algorithms. Most studies on improving the stability of DRL have addressed these with replay buffers and a target network using a delayed temporal difference (TD) backup, which is known for minimizing a loss function at every iteration. The loss functions were developed for DQN and DDPG, and it is well-known that there have been few studies on improving the techniques of the loss functions used in both DQN and DDPG. Therefore, we modified the loss function based on a temporal consistency (TC) loss and adapted the proposed TC loss function for the target network update in both DQN and DDPG. The proposed TC loss function showed effective results, particularly in a critic network in DDPG. In this work, we demonstrate that, in OpenAI Gym, both "cart-pole" and "pendulum", the proposed TC loss function shows enormously improved convergence speed and performance, particularly in the critic network in DDPG.

Keywords: deep Q-network; deep deterministic policy gradient; temporal consistency loss; temporal difference; deep reinforcement learning; target network update

check for updates

Citation: Kim, C. Temporal Consistency-Based Loss Function for Both Deep Q-Networks and Deep Deterministic Policy Gradients for Continuous Actions. *Symmetry* **2021**, *13*, 2411. https://doi.org/10.3390/sym13122411

Academic Editors: Alexander Zaslavski and Jian-Qiang Wang

Received: 10 October 2021
Accepted: 8 December 2021
Published: 13 December 2021

1. Introduction

Promising outputs have been accomplished in the field of deep reinforcement learning (DRL) that combines reinforcement learning (RL) [1] and deep learning (DL) [2]. With RL, we developed a framework for a behavioral policy that maximizes values regarding the control of unknown complex environments. With DL, we demonstrated that there is a high-level method of pattern recognition and image processing. In DRL, we applied DL using a deep neural network (DNN) as an approximation function for RL. DRL has achieved optimization applications, such as the games of Go and Alpha Go [3], which is one of the most incredible works. There are two well-developed representatives of model-free and off-policies in DRL: deep Q-network (DQN) [4,5] for discrete environments and deep deterministic policy gradient (DDPG) [6] for continuous action spaces.

Deep Q-networks [4,5] developed by Google Deep Mind learned to defeat 49 various Atari games through screen images. Q-learning [7] obtained an optimal action policy using an action-value function. In Atari games [4,5], DQN uses DL, such as a convolutional neural network (CNN), to extract feature patterns and RL, such as Q-learning, to train an agent. DDPG [6] combines the ideas from DQN [4,5], which uses experience replay buffers and slow-learning target networks, and a deterministic policy gradient (DPG) [8], which can operate over continuous action spaces. DDPG has two networks: an actor that proposes an action, given a state, and a critic that predicts whether the action is positive or negative, given a state and an action. DDPG also uses DNN approximations as a nonlinear function, such as DQN, for continuous real-valued action spaces. However, learning an agent is unstable and difficult using nonlinear function approximations [9]. To deal with these instabilities, DQN uses replay buffers with an off-policy method and a target Q-network with a delayed temporal difference (TD) backup. TD [1] is designed to evaluate a given policy and is interpreted as minimizing a loss function at every iteration of the value function.

It is known that DQN is used for discrete and low-dimensional action spaces and DDPG is used for continuous real-valued and high-dimensional action spaces. Both DQN and DDPG are known to have features that are symmetrical to each other. However, techniques in power grid control and operation [10] and energy management in building automation and control systems [11] require both the DQN and DDPG of DRL because their systems require an extensive exploration of high complexity, nonlinearity, and stochastic nature. In [10], DDPG is used to control the voltage set-points of generators, whereas DQN is used for shunt capacitors or to control transformer tap-ratios. The two different DRLs, namely DQN and DDPG, are applicable for training artificial intelligence (AI) techniques of autonomous voltage control in power grid control and operation. One study [11] uses two representative RL, DQN and DDPG, to exploit the availability of huge monitoring data and machine learning algorithms for improved strategies of load balancing in a simulated cooling network.

Pseudo-Code 1 DRL-based AI agent using both DQN and DDPG

```
1.   FMU is simulated by passed monitoring data and the state-vector is generated
2.   DRL agent processes the state-vector and the reward
       2–1. DQN for solving control problems with the set-point value for the temperature, which
            is designed to be within a discrete range, such as [0.95,0.975,1.0,1.025,1.05]
       2–2. DDPG for continuous action-spaces of float numbers in the range from −1 to +1, such
            as the parameters of the chiller and the valves of the cooling waters to the consumer
            sites
3.   The controls are forwarded to FMU for load balancing betweem sites
```

Pseudo-Code 1 describes the procedures for Figure 1, which demonstrates the schematic architecture of the framework used in [10,11]. In every iteration, the functional mock-up units (FMU) are simulated per time-step, and a state-vector is generated. Both the state-vector and the reward are processed by both DQN and DDPG, and the control signals for load balancing between sites are forwarded to the FMU. The DQN is for low-dimensional discrete data and DDPG is for continuous action-spaces of float numbers.

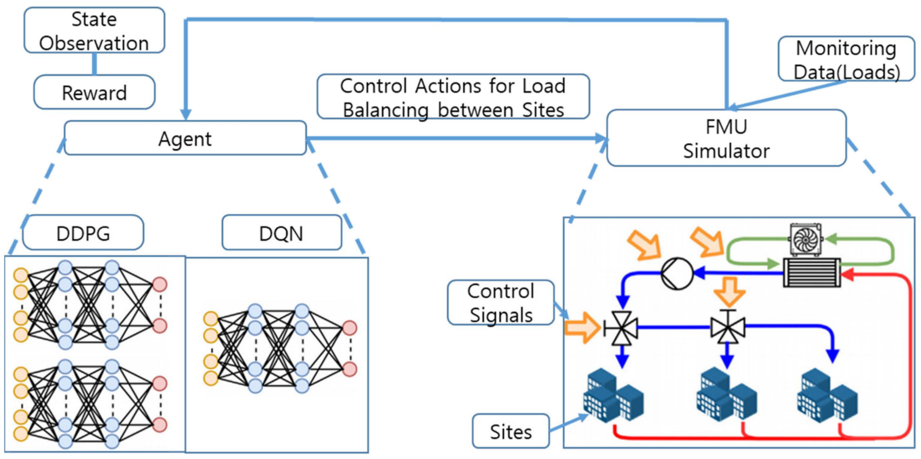


Figure 1. Schematic architecture of the framework used by both DQN and DDPG.

In this study, we considered the application of both DQN and DDPG similar to that of "step 2" in Pseudo-Code 1 and the agent in Figure 1 [11]. The most fundamental aspects to consider in both DQN and DDPG are the methods, replay buffers, and a target network [12]. In replay buffers, the correlations between sampled data are reduced because mini-batch learning is performed with random data. The target network is a neural network similar to a Q-network, even if it is updated slower than the Q-network. For a more stable learning process, less frequent updates of the target network are recommended. However, in principle, the use of the target network can cause the agent to be trained slowly and disrupt online RL, which is a desirable attribute [6,13]. This implies that the use of replay buffers and the target network are deviations from online RL. Moreover, the requirement of extremely large samples tends to be risky in actual applications. This might cause instability in long-term runs.

To stabilize the learning processes with the target network, Durugkar et al. [14] proposed constrained TD to prevent the target value from changing after the TD updates using the gradient projection technique. Pohlen et al. [15] proposed a temporal consistency (TC) loss to prevent the Q-function at every target state action from changing substantially by minimizing the target network. Therefore, the study by Pohlen et al. [15] was considered to alleviate the instability of the learning process. Ohnishi et al. [16] proposed constrained DQN to behave in two different methods: when the difference between the maximum value of the Q-function and the value of the target network is large, constrained DQN updates the Q-function more conservatively, and when this difference is small, constrained DQN behaves similar to that of conventional standard Q-learning. Studies [14–16] provide a family of target-based TD-learning algorithms [17]. Study [17] showed that the success of deep Q-learning is indispensable to use a separate target network to improve the performance of Q-learning, and provided insight into the theoretical approaches, and introduced three different update methods: averaging TD, double TD, and periodic TD, where the target network is updated in an averaging, symmetric, or periodic manner, respectively. The aforementioned studies are concerned only with DQN. Therefore, we focused on both DQN and DDPG from the insights of these studies.

We suggested a slightly modified TC loss function at each iteration, which originated from [15,16], for a periodic update of the target network. The constrained DQN [16] proposed a TC loss similar to that in [15], except using the source Q-function instead of the target Q-function. We modified the TC loss originating from both constrained DQN [16] and that in [15] for both DQN and DDPG, particularly for a critic network. In our study, the target network was updated based on the proposed TC loss. We mentioned the proposed TC loss as TC-DQN for DQN and TC-DDPG for the critic network of DDPG. Moreover, the characteristic features of TC-DQN and TC-DDPG were inherited from constrained DQN [16]; when the difference between the outputs of the Q-function and the target network is large, the update of the target network can be conservative, but when the difference between the outputs is small, the update can be aggressive, as in the case of standard Q-learning for both DQN and DDPG.

We implement the proposed TC loss functions, TC-DQN and TC-DDPG, for target network updates in standard tasks in OpenAI Gym, such as "cart-pole" for a discrete state space and "pendulum" for a continuous state space. Consequently, the proposed TC loss functions, TC-DQN and TC-DDPG, are more robust against fluctuations in the frequency of updates in the target network. The experimental results show that the proposed TC-DQN and TC-DDPG could be used as an additional component, as in [16]. Moreover, there is a big difference in the loss functions between Constrained DQN [16] and the proposed TC-DQN and TC-DDPG. Constrained DQN [16] uses the maximum value of Q-network for the additional subtracted gradient term of the loss function for DQN, and the difference in this paper is that target-network is used for both DQN and DDPG. The main contribution is that the proposed equation can be used simultaneously for both DQN and DDPG. Load balancing is a field that has been studied for a very long time, and it was possible to check the cases in which DQN and DDPG has been recently applied to the field through referenced papers [10,11]. We believe that the proposed TC-DQN and TC-DDPG could be useful in applications such as autonomous voltage control in power grid control and load shifting in a cooling supply system [10,11].

2. Notation and Background

2.1. Markov Decision Process (MDP)

An MDP [1] is characterized by (S, A, P, r, γ), where S denotes a finite state space, A denotes a finite action space, P(s, a, s') = P[s' | s, a] represents the state transition probability from state s to s' for action a, r: S $\times$ A $\rightarrow$ [0, σ] represents a uniform stochastic reward, and $\gamma \in (0, 1)$ denotes a discount factor. Further, $r^{\pi}(s)$ denotes a stochastic reward and $R^{\pi}(s)$ denotes the expectation for a policy π and a state s, that is, $R(s) = \sum_{a \in A}\pi(s, a)R(s, a)$. The infinite-horizon discounted value function for policy π is $J^{\pi}(s) = E[\sum_{i=0}\gamma^{i}r(s_i, a_i) \mid s_0 = s]$, where $s \in S$, and E denotes the expectation with regard to the state-action-reward trajectories. For pre-selected feature-functions $\varphi_1, \ldots, \varphi_n$: S $\rightarrow$ R, $\phi \in R^{|S| \times n}$ is defined as $\phi = \varphi(1) \ldots \varphi(|S|) \in R^{|S| \times n}$, where $\varphi(s) = \varphi_1(s) \ldots \varphi_n(s) \in R^n$. The goal of RL with the linear function approximation is to determine weight vector $\theta \in R^n$ such that $J = \phi\theta$ approximates the true value function J^{π}. In the standard TD-learning [1], the update rule is $\theta_{t+1} \leftarrow \theta_t - \alpha\eta(\theta_t)$, where $\eta(\theta_t) = -(r(s, a) + \gamma J_{\theta i}(s') - J_{\theta i}(s))\nabla_{\theta}J_{\theta i}(s)$. A key issue is that the stochastic gradient, $\eta(\theta_i)$, does not correspond to the true gradient of the loss function, $l(\theta)$. The asymptotic convergence of the TD-learning [1] is $\theta_{i+1} = \theta_i - \alpha_i\nabla_{\theta}l(\theta; \theta_i)$; the loss function $l(\theta;\theta_i) = \frac{1}{2}E_{s,a}[(E_{s',r}[r(s, a) + \gamma J_{\theta'}(s')] - J_{\theta}(s))^2]$, where θ denotes an online (source) variable and θ' denotes a target variable. At each iteration i, the target variable is set to the value of the current source variable, and a stochastic gradient step is performed as shown in Algorithm 1 [1].

Algorithm 1 **Standard TD-Learning**

```
Initialize $\theta o$ randomly and Set $\theta' o = \theta o$
For iteration k = 0, 1 ... do
        Sample $s \sim d(\cdot)$ and $\alpha \sim \pi(s,\cdot)$
        Sample $s'$ and $r(s, a)$
         Let $g_k = \phi(s)(r(s,a)+\gamma\phi(s')^T\theta'_k - \phi(s)^T \theta_k$
         Update $\theta_{k+1} = \theta_k - \alpha_k g_k$
         Update $\theta'_{k+1} = \theta_{k+1}$
    End for
```

2.2. Deep Q-Network (DQN)

In DQN [4,5] in Algorithm 2, DNNs and RL are successfully combined to approximate the action values for a given state s_t. At each time-step, based on current state s_t, the agent selects an action ε-greedily with regard to action value a_t, and stores a transition (s_t, a_t, r_t, s_{t+1}), characterized by the aforementioned MDP to a replay memory buffer ***D*** [12]. During the inner loop in Algorithm 2, DQN applies Q-learning updates with a mini-batch of experiences in ***D*** drawn randomly from the stored samples. After performing experiences replay, the agent executes an action in accordance with a ε-greedy policy. With a neural network as a function approximation, the actions of the agent represent the experience histories produced by a function φ such as (φ_t, a_t, r_t, φ_{t+1}). The parameters of the neural network with weight θ as a Q-network are optimized with stochastic gradient descent to minimize the loss in every iteration j, $(r_j + \gamma\max_{a'}Q^{\theta^-}(\varphi_{j+1}, a') - Q^{\theta}(\varphi_j, a_j))^2$. The gradient of the loss is back-propagated into weights θ of the online (source) network; the term θ^- denotes the weights of a target network; a periodic copy of the online network. The use of target networks and experience replay enables relatively stable Q-learning.

Algorithm 2 DQN with experience replay

```
Initialize replay memory $D$
Initialize $Q$ with the weight $\theta^{Q}$ for action-value function
Initialize $Q^{-}$ with the weight $\theta^{Q^{-}} = \theta^{Q}$ for target-net
For episode = 1, $M$ do
    Initialize sequence $S_1 = \{x_1\}$ and pre−processed sequence $\phi_1 = \phi(S_1)$
  For t = 1, $T$ do
      With probability $\epsilon$ select a random action $a_t$
          Otherwise select $a_t = argmax_a Q(\phi(S_t), a_t; \theta^{Q})$
       Execute action at and observe reward rt and new state $s_{t+1}$ and
          Pre−process $\phi_{t+1} = \phi s_{t+1}$
       Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
       Sample random mini−batch of transitions $(\phi_i, a_i, r_i, \phi_{i+1})$ from $D$
      Set $y_i = r_i$                if i + 1 = terminate
        $y_i = ri + \gamma max_{a'} Q\text{-}(\phi_{i+1}, a'; \theta^{Q-})$       otherwise
      Perform a gradient descent step on $(y_i - Q(s_i, a_i; \theta^{Q}))^2$
             With respect to the network parameters $\theta^{Q}$
          Every C steps reset $Q^{-} = Q$
  End for
End for
```

2.3. Deep Deterministic Policy Gradient (DDPG)

An efficient evaluation of the Q-value function is required to determine the optimal action in DQN. However, it is not solvable if the action space is continuous, although it is simple for discrete and small action spaces. In several applications, such as robotics, discretization is not desirable and might require large amounts of memory and computing power in the case of a fine discretization. Lillicrap et al. [6] presented an algorithm called DDPG, as shown in Algorithm 3, which is solvable for continuous applications with DRL; in contrast to the DQN, an actor-critic architecture is used. As policy μ in DDPG is a direct mapping from states to actions, such as μ: $S \to A$, it is currently the best policy, such as $\mu(s_t) = \max_{a'} Q(s_t, a')$. Actor μ and critic Q are estimated by function approximations $\mu(s \mid \theta^{\mu})$ and $Q(s \mid \theta^{Q})$, parameterized by θ^{μ} and θ^{Q}, respectively. With the insights of DQN, a target value for training is calculated using a slowly updated target Q-network and policy networks, denoted by $Q'(s \mid \theta^{Q'})$ and $\mu'(s \mid \theta^{\mu'})$, respectively. For each update time-step, a mini-batch of n samples is generated randomly. First, the target value y_i is computed using the target Q-network and policy networks, $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1} \mid \theta^{\mu'}) \mid \theta^{Q'})$. Then, the mean square error is obtained by loss $L(\theta^{Q}) = 1/n\sum i\ (y_i - Q(s_i, a_i \mid \theta^{Q}))^2$, and the policy is updated according to the mean of all samples, as stated in the DPG [8]: $\nabla_{\theta}{}^{\mu} R^{\mu} \leftarrow 1/n\sum i \nabla_a Q(s_i, a \mid \theta^{Q}) \mid_{a=\mu(si \mid \theta\mu)} \nabla_{\theta\mu} \mu(s_i \mid \theta^{\mu})$. The parameters $\theta^{Q'}$ and $\theta^{\mu'}$ of the target networks are slowly moved towards the parameters of their associates in each update step, $\theta^{Q'} \leftarrow (1 - \tau)\theta^{Q'} + \tau\theta^{Q}$ and $\theta^{\mu'} \leftarrow (1 - \tau)\theta^{\mu'} + \tau\theta^{\mu}$, with $\tau \in (0, 1]$.

Algorithm 3 DDPG

```
Initialize replay memory $D$
Initialize $Q$ with the weight $\theta^{Q}$ for critic-net
Initialize $Q-$ with the weight $\theta^{Q}- = \theta^{Q}$ for target-net of $Q$
Initialize $\mu$ with the weight $\theta^{\mu}$ for actor-net
Initialize $\mu-$ with the weight $\theta^{\mu-} = \theta^{\mu}$ for target-net of $\mu$
For episode = 1, $M$ do
Initialize a random process $N$ for action exploration
Initialize observation state $s1$
For t = 1, $T$ do
        Select action $at = \mu(st \mid \theta\mu) = Nt$ according to the current policy and exploration noise
Execute action at and observe reward rt and new state st + 1
Store transition $(st, at, rt, st + 1)$ in D
Set $yi = ri + \gamma Q-(si + 1, \mu-(si + 1 \mid \theta\mu- ) \mid \theta Q-)$
Update critic-net by minimizing the loss: $L = \frac{1}{N}\,\Sigma i\,(yi - Q(st, at) \mid \theta Q)2$
Update actor-net by using the sampled policy gradient:
            $\nabla\theta\mu\, J \approx \frac{1}{N}\,\Sigma i\,\nabla aQ(si, \mu(si) \mid \theta Q)\,\nabla\theta\mu\mu(si \mid \theta\mu)$
Update the target-nets:
$\theta\mu- = \tau\theta\mu + (1 - \tau)\theta\mu-$
$\theta Q- = \tau\theta Q + (1 - \tau)\theta Q-$
  End for
End for
```

3. Proposed TC Loss Functions for Both DQN and DDPG

3.1. Previously Developed Loss Functions

The update of Q-learning with a target network can be viewed as follows:

$$\theta_{t+1} \leftarrow \theta_t + \alpha(\mathrm{target}_Q - Q(s_t,a_t;\theta^{Q}{}_t))\nabla_\theta Q(s_t,a_t;\theta^{Q}{}_t), \tag{1}$$

where $\mathrm{target}_Q = r(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a; \theta^{Q-}{}_t)$, $\theta^{Q}{}_t$ denotes the source (online) variable, and $\theta^{Q-}{}_t$ denotes the target variable. The state-action value function $Q(s, a; \theta^{Q})$ is parameterized by θ. The update of the online variable $\theta^{Q}{}_t$ is similar to the stochastic gradient descent step. The term $r(s_t, a_t)$ represents the immediate reward of taking action a_t in state s_t, and target_Q denotes the target value under the target variable, $\theta^{Q-}{}_t$. When the target variable is set to be the same as the online variable at each iteration, learning the agent reduces to the standard Q-learning [7] and is known to be unstable with a nonlinear function approximation because of dynamic changes in the target, and the Q-function might diverge [1]. Several choices of target networks have been proposed in studies to overcome this instability: (i) periodic update, that is, it is copied from the online variable every $\tau > 0$ steps, as used for DQN [4,5]; (ii) symmetric update, that is, it is updated symmetrically as the online variable, first introduced in double Q-learning [18]; and (iii) Polyak averaging update, that is, it takes a weighted average over the past values of the online variable used in DDPG [6]. Studies [4–6,18] are categorized as target-based Q-learning [17]. A key issue is that the stochastic gradient does not correspond to the true gradient of the loss function to make the theoretical analysis rather subtle. When an agent selects actions stochastically according to a policy, in batch value prediction, a value function algorithm uses a fixed data batch to learn an estimate, which is never the same as the true value function [19]. Study [19] also addressed the issue of non-true probability of the agent action under the given policy with importance sampling to deal with the mismatch between the empirical weight and the correct weight.

3.2. Newly Proposed TC Loss Functions

We considered the developed TC loss function to minimize the instability of the learning process, particularly for the applications [10,11] in both DQN and DDPG. Therefore, we modified the loss function at each iteration, which originated from [15,16], for the target network update. We used a periodic update [17] based on the proposed TC loss. Moreover, we introduced the TC loss function for a critic network, particularly in DDPG. We used the critic and the actor of the target, $Q^-(s, a \mid \theta^{Q^-})$ and $\mu^-(s \mid \theta^{\mu-})$, respectively, similar to that in DDPG. In DDPG, the weights of these target networks are updated as follows: $\theta^{Q-} \leftarrow \tau\theta^Q + (1 - \tau)\theta^{Q-}$ with $\tau << 1$. However, in our study, we changed the update rule of the target network as follows.

First, $target_Q$ in (1) using DQN [4,5] was updated as follows:

$$\theta_{t+1} \leftarrow \theta_t + \alpha(target_{DQN} - Q(s_t,a_t;\theta^Q{}_t))\nabla_\theta Q(s_t,a_t;\theta^Q{}_t), \tag{2}$$

where $target_{DQN} = r(s_t, a_t) + \gamma Q^{Q-}(s_{t+1}, \text{target-action}; \theta^{Q-}{}_t)$ and target-action $= \max_{a'} Q^-(s_{t+1},a')$.

Second, $target_Q$ in (1) using DDPG [6] was updated as follows:

$$\theta_{t+1} \leftarrow \theta_t + \alpha(target_{DDPG\_Critic} - Q(s_t,\text{source-action};\theta^Q{}_t))\nabla_\theta Q(s_t,\text{source-action};\theta^Q{}_t), \tag{3}$$

where $target_{DDPG\_Critic} = r(s_t,\text{source-action}) + \gamma Q^-(s_{t+1},\text{target-action};\theta^{Q-}{}_t)$, source-action $= \mu(s_t \mid \theta^\mu{}_t) + \mathcal{N}_t$, target-action $= \mu^-(s_{t+1} \mid \theta^{\mu-}{}_{t+1}) + \mathcal{N}_{t+1}$, $\mu(s \mid \theta^\mu) = \max_a Q(s, a \mid \theta^\mu)$, and $\mathcal{N}$ = a random process from Ornstein-Uhlenbeck process [20] such as DDPG.

Pohlen et al. [15] added the TC loss function to alleviate the instability of the learning process between temporally adjacent target values. Although Huber loss was adopted in the original study [15], L2 loss was used in this study, as in [16], for more simplicity. We did not use a positive threshold of the constraint used in [16] because the hyper-parameters of the learning algorithm should be tuned individually for each task and the research. We did not expect an improvement in performance by applying the hyper-parameters because the hyper-parameters also had to be studied. Furthermore, it was considered that when the different components of the observation had different physical units, the ranges might vary across environments. This made it difficult for the network to learn effectively and determine hyper-parameters that generalize across environments with different scales of states [6].

For the differentiation of the modified TC loss function in our study, we defined $\mathcal{L}_{TC\text{-}DQN}(s_{t+1},\text{target-action};\theta^{Q-}{}_t)$ for DQN and $\mathcal{L}_{TC\text{-}DDPG\_Critic}(s_{t+1},\text{target-action}; \theta^{Q-}{}_t)$ for DDPG.

For DQN, (2) was updated with $\mathcal{L}_{TC\text{-}DQN}(s_{t+1}, \text{target-action};\theta^{Q-}{}_t)$ as follows:

$$\theta_{t+1} \leftarrow \theta_t + \alpha[(target_{DQN} - Q(s_t,a_t;\theta^Q{}_t))\nabla_\theta Q(s_t,a_t;\theta^Q{}_t) - \nabla_\theta \mathcal{L}_{TC\text{-}DQN}(s_{t+1},\text{target-action};\theta^{Q-}{}_t)], \tag{4}$$

where $\mathcal{L}_{TC\text{-}DQN}(s_{t+1}, \text{target-action};\theta^{Q-}{}_t) = \frac{1}{2}\sum_i(Q^-(i)(s_{t+1}, \text{target-action}) - Q^-(i-1)(s_{t+1}, \text{target-action}))^2$ and target-action $= \max_{a'} Q^-(s_{t+1},a')$.

For DDPG, (3) was updated with $\mathcal{L}_{TC\text{-}DDPG\_Critic}(s_{t+1},\text{target-action}; \theta^{Q-}{}_t)$ as follows:

$$\theta_{t+1} \leftarrow \theta_t + \alpha\,[(target_{DDPG\_Critic} - Q(s_t,\text{source-action};\theta^Q{}_t))\nabla_\theta Q(s_t,\text{source-action};\theta^Q{}_t) - \nabla_\theta L_{TC\text{-}DDPG\_Critic}(s_{t+1},\text{target-action}; \theta^{Q-}{}_t)], \tag{5}$$

where $\mathcal{L}_{TC\text{-}DDPG\_Critic}\,(s_{t+1},\text{target-action}; \theta^{Q-}{}_t) = \frac{1}{2}\sum_i(Q^-(i)(s_{t+1}, \text{target-action}) - Q^-(i-1)(s_{t+1}, \text{target-action}))^2$, source-action $= \mu(s_t \mid \theta^\mu{}_t) + \mathcal{N}_t$, target-action $= \mu^-(s_{t+1} \mid \theta^{\mu-}{}_{t+1}) + \mathcal{N}_{t+1}$, $\mu(s \mid \theta^\mu) = \max_a Q(s, a \mid \theta^\mu)$, and $\mathcal{N}$ = a random process.

The target-update was performed when $\mathcal{L}_{TC\text{-}DQN}(s_{t+1}, \text{target-action};\theta^{Q-}{}_t) \le \eta_{CDQN}$ and $\mathcal{L}_{TC\text{-}DDPG\_Critic}\,(s_{t+1},\text{target-action}; \theta^{Q-}{}_t) \le \eta_{TC\text{-}DDPG\_Critic}$, where η denotes a positive threshold of the constraint specified in [16]. The characteristics of the target network update could be applied flexibly in DQN and DDPG, as followed in [16]: when the difference between the outputs of the Q-function and the target network is large, the update of the target network can be conservative. Moreover, based on the suggestion in [16], the newly proposed TC loss functions could be used together with other methods to improve its performance.

Algorithm 4 includes both DQN and DDPG with the proposed TC loss functions, TC-DQN and TC-DDPG.

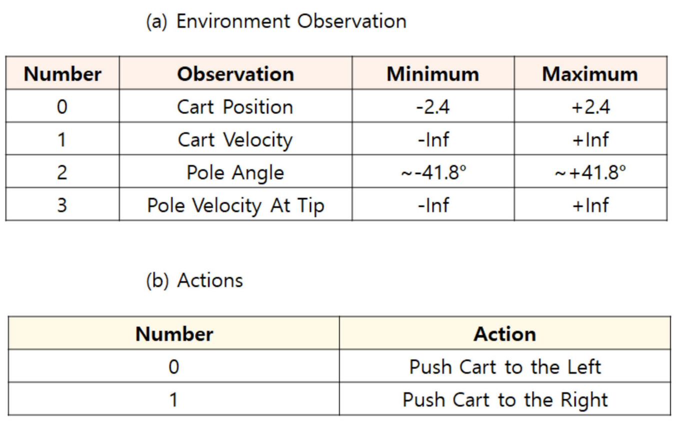
Algorithm 4 **The proposed algorithm with TC-DQN and TC-DDPG**

```
Initialize replay memory $D$
Initialize Q with the weight $\theta^{Q}$ for both action-value function and critic-net in both DQN and DDPG
Initialize $Q^{-}$ with the weight $\theta^{Q-} = \theta^{Q}$ for target-nets in both DQN and DDPG
Initialize $\mu$ with the weight $\theta^{\mu}$ for actor-net in DDPG
Initialize $\mu^{-}$ with the weight $\theta^{\mu-} = \theta^{\mu}$ for target-net of $\mu$ in DDPG
For episode = 1 , $M$ do
    Initialize a random process $N$ for DDPG
    Initialize observation state $s_1$
    For t = 1, $T$ do
        Derive the action in DQN $a_t$ with probability $\epsilon$ or $a_t = argmax_a Q(s_t, a; \theta^{Q})$ or in DDPG $a_t = \mu(s_t \mid \theta^{\mu}) + N_t$
            Execute action at and observe reward $r_t$ and new state $s_{t+1}$
            Store transition $(s_t, a_t, r_t, s_{t+1})$ in $D$
            Sample a random mini-batch of transitions $(s_i, a_i, r_i, s_{i+1})$ from $D$
            In DQN
                Set $y_i = r_i$            if i + 1 = terminate
                    $y_i = r_i + \gamma max_a{}' \, Q^{-}(s_{i+1}, a'; \theta^{Q-})$      otherwise
                    Update action-value function on $(y_i - Q(s_i, a_i; \theta^{Q}))^2$
                    Update target-net with the additional subtraction(TC-DQN) on
                      $-\frac{1}{2}\Sigma i)(Q^{-}(i)(s_{i+1}, max_a{}'Q^{-}(s_{i+1}, a'))- Q^{-}(i-1)(s_{i+1}, max_a{}'Q^{-}(s_{i+1}, a')))^2$
        In DDPG
          Set $y_i = r_i + \gamma Q^{-}(s_{i+1}, \mu^{-}(s_{i+1} \mid \theta^{\mu-}) \mid \theta Q^{-})$
          Update critic-net on $\frac{1}{N}\Sigma i \ ( y_i - Q(s_i, a_i) \mid \theta^{Q}))^2$
          Update actor-net on $\frac{1}{N}\Sigma i \ \nabla_a Q(s_i, \mu(s_i) \mid \theta^{Q}) \nabla_{\theta}\mu \ \mu(s_i \mid \theta^{\mu})$
          Update target-net for actor-net on $\theta\mu- = \tau\theta^{\mu} + (1 - \tau)\theta^{\mu}-$
          Update target-net for ciritic-net with the additional subtraction(TC-DDPG) on
              $-\frac{1}{2}\Sigma i \ (Q^{-}(i)(s_{i+1}, \mu^{-}(s_{i+1} \mid \theta^{\mu-}) + N_{i+1} - Q^{-}(i-1)(s_{i+1}, \mu^{-}(s_{i+1} \mid \theta^{\mu-}) + N_{i+1}))^2$
    End For
End For
```

4. Evaluation and Results

4.1. "Cart-Pole"

There are four observations and two discrete actions in "cart-pole" [21], as shown in Figure 2. The pole is attached to a cart that moves back and forth from left to right. The poles start straight. The goal is to not fall over when the cart is speeding up or slowing down. A reward of +1 is considered by the environment for every step when the pole remains vertical until the next action is completed. At the end of the episode, the angle of the pole is between -12° and $+12^{\circ}$, and the cart position is between -2.4 and +2.4. However, the requirements of implementation studies can be considered for better solutions [22]. A Q-learning agent will receive -100 if it falls before reaching the maximum length of the episode. Moreover, if the average reward over 10 consecutive episodes is 490 or more, Q-learning will end before the maximum length of the episode is reached. DQN with the proposed loss function was implemented using TensorFlow [23] and Keras [24] in OpenAI Gym [25].



(a) Environment Observation

Number	Observation	Minimum	Maximum
0	Cart Position	-2.4	+2.4
1	Cart Velocity	-Inf	+Inf
2	Pole Angle	~-41.8°	~+41.8°
3	Pole Velocity At Tip	-Inf	+Inf

(b) Actions

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

Figure 2. "cart-pole" [21] in OpenAI Gym [25].

In terms of quality comparison, as shown in Figures 3–5, in the best case, average case, and standard deviation, both the standard DQN and DQN with the proposed loss functions were considerably similar. As shown in Figure 5, the average standard deviation of the DQN with the proposed loss functions was 170, and the average standard deviation of the standard DQN was 166. This showed that the fluctuating patterns before reaching the end of the episode in both the cases were considerably similar. However, as depicted in Figure 6, in terms of quantity comparison, "When it is finished with the smallest steps," the DQN with the proposed loss functions was slightly faster than the standard DQN. The comparison was also conducted by Wilcoxon–Signed–Rank–Paired Test in R. "the number of steps in each episode" in Figure 6 had the *p*-value of Wilcoxon–Signed–Rank–Paired Test, 0.001953, which is more significant than the level, 0.05. This showed that the DQN with the proposed loss functions significantly improved compared to the standard DQN in almost the same period.

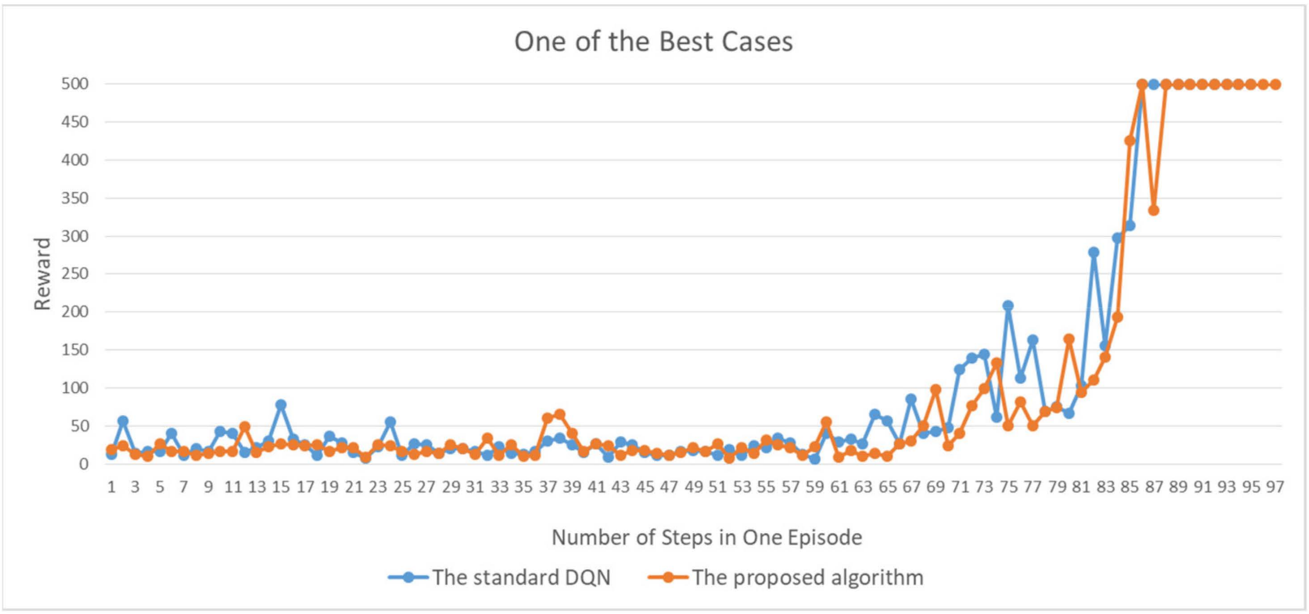


Figure 3. One best case comparison between the standard DQN and the DQN with the proposed TC loss function.

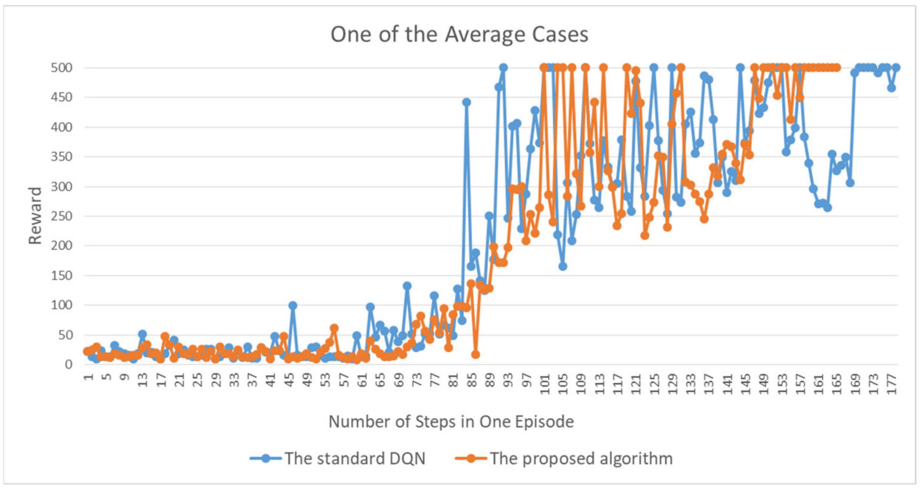


Figure 4. One average case comparison between the standard DQN and the DQN with the proposed TC loss function.

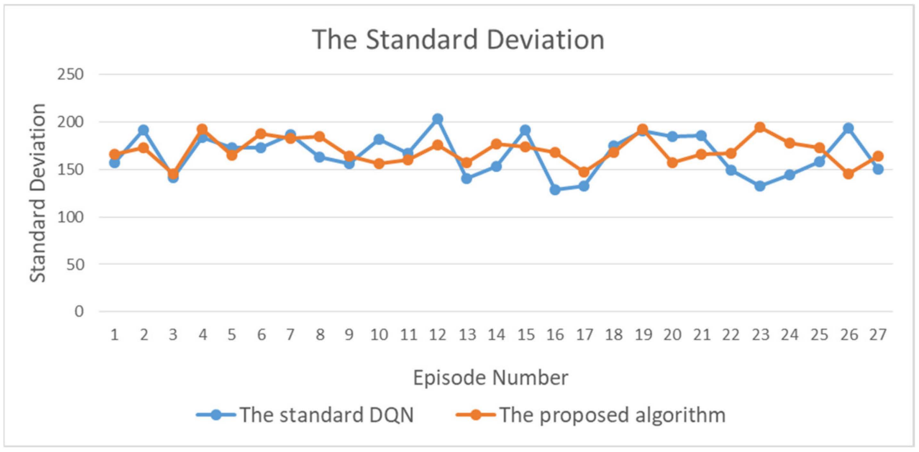


Figure 5. Standard deviation comparison between the standard DQN and the DQN with the proposed TC loss function.

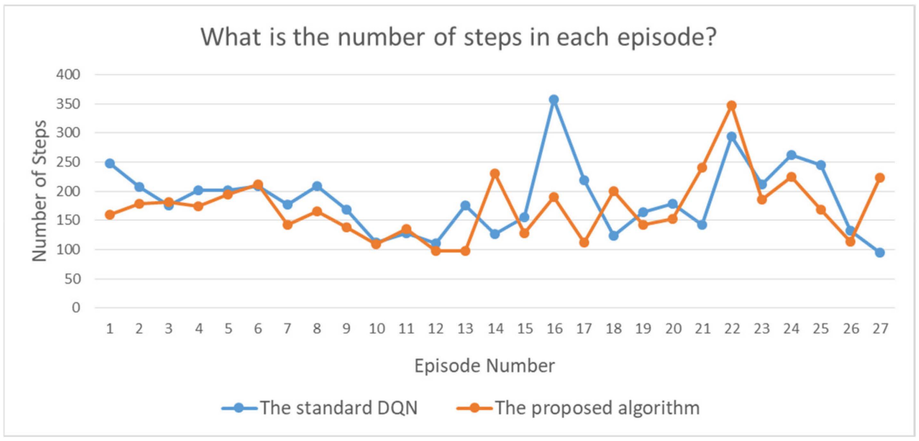
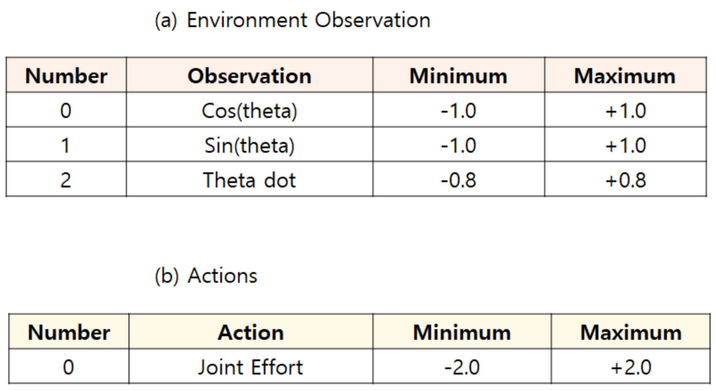


Figure 6. "What is the smallest step in every episode?" comparison between the standard DQN and the DQN with the proposed TC loss function.

4.2. "Pendulum"

In "pendulum" [26], the inverted "pendulum" starts in an arbitrary position, and the goal is to swing upward to remain vertical. Since the "pendulum" is an unresolved environment, there is no special reward threshold to be considered resolved. The “pendulum” environment has three observations and one individual action, as shown in Figure 7. There is also an exact equation for rewards: $-(\theta^2 + 0.1 \times \theta^2 + 0.001 \times \text{action}^2)$, where θ is normalized between $-\pi$ and $+\pi$. Therefore, the lowest cost is $-(\pi^2 + 0.1 \times 8^2 + 0.001 \times 2^2) = -16.2736044$ and the highest cost is θ [26]. The goal is to maintain zero degrees (vertical) with minimal rotational speed and minimal effort. DDPG with the proposed loss function was implemented using TensorFlow [23] and Keras [24] in OpenAI Gym [25].


(a) Environment Observation

Number	Observation	Minimum	Maximum
0	Cos(theta)	-1.0	+1.0
1	Sin(theta)	-1.0	+1.0
2	Theta dot	-0.8	+0.8

(b) Actions

Number	Action	Minimum	Maximum
0	Joint Effort	-2.0	+2.0

Figure 7. “pendulum” [26] in OpenAI Gym [25].

In “pendulum”, “when the episode ends” [26,27] did not exist. Therefore, all observations were made in terms of qualification, such as “How many above-average rewards happen in a period?” (for example, 200 time-step in an episode), as shown in Figure 8, “What is the maximum reward in every episode?” as shown in Figure 9, and “What is the median reward in every episode?” as shown in Figure 10. The standard deviations of both the DDPG with the proposed TC loss function and standard DDPG were compared, as shown in Figure 11. Since the reward started from -2000 and the time-step for the reward to reach 0 was typically approximately 200, the standard of a certain period was set to 200 time-step. The TC loss function proposed in this study significantly recorded the above-average rewards. The highest and median rewards were also noticeably higher than those of the standard DDPG. In particular, compared to the standard deviation, it could be predicted that the rewards tended to increase rapidly for a certain period of time-step. Moreover, as shown in Figure 12 “What is the accumulated reward per episode?” the proposed TC loss function could work appropriately until convergence.

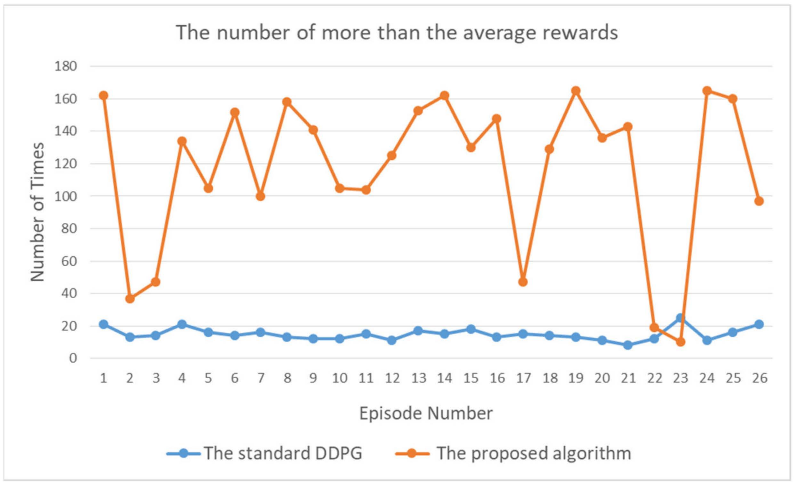


Figure 8. “How many above-average rewards happened?” comparison between the standard DDPG and the DDPG with the proposed TC loss function.

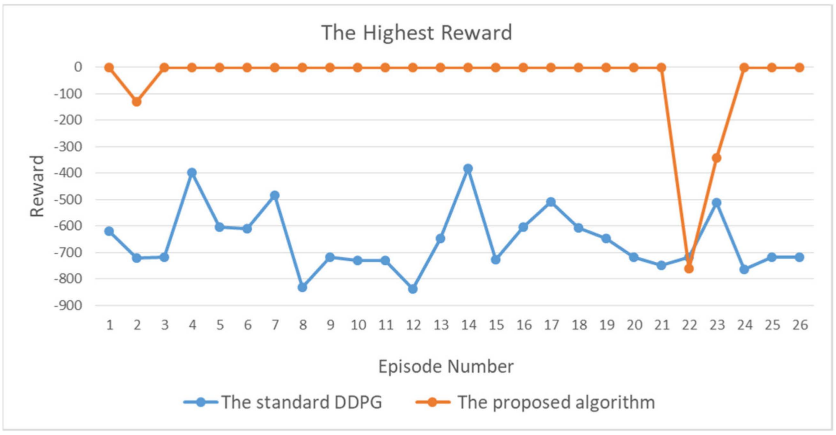


Figure 9. "What is the maximum reward in every episode?" comparison between the standard DDPG and the DDPG with the proposed TC loss function.

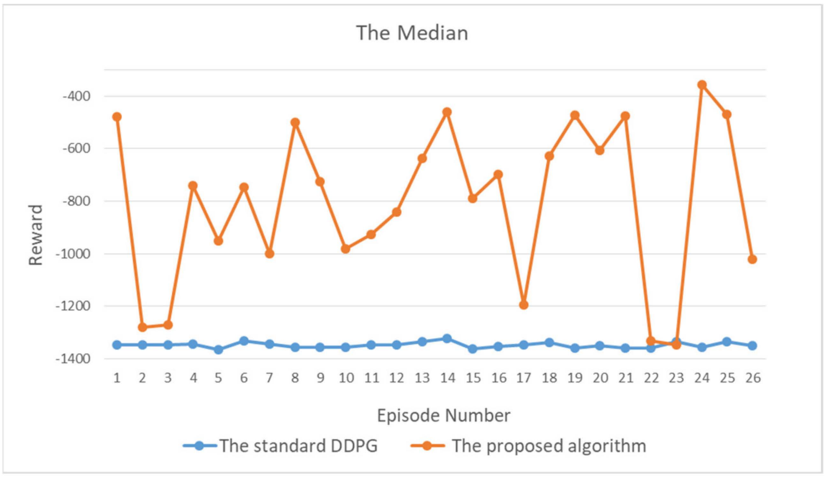


Figure 10. "What is the median reward in every episode?" comparison between the standard DDPG and the DDPG with the proposed TC loss function.

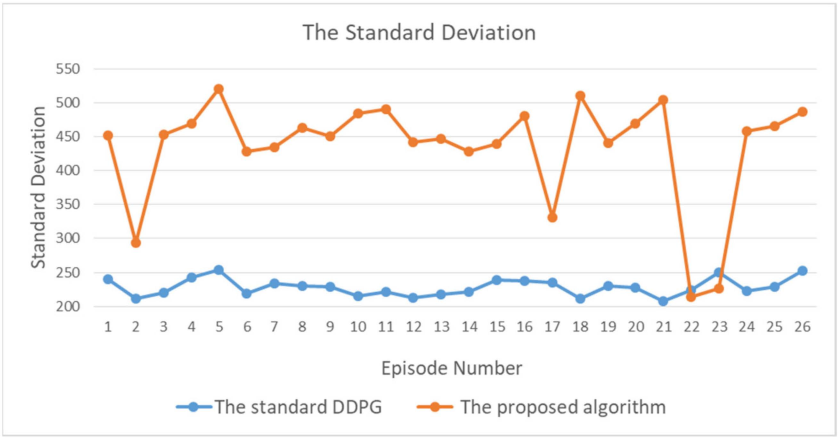


Figure 11. Standard deviation comparison between the standard DDPG and the DDPG with the proposed TC loss function.

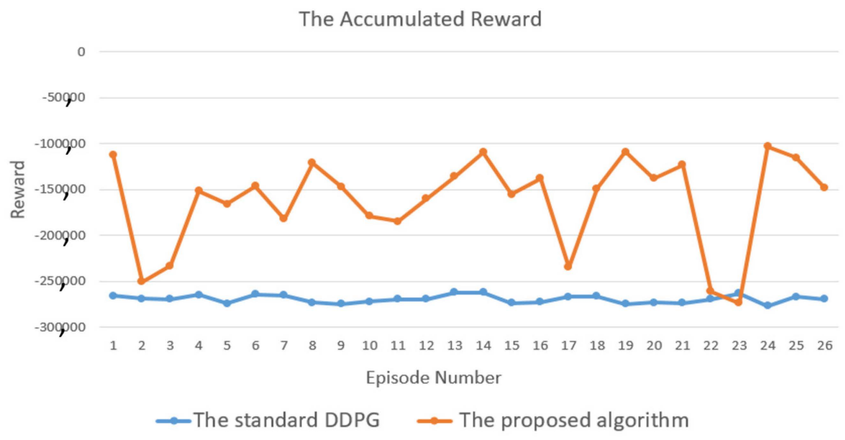


Figure 12. "What is the accumulated reward per episode?" comparison between the standard DDPG and the DDPG with the proposed TC loss function.

5. Discussion

The comparison of "cart-pole" as the representative of DQN and "pendulum" as the representative of DDPG indicated that the results in "pendulum" were significantly improved compared to those in "cart-pole". Studies conducted on various loss functions [15–17] indicated that the DQN with those loss functions gradually improved. In particular, the TC loss function proposed in this study showed better results in a continuous environment than in a discrete environment. Various methods have been proposed to improve different off-policy algorithms based on target network updates, particularly for DQN. Although this study was based on well-known algorithms such as DQN, it showed that the proposed TC loss function could be another suggestion for improving the performance, particularly for a different environment such as DDPG, and exploited as an additional component. However, we can find that the comparison by Wilcoxon-Signed-Rank-Paired Test shows the lack of performance in DQN can be dismissed. The proposed TC loss function showed a remarkable improvement in DDPG. This is different from [16]. Therefore, these efforts have led to a new TD algorithm, which is very valuable, particularly in a continuous area. Meanwhile, if there has been extensive research on updating loss functions to improve DQN, we might transmit such efforts to DDPG. Therefore, our next step is to study the results from various developed studies on DQN, which is the representative of the off-policy algorithm, such that it can be well applied to DDPG, which is another representative of the off-policy algorithm.

6. Conclusions

We proposed a novel TC loss function based on a previously developed TC loss and adapted the proposed TC loss function for target network updates for both DQN and DDPG, particularly for a critic network. Algorithms with the proposed TC loss function can be a family of target-based TD-learning. The target network update is used to deal with the mismatch between the estimate and the true value. Depending on the difference between the outputs of the learning agent and the target network, the target network update can be applied flexibly in both DQN and DDPG. We applied the proposed TC loss functions in DQN for a discrete environment and DDPG for a continuous environment for applications that can use both. Notably, the results in the continuous environment with DDPG significantly improved compared to those in the discrete environment with DQN. The proposed TC loss function in a well-known algorithm such as DQN did not exhibit extremely high performance. However, the lack of performance in DQN can be dismissed by Wilcoxon test. The proposed TC loss function exhibited a remarkable improvement in the environment with DDPG. This is significantly different from earlier studies. This can acquire enormously improved convergence speed and performance as a new TD algorithm, which is very valuable, particularly in a continuous environment. Therefore, we believe that the proposed TC loss functions could be useful in applications such as autonomous voltage control in power grid control and load shifting in a cooling supply system. Meanwhile, if extensive studies on improving DQN provide much better results, we can apply the efforts to DDPG through a correct adjustment because both DQN and DDPG are families of TD-learning off-policies.

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

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 1998; Volume 1.
2. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef] [PubMed]
3. 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]
4. 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.
5. 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]
6. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous control with deep reinforcement learning. In Proceedings of the 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, 2–4 May 2016; Available online: https://arxiv.org/abs/1509.02971 (accessed on 1 July 2021).
7. Watkins, C.J.C.H.; Dayan, P. Q-learning. *Mach. Learn.* **1992**, *8*, 279–292. [CrossRef]
8. Silver, D.; Lever, G.; Heess, N.; Degris, T.; Weirstra, D.; Riedmiller, M. Deterministic policy gradient algorithms. In Proceedings of the ICML'14 31st International Conference on Machine Learning, Beijing, China, 21–26 June 2014; Volume 32, pp. I-387–I-395.
9. Marcus, G. Deep Learning: A Critical Appraisal. *arXiv* **2019**, arXiv:1801.00631.
10. Duan, J.; Shi, D.; Diao, R.; Li, H.; Wang, Z.; Zhang, B.; Bian, D.; Yi, Z. Deep-Reinforcement-Learning-Based Autonomous Voltage Control for Power Grid Operations. *IEEE Trans. Power Syst.* **2020**, *35*, 814–817. [CrossRef]
11. Schreiber, T.; Eschweiler, S.; Baranski, M.; Müller, D. Application of two promising Reinforcement Learning algorithms for load shifting in a cooling supply system. *Energy Build.* **2020**, *229*, 110490. [CrossRef]
12. Lin, L.-J. *Reinforcement Learning for Robots Using Neural Networks*; Technical Report; School of Computer Science, Carnegie-Mellon University: Pittsburgh, PA, USA, 1993.
13. Kim, S.; Asadi, K.; Littman, M.; Konidaris, G. DeepMellow: Removing the need for a target network in deep Q-learning. In Proceedings of the 28th International Joint Conference on Artificial Intelligence, Macao, China, 10–16 August 2019; pp. 2733–2739. [CrossRef]
14. Durugkar, I.; Stone, P. TD Learning with Constrained Gradients. In Proceedings of the 31st Conference on Neural Information Processing Systems, NeurIPS 2017, Long Beach, CA, USA, 4–9 December 2017.
15. Pohlen, T.; Piot, B.; Hester, T.; Azar, M.-G.; Horgan, D.; Budden, D.; Barth-Maron, G.; Hasselt, H.-V.; Quan, J.; Vecerík, M.; et al. Observe and look further: Achieving consistent performance on Atari. *arXiv* **2018**, arXiv:1805.11593.
16. Ohnishi, S.; Uchibe, E.; Yamaguchi, Y.; Nakanishi, K.; Yasui, Y.; Ishii, S. Constrained Deep Q-Learning Gradually Approaching Ordinary Q-Learning. *Front. Neurorobot.* **2019**, *13*, 103. [CrossRef] [PubMed]
17. Lee, D.; He, N. Target-Based Temporal-Difference Learning. In Proceedings of the 36th International Conference on Machine Learning, Long Beach, CA, USA, 9–15 June 2019.
18. Hasselt, H.V.; Guez, A.; Silver, D. Deep reinforcement learning with double Q-Learning. In Proceedings of the AAAI'16: 30th AAAI Conference on Artificial Intelligence, Phoenix, AZ, USA, 12–17 February 2016; pp. 2094–2100.
19. Pavse, B.; Durugkar, I.; Hanna, J.; Stone, P. Reducing Sampling Error in Batch Temporal Difference Learning. In Proceedings of the 37th International Conference on Machine Learning, Online, 13–18 July 2020; Volume 119, pp. 7543–7552.
20. Uhlenbeck, G.E.; Ornstein, L.S. On the Theory of the Brownian Motion. *Phys. Rev.* **1930**, *36*, 82. [CrossRef]
21. Cart-Pole. Available online: https://gym.openai.com/envs/CartPole-v1/ (accessed on 28 September 2021).
22. cartpole_dqn.py. Available online: https://github.com/rlcode/reinforcement-learning-kr/blob/master/2-cartpole/1-dqn/cartpole_dqn.py (accessed on 28 September 2021).
23. Tensorflow. Available online: https://github.com/tensorflow/tensorflow (accessed on 28 September 2021).
24. Keras. Available online: https://keras.io/ (accessed on 28 September 2021).
25. OpenAI Gym. Available online: https://gym.openai.com/ (accessed on 28 September 2021).

26. Pendulum-V0. Available online: https://github.com/openai/gym/wiki/Pendulum-v0 (accessed on 28 September 2021).
27. pendulum_ddpg.py. Available online: https://github.com/dnddnjs/pendulum_ddpg/blob/master/pendulum_ddpg.py (accessed on 28 September 2021).