


Article

Enhanced DQN Framework for Selecting Actions and Updating Replay Memory Considering Massive Non-Executable Actions

Bonwoo Gu and Yunsick Sung * 

Department of Multimedia Engineering, Dongguk University-Seoul, Seoul 04620, Korea; griogrio@dgu.ac.kr

* Correspondence: sung@dongguk.edu

Abstract: A Deep-Q-Network (DQN) controls a virtual agent as the level of a player using only screenshots as inputs. Replay memory selects a limited number of experience replays according to an arbitrary batch size and updates them using the associated Q-function. Hence, relatively fewer experience replays of different states are utilized when the number of states is fixed and the state of the randomly selected transitions becomes identical or similar. The DQN may not be applicable in some environments where it is necessary to perform the learning process using more experience replays than is required by the limited batch size. In addition, because it is unknown whether each action can be executed, a problem of an increasing amount of repetitive learning occurs as more non-executable actions are selected. In this study, an enhanced DQN framework is proposed to resolve the batch size problem and reduce the learning time of a DQN in an environment with numerous non-executable actions. In the proposed framework, non-executable actions are filtered to reduce the number of selectable actions to identify the optimal action for the current state. The proposed method was validated in Gomoku, a strategy board game, in which the application of a traditional DQN would be difficult.

Keywords: Gomoku; game artificial intelligence; replay memory; Deep-Q-Network; reinforcement learning



Citation: Gu, B.; Sung, Y. Enhanced DQN Framework for Selecting Actions and Updating Replay Memory Considering Massive Non-Executable Actions. *Appl. Sci.* **2021**, *11*, 11162. <https://doi.org/10.3390/app112311162>

Academic Editors: Sławomir Nowaczyk, Mohamed-Rafik Bouguelia and Hadi Fanaee

Received: 21 October 2021

Accepted: 22 November 2021

Published: 24 November 2021

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



Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

The development of artificial intelligence technology has paved the way for varied improvements to virtual agents in virtual environments through active research [1]. Particularly, virtual agents are controlled in games by following predefined rules [2,3]. For example, the behavior tree (BT) algorithm defines each rule in a node and treats the entire set of rules as a tree to allow a virtual agent to consider a variety of rules [2]. Studies have been conducted on the genetic algorithm, via a combination of predefined rules, to reflect the rules of the most dominant genes in line with a changing environment [3,4]. When an agent is controlled based on rules, its performance generally depends on the level of segmentation of the corresponding rules.

DeepMind proposed the Deep-Q-Network (DQN), which combines a deep neural network (DNN) with Q-Learning [5]. The DQN controls Atari at the level of a game player; consequently, the DQN has demonstrated the potential to clear a game as a game player even if it receives only screenshots as inputs and controls the game based on them. However, two problems arise in the DQN. First, a limited number of the experience replays only recorded in replay memory are randomly selected and used; the DQN defines the concept of replay memory to learn the DNN through rewards [6]. Replay memory stores past gameplay experiences, allowing the DQN to avoid the local optimization problem of playing games in only one direction [7]; however, replay memory selects a limited number of experience replays according to an arbitrary batch size and updates them with the associated Q-function. Hence, relatively fewer experience replays of different states are used when the states of the randomly selected transitions become identical or similar.

It is recommended that the batch size is maximized to address the issue of the limited batch size in the DQN, and the most recently acquired experiences are selected in replay memory and used first [8]. The DQN may not be applicable in some environments where it is necessary to perform the learning process using more varied experience replays than is required by the limited batch size. In such cases, the Monte Carlo Tree Search (MCTS) algorithm can be used in the development [9], but its application becomes difficult in a more complex environment that would increase the complexity of the tree [10]. When the batch size of replay memory is limited, it is necessary to select experience replays in a manner that increases the learning effectiveness.

Second, there is a learning time issue due to non-executable actions. Among all Q-values of the current state, the action with the highest Q-value is selected as the final action in the traditional DQN. Because the traditional DQN is a model-free algorithm with an unknown execution of action, a problem of an increasing amount of repetitive learning occurs as more non-executable actions are selected. A method is required to distinguish and select executable actions based on the environment data corresponding to a smaller cost of acquiring the environment data to distinguish non-executable actions, rather than a smaller cost incurred by longer learning time.

In this study, an enhanced DQN framework is proposed to resolve the batch size problem and reduce the learning time of a DQN in an environment with numerous non-executable actions. First, the replay update approach is suggested. When a new transition is recorded in replay memory, the state most similar to a new state of the transition is found in replay memory with the approximate nearest neighbor (ANN) algorithm, and the corresponding target value is updated. This allows all experience replays stored in replay memory to be used, rather than using only the recorded experience replays randomly selected by the batch size, as in the traditional DQN. Second, the action selection method of the DQN is improved. In the proposed enhanced DQN framework, executable actions are filtered to reduce and identify the number of selectable and optimal actions for the current state. In a traditional DQN, the amount of unnecessary, repetitive learning increases with the number of non-executable actions. In contrast, the proposed enhanced DQN framework reduces the amount of learning by identifying whether an action is executable and also decreasing the overall number of selectable actions. The proposed method was validated in Gomoku, in which the application of the traditional DQN would be difficult. As the game progresses in Gomoku, the number of locations available for placing stones decreases and so does the number of executable actions. The proposed method was applied to the white stones, whereas the genetic algorithm-based Gomoku AI was applied to the black stones.

The rest of this paper is organized as follows. Section 2 describes related works on game AI model and reinforcement learning model. Section 3 describes an enhanced DQN action framework. In Section 4, the experiments that were conducted using the proposed method on representative complex environment, Gomoku, are described. Finally, Section 5 concludes summarizing our main approaches and future research directions.

2. Related Works

In virtual environments, optimal action should be taken in games to provide users with fun and interest within a limited environment that does not violate the rules of the games. Often used as a game AI model, a genetic algorithm is an evolved algorithm that is also used as a Gomoku AI model, [3]. Shah et al. created a Gomoku AI model by applying a genetic algorithm [4]. Although a tree-based Gomoku AI can be created, a bottleneck may occur with an increasing depth of the tree [11], which causes the tree search to fail. Wang et al. resolved this bottleneck issue resulting from a large depth of search by applying the genetic algorithm [12].

Owing to the recent development of game engines, developers can access tree search-based algorithms and easily create game AI [13]. For example, BT provides a method to structure the rules of an agent's actions [14]. BT is widely used in computer games and robotics because it allows efficient programming and structuring of the complex rules

of modular and reactive AI models. Allis et al. developed a Gomoku AI model called Victoria [15] using Threat-Space Search [16] and Proof-Number Search [17]. If Victoria moves a stone first, it always leads to a win. Cao et al. used an algorithm combining Upper Confidence Bounds applied to Trees (UCT) [18] and Adaptive Dynamic Programming (ADP) [19] and introduced a Gomoku AI model that could solve the problem of search depth defects more accurately than using a single UCT [20].

Because the performance of the algorithm combining convolutional neural networks (CNNs) [21] and Monte Carlo Tree Search (MCTS) [9] has been demonstrated by DeepMind's AlphaGo, the application of this algorithm in a Gomoku AI model has also been described. Li et al. created a DNN-based Gomoku AI model using MCTS and a CNN [9,22]. Yan et al. described the Gomoku AI model using a hard-coded convolution-based Gomoku evaluation function [23]. Shao et al. described a Gomoku AI model that predicts the position of the next stone with an accuracy of 42% through a DNN with various hyper-parameters [24].

Recently, game AI models based on reinforcement learning have been developed to clear games with minimum amounts of information required to control their progress. Oh et al. used deep reinforcement learning to describe a 62% real-time fighting game AI model against five professional gamers [25]. Tang et al. introduced a Gomoku AI model that combines MCTS and ADP to eliminate the "short-sighted" defect of the neural network evaluation function [26]. Zhao et al. designed a Gomoku AI model that is capable of self-teaching based on a policy that penalizes the last action of the ADP and the loser while rewarding that of the winner [27]. Gu et al. introduced a reinforcement learning algorithm that provides a CNN-based next best action by combining one-hot encoding-based vectors [28]. The proposed model combines one-hot encoding-based vectors by clustering them in an ANN by similar sentence. The application of the proposed model in Gomoku AI results in a win after 525 games when the ANN distance is 5. Furthermore, the amount of learning data is reduced by 12%. The ANN is also used in this study to cluster Q-values by similar state.

A model-free algorithm indicates the next state of AI in terms of probability when it executes an action in a certain state [29]. Lopez et al. introduced a KNN-td algorithm that combines the K-nearest Neighbor (KNN) algorithm [30] with Temporal Difference (TD) [31,32].

In Q-learning, which is another model-free algorithm, a Q-function is used to calculate the action appropriate for the current state [33]. A Q-table is used to update the action-values of each state in Q-learning; however, it is difficult to use the Q-table when there are numerous states or different types of corresponding actions [34]. Sung, Y. et al. proposed the Reward Propagation Method (RPM) to reduce the time required for Q-learning without reducing the size of the state [35]. To address this limitation of the Q-table, DeepMind's Volodymyr et al. implemented a neural network in a DQN algorithm and applied the algorithm to seven Atari 2600 games with complex states to achieve the capability to play at a professional level [5].

Replay memory that stores past game experiences is used in DQNs. Von Pilchau et al. proposed the Interpolated Experience Replay and applied it in the FrozenLake game. Compared to the traditional DQN, very fast learning speeds and high learning outcomes were achieved [36]. Schaul et al. developed a framework that prioritizes past game experiences in replay memory. This produce better performance in 41 out of 49 Atari games in comparison to the traditional DQN [37].

To address the limitation of replay memory in the traditional DQN, an experience replay selection approach is generally introduced to produce the optimal learning outcome; however, an optimal action cannot be chosen from experience replays that have not been selected in the replay memory according to the batch size. In this study, a method is proposed to resolve the batch size limitation by applying the ANN algorithm when recording experience replays in replay memory and to reduce the actions to be selected, thus limiting the amount of learning in an environment where a massive non-executable action occurs.

3. Action Selection and Replay Update for Non-Executable Massive Actions

In this study, an enhanced DQN model is proposed to select the optimal actions while resolving the problems associated with the batch size of replay memory and the amount of DNN learning in an environment with multiple non-executable and executable actions. The proposed method allows an agent to make optimal decisions based on the DQN, even in an environment with a high number of non-executable actions.

3.1. Framework Overview

An agent learns by interacting with the environment. In the environment, the state s_t is sent to the agent. The received action a_t is executed, and the state s_{t+1} and the reward r_t are sent. The agent consists of the DQN Action Estimator, action filter, and replay memory updater as shown in Figure 1. In the DQN Action Estimator, the Q-values of the current state s_t are estimated, similarly to the traditional DQN algorithm. In the action filter, the optimal action among the Q-values estimated through the DNN is determined by taking into account the state s_t . Specifically, the Q-values of the non-executable actions among the selectable actions in the state s_t are set to $-\infty$ such that they are not considered during the selection of the action a_t . The action a_t determined in the action filter is sent to the environment. In the replay memory updater, an ANN algorithm is used to determine whether there is a state most similar to the state s_t among those of the experience replays stored in the replay memory. The target values are updated based on this, and the Q-function is updated based on the states and target values stored in the replay memory. Specifically, the target is generated and updated based on s_t , s_{t+1} , $Q(s_t, a)$, and r_t and is stored in replay memory D .

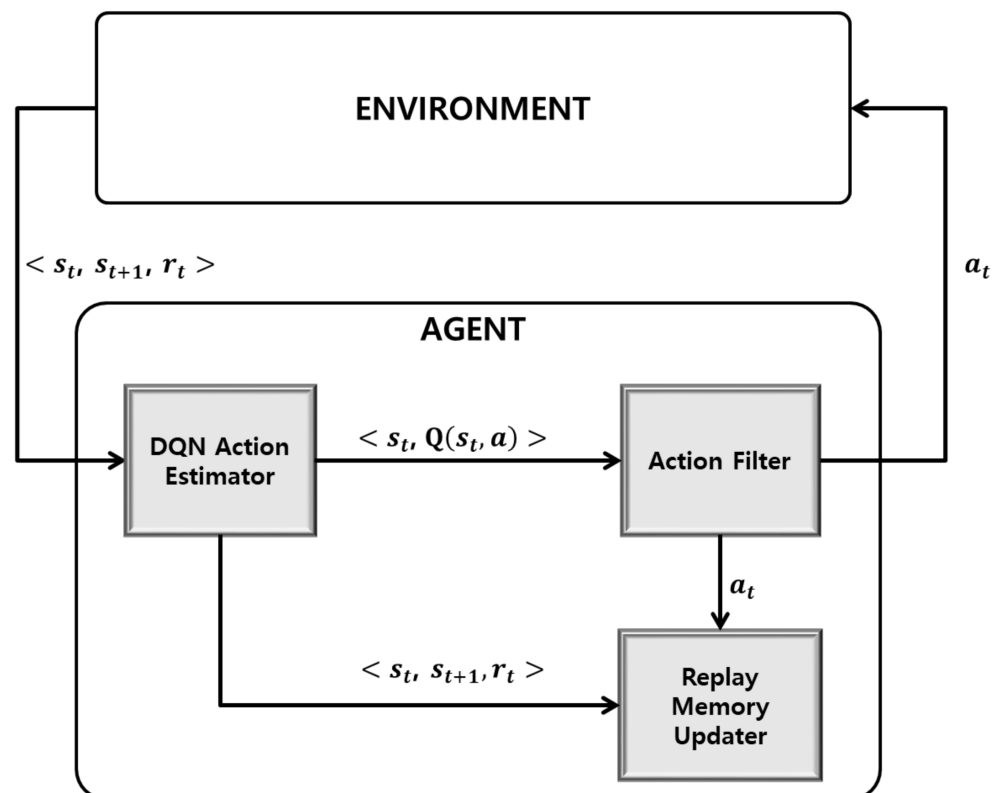


Figure 1. Proposed enhanced DQN framework.

Because the action estimator and action filter are performed together in the traditional DQN, and the action with the largest Q-value is selected among those that are predefined, it is possible to select an action that cannot be executed in practice; however, actions that cannot be executed in practice are excluded through the action filter in the proposed

method, which reduces the number of action selections, and ultimately, the amount of learning. Table 1 shows the procedure of the *Agent Process* in the proposed DQN. Replay memory D is initialized as an empty set. The s_t , s_{t+1} , and reward r_t are received in the environment, and the action a_t is repeatedly executed.

Table 1. Pseudocode of Agent Process.

PROCEDURE <i>Agent_Process</i>
BEGIN SET $D \leftarrow \emptyset$ FOR each episode FOR each time SET $s_t \leftarrow \text{CALL State_Receiver}$ SET $Q(s_t, a) \leftarrow \text{CALL Action_Estimator with } s_t$ SET $a_t \leftarrow \text{CALL Action_Filter with } s_t, Q(s_t, a)$ SET $s_{t+1}, r_t \leftarrow \text{CALL Action_Executer with } a_t$ CALL Replay_Memory_Updater with } D, s_t, s_{t+1}, a_t, r_t END FOR END FOR END

3.2. Action Filter

In the traditional DQN, $\text{argmax}_a Q(s_t, a)$ is defined as the action a_t to be executed at time t . In this study, the action filter prevents an agent from selecting non-executable actions by considering the state s_t of all actions that can be selected by the agent and identifying non-executable actions. Table 2 shows the procedure of the action filter process. The action filter determines whether an action can be executed in the state s_t and sets the mask $M(s_t, a)$ of the non-executable location to $-\infty$ depending on the transition of the state. An action that results in the largest sum between $Q(s_t, a)$ and $M(s_t, a)$ at the state s_t is set as the action a_t .

Table 2. Pseudocode of action filter.

PROCEDURE <i>Action_Filter</i>
INPUT: $s_t, Q(s_t, a)$ OUTPUT: a_t BEGIN Initialize $M(s_t, a)$ by 0 \forall_a SET $M(s_t, a) \leftarrow -\infty$, where a is not executable at s_t SET $a_t \leftarrow \text{argmax}_a (Q(s_t, a) + M(s_t, a))$ END

All Q-values within the replay memory updater are dependent on each other, rather than independent, because they are clustered into similar states through the ANN algorithm. Hence, the action with the maximum $(Q(s_t, a) + M(s_t, a))$ becomes the final action a_t at s_t , with all other actions that cannot be executed filtered by the action filter.

3.3. Replay Memory Updater

Because the replay memory randomly extracts the states and target values of the experience replays by the batch size and updates with the Q-function, the following problems occur. First, different target values can be recorded for similar or identical states. Consequently, the Q-function update may be error-prone when the batch size becomes larger. Second, there are cases where the action with the highest Q-value cannot be executed. Particularly, in an environment where a massive non-executable action occurs, the experience replay of such an action that cannot be executed in practice is recorded in the

replay memory. Therefore, if the batch size is limited, the experience replay corresponding to an executable action cannot be selected relatively in replay memory.

The proposed DQN-based reinforcement learning model shares a single target value by clustering similar, dependent states using the ANN algorithm. This provides space for additional dissimilar experience plays to be recorded in the replay memory. Moreover, because target values are updated for similar states, the Q-values of the actions output from the Q-function that learns these target values are similar and dependent to each other. Regarding the image classification research field, a CNN demonstrates higher performance than other traditional machine learning algorithms [38] and can be used to cluster similar states; however, the use of a CNN requires relatively more computation and memory than an ANN, and ANN learning must be repeated. This can lead to forgetting problems [39], causing a significant loss in performance. In addition, similarity generally cannot be measured in a CNN. For this reason, the proposed method uses an ANN algorithm, instead of a CNN.

Table 3 shows the procedure of the updated replay memory process with ANN. To identify similar states with the ANN, the maximum allowable difference in similarity is defined as the distance δ . The proposed DQN-based reinforcement learning model compares the individual states recorded in replay memory D to the state s_t using the ANN to find and process the most similar state. Similarity is calculated in terms of a Euclidean distance. When the similarity is greater than the distance δ or corresponds to the first experience replay, a new state is assumed, and a new target value y_t is calculated for DQN learning. When the similarity is smaller than δ and a similar state exists in replay memory D , y_i is updated. Action a_i is not updated because the ultimately selected action a_i may be different each time through the action filter even when the state is similar. If a_i is updated, the Q-value of the Q-function will be different each time. Because the purpose of the action filter is to find the optimal action by referring to the current state, a_i is not updated to avoid affecting the current state in the Q-function update. Once the update of replay memory is complete, the Q-function is updated.

Table 3. Pseudocode of replay memory updater.

PROCEDURE <i>Replay_Memory_Updater_with_ANN</i>	
Input: s_t, s_{t+1}, a_t, r_t	
BEGIN	
IF D is \emptyset	
SET $y_t \leftarrow Q(s_t, a) \cdot (1 - \alpha) + (r_t + \gamma \max_{a'} Q(s_{t+1}, a')) \cdot \alpha$	
SET $D \leftarrow D \cup \{[s_t, a_t, r_t, y_t, s_{t+1}]\}$	
ELSE	
# The Starting of ANN	
SET $d \leftarrow \min(s - s_t)$ where s is from D	
SET $i \leftarrow \operatorname{argmin}_i (s_i - s_t)$ where s_i is from D	
IF $d > \delta$	
SET $y_t \leftarrow Q(s_t, a) \cdot (1 - \alpha) + (r_t + \gamma \max_{a'} Q(s_{t+1}, a')) \cdot \alpha$	
SET $D \leftarrow D \cup \{[s_t, a_t, r_t, y_t, s_{t+1}]\}$	
ELSE	
SET $y_i \leftarrow y_i \cdot (1 - \alpha) + (r_t + \gamma \max_{a'} Q(s_{t+1}, a')) \cdot \alpha$	
END IF	
# The End of ANN	
END IF	
SET $l \leftarrow \frac{1}{ D } \sum_{j=1}^{ D } (y_j - Q(s_j, a_j))^2$ where y_j, s_j, a_j from D	
Update QFunction using the gradient descent algorithm by minimizing the loss l	
END	

4. Experiments

Experiments were performed by applying the proposed DQN algorithm using replay memory with an ANN to Gomoku games. Section 4.1 describes the learning and development environments, whereas Sections 4.2 and 4.3 present the performance verification of the proposed enhanced DQN-Framework-based Gomoku AI without an action filter. Sections 4.4 and 4.5 present the performance verification of the proposed enhanced DQN-Framework-based Gomoku AI with an action filter.

4.1. Experimental Environment

We verified the performance through various methods to confirm the performance of the Gomoku system proposed herein. We performed experiments with “the number of wins without an action filter,” “the number of batch sizes of replay memory without an action filter,” “the number of wins with an action filter,” and “the number of batch sizes of replay memory with an action filter” on the Gomoku game. In this study, the experiment was developed with C/C++, Python 3.0, and TensorFlow. A CPU with an I-7 processor, 16GB RAM, and a GTX-1050 GPU were used in the experimental environment.

In this study, a 7×7 board is used in the Gomoku games. The neural network of the proposed DQN consists of three hidden layers. Each hidden layer consists of 49,128,516 neurons. The output and input layers were designed to have 49 neurons, which corresponds to the total number of intersections on the board.

In the experiments, 1000 Gomoku games were iterated for each experimental case. We calculated the winning rate by grouping 1000 experimental data into ten categories. A Gomoku board is utilized as a DQN’s environment. States are described by the status of the Gomoku board. Each game was forced to start at the (4, 4) position. The proposed enhanced DQN framework was applied to the white stones (DQN’s Agent). The genetic algorithm was applied to the black stones. If black stones form the first five-in-a-row, it will be considered as a win for the black stones. Therefore, the goal of white stones is to prevent black stones from forming a five-in-a-row. If white stones form the first five-in-a-row or fill up all 49 cells such that the game can no longer proceed, it will be a win for the white stones.

4.2. Number of Winning Games without Action Filter

This section describes the results of the Gomoku game experiment without an action filter applied to the proposed enhanced DQN framework. Because the next-best-action function is unavailable without the action filter, reinforcement learning will be performed until another action is selected, if there is a stone in the current state.

Table 4 shows the reward policy for the DQN without an action filter. In Gomoku, the pattern of play changes consistently depending on the position of the stones each player has placed. Consequently, if a stone could not be placed because there was another stone already placed, a reward of -0.5 would be applied until a different action was selected. Additionally, if the player made the incorrect final action and the opponent won, a reward of -1 would be applied. If the player made the correct final action and won, a reward of $+1$ would be applied.

Table 4. Reward policy without action filter applied.

When the Opponent Wins (Black Win)	When the Player Wins (White Win)	When the Player Can’t Place Stone
-1 reward	$+1$ reward	-0.5 reward

Figure 2 shows the results of the experiment in which the traditional DQN was applied to Gomoku games in this study. In the traditional DQN, the content from random past plays—as many as the batch size—is remembered in the replay memory, and the Q-function is executed through DNN learning. In Gomoku games, if the opponent is in the same state but makes a different action, the replay memory will accumulate completely different actions for the identical state. It is impossible to determine the optimal action in DNN learning when completely different actions are recorded for the same state. As shown in Figure 2, the batch size was varied from 5 to 25 in this experiment. If batch size is over 30, it is impossible for DNN to learn optimal actions in the Gomoku game because of multiple different executed actions in the same state.

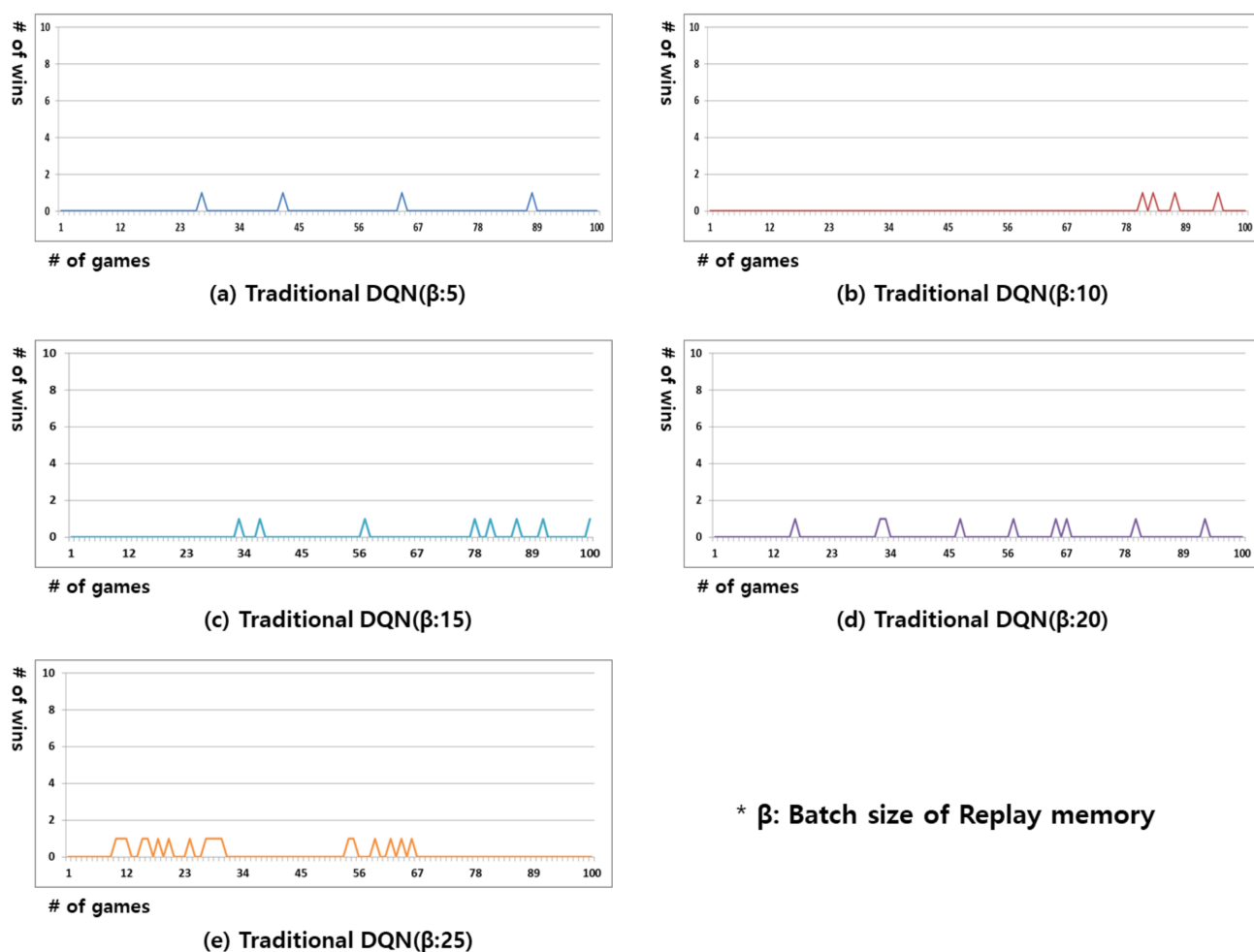


Figure 2. Result of win rate for the Gomoku game in the traditional DQN. From (a–e) replay memory’s batch size is 5, 10, 15, 20, and 25.

The results of the experiment in Figure 2 show that the winning rate was low regardless of the batch size. As demonstrated in the results, it was impossible to respond to all actions of the opponent by learning randomly extracted past data. Moreover, it was unknown whether the learning model could win the subsequent game after winning the current game. This is because the opponent could also play the game in various ways, and it would be very rare to have a state that is identical to the past game.

Figure 3 shows the results of the enhanced DQN framework without an action filter applied. Instead of randomly extracting past data from the replay memory, the ANN algorithm proposed in this study clusters similar states together. Through this, the batch size of replay memory can be increased without a limit. In addition, because the same target is updated between similar states, a single action can be set for those similar states.

All the experimental results in Figure 3 indicate that the winning rate was higher compared to the results in Figure 2. This implies that the experience replays clustered through the ANN algorithm have been properly structured to facilitate DNN learning; however, a new state arises every time the opponent makes a different action. As a result, the winning rate in all of the experiments in Figure 3 is not maintained but rather alternates between winning and losing. Figure 3a,e exhibit the most and least efficient results, respectively. The greater the ANN distance, the wider the range of experience replays considered to be similar. This results in a higher number of clustered experience replays, which is a poor identification of the state.

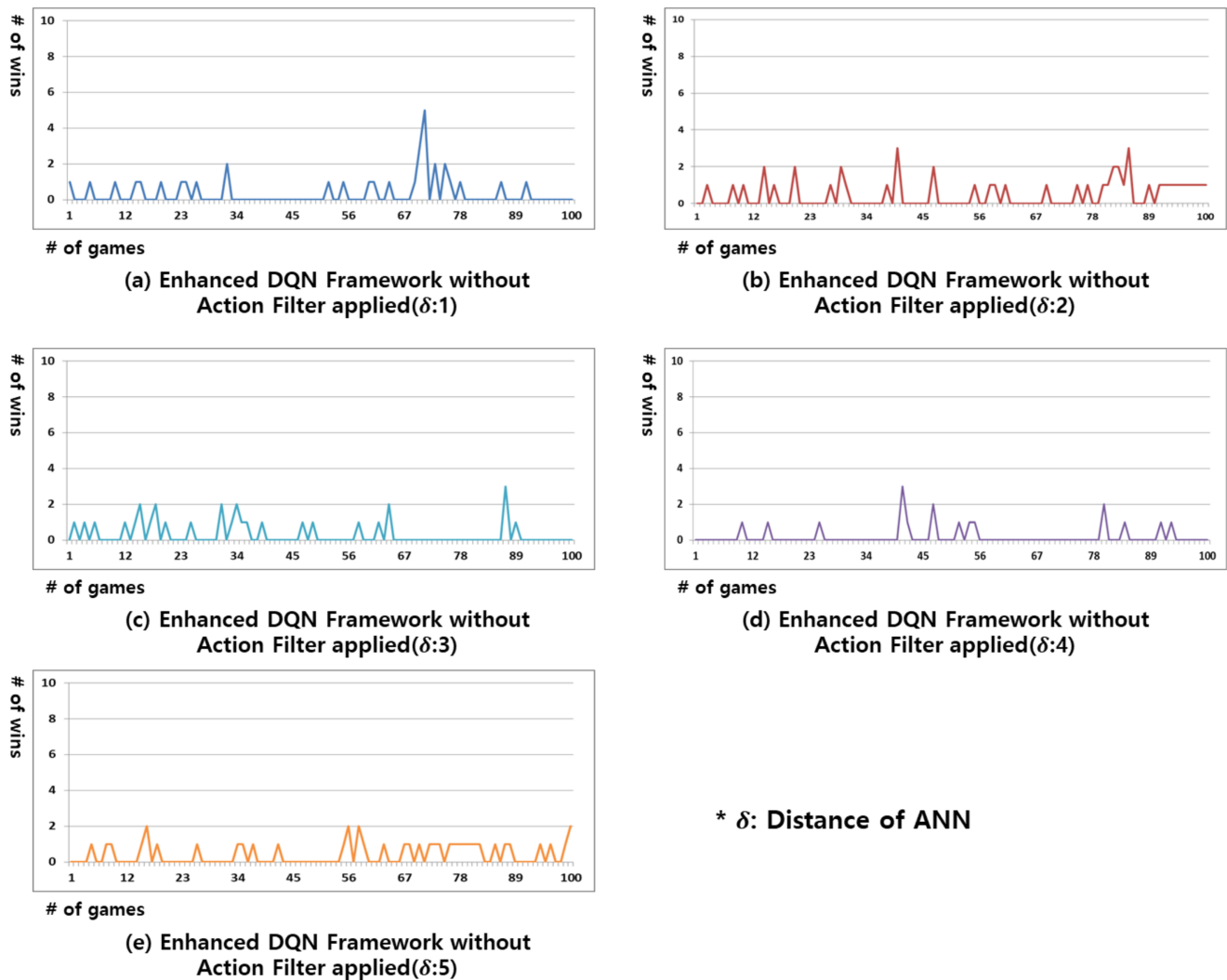


Figure 3. Result of win rate for the Gomoku game in the proposed enhanced DQN framework without action filter applied. From (a–e) replay memory’s distance of ANN is 1, 2, 3, 4, and 5.

4.3. Batch Sizes of Replay Memory without Action Filter

In this section, we confirm the number of experience replays stored in the replay memory without an action filter applied in the proposed enhanced DQN framework. In the existing replay memory, experience replays were selected based on the batch size specified by the user to update the Q-function. However, in the enhanced DQN framework proposed in this study, experience replays are accumulated for dissimilar states in replay memory, and the batch size increases according to the state in the game.

Figure 4 shows the batch sizes without an action filter applied in the proposed enhanced DQN framework. An ANN distance of 1 resulted in the largest batch size. On the other hand, the batch size was the smallest with a large ANN distance. The result in Figure 3a was the most efficient in Figure 3 because experience replays that had been organized for dissimilar states were used. As shown in Figure 4, the proposed enhanced DQN framework could resolve the batch size limitation of replay memory.

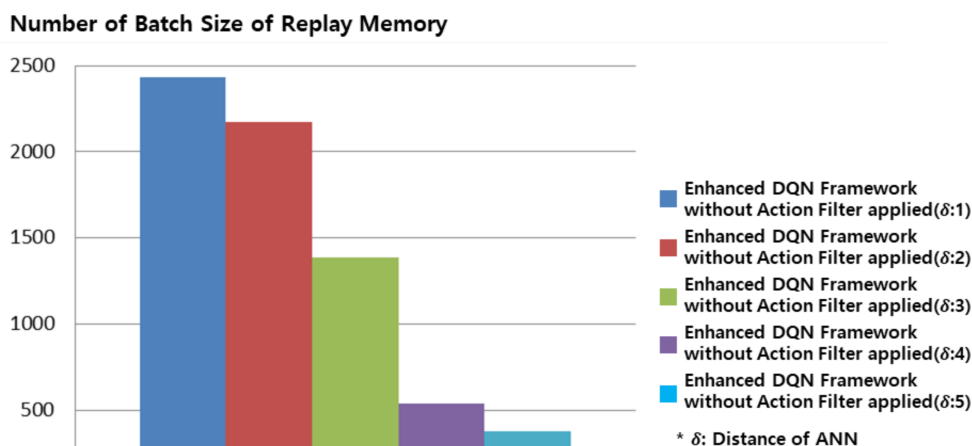


Figure 4. Replay memory batch size for the Gomoku game when proposed enhanced DQN framework without action filter is applied. From replay memory ANN distance is 1–5 and replay memory batch size reduced from 2433 to 378.

4.4. Number of Winning Games with Action Filter

This section provides the results of the Gomoku game experiment with an action filter applied to the proposed enhanced DQN framework. Because the next best action is made possible by the action filter, the presence or absence of a stone in the current state is not considered for the reward. Table 5 shows the reward policy of the proposed enhanced DQN framework with an action filter applied to decision making. If the player made the incorrect final action and the opponent won, a reward of -1 would be applied. If the player made the correct final action and won, a reward of $+1$ was applied.

Table 5. Reward policy with action filter applied.

When the Opponent Wins (Black Win)	When the Player Wins (White Win)
-1 reward	$+1$ reward

Figure 5 shows the results of the Gomoku games played with an action filter applied to the decision making of the traditional DQN. In the traditional DQN, there is no next best action available as similar states are not clustered together. Therefore, even when action filter is applied, the experiment results are similar to those in Figure 2.

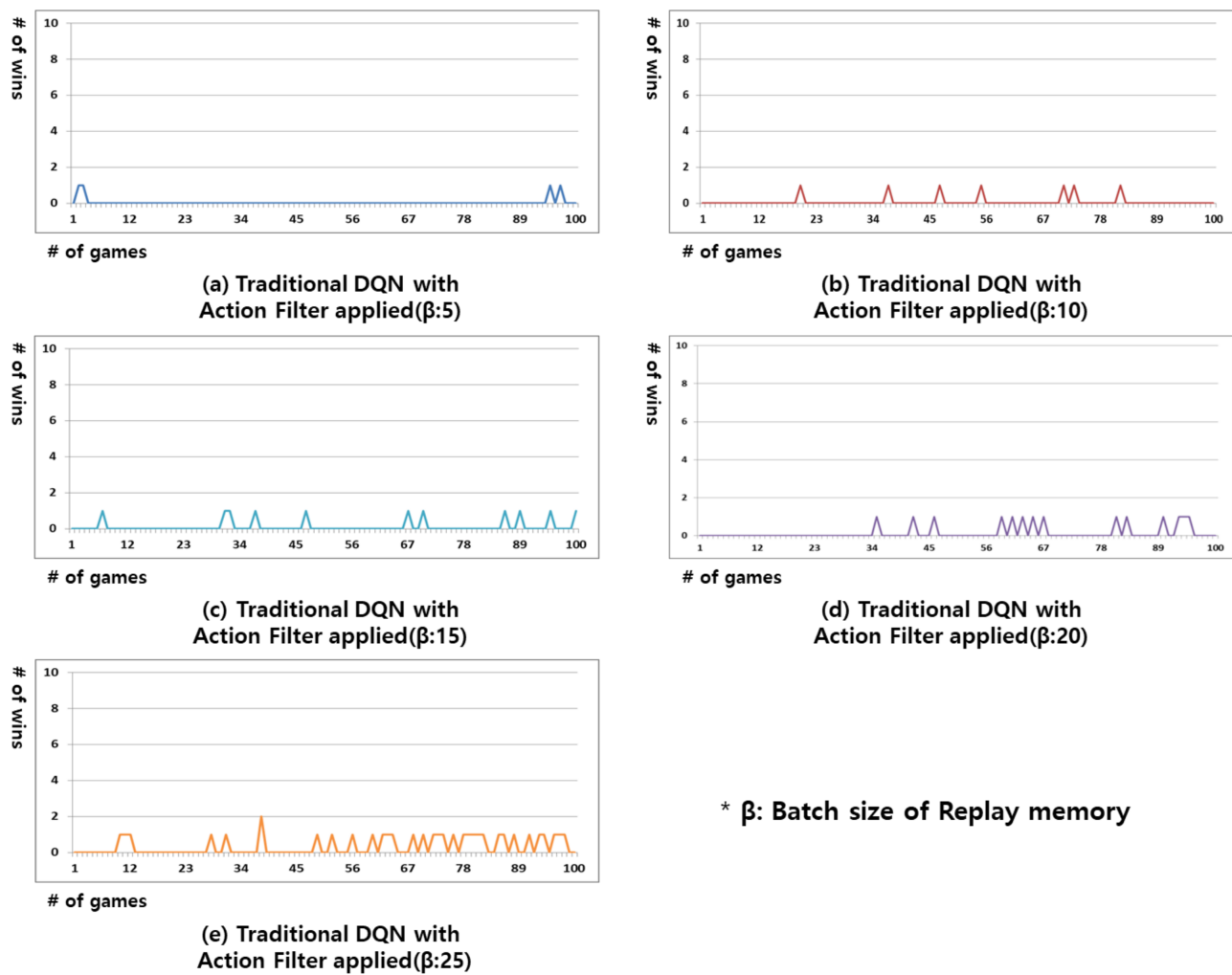


Figure 5. Result of win rate for the Gomoku game in the traditional DQN with action filter applied. From (a–e) replay memory’s batch size is 5, 10, 15, 20, and 25.

Figure 6 shows the results of the experiment in which an action filter is applied to the decision making of the proposed enhanced DQN framework. For the targets that are clustered by similar states through the ANN algorithm, the Q-values of the individual actions are in a dependent relationship, rather than an independent one. Thus, if the action with the current highest Q-value cannot be selected by the action filter, the action with the second highest Q-value becomes the next best action in the current state. The experimental results in Figure 6 show a significantly higher winning rate than those in Figures 2, 3 and 5. As shown in Figure 6c, the winning rate in this experiment is the highest with an ANN distance of 3 (green box); however, if the ANN distance is unnecessarily small, as seen in Figure 6a, similar states cannot be properly clustered together. If the ANN distance is larger than necessary, as in Figure 6e, the criterion for identifying similar states becomes too broad and different states may be mistakenly considered to be similar and clustered together, resulting in inaccurate identification of the state. The experiment in Figure 6 confirms that the winning rate cannot be maintained if the ANN distance is unnecessarily large or small.

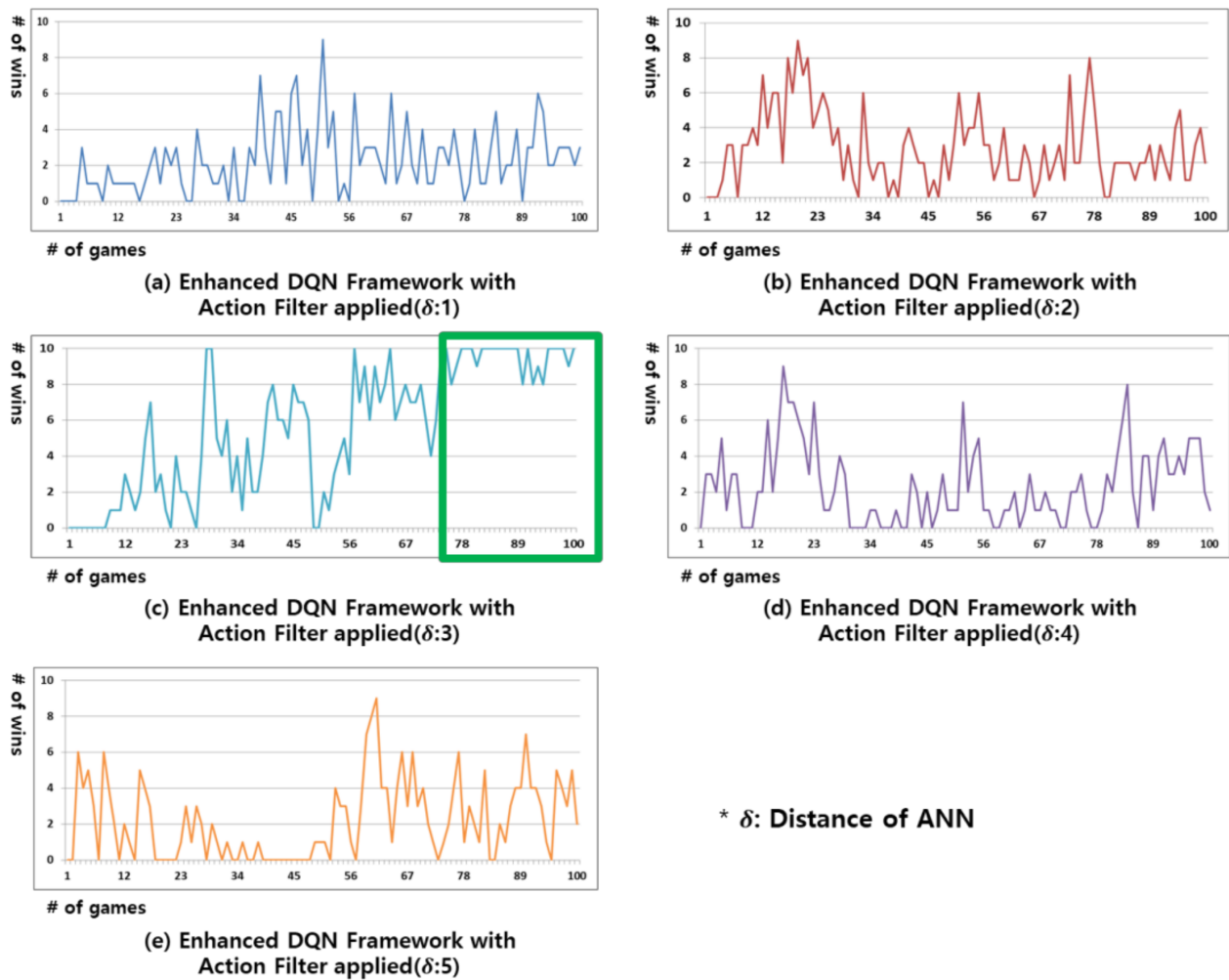


Figure 6. Result of win rate for the Gomoku game in the proposed enhanced DQN framework with action filter applied. From (a–e) replay memory’s distance of ANN is 1, 2, 3, 4, and 5.

4.5. Number of Batch Sizes of Replay Memory with Action Filter

In this section, we confirm the number of experience replays stored in replay memory with an action filter applied in the proposed enhanced DQN framework. Because the action filter is not applied in Section 4.3, the replay memory needs to be updated every time a stone is placed in a position that already has a stone; however, the number of replay memory updates can be reduced with the application of the action filter.

Figure 7 shows the batch sizes with an action filter applied to the decision making of the proposed enhanced DQN framework. The results in Figure 7 are similar to those in Figure 4; however, the amount of ANN learning is reduced by the ability to choose the next best action with an action filter. While 1386 batch data were required in Figure 4c, only 99 batch data were used to achieve a high winning rate in Figure 7c with an action filter applied. This experiment demonstrates that the proposed enhanced DQN framework and action filter allow an effective action to be selected in an environment that limits the selection of actions.

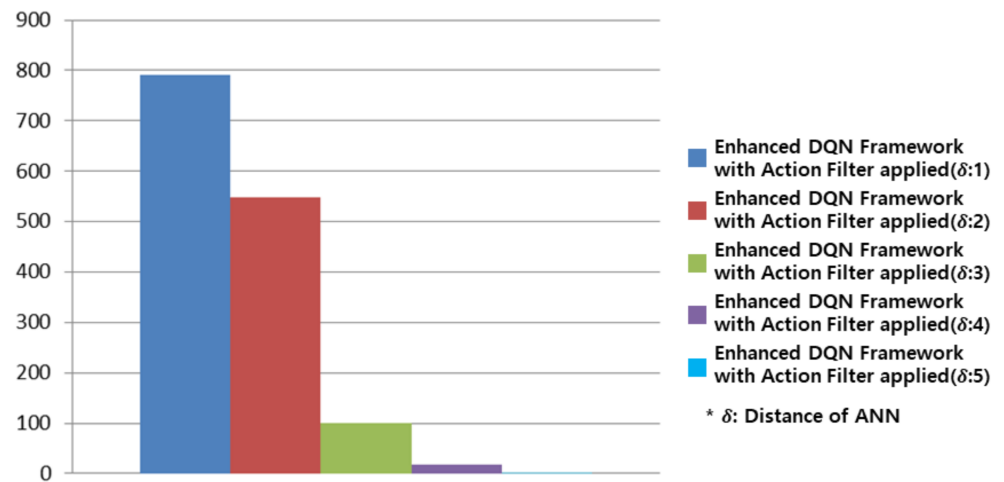
Number of Batch Size of Replay Memory

Figure 7. Number of replay memory's batch size for the Gomoku game in proposed enhanced DQN framework with action filter applied. From replay memory's distance of ANN is from 1 to 5 and replay memory's batch size reduced from 790 to 2.

5. Conclusions

In this study, a DQN-based reinforcement learning algorithm is proposed to select the next best action in an environment where the selection of actions is limited. In the proposed enhanced DQN framework, an ANN algorithm was applied to the replay memory to cluster experience replays by similar states such that they would no longer be in an independent relationship but rather clustered in a dependent relationship. Consequently, the relationship between the target values required to update the Q-function also became dependent. By applying an action filter to the decision making of DQN, it was possible to select the next best action that would consider the current state of a situation where a non-executable action existed. The proposed enhanced DQN framework was applied to Gomoku games for validation. On the one hand, when the action filter was not applicable, an ANN distance of 1 achieved the highest efficiency, but the winning rate was not maintained because it was impossible to respond to all actions of the opponent. On the other hand, when the action filter was applicable, the winning rate could be maintained by responding to all actions of the opponent when an ANN distance of 3 was achieved. Moreover, the amount of DNN learning was significantly reduced. In the future, we plan to apply and test the DQN algorithm proposed for other numerous virtual environments. We will also address and investigate the problems associated with replay memory resulting from these specific environments.

Author Contributions: B.G. is the main author who was in charge of writing this manuscript and doing experiments. Y.S. revised and supervised this manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Dongguk University Research Fund of 2020. This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2018R1D1A1B07049990).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not Applicable.

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

References

- Jeong, D.K.; Yoo, S.; Jang, Y. VR sickness measurement with EEG using DNN algorithm. In Proceedings of the 24th ACM Symposium on Virtual Reality Software and Technology, Tokyo, Japan, 28 November–1 December 2018; pp. 1–2.
- Kusyk, J.; Sahin, C.S.; Uyar, M.U.; Urrea, E.; Gundry, S. Self-organization of nodes in mobile ad hoc networks using evolutionary games and genetic algorithms. *J. Adv. Res.* **2011**, *2*, 253–264. [\[CrossRef\]](#)
- Marks, R.E. Playing games with genetic algorithms. In *Evolutionary Computation in Economics and Finance*; Physica: Heidelberg, Germany, 2002; pp. 31–44.
- Shah, S.M.; Singh, D.; Shah, J.S. Using Genetic Algorithm to Solve Game of Go-Moku. *IJCA Spec. Issue Optim. On-Chip Commun.* **2012**, *6*, 28–31.
- 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.
- Fan, J.; Wang, Z.; Xie, Y.; Yang, Z. A theoretical analysis of deep Q-learning. In Proceedings of the Learning for Dynamics and Control, PMLR, Online, 11–12 June 2020; pp. 486–489.
- Conti, E.; Madhavan, V.; Such, F.P.; Lehman, J.; Stanley, K.O.; Clune, J. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. *arXiv* **2017**, arXiv:1712.06560.
- Fedus, W.; Ramachandran, P.; Agarwal, R.; Bengio, Y.; Larochelle, H.; Rowland, M.; Dabney, W. Revisiting fundamentals of experience replay. In Proceedings of the International Conference on Machine Learning, PMLR, Virtual, 13–18 July 2020; pp. 3061–3071.
- Li, X.; He, S.; Wu, L.; Chen, D.; Zhao, Y. A Game Model for Gomoku Based on Deep Learning and Monte Carlo Tree Search. In Proceedings of the 2019 Chinese Intelligent Automation Conference, Zhenjiang, China, 20–22 September 2019; Springer: Singapore, 2019; pp. 88–97.
- David, O.E.; Netanyahu, N.S.; Wolf, L. Deepchess: End-to-end deep neural network for automatic learning in chess. In Proceedings of the International Conference on Artificial Neural Networks, Barcelona, Spain, 6–9 September 2016; Springer: Cham, Switzerland, 2016; pp. 88–96.
- Chen, Y.H.; Tang, C.Y. On the bottleneck tree alignment problems. *Inf. Sci.* **2010**, *180*, 2134–2141. [\[CrossRef\]](#)
- Wang, J.; Huang, L. Evolving Gomoku solver by genetic algorithm. In Proceedings of the 2014 IEEE Workshop on Advanced Research and Technology in Industry Applications (WARTIA), Ottawa, ON, Canada, 29–30 September 2014; pp. 1064–1067.
- Zheng, P.M.; He, L. Design of gomoku ai based on machine game. *Comput. Knowl. Technol.* **2016**, *2016*, 33.
- Colledanchise, M.; Ögren, P. Behavior trees in robotics and AI: An introduction. *arXiv* **2017**, arXiv:1709.00084.
- Allis, L.V.; Herik, H.J.; Huntjens, M.P. *Go-Moku and Threat-Space Search*; University of Limburg, Department of Computer Science: Maastricht, The Netherlands, 1993.
- Yen, S.-J.; Yang, J.-K. Two-stage Monte Carlo tree search for Connect6. *IEEE Trans. Comput. Intell. AI Games* **2011**, *3*, 100–118.
- Herik, J.V.D.; Winands, M.H.M. Proof-Number Search and Its Variants. In *Flexible and Generalized Uncertainty Optimization*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 91–118.
- Coquelin, P.A.; Munos, R. Bandit algorithms for tree search. *arXiv* **2007**, arXiv:cs/0703062.
- Wang, F.-Y.; Zhang, H.; Liu, D. Adaptive Dynamic Programming: An Introduction. *IEEE Comput. Intell. Mag.* **2009**, *4*, 39–47. [\[CrossRef\]](#)
- Cao, X.; Lin, Y. UCT-ADP Progressive Bias Algorithm for Solving Gomoku. In Proceedings of the 2019 IEEE Symposium Series on Computational Intelligence (SSCI), Xiamen, China, 6–9 December 2019.
- O’Shea, K.; Nash, R. An introduction to convolutional neural networks. *arXiv* **2015**, arXiv:1511.08458.
- Xie, Z.; Fu, X.; Yu, J. AlphaGomoku: An AlphaGo-based Gomoku Artificial Intelligence using Curriculum Learning. *arXiv* **2018**, arXiv:1809.10595.
- Yan, P.; Feng, Y. A Hybrid Gomoku Deep Learning Artificial Intelligence. In Proceedings of the 2018 Artificial Intelligence and Cloud Computing Conference, Tokyo, Japan, 21–23 December 2018.
- Shao, K.; Zhao, D.; Tang, Z.; Zhu, Y. Move prediction in Gomoku using deep learning. In Proceedings of the 2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC), Wuhan, China, 11–13 November 2016.
- Oh, I.; Rho, S.; Moon, S.; Son, S.; Lee, H.; Chung, J. Creating Pro-Level AI for a Real-Time Fighting Game Using Deep Reinforcement Learning. *arXiv* **2019**, arXiv:1904.03821. [\[CrossRef\]](#)
- Tang, Z.; Zhao, D.; Shao, K.; Le, L.V. ADP with MCTS algorithm for Gomoku. In Proceedings of the 2016 IEEE Symposium Series on Computational Intelligence (SSCI), Athens, Greece, 6–9 December 2017.
- Zhao, D.; Zhang, Z.; Dai, Y. Self-teaching adaptive dynamic programming for Gomoku. *Neurocomputing* **2012**, *78*, 23–29. [\[CrossRef\]](#)
- Gu, B.; Sung, Y. Enhanced Reinforcement Learning Method Combining One-Hot Encoding-Based Vectors for CNN-Based Alternative High-Level Decisions. *Appl. Sci.* **2021**, *11*, 1291. [\[CrossRef\]](#)
- Geffner, H. Model-free, Model-based, and General Intelligence. *arXiv* **2018**, arXiv:1806.02308.
- Peterson, L.E. K-nearest neighbor. *Scholarpedia* **2009**, *4*, 1883. [\[CrossRef\]](#)
- Bradtke, S.J.; Barto, A.G. Linear least-squares algorithms for temporal difference learning. *Mach. Learn.* **1996**, *22*, 33–57. [\[CrossRef\]](#)

32. De Lope, J.; Maravall, D. The knn-td reinforcement learning algorithm. In Proceedings of the International Work-Conference on the Interplay between Natural and Artificial Computation, Santiago de Compostela, Spain, 22–26 June 2009; Springer: Berlin/Heidelberg, Germany, 2009.
33. Watkins, C.J.C.H.; Dayan, P. Q-learning. *Mach. Learn.* **1992**, *8*, 279–292. [[CrossRef](#)]
34. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 2018.
35. Sung, Y.; Ahn, E.; Cho, K. Q-learning reward propagation method for reducing the transmission power of sensor nodes in wireless sensor networks. *Wirel. Pers. Commun.* **2013**, *73*, 257–273. [[CrossRef](#)]
36. Von Pilchau, W.B.P.; Stein, A.; Hähner, J. Bootstrapping a dqn Replay Memory with synthetic experiences. *arXiv* **2002**, arXiv:2002.01370.
37. Schaul, T.; Quan, J.; Antonoglou, I.; Silver, D. Prioritized experience replay. *arXiv* **2015**, arXiv:1511.05952.
38. Wang, J.; Yang, Y.; Mao, J.; Huang, Z.; Huang, C.; Xu, W. Cnn-rnn: A unified framework for multi-label image classification. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 2285–2294.
39. Liu, H.; Yang, Y.; Wang, X. Overcoming Catastrophic Forgetting in Graph Neural Networks. *arXiv* **2020**, arXiv:2012.06002.