


Article

# Enhanced Reinforcement Learning Method Combining One-Hot Encoding-Based Vectors for CNN-Based Alternative High-Level Decisions

Bonwoo Gu<sup>1</sup> and Yunsick Sung<sup>2,\*</sup> <sup>1</sup> Department of M&S, SIMNET Cooperation, Daejeon 34127, Korea; griogrio@simnet.co.kr<sup>2</sup> Department of Multimedia Engineering, Dongguk University-Seoul, Seoul 04620, Korea

\* Correspondence: sung@dongguk.edu

**Abstract:** Gomoku is a two-player board game that originated in ancient China. There are various cases of developing Gomoku using artificial intelligence, such as a genetic algorithm and a tree search algorithm. Alpha-Gomoku, Gomoku AI built with Alpha-Go's algorithm, defines all possible situations in the Gomoku board using Monte-Carlo tree search (MCTS), and minimizes the probability of learning other correct answers in the duplicated Gomoku board situation. However, in the tree search algorithm, the accuracy drops, because the classification criteria are manually set. In this paper, we propose an improved reinforcement learning-based high-level decision approach using convolutional neural networks (CNN). The proposed algorithm expresses each state as One-Hot Encoding based vectors and determines the state of the Gomoku board by combining the similar state of One-Hot Encoding based vectors. Thus, in a case where a stone that is determined by CNN has already been placed or cannot be placed, we suggest a method for selecting an alternative. We verify the proposed method of Gomoku AI in GuPyEngine, a Python-based 3D simulation platform.



**Citation:** 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. <https://doi.org/10.3390/app11031291>

Academic Editor: Joonki Paik  
Received: 23 December 2020  
Accepted: 27 January 2021  
Published: 1 February 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/>).

**Keywords:** gomoku; game artificial intelligence; convolutional neural-networks; one-hot encoding; reinforcement learning

## 1. Introduction

Gomoku is a two-player board game that originated in ancient China. Two players alternate in placing a stone of their choice of color, and the player who first completes the five-in-a-row horizontally, vertically, or diagonally wins the game. Various studies have attempted to develop Gomoku using artificial intelligence (AI), for example, using a genetic algorithm and tree search algorithm [1,2]. Deep learning algorithms have been actively employed in the recent years after Alpha-Go showed its outstanding performance. For example, efficient Gomoku board recognition and decision-making was made possible while using a convolution layer of the deep-learning convolutional neural networks (CNN) algorithm [3]. However, as CNN-based decision-making determines a single optimal position in the same recognized Gomoku board state, some cases arise where a stone cannot be placed in the relevant position. In order to address this problem, Alpha-Gomoku (Gomoku AI built with Alpha-Go's algorithm) defines all possible situations in the Gomoku board while using Monte-Carlo tree search (MCTS) and minimizes the probability of learning other correct answers in a duplicated Gomoku board situation [4]. However, in the tree search algorithm, the accuracy drops because the classification criteria are manually set.

In this paper, we propose an improved reinforcement learning-based high-level decision algorithm while using CNN. The proposed algorithm expresses each state as one-hot-encoding-based vectors and determines the state of the Gomoku board by combining the similar state of one-hot-encoding-based vectors. Thus, in a case where a stone that is determined by CNN has already been placed or cannot be placed, we suggest a method

for selecting an alternative place. As similar states are combined, the amount of learning does not increase exponentially, which is advantageous. We verify the performance of the proposed reinforcement learning algorithm by applying it to Gomoku. We also verify the proposed method of Gomoku AI in GuPyEngine, a Python-based 3D simulation platform. This paper is expected to contribute to the field of incremental algorithms of reinforcement learning and deep learning-based 3D simulation by introducing the functions and performance of GuPyEngine.

The rest of this paper is organized, as follows. Section 2 describes related work on Gomoku AI, while Section 3 describes the proposed self-teaching Gomoku framework. Section 4 explains the GuPyEngine and the experiments that were conducted in this paper using the proposed method on Gomoku. Finally, Section 5 concludes this study summarizing the main findings and future research directions.

## 2. Related Works

Several researchers have attempted to develop Gomoku AI models using various algorithms. A genetic algorithm is an evolutionary-type algorithm that is used in various game AI models and it has been used in Gomoku AI models [2]. Shah, S. et al. [5] built a Gomoku AI model by applying the genetic algorithm. Wang et al. solved the bottleneck [6] that arises during a deep search in a tree-based Gomoku AI using the genetic algorithm [7].

In addition to the genetic algorithm, game AI models have been built using various tree search-based algorithms [1]. Typically, behavior trees (BT) allow for the efficient structuring of the behavior of artificial agents [8]. BT is widely used in computer games and robotics because the approach is efficient for building complex AI models that are modular as well as reactive. Allis et al. described a Gomoku AI model using the threat-space search [9] and proof-number search [10,11]. This algorithm was implemented in the Gomoku program, called Victoria. When Victoria first moved a stone, the algorithm yielded a winning result without fail. Cao et al. presented a Gomoku AI model using an algorithm that combined the upper confidence bounds that were applied to the trees (UCT) [12] and adaptive dynamic programming (ADP) [13,14]. This algorithm could solve the search depth defect more accurately and efficiently than the case when only a single UCT was used, thereby improving the performance of Gomoku AI.

Gomoku models using CNN have also been developed, as Deep-Mind's Alpha-Go demonstrated outstanding deep learning performance. Li et al. built a Gomoku model using MCTS and deep learning [15,16]. An expert-level AI model was developed without use of human expert knowledge by integrating the effective depth exploration of MCTS and deep neural networks (DNN). Yan et al. proposed a hybrid Gomoku AI model that improved the performance by including a hard-coded convolution-based Gomoku evaluation function [17]. Shao et al. predicted the next stone position with an accuracy of 42% while using only the current stone state of the board through the use of various DNNs with different hyper-parameters [18].

Various studies have developed expert-level game AI models using reinforcement learning models. For instance, Deep-Mind used the Deep Q-Networks (DQN) algorithm of reinforcement learning to apply it to seven Atari 2600 games, thereby enabling them to play at an expert-level [19]. Oh et al. considered a real-time fighting AI game while using deep reinforcement learning [20]. The AI model achieved a winning rate of 62% when competing with five professional gamers. Zhentao et al. presented an efficient Gomoku AI by integrating MCTS and ADP [21]. This algorithm could effectively eliminate the "short-sighted" defect of the neural network evaluation function, yielding results exceeding the Gomoku model adopting only ADP. Zhao et al. designed a Gomoku AI model that is capable of self-teaching through ADP [22]. Self-teaching is a method that penalizes the final action of the loser while rewarding the last action of the winner. The present study also uses "Record" for self-teaching, inspired by this basic idea.

With the help of reinforcement learning, a model-free algorithm can represent the next state of AI as a probability when AI acts in a specific state [23]. In Q-Learning, the

next Q state is calculated while using Q-values [24]. In this regard, de Lope et al. designed an efficient KNN-TD algorithm for a model-free algorithm by integrating the K-nearest neighbor (KNN) algorithm [25] and the temporal difference (TD) [26,27].

In this paper, by combining one-hot-encoding-based vectors, we build a model-free Gomoku AI that is capable of self-teaching. After generating the state vectors by performing convolution operations on the relationship between the stones that were placed on the Gomoku board, we determine similar situations for each situation vector inputted with an approximate nearest neighbor (ANN) [28]. In the case of a similar situation, the one-hot-encoding-based vectors are combined; subsequently, a CNN is utilized. The row values of the one-hot-encoding-based vectors are independent of each other, but they are converted into a related row that is not independent of each other by combining one-hot-encoding-based vectors. Following this approach, when applying self-teaching, if a stone cannot be placed in an optimal position, then the next optimal position can be considered. We also develop ‘GuPyEngine’ for Python and 3D programming education. For the proposed method, we conduct experiments by implementing a Gomoku system in GuPyEngine.

### 3. Advanced CNN-Based Decision-Making

This paper proposes an enhanced reinforcement learning model to improve the training data of a CNN, in order to enable optimal decision-making and alternative selection. By selecting an alternative, it solves the problem of a CNN-based decision-making method, allowing for only the optimal solution to be selected, and it minimizes the forgetting problem. The forgetting problem is that, given that the weights through previous learning in neural network algorithms are changed by new learning [29], there is the phenomenon that the weights are forgotten, which is generally invoked when the inputs are similar and labels are different.

#### 3.1. Framework Overview

Figure 1 shows the proposed framework. As can be seen, the framework is based on multiple encoding-based vector combinations, containing the following stages: “State Vector Generation”, “CNN-based Estimation”, “Decision Making”, “One-Hot-Encoding-based Vector Generation”, “ANN-based one-hot encoding vector Combination”, and “Enhanced CNN Training”.

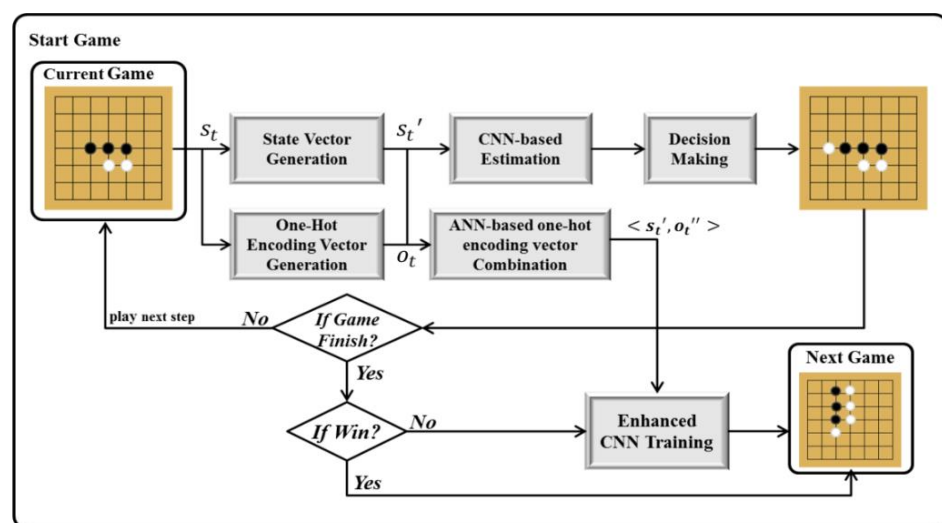


Figure 1. Proposed self-teaching Gomoku framework.

The state vector generation stage generates the vectors for simulating the states. When  $f^1(s_t)$  is the state vector generation function,  $f^1(s_t) = \langle s'_{t,1}, s'_{t,2}, \dots, s'_{t,n} \rangle$ , where  $s_{t,i}$  is the  $i$ -th element of  $s_t$  and  $s'_{t,i}$  is the converted value of  $s_{t,i}$ , which is defined while considering both the environment and CNN. In the CNN-based estimation stage, alternative actions

are derived by applying the state vector to a CNN. Moreover, in this stage, the proposed CNN uses convolution with a kernel size of  $3 \times 3$  [18]. As in the game of Gomoku, the position of stones must be accurately determined, the minimum kernel size should be used. In the decision-making stage, a case might arise where the action corresponding to the highest probability in the CNN result cannot be performed. This case is considered as  $-1$ , which makes the selection impossible, thereby selecting and performing an alternative action with the highest probability.

The one-hot-encoding-based vector generation stage generates the vectors for simulating the actions.  $f^2(s_t)$  returns  $o_t$  the one-hot-encoding-based vector of  $s_t$ . The ANN-based one-hot-encoding vector combination stage receives  $\langle s'_t, o_t \rangle$  and then returns  $\langle s'_t, o''_t \rangle$  as training data, where  $o'_t$  corresponds to the most similar vector in the training data. Moreover,  $o''_t$  combines  $o_t$  with  $o'_t$ . The following section describes how these two vectors are combined. The CNN training stage makes the CNN learn with  $\langle s'_t, o''_t \rangle$ .

### 3.2. ANN-Based One-Hot Encoding Vector Combination Stage

In this paper, we define “Record” as the action of going back several moves at the end of each game and placing a stone in the position where the opponent’s stone is likely to be placed, while using the proposed method. There are two considerations to update rewards as below. First, when the player wins, no reward is received. If the rewards of wins are reflected, a specific row in the combined one-hot-encoding-based vector is increased rapidly. If so, the diverse kinds of stone’s positions are not considered. As a result, Gomoku falls in one way, which is the local optimum problem [30]. Next, whenever the player loses, reward is only updated. For examples, in the case of one record, it goes back by one move from the end of the corresponding game, and reward is updated by 1 considering the situation after replacing the last position of the opponent’s stone with the player’s own stone, and then encoding-based vectors are combined by one vector. In the case of three records, it repeats one record three times to replace the black stone with the white stone by going back three moves.

Figure 2 shows the combination of one-hot-encoding-based vectors for the proposed Gomoku system. In Gomoku, the relationship between similar stone moves occurs frequently. Depending on the player’s tendency, the moves can completely differ from one another, even in a similar situation. The proposed Gomoku system divides similar situations for each feature vector while using the ANN algorithm. In the divided similar situations, one-hot encoding-based vectors are combined in order to calculate the probability. This solves the problem of an exponential increase in the number of training data.

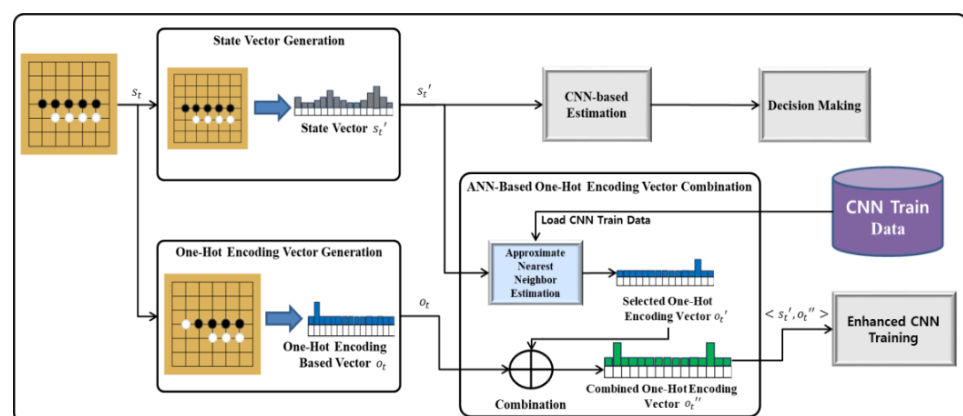


Figure 2. Approximate nearest neighbor (ANN)-based one-hot encoding vector combination.

ANN algorithms are diversely utilized for large-scale image searching and classification problems [28]. In this paper, the similarity among one-hot-encoding-based vectors is calculated by Euclidean distance, a representative ANN algorithm. If the similarity is smaller than a specific threshold, then the corresponding vectors are combined. A new

vector that contains the corresponding vector is inserted if the similarity is bigger than the threshold when that there is no similar vector.

One-hot encoding-based vector is a binary vector that sets one value of the corresponding index of labeled data by 1 and sets others by 0 [31]. Given that, when learning by a neural network with labeled data as numbers, the learning results can be deduced by decimal points. If so, categorical data cannot be accurately classified. However, if one-hot encoding-based vector is applied, categorical data can be classified accurately. Given that only one stone can be placed at a specific place in Gomoku, the possible positions of stones are treated with 49 positions. For example, one-hot-encoding-based vector expression is one of hot-issued approaches for classifying categorical data by Deep Neural Network (DNN) [31–34]. In the paper, the positions of stones considering states are expressed by one-hot-encoding-based vector.

By combining one-hot-encoding, 49 row relationships that are independent of each other are made non-independent. The CNN result of learning the non-independent one-hot encoding outputs the result that completely resembles the learned one-hot-encoding vectors. Thus, the CNN result with each non-independent row is outputted. Although there is a black or white stone in the position with the highest probability in the CNN result, the position with the next highest probability is non-independently related to the current situation. Therefore, it becomes the next best position for the current situation. In addition, as one-hot-encoding is continuously combined for each game in similar situations, the learning result varies according to the user's tendency.

### 3.3. CNN Training Stage

The record data generated for each game are combined for each similar situation through the ANN. Nodes are generated for each different situation in the ANN, and each node has one-hot-encoding-based vectors that are combined (Figure 3).

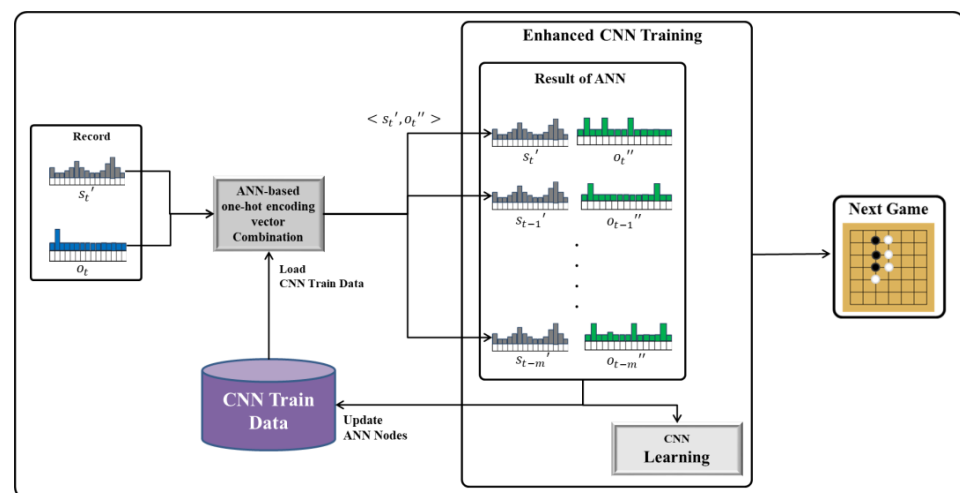


Figure 3. Enhanced convolutional neural networks (CNN) training.

The vector for each ANN node frequently varies every time that learning is performed with the record data. Hence, in the 'Enhanced CNN Training' stage of each game, the existing CNN model is removed, and a new CNN model is regenerated based on the learned ANN result. In addition, the training data of the ANN are regenerated with the latest ANN node. As similar situations are combined through the ANN, the amount of training data does not increase incrementally, even though CNN training is performed for each game. This also mitigates the forgetting problem by reducing the amount of useless data learning during CNN training.



## 4. Experiments

We verify the performance of the proposed reinforcement learning model by applying it to a Gomoku game. Section 4.1 describes the learning and development environments, whereas Section 4.2 introduces GupyEngine. Sections 4.3–4.5 present the performance verification of the proposed reinforcement learning Gomoku AI model while considering various criteria.

### 4.1. Experimental Environment

We verified the performance through various methods in order to confirm the performance of the Gomoku system proposed in this paper. We performed experiments with “the number of wins”, “the number of ANN nodes”, and “the next best solution” of the Gomoku game. We verified the difference in performance according to the number of records generated for each game. In this paper, the experiment was developed with C/C++, Python 3.0, and TensorFlow. CPU I-7, RAM 16GB, and GPU GTX-1050 were used in the experimental environment.

In the experiments, 600 Gomoku games were iterated for each experimental case. The genetic algorithm was applied to black stones. The same moves for each game were prevented by randomly initializing X and Y in positions between 3 and 5. As the performance of ANN varies depending on the defined distance, we verified the performance according to various distances in the experiments.

The proposed method was applied to white stones. The goal of the white stone was to prevent the black stone from completing five-in-a-row, by predicting the position of the black stone. The proposed Gomoku system considered the white stone as the winner if it completes the five-in-a-row first, or if all 49 intersections are filled. If a white stone wins, it moves to the next game, whereas, if a black stone wins, it relearns and proceeds to the next game.

In the CNN-based estimation stage, a total of three hidden layers were considered, and each hidden layer had 49, 128, and 516 neurons. The output layer and input layer were designed to have 49 ( $7 \times 7$ ) neurons, which corresponds to the total number of intersections on the board. The CNN result of the proposed Gomoku system was outputted in the form of a vector with 49 rows, in order to predict the position of the next white stone by matching 1:1 to the 49 intersections of the board.

### 4.2. GuPyEngine

GuPyEngine program is a Python-based simulation platform. Figure 4 depicts screenshots of the GuPyEngine, which aims to easily and simply implement a three-dimensional (3D) simulation environment and simulate Python-based math and physics, among others. OpenGL 4.0 was used for 3D rendering, and the GUI supports high-speed rendering using GDI+ and OpenCV. The state changes in 3D objects, such as position, rotation, and scale, were supported, as well as camera, FBX file Loader, and bone key animation.

The Python code can be linked for each 3D object and written by linking with the Pycharm Tool, as when using the Unity 3D Tool (Figure 5). Currently, in GuPyEngine, we can change the object state, perform creation and deletion, and modify the object texture using the Python code. In addition, it is possible to install and link the code with deep learning tools, such as Tensor Floor and Keras, through PIP, due to the development of the 32 bit and 64 bit versions. The GupyEngine supports PIP versions of up to 3.0, and errors in Python code are displayed in the console (cmd.exe) window to facilitate debugging.

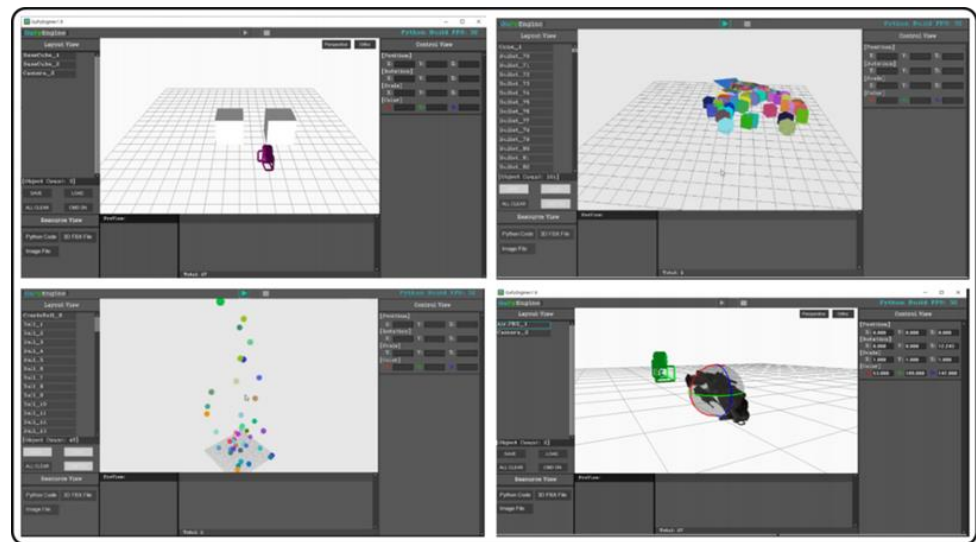


Figure 4. Screenshots of GuPyEngine.

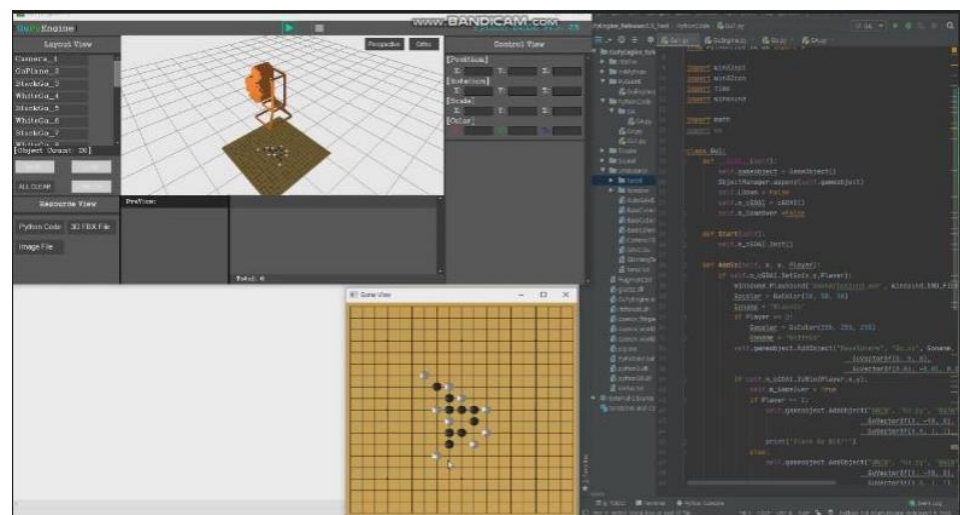


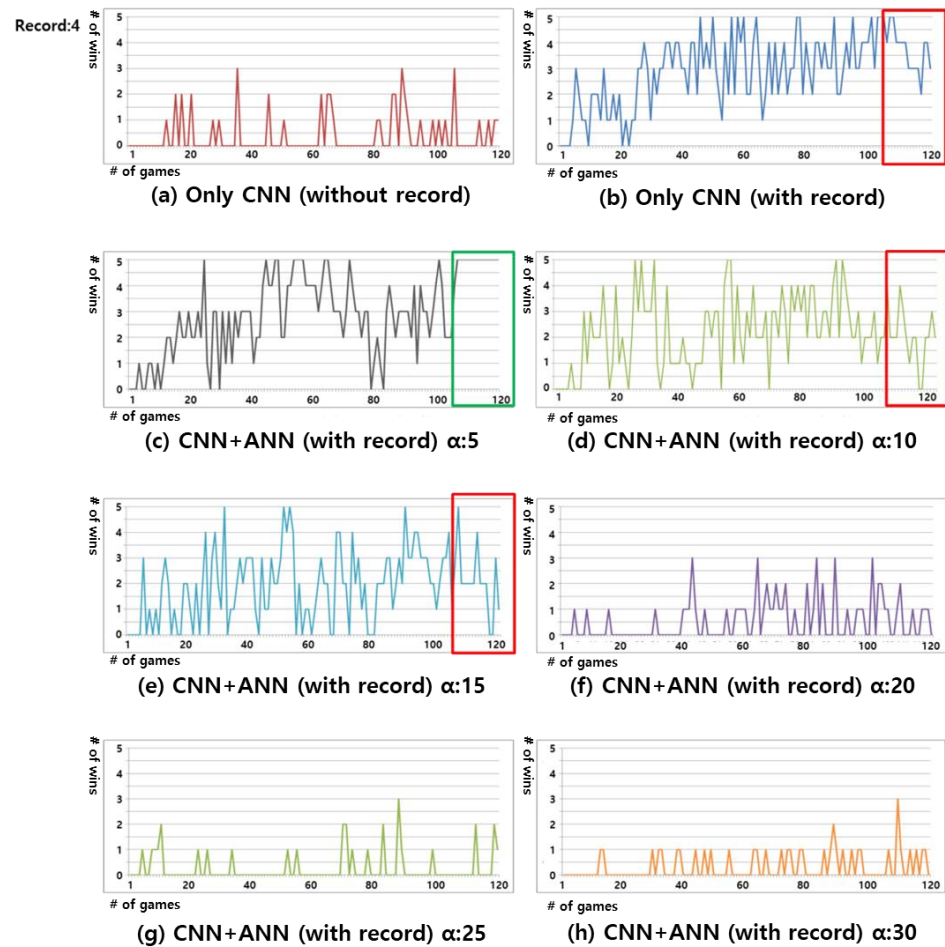
Figure 5. Proposed Gomoku system’s Python code in GupyEngine with Pycharm.

GuPyEngine rebuilds and uses Python code in C/C++. Hence, Python can be used with high processing speed, and external libraries are easily incorporated. A function for converting Python code into DLL/So files will be developed in the future for GuPyEngine; this will ensure that the content that is written in Python code in GuPyEngine can be easily linked with Unity 3D Engine, Unreal Engine, and other external programs. The Gomoku system that is proposed in this paper was developed by installing and using TensorFlow in GuPyEngine.

#### 4.3. Number of Winning Games

We calculated the winning rate by grouping 600 experimental data into five categories. Figure 6 shows the experimental results regarding the winning rate for four Records. Figure 6a shows that the winning rate is low, owing to the frequent forgetting problems, as only the CNN was used without relearning from the records. Figure 6b confirms that the winning rate gradually increases when the records were relearned. However, new moves of the GA AI continue to be created, which results in frequent variations in the winning rate. Figure 6c shows that the winning rate gradually increases, and all the wins were observed from the 525th data, as seen in the green box area that is depicted in Figure 6c. The figure also shows that the result of learning all the possible moves that the GA AI can

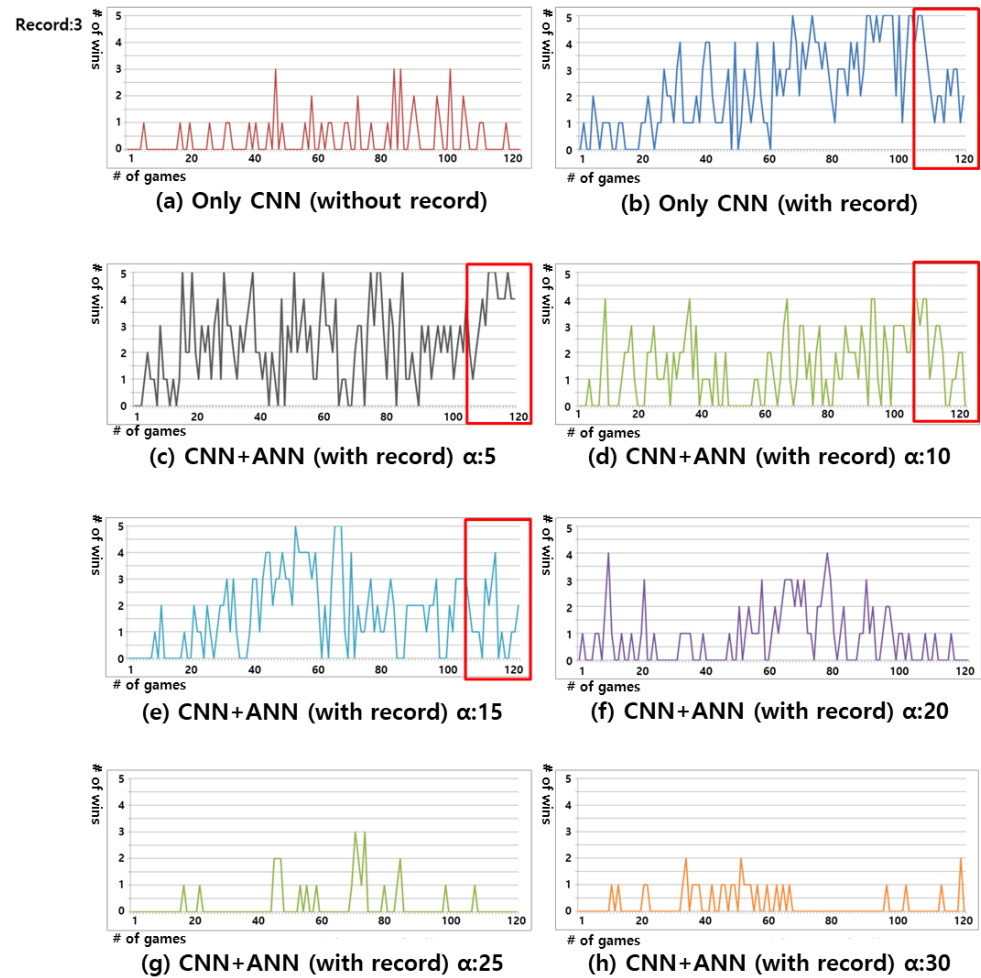
perform to place the stone, as the ANN minimizes the forgetting problem. Gomoku is a game that requires the exact stone position to be determined. Therefore, the ideal result is derived when the distance  $\alpha$  of the ANN is five. Figure 6d–h show the results of the gradually increasing the  $\alpha$  of the ANN. The figures show that, as the  $\alpha$  of the ANN is higher than required, more errors occur, which hinders the learning. Figure 6d,e display similar shapes. ANN learning deteriorates the forgetting problem, which results in a decrease in the winning rate, as seen in the red box areas in Figure 6d,e. Figure 6h shows a lower winning rate than Figure 6a.



**Figure 6.** Result of win rate when record count was four. Each graph  $y$ -axis refers to the number of wins from 0 to 5, and the  $x$ -axis refers to the number of games from 0 to 120. The figures show the experimental result using: (a) Only CNN without Record relearning; (b) CNN with Record relearning without ANN; (c) Proposed Gomoku system with ANN  $\alpha:5$ ; (d) Proposed Gomoku system with ANN  $\alpha:10$ ; (e) Proposed Gomoku system with ANN  $\alpha:15$ ; (f) Proposed Gomoku system with ANN  $\alpha:20$ ; (g) Proposed Gomoku system with ANN  $\alpha:25$ ; and, (h) Proposed Gomoku system with ANN  $\alpha:30$ .

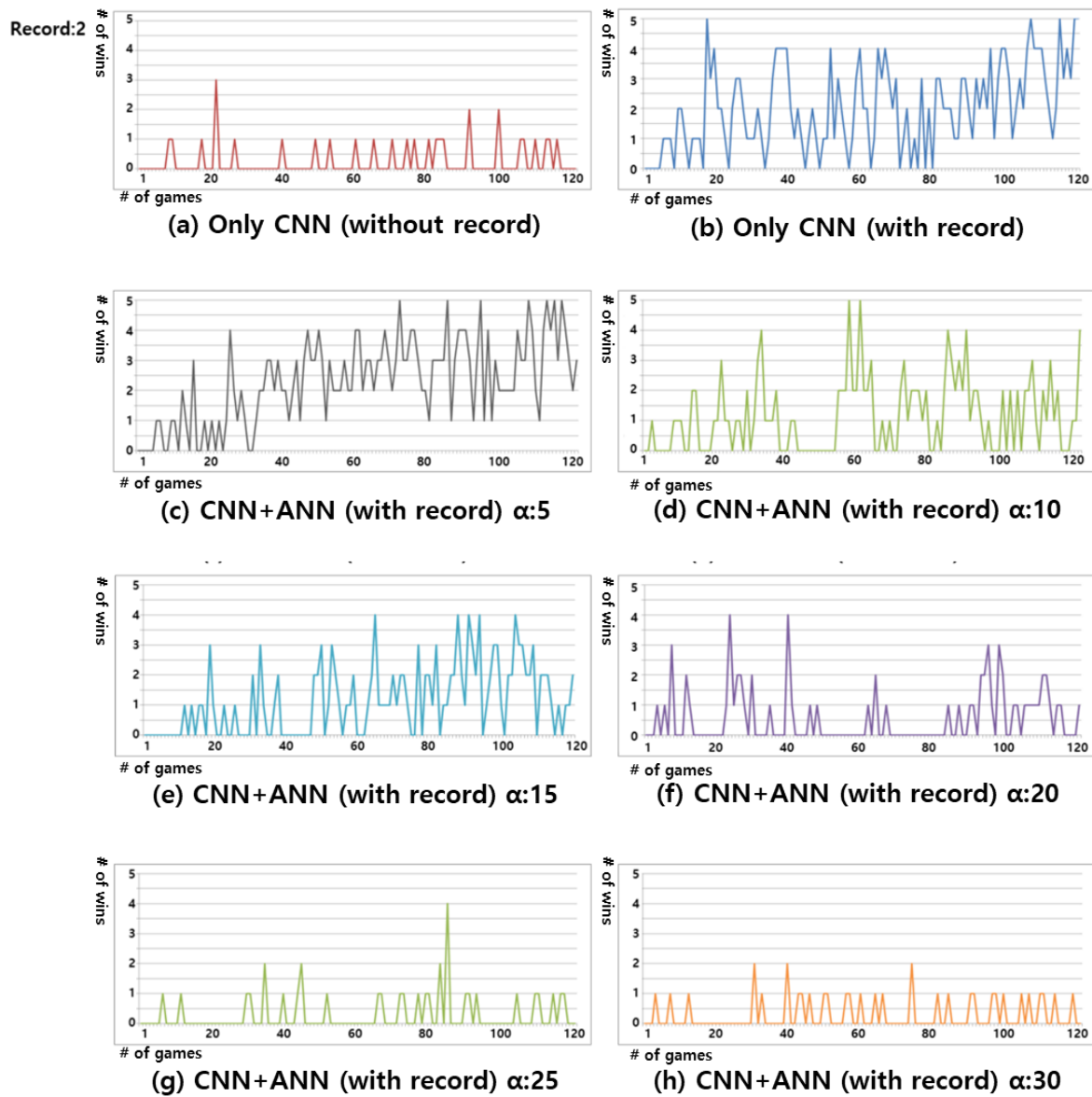
Figure 7 shows the experimental results regarding the winning rate for three records. The results of Figure 7 show that, overall, the winning rate is lower than those seen in Figure 6. The proposed Gomoku system performs Enhanced CNN Training for each game. Therefore, the results of this experiment show that the winning rate increases nearing the end rather than the beginning. However, in Figure 7b–e, the red box area shows fewer learning records than in the previous experiment. In particular, Figure 7c confirms that the winning rate in the all-winning section, as shown in Figure 6c, also drops, owing to the lack of records. However, as in Figures 6 and 7c shows a very high winning rate when compared to other experimental cases presented in Figure 7, showing that the ANN  $\alpha$  of five corresponds to the optimized case in the proposed Gomoku system.



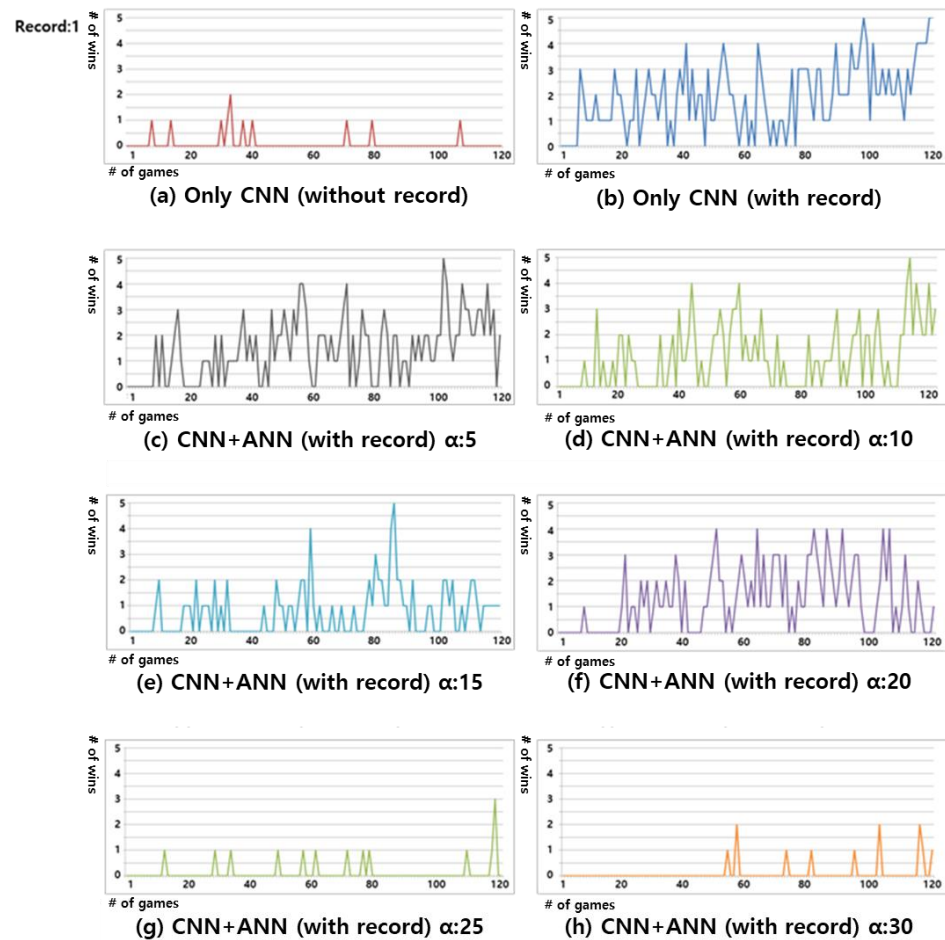


**Figure 7.** Result of Win rate when Record count was three. Each graph  $y$ -axis refers to the number of wins from 0 to 5, and the  $x$ -axis refers to the number of games from 0 to 120. The figures show the experimental result using: (a) Only CNN without Record relearning; (b) CNN with Record relearning without ANN; (c) Proposed Gomoku system with ANN  $\alpha:5$ ; (d) Proposed Gomoku system with ANN  $\alpha:10$ ; (e) Proposed Gomoku system with ANN  $\alpha:15$ ; (f) Proposed Gomoku system with ANN  $\alpha:20$ . (g) Proposed Gomoku system with ANN  $\alpha:25$ ; and, (h) Proposed Gomoku system with ANN  $\alpha:30$ .

Figures 8 and 9 show the experimental results regarding the winning rates for the two records and the single record, respectively. These results confirm that the number of records to be learned is insufficient and, therefore, the winning rate drops in all of the sections and the “Self-Teaching” occurs frequently. The experiments confirmed the satisfactory functioning of the ANN in the proposed Gomoku system when characterizing similar situations. We could also confirm that the optimal result was derived when the  $\alpha$  of the ANN was five. Furthermore, when the number of  $\alpha$  was greater than required, for example, in the range 25–30, it was observed that the CNN learning was rather hindered.



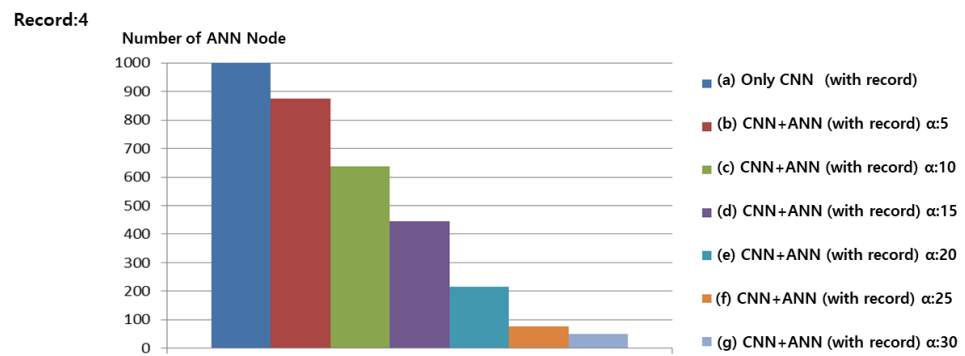
**Figure 8.** Results of win rate when the record count was two. Each graph  $y$ -axis refers to the number of wins from 0 to 5, and the  $x$ -axis refers to the number of games from 0 to 120. The figures show the experimental result using: (a) only CNN without Record relearning; (b) CNN with Record relearning without ANN; (c) Proposed Gomoku system with ANN  $\alpha:5$ ; (d) Proposed Gomoku system with ANN  $\alpha:10$ ; (e) Proposed Gomoku system with ANN  $\alpha:15$ ; (f) Proposed Gomoku system with ANN  $\alpha:20$ ; (g) Proposed Gomoku system with ANN  $\alpha:25$ ; and, (h) Proposed Gomoku system with ANN  $\alpha:30$ .



**Figure 9.** Result of Win rate when Record count was one. Each graph  $y$ -axis refers to the number of wins from 0 to 5, and the  $x$ -axis refers to the number of games from 0 to 120. The figures show the experimental result using: (a) only CNN without Record relearning; (b) CNN with Record relearning without ANN; (c) Proposed Gomoku system with ANN  $\alpha:5$ ; (d) Proposed Gomoku system with ANN  $\alpha:10$ ; (e) Proposed Gomoku system with ANN  $\alpha:15$ ; (f) Proposed Gomoku system with ANN  $\alpha:20$ ; (g) Proposed Gomoku system with ANN  $\alpha:25$ ; and, (h) Proposed Gomoku system with ANN  $\alpha:30$ .

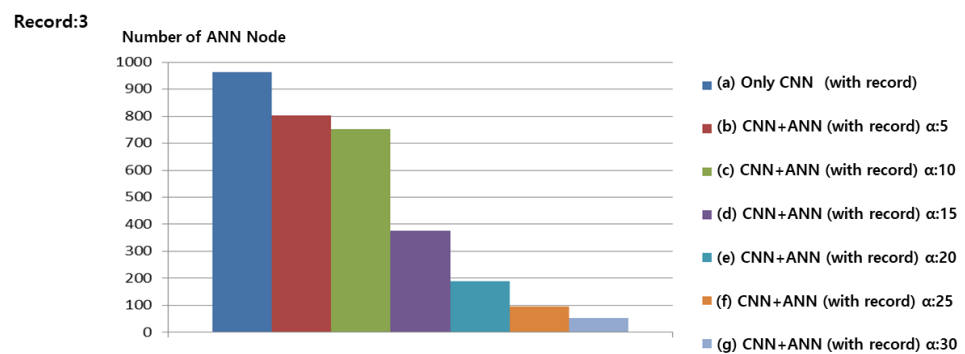
#### 4.4. Number of ANN Nodes

The number of ANN nodes refers to the number of cases grouped by similar situations in Gomoku, which indicates the amount of CNN training data. Figure 10a shows the results of saving four Records data for each game without ANN. During 600 games, 1000 records data were saved as record data, except for the winning game. Figure 10b shows the ANN and, because record with  $\alpha$  less than five are combined into a similar situation, the number of record data is reduced when compared to Figure 10a. Moreover, 874 record data were saved. Figure 10c–g show the results of saving 633, 448, 217, 78, and 50 record data, respectively, from 600 games.



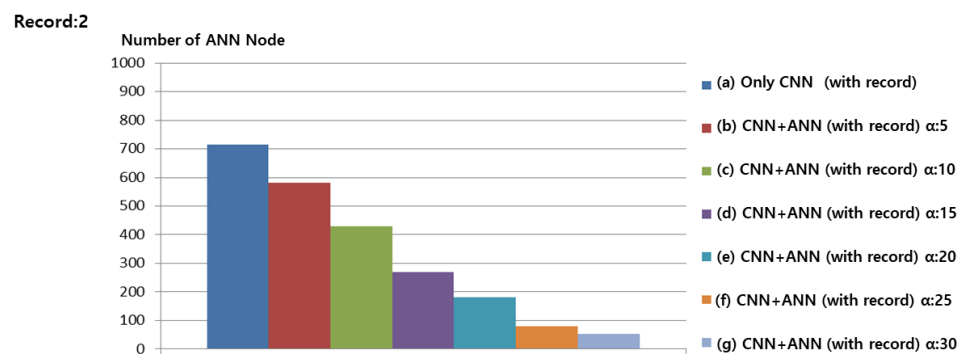
**Figure 10.** Number of ANN Nodes when Record was four.

With an increase in the  $\alpha$  of the ANN, the saved record data become smaller. Figure 10b shows less training data when compared to Figure 10a. Figure 6b,c show that the winning rate is also higher than that when using only the existing CNN. Thus, it is considered to be ideal learning result. However, with an increase in  $\alpha$ , records that are not similar are also merged, as seen in Figure 10g,h. Thus, CNN learning is hindered, and the winning rate is dropped, as in Figure 6g,h. Figure 11 shows similar results to those presented in Figure 10.



**Figure 11.** Number of ANN Nodes when Record was three.

Figures 12 and 13 confirm that, as the number of Record data to be learned is minimal, overall, the number of ANN nodes is less than required. These experiments also confirm that the amount of CNN training data does not increase linearly when the ANN is combined according to a similar situation. When the  $\alpha$  of the ANN is 5, the amount of training data was reduced by 12% on average, while still ensuring an effective performance.



**Figure 12.** Number of ANN Nodes when Record was two.

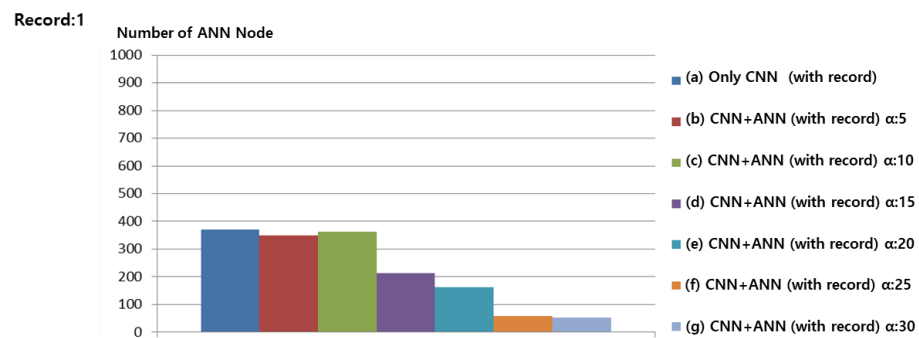


Figure 13. Number of ANN Nodes when Record was one.

#### 4.5. Next Best Answer Selection

The combination of one-hot encoding showed that the rows have a dependent relationship. Thus, we also experimented in order to verify the number of times the next best answer was selected in the proposed Gomoku system. The experimental criteria considered four records and the ANN  $\alpha$  of five, thereby yielding the most optimal results.

Figure 14 shows the number of times that the next best answer was selected in the proposed Gomoku system. As can be seen, the highest rank was selected 3483 times. When a new case was added through the ANN, the proposed CNN Gomoku system performed the greatest number of selections, because there was no other one-hot encoding to merge. If a stone was already placed in the highest rank or could not be placed, then the second-best result was selected. The number of the second-best was 1693, third-best was 751, fourth-best was 472, and fifth-best was 322. The number of times that the other rankings were selected was 1377. Through this experiment we confirmed that the proposed Gomoku system provides the next best answer by combining the independent one-hot encoding.

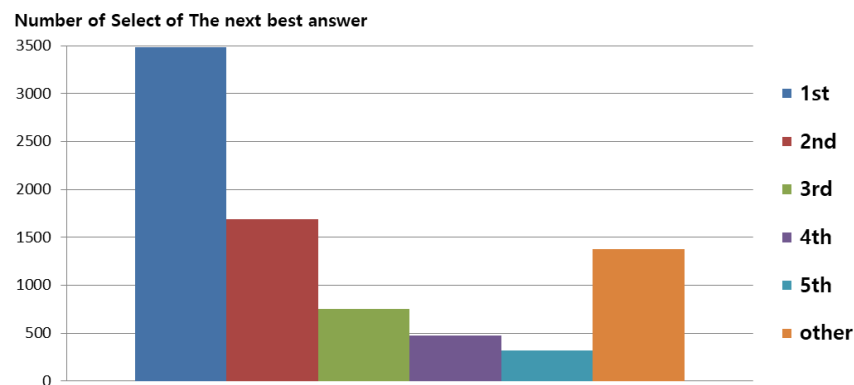


Figure 14. Result of selecting the next best answer.

### 5. Conclusions

We proposed a new reinforcement learning algorithm based on a CNN. We verified the performance of the proposed algorithm by applying it to Gomoku. We enabled the CNN to determine the correct answers according to the situation by itself by determining similar situations with an ANN and combining the rows of one-hot-encoding-based vectors into the determined similar situations. As the rows of the independent one-hot-encoding-based vectors become non-independent relationships through the proposed algorithm, the next best answer can be selected if there exists a case where it is impossible to place a stone for the correct answer with the highest probability. This minimizes the forgetting problem, which arises when indiscriminately learning the data. Thus, it was possible to obtain a result of all wins. Further, the average amount of training data was reduced by 12% on average when the Dis of the ANN was five, as the amount of data did not increase incrementally for each game.



We developed the proposed reinforcement learning based Gomoku AI and verified its performance in GuPyEngine. We also introduced the functions and performance of the GuPyEngine. In the future, we will perform various experiments and optimizations on the linkage with deep learning tools for GuPyEngine. In this way, we expect to contribute to 3D simulation research by improving the function of the 3D simulation platform. Further, we will apply the proposed reinforcement learning algorithm to other games and conduct relevant experiments.

**Author Contributions:** Conceptualization, B.G. and Y.S.; writing—review and editing, B.G. & Y.S.; validation, B.G.; supervision, Y.S. All authors have read and agreed to the published version of the manuscript.

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

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Data are not availability for public.

**Conflicts of Interest:** The authors claim no conflict of interest.

## References

1. Zheng, P.M.; He, L. Design of gomoku ai based on machine game. *Comput. Knowl. Technol.* **2016**, *2016*, 33.
2. Marks, R.E. Playing games with genetic algorithms. In *Evolutionary Computation in Economics and Finance*; Physica: Heidelberg, Germany, 2002; pp. 31–44.
3. O’Shea, K.; Nash, R. An introduction to convolutional neural networks. *arXiv* **2015**, arXiv:1511.08458.
4. Browne, C.B.; Powley, E.; Whitehouse, D.; Lucas, S.M.; Cowling, P.I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; Colton, S. A survey of monte carlo tree search methods. *IEEE Trans. Comput. Intell. AI Games* **2012**, *4*, 1–43. [[CrossRef](#)]
5. 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.
6. Chen, Y.H.; Tang, C.Y. On the bottleneck tree alignment problems. *Inf. Sci.* **2010**, *180*, 2134–2141. [[CrossRef](#)]
7. 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.
8. Colledanchise, M.; Ögren, P. Behavior trees in robotics and AI: An introduction. *arXiv* **2017**, arXiv:1709.00084.
9. Yen, S.-J.; Yang, J.-K. Two-stage Monte Carlo tree search for Connect6. *IEEE Trans. Comput. Intell. AI Games* **2011**, *3*, 100–118.
10. Herik, J.V.D.; Winands, M.H.M. Proof-Number Search and Its Variants. In *Flexible and Generalized Uncertainty Optimization*; Springer: Berlin, Heidelberg, 2008; pp. 91–118.
11. 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.
12. Coquelin, P.A.; Munos, R. Bandit algorithms for tree search. *arXiv* **2007**, arXiv:cs/0703062.
13. Wang, F.-Y.; Zhang, H.; Liu, D. Adaptive Dynamic Programming: An Introduction. *IEEE Comput. Intell. Mag.* **2009**, *4*, 39–47. [[CrossRef](#)]
14. 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.
15. 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.
16. Xie, Z.; Fu, X.; Yu, J. AlphaGomoku: An AlphaGo-based Gomoku Artificial Intelligence using Curriculum Learning. *arXiv* **2018**, arXiv:1809.10595.
17. 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.
18. 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), IEEE, Wuhan, China, 11–13 November 2016.
19. 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.
20. 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.
21. 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.

22. Zhao, D.; Zhang, Z.; Dai, Y. Self-teaching adaptive dynamic programming for Gomoku. *Neurocomputing* **2012**, *78*, 23–29. [[CrossRef](#)]
23. Geffner, H. Model-free, Model-based, and General Intelligence. *arXiv* **2018**, arXiv:1806.02308.
24. Watkins, C.J.C.H.; Dayan, P. Q-learning. *Mach. Learn.* **1992**, *8*, 279–292. [[CrossRef](#)]
25. Peterson, L.E. K-nearest neighbor. *Scholarpedia* **2009**, *4*, 1883. [[CrossRef](#)]
26. Bradtke, S.J.; Barto, A.G. Linear least-squares algorithms for temporal difference learning. *Mach. Learn.* **1996**, *22*, 33–57. [[CrossRef](#)]
27. 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.
28. O'Hara, S.; Draper, B.A. Are you using the right approximate nearest neighbor algorithm? In Proceedings of the 2013 IEEE Workshop on Applications of Computer Vision (WACV), IEEE, Tampa, FL, USA, 15–17 January 2013.
29. Liu, H.; Yang, Y.; Wang, X. Overcoming Catastrophic Forgetting in Graph Neural Networks. *arXiv* **2020**, arXiv:2012.06002.
30. 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.
31. Hancock, J.T.; Khoshgoftaar, T.M. Survey on categorical data for neural networks. *J. Big Data* **2020**, *7*, 1–41. [[CrossRef](#)]
32. Potdar, K.; Pardawala, T.S.; Pai, C.D. A comparative study of categorical variable encoding techniques for neural network classifiers. *Int. J. Comput. Appl.* **2017**, *175*, 7–9. [[CrossRef](#)]
33. Brett, L. *Machine Learning with R*; Packt Publishing Limited: Birmingham, UK, 2013; ISBN 978-1782162148.
34. Cerda, P.; Varoquaux, G.; Kégl, B. Similarity encoding for learning with dirty categorical variables. *Mach. Learn.* **2018**, *107*, 1477–1494. [[CrossRef](#)]