applied sciences

MDPI

Training a Minesweeper Agent Using a Convolutional Neural Network

Wenbo Wang [1] and Chengyou Lei [2,*]

[1] College of Electronic Engineering (College of Artificial Intelligence), South China Agricultural University, Guangzhou 510642, China; wwb@scau.edu.cn
[2] Wuhan Second Ship Design and Research Institute, Wuhan 430205, China
* Correspondence: leichengyou@126.com

Abstract: The Minesweeper game is modeled as a sequential decision-making task, for which a neural network architecture, state encoding, and reward function were herein designed. Both a Deep Q-Network (DQN) and supervised learning methods were successfully applied to optimize the training of the game. The experiments were conducted on the AutoDL platform using an NVIDIA RTX 3090 GPU for efficient computation. The results showed that in a 6 × 6 grid with four mines, the DQN model achieved an average win rate of 93.3% (standard deviation: 0.77%), while the supervised learning method achieved 91.2% (standard deviation: 0.9%), both outperforming human players and baseline algorithms and demonstrating high intelligence. The mechanisms of the two methods in the Minesweeper task were analyzed, with the reasons for the faster training speed and more stable performance of supervised learning explained from the perspectives of means–ends analysis and feedback control. Although there is room for improvement in sample efficiency and training stability in the DQN model, its greater generalization ability makes it highly promising for application in more complex decision-making tasks.

Keywords: convolutional neural network (CNN); Minesweeper game; deep Q-network (DQN); supervised learning; sequential decision making; deep reinforcement learning; deep neural network; feedback control; artificial general intelligence (AGI)

Academic Editor: Paolo Renna

Received: 17 January 2025
Revised: 23 February 2025
Accepted: 24 February 2025
Published: 25 February 2025

Citation: Wang, W.; Lei, C. Training a Minesweeper Agent Using a Convolutional Neural Network. *Appl. Sci.* **2025**, *15*, 2490. https://doi.org/10.3390/app15052490

1. Introduction

Minesweeper was first released in 1992 on the Windows operating system, aiming to train users in balancing the use of the left and right mouse buttons, improving mouse movement speed and click accuracy, thereby making the operation of PCs more natural and fluid. This number-based logic game became highly popular, providing an engaging way to pass time, and eventually became a standard game in subsequent versions of the Windows operating system [1].

The game interface consists of a minefield, a mine counter, and a timer, as shown in Figure 1. The minefield is rectangular, with mines randomly placed within it. Players can customize the size of the minefield and the number of mines, ranging from a minimum grid size of 8 × 8 to a maximum of 30 × 24 (with 30 being the maximum length and 24 the maximum width). The number of mines can vary between 10 and 667, with the maximum being 667 in the configuration of 29 × 23.

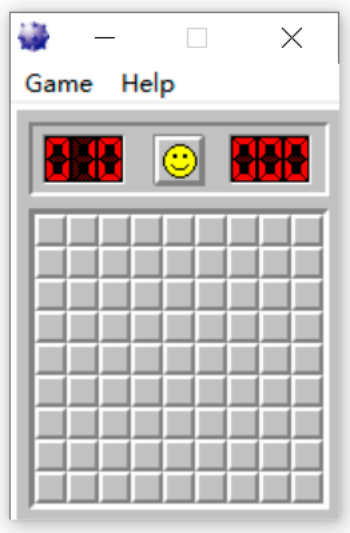

Figure 1. Minesweeper game interface on Windows XP system (beginner level, 9 × 9 grid, 10 mines).

The objective of Minesweeper is for the player to uncover all non-mine cells as quickly as possible to complete the game and achieve victory. If the player uncovers a cell containing a mine, it will trigger an explosion and result in a loss. Typically, Minesweeper is designed to ensure that the player's first move does not trigger a mine.

Minesweeper is a game that combines practicality, fun, and competitiveness, testing the player's reaction speed, logical reasoning, and adaptability. Additionally, Minesweeper possesses several characteristics that make it an ideal benchmark for testing intelligent algorithms [2]:

(1) Partially Observable Environment: In Minesweeper, players do not have access to the full minefield information and can only see the uncovered cells. This partial observability requires algorithms to make decisions with incomplete information, similar to the challenges faced in many real-world scenarios;
(2) Probabilistic Reasoning: The game involves uncertainty, with players inferring the positions of mines based on displayed numbers. Intelligent algorithms must handle this probabilistic reasoning to make optimal decisions, making Minesweeper an ideal tool for evaluating algorithms with probabilistic reasoning capabilities;
(3) Sequential Decision Making: Every move in Minesweeper is related to previous actions, requiring optimization of a series of decisions. This characteristic mirrors many real-world tasks, where planning and executing a sequence of actions is essential;
(4) Large State Space: Although the rules are simple, the state space of Minesweeper is vast due to the numerous configurations of mines and uncovered cells. Testing algorithms in such a complex state space helps assess their generalization ability and efficient exploration capabilities;
(5) Reward-based Feedback: The game has clear win/loss outcomes, along with intermediate feedback provided by the revealed numbers. This reward structure is highly suitable for reinforcement learning algorithms, which learn by maximizing rewards.

In summary, Minesweeper is a logical, uncertain, and complex game that requires moderate storage and computational resources, making it an ideal platform for testing intelligent algorithms.

The process followed by a Minesweeper agent is as follows: first, the Minesweeper grid is encoded to create an observable state. Then, based on the current state and the selected algorithmic model, the optimal action is determined. The current state, the chosen action, and the rules of Minesweeper collectively determine the next state, forming a sequential decision-making process.

Depending on the type of algorithmic model, intelligent algorithms for Minesweeper can be divided into two main categories:

(1) Deterministic Algorithms: These algorithms formalize human players' experience and techniques into an expert system, requiring no further training. Two representative algorithms are the Single-Point (SP) algorithm and the Constraint Satisfaction Problem (CSP) algorithm [3,4]. The SP algorithm is based on the fundamental Minesweeper technique: if a cell shows 8, then each adjacent cell must contain a mine, and similar rules apply to other numbers. The CSP algorithm treats Minesweeper as a more general constraint satisfaction problem, where the revealed numbers provide constraints on the distribution of mines. Essentially, CSP is similar to SP, as both are based on constraint-driven numerical logic, but while SP only considers individual 3 × 3 grids, CSP considers the correlations between adjacent 3 × 3 grids or even larger areas. As a result, CSP has higher complexity and a higher win rate. The win rates for SP are 75.28%, 42.07%, and 1.28% for beginner, intermediate, and expert levels, respectively. For CSP, the win rates are 91.25%, 75.94%, and 32.90% [5]. It should be noted that classifying these algorithms as "intelligent" is debatable, as true intelligence involves not only high win rates but also the ability to learn the rules and techniques of Minesweeper through training. Nevertheless, SP and CSP algorithms serve as benchmarks for evaluating other algorithms, as they effectively reflect the highest level of human player performance;

(2) Learning-based algorithms mimic the process by which human players acquire Minesweeper skills. The agent is trained through repeated gameplay, typically involving exploration and trial and error. The key is to design a selection mechanism that evaluates whether the agent's actions are moving toward the goal. If progress is made, the agent is encouraged to continue searching in the same direction; if no progress is made, the current path is abandoned to avoid blind searching. The Q-learning algorithm selects actions by evaluating the cumulative rewards of each game. As long as the reward function is well-designed, this algorithm is theoretically a general approach. Reference [6] shows that, under a 4 × 4 grid with three mines, the algorithm's win rate can reach 70% after 1 million training episodes. However, due to its reliance on table-based state-action value storage, the algorithm has significant limitations in tasks with large state spaces (i.e., the state explosion problem). Reference [6] points out that in a 6 × 6 grid experiment, the algorithm is impractical on the hardware platform used due to excessive computation time. By replacing the traditional Q-learning table with a deep neural network, the Deep Q-Network (DQN) can effectively address the computational challenges of large state spaces. In Reference [7], the DQN model achieved a win rate of 90.2% in a 6 × 6 grid with four mines, but its training steps exceeded 17 million, resulting in prolonged computation time. Although the DQN model shows great potential for complex decision-making tasks, there is still room for improvement in sample efficiency and training stability.

The combination of deep learning and reinforcement learning is a popular research direction, enabling the creation of highly complex models and achieving impressive results in various fields, such as large language models such as ChatGPT [8], autonomous driving [9,10], and intelligent robots [11]. Computer games, such as those on the OpenAI Gym platform, have played a key role in the development of artificial intelligence technologies dominated by deep neural networks [12,13], providing an ideal validation and testing environment that drives algorithmic breakthroughs. The rich data and complex environments of games promote the optimization of deep learning algorithms. For instance, Google DeepMind successfully trained agents surpassing human performance in 49 Atari games with varying levels of complexity, action spaces, and reward structures

using DQN [14], demonstrating the method's effectiveness and robustness in complex environments. These results preliminarily validate the feasibility of using deep neural networks to train a Minesweeper agent, offering valuable insights contributing to this study's methodology and design.

The approach of this research is as follows: (a) the Minesweeper game environment was developed using the OpenAI Gym open-source toolkit, and a simulation platform for deep neural network-based intelligent algorithms was built, supporting the option to disable the game screen display to accelerate training speed; (b) for the training method of Minesweeper agents based on the Deep Q-Network, the neural network architecture, grid state encoding, and reward function were optimized according to the characteristics of the game, and a general method for determining model hyper-parameters was proposed, significantly improving sample efficiency; (c) a supervised learning-based training method for Minesweeper agents was proposed, greatly improving training speed and stability; (d) the correctness and effectiveness of the two methods were validated in a 6 × 6 grid with four mines, and the mechanisms of the methods were explained.

The related code can be accessed via GitHub repository: https://github.com/wwb-chat/minesweeper-applsci-3458345.

2. Algorithmic Principles

The input to the Minesweeper game is the encoded state of the minefield, and the output is the probability of each cell being a mine (i.e., the mine distribution probability). Since both the input and output are matrices containing spatial information, this problem is well-suited for processing using convolutional neural networks (CNN).

In Minesweeper, the agent's goal is to maximize its win rate, which requires training it to infer the positions of non-mine cells by observing the minefield. There are two feasible approaches to achieve this: one is reinforcement learning, where a reward mechanism is used to promote learning—rewards are given for selecting non-mine cells, additional rewards are provided when the choice reveals more information, and a higher reward is granted upon winning the game. The other approach is supervised learning, where an artificial neural network is used to approximate the probability of each cell being a mine (or not). The network is then optimized through backpropagation by minimizing the difference between the predicted probabilities and the actual minefield labels, ultimately training a model capable of playing Minesweeper.

2.1. Training a Minesweeper Agent with DQN

By designing an appropriate reward mechanism, the agent's goal can be formulated as maximizing the cumulative reward in a game round. Define the function $Q^*(s,a)$ as the maximum expected cumulative reward the agent can achieve by taking action a in state s :

$$Q^*(s,a) = \max_{\pi} E\left[R_t \middle| s_t = s, a_t = a, \pi\right] \tag{1}$$

where π represents the policy for selecting action a in state s, R_t is the cumulative reward starting from time t, and E denotes the mathematical expectation.

The $Q^*(s,a)$ function reflects the quality of taking action a in state s, and is therefore referred to as the optimal Q-function. Since this method uses a deep neural network to map states to actions, the network is called the Q-network, which is the basis for the name "Deep Q Network". The Q-network is the Minesweeper agent that needs to be trained.

Based on the agent's goal and the definition of the optimal Q-function, the optimal policy $\pi^*(s)$ is

$$\pi^*(s) = \arg\max_a Q^*(s,a) \tag{2}$$

Therefore, the key to finding the optimal policy is accurately estimating the optimal Q-function. A commonly used method for this is estimation via the Bellman equation:

$$Q^*(s,a) = E\left[r + \gamma \max_{a'} Q_\theta(s',a') \middle| s,a \right] \tag{3}$$

where s' represents the next state, r is the current reward, and γ is the discount factor. The Bellman equation is a recursive estimation method, which states that the maximum cumulative reward for a given state is the sum of the immediate reward and the maximum cumulative reward for the next state, a process that aligns with intuition.

The DQN model uses the Q-network to approximate the optimal Q-function, i.e., $Q_\theta(s,a) \approx Q^*(s,a)$.

By having the Q-network repeatedly play Minesweeper and explore different states, the action-value function $Q_\theta(s,a)$ for all actions is computed through forward propagation for the current state s, and the action with the highest function value is selected. After executing this action, a reward and the next state s' are obtained. The Q-network's parameters are updated using back-propagation to minimize the loss function, with the learning objective being to reduce this loss. After sufficient training, the Q-network will be able to effectively perform the Minesweeper task.

The loss function is defined as the squared error (abbreviated as SE):

$$L_{SE}(\theta) = \left[r + \gamma \max_{a'} Q_\theta(s',a') - Q_\theta(s,a) \right]^2 \tag{4}$$

To enhance training stability, the practical DQN algorithm typically incorporates mechanisms such as experience replay with random sampling and periodic target network updates. Additionally, to balance exploration and exploitation, the widely used ε-greedy strategy is employed in reinforcement learning. The DQN algorithm used in this paper is shown in Algorithm 1.

Algorithm 1 Training a Minesweeper agent based on DQN

Input: Encoded minesweeper grid state
Output: Q-network parameters

Initialize replay buffer D
Randomly initialize Q-network and target Q-network, $\theta^- = \theta$
Observe the initial state s
Repeat
    Select an action a
        With probability ε, choose a random action
        Otherwise, select an action based on $\pi(s) = \arg\max_a Q(s,a;\theta)$
    Execute the action a
    Observe the reward r and new state s'
    Store the experience $\langle s,a,r,s' \rangle$ in D
    If D is full, remove the oldest experience and add the new data
    Randomly sample a batch of size batch_size from D
    Compute the target x_i for each sample in the batch
        If the state is terminal, $x_i = r_i$

14: Otherwise, $x_i = r_i + \gamma \max_{a'} Q'\left(s_i', a'; \theta^-\right)$

Compute the MSE loss between the Q-function and the target Q-function

15: $L(\theta) = \frac{1}{batch\_size} \sum_{i=1}^{batch\_size} \left[x_i - Q(s_i, a_i; \theta)\right]^2$

16: Train the Q-network using the loss function $L(\theta)$

17: Every T training steps, copy the Q-network to the target Q-network, $\theta^- = \theta$

18: $s = s'$

19: Until the termination criteria is met

2.2. Training a Minesweeper Agent with Supervised Learning

The goal of supervised learning is to train the model using the true labels of the Minesweeper grid to predict the probability of each cell containing a mine. Given a grid state s, the model $f(s, \theta)$ is trained to predict the probability p_i (for $i = 1, 2, \cdots, MN$) that each cell contains a mine. The model is optimized by minimizing the cross-entropy (abbreviated as CE) loss function, defined as follows:

$$L_{CE}(\theta) = -\sum_i \left[y_i \log(p_i) + (1 - y_i) \log(1 - p_i)\right] \quad (5)$$

Here, p_i is the predicted mine probability, and y_i is the true label (1 for mine, 0 for non-mine). During gameplay, the agent makes decisions based on the cell with the lowest mine probability, consistently following the "safest" strategy for each move. In this way, the agent continuously explores and accumulates more game data, gradually optimizing the model and improving the accuracy of mine predictions.

Each cell is either a mine or not, with a label of 1 for mines and 0 for non-mines (viewing this as the probability of a cell being a mine is a useful way to understand the problem). For each game of Minesweeper, the grid labels are actually known, or at least can be obtained. This can be explained in two ways: one case is when the Minesweeper environment is designed by us, and the internal data structure of the game is transparent, allowing the grid labels to be directly used as training data; another case is treating the Minesweeper process as an exploration of a black box. If the game is won, it means all non-mine cells have been revealed, and the remaining unrevealed cells are mines. If the game ends in failure due to stepping on a mine, all mines will be revealed (as standard in Minesweeper), and the non-mine cells will also be disclosed. Therefore, regardless of the outcome, the true labels of the grid can be obtained. By backtracking through a game, a set of training data can be gathered. Similarly, humans discover key mistakes through review and reduce the likelihood of repeating them by accumulating experience. Both machines and humans essentially use similar methods: backtracking and reviewing are simply different expressions of the same concept. Machines use a loss function to measure the severity of errors and train the model by minimizing the loss function, thus reducing the frequency of errors and lowering their severity. Overall, this approach is intuitive and effective.

The supervised learning algorithm used in this paper is shown in Algorithm 2.

Algorithm 2 Training a Minesweeper agent based on supervised learning

Input: Encoded minesweeper grid state

Output: Deep neural network parameters

1: Initialize the training dataset D

2: Randomly initialize the neural network θ

3: Repeat

4: Retrieve the current grid state s and the ground truth label y

5: If s is a terminal state, start a new game and return to step 3
6: Otherwise, store the training data sample $\langle s, y \rangle$ into D
7: Select the grid cell with the minimum mine probability according to $x_{\min} = \arg\min_x f(s, \theta)$
8: Update the grid state s
9: If the size of D reaches sample_size, optimize the neural network
10: For each training epoch epoch = 1 to nEpochs
11: Shuffle D, and divide D into batches of size batch_size, i.e., $D = \{B_1, B_2, \cdots, B_m\}$
12: For each batch in D
13: For each sample $\langle s_j, y_j \rangle$ in B_i
14: Predict the mine probability p_j for each grid cell using the model $f(s_j, \theta)$
15: Calculate the sample loss function $L(y_j, p_j) = -\sum_{k=1}^{MN} \left[y_{j,k} \log(p_{j,k}) + (1 - y_{j,k}) \log(1 - p_{j,k}) \right]$, Here, $y_{j,k}$ represents the mine label of the k-th cell, and $p_{j,k}$ is the predicted probability that the k-th cell contains a mine.
16: Compute the average loss for the batch $L(\theta) = \frac{1}{batch\_size} \sum_{j=1}^{batch\_size} L(y_j, p_j)$
17: Update the model parameters θ using gradient descent with $L(\theta)$
18: Until the termination criteria are met

3. Model Design

3.1. Encoding of Minefield States

Both humans and computers need to "observe" the minefield state and make decisions on the next move based on this state. However, humans and various computational algorithms encode the minefield state differently, with three main types of encoding representations.

3.1.1. Single-Channel Encoding

This is also known as image encoding because its data format is similar to a grayscale image. The encoding has a dimension of 1 × m × n, where m and n represent the number of rows and columns of the minefield, respectively. The encoding rule is that unknown cells are represented as −1, while known cells are represented by the corresponding numbers (0–8). This encoding method is illustrated in Figure 2.

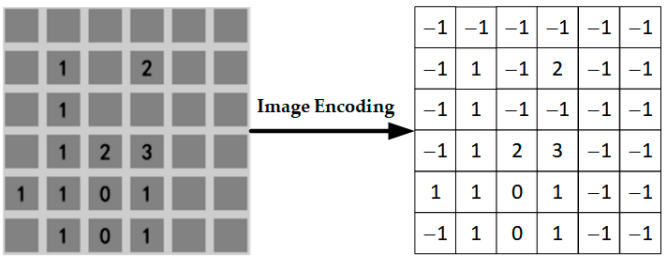

Figure 2. Illustration of single-channel encoding representation.

3.1.2. One-Hot Encoding

This encoding, also known as full encoding, has a dimension of 1 × m × n. Channels 1 to 9 correspond to the one-hot encoding of numbers 0 to 8. For example, if a cell contains the number 3, the value in channel 4 for that cell is 1, and 0 otherwise. Channel 10 corresponds to the one-hot encoding indicating whether a cell is unknown (unrevealed); if the cell is unknown, it is 1, otherwise, it is 0. This encoding method is similar to constructing complex images from simple layers. Full encoding is a further decomposition of image encoding, where each channel represents a specific cell state and only 0 s and 1 s are used to represent these states. The term "full" implies that this decomposition is in its simplest form. This encoding method is illustrated in Figure 3.

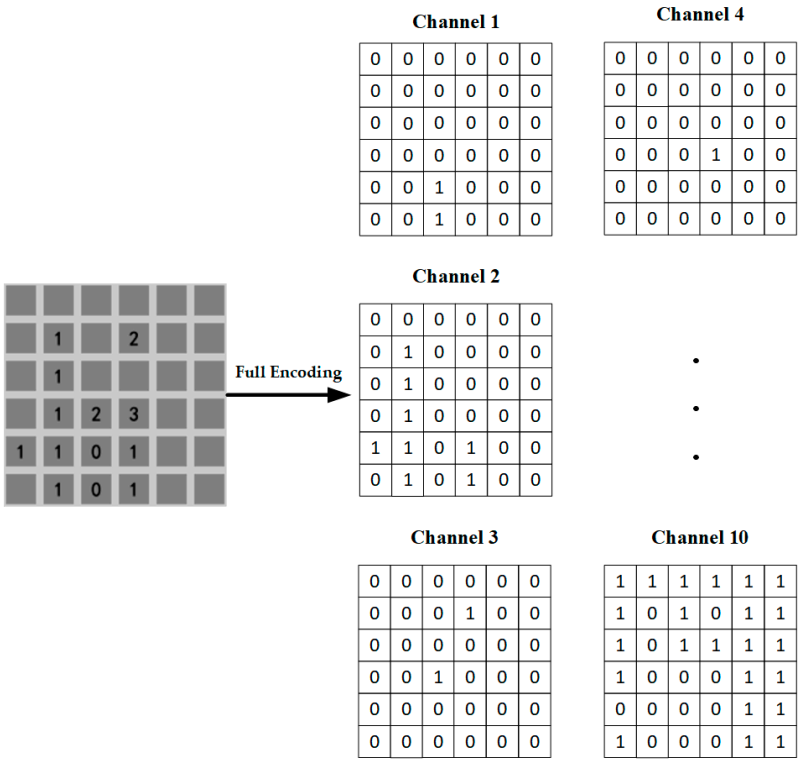

Figure 3. Illustration of full encoding representation.

3.1.3. Dual-Channel Encoding

This encoding, also known as condensed encoding, has dimensions of 2 mn and is a simplified version of the full encoding. The first channel is the combination of the first nine channels from the full encoding. If a cell is unknown or uncovered with a zero (0), it is encoded as 0; otherwise, it is encoded with the corresponding number (1–8). The second channel is identical to the tenth channel of the full encoding. This encoding method is illustrated in Figure 4.

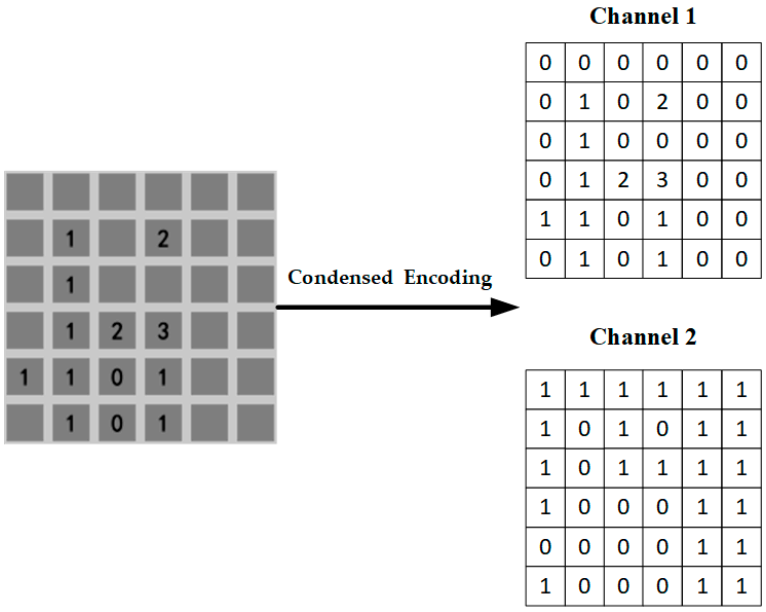

Figure 4. Illustration of dual-channel encoding representation.

Both rule-based reasoning algorithms, such as SP and CSP, and human-based approaches use image encoding for Minesweeper. However, for neural network-based algorithms, image encoding is not an ideal data format. For the specific problem studied in this paper, full encoding yields the best results, followed by condensed encoding, with minimal difference between the two. In general, full encoding is more advantageous for neural networks, as it provides a richer representation of input information, allowing the network to learn features more efficiently. This is akin to how a deeper understanding of concepts in humans often leads to less complex and more efficient reasoning and computation.

3.2. Neural Network Architecture

The example models used in Atari game benchmark tasks provide a good experimental starting point for the neural network architecture design in this paper [14,15]. Based on the characteristics of the Minesweeper game, the neural network architecture used in this study was determined with two main considerations: first, the input to the neural network consists of tensor data formed by encoding the minefield cells, rather than the pixel matrix of the minefield image. Therefore, pooling layers can be omitted to simplify the structure. Second, to enhance scalability, a relatively large neural network was adopted, which can be directly applied to training tasks for larger-scale Minesweeper agents without modification, though training time and instability may increase.

This paper uses a seven-layer neural network model, which consists of a series of convolutional layers followed by a series of fully connected layers, as shown in Figure 5.

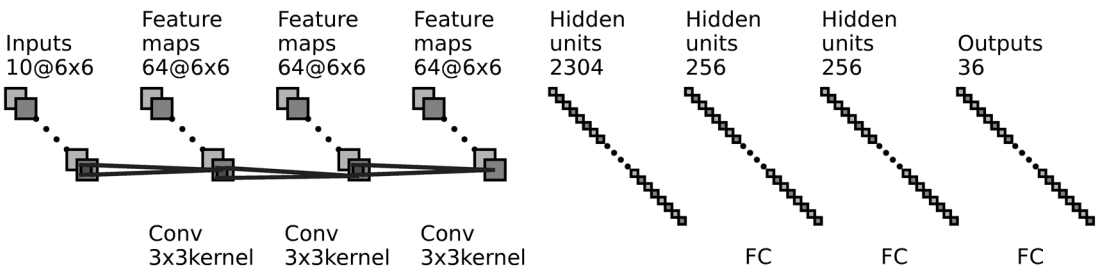

Figure 5. Convolutional neural network model of the Minesweeper game agent.

- Input Layer: A tensor of shape (10, 6, 6), which is the result of one-hot encoding the minefield state;
- Convolutional Layer 1: 10 input channels, 64 output channels, 3 × 3 kernel size;
- Convolutional Layer 2: 64 input channels, 64 output channels, 3 × 3 kernel size;
- Convolutional Layer 3: 64 input channels, 64 output channels, 3 × 3 kernel size;
- Fully Connected Layer 1: Input size 2304 (the output shape of the convolutional layers is (64, 6, 6), and the flattened size is 64 × 6 × 6 = 2304), output size 256, with ReLU activation;
- Fully Connected Layer 2: Input size 256, output size 256, with ReLU activation;
- Fully Connected Layer 3: Input size 256, output size 36, corresponding to the 36 possible actions.

3.3. Reward Function

The success of reinforcement learning methods, including DQN, heavily relies on a well-designed reward function. However, designing such a function is often challenging and requires careful analysis based on the specific context. In the case of Minesweeper, each action leads to a state change in the minefield. Even for a 6 × 6 grid with four mines, the number of possible states in the minefield can be extremely large. However, by analyzing the state changes before and after the action, the actions can be classified into five types, each assigned a reasonably intuitive reward value. Through testing and comparison, a satisfactory reward function can be obtained.

1. Action leading to a Win state: The goal of Minesweeper is to uncover all non-mine cells. Successfully achieving this means correctly determining the status of each cell (whether it is a mine or not). The reward for winning the game is set to the number of cells, $m \times n$, where m is the number of rows and n is the number of columns in the minefield;
2. Action leading to a Lose state: In contrast to winning, if an uncovered cell is a mine, the game is lost. The reward for losing the game is set to $-m \times n$. This design aligns with the characteristic of Minesweeper that "one wrong move leads to complete failure";
3. Progress state: Uncovering a non-mine cell brings the agent closer to winning the game and is rewarded with +1. This reward value is based on the idea that successfully uncovering a cell has a value of +1, consistent with the reward settings for the Win and Lose states;
4. Guess state: If all eight neighboring cells of an uncovered cell are still hidden, even if the uncovered cell is not a mine, no reward should be given. Instead, the agent should incur a penalty (negative reward). There are two reasons for this: first, such an action is essentially a random guess and does not contribute to improving the agent's Minesweeper skills; second, unless the mine density is very low, repeatedly taking such actions is likely to hit a mine, making it almost impossible to win the game. Based on preliminary testing, the reward for random guessing was set to −0.5;
5. YOLO state: YOLO is an acronym for "You Only Live Once", which encourages people to embrace risk and live boldly. This state refers to attempting to uncover an already uncovered cell, which does not change the minefield and results in no progress. "You can't beat something with nothing"—avoiding risk may prevent failure, but it also forfeits the opportunity for success. A penalty for such actions is necessary to prevent the agent from choosing a "no-action" strategy (lie flat or tang ping), where it repeatedly selects already uncovered cells to avoid the penalty for losing while giving up the reward for winning. A reasonable penalty for this "no-action" behavior helps the agent escape local optima and encourages it to continue striving for the global optimum.

The five types of actions and their corresponding rewards are shown in Figure 6. In this paper, the reward data structure is designed as a dictionary. For a 6 × 6 grid with four mines, the reward structure is defined as follows: rewards = {"Win": 36, "Loss": −36, "Progress": 1, "Guess": −0.5, "YOLO": −0.5}.

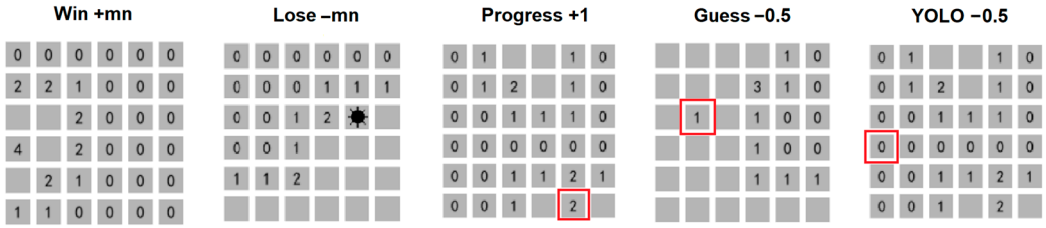

Figure 6. Schematic of the minesweeper reward function (The red box represents the currently clicked cell).

3.4. Model Hyper-Parameters

3.4.1. Experience Replay Buffer Parameters

The parameters related to the experience replay buffer are defined as follows: REPLAY_SIZE represents the size of the experience replay buffer, REPLAY_START_SIZE is the minimum amount of experience required to fill the buffer before sampling and training can begin, and BATCH_SIZE is the batch size used during training.

The primary purpose of setting REPLAY_START_SIZE is to ensure that a sufficiently diverse set of experience samples is accumulated before training starts, thus preventing the model from prematurely learning a suboptimal policy. Typically, REPLAY_START_SIZE should be at least 10 to 50 times the BATCH_SIZE. Additionally, to enhance the stability of DQN model training and ensure sample diversity, REPLAY_START_SIZE should be significantly smaller than REPLAY_SIZE, typically set to 1% to 10% of REPLAY_SIZE.

For complex environments, such as Atari games or large benchmark tasks, a large number of experience samples is required to capture environmental dynamics and the outcomes of different state-action pairs. A common configuration is to set REPLAY_START_SIZE to 50,000 and REPLAY_SIZE to 1,000,000. In contrast, although Minesweeper has a certain level of complexity in its state space (e.g., cell diversity, mine distribution), its action space and state transitions are relatively simple and more dependent on local strategies. Therefore, a smaller REPLAY_START_SIZE is sufficient for training to begin, and it can achieve satisfactory results without accumulating a large number of samples.

Setting model hyper-parameters is often a matter of trade-offs. As the engineering saying goes, "Choose two out of quality, speed, and cost", where quality refers to the performance of the model, speed refers to training efficiency, and cost refers to computational resources. Under a given computational capacity, selecting REPLAY_START_SIZE requires balancing training efficiency with the performance of the model. A smaller REPLAY_START_SIZE accelerates training but may affect model performance due to insufficient samples; a larger REPLAY_START_SIZE improves performance but may delay training progress.

The strategy adopted in this paper is to start with a smaller REPLAY_START_SIZE, following the aforementioned principles to set the corresponding REPLAY_SIZE and BATCH_SIZE. Under acceptable training speeds, REPLAY_START_SIZE is gradually increased until model performance no longer improves significantly or reaches a satisfactory level. If the target is not achieved, it indicates that the current computational capacity cannot balance efficiency and performance. In this case, computational power should be

increased, and the process repeated until an optimal parameter configuration is found. The experience replay buffer parameters used in this paper are detailed in Table 1.

Table 1. Parameters of the experience replay buffer.

Parameter Name	Parameter Value	Empirical Value
REPLAY_START_SIZE	10,000	Complex Environment: 50,000 Simple Environment: 1000
REPLAY_SIZE	300,000	10–100 times REPLAY_START_SIZE
BATCH_SIZE	128	<0.02–0.1 times REPLAY_START_SIZE

3.4.2. Discount Factor Parameter

In the Minesweeper game, a discount factor (GAMMA) of 0 represents a short-term survival strategy, where "survival" is treated as "victory." A larger discount factor, on the other hand, represents a long-term strategy, considering the relevance of states over a longer time span. For the case study presented in this paper, both extreme values, GAMMA = 0 and GAMMA > 0.9, yielded good results, which may be reasonable, as reflected in the title of Andy Grove's renowned management book *Only the Paranoid Survive*. This paper adopted a GAMMA value of 0.9.

3.4.3. Batch Processing Parameters in Supervised Learning

In supervised learning, the following parameters are associated with batch processing during model training:

1. nEpochs (Number of Epochs): Represents the number of complete training iterations over the entire training dataset. Each epoch corresponds to one full pass through the dataset;
2. nSamples (Number of Samples): Refers to the total number of samples in the training dataset, i.e., the number of data points fed into the model for training;
3. batchSize (Batch Size): Refers to the number of samples used to update the model parameters during each training step. The dataset is divided into equal-sized batches, each containing batchSize samples. The model performs forward and backward propagation on each batch, followed by an update to the model parameters.

The relationship between the number of training iterations and these three parameters can be expressed as

$$Total\ number\ of\ training\ iterations = \frac{nEpochs \times nSamples}{batchSize} \quad (6)$$

Equation (6) illustrates the total number of parameter updates performed during training. Increasing the batchSize reduces the number of iterations per epoch, but increases the number of samples processed in each iteration. Conversely, decreasing the batchSize increases the number of iterations per epoch, while reducing the number of samples processed in each iteration. Choosing an appropriate batchSize can affect both the training speed and performance of the model.

In this study, the supervised learning method uses the following batch processing parameters: nEpochs = 10, nSamples = 320, batchSize = 32.

4. The Training Process and Results

The model training in this study was conducted on the AutoDL platform, which provides a high-performance computing environment with the following specifications: the operating system is Ubuntu 18.04, with Python 3.8 and PyTorch 1.7.0, supporting CUDA 11.0 acceleration. The GPU used is a single NVIDIA RTX 3090 (24 GB VRAM), manufac-

tured in Santa Clara, CA, USA. The CPU is a 12-core virtual processor (Intel Xeon Platinum 8375C, 2.90 GHz), with 72 GB of memory. Storage includes a 30 GB system disk and a 50 GB SSD data disk, meeting the requirements for large-scale data processing.

The training curves of the two methods are shown in Figure 7. The DQN model exhibited an S-shaped curve, with slow initial win rate growth due to random exploration. After approximately 1.5 million steps of training, the win rate stabilized above 90% (training stop condition set to 99%). In contrast, the supervised learning curve resembled the Receiver Operating Characteristic (ROC) curve of a high-performance classification model, with the win rate increasing rapidly from the start. After around 500,000 steps, the win rate also stabilized above 90% (training stop condition set to 99%).

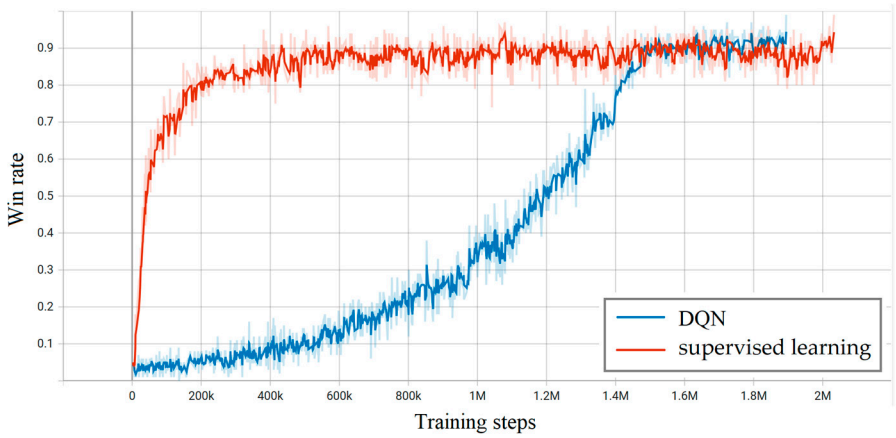

Figure 7. Training curves of the two methods (smoothing parameter = 0.6).

To evaluate the performance of Minesweeper agents trained using two algorithms (DQN and SL), 1000 test runs were conducted, each containing 1000 games. In each test run, the win rate of the model was recorded, and the average win rate and standard deviation across all test runs were calculated to assess the model's performance. The win rate results for DQN and SL are shown in Table 2. To provide a comprehensive evaluation of the methods' effectiveness, their performances were compared with those of the SP algorithm, the CSP algorithm, and human players, with the SP algorithm implemented in this paper and the performance of the latter two reported in [7].

The results showed that the win rates of DQN and SL were more than 10 percentage points higher than that of the SP algorithm, with less fluctuation, as expected. This is because both DQN and SL, based on deep neural networks, optimize decisions over larger spatial dimensions and longer time spans. The results in Table 1 demonstrate that DQN and SL outperformed both human players and the baseline algorithms (SP and CSP) in Minesweeper, exhibiting high levels of intelligence. Furthermore, although DQN slightly outperformed SL, its training speed and robustness were inferior to those of SL. The robustness here refers to the fact that DQN exhibited greater fluctuation during training, leading to poorer results in some instances, while SL maintained a higher win rate across more training runs with less fluctuation in the results.

Table 2. Comparison of win rates among different models or algorithms.

Model/Algorithm	DQN	SL	SP	CSP	Human Players
Win Rate	93.30% ± 0.77%	91.2% ± 0.9%	77.99% ± 1.30%	84~90%	87% ± 4.5%

5. Discussion

5.1. Mechanisms of Two Algorithms in the Minesweeper Task

Compared to the DQN model, supervised learning not only converged more rapidly but also exhibited enhanced training stability when applied to the training of a Minesweeper agent. This section provides an analysis of the mechanisms underlying both algorithms in the Minesweeper task, aiming to elucidate the reasons for their performance differences.

The DQN model is a classic deep reinforcement learning algorithm. It follows a reinforcement learning framework and trains the agent through trial and error: the network repeatedly plays the Minesweeper game, selecting a cell each time. If the cell is not a mine, back-propagation is used to inform the network that the probability of the cell being safe should be higher. If it is a mine, back-propagation similarly updates the network to increase the probability of the cell being a mine. The amount by which the probability should be increased or decreased depends on the set learning rate. Additionally, the DQN model uses deep learning as a computational tool by incorporating artificial neural networks, significantly expanding the range of problems that reinforcement learning can address. For example, even for a simple 6 × 6 grid with four mines, Monte Carlo simulations show that the number of possible minefield states exceeds 5.3 million. Traditional Q-learning requires generating a table that includes all possible states and performing a search to map states to actions. This table-driven approach is computationally and storage-intensive, making it impractical on current hardware. The DQN model replaces the table with a neural network, allowing state-to-action mapping through simple network calculations. The artificial neural network used in this study contains 896 nodes and 676,804 parameters, achieving a two-order-of-magnitude compression of the minefield state space. In summary, the DQN model is a network-driven approach that integrates the feedback mechanism with data compression, and after approximately 8 h of training, it produced a network capable of playing Minesweeper.

Supervised learning, on the other hand, uses the true labels of the minefield to train the model, predicting the probability of each cell being a mine. The agent progressively uncovers the minefield by selecting the cell with the lowest mine probability, similar to a human player's review and backtracking process. This method is intuitive and efficient, helping to improve the agent's decision-making quality and training stability. The algorithmic structure of supervised learning is similar to that of DQN, and both methods in this study employ the same deep neural network. The key difference is that supervised learning uses simple minefield labels instead of the complex reward mechanism in DQN, significantly simplifying the feedback process and accelerating training. In about 1.8 h of training, supervised learning could produce a network capable of playing Minesweeper.

An apt analogy for the relationship between DQN and supervised learning would be comparing electric vehicles to gasoline vehicles: while both share similar overall structures, the electric vehicle eliminates more complex components such as the engine and transmission system, replacing them with simpler components such as the power battery system, electric drive, and control systems, leading to superior acceleration and lower maintenance costs.

5.2. Performance Analysis: Why Supervised Learning Trains Faster than DQN

From the perspective of means-to-ends analysis, both methods optimize by comparing the current state with the target and gradually narrowing the gap. The key difference lies in the goals: supervised learning aims to match the true labels of the minefield state, which provides more global information, while DQN uses cumulative rewards as the target, offering more local information. To illustrate with an analogy: a ship sailing across

the vast ocean at night is heading toward a destination, but its navigation system is malfunctioning and can only intermittently provide the distance to the destination. The ship must adjust its course based on these distances, gradually narrowing the gap to the destination and attempting to find the optimal path. If there is a lighthouse near the destination, the efficiency of the voyage will greatly improve. Heading directly toward the lighthouse is likely the optimal or satisfactory choice, significantly reducing the complexity of finding the optimal course.

5.3. Performance Analysis: Why Supervised Learning Trains More Stably than DQN

From the perspective of feedback control, both algorithms adjust network parameters using a loss function as a feedback signal to improve win rates. The main difference lies in the nature of their loss functions: supervised learning's loss function measures the difference between the predicted minefield state and the true labels, providing a more direct and shorter feedback path, whereas DQN's loss function is based on the update increments of the optimal Q-function, which represents the expected cumulative reward. Although rewards are positively correlated with minefield state predictions, the Q-function update process introduces indirectness, resulting in a longer feedback path. Improper reward function design can lead to insufficient feedback, thereby reducing training performance. Designing the reward function is a challenging task, as many seemingly reasonable reward values may not perfectly align with the underlying logic of the Minesweeper game, leading to unstable training performance.

Wiener, in Reference [16], provided an insightful analogy: during the process of picking up a pencil, the brain must continually receive feedback about the gap between the hand and the pencil to adjust the movements, which are then transmitted through the spinal cord to motor neurons to reduce this gap. If feedback is insufficient, the movement cannot be completed, as seen in patients with motor ataxia caused by syphilis of the central nervous system. Conversely, excessive feedback can lead to tremors, as seen in patients with cerebellar damage, who may exhibit excessive reaching movements and uncontrollable oscillations. On the other hand, longer feedback chains exacerbate the instability of model training performance. Arnol'd, in Reference [17], pointed out that the longer the derivation chain, the less reliable the final conclusion, a perspective that helps to understand the impact of long feedback paths on training stability. Control behavior is a complex process of random trial and error, and longer feedback chains make the outcomes more uncertain.

5.4. A Glimpse into the Future: Artificial General Intelligence = Deep Learning + Reinforcement Learning?

The results of this study indicate that the DQN model shows significant potential in enhancing the agent's performance on specific tasks and exhibits some ability to generalize from past experiences. However, its limitations are also apparent. First, the sample efficiency of the DQN model is relatively low. Achieving satisfactory performance often requires a large number of training samples, which can lead to wasted computational resources in practical applications, especially in scenarios where data acquisition is costly or computational resources are limited. Additionally, DQN exhibits instability during training, with significant fluctuations in the number of training episodes and performance between episodes, which can result in suboptimal convergence or poor model performance. In contrast, supervised learning offers higher sample efficiency and better training stability, but its limitations are also apparent, primarily the need for true labels of the target states. For many real-world problems, such as Atari games, the target state may not exist or its label may be inaccessible, limiting the applicability of supervised learning.

The case study in this research highlights that AI technologies based on deep neural networks have yet to resolve the trade-off between model generalization and computational efficiency. The view that "general artificial intelligence is a combination of deep learning and reinforcement learning" should be approached with caution, as this perspective appears overly optimistic. At present, deep reinforcement learning methods, including DQN, lack the strong generalization required for achieving artificial general intelligence. From a more pessimistic viewpoint, deep learning without reinforcement learning is blind, and reinforcement learning without deep learning is lame. The combination of deep learning and reinforcement learning is akin to a blind person carrying a lame person across a river—while it may succeed in certain cases, it is not a lasting or effective solution.

6. Conclusions

In the training of the Minesweeper game agent, the DQN model strikes a good balance between generalization and computational complexity, demonstrating its potential for applications in large state spaces and complex decision-making tasks. Supervised learning, leveraging the known true labels of the minefield, simplifies the algorithm structure and significantly improves training efficiency. Generally, for problems with abundant data samples, high-quality annotations, and high determinism, DQN tends to underperform compared to supervised learning. For problems requiring strong generalization, supervised learning often falls short, whereas reinforcement learning methods such as DQN offer a clear advantage.

For larger grids and higher mine densities in the Minesweeper environment, the framework proposed in this paper remains applicable, though it may require more computational resources, longer training times, and could experience increased instability during training. To further optimize the training process, future research can explore two directions: first, applying a Curriculum Learning strategy, starting with the trained agent and progressively increasing task difficulty as the agent's performance improves in more complex environments, thereby enhancing learning efficiency and stability; second, integrating both methods, with supervised learning leading the training process and fine-tuning parameters in the final stage using DQN or other deep reinforcement learning methods. By combining their strengths, it is expected that a more efficient and higher-performing Minesweeper agent can be trained.

Author Contributions: Conceptualization and design of the research, W.W.; funding acquisition and methodology, C.L.; writing—original draft, W.W.; writing—review and editing, C.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Young Innovative Talents Project in Guangdong Province, grant number 2019KQNCX013.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

Acknowledgments: We would like to thank the reviewers for their valuable comments and suggestions on this paper. We also appreciate Professor Xiao Jie from South China Agricultural University for her assistance in improving the language quality of this paper.

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

References

1. Minesweeper (Video Game). Available online: https://en.wikipedia.org/wiki/Minesweeper_(video_game) (accessed on 20 December 2024).
2. Kaye, R. Minesweeper is NP-complete. *Math. Intell.* **2000**, *22*, 9–15.
3. Russell, S.J.; Norvig, P. *Artificial Intelligence: A Modern Approach,* 4th ed.; Pearson Education: London, UK, 2021; pp. 164–191.
4. Studholme, C. Minesweeper as a Constraint Satisfaction Problem. 2000. Available online: https://www.cs.toronto.edu/~cvs/minesweeper/minesweeper.pdf (accessed on 17 January 2025)
5. Becerra, D. Algorithmic Approaches to Playing Minesweeper. Bachelor's Thesis, Harvard College: Cambridge, MA, USA, 1 April 2015.
6. Gardea, L.; Koontz, G.; Silva, R. Training a Minesweeper Solver. *Autumn CS* **2015**, *229*, 1–5. Available online: https://cs229.stanford.edu/proj2015/372_report.pdf (accessed on 17 January 2025)
7. Hansen, J.J.; Havtorn, J.D.; Johnsen, M.G.; Kristensen, A.T. Evolution strategies and reinforcement learning for a minesweeper agent. *Deep. Learn. DTU Comput.* **2017**, *02456*, 1–8.
8. Singh, M.; Cambronero, J.; Gulwani, S.; Le, V.; Negreanu, C.; Verbruggen, G. Codefusion: A pre-trained diffusion mode for code generation. In Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, Singapore, 6–10 December 2023. Available online: https://www.microsoft.com/en-us/research/uploads/prod/2023/11/CodeFusion-Revised-CameraReady.pdf (accessed on 17 January 2025)
9. Barua, D.K.; Halder, M.; Shopnil, S.; Islam, M.M. Human trajectory imputation model: A hybrid deep learning approach for pedestrian trajectory imputation. *Appl. Sci.* **2025**, *15*, 745.
10. Li, M.; Liu, M.; Zhang, W.; Guo, W.; Chen, E.; Hu, C.; Zhang, M. An algorithm for predicting vehicle behavior in high-speed scenes using visual and dynamic graphical neural network inference. *Appl. Sci.* **2024**, *14*, 8873.
11. Jeong, H.; Lee, H.; Kim, C.; Shin, S. A Survey of Robot Intelligence with Large Language Models. *Appl. Sci.* **2024**, *14*, 8868.
12. Huang, S.; Xiao, S.; Chen, Y.; Yang, J.; Shi, Z.; Tan, Y.; Wang, S. Unified structure-aware feature learning for graph convolutional network. *Expert Syst. Appl.* **2024**, *254*, 124397.
13. Li, H.; Pang, X.; Sun, B.; Liu, K. A concise review of intelligent game agent. *Entertain. Comput.* **2025**, *52*, 100894.
14. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533.
15. 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.
16. Wiener, N. *Cybernetics or Control and Communication in the Animal and the Machine*, 2nd ed.; Reissue of the 1961; MIT Press: Cambridge, MA, USA, 2019; pp. 75–76.
17. Arnol'd, V.I. On teaching mathematics. *Russ. Math. Surv.* **1998**, *53*, 229–236.