applied sciences

MDPI

Article

Intelligent Decision Making Based on the Combination of Deep Reinforcement Learning and an Influence Map

Xiaofeng Lu [1,*], Ao Xue [1], Pietro Lio [2] and Pan Hui [3]

[1] School of Cyberspace Security, Beijing University of Posts and Telecommunications, Beijing 100876, China
[2] Computer Laboratory, University of Cambridge, Cambridge CB2 1PZ, UK
[3] Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, China
* Correspondence: luxf@bupt.edu.cn

Abstract: Almost all recent deep reinforcement learning algorithms use four consecutive frames as the state space to retain the dynamic information. If the training state data constitute an image, the state space is used as the input of the neural network for training. As an AI-assisted decision-making technology, a dynamic influence map can describe dynamic information. In this paper, we propose the use of a frame image superimposed with an influence map as the state space to express dynamic information. Herein, we optimize Ape-x as a distributed reinforcement learning algorithm. Sparse reward is an issue that must be solved in refined intelligent decision making. The use of an influence map is proposed to generate the intrinsic reward when there is no external reward. The experiments conducted in this study prove that the combination of a dynamic influence map and deep reinforcement learning is effective. Compared with the traditional method that uses four consecutive frames to represent dynamic information, the score of the proposed method is increased by 11–13%, the training speed is increased by 59%, the video memory consumption is reduced by 30%, and the memory consumption is reduced by 50%. The proposed method is compared with the Ape-x algorithm without an influence map, DQN, N-Step DQN, QR-DQN, Dueling DQN, and C51. The experimental results show that the final score of the proposed method is higher than that of the compared baseline methods. In addition, the influence map is used to generate an intrinsic reward to effectively resolve the sparse reward problem.

Keywords: reinforcement learning; deep reinforcement learning; influence map; sparse reward

Citation: Lu, X.; Xue, A.; Lio, P.; Hui, P. Intelligent Decision Making Based on the Combination of Deep Reinforcement Learning and an Influence Map. *Appl. Sci.* **2022**, *12*, 11458. https://doi.org/10.3390/app122211458

Academic Editor: Keun Ho Ryu

Received: 13 October 2022
Accepted: 5 November 2022
Published: 11 November 2022

1. Introduction

Reinforcement learning is a branch of machine learning, the purpose of which is to study how the agent learns through trial and error. Reinforcement learning uses reward or punishment mechanisms to make agents more inclined toward certain behaviors in the future [1]. Compared with traditional machine learning algorithms, reinforcement learning has no supervisors—only reward signals.

The human brain understands and recognizes things visually and builds and improves its own cognitive system. When traditional reinforcement learning uses images as the training state, reinforcement learning algorithms, such as Q-Learning, SARSA, etc., cannot find the optimal function because they are limited by the considerable possibilities of the state space. Convolutional neural networks (CNNs) have achieved considerable success in the field of computer vision, so their combination with reinforcement learning has been proposed. The first deep reinforcement learning algorithm, Deep Q-Network (DQN), was proposed by DeepMind [2]. Since its combination with deep learning, reinforcement learning has made considerable breakthroughs. Like humans, deep reinforcement learning agents receive information from high-dimensional input, such as vision, to train their own neural networks and obtain higher scores in each learning environment under the specific reward function.

Deep reinforcement learning is currently one of the most popular areas of artificial intelligence (AI) and has achieved amazing results in many games. It does not require additional information, such as data tags, and is worthy of further exploration for future applications. If a neural network uses a single frame as the input, it is difficult for the neural network to analyze the dynamic information of the image. In deep reinforcement learning, neural networks usually use four consecutive frames as the input to learn the dynamic information contained in the image. Such a design allows the CNN to learn more information.

However, the four consecutive frames retain dynamic information at the cost of a certain amount of redundancy. If this portion of the redundant information could be reduced, the training speed and memory usage of the reinforcement learning agent could be improved. An influence map is an AI decision-making tool that is regularly used in games to describe the current state. In addition, the dynamic influence map can express the motion information of the agent. If reinforcement learning could learn the dynamic information from the influence map, it would be of great help to the learning process of the agent.

The main research goal of this study Is to explore the combination of dynamic influence maps and deep reinforcement learning. If a dynamic influence map can replace four consecutive frames as the neural network input, then all deep reinforcement learning algorithms in the same learning environment will be improved to a certain extent. In deep reinforcement learning, there are many scenarios of sparse rewards, i.e., the agent can only receive rewards after the conclusion of the game, such as in Go and chess. In this situation, the agent does not receive any external reward most of the time, which may result in the inability to optimize the policy. In this paper, we propose that in the task of sparse rewards, an influence map can be used to optimize the agent policy and improve the overall performance of deep reinforcement learning.

To verify the possibility of combining a dynamic influence map with reinforcement learning, Ms. Pac-Man, a popular test environment in the field of AI [3–8], is used as the learning and evaluation environment. In this kind of environment, the complete capabilities of the dynamic influence map, which represents the dynamic information of the current state of the game, can be displayed. It is these capabilities that facilitate its combination with reinforcement learning.

The contributions of this article are as follows.

1. One frame of an original image superimposed on the influence map is used to express dynamic information. Our method inputs less data into the neural network than the method using of four consecutive frames in deep reinforcement learning. It achieves better performance than the use of four consecutive frames in deep reinforcement learning.

2. In the sparse reward task, the use of an influence map is proposed to generate an intrinsic reward when there is no external reward. Sparse reward is an issue that must be solved in refined intelligent decision making.

3. The experiments conducted in this study prove that the combination of a dynamic influence map and deep reinforcement learning is effective. The experimental results show that the proposed method exhibits a greater improvement than traditional deep reinforcement learning in terms of agent performance, training speed, and memory usage. The influence map method proposed in this paper can be used in any algorithm. For some training tasks, if the algorithm cannot be further improved, the influence map can be added for training to improve the upper performance limit.

2. Related Work

2.1. Deep Reinforcement Learning

After the first deep reinforcement learning algorithm, Deep Q-Network, was introduced [2], a large number of algorithms based on the Deep Q-Network were proposed. Hasselt et al. [9] proposed Double Q-Learning, which improves the performance by reducing overestimating the deviation. Prioritized Experience Replay (PER) assigns a priority to

each experience and uses importance sampling weights to correct the priority deviation [10]. Sutton [11] proposed a DQN algorithm, which uses multi-step time difference to reduce the deviation of the value function estimation. Dueling DQN divides the structure of the neural network into two routes [12] corresponding to the advantage value and the state value. NoisyNet adds noise to the neural network parameters to the encourage exploration behavior of agents [13]. Bellemare et al. [14] proposed the C51 algorithm, which treats the expected return as a distribution rather than a value. The Rainbow algorithm combines the improvements of the above DQN algorithm, achieving the best performance [15].

Among policy-based algorithms, the REINFORCE algorithm and the Vanilla Policy Gradient algorithm were originally proposed by Williams [16] to optimize the policy by optimizing the no-discount expected return function. A policy-based algorithm makes unbiased predictions but encounters high variance and hard convergence problems. Konda and Tsitsiklis [17] proposed the Actor-Critic based on the policy gradient algorithm, using two models to evaluate the current state and action. To improve the Actor-Critic algorithm, Asynchronous Advantage Actor-Critic (A3C) was proposed [18]. Compared to the DQN algorithm, it supports parallel training, so the training speed is considerably improved. The Trust Region Policy Optimization (TRPO) algorithm uses KL divergence to limit the range of policy changes [19]. Wu et al. [20] proposed Actor-Critic using Kronecker-Factored Trust Region (ACKTR), which reduces the calculation cost of KL divergence calculation and has better sampling utilization. The Actor-Critic with Experience Replay (ACER) algorithm is an improvement of the A3C algorithm [21]. It uses experience replay to make training more stable and efficient. Schulman et al. [22] proposed the Proximal Policy Optimization (PPO) algorithm, proposing the idea of clipping, using a similar idea to that of TRPO, considerably reducing computational consumption.

The Deterministic Policy Gradient (DPG) proposed by Silver et al. [23] is another type of policy-based algorithm, which replaces the stochastic policy with a deterministic policy. The DDPG proposed by Lillicrap [24] is an improvement of DPG. By adding an actor-critic structure to DPG, the performance was considerably improved. However, there are problems of divergence and hyperparameter sensitivity. Fujimoto et al. [25] proposed the Twin Delayed DDPG (TD3) algorithm to improves the DDPG algorithm. It reduces the training variance by delaying policy update and adding noise to the target network, making the DDPG algorithm more stable. The SAC algorithm combines the ideas of Q-Learning and the DPG algorithm simultaneously [26], introducing the maximum entropy term, the mathematical proof underlying which can ensure convergence to the optimal policy.

As a distributed algorithm, Horgan et al. [27] proposed the Ape-x algorithm, which separates interaction from training and uses a large number of actors to collect data. It has achieved state-of-the-art performance on many benchmarks. In addition, the Distributed Distributional Deep Deterministic Policy Gradient (D4PG) distributed algorithm uses DDPG as the basic algorithm and achieves satisfactory performance [28]. Similarly, the Importance Weighted Actor-Learner Architecture (IMPALA) uses Actor-Critic as the basic algorithm, with multiple learners to perform asynchronous gradient updates [29].

In the application of reinforcement learning, AlphaGo developed by DeepMind and the successor, AlphaZero [30], defeated the top human players in Go. OpenAI Five developed by OpenAI [31] defeated the world champion in Dota2.

2.2. Using an Influence Map in Game Decision Making

An influence map is an AI decision-making that is usually used in games and is able to describe the current game state. Nathan et al. [32] used an influence map AI in Pac-Man, implementing a simple hill-climbing optimization method to find the hyperparameter combination. Su-Hyung et al. [33] combined neural networks and an influence map to evolve NPC behaviors in simulation games. Hogler [34] used an influence map to realize the "Flocking" technique, which is the smooth movement of the team in real-time strategy games. An influence map was combined with Monte Carlo Tree Search (MCTS) [35], using MCTS to search for nearby paths and an influence map to find the global path. An influence

map was combined with the A* algorithm to choose a safe path in a pathfinding game [36]. Athanasios et al. [37] used an influence map to simulate crowds in a game and generated pedestrians with varying behaviors. Phillipa et al. [38] used an influence map to coevolve the agents of both sides in real-time strategy games. Chris et al. [39] used genetic algorithms combined with influence maps to evolve AI in games.

Johan et al. [40] designed an influence map AI for the ghost and the Pac-Man in Ms. Pac-Man. They assigned different weights to the influence of pills, ghosts, and power pills. The results showed that the controller based on the influence map performed well. Lu [41] proposed a dynamic influence model that changes the spread of influence in different directions by adjusting the distances. Kyungeun et al. [42] combined Q-Learning with an influence map. Compared with using Q-Learning alone, the same learning result was achieved using only 4.6% of the state space. Heckel et al. [43] proposed the use of a navigation grid to represent an influence map, which was proven to reduce memory usage.

3. Dynamic Influence Map

3.1. Decision Making with an Influence Map

An influence map is a decision-making tool used in AI games. In the game, props such as food, wealth, and tools have a positive impact on the current position, and the agent tends to move to places with greater positive influence. Enemies and negative props have a negative impact on the location, so the agent needs to avoid these locations. Various objects in the game have varying influences on their surrounding positions, and the influence map represents the calculation of the sum of the influence of each position on the map.

When an object spreads influence to the outside world, the numerical influence value in each location on the map is affected by the spread and attenuation modes, so the calculated influence map changes. For example, the Euclidean distance, Manhattan distance, etc., can be used to measure the distance when the influence value spreads. An adjacent spread mode, such as the flood-fill algorithm, can also be used for spreading. However, the spread of the influence value is no longer measured by the distance. In addition, the influence cannot spread infinitely, and the influence value gradually attenuates as spreading occurs. When a threshold is reached, the influence value is set to 0. The attenuation process can be calculated using linear or exponential attenuation.

A traditional influence map usually has the same influence attenuation speed in the surroundings around the source, but this convention cannot reflect the mobility of the moving game units. For some moving game objects, their influence values differ depending on the direction. Specifically, moving units exert greater influence on other objects in the direction of their movement, and locations in the opposite direction of their movement have less influence. This is the core concept of dynamic influence maps.

3.2. Spread and Attenuation Modes

3.2.1. Spread Modes

When the starting point spreads influence to different locations on the map, the spread mode refers to the method used to measure the distance from the starting point to the destination point. The most commonly used methods include the Euclidean distance, Manhattan distance, path length between two points, and adjacent spread. Among these methods, the use of the two-point path length as the spread mode requires all paths to be calculated in advance because the real-time path finding affects the spread efficiency.

The coordinate system is established with the upper-left corner of the game scene as the coordinate origin.

(1) Euclidean Distance

When the Euclidean distance is used as the spread mode, the distance between the spread source and each point in the map is expressed as follows:

$$d_{a,b} = d_{euclidean(a,b)} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}, \quad (1)$$

where $d_{euclidean(a,b)}$ represents the Euclidean distance between point a and point b. The coordinates of point a are (x_1, y_1), and the coordinates of point b are (x_2, y_2).

This spread mode calculates the distance quickly because it only needs two coordinates to obtain a relatively acceptable result. However, if there are obstacles and other information in the path, this spread mode cannot perceive the distance.

(2) Manhattan distance

When the Manhattan distance is used as the spread mode, the distance between the spread source and the target point is measured as follows:

$$d_{(a,b)} = d_{Manhattan(a,b)} = |x_1 - x_2| + |y_1 - y_2|, \quad (2)$$

where $d_{Manhattan(a,b)}$ represents the Manhattan distance between point a and point b. The coordinates of point a are (x_1, y_1), and the coordinates of point b are (x_2, y_2).

(3) Path distance

In addition, the path between two points can be used to represent the distance between these points. The measurement method is as follows:

$$d_{(a,b)} = d_{path(a,b)}, \quad (3)$$

where $d_{path(a,b)}$ represents the path length obtained using the pathfinding algorithm. Compared with the Euclidean and Manhattan distance spread modes, this method can take into account the existence of obstacles. Because the pathfinding algorithm usually finds the shortest path, the measurement of distance is often quite accurate. However, in an actual game, especially real-time strategy games such as war games, the response time is an extremely important factor, and the calculation speed of the pathfinding algorithm seriously affects the real-time response ability.

(4) Adjacent spread

The adjacent spread method only needs to be executed once to spread the initial influence from the starting point to all points on the map. Using this method, a considerable amount of time can be saved as compared to the use of the pathfinding algorithm. Compared with the use of the Euclidean or Manhattan distance as the spread mode, the adjacent spread mode considers obstacles and other factors. Moreover, when a game unit moves, the influence map can be dynamically calculated and changed accordingly, which is more adaptable to dynamic games.

Adjacent propagation does not calculate the path length between any two points directly. Instead, it propagates the influence of the adjacent grids of the game object. If the influence values of some of the adjacent grids are changed, then an AI agent propagates the new influence values from these grids with the same method. The influence value can be propagated to four or eight directions in square grids (as shown in Figure 1a,b, respectively) or to six directions in hexagonal grids (as shown in Figure 1c). This process is similar to the flood-fill algorithm.

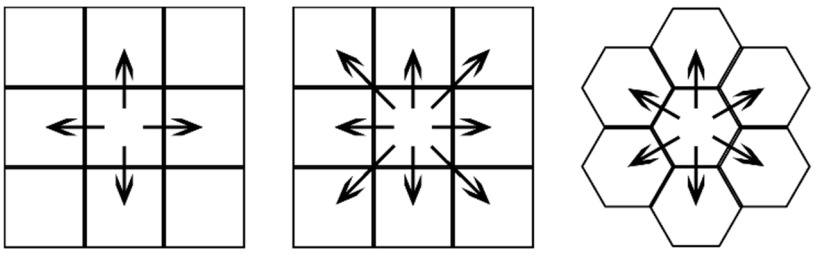

Figure 1. Propagation directions. (**a**) Four directions (**b**) Eight directions (**c**) Six directions.

3.2.2. Attenuation Modes

Attenuation modes are generally divided into exponential and linear attenuation modes. As the distance between the starting point and the target point increases, the influence gradually decreases. In addition, when the influence is attenuated to a certain degree, it is usually set to 0 to reduce the amount of calculation.

Consider point a as the starting point of spreading and point b as a target point on the map. I_a represents the initial influence of point a, and $I_{a,b}$ represents the exerted influence from point a to point b.

Exponential attenuation: The relationship between the influence and distance is given by Equation (4).

$$I_{a,b} = \begin{cases} I_a \cdot \gamma^{d_{a,b}}, & d_{a,b} < D_{max} \\ 0, & d_{a,b} \geq D_{max} \end{cases}, \tag{4}$$

where γ is the exponential attenuation parameter, $d_{a,b}$ represents the distance from point a to point b, and D_{max} represents the maximum spread distance. After exceeding this distance, the influence value drops to 0. The value of the exponential attenuation parameter (γ) is usually between 0 and 1. The lower the value, the faster the attenuation speed of the starting influence.

Linear attenuation: The relationship between influence and distance is given by Equation (5).

$$I_{a,b} = \begin{cases} I_a - \beta \cdot d_{a,b}, & d_{a,b} < D_{max} \\ 0, & d_{a,b} \geq D_{max} \end{cases}, \tag{5}$$

where β is the linear attenuation parameter.

When these two attenuation modes use adjacent spread, the adjacent spread no longer spreads after the influence value of a certain point is less than a preset value instead of stopping spreading after reaching the maximum spread distance.

3.3. Dynamic Influence Map

In this study, the adjacent spread mode and linear attenuation are used to calculate the spread of influence. Compared with Manhattan distance and Euclid distance, adjacent spread can take into account the factors of obstacles, making the distance measurement more accurate. Compared with the distance obtained by the pathfinding algorithm, the calculation speed of adjacent spread is faster. Both exponential attenuation and linear attenuation can be used. Considering the computational complexity, the exponential operation consumes more time, so linear attenuation is selected.

3.3.1. Dynamic Influence Map Calculation

Because the object is in motion, its influence decays at different rates in different directions. When the target point is in front of the moving direction of the object, the influence of the object decays slowly. When the target point is behind the moving direction of the object, the influence of the object declines faster. One way to adjust the rate at which an object's influence decays is to change the distance in the formula between the object and the target point.

As shown in Figure 2, the spread source point is a, and the target point is b. $\vec{v}$ is the motion direction of node a, and $\vec{I_{a,b}}$ is the direction from a to b. The specific method for adjusting the distance is given by Equation (6) [41].

$$d'_{a,b} = \frac{d_{a,b}}{1 + \cos\left(\vec{I_{a,b}}, \vec{v}\right) \cdot \alpha}, \tag{6}$$

where $d_{a,b}$ indicates the actual distance from point a to point b, $d'_{a,b}$ represents the adjusted distance, $\vec{I_{a,b}}$ represents the direction vector from point a to point b, and $\vec{v}$ represents the current movement direction of spread source point a. Moreover, α represents the parameter

of distance adjustment, which is between 0 and 1. The larger the value of α, the greater the value of the spreading influence of source point a in the direction of its movement. In this study, $d'_{a,b}$ is used to calculate the influence of b according to Equation (5).

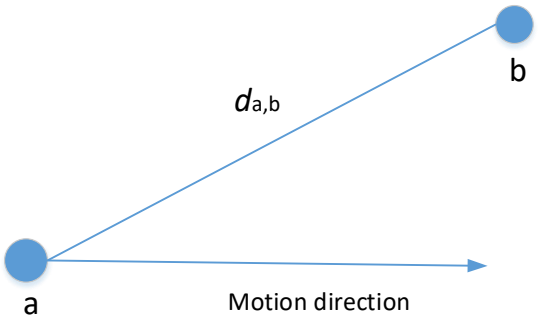

Figure 2. The distance scaling of a dynamic influence map.

Let $IM(x)$ indicate the influence value at point x in the influence map. The calculation formula of the influence map can then be expressed as Equation (7).

$$IM(x) = \sum_{i \in U} I_{i,x}, \quad (7)$$

where $I_{i,x}$ represents the magnitude of influence exerted by game unit i on point x.

3.3.2. Dynamic Influence Map in Ms. Pac-Man

By setting the influence values of different units and using different modes for spreading and attenuation, the influence map can express the current state information, trend, and direction of movement using numerical values. In this study, Ms. Pac-Man is used as the learning environment, and an initial influence value is set for each game unit, such as ghosts, pellets, and Ms. Pac-Man. At each moment, the locations of these units are taken as the starting points, the directions of movements are considered, the influence values are spread to the whole map, and these influence values are added to form the global dynamic influence map. Such a dynamic influence graph contains motion information and can be used to replace the four consecutive frames as the state space in reinforcement learning.

4. Using the Influence Map as the State Space in Reinforcement Learning

4.1. Deep Reinforcement Learning Algorithm Ape-x

Ape-x is a distributed deep reinforcement learning algorithm that achieves state-of-the-art performance in many reinforcement learning tasks. Ape-x is characterized by a distributed architecture for large-scale deep reinforcement learning, which separates data collection from training. Using their local neural networks, multiple actors interact with multiple copies of environments simultaneously. The network selects actions, collects interactive data, and stores them in the global experience replay. The learner is responsible for sampling data from the global experience replay and training the core neural network. The neural networks between actors share the same parameters, and the learner sends the neural network parameters to the actors at intervals. Different actors have different epsilons (an epsilon-greedy algorithm) and remain unchanged during the whole interaction. This design allows the actors to generate sufficiently varies learning data when interacting with the environment.

In this study, DQN is used as the basic algorithm to train the Ape-x distributed algorithm. DQN uses a neural network to fit the state-action value function of Q-Learning. The DQN neural network takes consecutive frames as input and outputs the Q-value for each possible action to evaluate the quality of the actions. Because the action space is discrete, it is difficult to use the objective function to directly optimize the policy. Moreover, DQN estimates the state-action value function of each action and then selects the appropriate

action, so it is suitable for use in the discrete action space in the Atari environment. The architecture of the Ape-x algorithm is presented in Figure 3.

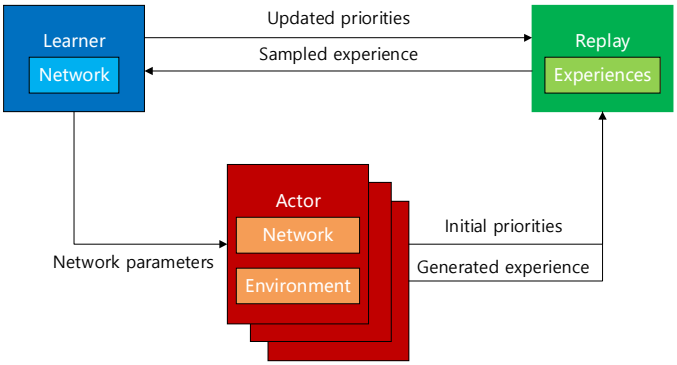

Figure 3. Architecture of the Ape-x algorithm.

As shown in Figure 3, the Ape-x architecture has multiple actors, each of which interacts in its independent environment instance. First, the initial priorities in the local experience replay pool are calculated, and local experience is then sent to the global experience replay pool. There is only one learner, the training data of which are sampled from the global experience replay pool; the core neural network is trained, and the data priorities are updated.

The Ape-x algorithm can support many actors to collect data simultaneously, and the learner is only used to train the core neural network. In this way, the training speed and data collection speed can be considerably improved.

4.2. Using an Image as State Space

The reinforcement learning toolkit Gym contains many reinforcement learning tasks. Among them, the Atari 2600 game simulator is often used to compare the performance of various algorithms. All these tasks use images as the state space, so neural networks are also built based on images in a large number of reinforcement learning algorithms. An image can represent very large states as a high-dimensional input, so traditional reinforcement learning algorithms, such as Q-Learning, usually cannot be used for training. At this time, a neural network must be added to understand the image state space and fit the state-action value function.

When using images as the state space, the environment returns an RGB image at every moment of interaction. To save storage space and speed up training, images are usually cropped and downsampled to low-resolution images, then converted to grayscale images. The grayscale conversion can be calculated by Equation (8).

$$Gray = 0.2989 \times R + 0.5870 \times G + 0.1140 \times B, \tag{8}$$

The convention of deep reinforcement learning algorithms is to return one image every four frames to speed up the interaction. After four images are accumulated, they are combined into one four-channel image in which each channel is the raw image. The third and fourth frames are maximized to prevent image flicker. In addition, at the beginning of each episode, the agent usually chooses to act randomly or to stand by for as many as 30 frames to increase the randomness of the interaction.

4.3. Combining the Original Image and Influence Map as the State Space

Influence maps allow Pac-Man to sense danger and make smarter decisions. In Figure 4, Pac-Man moves from point A to point B. Although there are pellets ahead of his movement direction, the dangerous influence of the ghost also spreads to Pac-Man's point B, so he chooses to change direction. Moreover, the absorbing influence of the pellets below him also spreads to point B, so Pac-Man chooses to move downward.

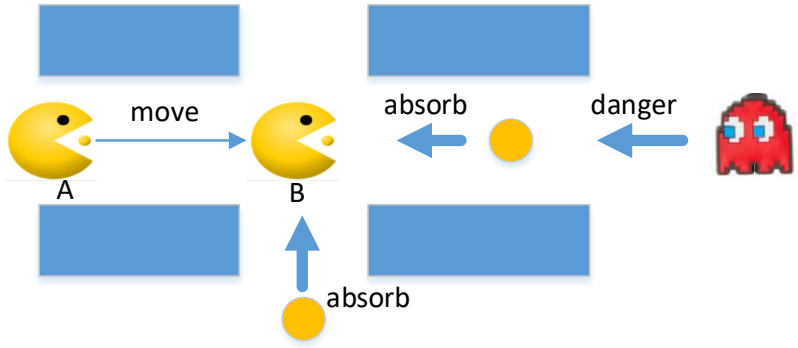

Figure 4. The role of influence maps in decision making.

When the reinforcement learning agent is trained, the agent receives the state returned by the environment and continuously learns from the interaction by understanding the information contained in the state. Therefore, the state returned by the environment directly determines the knowledge learned by the agent. Different state spaces cause the agent to prefer different policies, thereby affecting the degree of intelligence and the decision-making ability. As a high-dimensional expression, an image has considerably vast state possibilities. For example, a grayscale image with a resolution of 100×100 has $10,000^{256}$ different states, which contain large amounts of information. The more an agent can learn from such a state space, the stronger its decision-making ability.

To learn as much knowledge as possible, the original image was used as the state space in most previous research. However, an influence map is not actually an image but a matrix. The values in the matrix represent the influence value of a certain location, not the pixel value. If the values in the matrix are scaled to the range of RGB pixel values, such as 0–255, according to certain rules, or if RGB pixel values are represented by floating-point numbers, such as 0–1, the influence map can also be regarded as an image.

Now, the influence map can be superimposed on the original image. However, the number of pixels differs between the original image and the influence map. For the two images to be superimposed, the resolution must be the same. The original image can be changed to a tensor with a shape of (84, 84) by cropping, downsampling, grayscale, etc. The influence map can be upsampled by bilinear interpolation to (84, 84). In this way, two images with the same resolution can be superimposed on the z-axis to form a dual-channel image with a shape of (84, 84, 2). The two channels generated in this way are shown in Figure 5.

Because the dynamic influence map has different attenuation rates in each surrounding direction, the influence can express certain motion information in an image. Compared with the traditional four consecutive frames, the influence map can also represent motion information while reducing memory usage.

4.4. Hyperparameter Selection for the Influence Map

To obtain the influence map, if the influence of all pixels is calculated, the amount of calculation is inevitably increased, causing the interaction between the reinforcement learning agent and the environment to be much slower. Therefore, in addition to removing some text from the image, an 8×8 pixel is used as a single grid to calculate the influence map.

All the objects in the game must be considered when calculating the influence map, and each object must be allowed to spread its influence in the whole maze. The setting of the initial influence of the object directly affects the influence map generated by it, and the global influence map is affected when all influence maps are added up. Several hyperparameter search algorithms, including grid search [44], random search [45], and genetic algorithms [46], are used to find the most suitable hyperparameters the influence map for Ms. Pac-Man.

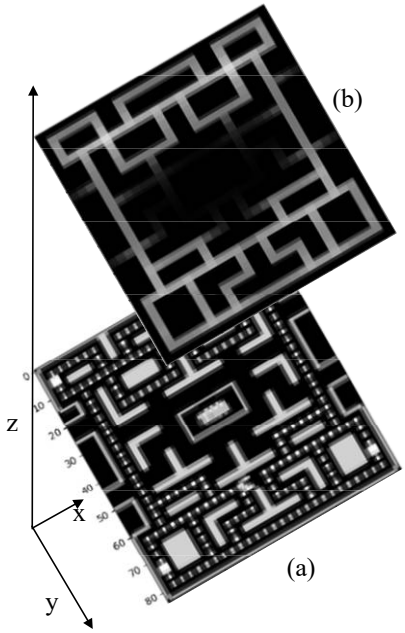

Figure 5. Dual-channel image after superimposition. (**a**) The original image. (**b**) The influence map.

To obtain the optimal hyperparameters, a game AI based on the influence map is implemented, and the actions performed at each time step are selected according to the influence values in the four surrounding directions. Adjacent spread and exponential attenuation are used when influence is spreading.

The genetic algorithm is a global search algorithm that imitates the biological evolution mechanism. It selects a relatively suitable hyperparameter combination via evolution, mutation, and survival of the fittest. In the genetic algorithm used in this study, each hyperparameter is represented by 4 bits, i.e., the range of all the hyperparameters of the influence map is divided into 16 (2^4) parts. The gene of each individual is a bit array composed of 7 hyperparameters, with a total of 28 bits. The genetic algorithm is implemented for 50 generations of evolution, and there are 100 individuals in each generation.

4.5. Using the Influence Map to Solve the Sparse Reward Problem

4.5.1. Sparse Reward

In deep reinforcement learning tasks, many environments have sparse rewards problems. For example, in tasks such as those in Go and chess, the agent can only receive rewards after the entire game is over, causing the agent to receive no reward most of the time and possibly resulting in the inability to optimize the policy.

There exists a handling technique called curiosity-driven learning [47], the idea of which is that in addition to the external reward provided by the environment, the agent provides an intrinsic reward for itself. The intrinsic reward is generated through an intrinsic curiosity module (ICM). The module predicts what will happen in the next state and compares it with the actual next state. If the predicted result differs from the actual result too much, it means that the next state is encountered less frequently, and a larger intrinsic reward is then be generated to encourage exploration of the uncertainty. Through this intrinsic reward, the agent tends to explore more diverse states, which can increase the diversity of collected data to a certain extent.

Curiosity-driven learning fails in some situations, which is called the noisy TV problem. In these situations, the state always gives the agent a higher intrinsic reward when the next state is randomly generated. The generated reward causes the agent to tend to stay in the current state, even if this state is actually worthless.

To solve the noisy TV problem, OpenAI proposed random network distillation (RND) [48]. The idea behind this technique is similar to curiosity-driven learning, but two neural networks are used to evaluate the current state. The parameters of one neural network are fixed after random initialization and remain frozen, whereas the other neural network continues learning and training. When the two neural networks have a large difference in evaluating the same state, the current state must be explored, and the agent is given a larger intrinsic reward. If a certain state is experienced many times, another neural network gradually reduces the evaluation error during the training, and the intrinsic reward obtained becomes increasingly smaller. This algorithm is based on a simple idea, but the effect of solving the noisy TV problem is desirable. For some reinforcement learning algorithms that already have two neural networks, it is very time-consuming to maintain the inference and training of the four neural networks simultaneously; thus, the training speed of RND is considerably reduced.

4.5.2. Using the Influence Map to Generate an Intrinsic Reward

When Ms. Pac-man eats a large number of pellets in the maze, she does not try to explore the pellets in the corners of the maze. This is a problem similar to the sparse reward problem, i.e., the agent always obtains a reward of zero in continuous exploration, which makes it more difficult to optimize the policy of the agent at this time.

We propose the use of influence maps to solve the sparse reward problem. If the external reward obtained is 0, the value in the influence map is used to generate an intrinsic reward for the agent. This can solve the sparse reward problem to a certain extent. Suppose the agent performs action an in the process of exploring the environment, and the external reward obtained is 0; the intrinsic reward is then calculated by Equation (9).

$$r_{intrinsic} = \text{clip}(influence_{next}/50, -1, 1) \times 10 \quad (9)$$

where "clip" is a function that limits the upper and lower bounds of an array, and $influence_{next}$ indicates the influence value of the location reached after performing action a.

5. Experimental Results

5.1. Hyperparameters of the Influence Map

When calculating the influence map, all objects in the map need to be considered, and each object must be allowed to spread influence in the whole map. The initial influence of an object directly affects the influence map generated by a single object. Then, when all influence maps are superimposed, the final influence map is affected. Explanations of the hyperparameters are reported in Table 1.

Table 1. Explanation of the hyperparameters in the influence map.

Hyperparameter	Explanation
Iop (influence of pill)	The initial influence of pills in the map
Iopp (influence of power pill)	The initial influence of power pills in the map
Ioeg (influence of edible ghost)	The initial influence of edible ghost in the map
Iog (influence of ghost)	The initial influence of ghost in the map (negative)
attenuation	Attenuation parameter: the degree of attenuation of influence spreading
spreadDistance	Maximum spread distance: stop spreading after reaching it
directionParam	Distance adjustment parameter α: the increasing degree of influence in the moving direction

To find the most appropriate influence map hyperparameters, in this experiment, we used several hyperparameter search algorithms, including grid search, random search, and genetic algorithm, to find the best combination of super parameters for Miss Pac-Man. For each hyperparameter search algorithm, 100 episodes were played, and the average score was taken to evaluate the performance of this group of hyperparameters. Table 2 shows the

results. The best score of the grid search was 15,378 points, the best score of the random search was 14,842 points, and the best score of the genetic algorithm was 15,828 points. The hyperparameters used to calculate the influence graph with the genetic algorithm are reported in Table 3.

Table 2. Results of the hyperparameter search algorithms.

Hyperparameter Search Algorithm	Score
Grid search	15,378
Random search	14,842
Genetic algorithm	15,828

Table 3. Hyperparameters in the influence map.

Hyperparameter	Value
Iop (influence of pill)	14
Iopp (influence of power pill)	30
Ioeg (influence of edible ghost)	60
Iog (influence of ghost)	−80
attenuation	0.27
spreadDistance	5
directionParam	0.53

5.2. Comparison of Motion Information Representation in the State Space

5.2.1. Hyperparameters and Comparative Experiments

A comparative experiment was conducted in which four state spaces were included, namely a single-frame image, two consecutive frames, four consecutive frames, and an influence map superimposed image. The deep reinforcement learning algorithm used in the comparative experiments was the Ape-x distributed algorithm.

The specific hyperparameter settings are reported in Table 4.

Table 4. Hyperparameters in deep reinforcement learning Ape-x algorithm.

Hyperparameter	Value
Frame skip	4
No-op start	Maximum value is 30
State shape	(84, 84)
Frame stack	4
Action size	4
Batch size	64
Gamma	0.99
N step	5
Replay buffer size	200,000
Episode limit	50,000
Local replay buffer size	200
Number of actors	11
Epsilons of actors	$\epsilon_i = 0.4^{1+\frac{i}{number\ of\ actors-1}*7}, i \in actors$
Actor parameter update frequency	200
Learning rate	0.00025/4
Target network update frequency	2500

Finally, the "frame stack" items were set to differ, and the other hyperparameters were set to be the same to ensure the control variables. The settings of the "frame stack" are presented in Table 5.

Table 5. "Frame stack" in four-group comparison experiments.

State Space	Frame Stack
Influence map superimposed image	2 (influence map + original image)
Four consecutive frames	4 (four consecutive original images)
Two consecutive frames	2 (original image 1 + original image 2)
Single-frame image	1 (original image 1)

5.2.2. Results

All experiments were carried out on a server with a 1080 Ti graphics card, 48 GB memory, and an Intel i7-8700k CPU. Tensorflow was used to build the models, and Tensorboard, a visualization tool that comes with Tensorflow, was used to export the charts. Generally, rewards are used as the judgment index of the model performance of reinforcement learning. In this paper, game score was used as the index to compare the proposed model with other existing models.

Figure 6 presents the training scores of the four image state spaces. These four curves show that the more images that were superimposed, the higher the game score. With one exception, the calculation of the influence map includes the object motion, and the score was therefore similar to that of the four-frame image overlay method. After training a certain number of times, the influence map scored even higher. As shown in Figure 6, at the abscissa of 1.2 M, the four curves from top to bottom are those of the influence map superimposed image, four consecutive frames, two consecutive frames, and single-frame image. The corresponding ordinates of the four curves are presented in Table 6.

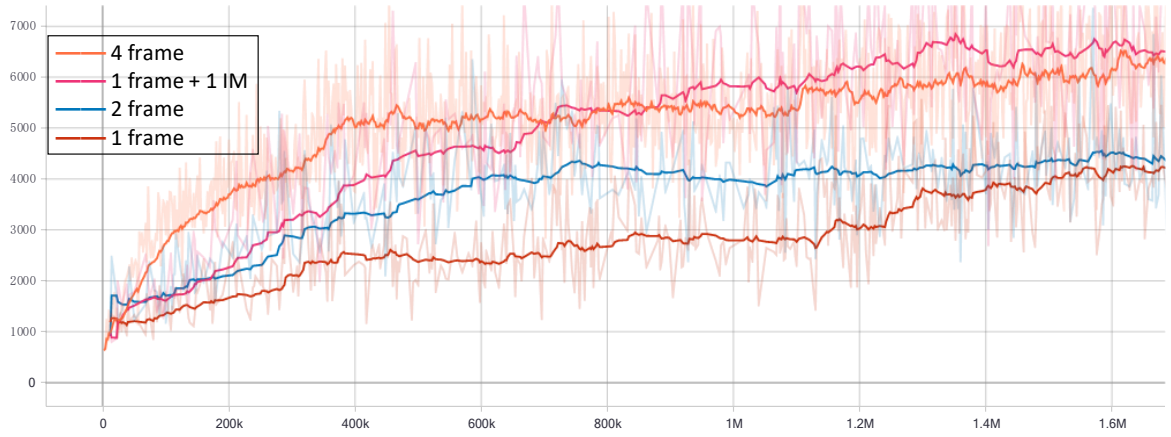

Figure 6. Training scores of four different image state spaces.

Table 6. Ordinate comparison of four training scores at 1.2 M gradient descent.

State Space	Smooth Value	Actual Value	Training Time
Single-frame image	3079	3443	4 h 35 m 1 s
Two consecutive frames	4119	4343	5 h 20 m 56 s
Four consecutive frames	5509	6103	8 h 17 m 20 s
Influence map superimposed image	6201	6840	5 h 10 m 33 s

The four curves shown in Figure 6 all exhibit upward trends; the more frames superimposed, the faster the upward speed. However, when one, two, and four frames are superimposed reach a certain score, the curve almost stops increasing and tends to be flat. The score of the influence map superimposed image continues to increase, showing that the dynamic information carried by the influence map enables agents to learn more environmental knowledge.

Table 7 reports the corresponding ordinates of the four curves at the abscissa of 1.4 M. The comparative experimental results presented in Figure 5 show that the state space of

the influence map superimposed image achieved better performance; the performance was 11.8–12.6% higher than that of the four consecutive frames method. In addition, as shown by the comparison of the use of the two consecutive frames and the influence map superimposed image, the reason for the better performance is not the overlay of consecutive images but the role played by the influence map. The influence map can indicate the trend and focus of the current map. In addition to expressing certain motion information, it also plays a role similar to that of an attention mechanism. The brightness contrast of the influence map may cause the convolution kernel of the CNN to pay more attention to brighter localities, which allows the neural network to take into account the information provided by the influence map when selecting actions.

Table 7. Ordinate comparison of four training scores at 1.4 M gradient descent.

State Space	Smooth Value	Actual Value	Training Time
Single-frame image	3786	4153	5 h 23 m 26 s
Two consecutive frames	4258	3733	6 h 17 m 25 s
Four consecutive frames	5836	4860	9 h 42 m 56 s
Influence map superimposed image	6529	7593	6 h 4 m 52 s

5.2.3. Training Speed

In addition to a certain improvement in performance, Tables 6 and 7 show that the training time of the four consecutive frames was the longest, whereas that of the single-frame image was the shortest.

Figure 7 shows the training speed, revealing that the greater the number of stacked images, the more time the neural network requires for backpropagation, which increases the training time. Using the single frame image, gradient descent could be performed for 71 batches with an average batch size of 64 per second. Using the four consecutive frames, an average of only 40 batches were dealt with per second. The training speed of the other two methods was around 60 batches per second. The reason for the difference is that the input image depth of the neural network is reduced, which reduces the depth of the convolution kernel of the CNN, and merely convolution operations must be performed. When using the influence map superimposed image as the state space, the training speed was nearly 59% faster than that achieved using four consecutive frames, which means that the training time could be reduced by 38%. This is a considerable improvement for practical application. The training speeds of the four methods presented in Figure 7 are basically unchanged throughout the whole process, with a slight decline compared with the beginning because when the experience replay is full, some existing data are overwritten when new data are added. This operation takes slightly longer direct addition.

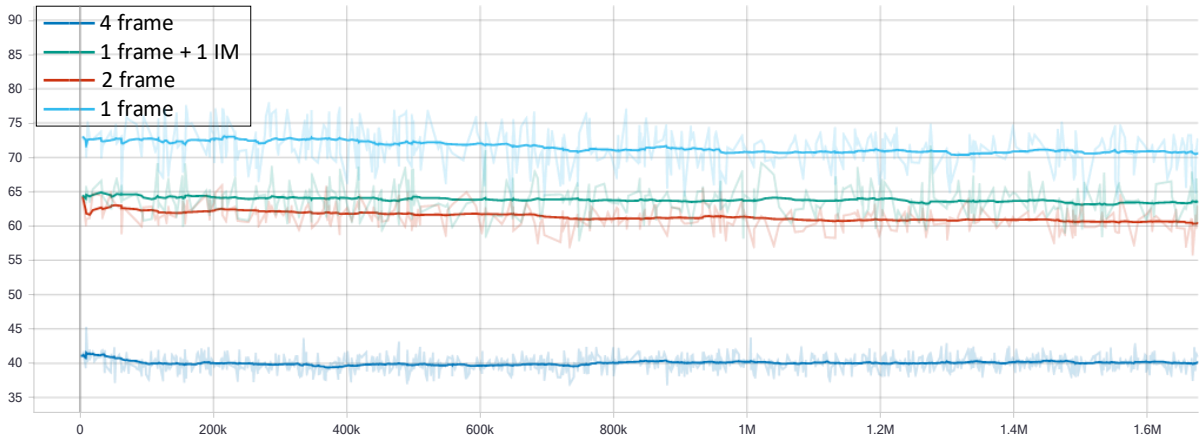

Figure 7. Training speed of four different image state spaces.

5.2.4. Memory and Video Memory Usage

In addition to the improvement mentioned previously, the state space of the influence map superimposed image was found to have two more obvious advantages.

First, compared to the use of four consecutive frames, the memory usage of the influence map superimposed image was directly reduced by half because the depth of the image state was reduced by half. The experience replay pool can store twice as much data as before, and, generally, an increase in the number of experience replays improves the performance of the algorithm [27]. In the comparative experiment, to ensure the control variables, the same experience replay pool was used for all four groups.

Furthermore, because the input shape of the neural network was changed from (84, 84, 4) to (84, 84, 2), the size of the video memory occupied by the influence map superimposed image was also considerably reduced. In other words, for distributed reinforcement learning algorithms, such as Ape-x, more actors can be used to explore the environment and further increase the distribution diversity of the collected data. In this four-group comparative experiment, to ensure the control variables, the same number of actors was used to interact with the environment and collect data for all four groups.

5.3. Comparison with Other Reinforcement Learning Algorithms

Comparative experiments were also conducted with other deep reinforcement learning algorithms, namely DQN [49], Dueling DQN [12], Double DQN [9], N-Step DQN [11], Prioritized Experience Replay [10], C51 [14], QR-DQN [50], IQN [51], and Rainbow [15]. These comparison reinforcement learning algorithms all use four image frames to represent dynamic information. All the algorithms were trained for 1.6 million steps, and the corresponding scores are presented in Figure 8.

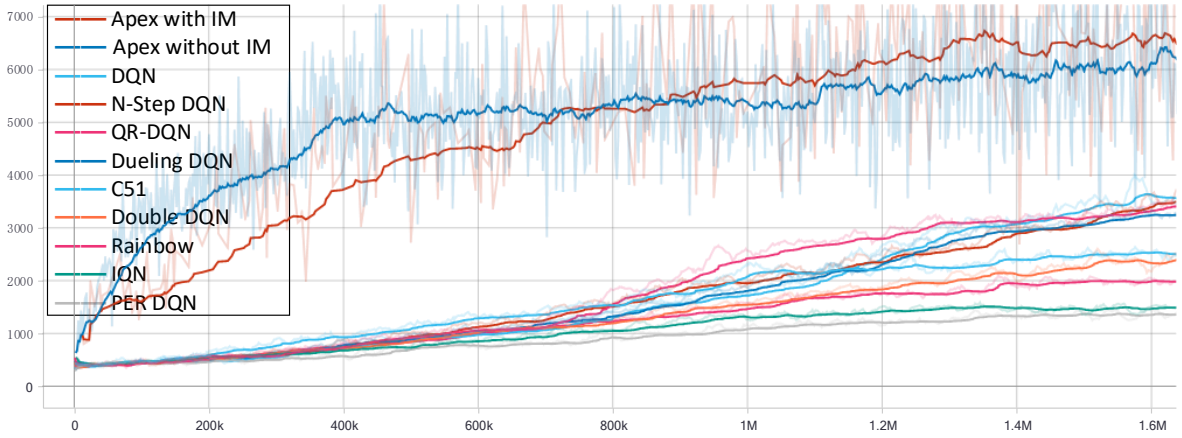

Figure 8. The experimental results achieved in this study compared with those obtained by other reinforcement learning algorithms.

Table 8 reports the ordinates of each algorithm from high to low at the abscissa of 1.6 M. The experimental results show that the Ape-x algorithm, which uses an influence map as the state space, performed the best. Figure 8 shows that the training scores of the Ape-x algorithms with and without an influence map increased rapidly at the beginning, then increased slowly. The curve began to rise sharply because the Ape-x algorithm is a distributed reinforcement learning algorithm; there were multiple agents interacting and collecting data in different environments simultaneously, and the data sampled during training also had a satisfactory distribution. In other words, the Ape-x algorithm is characterized by a high sampling efficiency. Another reason is that the Ape-x algorithm separates training and interaction and uses an agent to train the policy alone. The agent can train the collected data in time and reduce the waste of interactive data. Therefore, the final score of the Ape-x algorithm was far superior to that of the other algorithms. In other words, the agent exhibited a better training effect in each step.

Table 8. Ordinate comparison of the algorithms at 1.6 M gradient descent.

Algorithm	Smooth Value	Actual Value
Ape-x with IM	6645	7947
Vanilla Ape-x	6132	6827
DQN	3661	3490
N-Step DQN	3486	3669
QR-DQN	3369	3446
Dueling DQN	3286	3305
C51	2545	2561
Double DQN	2299	2241
Rainbow	2003	2015
IQN	1470	1519
Prioritized Experience Replay DQN	1363	1445

The top two curves in Figure 8 correspond to the Ape-x algorithms with and without an influence map, respectively, demonstrating that the influence map enabled the Ape-x algorithm to learn more about the environment. At the beginning of training, the state of the influence map returned by the environment could not be effectively understood by the agent. In the image, the score of the Ape-x algorithm without the influence map was slightly higher at first. However, with the continuation of the neural network training process, the agent using the influence map grasped the hidden information of the environment represented by the influence map. Its score exceeded that of the Ape-x algorithm without the influence map, corresponding to the second half of the top two curves in the figure.

Because the influence map contains the current trend, focus, and movement information of the image, the agent using the influence map can obtain additional information. Therefore, the performance of the Ape-x algorithm with the influence map was better than that of the Ape-x algorithm without the influence map. Table 8 indicates that the final score of the Ape-x algorithm with the influence map was far higher than that of the other algorithms.

5.4. Sparse Reward Experiment

According to this intrinsic reward formula, a comparative experiment was implemented using four consecutive frames as the state space. The results are shown in Figure 9. Influence, as an intrinsic motivator, did not lead to a significant difference in the game scores at the beginning of training. However, later in the game, the increased intrinsic incentives caused the decision-making agent to tend to choose actions that had not been chosen before, thereby increasing the randomness of the strategy and the variety of the collected data. This resulted in some gaps between the scores of the two methods late in the game.

As shown in Figure 9, at the abscissa of 1.2 M, the curves from top to bottom are training curves with and without the addition of an intrinsic influence reward. Table 9 presents the ordinates of the two training curves at the 1.2 M abscissa. The use of an intrinsic influence reward slightly reduces the training speed, owing to the calculation of the influence map. Moreover, compared to the situation with no intrinsic reward, the score was found to be increased by about 10%. Figure 9 and Table 9 show that when there was no sparse reward at the beginning, the scores of the two curves were almost the same. After reaching 5000 points, after a large number of pellets had been eaten, the methods were trapped by the sparse reward problem. It is obvious that the performance curve was further improved when using the influence map as an intrinsic reward. Because Ms. Pac-Man is not a purely sparse reward task, if some extremely sparse learning environments are used, such as Go, Wargaming, etc., a greater improvement may be obtained after using the influence map value as an intrinsic reward.

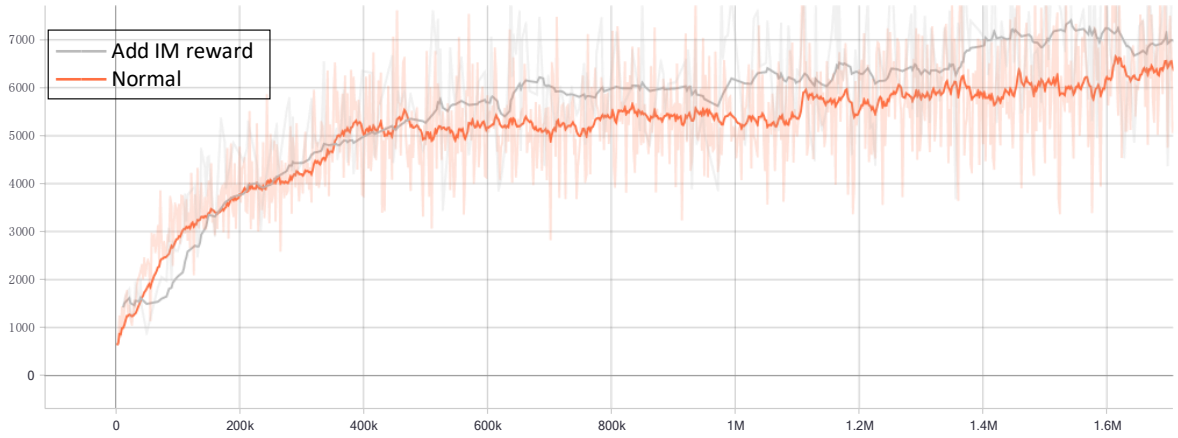

Figure 9. Comparison of the training scores before and after the use of the influence value as an intrinsic reward.

Table 9. Ordinate comparison of two training scores at 1.2 M gradient descent.

Reward Function	Smooth Value	Actual Value	Training Time
Without intrinsic reward	5441	5567	8 h 18 m 59 s
Using influence map value as an intrinsic reward	6339	6450	8 h 45 m 20 s

6. Discussion

Compared with the direct use of four consecutive frames, the influence map cannot be obtained directly from the game environment but must be calculated from the learning environment. The calculation of the influence map requires the consideration of the initial influence value, the spreading mode, the attenuation mode, and the hyperparameters. These calculations add complexity to the learning process. However, if these influence-related attributes can be obtained from the environment, the use of a dynamic influence map in reinforcement learning can improve the performance of almost all reinforcement learning algorithms, reduce the occupancy rate of hardware resources, and increase the training speed.

In recent years, computing technology, especially machine learning, has been used for many applications in engineering [52–54]. An influence map provides additional preconditions for decision making, which can make machine-learning-based decision-making processes more scientific. Therefore, an image-based machine learning model with an influence map can achieve improved engineering performance.

7. Conclusions

In this study, Ms. Pac-Man was used as the learning environment to explore the method of combining an influence map with reinforcement learning. Almost all other deep reinforcement learning algorithms use four consecutive frames of images as the input of the neural network. In this study, a raw frame of an image and the influence map were used for superimposition. The performance achieved using a influence map superimposed image was about 11.8–12.6% higher than that achieved using four consecutive frames. Compared with the use of four consecutive frames as the state space, the training speed of the proposed method was increased by 59%, the video memory usage was reduced by 30%, and the memory used by the experience replay was reduced by 50%. The results prove the feasibility of the use of a dynamic influence map in deep reinforcement learning algorithms. With respect to the sparse reward tasks in deep reinforcement learning, the influence map value was used to generate an intrinsic reward when there was no external reward. Even in the case of Ms. Pac-Man, which is not a completely sparse reward task, the influence map improved the score by about 10%.

At present, the strategy learned by the algorithm does not contain enough long-term goals. To allow the agent to have long-term memory, a recurrent neural network, such as long short-term memory (LSTM), can be added to the Ape-x deep reinforcement learning algorithm to further improve the decision-making intelligence of the agent. In the future, influence maps will be combined with other reinforcement learning algorithms to study the universality of influence maps.

Author Contributions: Conceptualization, X.L. and A.X.; methodology, X.L. and A.X.; software, A.X.; validation, X.L. and A.X.; formal analysis, X.L. and A.X.; investigation, X.L. and A.X.; resources, A.X.; data curation, A.X.; writing—original draft preparation, X.L. and A.X.; writing—review and editing, X.L., P.L. and P.H.; visualization, A.X.; supervision, X.L., P.L. and P.H.; project administration, X.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Key R&D Program of China (Grant No. 2020YFB2104700), and the National Natural Science Foundation of China (grant No. 62136006).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 2018.
2. 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.
3. Rohlfshagen, P.; Liu, J.; Perez-Liebana, D.; Lucas, S.M. Pac-man conquers academia: Two decades of research using a classic arcade game. *IEEE Trans. Games* **2017**, *10*, 233–256. [CrossRef]
4. Fitzgerald, A.; Congdon, C.B. RAMP: A Rule-Based Agent for Ms. Pac-Man. In Proceedings of the 2009 IEEE Congress on Evolutionary Computation (CEC 2009), Trondheim, Norway, 18–21 May 2009; pp. 2646–2653.
5. Samothrakis, S.; Robles, D.; Lucas, S. Fast approximate max-n monte carlo tree search for ms pac-man. *IEEE Trans. Comput. Intell. AI Games* **2011**, *3*, 142–154. [CrossRef]
6. Alhejali, A.M.; Lucas, S.M. Evolving Diverse Ms. Pac-Man Playing Agents Using Genetic Programming. In Proceedings of the 2010 UK Workshop on Computational Intelligence (UKCI 2010), Colchester, UK, 8–10 September 2010.
7. Yuan, B.; Li, C.; Chen, W. Training a Pac-Man Player with Minimum Domain Knowledge and Basic Rationality. In Proceedings of the 6th International Conference on Intelligent Computing (ICIC 2010), Changsha, China, 18–21 August 2010; Volume 93 CCIS, pp. 169–177.
8. Van Seijen, H.; Fatemi, M.; Romoff, J.; Laroche, R.; Barnes, T.; Tsang, J. Hybrid reward architecture for reinforcement learning. *arXiv* **2017**, arXiv:1706.04208.
9. Van Hasselt, H.; Guez, A.; Silver, D. Deep Reinforcement Learning with Double Q-Learning. In Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI 2016), Phoenix, AZ, USA, 12–17 February 2016; pp. 2094–2100.
10. Schaul, T.; Quan, J.; Antonoglou, I.; Silver, D. Prioritized experience replay. *arXiv* **2015**, arXiv:1511.05952.
11. Hernandez-Garcia, J.F.; Sutton, R.S. Understanding multi-step deep reinforcement learning: A systematic study of the DQN target. *arXiv* **2019**, arXiv:1901.07510.
12. Wang, Z.; Schaul, T.; Hessel, M.; Van Hasselt, H.; Lanctot, M.; De Frcitas, N. Dueling Network Architectures for Deep Reinforcement Learning. In Proceedings of the 33rd International Conference on Machine Learning (ICML 2016), New York, NY, USA, 19–24 June 2016; Volume 4, pp. 2939–2947.
13. Fortunato, M.; Azar, M.G.; Piot, B.; Menick, J.; Osband, I.; Graves, A.; Mnih, V.; Munos, R.; Hassabis, D.; Pietquin, O. Noisy networks for exploration. *arXiv* **2017**, arXiv:1706.10295.
14. Bellemare, M.G.; Dabney, W.; Munos, R. A Distributional Perspective on Reinforcement Learning. In Proceedings of the 34th International Conference on Machine Learning (ICML 2017), Sydney, NSW, Australia, 6–11 August 2017; Volume 1, pp. 693–711.
15. Hessel, M.; Modayil, J.; Van Hasselt, H.; Schaul, T.; Ostrovski, G.; Dabney, W.; Horgan, D.; Piot, B.; Azar, M.; Silver, D. Rainbow: Combining improvements in deep reinforcement learning. In Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI 2018), New Orleans, LA, USA, 2–7 February 2018; pp. 3215–3222.
16. Williams, R.J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.* **1992**, *8*, 229–256. [CrossRef]
17. Konda, V.R.; Tsitsiklis, J.N. Actor-critic algorithms. In Proceedings of the 13th Annual Neural Information Processing Systems Conference (NIPS 1999), Denver, CO, USA, 29 November–4 December 1999; pp. 1008–1014.

18. Mnih, V.; Badia, A.P.; Mirza, L.; Graves, A.; Harley, T.; Lillicrap, T.P.; Silver, D.; Kavukcuoglu, K. Asynchronous Methods for Deep Reinforcement Learning. In Proceedings of the 33rd International Conference on Machine Learning (ICML 2016), New York, NY, USA, 19–24 June 2016; Volume 4, pp. 2850–2869.
19. Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; Moritz, P. Trust region policy optimization. In Proceedings of the 32nd International Conference on Machine Learning (ICML 2015), Lile, France, 6–11 July 2015; Volume 3, pp. 1889–1897.
20. Wu, Y.; Mansimov, E.; Liao, S.; Grosse, R.; Ba, J. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. *arXiv* **2017**, arXiv:1708.05144.
21. Wang, Z.; Bapst, V.; Heess, N.; Mnih, V.; Munos, R.; Kavukcuoglu, K.; de Freitas, N. Sample efficient actor-critic with experience replay. *arXiv* **2016**, arXiv:1611.01224.
22. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347.
23. Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D.; Riedmiller, M. Deterministic policy gradient algorithms. In Proceedings of the 31st International Conference on Machine Learning (ICML 2014), Beijing, China, 21–26 June 2014; Volume 1, pp. 605–619.
24. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous control with deep reinforcement learning. *arXiv* **2015**, arXiv:1509.02971.
25. Fujimoto, S.; Van Hoof, H.; Meger, D. Addressing Function Approximation Error in Actor-Critic Methods. In Proceedings of the 35th International Conference on Machine Learning (ICML 2018), Stockholm, Sweden, 10–15 July 2018; Volume 4, pp. 2587–2601.
26. Haarnoja, T.; Zhou, A.; Abbeel, P.; Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Proceedings of the 35th International Conference on Machine Learning (ICML 2018), Stockholm, Sweden, 10–15 July 2018; Volume 5, pp. 2976–2989.
27. Horgan, D.; Quan, J.; Budden, D.; Barth-Maron, G.; Hessel, M.; Van Hasselt, H.; Silver, D. Distributed prioritized experience replay. *arXiv* **2018**, arXiv:1803.00933.
28. Barth-Maron, G.; Hoffman, M.W.; Budden, D.; Dabney, W.; Horgan, D.; Tb, D.; Muldal, A.; Heess, N.; Lillicrap, T. Distributed distributional deterministic policy gradients. *arXiv* **2018**, arXiv:1804.08617.
29. Espeholt, L.; Soyer, H.; Munos, R.; Simonyan, K.; Mnih, V.; Ward, T.; Doron, Y.; Firoiu, V.; Harley, T.; Dunning, I.; et al. IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. In Proceedings of the 35th International Conference on Machine Learning (ICML 2018), Stockholm, Sweden, 10–15 July 2018; pp. 1407–1416.
30. Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv* **2017**, arXiv:1712.01815.
31. Berner, C.; Brockman, G.; Chan, B.; Cheung, V.; Dębiak, P.; Dennison, C.; Farhi, D.; Fischer, Q.; Hashme, S.; Hesse, C. Dota 2 with large scale deep reinforcement learning. *arXiv* **2019**, arXiv:1912.06680.
32. Wirth, N.; Gallagher, M. An influence map model for playing Ms. Pac-Man. In Proceedings of the 2008 IEEE Symposium on Computational Intelligence and Games (CIG 2008), Perth, WA, Australia, 15–18 December 2008; pp. 228–233.
33. Jang, S.-H.; Cho, S.-B. Evolving neural NPCs with layered influence map in the real-time simulation game 'Conqueror'. In Proceedings of the 2008 IEEE Symposium on Computational Intelligence and Games, Perth, WA, Australia, 15–18 December 2008; pp. 385–388.
34. Danielsiek, H.; Stuer, R.; Thom, A.; Beume, N.; Naujoks, B.; Preuss, M. Intelligent moving of groups in real-time strategy games. In Proceedings of the 2008 IEEE Symposium on Computational Intelligence and Games, Perth, WA, Australia, 15–18 December 2008; pp. 71–78.
35. Park, H.; Kim, K.-J. Mcts with influence map for general video game playing. In Proceedings of the 2015 IEEE Conference on Computational Intelligence and Games (CIG), Tainan, Taiwan, 31 August–2 September 2015; pp. 534–535.
36. Jong, D.; Kwon, I.; Goo, D.; Lee, D. Safe Pathfinding Using Abstract Hierarchical Graph and Influence Map. In Proceedings of the 2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI), Vietri sul Mare, Italy, 9–11 November 2015; pp. 860–865.
37. Krontiris, A.; Bekris, K.E.; Kapadia, M. Acumen: Activity-centric crowd authoring using influence maps. In Proceedings of the 29th International Conference on Computer Animation and Social Agents, Geneva, Switzerland, 23–25 May 2016; pp. 61–69.
38. Avery, P.; Louis, S. Coevolving influence maps for spatial team tactics in a RTS game. In Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, Portland, OR, USA, 7–11 July 2010; pp. 783–790.
39. Miles, C.; Quiroz, J.; Leigh, R.; Louis, S.J. Co-evolving influence map tree based strategy game players. In Proceedings of the 2007 IEEE Symposium on Computational Intelligence and Games, Honolulu, HI, USA, 1–5 April 2007; pp. 88–95.
40. Svensson, J.; Johansson, S.J. Influence Map-based controllers for Ms. PacMan and the ghosts. In Proceedings of the 2012 IEEE Conference on Computational Intelligence and Games (CIG), Granada, Spain, 11–14 September 2012; pp. 257–264.
41. Lu, X.; Wang, X. A Dynamic Influence Map Model Based on Distance Adjustment. In Proceedings of the 2018 IEEE 3rd International Conference on Communication and Information Systems (ICCIS), Singapore, Singapore, 28–30 December 2018; pp. 183–187.
42. Cho, K.; Sung, Y.; Um, K. A Production Technique for a Q-table with an Influence Map for Speeding up Q-learning. In Proceedings of the 2007 International Conference on Intelligent Pervasive Computing (IPC 2007), Jeju Island, Korea, 11–13 October 2007; pp. 72–75.

43. Heckel, F.W.; Youngblood, G.M.; Hale, D.H. Influence points for tactical information in navigation meshes. In Proceedings of the 4th International Conference on Foundations of Digital Games, Orlando, FL, USA, 26–30 April 2009; pp. 79–85.
44. Siji George, C.G.; Sumathi, B. Grid Search Tuning of Hyperparameters in Random Forest Classifier for Customer Feedback Sentiment Prediction. *Int. J. Adv. Comput. Sci. Appl.* **2020**, *11*, 173–178. [CrossRef]
45. Bergstra, J.; Bengio, Y. Random Search for Hyper-Parameter Optimization. *J. Mach. Learn. Res.* **2012**, *13*, 281–305.
46. Langazane, S.N.; Saha, A.K. Effects of Particle Swarm Optimization and Genetic Algorithm Control Parameters on Overcurrent Relay Selectivity and Speed. *IEEE Access* **2022**, *10*, 4550–4567. [CrossRef]
47. Pathak, D.; Agrawal, P.; Efros, A.A.; Darrell, T. Curiosity-driven exploration by self-supervised prediction. In Proceedings of the 30th IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW 2017), Honolulu, HI, USA, 21–26 July 2017; Volume 2017, pp. 488–489.
48. Burda, Y.; Edwards, H.; Storkey, A.; Klimov, O. Exploration by random network distillation. *arXiv* **2018**, arXiv:1810.12894.
49. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef] [PubMed]
50. Dabney, W.; Rowland, M.; Bellemare, M.G.; Munos, R. Distributional reinforcement learning with quantile regression. In Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI 2018), New Orleans, LA, USA, 2–7 February 2018; pp. 2892–2901.
51. Dabney, W.; Ostrovski, G.; Silver, D.; Munos, R. Implicit quantile networks for distributional reinforcement learning. In Proceedings of the 35th International Conference on Machine Learning (ICML 2018), Stockholm, Sweden, 10–15 July 2018; Volume 3, pp. 1774–1787.
52. Mousavi, N.S.; Vaferi, B.; Romero-Martinez, A. Prediction of surface tension of various aqueous amine solutions using the UNIFAC model and artificial neural networks. *Ind. Eng. Chem. Res.* **2021**, *60*, 10354–10364. [CrossRef]
53. Alanazi, A.K.; Alizadeh, S.M.; Nurgalieva, K.S.; Nesic, S.; Grimaldo Guerrero, J.W.; Abo-Dief, H.M.; Eftekhari-Zadeh, E.; Nazemi, E.; Narozhnyy, I.M. Application of Neural Network and Time-Domain Feature Extraction Techniques for Determining Volumetric Percentages and the Type of Two Phase Flow Regimes Independent of Scale Layer Thickness. *Appl. Sci.* **2022**, *12*, 1336. [CrossRef]
54. Zhou, Z.; Davoudi, E.; Vaferi, B. Monitoring the effect of surface functionalization on the CO_2 capture by graphene oxide/methyl diethanolamine nanofluids. *J. Environ. Chem. Eng.* **2021**, *9*, 106202. [CrossRef]