mathematics

Article

Improving Agent Decision Payoffs via a New Framework of Opponent Modeling

Chanjuan Liu [1], Jinmiao Cong [1], Tianhao Zhao [1] and Enqiang Zhu [2,*]

[1] School of Computer Science and Technology, Dalian University of Technology, Dalian 116024, China; chanjuanliu@dlut.edu.cn (C.L.); cjm111@mail.dlut.edu.cn (J.C.)
[2] Institute of Computing Science and Technology, Guangzhou University, Guangzhou 510006, China
* Correspondence: zhuenqiang@gzhu.edu.cn

Abstract: The payoff of an agent depends on both the environment and the actions of other agents. Thus, the ability to model and predict the strategies and behaviors of other agents in an interactive decision-making scenario is one of the core functionalities in intelligent systems. State-of-the-art methods for opponent modeling mainly use an explicit model of opponents' actions, preferences, targets, etc., that the primary agent uses to make decisions. It is more important for an agent to increase its payoff than to accurately predict opponents' behavior. Therefore, we propose a framework synchronizing the opponent modeling and decision making of the primary agent by incorporating opponent modeling into reinforcement learning. For interactive decisions, the payoff depends not only on the behavioral characteristics of the opponent but also the current state. However, confounding the two obscures the effects of state and action, which then cannot be accurately encoded. To this end, state evaluation is separated from action evaluation in our model. The experimental results from two game environments, a simulated soccer game and a real game called quiz bowl, show that the introduction of opponent modeling can effectively improve decision payoffs. In addition, the proposed framework for opponent modeling outperforms benchmark models.

Keywords: computational intelligence; opponent modeling; deep neural networks; reinforcement learning; interactive decision making

MSC: 68T42; 91A25

check for updates

Citation: Liu, C.; Cong, J.; Zhao, T.; Zhu, E. Improving Agent Decision Payoffs via a New Framework of Opponent Modeling. *Mathematics* **2023**, *11*, 3062. https://doi.org/10.3390/math11143062

Academic Editor: Hsien-Chung Wu

Received: 29 May 2023
Revised: 19 June 2023
Accepted: 1 July 2023
Published: 11 July 2023

1. Introduction

Decision intelligence, i.e., the ability to make appropriate action choices and decisions in various decision environments, is one of the core functionalities in artificial intelligent, such as intelligent transportation and warehouse robots. Studies on decision intelligence [1] have attracted increasing attention since the successful implementation of the AI Go player Alphago [2].

An important issue for agents working in a strategic setting (on either collaborative or competitive tasks) is to predict the behaviors or infer the intentions of other agents [3–5] because all active agents can affect the world state [6]; thus, the payoff of an agent depends on both the environment and actions of other agents. Therefore, strategic reasoning [7,8] based on the characteristics of other agents is very important for the effective decision making of an agent. Some behavioral characteristics or strategy biases are often exposed when agents participate in an interaction. An agent can exploit these characteristics by modeling other agents to make better interactive decisions. For example, an agent can make use of suboptimal agents in a multiplayer game by predicting poor moves, and a financial trading agent that knows the intentions of its opponents can reach an agreement faster [9].

Challenge. This method of modeling other agents involved in the interaction is called *opponent modeling* [10–12]. The classical approach consists of establishing an opponent model based on interaction histories [13,14]. The model can be regarded as a function that takes the interaction history as input and outputs a prediction about the opponents.

Existing opponent modeling methods [12,15] focus mainly on modeling the actions, types, preferences, and beliefs of opponents. In action modeling [16–18], an opponent's future actions are predicted by explicitly reconstructing the opponent's decision-making method, such as the action frequency. In 1995, ref. [19] proposed modeling the strategies of agents as finite automata for multiagent system modeling. Opponent preference modeling [20–22] is used to predict and analyze an opponent's strategy by considering the impetus for the opponent's strategy, including the opponent's intention, payoff function, target plan, etc. The authors of [23] proposed a regularized opponent model with a maximum entropy objective. Belief modeling simulates the reasoning process of other agents to predict their strategies. For example, ref. [24] proposed a level-k thinking scheme for adversarial reinforcement learning. Reference [25] developed a recursive rule that allows for explicit updates of the belief over the opponent's hidden types, which led to a substantial enhancement in the quality of decision making. In opponent-type modeling [26,27], classification is used to predict the opponent's behavioral style. For example, ref. [28] modeled the opponent type for real-time strategy games based on the use of a hierarchical structure.

In an adaptive system [29], an agent is constantly learning and making dynamic adaptive adjustments. An opponent model should be constructed to enable adaptive changes to be made based on the opponent's real-time strategy. Methods have been proposed that realize dynamic opponent modeling to varying degrees. For example, the authors of [30] required periodic selection in a variety of fixed strategies. However, the aim of these methods is to establish an explicit opponent model, that is, a model (such as a decision tree, neural network, or Bayesian model) is constructed to directly predict the parameters of opponents, such as actions or domain-specific strategies.

Although considerable progress has been made in this research direction, opponent prediction is independent of the decision-making process in these models, which makes it difficult to coordinate the processes of modeling and planning. Moreover, it is more important for an agent to increase its payoff than for opponent characteristics to be accurately predicted. To consider dynamic opponent strategies and build an agent that can dynamically make decisions to increase payoffs based on opponent behavior, opponent prediction should be combined with the decision-making process of the agent.

Contribution. Therefore, we propose a framework synchronizing opponent modeling and agent decision making by designing two Q-network models that exhibit strong generalization capabilities in reinforcement learning. Note that for interactive decisions, the payoff depends on not only the behavioral characteristics of the opponent but also the current state. Thus, it is necessary to encode the state and action at the same time. Confounding the state and action obscures the effects of state and action, which then cannot be accurately encoded, which is a common issue found in most explicit modeling approaches. For example, the inaccuracy of explicit opponent modeling employed by [31] leads to interference in the decision-making process of the agents. To this end, in our designed neural network, state evaluation is first separated from action evaluation, and then, the state and action values are integrated, which mitigates the issue of error accumulation. As a specific opponent model is not required, the proposed framework can be generalized and adapted to various interactive decision-making scenarios. We also propose a training algorithm for such opponent networks. By introducing two separate networks, one for action selection and another for estimating the Q-values associated with those actions, we reduce the problem of overestimation in Q-value estimation.

Related literature. This study was inspired by the recent success of applying opponent modeling to deep reinforcement learning [32–35], using a neural-network-based model, called a deep reinforced opponent network (DRON). DRON is a modification of DQN [36], where one network is used to evaluate Q-values and another network is responsible for modeling the opponent. The opponent's actions are used as inputs to capture the current opponent's characteristics, which are then utilized for learning the opponent's strategy. Both the model and the training algorithm that we use in this study are different from DRON, although the target is similar. In the proposed model, the analysis of the current state is the premise of predicting the opponents' actions. Therefore, within the proposed framework, the network structure contains a separate module for evaluating the decision state. Within our approach, an appropriate learning algorithm for the opponent network is used to objectively evaluate the actions of opponents, which critically affects the quality of decisions when dynamic opponents are involved.

Experiments were conducted demonstrating that the introduction of opponent modeling can effectively improve decision payoffs, and the proposed opponent model outperformed the DRON and deep Q-network (DQN) benchmark models.

Paper structure. The rest of this article is structured as follows. The fundamentals of deep reinforcement learning without opponent modeling are introduced in Section 2. Our opponent model, including the network structure and training algorithm, is introduced in Sections 3 and 4. The experimental results are comparatively analyzed in Section 5. In Section 6, the findings of the study are summarized, and future research directions are identified.

2. Preliminaries

2.1. Reinforcement Learning

In reinforcement learning [37], an agent observes its environment, takes actions, receives rewards, and thereby learns to act in a manner that maximizes rewards [34,38].

A reinforcement learning task is typically modeled as a Markov decision process (MDP) [38]. An MDP is a tuple $\langle S, A, P, R, \gamma \rangle$, where S is a finite set of possible states, A is a collection of actions, $P(s, a, s')$ is the probability of the environment transitioning from state s to state s' after the agent takes action a, $R : S \times A \to \mathbb{R}$ is the reward function indicating the reward obtained when the agent takes action a in state s, and γ is the *discount factor*.

A policy is defined as the probability distribution $\pi(a|s)$ for performing an action a in state s. The action-value function $q_\pi(s, a)$ represents the expected return from adopting the policy π. The optimal policy π^* is defined by the Bellman equation [39], shown as Formula (1):

$$q^*(s,a) = \sum_{s' \in S} P(s,a,s')(R(s,a,s') + \gamma \max_{a' \in A} q^*(s',a')) \tag{1}$$

2.2. Q-Learning

The temporal difference (TD) algorithm is a model-free reinforcement learning algorithm. Q-learning is a typical TD algorithm that evaluates the results of an action performed in the current state s_t and the reward R_t while observing the future state s_{t+1}. Based on the evaluation of all behaviors in all states, the overall optimal behavior is learned by judging the long-term discounted returns. The iterative formula [40] of the action-state value function is shown as Formula (2):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[U_t - Q(s_t, a_t)] \tag{2}$$

In Formula (2), $U_t = R_t + \gamma \max_{a \in A} Q(s_{t+1}, a)$, α is the learning rate and γ is the discount factor.

2.3. DQN

For complex problems with continuous states, it is difficult to express the action-value function as a lookup table; thus, a continuous approximation is required. The DQN algorithm, a modern variation on Q-learning in which a deep neural network is used to approximate the Q-function, was proposed in 2015 [36].

The basic DQN algorithm is described below.

(1) A deep neural network is used to approximate the action-value function using a parameter ω given by $Q(s, a, \omega) \approx q_\pi(s, a)$.

(2) As shown in Formula (3), the mean square error (MSE) is used as the loss function in the approximate network for the action-value function. Two unrelated Q-networks (Q_network_Local and Q_network_Target) are used to calculate the predicted value (weight ω) and the target value (weight ω'). After several steps, the target network is frozen, and the weight of the actual Q-network is copied to the target network.

$$L(\omega) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a'; \omega') - Q(s, a; \omega)\right)^2\right] \tag{3}$$

In this formula, s' and a' represent the subsequent state and action, respectively.

(3) As shown in Formula (4), the gradient is calculated to update ω:

$$\frac{\partial L(\omega)}{\partial \omega} = E[(r + \gamma \max_{a'} Q(s', a', \omega') - Q(s, a, \omega)\frac{\partial Q(s, a, \omega)}{\partial \omega}] \tag{4}$$

Repeating the above steps, DQN continuously improves the discrepancy between its predicted Q-values and the actual target Q-values, while gradually adjusting the parameters of the neural network to approximate the optimal Q-function. In summary, DQN has made significant contributions to the advancement of reinforcement learning by combining deep learning techniques with the Q-learning framework, enabling agents to learn approximate optimal policies in complex environments [41–43].

3. Opponent Modeling Based on Deep Reinforcement Learning

In decision-making scenarios, an agent participating in an interaction uses a strategy to improve its payoffs, which is more important than modeling the opponent accurately. Therefore, unlike traditional decision making based on an explicit opponent model, we propose an implicit model that coordinates opponent modeling and action-planning by integrating opponent modeling into the reinforcement learning process.

3.1. Decision Process with Opponent Modeling

In a multiagent decision-making setting, the actions of all agents affect the environment. From an agent's point of view, the result of an operation does not remain stable in a given state but depends on the operations of other agents.

We use a to represent the actions of agents we control (primary agents) and o to represent the joint actions of all other agents (auxiliary agents). Then, a joint action (a, o) by all agents is composed of the actions of the primary agent and opponents. The state transition probability becomes $P(s'|s, a, o)= Pr(S_{t+1} = s'|S_t{=}s,\ A_t{=}(a, o))$, which means the probability of the environment state transitions from s to s' after primary agents and auxiliary agents perform a joint action (a, o). The reward function becomes $R(s, a, o)= E[R_{t+1}|S_t{=}\ s,\ A_t{=}(a, o)]$, in which E represents the mathematical expectation. $R(s, a, o)$ outputs the reward when the environment state is s and primary agents and auxiliary agents perform a joint action (a, o).

In the multiagent interaction environment, the goal of the primary agent is to learn an optimal strategy in response to the joint strategy π^o of the opponents. The opponents can be regarded as part of the environment, and the optimal action-value function relative to the joint strategy of the opponents can be redefined as $q_{*|\pi^o}{=}\max\limits_{\pi} q_{\pi|\pi^o}(s, a)$, where $s \in S$ and $a \in A$. The recursive expression of the action-value function becomes Formula (5):

$$q_{\pi|\pi^o}(s_t, a_t) = \sum_{o_t} \pi^o_t(o_t|s_t) \sum_{s_{t+1}} P(s_t, a_t, o_t, s_{t+1})[R(s_t, a_t, o_t) + \gamma E_{a_{t+1}}[q_{\pi|\pi^o}(s_{t+1}, a_{t+1})]] \tag{5}$$

In Formula (5), the first summation sums over the possible values of the opponent's behavior o_t, weighted by the opponent policy $\pi^o_t(o_t|s_t)$ to consider the probability of opponent actions; the second summation sums over the possible values of the next state

s_{t+1}, weighted by the transition probability $P(s_t, a_t, o_t, s_{t+1})$ to consider the probability of transitioning from the current state through the action and opponent behavior to the next state. Within the summation, there is a reward function $R(s_t, a_t, o_t)$ that evaluates the immediate reward of taking action a_t in the current state and facing the opponent taking action o_t. Additionally, the equation includes an expectation operation $E_{a_{t+1}}$, which represents the expectation over the action a_{t+1} in the next time step. This expectation is taken with respect to the state-action value function $q_{\pi|\pi^o}(s_{t+1}, a_{t+1})$ in the next state s_{t+1}, based on the current policy π and opponent policy π^o. Overall, this equation computes the state-action value function based on the current state and action, capturing the long-term cumulative rewards under the given policy and opponent policy.

3.2. Deep Reinforcement Network with Opponent Modeling

To coordinate opponent modeling and action-planning and thus optimize agent decision making, we designed two modules for the deep reinforcement network: an action-planning network and an opponent network. Moreover, we constructed two types of Q-networks, namely, DualFc2 (Dual and Fully Connected 2 Network) and DualMoe (Dual and Mixture of Experts Network), which are derived through distinct methods of integrating the two modules mentioned above.

Figure 1 presents the structure of the DualFc2 network, which is composed of the action-planning network and the opponent network through a full-connection layer. The state information F_s is input to the input layer I_s in the action-planning network, and the opponent information F_o is input to the input layer I_o in the opponent network. The vectors from the two input layers I_s and I_o are integrated in the first full-connection hidden layer H_{so} and then processed by the second full-connection hidden layer H_{va}.

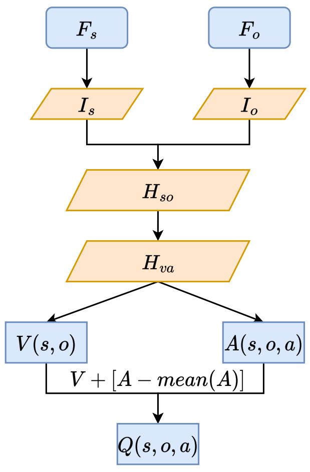


Figure 1. Structure of the DualFc2 network.

Note that in interactive decision making, the agent's payoff depends on both the relevant characteristics of the opponent's behavior and the current state. Therefore, it is necessary to encode the state and action simultaneously. Confounding the state and action models in the network obscures the effects of state and action, which then cannot be modeled accurately. To this end, in our novel scheme, state evaluation is first separated from action evaluation, and the state and action values are then integrated. That is, H_{va} outputs the state value $V(s, o)$ and the action-advantage value $A(s, o, a)$, respectively. These two values are integrated using the VPA operation [44] defined in Formula (6):

$$Q(s,o,a) = V(s,o) + (A(s,o,a) - \frac{1}{|\mathcal{A}|}\sum_{a' \in \mathcal{A}} A(s,o,a')) \tag{6}$$

In the VPA operation, the state value is expanded to have the same dimension as the action-advantage value, and each component of the action-advantage value is subtracted from the mean value of all the components. Thus, when the action-advantage value is almost the same as the state value, the final Q value $Q(s, o, a)$ is more influenced by the state value network. The final Q value is obtained by adding the state and action-advantage values together.

Figure 2 shows the DualMoe network, which is composed of two parts. One part is the expert network (experts): opponent information is input, processed through the rectified linear unit (ReLU) activation function in the input layer I_s, and then outputted to a specific number of expert subnetworks (expert). Each expert captures one type of opponent strategy via a dual subnetwork, just like the state-value network and the action-advantage-value network in the DualFc2 network. The other part of the DualMoe network is the gater network. The gater network receives decision information and opponent information. This information is transmitted as a vector to the input layer of the gater I_{so} and then enters the hidden layer H_o. The output is weight information representing the probability of each expert. The hidden layer uses the Softmax() activation function to complete the normalization process. The Softmax() function first exponentiates the input data and then normalizes them, ensuring that all data fall within the range of 0 to 1. It transforms the output of a neural network into a probability distribution, enabling gradient propagation and facilitating the algorithm's effective update of the network parameters, thereby improving the performance of the model. After the hidden layer H_o, the gater outputs the weight vector W followed by the weighted sum of the V and A values given by the expert network according to the formulas $v(s, o) = \sum_{i=1}^{k} w_i \times v_i(s, o)$ and $A(s, o, a) = \sum_{i=1}^{k} w_i \times A_i(s, o, a)$, respectively. The final value Q is calculated by the same method used in the DualFc2 network.

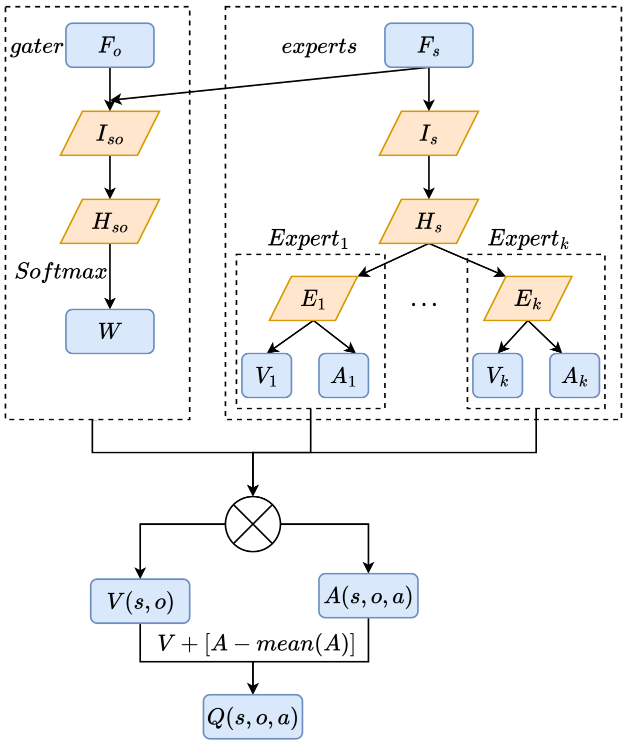


Figure 2. Structure of the DualMoe network.

4. Reinforcement Learning Algorithm for the Opponent Network

A reinforcement learning algorithm for the opponent network is used to learn the optimal strategy under interactive decision scenarios. DQN [36] is one of the most popular reinforcement learning algorithms available. However, learning in DQN does not involve opponent information, and the same action value is used for both action selection and evaluation. At each iteration, the maximum action value Q is selected in the target network according to the state s, shown as Formula (7). Although this greedy strategy can cause Q to quickly approach the possible optimization goal, overestimated values are likely to be selected, resulting in the imprecise estimation of actions.

$$y_t = r_{t+1} + \gamma \max_a Q\left(s_{t+1}, a; \theta_t^-\right) \tag{7}$$

To make stable decisions with dynamic opponents, we propose an algorithm (Algorithm 1) called DecoupleRlO (decoupled reinforcement learning for an opponent network to train the DualFc2 and DualMoe networks), inspired by the concept of DoubleDqn. However, unlike in DoubleDqn [45], the action of a player is combined with the actions of the opponents to produce an action profile, such that the opponent's information is implicitly taken into account in the action evaluation. The selection and evaluation of actions are also decoupled; i.e., the max operation is carried out to select the optimal action, and the action value is then evaluated by the target network, shown as Formula (8). The actions of the agents are thereby objectively evaluated.

$$y_t = r_{t+1} + \gamma Q\left(s_{t+1}, o, \underset{a}{\operatorname{argmax}} Q(s_{t+1}, o, a; \theta_t); \theta_t^-\right) \tag{8}$$

Algorithm 1 DecoupleRlO

```
1:  initialize replay memory $D$ to capacity $N$;
2:  initialize $Q$ network with random weights $\theta$;
3:  initialize target network with weights $\theta^- = \theta$;
4:  for i_episode in range(1,max_episode) do
5:      reset initial state $s$;
6:      for i_step in range(0,max_step_limit) do
7:          with probability $\epsilon$ select a random action $a_t$ or select $a_t = \arg\max(Q(s_t, o_t, a; \theta))$;
8:          carry out action $a_t$, observe reward $r_t$ and new state $s_{t+1}$;
9:          store experience $\langle s_t, o_t, a_t, r_t, s_{t+1} \rangle$ in replay memory $D$;
10:         sample random minibatch of transitions $\langle s_j, o_j, a_j, r_j, s_{j+1} \rangle$ from replay memory $D$;
11:         set $y_j = r_j$ if episode terminates at step $j+1$;
12:         otherwise set:
            $y_j = r_j + \gamma Q(s_{j+1}, o_{j+1}, \arg\max_a Q(s_{j+1}, o_{j+1}, a; \theta_t); \theta_t^-)$;
13:         perform a gradient decent step on $(y_j - Q(s_j, o_j, a_j; \theta)^2$ with respect to $\theta$;
14:         every C steps reset $\theta^- = \theta$;
15:     end for
16: end for
```

In Algorithm 1, the replay memory D is first initialized to store each (s, a, r, s') state transition. Subsequently, the random parameters are used to initialize the Q network. The Q network can use the DualFc2 network or the DualMoe network in this paper, and then initialize the target network $target_Q$ with the parameters of the Q network Q. The $target_Q$ is used to obtain the y_t value during training, which can make the network fitting process relatively stable.

After the initialization process, the environment state s is reset. For each episode, there is a maximum step limit. During each step of training, an action a is selected using the ϵ-greedy algorithm. This algorithm chooses a random action with a probability of ϵ, and with a probability of $1 - \epsilon$, it selects the action a with the maximum estimated value output by the action-value network for the current state. Once the action is selected and executed, the agent receives a reward value r, while the environment transitions to the next state s'. This process yields a transition (s, a, r, s'). The transition is stored in the replay buffer D, and at regular intervals, mini-batches of samples are extracted from D to train the Q network and target network.

After sampling, the next step is to calculate the target Q value y_t. If the episode has terminated, the value of y_t is simply the obtained reward r_t. If the episode has not terminated, the target Q value y_t is calculated using the target network according to Formula (8), and the Q network is optimized by backpropagation using Formula (9) as the loss function. Additionally, every few steps, the parameters of the target network are updated to match the current Q network. Finally, the environment state is updated.

$$Loss = \frac{1}{2}(y_t - Q(s_t, o, a; \theta_t))^2 \tag{9}$$

In this study, the DecoupleRlO algorithm was used to train the DualFc2 and DualMoe networks to produce the decision-making agents DecoupleDualfc2 and DecoupleDualmoe, respectively. These two agents are jointly produced by the opponent network structures (DualFc2 and DualMoe) and the reinforcement learning algorithm DecoupleRlO. For comparison, we built four additional agents. The basic DQN algorithm without opponent modeling was used to train the DualFc2 and DualMoe networks to obtain the DualFc2 and DualMoe agents, respectively, which were used to investigate the role of the opponent networks. The DecoupleRlO training algorithm was used to train two benchmark network structures, DRON-Fc2 and DRON-Moe (the networks proposed in [32]), to obtain the DecoupleFc2 and DecoupleMoe agents, respectively, to demonstrate the performance of the DecoupleRlO learning algorithm in training DRON-FC2 and DRON-Moe.

5. Experiment and Analysis

We evaluated the model performance for two tasks, a simulated *soccer game* and a real game *quiz bowl*. The experimental settings were the same as those used in He's study [32]. Two players were used for both tasks, with different behaviors for each opponent. The main experimental parameters were as follows: the discount coefficient was γ = 0.9, the optimizer was Adagrad, the learning rate was 0.0005, and the minibatch size was 64. A greedy exploration strategy ϵ was used in the training process. The exploration rate was initially 0.3 and decayed linearly to 0.1 within 500,000 steps. All the models were trained for 50 epochs, and the cross entropy was used as a loss function for multitask learning.

5.1. Soccer Game

The first test scenario is a variant of a soccer game [46]. As shown in Figure 3, the game is played by two players, A and B, on a 27 $\times$ 18 grid. At the beginning of the game, players A and B are arranged in random squares in the left and right halves of the grid, and the ball is randomly assigned to one player (Figure 3(1)). Players can choose five actions: move N, S, W, or E or stand still (Figure 3(1)). Each player moves one square at a time (except when standing still). The goals have a size of 1 $\times$ 2 and are located at the center of the sidelines on the left and right sides of the grid. Players are not allowed to enter the gray areas around the goals. Any action that takes the player to a shadowed square or outside the boundary (Figure 3(5) and (6)) is invalid. If two players move onto the same square, the player who was holding the ball before the move was made loses the ball to the other player (Figure 3(4)), and the action is invalid. A player that moves the ball into the opponent's goal scores a point (Figure 3(2) and (3)), and the game ends. If neither player scores a point within one hundred moves, the game ends in a tie.

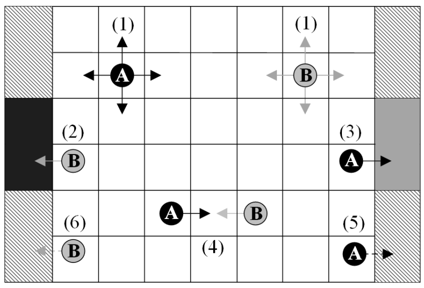


Figure 3. A soccer game.

5.1.1. Experimental Parameters

Two modes are used for the agent in the experiment: offensive and defensive. In offensive mode, the player always prioritizes offense over defense; in defensive mode, the player only defends its own goals. The agent randomly selects between the two modes in each game to create a changing strategy.

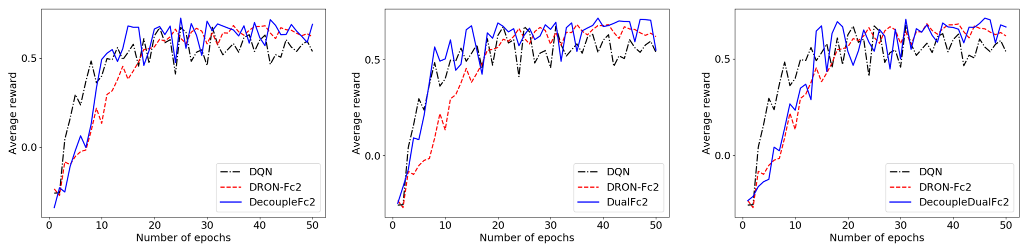
The input state information for the neural network includes the following: the coordinate (x_a, y_a) of player A, the coordinate of (x_b, y_b) player B, the size of the football field $(x_{min}, y_{min}, x_{max}, y_{max})$, player A's goal position, player B's goal position (the coordinates of each goal have a total of 6 dimensions) and ball possession (where 1 indicates that player A has possession of the ball, and 0 indicates that player B has possession of the ball).

The opponent features include the following: the frequency of the opponent's moves, the opponent's latest move sequence, the sequence of the opponent's recent actions, and the frequency at which the opponent loses the ball.

5.1.2. Analysis of the Results

In addition to the six players described in Section 4, three players are selected as baselines: a basic DQN player and the opponent network players DRON-Fc2 and DRON-Moe. After each epoch, we use 5000 randomly generated games (the test set) to evaluate the strategy and calculate the average reward.

First, we compare the performance of the -Fc2 agents (i.e., DecoupleDualFc2, DecoupleFc2, and DualFc2). The games include 200,000 total rounds, with 50 epochs and 4000 times for each epoch. Figure 4 shows the trend in the average rewards for the 5000th randomly generated game (the test set) after each epoch. In Figure 4a–c, the black dotted line represents the DQN model results, the red dotted line represents the DRON-Fc2 model results, and the blue curve represents the results of DecoupleFc2, DualFc2, and DecoupleDualFc2, respectively. It can be seen that the rewards of the three -Fc2 players are higher than those of the baseline players (i.e., DRON-Fc2 and DQN).



(**a**) (**b**) (**c**)

Figure 4. Reward curves for our agents vs. DQN and DRON-Fc2 for the soccer game. (**a**) Performance of DecoupleFc2. (**b**) Performance of DualFc2. (**c**) Performance of DecoupleDualFc2.

We record the maximum reward and the average reward for the last 10 epochs for each player. Table 1 shows that the maximum and average rewards of the -Fc2 agents exceed those of DRON-Fc2 by different quantities and are significantly higher than those of DQN. Among the three -Fc2 agents, DualFc2 achieves the highest average reward, DecoupleFc2 obtains the highest maximum reward, and the reward achieved by DecoupleDualFc2 is in between those of DualFc2 and DecoupleFc2.

The -Moe agents (i.e., DecoupleDualMoe, DecoupleMoe, and DualMoe) are trained using the same parameters as those used to train the -Fc2 agents. Figure 5a–c compare the reward curves for the -Moe players against those of the DQN and DRON-Moe players, respectively. The black dotted line represents the DQN player, the red dotted line represents the DRON-Moe player, and the blue curve represents the -Moe players during training. The rewards for players with opponent modeling are significantly higher than those for the DQN player, and the -Moe players achieve clearly higher rewards than the DRON-Moe player.

Table 1. Comparison of rewards achieved using DecoupleDualFc2 in soccer.

Player	Maximum Reward	Average Reward
DQN	0.6658	0.5479
DRON-Fc2	0.6826	0.6439
DualFc2	0.7154	0.6723
DecoupleFc2	0.7242	0.6472
DecoupleDualFc2	0.7116	0.6590

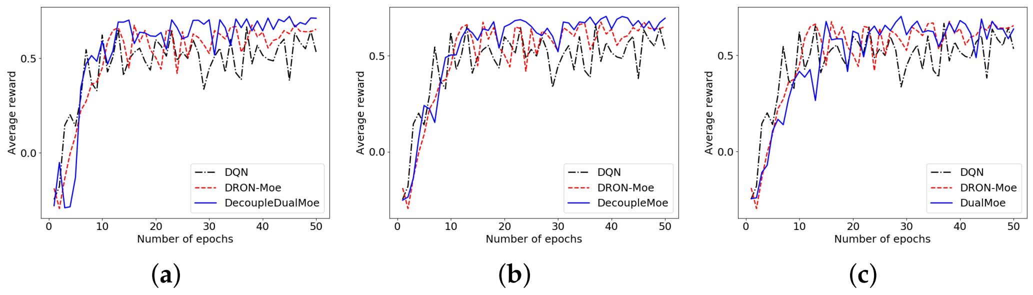


Figure 5. Reward curves for our agents vs. DQN and DRON-Moe for the soccer game. (**a**) DecoupleDualMoe vs. DQN and DRON-Moe. (**b**) DualMoe vs. DQN and DRON-Moe. (**c**) DualMoe vs. DQN and DRON-Moe.

The maximum reward and average reward for the last 10 epochs in the soccer game for the -Moe players are similarly recorded and are listed in Table 2. The maximum rewards for the players built in this study are significantly higher than those for the baseline DRON-Moe and DQN players, and the highest reward is obtained using the DecoupleDualMoe model. In terms of the average reward for the last 10 epochs, the performance of the DualMoe and DRON-Moe players is basically the same, and the other two -Moe players significantly outperform the DRON-Moe player. The DecoupleDualMoe model exhibits the best performance in terms of both average and maximum rewards.

Table 2. Comparison of rewards for DecoupleDualMoe in soccer.

Player	Maximum Reward	Average Reward
DQN	0.6658	0.5479
DRON-Moe	0.6764	0.6286
DualMoe	0.7022	0.6206
DecoupleMoe	0.7056	0.6758
DecoupleDualMoe	0.722	0.6949

These results indicate that the incorporation of opponent modeling optimizes agents' strategies and that the proposed opponent modeling framework is more effective than the existing method.

5.2. Quiz Bowl

A quiz bowl is a simple game that is usually played between two teams. Questions are read to players, who "buzz in" (usually before finishing reading the question) to provide an answer. A player that answers a question correctly is awarded 10 points. A player that answers a question incorrectly loses 5 points, and another player can buzz in to answer. If no player answers the questions correctly, no points are awarded.

5.2.1. Experimental Setting

Two models are required to train a player for the quiz bowl game: a content model that predicts the answer to a given (partial) question and a buzzing model that determines the time at which to buzz in.

In the content model, a recurrent neural network with a gated recurrent unit (GRU) is adopted for text classification. This network can read questions sequentially and output a probability distribution vector of the answers.

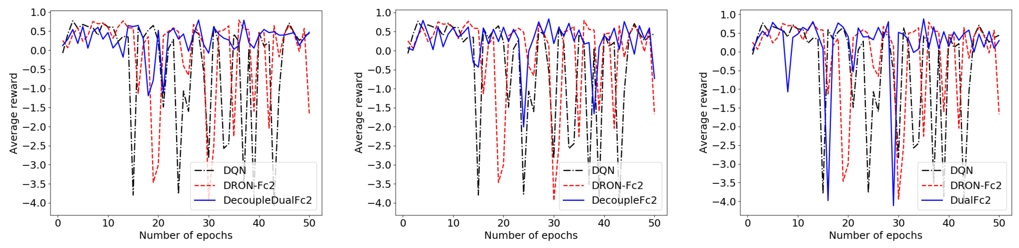
The buzzing model is trained based on the content model. The different reinforcement learning methods presented in the previous section are used to build agents with different buzzing strategies.

The training data are taken from [32] and contain 1045 answers, 37.7 k questions, and 3610 users. These data are divided into two disjoint subsets: one is used to train the content model, and the other is used to train the buzzing model. The players are divided into two categories according to the buzzing time: impulsive players and conservative players. Impulsive players typically answer questions early but with a low accuracy. Conservative players tend to answer later with a high accuracy.

The state information input to the neural network includes the probability distribution vector output of the content model, the number of words seen, and the current game status. The opponent characteristics include the number of questions answered by the opponent, the average buzzing time of the opponent, and the error rate of the opponent.

5.2.2. Results

Based on the content model, we trained each abovementioned player for 50 epochs on the quiz bowl game, and testing was performed using the test set after each epoch. Figure 6a–c record the changes in the rewards of the -Fc2 players. The DQN player cannot adapt to changes in the strategy of the opponents, resulting in sharp fluctuations in the rewards during the later stage of training, whereas the rewards for all the players with opponent modeling are more robust. The rewards obtained with the DualFc2, DecoupleFc2, and DecoupleDualFc2 models are significantly more stable than those obtained with the baseline model DRON-Fc2.



(**a**) (**b**) (**c**)

Figure 6. Reward curves for our agents vs. DQN and DRON-Fc2 in the quiz bowl game. (**a**) DecoupleDualFc2 vs. DQN and DRON-Fc2. (**b**) DecoupleFc2 vs. DQN and DRON-Fc2. (**c**) DualFc2 vs. DQN and DRON-Fc2.

The highest reward and the average reward for the last 10 epochs are similarly recorded for the -Fc2 players and are shown in Table 3. Incorporating the opponent model provides the players with clear advantages over the DQN network in terms of both the highest and average rewards. The highly effective proposed network structure and learning algorithm result in the DualFc2 player obtaining the highest reward and DecoupleDualFc2 obtaining the highest average reward.

Table 3. Comparison of rewards for DecoupleDualFc2 in the quiz bowl game.

Player	Maximum Reward	Average Reward
DQN	0.7709	−0.2171
DRON-Fc2	0.7918	−0.0736
DualFc2	0.8788	0.3426
DecoupleFc2	0.8331	0.2781
DecoupleDualFc2	0.7910	0.4099

Figures 7a–c show the performance of the -Moe players. In terms of the degree of fluctuations in the rewards, the performance of the players with opponent modeling is clearly more robust than that of the DQN model without opponent modeling. The -Moe players obtain a higher maximum reward than the DRON-Moe player.

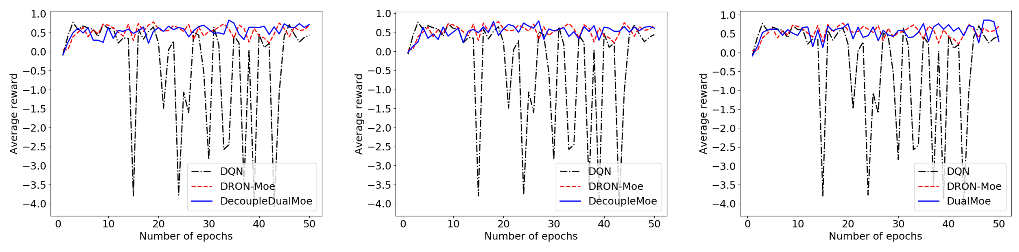


Figure 7. Reward curves for our agents vs. DQN and DRON-Moe in the quiz bowl game. (**a**) DecoupleDualMoe vs. DQN and DRON-Moe; (**b**) DecoupleMoe vs. DQN and DRON-Moe; (**c**) DualMoe vs. DQN and DRON-Moe.

The maximum rewards and the average rewards for the last 10 epochs are similarly recorded and calculated and are shown in Table 4. The maximum and average rewards of the players with opponent modeling are greater than those of the DQN player. The DecoupleDualMoe player obtains the highest average reward, and the DualMoe player achieves the highest maximum reward.

Table 4. Comparison of rewards for DecoupleDualMoe in the quiz bowl game.

Player	Maximum Reward	Average Reward
DQN	0.7709	−0.2171
DRON-Moe	0.7819	0.5292
DualMoe	0.8606	0.6174
DecoupleMoe	0.8017	0.5911
DecoupleDualMoe	0.8307	0.6346

These results indicate that our model, consisting of a network structure and training algorithm for opponent modeling, is effective in helping agents make better decisions and increase payoffs in interactive scenarios.

6. Conclusions and Future Work

Opponent modeling is an important issue for agents to make appropriate decisions in various strategic settings. A novel implicit opponent modeling framework was developed in this study to improve the payoffs of players' decisions under interactive scenarios. Two opponent networks, DualFc2 and DualMoe, were constructed by using deep neural networks. A novel reinforcement learning algorithm for the opponent networks with a coupling strategy was proposed.

The most relevant study in the literature to the present study is based on the DRON approach in which opponents are modeled implicitly. The DRON network structures and learning algorithms are different from those used in our proposed method. More importantly, our approach uses separate modules to evaluate the state and the action in the network structure and objectively evaluates the actions of the opponents via an appropriate learning algorithm for the opponent network.

Experiments were conducted demonstrating that our opponent modeling framework can effectively improve agents' payoffs under dynamic interactive scenarios over those obtained using DQN without opponent modeling and DRON.

Future research on opponent modeling still faces some challenges. For example, the existing opponent modeling approach often assumes that agents can obtain complete information for the environment and opponents. However, environmental or agent characteristics are only partially observable in many application scenarios, such that the agent obtains incomplete or uncertain observations of the environment or other agents. This partial observation increases the difficulty of opponent modeling. Methods for opponent modeling under partially observable conditions have been proposed in the field of extensive games. Opponent modeling under partially observable conditions and wider decision intelligence scenarios warrants further exploration.

Author Contributions: Methodology, C.L.; Software, J.C.; Validation, T.Z.; Resources, E.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This study was supported by the Natural Science Foundation of Guangdong Province of China under Grant 2021A1515011940; and the Natural Science Foundation of Liaoning Province of China under Grant 2021-MS-114.

Data Availability Statement: The data that support the findings of this study are available from the corresponding author upon reasonable request.

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

References

1. Sun, C.; Liu, W.; Dong, L. Reinforcement Learning With Task Decomposition for Cooperative Multiagent Systems. *IEEE Trans. Neural Netw. Learn. Syst.* **2021**, *32*, 2054–2065. [CrossRef] [PubMed]
2. Silver, D.; Huang, A.; Maddison, C.; Guez, A.; Sifre, L.; Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. Mastering the game of Go with deep neural networks and tree search. *Nature* **2016**, *529*, 484–489. [CrossRef] [PubMed]
3. Ferreira, J.C.L. Opponent Modelling in Texas Hold'em: Learning Pre-Flop Strategies in Multiplayer Tables. Master's Thesis, University of Porto, Porto, Portugal, 2012.
4. Chen, H.; Wang, C.; Huang, J.; Kong, J.; Deng, H. XCS with opponent modelling for concurrent reinforcement learners. *Neurocomputing* **2020**, *399*, 449–466. [CrossRef]
5. Sui, Z.; Pu, Z.; Yi, J.; Wu, S. Formation Control With Collision Avoidance Through Deep Reinforcement Learning Using Model-Guided Demonstration. *IEEE Trans. Neural Netw. Learn. Syst.* **2021**, *32*, 2358–2372. [CrossRef] [PubMed]
6. Noor, U.; Anwar, Z.; Malik, A.W.; Khan, S.; Saleem, S. A machine learning framework for investigating data breaches based on semantic analysis of adversary attack patterns in threat intelligence repositories. *Future Gener. Comput. Syst.* **2019**, *95*, 467–487. [CrossRef]
7. Li, Z. Artificial Intelligence Algorithms for Strategic Reasoning over Complex Multiagent Systems. In Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems, London, UK, 29 May–2 June 2023; pp. 2931–2933.
8. Bondi, E. Bridging the Gap Between High-Level Reasoning in Strategic Agent Coordination and Low-Level Agent Development. In Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, Montreal, QC, Canada, 13–17 May 2019; pp. 2402–2404. [CrossRef]
9. Tsantekidis, A.; Passalis, N.; Toufa, A.S.; Saitas-Zarkias, K.; Chairistanidis, S.; Tefas, A. Price Trailing for Financial Trading Using Deep Reinforcement Learning. *IEEE Trans. Neural Netw. Learn. Syst.* **2021**, *32*, 2837–2846. [CrossRef]
10. Albrecht, S.V.; Stone, P. Autonomous agents modelling other agents: A comprehensive survey and open problems. *Artif. Intell.* **2018**, *258*, 66–95. [CrossRef]
11. Yu, X.; Jiang, J.; Zhang, W.; Jiang, H.; Lu, Z. Model-based opponent modeling. *Adv. Neural Inf. Process. Syst.* **2022**, *35*, 28208–28221.
12. Nashed, S.; Zilberstein, S. A survey of opponent modeling in adversarial domains. *J. Artif. Intell. Res.* **2022**, *73*, 277–327. [CrossRef]
13. Lockett, A.J.; Chen, C.L.; Miikkulainen, R. Evolving explicit opponent models in game playing. In Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, London, UK, 7–11 July 2007; pp. 2106–2113. [CrossRef]
14. Schwarting, W.; Seyde, T.; Gilitschenski, I.; Liebenwein, L.; Sander, R.; Karaman, S.; Rus, D. Deep Latent Competition: Learning to Race Using Visual Control Policies in Latent Space. In Proceedings of the Conference on Robot Learning, London, UK, 8–11 November 2021; pp. 1855–1870.
15. Liu, L.; Yang, J.; Zhang, Y.; Zhang, J.; Ma, Y. An Overview of Opponent Modeling for Multi-agent Competition. In Proceedings of the International Conference on Machine Learning for Cyber Security, Guangzhou, China, 2–4 December 2022; Springer: Berlin/Heidelberg, Germany, 2022; pp. 634–648. [CrossRef]
16. Freire, I.T.; Puigbò, J.Y.; Arsiwalla, X.D.; Verschure, P.F. Modeling the opponent's action using control-based reinforcement learning. In Proceedings of the Biomimetic and Biohybrid Systems: 7th International Conference, Living Machines 2018, Paris, France, 17–20 July 2018; Springer: Berlin/Heidelberg, Germany, 2018; pp. 179–186. [CrossRef]
17. Kim, M.J.; Kim, K.J. Opponent modeling based on action table for MCTS-based fighting game AI. In Proceedings of the 2017 IEEE conference on computational intelligence and games (CIG), New York, NY, USA, 22–25 August 2017; pp. 178–180. [CrossRef]

18. Gabel, T.; Godehardt, E. I Know What You're Doing: A Case Study on Case-Based Opponent Modeling and Low-Level Action Prediction. In Proceedings of the ICCBR (Workshops), Frankfurt, Germany, 28–30 September 2015; pp. 13–22.
19. Carmel, D.; Markovitch, S. Opponent Modeling in Multi-Agent Systems. In Proceedings of the Workshop on Adaption and Learning in Multi-Agent Systems, IJCAI '95, Montreal, QC, Canada, 21 August 1995; pp. 40–52. [CrossRef]
20. Hosokawa, Y.; Fujita, K. Opponent's Preference Estimation Considering Their Offer Transition in Multi-Issue Closed Negotiations. *IEICE Trans. Inf. Syst.* **2020**, *103*, 2531–2539. [CrossRef]
21. Zafari, F.; Nassiri-Mofakham, F. Popponent: Highly accurate, individually and socially efficient opponent preference model in bilateral multi issue negotiations. *Artif. Intell.* **2016**, *237*, 59–91. [CrossRef]
22. Efstathiou, I.; Lemon, O. Learning Better Trading Dialogue Policies by Inferring Opponent Preferences. In Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems, Singapore, 9–13 May 2016; pp. 1403–1404.
23. Tian, Z.; Wen, Y.; Gong, Z.; Punakkath, F.; Zou, S.; Wang, J. A regularized opponent model with maximum entropy objective. In Proceedings of the IJCAI International Joint Conference on Artificial Intelligence, International Joint Conferences on Artifical Intelligence (IJCAI), Macao, China, 10–16 August 2019; Volume 28, pp. 602–608. [CrossRef]
24. Gallego, V.; Naveiro, R.; Insua, D.R.; Gómez-Ullate, D. Opponent Aware Reinforcement Learning. *arXiv* **2019**, arXiv:1908.08773.
25. Shen, M.; How, J.P. Robust opponent modeling via adversarial ensemble reinforcement learning. In Proceedings of the International Conference on Automated Planning and Scheduling, Guangzhou, China, 2–13 August 2021; Volume 31, pp. 578–587. [CrossRef]
26. Baarslag, T.; Hendrikx, M.J.; Hindriks, K.V.; Jonker, C.M. Learning about the opponent in automated bilateral negotiation: A comprehensive survey of opponent modeling techniques. *Auton. Agents Multi-Agent Syst.* **2016**, *30*, 849–898. [CrossRef]
27. Wu, Z.; Li, K.; Xu, H.; Zang, Y.; An, B.; Xing, J. L2E: Learning to exploit your opponent. In Proceedings of the 2022 International Joint Conference on Neural Networks (IJCNN), Padua, Italy, 18–23 July 2022; pp. 1–8. [CrossRef]
28. Schadd, F.; Bakkes, S.; Spronck, P. Opponent modeling in real-time strategy games. In Proceedings of the GAME-ON 2007, Bologna, Italy, 20–22 November 2007; pp. 61–70.
29. Zhang, W.; Wang, X.; Shen, J.; Zhou, M. Model-based Multi-agent Policy Optimization with Adaptive Opponent-wise Rollouts. In Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Montreal, QC, Canada, 19–27 August 2021; pp. 3384–3391. [CrossRef]
30. Hernandez-Leal, P.; Zhan, Y.; Taylor, M.E.; Sucar, L.E.; de Cote, E.M. Efficiently detecting switches against non-stationary opponents. *Auton. Agents Multi-Agent Syst.* **2017**, *31*, 767–789. [CrossRef]
31. Foerster, J.N.; Chen, R.Y.; Al-Shedivat, M.; Whiteson, S.; Abbeel, P.; Mordatch, I. Learning with Opponent-Learning Awareness. In Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, 10–15 July 2018; pp. 122–130. [CrossRef]
32. He, H.; Boyd-Graber, J.; Kwok, K.; Daumé, H., III. Opponent modeling in deep reinforcement learning. In Proceedings of the International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016; pp. 1804–1813. [CrossRef]
33. Everett, R.; Roberts, S.J. Learning Against Non-Stationary Agents with Opponent Modelling and Deep Reinforcement Learning. In Proceedings of the AAAI Spring Symposia, Palo Alto, CA, USA, 26–28 March 2018.
34. Lu, H.; Gu, C.; Luo, F.; Ding, W.; Liu, X. Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning. *Future Gener. Comput. Syst.* **2020**, *102*, 847–861. [CrossRef]
35. Haotian, X.; Long, Q.; Junjie, Z.; Yue, H.; Qi, Z. Research Progress of Opponent Modeling Based on Deep Reinforcement Learning. *J. Syst. Simul.* **2023**, *35*, 671. [CrossRef]
36. 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. [CrossRef]
37. Michale, J. Positive and Negative Reinforcement, A Distinction That Is No Longer Necessary; or a Better Way to Talk about Bad Things. *J. Organ. Behav. Manag.* **2005**, *24*, 207–222. [CrossRef]
38. Schmidhuber, J. Deep learning in neural networks: An overview. *Neural Netw.* **2015**, *61*, 85–117. [CrossRef]
39. Barron, E.; Ishii, H. The Bellman equation for minimizing the maximum cost. *Nolinear Anal. Theory Method Appl.* **1989**, *13*, 1067–1090. [CrossRef]
40. Jang, B.; Kim, M.; Harerimana, G.; Kim, J.W. Q-learning algorithms: A comprehensive classification and applications. *IEEE Access* **2019**, *7*, 133653–133667. [CrossRef]
41. Weitkamp, L.; van der Pol, E.; Akata, Z. Visual rationalizations in deep reinforcement learning for atari games. In Proceedings of the Artificial Intelligence: 30th Benelux Conference, BNAIC 2018, 's-Hertogenbosch, The Netherlands, 8–9 November 2018; pp. 151–165. [CrossRef]
42. Fayjie, A.R.; Hossain, S.; Oualid, D.; Lee, D.J. Driverless car: Autonomous driving using deep reinforcement learning in urban environment. In Proceedings of the 2018 15th International Conference on Ubiquitous Robots (UR), Honolulu, HI, USA, 26–30 June 2018; pp. 896–901. [CrossRef]
43. Zhang, W.; Gai, J.; Zhang, Z.; Tang, L.; Liao, Q.; Ding, Y. Double-DQN based path smoothing and tracking control method for robotic vehicle navigation. *Comput. Electron. Agric.* **2019**, *166*, 104985. [CrossRef]
44. Wang, Z.; Schaul, T.; Hessel, M.; Hasselt, H.; Lanctot, M.; Freitas, N. Dueling network architectures for deep reinforcement learning. In Proceedings of the International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016; pp. 1995–2003. [CrossRef]

45. Van Hasselt, H.; Guez, A.; Silver, D. Deep reinforcement learning with double q-learning. In Proceedings of the AAAI Conference on Artificial Intelligence, Phoenix, AZ, USA, 12–17 February 2016; Volume 30. [CrossRef]
46. Littman, M.L. Markov Games as a Framework for Multi-Agent Reinforcement Learning. In Proceedings of the Eleventh International Conference on International Conference on Machine Learning, ICML'94, New Brunswick, NJ, USA, 10–13 July 1994; pp. 157–163. [CrossRef]