DQN for Coordinating Multi-agent Cooking

Yiwei Zhang,
Department of Engineering, Glasgow University, Glasgow, United Kingdom
2273664z@student.gla.ac.uk

Abstract. Reinforcement learning (RL) is a very widely used field. The difference between RL and other branches of machine learning (such as supervised learning and unsupervised learning) is that RL centers on interactive learning. The RL model (also known as the agent) learns in interaction with the environment to maximize the reward function. In the paper "too many cooks," the authors developed a method called Bayesian Delegation to enable human-like coordination by inferring the sub-tasks of others quickly. However, limitations still exist in the partial order of sub-tasks. First, implementing the sub-task in terms of efficient actions or which agent(s) should work on it is not specified. Second, the sub-tasks may be finished in many different orders since the ordering of sub-tasks is partial. The project proposes solutions to these challenges using Deep Q-Learning (DQN) and Bayesian Inference. In the DQN experiment, value approximation performs well in the simple multi-agent environment.

Keywords: Coordination; Multi-agent Reinforcement Learning; Deep Q-Learning Network; Bayesian Inference.

1. Introduction

RL is a complex and vast field, and what distinguishes RL from other branches of machine learning, such as supervised and unsupervised learning, is that RL is centered on interactive learning. RL models (also known as agents) learn in interaction with the environment to maximise a reward function. In RL, the designer does not teach the agent how to do things; we only define what the agent is expected to achieve [1]. Then, based on specific experimental results, rewards are determined based on the success or failure of the agent. RL is a better tool for making decisions in complex environments. The central mathematical concept is that of the Markov decision process. This paper builds on the work of "too many cooks" and mentions that "when agents jointly plan for a single sub-task, they currently have no way of knowing when they have completed their individual "part" of the joint effort" for discussion. This paper uses the model to measure how multiple agents divide work. The first model is based on the traditional DQN, builds the target network and evaluation network, and infers the task assignment of the agent.

2. Related Work

Reinforcement Learning Reinforcement learning involves two problems: optimal control and learning by trial and error [2]. The dynamic programming method is widely used for optimal control problems, which was proposed by Richard Bellman in 1950. The Bellman equation is a commonly used component of RL algorithms. For trial-and-error learning, Edward Thorndike is the first person to elaborate and define the concept. In the 1980s, reinforcement learning was proposed based on these two fields. In recent years, the combination of reinforcement learning and deep learning has achieved certain results.

Machine learning involves learning from structured or unstructured data through various algorithms [2]. At present, the exponential growth of digital data and the improvement of computer computing power provide a good foundation for machine learning. Reinforcement learning is a branch of machine learning. For reinforcement learning systems, the agent is preset to have no prior knowledge. In reinforcement learning systems, the agent first collects knowledge and then adjusts its actions based on the collected knowledge to maximise the objective. However, the problem is how to judge whether the collected knowledge is enough. Insufficient effective knowledge collected or

continuous collection of knowledge has repeatedly appeared in the research of reinforcement learning systems [2].

Multi-Agent Reinforcement Learning. Multi-Agent Coordination has relatively common practical application scenarios. Planning and coordination are two terms that need to be mentioned in multi-agent reinforcement learning [1]. In centralised planning, the system already assigns the planning steps that each agent needs to perform. However, the time required for centralised planning is relatively long, and when a node fails, the whole system may appear unstable. This situation is often not suitable for real-time programs when the number of agents is large. The Multi-Agent Coordination coordinates and generates plans and action steps through various agents.

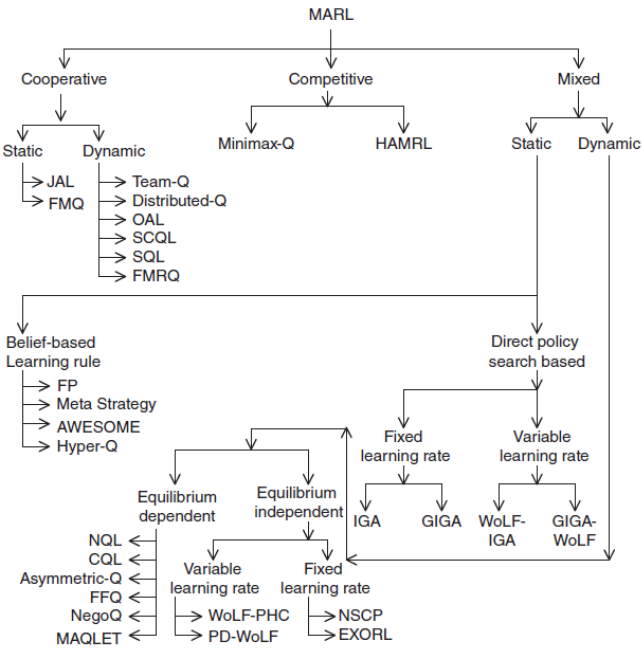

Fig 1. Classification of multi-agent reinforcement learning [1]

Based on the task type, MARL is classified into three categories: cooperative, competitive and hybrid (Fig.1). The too many cooks in this article belong to the cooperative task type. For the cooperative task type, the reward functions of each agent are related to each other. For cooperative task types, Nash equilibrium is commonly used to calculate the equilibrium point. Furthermore, for multi-agent environments, some algorithms can only be used for static games [3].

Multi-agent reinforcement learning consists of several parts. RL is used for single-agent reinforcement learning, and GT is used for multi-agent action evaluation. DP is a standard method for solving MDP problems, which is used to decompose a complex problem into finite subproblems that can be solved by the DP algorithm. Marl combines RL, GT, DP, etc. [1]. Figure 2 divides different agents according to whether they cooperate or not, whether the agent has prior knowledge about other agents, the degree of compliance with the coordination protocol, whether there is a central decision-making agent in the decision-making process, inter-agent communication dependencies, and team composition heterogeneity or homogeneity. The multi-robot system (Fig.3). The article' too many cooks' first quantitatively discusses Homogeneous teams, multi-agents use the same algorithm, and compares the experiments of BD, UP, FB, D&C, Greedy 5 algorithms performing the same task The results; and then quantitatively discuss the Heterogeneous teams, that is, two agents use different algorithms, and compare the experimental results of five algorithms performing the same task. Through the comparative analysis of the data, the adaptation scenarios and performance of the five algorithms are displayed.

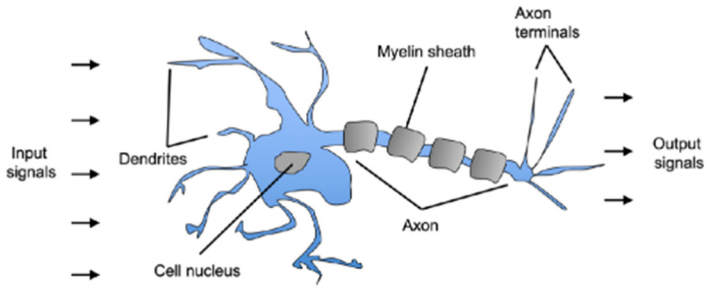

Fig 2. A neuron processing chemical and electrical signals [4]

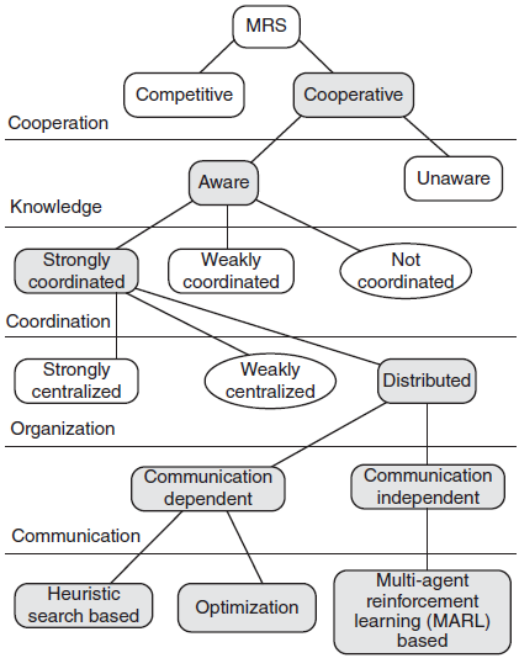

Fig 3. Classification of multi-robot systems [1].

This paper briefly discusses algorithms such as greedy. For the greedy algorithm, the agent can in principle maximise the objective through the greedy strategy. However, when multiple agents make decisions independently of each other and use greedy strategies in parallel, the resulting joint actions may become suboptimal rather than optimal [3].

The performance quantification parameters of the multi-agent system mainly include: fewer time steps, higher completion. From the observation of the operation results of the open-divider, partial-divider and full-divider of too many cooks, it can be found that it is consistent with the above theoretical analysis Seems to basically match. When the initial environment requires multi-agent coordination (from non-closed to semi-isolated to isolated), the independent greedy strategy of each agent leads to a relatively slower system running time. This results in the greedy algorithm ranking at the end of the five algorithmic strategies.

Then how to achieve coordination and cooperation among multi-agents in a complex environment, overall greedy, and equilibrium. Whether too many cooks are fully cooperative or not, these issues need to be discussed. Whether full cooperation can maximise the goal, whether acting alone is not full cooperation, and whether acting alone is not coordination. These issues also need to be discussed. Wu et al. analyzed that when faced with more complex multi-agent coordination problems, Bayesian commissioned data outperforms other commissions. Furthermore, agents using the Bayesian delegation algorithm can optimise the overall operation when multi-agents coordinate.

3. Method

This article is based on the paper "too many cooks" and discusses using techniques such as dqn and q-leaning to build a basic model to measure how multi-agent agents divide work to complete cooking. This article tries to build a primary DQN network based on torch and OpenAI's gym toolkit. The network aims to optimise multi-agent coordination to complete cooking. When performing multi-agent coordination and cooperation, the priority of subtasks and the strategy selection of agents are the main considerations in this paper.

3.1 Environment

This paper builds on the work of 'too many cooks' and uses the OpenAI Gym toolkit. The OpenAI Gym toolkit aims to facilitate model development for reinforcement learning. The toolkit contains multiple pre-configured reinforcement learning environments and optimises the process of building and using reinforcement learning environments [4].

The first is to build the basic environment. The environment of this paper is based on the grid world environment. This article uses the pyglet library to render the visual environment. Regarding the establishment of the simulation environment, the first thing to consider is the choice of the initial environment framework. For continuous actions, the researcher can consider using the particle environment of OpenAI. The kitchen environment setting in this article is mainly based on conventional discrete actions such as direction movement, so the initial environment is based on the grid world environment.

N_ACTIONS = 4
N_STATES = 5
N_ACTION = Action_space
N_STATES = State_space
Action_space = [0: cut tomato 1: cut lettuce 2: cooperate 3: mix]
State_space = [$state_{agent1}$, $state_{agent2}$, $state_{tomato}$, $state_{lettuce}$, $state_{salad}$]

The agent has three states of none, cut1 and cut2, tomato and lettuce have three states of not cut, 1 and 2, and salad has three states of none, 1 and 2 (Figure 4).

```
playground = [1, 1, 1, 1, 1,3,1,
              5, 0, 2, 1, 2,0,4,
              5, 0, 0, 1, 0,0,1,
              6, 0, 0, 1, 0,0,1,
              1, 0, 0, 1, 0,0,1,
              1, 0, 0, 1, 0,0,1,
              1, 1, 1, 1, 1,1,1]
```

Fig 4. Kitchen Simulation environment

In this paper, the kitchen simulation environment is built first. This is a simulated environment with seven rows and seven columns. The objective objects corresponding to the values are: [1: table 2: agent 3: tomato 4: lettuce 5: operation table 6: destination]. This paper adjusts the simulation for three different states of open-divider, partial-divider and full-divider environment.

When the agent moves next to the ingredients and chooses to cut the ingredients, its status value will change. For example, when the agent cuts the tomato, its state becomes [2[cut_tomato]]. This paper also optimises and adjusts the reward structure (Fig.5-6). In different scenarios, the reward obtained by the agent is different. For example, when an agent chooses to cooperate, its reward value is greater than cooking alone. In addition, when the ingredients are in different states, the rewards are

also different to avoid the agent repeating invalid action strategies. When cooking such as salad is completed, the agent will also get a certain value of the reward.

```
a1 = dqn.choose_action(s)  # s was initial state
print(a1, s, "A1")
action_space = [0, 1, 2, 3, ]  # 0:cut tomato 1:cut lettuce 2:cooperate 3:mix
```

Fig 5. The agent first chooses the next action through the dqn network.

```
BATCH_SIZE = 10  # the number of sample
LR = 0.01  # learning rate
EPSILON = 0.9  # greedy policy
GAMMA = 0.9  # reward discount
TARGET_REPLACE_ITER = 50  # the frequency of update the target-net
MEMORY_CAPACITY = 2000  #

N_ACTIONS = 4
N_STATES = 5
s = np.array([0, 0, 0, 0, 0])
env = environment2_test_full_divider.Env()
reward_list = []
```

Fig 6. DQN network hyperparameter configuration

The parameters of the learning rate, discount factor, and ε greedy strategy is shown in the figure 7.

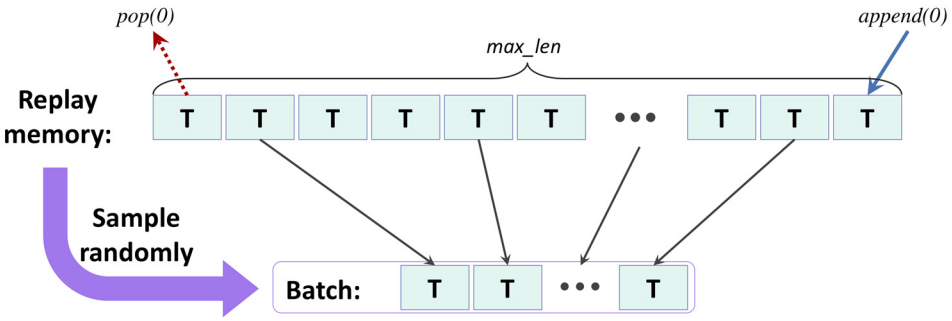

Fig 7. The basic process of memory replay [4]

The grid world environment built in this article presets two agents. Env.step for configuration and action. Agent explores the environment and generates a five-element array, and the data is stored in the memory cache. The memory buffer is used for replay memory. Randomly select a batch of samples from the memory buffer for computing loss and updating network parameters.

3.2 DQN and Memory Reply

In 1989, Christopher Watkins proposed the q-learning component of DQN. In 1992, Long-Ji proposed the experience playback mechanism [5]. In 2013, Mnih et al. (2013) proposed DQN and deep-q networks. In 2015, double DQN and PER (prioritised experience replay) appeared. For 'too many cooks', the article mentioned that using DQN for optimisation is not successful for its limited computing resource configuration, so this paper tries to conduct a simple preliminary discussion.

Multi-agent policy evolution and experience discarding problems make it non-trivial to apply experience replay to multi-agent environments [6]. A fundamental problem in the field of multi-agents is the dimensionality explosion problem. The increasing number of agents leads to an

exponential increase in complexity. What followed was a sharp increase in the amount of calculations. In addition, the specific application of policy gradient and performance collapse are also issues that DQN needs to consider.

The article uses torch to build a DQN network. The article tries to run too many cooks through DQN network optimisation. The first is the construction of the neural network. The network has four layers. The first layer is a fully connected layer, which is configured with 50 neuron nodes; the second layer is a fully connected layer, which is configured with 50 neuron nodes. The third layer is connected to the hidden layer and outputs values using the activation function ReLu. The fourth layer is the hidden layer connected to the output layer, and the output value of this layer is the desired strategy selection value.

Unlike the traditional DQN, the article has two networks: the target network target_net and the evaluation network eval_net. Furthermore, the article utilises experience replay, which can help us to train more efficiently. We adjust the degree of increasing the reward so that the reward can converge quickly.

The basic principle of dqn is:

The state-value function:

$$\upsilon_\pi(s) \stackrel{\text{def}}{=} E_\pi[G_t|S_t = s] = E_\pi[\sum_{k=0} \gamma^{k+1} R_{t+k+1}|S_t = s] \tag{1}$$

The value function uses the value of Gt as the standard to measure the pros and cons of the agent in each state.

The action-value function

$$q_\pi(s,a) \stackrel{\text{def}}{=} E_\pi[G_t|S_t = s, A_t = a] = E_\pi[\sum_{k=0} \gamma^{k+1} R_{t+k+1}|S_t = s, A_t = a] \tag{2}$$

Each action-state pair can be defined as a value $q_\pi(s,a)$, which is generally called the action-value function.

Bellman equation

$$\upsilon_\pi(s) = \sum_{a\in\hat{A}} \pi(a|s) \sum_{s'\in\hat{S}, r\in\hat{R}} p(s',r|s,a)[r + \gamma\upsilon_\pi(s')] \tag{3}$$

The value function s of a state is linked to its subsequent state's value function s' through the Bellman equation. The Bellman equation simplifies the computation of the value function because the equation eliminates the iterative loop of the reinforcement learning algorithm along the time axis.

DQN networks can only be used in environments with discrete action spaces [7]. DQN is an off-policy algorithm. DQN uses the Bellman equation to learn the optimal q-function. The optimal q-function enables DQN to learn from the experience collected by any agent. The Bellman equation of DQN is:

$$Q^\pi_{DQN}(s,a) \approx r + \gamma max Q^\pi_{a'}(s',a') \tag{4}$$

For the DQN algorithm, there are three main strategies for the agent to collect experience: greedy, ε-greedy and Boltzmann.

3.3 Bayesian Inference & BRTDP

Statistical Inference is based on sample observation data with randomness. Statistical Inference uses specific problem conditions and assumptions to make inferences about unknown things in the form of probability. Statistical Inference mainly analyses and calculates two types of problems: parameter estimation and hypothesis testing. Probability distributions are fundamental elements in the process of statistical analysis. Probability distribution is an abstract mathematical expression of objective laws in the objectively real world. Bayesian Inference is a theory in statistical Inference.

The basis of the theory is Bayes' theorem. When applying Bayesian theory, the prior probability is generally quantified first, and then the prior probability is multiplied by the likelihood function, and the value is normalized to obtain the posterior probability distribution through calculation. the posterior distribution can be expressed as below according to Bayes' theorem, given class variable y and dependent feature vector $(x_1, \ldots, x_n)$:

$$\mathrm{P}\,(y \mid x_1, \ldots, x_n) = \frac{P(y)P(x_1,\ldots,x_n \mid y)}{P\,(x_1,\ldots,x_n)} \tag{5}$$

Based on the naive conditional independence assumption that

$$\mathrm{P}(x_i \mid y, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = P(x_i \mid y)$$

For all i, this relationship is simplified to

$$\mathrm{P}\,(y \mid x_1, \ldots, x_n) = \frac{P(y)\prod_{i=1}^{n}\mathrm{P}(x_i \mid y)}{P(x_1,\ldots,x_n)} \tag{6}$$

Since $\mathrm{P}\,(x_1, \ldots, x_n)$ is constant given the input, we can use the following classification rule:

$$\mathrm{P}\,(y \mid x_1, \ldots, x_n) \propto P(y)\prod_{i=1}^{n}\mathrm{P}(x_i \mid y) \tag{7}$$

$$\hat{y} = arg\max_{y} P(y)\prod_{i=1}^{n}\mathrm{P}(x_i \mid y) \tag{8}$$

The code [8] is based on Bayesian Inference; it requires a database of running actions in different states. Then with the underlying database, once the current state of each agent is received, the code will calculate the likelihood of every possible action and return the action with the highest probability. Laplacian correction is also added to the code to ensure the accuracy of the code. In this case, the code will weaken the lousy effect caused by the lack of a database.

'Too many cooks' uses Bayesian reasoning and reverse programming in its Bayesian delegation algorithm. Inverse programming uses the BRTDP (bounded real time dynamic programming) algorithm. Inverse planning, this method is a posteriori reasoning. The planner based on model calculates the likelihood. Each agent performs posterior Inference based on the observed history (prior) of other agents. After the calculation (BRTDP, hyperparameters need to be set), update the respective values of v and q, and then select and execute a separate operation or cooperation strategy [9]. As can be seen from the article, 'too many cooks' evolve each agent by using a distributed approach (Fig 8) [10].

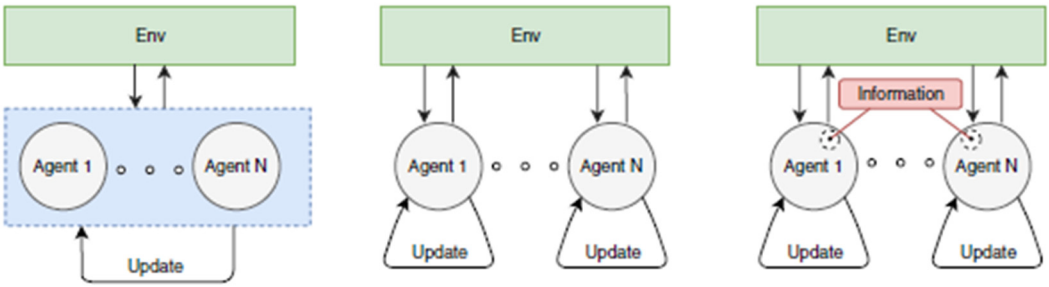

Fig 8. Training schemes in the multi-agent setting [10].

4. Result and Discussion

The previous chapter expounded the basic principles of related modules. This chapter will explain the DQN and q-learning algorithms based on reinforcement learning from the perspectives of

experimental code structure, experimental evaluation indicators, experimental process parameter adjustment, and various algorithm experimental results and performance comparisons. Effect comparison.

4.1 Experiment Details

Algorithm: DQN
Network architecture: neural network, dqn
Optimiser: ReLU
Training Frequency: 200,500,1000
Environment: open-divider, partial-divider, full-divider
Batch: 50, 100:
Evaluation: award

4.2 Effect of Simulated Environment on Performance

The experiment uses the same double DQN network structure. This paper discusses the computational performance of network architectures by comparing different kitchen preset simulation environments.

Through experiments (Fig.9), the article found that the complexity of the simulation environment does affect the coordinated operation of multi-agents. As the simulation environment becomes more complex, from open-divider to full-divider, the system runs longer and longer.

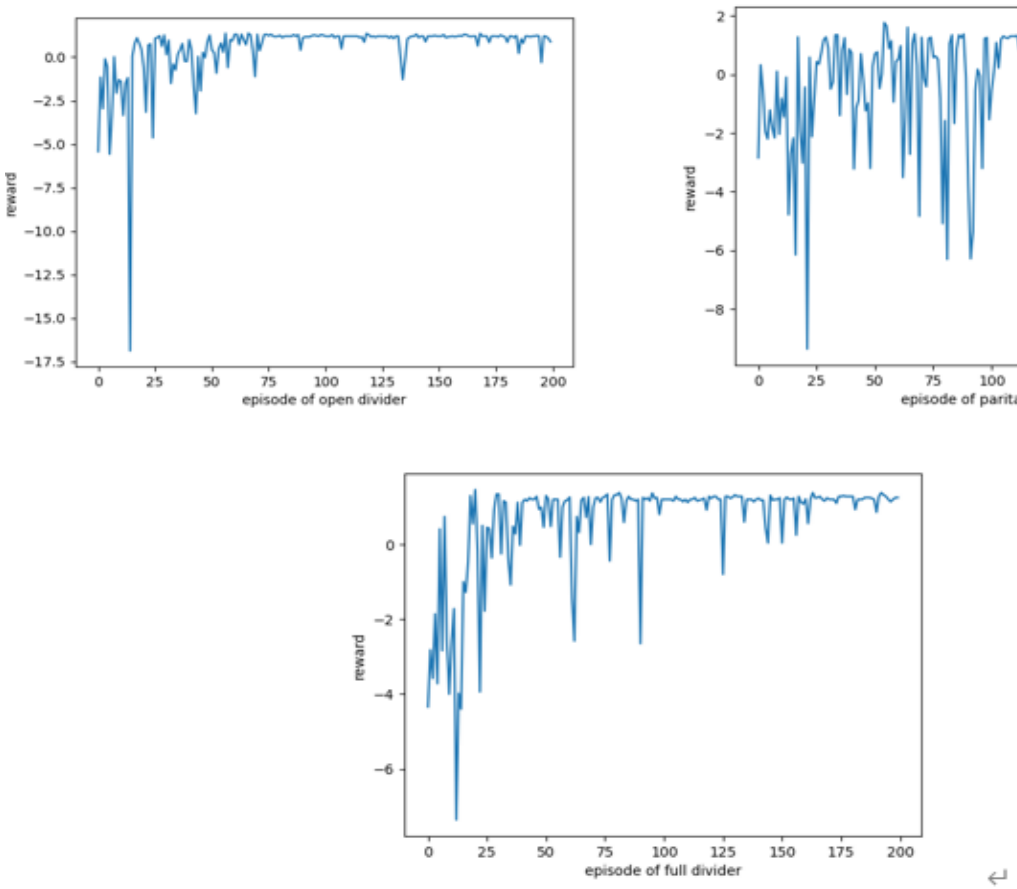

Fig 9. Performance of different Simulation environment

4.3 Effect of Episodes on Performance

The experiment uses the same double DQN network structure and preset simulation environment, compares different episodes, and discusses the impact of episodes.

Through experiments (Fig.10), the article found that the number of episodes affects the reward of multi-agent coordination. As the episode increases, the reward also shows an increasing trend. However, as the episode continues to increase, the size of the reward increase is limited and tends to a fixed value. This may be due to the overall reward cap of the reward structure.

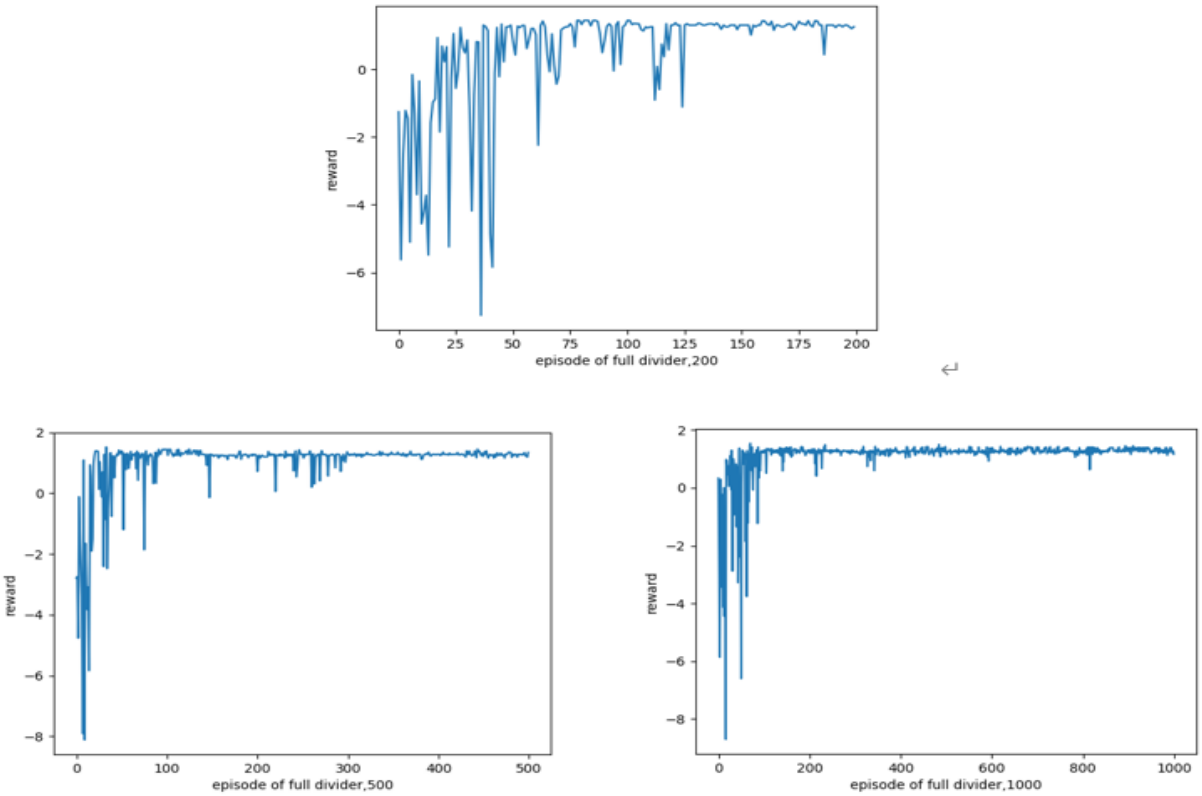

Fig 10. DQN model train network shows the relationship between episode and reward.

4.4 Effect of Hyperparameters on Performance

Through experiments (Fig.11), the article also found that the system performance can be effectively optimised by adjusting the hyperparameter configuration of the network. For example, under the same simulation environment, the agent's overall reward time is reduced by adjusting the batch size. This may be related to the number of states in the discrete state space of the simulation environment.

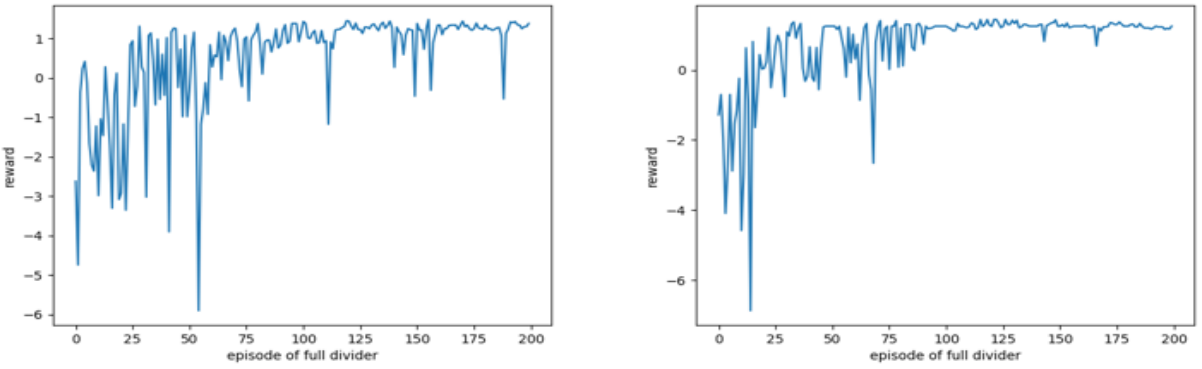

Fig 11. Effect of hyperparameters on performance

In general, the article has obtained basic experimental results by building a DQN network and experimenting. The basic goal is accomplished.

4.5 Discussion

This article builds a Double DQN network. The network can be used for multi-agents to coordinate different cooking tasks. The network optimises problems such as high q values in general DQN networks by using the target network and evaluation network [4]. This paper uses the same DQN network structure to process three different simulation environments, and analyses and compares the running results. Overall, this paper preliminarily builds a DQN network for optimising multi-agent coordinated cooking. Data and graphs show that the results are starting to pay off. However, limited by computing resources and time constraints, the built DQN network still needs further optimisation and adjustment.

DQN networks are generally applicable to 'very large grids' or 'continuous state spaces'. Due to the small discrete state space of the grid in this paper, the effect and validity of using the DQN network model needs further analysis. In addition, in the simulation environment, subtasks such as cutting vegetables are just a binary change of a parameter in the simulation environment, but in the actual environment, the actual scene that needs to be faced when deploying robots for coordinated operation is still relatively complicated.

In addition, the DQN network built in this paper is relatively simple and still needs further optimisation and adjustment. Different strategies for agents to collect experience can be further analyzed comparatively. The article considers that the adjustment of the network structure is carried out through the framework of the SLM Lab platform. 'Too many cooks' did not specifically introduce the actual steps of trying to use DQN. This article considers how to optimise the addition of reward structure.

5. Conclusion

This article discusses the five algorithms mentioned by 'too many cooks'. The general usage scenarios of the five algorithms are analysed from the perspective of principle. This paper also constructs a DQN network to try to optimise the task of multi-agent coordination cooking, and discusses the feasibility analysis and preliminary preparation of dqn applied to this environment. The previous article mentioned that when conducting experiments in too many cooks, due to limited hardware computing resource preset conditions, effective dqn end-to-end optimisation was not achieved. This paper first constructs a neural network (DNN) through PyTorch. Through the DNN network, the parameterised function $v_W(x_s)$ outputs estimated action values $q_w(x_s, a)$. When using the neural network for the model outputs $q_w(x_s, a)$, two problems arise. First, the output of other states may be affected by the update for the state-action pair. In addition, the samples from episodes form a sequence of transitions when training a Neural network. In order to solve these problems, this paper also uses 'replay memory' to allow the agent to learn from recently collected experience. Furthermore, this paper evaluates the performance of the same DQN network in three different environments (open-divider, partial-divider and full-divider).

References

[1] Arup Kumar Sadhu and Konar, A.. Multi-Agent Coordination. The second international joint conference, 2003, 349-361.

[2] Sanghi, N. Introduction to Reinforcement Learning. Deep Reinforcement Learning with Python, 2021.1–17. doi:10.1007/978-1-4842-6809-4_1.

[3] Busoniu, L., Babuska, R. and De Schutter, B. Multi-Agent Reinforcement Learning: A Survey. 2006 9th International Conference on Control, Automation, Robotics and Vision. 2006, 14(2):33-42.

[4] Raschka, S., Liu, Y. (Hayden), Mirjalili, V. and Dzhulgakov, D. Machine Learning with Pytorch and Scikit-Learn: Develop Machine Learning and Deep Learning Models with Python. Birmingham: Packt Publishing,2022, 581-597.

[5] Graesser, L. and Wah Loon Keng. Foundations of Deep Reinforcement Learning. Addison-Wesley Professional. 2019 6(212):89-102.

[6] Gronauer, S. and Diepold, K. Multi-agent deep reinforcement learning: a survey. Artificial Intelligence Review, 2021, 55(2),895–943.

[7] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M.. Playing Atari with Deep Reinforcement Learning. 2013, arXiv.org. doi:10.48550/arXiv.1312.5602.

[8] Wu, S.A., Wang, R.E., Evans, J.A., Tenenbaum, J.B., Parkes, D.C. and Kleiman-Weiner, M. Too Many Cooks: Bayesian Inference for Coordinating Multi-Agent Collaboration. Topics in Cognitive Science, 2012, 13(2), 414–432.

[9] Buşoniu, L., Babuška, R. and De Schutter, B. Multi-agent Reinforcement Learning: An Overview. Innovations in Multi-Agent Systems and Applications, 2010, 183–221.

[10] Raja G, Kottursamy K , Dev K , et al. Blockchain-Integrated Multiagent Deep Reinforcement Learning for Securing Cooperative Adaptive Cruise Control. IEEE transactions on intelligent transportation systems, 2022(7):23.