AY2019 Master Thesis

Cooperative Behavior on Limited Resource Using Deep Reinforcement Learning in Multi-Agent System

LI Yining

A Thesis Submitted to the Department of Computer Science and Communications Engineering, the Graduate School of Fundamental Science and Engineering of Waseda University in Partial Fulfillment of the Requirements for the Degree of Master of Engineering

Student ID	5118FG14-1
Date of Submission	January 29th, 2020
Advisor	Prof. Toshiharu Sugawara
Research Guidance	Intelligent Software

Contents

Abstract

We studied a group of cooperative agents' behavior when there are conflicts between their actions occurring in the environment. All the agents try to achieve a common goal but if their needed resources are scarce, it will results in system conflicts. We make use of deep reinforcement learning (DRL) to make agents in the distributed system learn how to avoid conflicts automatically without any prior knowledge on overall cooperation. The DRL is achieved by training a deep neural network that simulates the Q function of reinforcement learning, and the deep neural network that simulates Q function is termed as DQN. Our goal is to optimize the system's overall performance which is measured by agents' total reward in one episode. In this paper,we conducted experiments by comparing system's performances under different environmental settings, including agent numbers, task density, conflict density and other neural network parameters, various types of DQN design and so on. We also analyse agents' learning behaviors from different perspectives. In this paper, we also introduced specific experimental implementation especially the use of deep learning framework – Keras. In our experiments, the results can prove that even though the self-autonomous agents are doing tasks in a distributed environment and with decentralized control, they can still work cooperatively and optimize the overall performance with an appropriate DQN deisgn. After sufficient training episodes, the deep Q-network can converge to a stable state even though the environment is dynamic, which also denotes the reasonability and success of our simulation environment's design.

1 Introduction

Multi-agent system (MAS)[18], which is defined as a system that comprises two or more agents, which cooperate with each other while achieving local goals[1], is an extending field of classical fields like game theory and decentralized control with modern approaches like machine learning[2]. The word ”agent” has many meanings as well as many forms, but we generally discuss about those parts with autonomous and intelligent properties in a system. MAS is a more natural alternative to build intelligent systems, and thus it gives a solution to the current complex real world problems that need to be solved. MAS is also a more realistic abstraction model of our real world comparing to single agent model which only study individual behaviors. So single agent system is not suitable for studying communication, interaction and interference among different parts of a system. The emergence of MAS study comes from growing computational need, because modeling and computation tasks are becoming much more complex as the size continues to increase, and thus it is laborious and difficult to handle using centralized methods[3] or approaches for single agent system. There are a wide range of real world applications deriving from the study of MAS, for example, distributed intelligent electronic devices, distributed energy resources, cloud computing, Internet of Things (IoT), distributed control, resource management, collaborative decision support systems, data mining, etc. Furthermore, the tendency of using MAS to develop applications is increasing every year due to the world’s explosive growth of data.

However, with all these benefits of MAS, there are a lot of unsolved problems and are generally even more complicated than the similar cases in a single agent system. This is mainly because of the complex nature of MAS. On the other hand, the current technology is still far from achieving many commercial-product-level applications. Furthermore it’s even harder than a single agent system when implementing a multi-agent environment using same algorithm in a program. This is because asynchronization and synchronization among agent objects for computing on a variable should be carefully considered in order to obtain the correct simulation environment. Generally speaking, there are two types of control approaches for MAS: centralized and decentralized control [4]. By having centralized control, the global information of the environment has been used to calculate the path, trajectory or position of the agents before all [5, 6, 7, 8, 9, 10]. However, in centralized control the computation will become high since there is only one centralized processor that control over all of the system. That’s why decentralized control is needed in order to reduce computing load. Decentralized control has a good scalability, adaptivity and flexibility in a complex system. In the decentralized control strategy, each part works freely utilizing its local information so removing or adding a new part makes no influence on other parts' self-control. However, in decentralized control, the main problem is the task has to be distributed in a robust an efficient manner to ensure that every agent is able to perform its individual task cooperatively with another agents to achieve certain target [4]. However, no matter it's centralized or decentralized control to design a MAS, system conflict is a common problem that affects the whole system's performance and efficiency significantly. In a real MAS application, which usually possess a large amount of resources, system conflicts can cause a huge loss if there is no handling for this, so we conducted this research mainly focusing on how to reduce system conflict in MAS. In multi-agent environments, when agents work cooperatively, they will select some policies or strategies to eliminate conflicts among agents. Such strategies include negotiation, arbitration, and voting. In many reality situations, the system common shared resource is often not allocated to every agent or even sparser comparing to agents' requests. For example, when a large amount of http requests are all sent to a server, there will be serious bandwidth congestion, which largely damage the network's performance. So in such systems, conflicts are inevitably to happen by reason of the fact that system's common resource is shared among agents, and especially when agents are under decentralized control, each agent tries to optimize its own performance, the problem is even worse. If there is no proper regulation, it's very hard for the whole system to work as expected.

2 Related Work

As is mentioned above, system conflict can not only reduce efficiency, but also can cause huge economic losses in a reality application. Many researchers are studying how to reduce conflicts in a large distributed system and their study mainly focus on path-planning strategy or negotiation. Jingjin Yu *et al.* [11] studied the problem of optimal multi-robot path planning on graphs focusing on structural and computational complexity issues. Wolfgang *et al.* [12] put forward a method named Multi-Agent Path-Finding(MAPF) to reduce conflicts in MAS. Their approach is based on creating a search forest (CBS-TA) on demand and focuses on path planning dynamically according to CBS-TA and got complete and optimal results.

Path-planning method is a good way to solve conflict problems, however, such approaches are primarily implemented in a supervised learning scenario, which require large efforts on algorithm design mathematically and labeled data collecting. However, most MAS environments have countless states and is hard to label training data. Even those simple as a Markov decision process (MDP) environment can have extremely large amounts of scenarios that is not easy to get the correct path option. In a complicated and dynamic MAS environment, reinforcement learning seems to be a more practical solution because it is aimed to implement human-level control in a real world [13]. Recent years have seen more research on multi-agent reinforcement learning (MARL) with good results. However, natural extension from single agent environment's methods to MARL has been proven failure [14, 15] , which is mainly due to the high-dimensional state-action space that has to be explored by agents.

Recently, deep Q-learning has been applied to multi-agent systems to achieve cooperation and coordination behaviors. For example, Elhadji Amadou Oury Diallo and Toshiharu Sugawara[16] proposed a n fully end-to-end learning method to study how to make agents minimize their encounters with opponents by strategically forming a group and by combining their knowledge of the environment to maximize the probability of avoiding opponents and obstacles by using Deep Q-network (DQN). Yuki Miyashita *et al.* [17] researched on how agents with their independent DQN cooperate in a distributed system.

However, these research are all facing such a common problem in MARL that because other agents are also part of the environment to some agent, it's not easy to balance the environmental dynamics and Markov assumptions required for the convergence of Q-learning algorithms such as DQN. A number of algorithms have been proposed that help address this issue, e.g., LOLA, RIAL, and Q-MIX. At a high level, these algorithms take into account the actions of the other agents during RL training, usually by being partially centralized for training, but decentralized during execution. Implementation wise, this means that the policy networks may have dependencies on each other[25]. Similarly, policy-gradient algorithms like A3C and PPO may struggle in multi-agent settings, as the credit assignment problem becomes increasingly harder with more agents. Therefore, in spite of the current success in some MARL research, there are still a alot more waiting to be explored in this field.

3 Purpose of this research

In this paper, we intended to study how to reduce the conflicts as many as possible by a group of autonomous and cooperative agents' learning as well as what kind of cooperative behaviors these agents can learn in a distributed system. At the same time, we aim to implement such ideas in a fully automatic way. There is no human interfere in the process where agents executing tasks in the system. Our goal is to train agents to learn a stable conflict-avoidance policy in a dynamic environment by themselves. Based on previous research, we primarily use DQN as our baseline algorithm to study how agents react when conflicts occur so as to coordinate with other agents and achieve cooperation gooals in the system. We also want to understand why and how agents can learn a good policy by observing the environment from a deeper perspective. We also want to investigate how deep reinforcement learning and convolutional neural network help agents find a stable pattern and learn strategies online in a highly dynamic environment. The study also sought explanation as to why different policies are learned when conflict density is changed. Finally, we want to examine how DQN parameters affect agents learning ability and attempt to investigate how to obtain the finest-tuning set of parameters.

4 DQN in multi-agent system and the implementation

In this chapter, we will go to more details about DQN and multi-agent system's implementation, and how to abstract the real world into a model. We will also introduce the difficulties and challenges in this research.

4.1 Multi-agent reinforcement learning

As is mentioned above, MAS is a complex and dynamic system which makes it difficult or even impossible to solve tasks with preprogrammed methods. Moreover, in an environment that changes over time, a hardwired behavior may become inappropriate. So it's more natural to make agents discover a solution on their own using learning. In addition to benefits owing to the distributed nature of the multi-agent solution, such as the speed-up is made possible by parallel computation, multiple RL agents can also harness new benefits from sharing experience, e.g., by communication, teaching, or imitation[19]. Even so, RL still has more advantage over other methods for the problem of dynamic environment, and together with the simplicity and generality of the setting, this makes reinforcement learning attractive also multi-agent system.

A reinforcement learning (RL) agent learns by interacting with its dynamic environment [20]. Even though RL's feedback is less informative than supervised learning methods, it's more informative than unsupervised methods. So in this case where supervised learning cannot be applied, RL is a better solution to such problems. Reinforcement learning has two large categories considering the environment state: continuous and discrete reinforcement learning. In this paper, we see the problem as discrete reinforcement learning problem. Starting from single agent case, the concrete RL procedure is : at each time step, the agent perceives the state of the environment and takes an action, which causes the environment to transit into a new state. A scalar reward signal evaluates the quality of each transition, and the agent's goal is to maximize the cumulative reward during the whole process.

A single agent RL process can be modeled as the Markov decision process(MDP). A MDP is a discrete time stochastic control process, which provides a math framework for modeling many real world decision situations. A finite Markov decision process is a tuple $\langle S, A, f, \rho \rangle$, where S is the finite set of environment states, A is the finite set of agent actions, $f : S \times A \times S \to [\,0, 1]$ is the state transition probability function, and

$\rho : S \times A \times S \to \mathbb{R}$ is the reward function. The behavior of the agent is described by its policy, which specifies how the agent chooses its actions given the state. So the goal of reinforcement learning to make agents take actions with respect to different states in the process of interacting with environment in order to maximize the expected discounted return till the terminal state. The expected discounted return R from time step t is defined as :

$$R^h(s) = E\{\sum_{t'=t}^{T} \gamma^{t'-t} r_{t'+1} | s_0 = s, h\}$$

h : policy

T : assume total time step is T

γ : discount rate

r : reward for each transition

s, s_0 : state

Multi-agent reinforcement learning (MARL) can be extended from the single agent reinforcement learning. To a certain agent, its environment includes both the non-agent part and other agents. And thus, the formula of expected cumulative reward R for MARL is rewrite as:

$$R^h{}_i(s) = E\{\sum_{t'=t}^{T} \gamma^{t'-t} r_{i,t'+1} | s_0 = s, h\}$$

The core concept of RL is Q function which is used to estimate the optimal Q-value of each state and the value of action for a specific state, and thus this algorithm is named as Q-learning. The optimal Q-value of a state s and action a is the maximum discounted cumulative reward that the agent can receive after taking action a in state s:

$$Q^*(s,a) = max_\pi E\{\sum_{t'=t}^{T} \gamma^{t'-t} r_{t'+1} | s_0 = s, a_0 = a, h\}$$

Similar to the extension on the formula of R from single agent to multi-agent, Q function in multi-agent environment is rewrited as :

$$Q^*{}_i(s,a) = max_\pi E\{\sum_{t'=t}^{T} \gamma^{t'-t} r_{i,t'+1} | s_0 = s, a_0 = a, h\}$$

As one RL algorithm, Q-learning is a method for estimating the optimal Q^* that does not require any knowledge of the transition model. In Q-learning the agents repeatedly interacts with the environment and tries to estimate Q^* by trial and error. There are two types of solutions for Q function : Monte-Carlo and Temporal Difference.

However, with all these benefits of RL in MAS, the difficulty cannot be ignored as well. This is due to the computation dimensionality of Q values which has exponential growth of the discrete state-action space in the number of state and action variables (dimensions). Because basic RL algorithms like Q-learning estimate values for each possible discrete state or state-action pair, this growth leads directly to an exponential increase of their computational complexity. The complexity of MARL is exponential also in the number of agents, because each agent adds its own variables to the joint state-action space. This makes the curse of dimensionality more severe in MARL than in single-agent RL [19]. So this is where deep Q-network is introduced, because we can get estimated Q values from a deep neural network so that the optimal state-action value can be obtained.

4.2 Deep Q-network

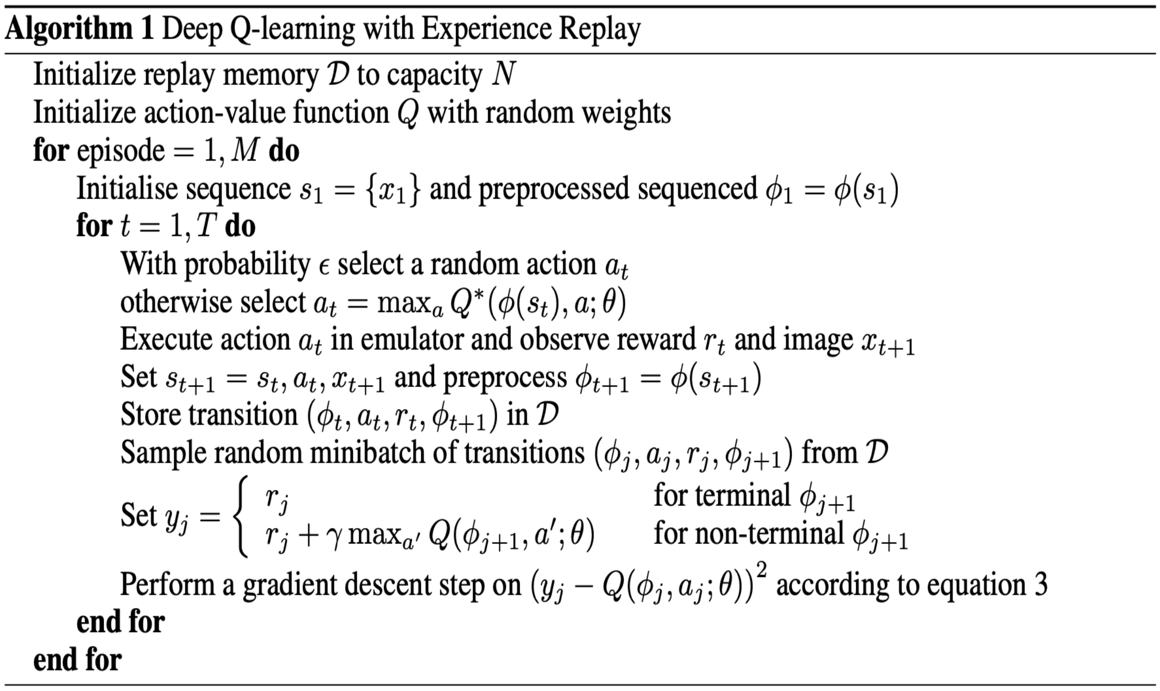
As is talked above, if we want to get the Q values from a high-dimension RL model, using deep neural network is a good choice. Deep learning has made it possible to extract high-level features from raw sensory data, leading to breakthroughs in computer vision and speech recognition[21, 22, 23]. DQN is a model-free approach to Reinforcement Learning based on Deep Neural Networks for estimating the Q-function over high-dimensional and complex state space. DQN is parameterized by a set of network weights θ , which can be updated by a variety of RL algorithms. The parameters θ are learned by gradient descent or other optimization methods which iteratively minimizes the loss function $L(\theta)$ using samples (s, a, r, s').

At time t, to get optimal Q-value approximation from the network, parameters $\theta_{i,t}$ in the network of agent i are updated to minimize the loss function $L_{i,t}(\theta_{i,t})$.

There are also many choices for loss functions, and each loss function is optimal for different problems. The most commonly used is mean square loss function, and in this problem, its form is defined as:

$$L_{i,t}(\theta_{i,t}) = \mathbb{E}_{(s_i,a_i,r_i,s'_i)}[(r_i + \gamma \max_{a'_i} Q_i(s'_i, a'_i; \theta^-_{i,t}) - Q_i(s_i, a_i; \theta_{i,t}))^2],$$

While in our experiment, we use the Huber loss. The Huber loss acts like the mean squared error when the error is small, but like the mean absolute error when the error is large. Note that in this paper we use the Double DQN, target network parameters $\theta^-_{i,t}$ are copied from θ periodically to stabilize learning for deep Q-network; The algorithm is shown as Fig1.



Algorithm 1 Deep Q-learning with Experience Replay

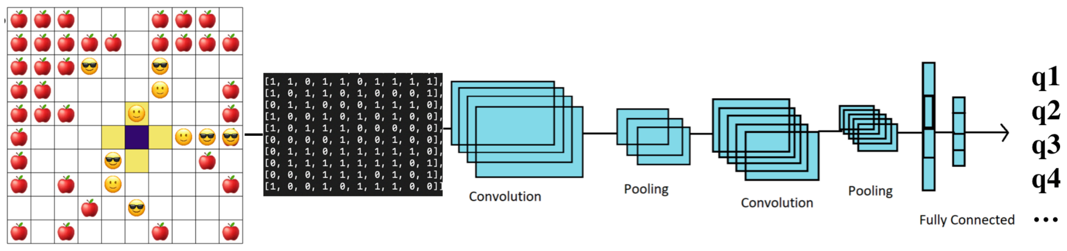
```
Initialize replay memory $\mathcal{D}$ to capacity $N$
Initialize action-value function $Q$ with random weights
for episode = $1, M$ do
    Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for $t = 1, T$ do
        With probability $\epsilon$ select a random action $a_t$
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
    end for
end for
```

Figure 1: DQN with experience replay
https://www.cs.toronto.edu/ vmnih/docs/dqn.pdf

4.3 Convolutional neural network

Neural Network has a long history in studying artificial intelligence and is still a cutting-edge field in AI, mainly because of the rise of deep neural network. In theory, a deep neural network can approximate any function as long as more layers are added. Sample data's different local features are extracted when they passing through different layers of a DNN. Convolutional Neural Network(CNN) is such a neural network that points at input samples with spacial local features, and often used in image recognition.

In this paper, we mainly adopt CNN for the main part of the structure of DQN, but the option is flexible, depending on specific problem. And there are also some researches are using LSTM as the main layer and get good results[24]. But there are various benefits that make CNN suitable for such dynamic environment. One reason is that the model environment is a grid world which can be seen as an image, and the input matrices can be seen as the input pixels similar to image processing while using CNN.



```
[1, 1, 0, 1, 1, 0, 1, 1, 1, 1],
[1, 0, 1, 1, 0, 1, 0, 0, 0, 1],
[0, 1, 1, 0, 0, 0, 1, 1, 1, 0],
[1, 0, 0, 1, 0, 1, 0, 1, 0, 0],
[1, 0, 1, 1, 1, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 1, 0, 0, 1, 0, 0],
[0, 1, 1, 0, 1, 1, 1, 1, 1, 0],
[0, 1, 1, 1, 1, 1, 1, 1, 0, 1],
[0, 0, 0, 1, 1, 1, 0, 1, 0, 1],
[1, 0, 0, 1, 0, 1, 1, 1, 0, 0]]
```

Figure 2: An example showing data flow in this model

In addition to the convolutional layers that we have just described, convolutional neural networks accompany the convolution layer with pooling layers, which are usually applied immediately after the convolutional layers. There are several ways to condense the information, the commonly used methods are max-pooling, which as a value keeps the maximum value of those that were in the input window or use the average value in the input window.

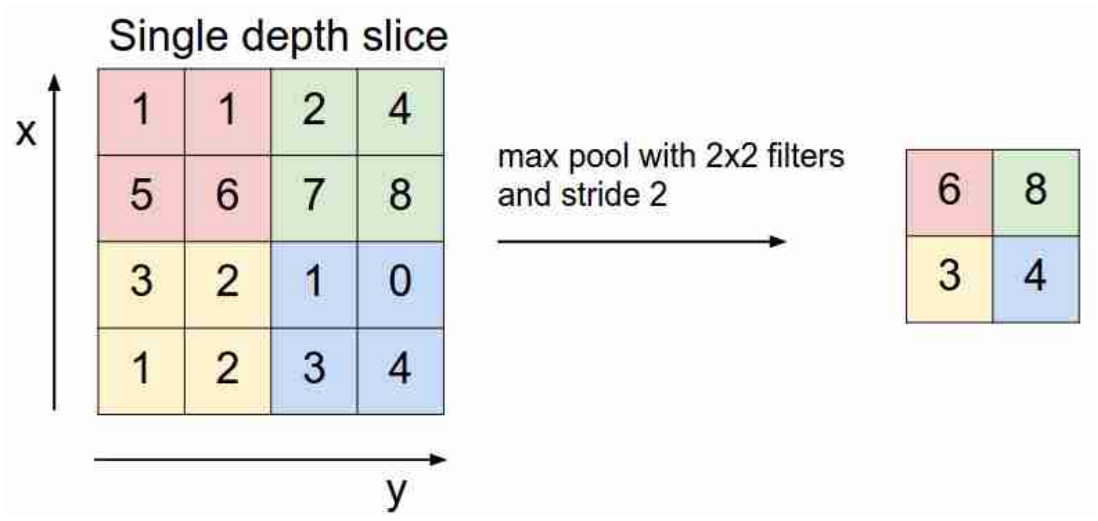


Figure 3: max-pooling
https://images.app.goo.gl/MYYz9xGURfb7sGoj7

In theory, a deep neural network can approximate any function as long as more layers are added. Sample data's different local features are extracted when they passing through different layers of a neural network. Convolutional Neural Network(CNN) is such a neural network that points at input samples with spacial local features, and often used in image recognition, recommender systems and natural language processing. In this problem, the input of states is actually a batch of two dimensional matrix, and the nature of this problem is to let neural network recognize different patterns given the environment matrices, therefore it can be seen as an image so this is actually an implicit image classification problem. So CNN approaches can be extended naturally to this problem.

4.4 Implementation by Tensorflow and Keras

There are various open source deep learning frameworks for building neural network in an easier way. The popular ones like Keras, Pytorch, Tensorflow, Theano, Caffe, MXNet are all sharing their own nice features. Among them, Keras is a higher level framework that makes user think less about how backend is implemented and thus focus more on the algorithm itself. Keras also supports backends like Tensorflow, CNTK and Theano.

In the experiment for our research, we use Keras 2.2.4 to implement our experiment neural network structure, and Tensorflow as backend.

There are some advantages of using Keras comparing other frameworks:

- Keras models can be developed with a range of different deep learning backends, and can also be trained on a wide range of different hardware platforms beyond CPUs.

- Keras has built-in support for multi-GPU data parallelism that can speed up computation to a large extent.
- Keras is noot only easy to use, but also easy to change model into products.
- Keras is backed by key companies in the deep learning ecosystem like Google, Microsoft, NVIDIA and AWS.

There is one thing to note that it's very important to make sure what the input tensor's dimension is. In our problem, the input tensor is a 4 dimensional array with the shape (batch, height, width, channel).

- batch: number of samples from replay memory.
- height: the grid world's height size.
- width: the grid world's width size.
- channel: borrowing the concept of image processing, which in our problem is matrix number representing different observing environmental information.

5 Problem Formulation

5.1 Model description

The problem is set in a $N \times N$ grid world (or game board), which could be described by the Fig 5. Emoji shape represents agents, apple shape represent tasks, the centered dark square represents the destination in this environment, which agents should transport a task to. Around the dark destination, there are four yellowish squares representing access points or waiting zones which means when an agent carrying a task arrives at such points and choose the action to execute taks can successfully execute tasks and get a score. We numbered each access point for later discussion, and each access point is numbered as Fig 4.

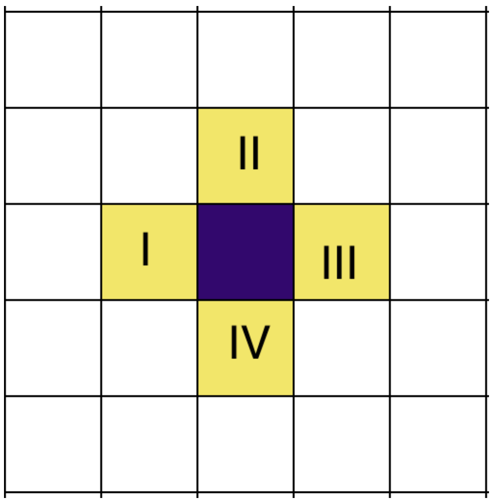


Figure 4: access points number

This model can be described as a game with the following rules:

- At the beginning of game, it's initialized as that agents and tasks are randomly placed
- The destination's position is fixed throughout game (scarce resource)
- There are four access points(waiting zone) around the destination, any agent who arrives at such place is eligible to score(put ball at the Goal), but at any time step, at most one agent can score (use the resource), others who are also eligible can only wait or leave
- Each agent can only carry no more than one task while moving on the board
- Agents can only observe its surrounding area's information within a certain range

- Agents know the position of the destination and can observe the destination's surrounding information within a specific range
- Agents execute tasks with a certain amounts of steps. When step count is larger than the limit, an episode terminates, and another episode begins, and the environment is randomly initialized again.
- Previous episodes are experience and stored in agents' separate memories. Agents use such experience to take actions in a new episode.
- Purpose of this game is to score as many as the agents can (agents work cooperatively to maximize total reward)

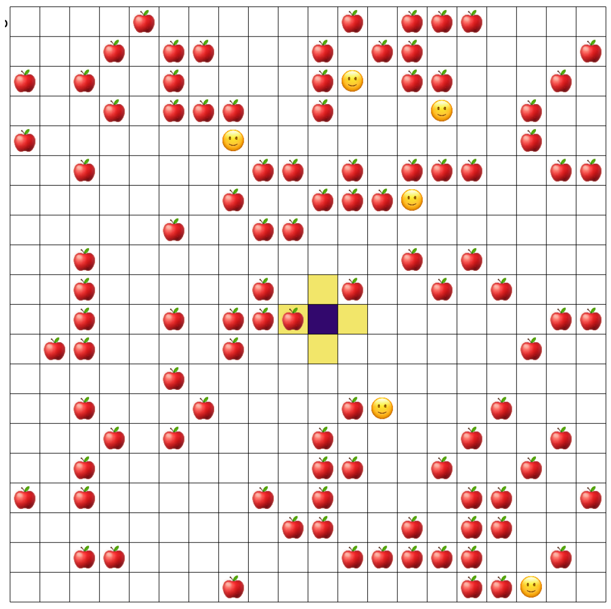


Figure 5: model environment demo

From a mathematical perspective, the whole environment for this problem can be defined as tuple $\langle I, m, N, \{A_i\}, \{\Omega_i\}\rangle$, where $I = \{1, \cdots, n\}$ is a set of finite number of agents, m is the number of tasks, N is the edge size of the environment, A_i is the action set of agent i, $\{A_i\}$ denotes all the agents' action set in the environment, and Ω_i is the observation space of agent i.

In our model, all the agents are independent and self-autonomous with its own Deep Q-network, so we don't take joint action into account, for it is unnecessary and inducing unnecessary complexities. The cooperative behavior is represented by reward function. In later part of this paper, we will introduce more details on this reward scheme . Because the grid world environment is a discrete reinforcement learning model, so we introduce discrete time t = 0,1,2,3,... as discrete execution steps which is also compatible with MDP.

Action : We define the action to "collect a task" as when an agent without any tasks(in the figure represented as apple) in hand (empty agents) arrives at the cell within which an apple is located in. Similarly we define the action to "execute the task (take an apple to the destination)" as when an agent who is carrying an apple in hand (non-empty) arrives at the access points. The possible actions of an agent can be modeled as *UP, DOWN, LEFT, RIGHT*.

Observation : each agent has only a limited local view (see Fig 6) around itself. More specifically, the local observation includes tasks' positions. However,the position of the destination is treated as global knowledge as well as the all the agents' locations. Such setting can help agents focus on collecting tasks when it's empty and after it has collected one task, it can head to the destination to execute task and try to avoid conflicts.

5.1.1 Reward scheme

In reinforcement learning, how to design a reward to each state is the key to help agents learn correct behavior. In our experiment, the reward is set as below:

At each step:

- if agent moves to an empty cell, then it gets negative reward $r = -0.001$;
- if agent carrying a task moves to a cell with a task in it, it will ignore the task, $r = 0$;
- if agent carrying a task moves to the waiting zone, and is able to successfully execute it, then it will get a reward $r = 1.0$
- if agent turns into waiting status, then it will get a reward $r = -0.005$

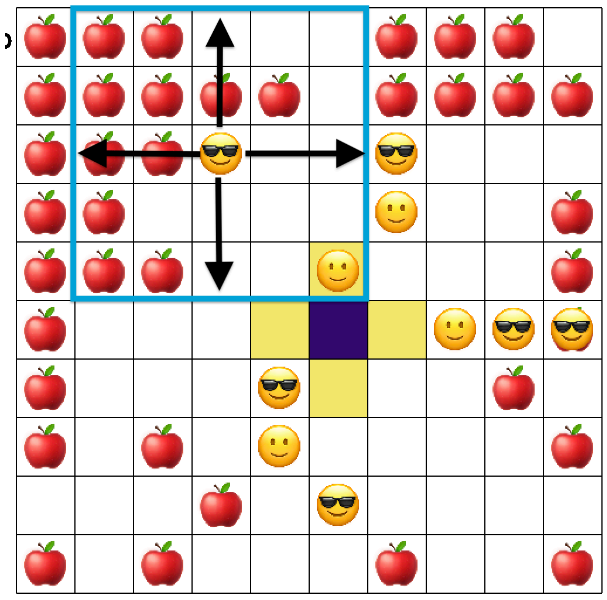


Figure 6: partial observation 6

As we can see from the above setting, when agents turn into waiting status, they can get a relatively larger punishment (negative reward). This is because we want to reduce the waiting time as well as reduce the cases when some agent just occupy the access point but do nothing, which results in deadlock on resource. So the longer one agent chooses to wait, the more negative it will get which is incompatible with DQN's optimization.

5.1.2 Conflict case

To simplify the problem, we primarily consider such scenarios as the main conflict: when more than one agent reach the access points, only one can successfully execute the task while others can choose either to wait or leave (gaining different rewards). And if agents choose to wait, then the access point is seen as waiting zone. The number of agents in the waiting zone is seen as the **conflict count**.

For example, Fig 7 is one of the steps that 3 agents with tasks are trying to use the common resource at the same time. But the next step, shown as Fig 8 shows that only one agent successfully executed the task (the agent at I access point)

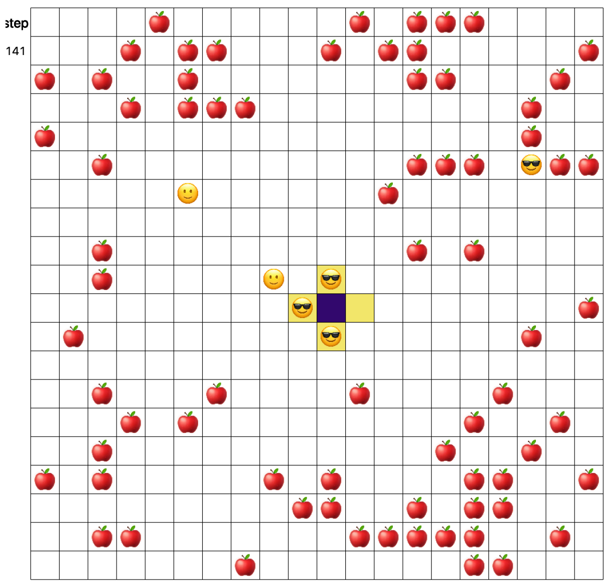


Figure 7: conflict occurs

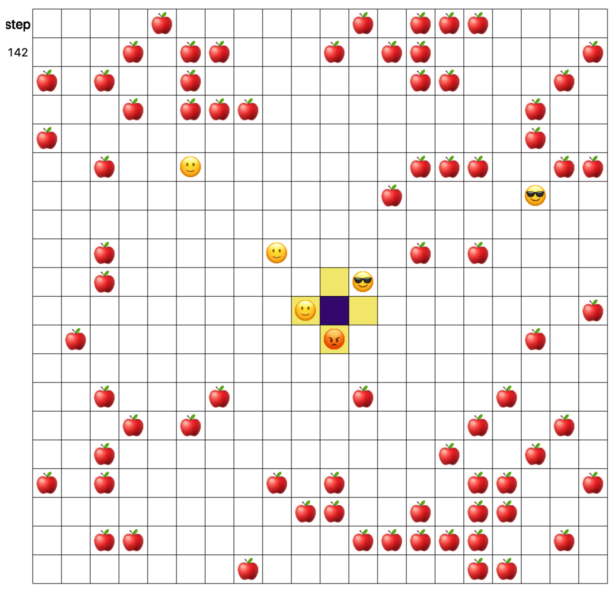


Figure 8: different choices to solve conflict

6 Simulation Model

It's flexible when designing DQN architecture. In our problem, we have two kinds of input matrices with different dimensions, so we will use two input layers with different dimensions as input branches. Then concatenate two intermediate results, and input it to the next layer. In the end, the neural network output 4 Q-values, and agents adopt the action with the largest Q-value to decide next step's action.

6.1 DQN architecture

The specific network architecture is shown as Fig 9. In each branch, input matrices representing observations of one agent both follow the same type of layers with different parameters. They first enter 2 convolutional layers, then down-grade the dimension to a vector, and enter dense layer. Two vectors from the previous dense layers are concatenated and then enter another dense layer to get the final output – 4 scalar Q-values evaluating each action.

We used two convolutional layers for the input based on the nature of the environment. Two convolutional layers can already extract spatial features well based on previous researches, and in our problem, not too many layers can also avoid overfitting.

When combining two kinds of intermediate results with different dimensions, we adopted direct concatenation of these two vectors. Same as DQN structure design, the combination approach is also flexible, and the only difference among these methods is the neural network parameters.

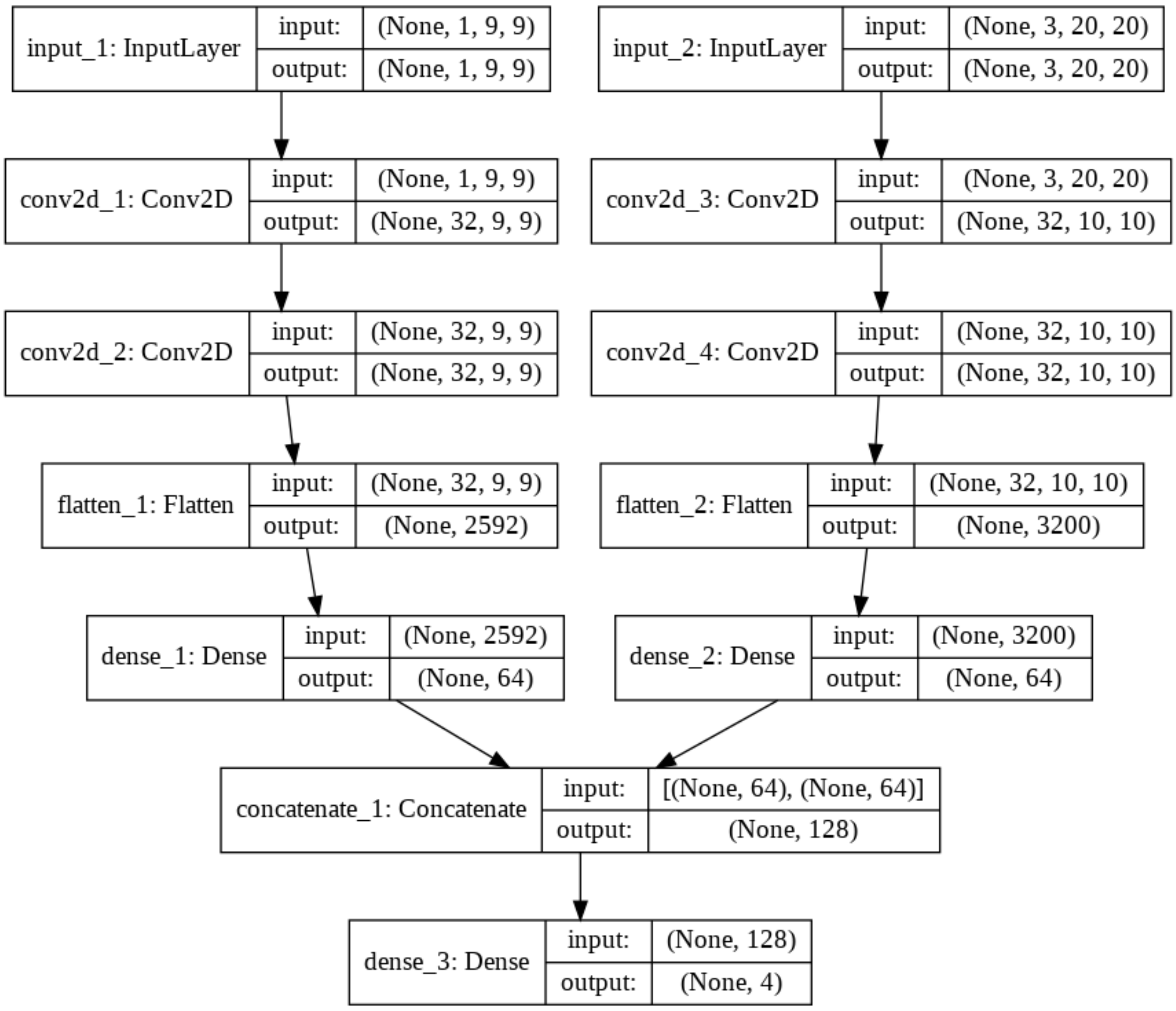


Figure 9: DQN architecture

6.2 Replay memory

In reinforcement learning, the agent observes a stream of experiences and uses each experience to update its internal beliefs. For example, an experience could be a tuple of (state, action, reward, next state), and the agent could use each experience to update its Q function. In the earlier stage of experiment, we also did a trial using Prioritized Experience Replay [26], this method differs from traditional ones because it considers the significance of a transition experience by assigning a weight value to that transition. It shows a much better result in their experiment but in our problem, the improvement is not very large, so in this experiment, we use DDQN with traditional experience replay, in which experience transitions were uniformly sampled from a replay memory, and also has a good effect to avoid overfitting. Agent i stores the experienced data $c_{i,t} = (s_{i,t}, a_{i,t}, r_{i,t}, s_{i,t+1})$ into its own memory $D_{i,t-1} = \{c_{i,t-d}, \cdots, c_{i,t-1}\}$, where $d > 0$ is memory capacity, at t steps. The basic workflow is shown in Fig 10

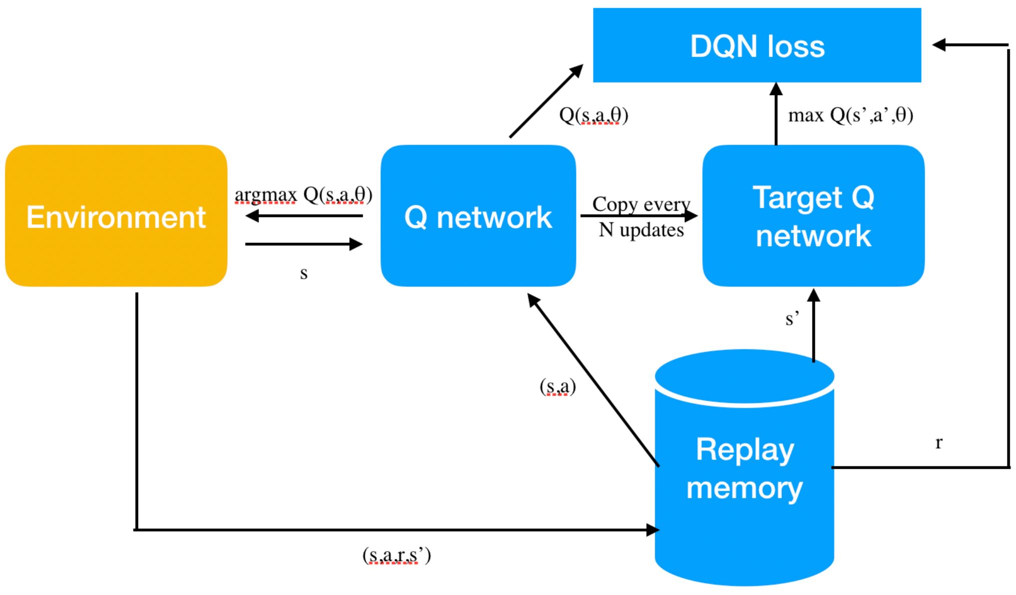


Figure 10: Experience replay

6.3 Epsilon decay

Balancing the ratio of exploration/exploitation is a great challenge in reinforcement learning (RL) that has a great bias on learning time and the quality of learned policies. In this paper, we adopt ϵ exponential decay by episode to optimize learning performance. At the beginning of the training process, the ϵ value is large so that agents work almost randomly and can gather as many experience as they can, while in the later part of training agents mainly make use of their experience because the ϵ value now is very small.

6.4 Optimization

Optimizers update the weight parameters to minimize the loss function. The commonly used optimizers for deep learning are Stochastic Gradient Descent, RMSProp, Adam, Batch Gradient Descent etc. Many DQN related researches are using RMSprop and got fairly good result. But in our problem, we adopt Adam optimizer in training. Adam is an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. The method is straightforward to implement, is computationally efficient, has little memory requirements, and thus this is suitable for minimizing the loss function of our problem.

7 Experiment and Analysis

We primarily conducted two types of experiments with different conflict density in the environment and found that agents in different environment have learned different policies.

7.1 Experiment Data and Parameter

We compared agents' behavior in two types of environment. Agents in both environment are both using the same DQN architecture. One is a sparse environment where fewer conflicts happen.

7.1.1 Sparse environment

Details are shown in Table 1 Sparse environment has fewer agents in a larger grid world. So in such an environment, agents have more opportunities to get a score with fewer obstacles in the way.

Table 1: agent parameter:sparse environment

parameter	value
(height, width)	(20,20)
agent number	6
task number	100
episodes	5000
step per episode	1000
steps update T_{train}	1000
ε decay rate γ_{ε}	0.9992 (max:1, min:0.1)
Discount rate γ_Q	0.95
Adam learning rate η	0.0001
replay memory size	10000
mini batch size	64
θ_t^- update step T_{copy}	100

7.1.2 Conflict-prone environment

We set more agents in a smaller grid world, so in such environment, on the one hand, each agent wants to get more reward for itself, while on the other hand, such individual purpose would result in more conlicts among agents who all want to access to the common resource. The specific setting is shown as Table 2.

Table 2: agent parameter: conflict-prone environment

parameter	value
(height, width)	(10,10)
agent number	10
task number	70
episodes	5000
step per episode	200
steps update T_{train}	1000
ε decay rate γ_ε	0.9992 (max:1, min:0.1)
Discount rate γ_Q	0.95
Adam learning rate η	0.0001
replay memory size	10000
mini batch size	64
θ_t^- update step T_{copy}	100

7.2 Experiment Result Analysis

For both environment, we did 10 independent experiments separately and the following shows the average results on these 10 experiments.

Fig 11 shows at the almost end the game (1000 steps in total), most tasks have been executed. Fig 12 shows how total reward is changing by training in the sparse

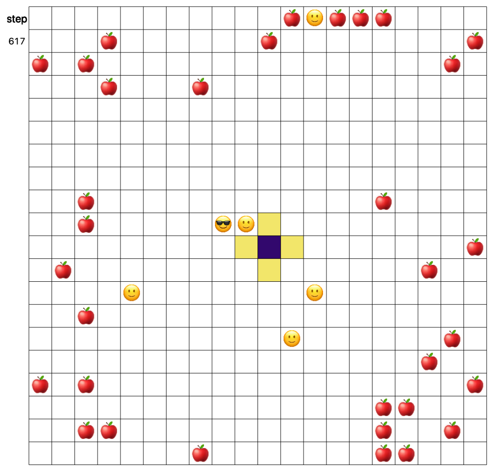


Figure 11: An example showing the end of the game

environment. And the corresponding loss is shown by Fig 13. Fig **??** shows the conflict rate (defined as conflict count divided by finished task number) changing by episode.

As we can see from the results, in later part of training, total reward can almost reach the highest value which means all the tasks in the environment have been executed

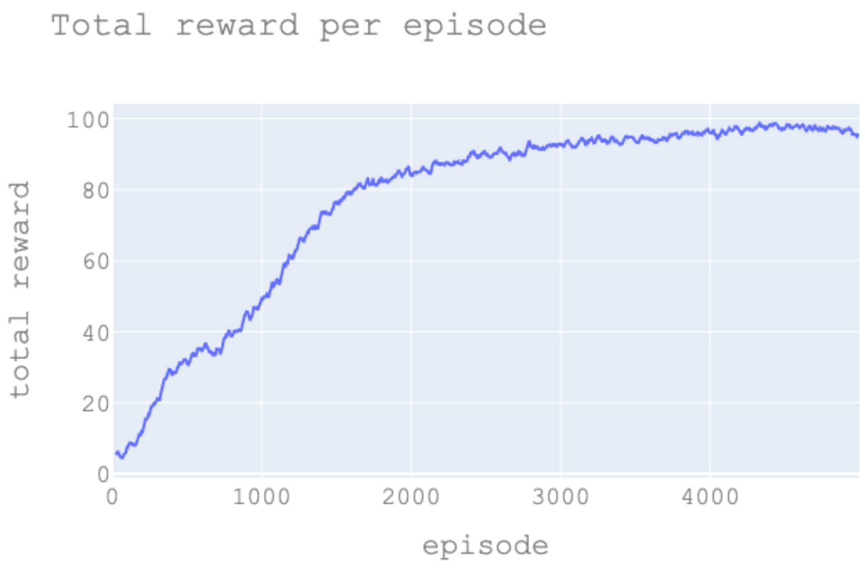


Figure 12: Total reward of sparse environment

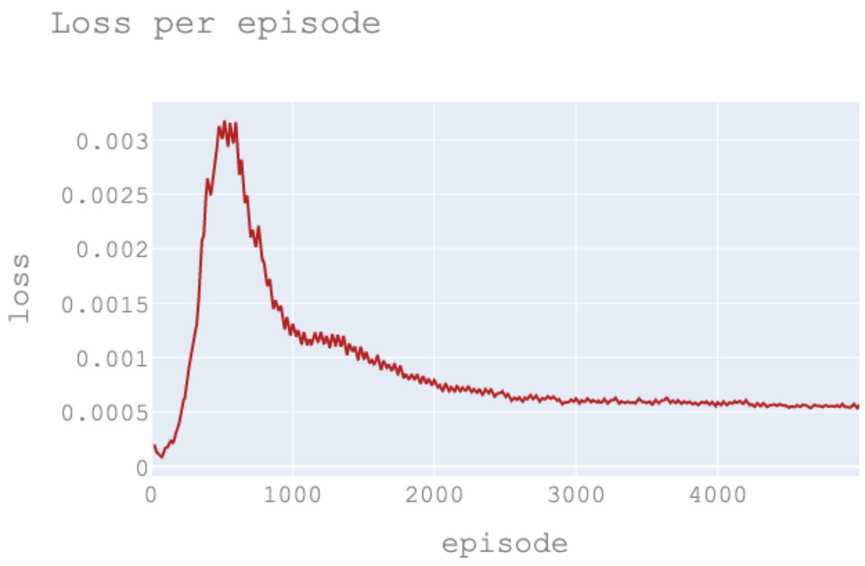


Figure 13: Loss of sparse environment

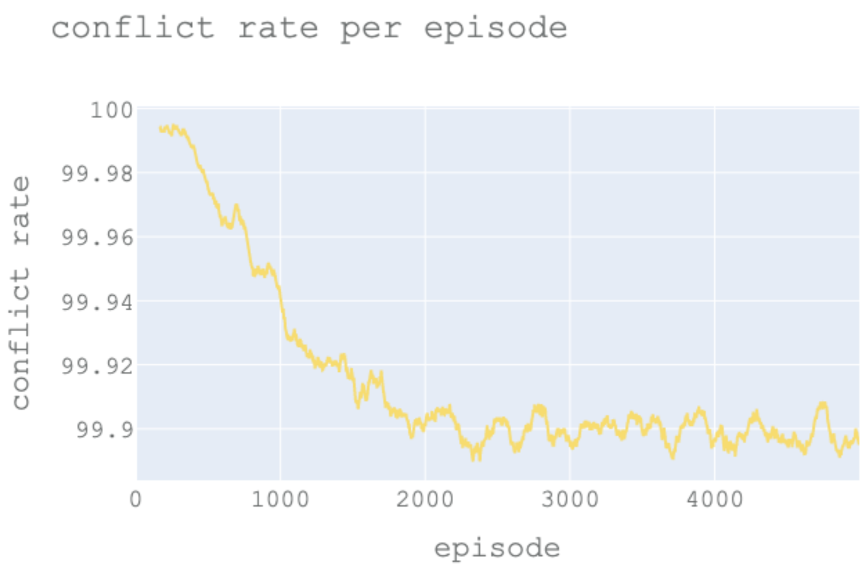


Figure 14: conflict rate of sparse environment

successfully. At the same time, loss value is also turning flat in later episodes while training. At the beginning, loss value is increasing, which makes sense that at early stages, agents are behaving almost randomly, which largely incompatible with the target network.

Next is a summary of different agents choosing different access points when execute their tasks. As we can see from Fig 15, Fig 16, Fig 17, Fig 18 that even though the ϵ is largely different, there is always clear difference on position choices among agents in a certain episode.

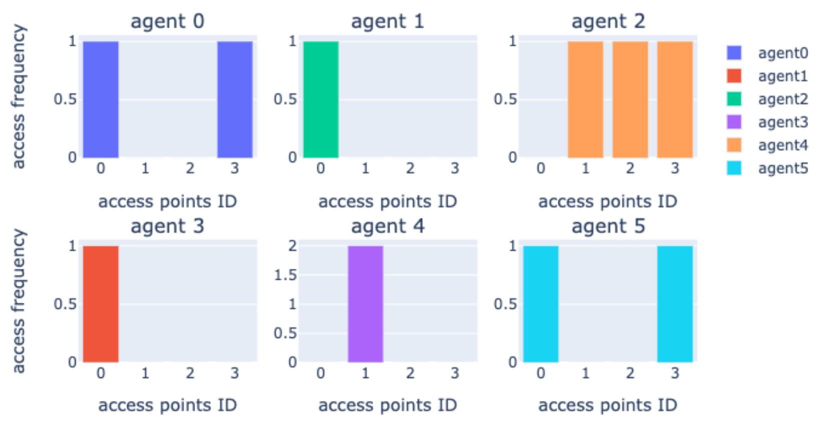


Figure 15: Agent's preference on access point, episode=100

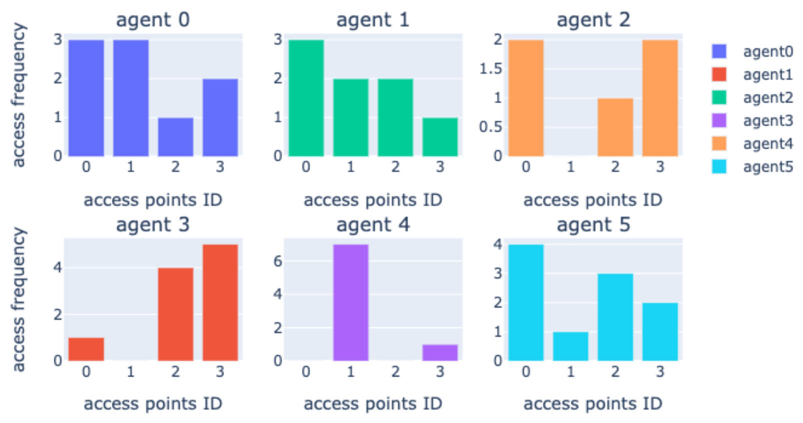


Figure 16: Agent's preference on access point, episode=1000

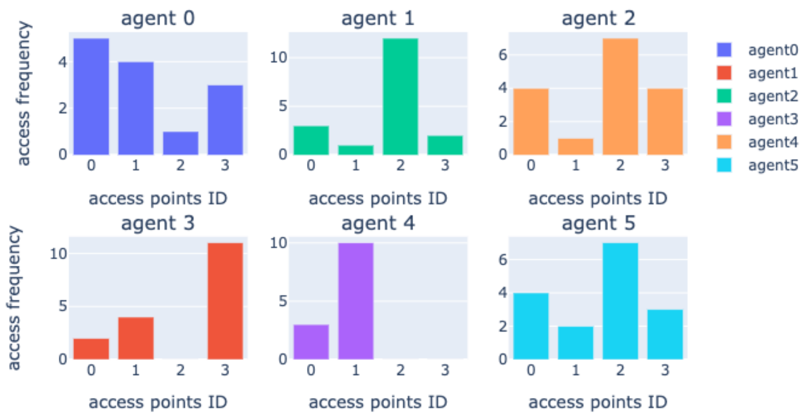


Figure 17: Agent's preference on access point, episode=4000

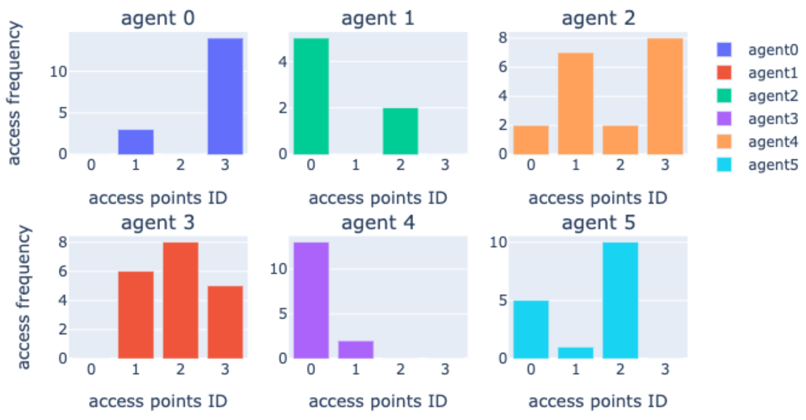


Figure 18: Agent's preference on access point, episode=5000

However, if we see all the agents as a whole, there is no big difference on access points choices, see Fig 19

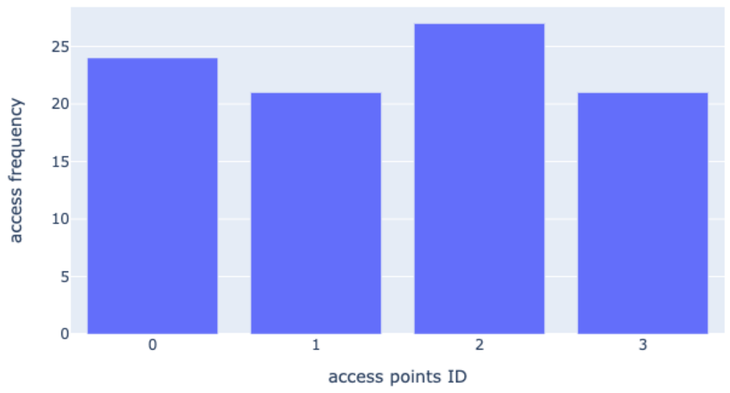


Figure 19: All agents' preference on access point, episode=5000

The following graph Fig 20 shows at what step tasks are executed the most. We can see from this figure that agents can finish almost all the tasks within about 500 steps in an episode.

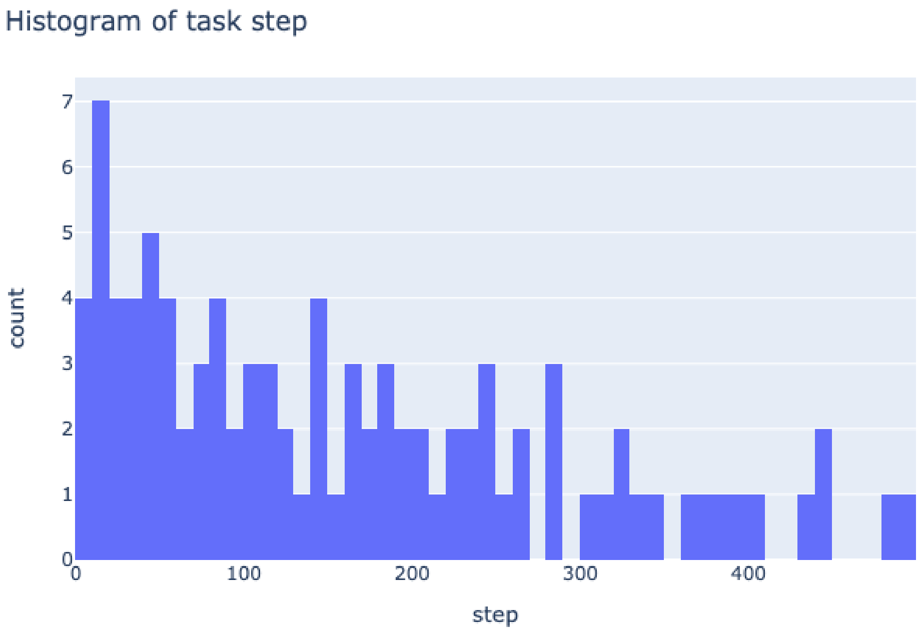


Figure 20: Step distribution of task execution, N=20

In the conflict-prone environment, when $N = 10$, agent number is 10, total reward plot is similar to the sparse environment, see Fig 21

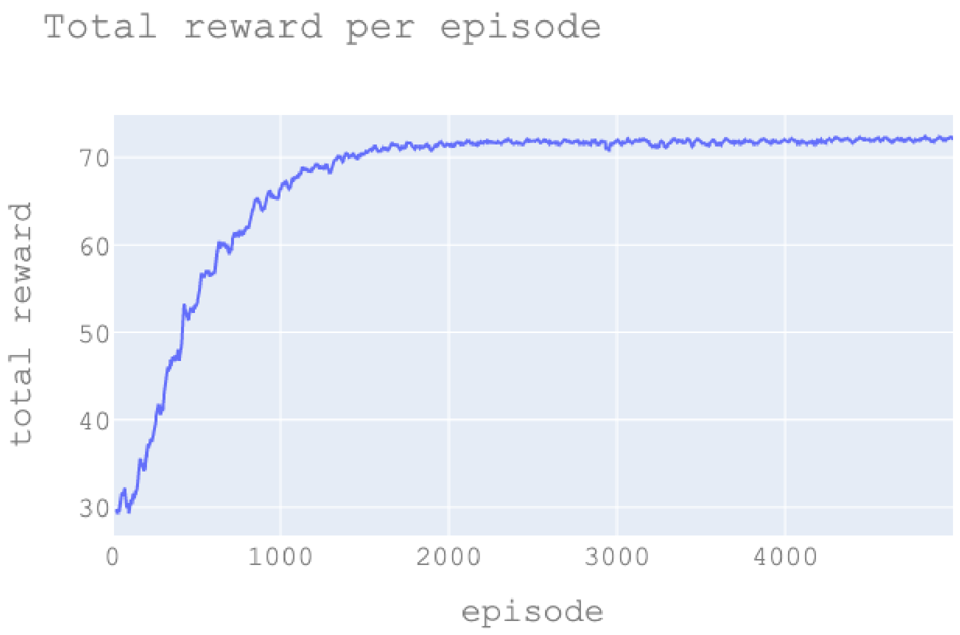


Figure 21: Total reward, N=10

However, the loss function varies from the sparse case. Although the loss reaches to a relatively flat value in the end, the value is relatively higher than the beginning. This might be resulted from the environmental random seed that initialize neural network' parameters in an inappropriate starting value. While at the same time, comparing to the counterpart of sparse environment, the convergent loss value is higher in conflict-prone environment, mainly because more conflicts make agents easier to change the policy. Fig 22

Meanwhile, the individual accessing frequency and step distribution of executing tasks are similar to the sparse case, see Fig 24 and Fig 25, however, the total access frequency is also a little different from the sparse system, as you can see that there is a clear frequency different between No.1 and No.3 access points. See Fig 23

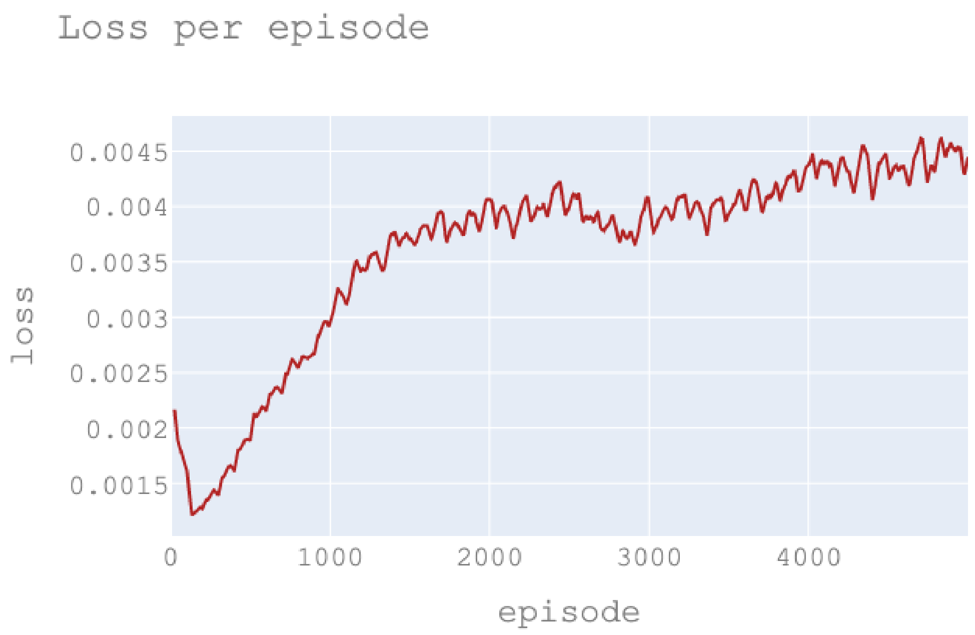


Figure 22: Loss, N=10

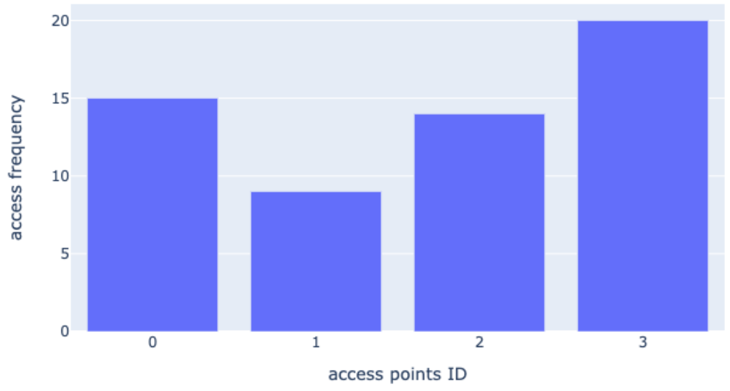


Figure 23: Overall access points preference, N=10

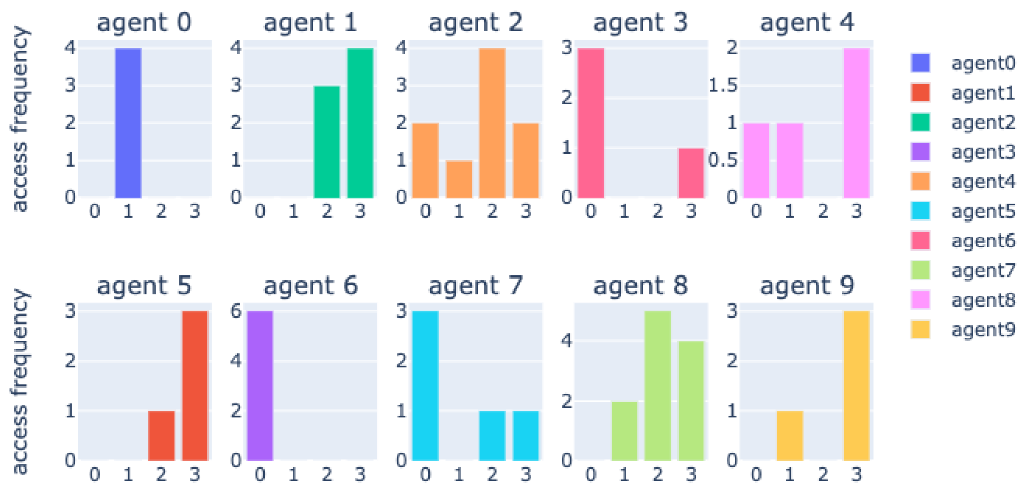


Figure 24: individual access points preference, N=10

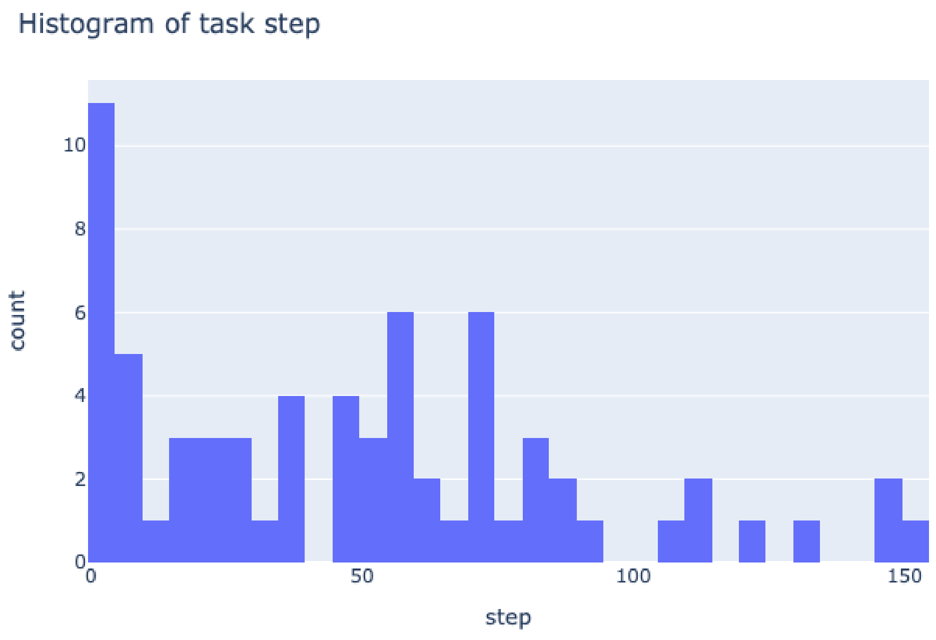


Figure 25: Step distribution of task execution, N=10

8 Conclusion

By conducting two kinds of experiments to compare the performance with different conflict density, we show that agents can execute the task well and even can obtain the highest score in the game, which means they have learned an effective policy and achieve cooperation. Aside from that, some interesting findings are listed below.

- Agents prefer a certain access point when executes its own task, and choose a specific direction to move away;
- when there is conflict happening, agents prefer to leave the access points and move to the cell next to access points rather than simply waiting to execute. Such behavior means that they have learned a policy to avoid conflicts.
- After agents collect a task, they can automatically find the shortest path to the access points.

9 Acknowledgement

First I would like to give my great thanks to Professor Sugawara. He is a very kind professor who is encouraging me all the time and instructed me with great care. With his help, I get to know the wonderful world of machine learning and the state-of-the-art technology. Besides, I also want to give my sincere thanks to my lab friends who gave me a lot of useful advice and instructions as well, especially Dr. Sugiyama, Mr. Miyashita and Mr. Oury who helped me get into the right direction of my research. I'm also grateful to my other friends for their company, and the friendship is motivating me to do my research with passion always.

References

[1] Weiss G. Multiagent systems: a modern approach to distributed artificial intelligence. Cambridge: MIT Press; 1999.

[2] Nikos Vlassis, A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence. *Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool*, pp. 5, 2007.

[3] Jing Xie, Chen-Ching Liu (2017) Multi-agent systems and their applications. *Journal of International Council on Electrical Engineering*,7:1, 188-197, DOI: 10.1080/22348972.2017.1348890.

[4] Zool Hilmi Ismail, Nohaidda Sariff (2018) A Survey and Analysis of Cooperative Multi-Agent Robot Systems: Challenges and Directions. DOI: 10.5772/intechopen.79337

[5] Brunete A, Hernando M, Gambao E, Torres JE. A behavior based control architecture for heterogeneous modular, multi configurable, chained micro robots. Robotics and Autonomous Systems. 2012;60:1607-1624

[6] Simonin O, Grunder O. A cooperative multi robot architecture for moving a paralyzed robot. Mechatronics. 2009;19:463-470

[7] Oleiwi BK, Al-Jarrah R, Roth H, Kazem BI. Integrated motion planing and control for multi objectives optimization and multi robots navigation. In: 2nd IFAC Conference on Embedded Systems, Computer Intelligence and Telematics CESCIT 2015. Vol. 48. 2015. pp. 99-104

[8] Atinc GM, Stipanovic DM, Voulgaris PG. Supervised coverage control of multi agent systems. Automatica. 2014;50:2936-2942

[9] Posadas JL, Poza JL, Simo JE, Benet G, Blanes F. Agent based distributed architecture for mobile robot control. Engineering Applications of Artificial Intelligence. 2008;21:805-823

[10] Sariff N, Buniyamin N. An overview of autonomous robot path planning algorithms. In: 4th Student Conference on Research and Development (SCORED 2006); Shah Alam, Malaysia. June 2006. pp. 184-188

[11] Optimal Multirobot Path Planning on Graphs: Complete Algorithms and Effective Heuristics, *Journal IEEE Transactions on Robotics*, Vol.32, No.5, pp.1163–1177 (2016).

[12] Wolfgang Honig, Dadoun, Scott Kiesel, Andrew Tinka, Joseph W. Durham and Nora Ayanian: Conflict-Based Search with Optimal Task Assignment, *In Proc. of the 17th International Conference on Autonomous Agents and Multiagent Systems, Stockholm, Sweden*, pp.757–765 (2018).

[13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg Demis Hassabis : Human-level control through deep reinforcement learning, *Nature*,Vol.518, pp.529–533 (2015).

[14] Jakob Foerster, Richard Y. Chen,Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel,Igor Mordatch : Learning with Opponent-Learning Awareness,*Proceeding AAMAS '18 Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*,pp.122–130 (2018).

[15] Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip H. S. Torr, Pushmeet Kohli, Shimon Whiteson : Stabilising Experience Replay for Deep Multi-Agent Reinforcement Learning, *arXiv preprint arXiv:1702.08887*, (2017).

[16] Elhadji Amadou Oury Diallo, Toshiharu Sugawara, Learning Coordination in Adversarial Multi-Agent DQN with dec-POMDPs; Proceedings of the 31st IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2019), pp. 198-205, IEEE Xplore, Portland, USA, Nov. 4-6, 2019. (accepted as a full paper) IEEE Xplore, DOI:10.1109/ICTAI.2019.00036

[17] Yuki Miyashita, Toshiharu Sugawara, "Coordination Structures Generated by Deep Reinforcement Learning in Distributed Task Executions," Proceedings of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS2019), pp 2129-2131, Montreal, Canada, May 13-17, 2019.

[18] Michael Wooldridge. An introduction to multiagent systems. John Wiley Sons, 2009.

[19] L. Bus oniu, R. Babusˇka, and B. De Schutter, "Multi-agent reinforcement learning: An overview," Chapter 7 in Innovations in Multi-Agent Systems and Applications – 1 (D. Srinivasan and L.C. Jain, eds.), vol. 310 of Studies in Computational Intelligence, Berlin, Germany: Springer, pp. 183–221, 2010.

[20] Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: A survey. Journal of Artificial Intelligence Research 4, 237–285 (1996)

[21] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg Demis Hassabis : Human-level control through deep reinforcement learning, *Nature*,Vol.518, pp.529–533 (2015).

[22] George E. Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. Audio, Speech, and Language Processing, IEEE Transactions on, 20(1):30 –42, January 2012.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton. Imagenet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems 25, pages 1106–1114, 2012.

[24] Chung-Jae Choe, Seungho Baek, Bongyoung Woon, Seung-Hyun Kong. Deep Q Learning with LSTM for Traffic Light Control. 2018 24th Asia-Pacific Conference on Communications (APCC). IEEE, DOI: 10.1109/APCC.2018.8633520, 2018

[25] Eric Liang, Richard Liaw, Scaling Multi-Agent Reinforcement Learning, 2018, url:https://bair.berkeley.edu/blog/2018/12/12/rllib/

[26] Tom Schaul, John Quan, Ioannis Antonoglou, David Silver, Prioritized Experience Replay, arXiv:1511.05952 [cs.LG]