Karlsruhe Institute of Technology

Fraunhofer
IOSB

Design of artificial intelligence agent for supply chain management using deep reinforcement learning based on NegMAS Library

Masterarbeit

KIT – Karlsruher Institut für Technologie
Fraunhofer IOSB – Fraunhofer-Institut für Optronik, Systemtechnik und Bildauswertung

Ning Yue

10. Mai 2021

Verantwortlicher Betreuer:	Prof. Dr.-Ing. Jürgen Beyerer
	Prof. Dr.-Ing. Thomas Längle
Betreuende Mitarbeiter:	Junior Prof. Dr.-Ing. Yasser Mohammad
	Tim Zander, Ph.D.

Interaktive Echtzeitsysteme IES

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die Arbeit selbständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht habe und die Satzung des Karlsruher Instituts für Technologie zur Sicherung guter wissenschaftlicher Praxis in der gültigen Fassung beachtet habe.

Karlsruhe, den 10. Mai 2021

(Ning Yue)

Thanks

I would like to express my sincere gratitude to the Karlsruhe Institute of Technology(@kit) for letting me fulfill my dream of being a student here. I would also like to thank the **Fraunhofer Institute of Optronics, System Technologies and Image Exploitation (IOSB)**, **Vision and Fusion Laboratory (IES)**, **NEC-AIST AI Cooperative Research Laboratory** for giving me the opportunity to write an interesting thesis.

I would most like to express my sincere and warm gratitude to my supervisor, Yasser Mohammad - for his guidance and support throughout his study and especially for his confidence in me. As well as, to my supervisor, Tim Zander - for the many precious opportunities given to me and support me in realizing my ideas. Also to Tim Baarslag, who offered pointer suggestions for my work, to Xiang Chen - my classmate and friend who gave me the possibility to experiment with his supercomputer, checked the thesis and spent a fulfilling and meaningful master's study career together, to Ganesh Vanave - for sharing the information and giving suggestion for my experiments, to Jinmin Gao, Minhao Xu - paid attention to my experiment, and listened to my thoughts, given suggestions and were patient with me.

Finally, I have to thank my parents - for their endless love, support and encouragement throughout my life and for giving me strength to reach for the stars and chase my dreams. My old brother, sister-in-law, aunt, uncle and cousins deserve my wholehearted thanks as well.

Abstract

Consider an agent that can cooperate with others and autonomously negotiate, to reach an agreement. These agents could achieve great results, such as increasing the profitability of a factory. This type of agent is practical in complex and realistic environments (e.g. supply chain management). In the experiments of this thesis, it is proposed to use some creative methods to implement learnable agents to achieve these goals. When interacting with others continuously, the agent's strategy will be improved. The learned strategy enables autonomous agents to negotiate in real time with multiple different types of unknown opponents on complex and multiple issues. In the last decades, a lot of work tried to develop negotiation agency by simplifying the negotiation environment or with the help of expert knowledges. In recent years, many interesting end-to-end multi-agent deep reinforcement learning methods are proposed and successfully applied in complex environments, such as Dota 2, Starcraft. Hence, some questions will naturally be raised: How to use these new methods and how about the results without simplifying the environments and without the help of extra knowledges? The work of this thesis attempts to establish training environments and implement end-to-end negotiation agents in complex negotiation environments through some deep reinforcement learning methods, such as QMIX and MADDPG.

Kurzfassung

Stellen Sie sich einen Agenten vor, der mit anderen zusammenarbeiten und autonom verhandeln kann, um eine Zustimmung zu erzielen. Diese Agenten könnten großartige Ergebnisse erzielen, beispielsweise die Rentabilität der Fabrik. Diese Art von Agent ist in komplexen und realistischen Umgebungen (z. B. Supply Chain Management) praktisch. In den Experimenten dieser These wird vorgeschlagen, einige kreative Methoden zu verwenden, um lernbare Agenten zu implementieren, um die oben genannten Ziele zu erreichen. Wenn Sie kontinuierlich mit anderen interagieren, wird die Strategie des Agenten verbessert. Die erlernte Strategie ermöglicht es autonomen Agenten, in Echtzeit mit mehreren verschiedenen Arten unbekannter Gegner über komplexe und vielfältige Themen zu verhandeln. In den letzten Jahrzehnten wurde viel Arbeit geleistet, um eine Verhandlungsagentur durch Vereinfachung des Verhandlungsumfelds oder mithilfe von Expertenwissen zu entwickeln. In den letzten Jahren wurden viele interessante End-to-End-Lernmethoden für die Tiefenverstärkung mit mehreren Agenten vorgeschlagen und in komplexen Umgebungen wie Dota 2, Starcraft erfolgreich angewendet. Daher werden natürlich einige Fragen aufgeworfen: Wie werden diese neuen Methoden angewendet und wie steht es mit den Ergebnissen, ohne die Umgebung zu vereinfachen und ohne die Hilfe zusätzlicher Kenntnisse? Die Arbeit dieser These versucht, Trainingsumgebungen einzurichten und End-to-End-Verhandlungsagenten in komplexen Verhandlungsumgebungen durch einige multi-agenten Lernmethoden wie QMIX und MADDPG zu implementieren.

Summary of Notation

Lower case letters are used for the values of random variables and for scalar functions. Capital ltters are used for random variables and major algorithms variables.

General identifier

s	state or step
a	action (RL) or agent (autonomous negotiation)
t	discrete time step
o	observation
T	final time step of an episode
π	policy, decision-making rule
μ	policy, determinstic policy
γ	discount-rate parameter or multiplication factor in utility function of an agent
α	learning rate or multiplication factor in utility function of an agent
ε	probability of random action in ε-greedy policy
θ	parameters of policy (i.g., network)
ω	parameters of network
φ	preprocessing function for state
Q	Q value in reinforcement learning
V	V value in reinforcement learning
J	objective function
μ	utility of contract (offer), utility of agent, expected profit of a factory
p	price
q	quantity
m	cost of production
P	total producible quantity
d	step (day) of simulation

Special identifier

S_t	state at t
A_t	action at t
R_t	reward at t
G_t	return (cumulative discounted reward) following t
θ_k	parameters of old policy, in page 21
$\hat{A}$	advantage function, used in ppo, 21
$\pi(s)$	action taken in state s under `deterministic` policy π
π_{θ_k}	old policy in ppo, in page 21
$\pi(a\|s)$	probability of taking action a in state s under stochastic policy π
$p(s',r\|s,a)$	probability of transitioning to state s', with reward r, from s, a
$v_\pi(s)$	value of state s under policy π (except return)
$v_*(s)$	value of state s under the optimal policy
$q_\pi(s,a)$	value of taking action a in the state s under policy π
$q_*(s,a)$	value of taking action a in the state s under optimal policy
$V_t(s)$	estimate (a random variable) of $v_\pi(s)$ or $v_*(s)$
$Q_t(s,a)$	estimate (a random variable) of $q_\pi(s,a)$ or $q_*(s,a)$
Q_{tot}	centralised action-value function
$\mathbf{w}, \mathbf{w}_t$	vector of (possibly learned) weights

General quantities

$\mathcal{S}$	set of nonterminal states
$\mathcal{A}(s)$	set of actions possible in state s
$\mathcal{R}$	set of possible rewards
$\mathcal{C}$	set of contracts (i.e., agreements signed by agent)

Contents

1. Introduction

Computer software and hardware development leads to the appearance of non-human software agents. An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors[BM07].

In economics, autonomous agent can be considered as a specific type of agent, with a focus on generating economic value. This technology will be at the forefront of the next industrial revolution, affecting numerous billion dollar industries such as transportation and mobility, finance, supply chain, energy trading, social networks and marketplaces and e-commerce. The detail about application field of autonomous agent will be listed in chapter Background 2. All the work in this article is centered on the application of automatic agent in autonomous negotiation and in the supply chain.

A supply chain is a network of suppliers, factories, warehouses, distribution centers and retailers, through which raw materials are acquired, transformed, produced and delivered to the Customer. In the network we could find many entities whose can be considered as agents mentioned in the Multi-agent systems (MAS). MAS is suitable the domains that involve interactions between different people or organizations with different (possibly conflicting) goals and proprietary information. Supply chain network is a typical MAS. There are many approaches are proposed in order to solve the problem in this system, such as negotiation-based Multi-agent System[Che+99; BBB12]. Supply chain management (SCM) involves the following stages: planning, procurement, production, delivery and return. At each stage it has its own processes, problems and solutions[PB11].

SCM world designed in simulator called supply chain management leagure (SCML) based on opensource package NegMAS by `Yasser Mohammad` simulates a supply chain consisting of multiple factories that buy and sell products from one another. The factories are represented by autonomous agents that act as factory managers. Agents are given some target quantity to either buy or sell and they negotiate with other agents to secure the needed supplies or sales.

Their goal is to turn a profit, and the agent with the highest profit (averaged over multiple simulations) wins. SCML is characterized by profit-maximizing agents that inhabit a complex, dynamic, negotiation environment[Moh+19]. All definitions related to SCML and NegMAS will be provided in Background 2.6.3 and 2.6.2. Because all subsequent work in this thesis will be carried out around these platforms. Before discussing the details of the work of this thesis, we need to give specific motivations for the work.

1.0.1. Motivation

Negotiation is a complex problem, in which the variety of settings and opponents that may be encountered prohibits the use of a single predefined negotiation strategy. Hence, the agent should be able to learn such a strategy autonomously[Bak+19]. By autonomy it means "independent or self-governing". In the context of an agent, this means it can act without constant interference from its owner[Fet19]. Autonomous negotiation agent is meaningful in many realistic environment, such as SCM. The current increased hardware resources make it possible to use computer systems to model real-world environments to evaluate the problems. According to the modeled environment, it will be easier to find more possible solutions with the help of machine learning technology.

In the work of this thesis, some modeled negotiation environments have been developed, such as single-agent environment (bilateral negotiation), to analyze whether deep reinforcement learning can be used to allow agents to automatically learn some good strategies. Compared with the single agent environment, in a supply chain environment, there are many agents with the same goals. After analyzing the simple environment, the situation is needed to be explored whether multi-agent deep reinforcement learning methods can be used to obtain better results in multi-agent environment (concurrent bilateral negotiations).

What is the significance of applying Deep Reinforcement Learning in supply chain management?

Due to the great success of Alphago Zero[Sil+17] and OpenAI Five[Ber+19], reinforcement learning has entered a new historical stage. As a general machine learning method, whether it can be used to improve the management ability of factories in the supply chain world is a very natural idea. However, deep reinforcement learning has many problems. Whether it is effective or not in supply chain needs to be tested through experiments. The supply chain world is a very typical multi-agent system, and the goal of maximizing profits is also a typical multi-step decision-making problem. Hence, many multi-agent reinforcement learning methods have great practical significance and research value here.

How good strategy can be learned by Deep Reinforcement Learning (DRL) in single agent environment (bilateral negotiation)?

Before testing deep reinforcement learning in a multi-agent environment, single-agent bilateral negotiation is a better useful test environment. The result will help to analyze the role of algorithm in autonomous negotiation game. It has a certain practical significance. In recent years, many single agent independent DRL algorithms has been proposed, such as DQN, PG, A2C etc. The application of these algorithms in automatic negotiation will be a very meaningful study. The performance of these algorithms will provide inspiration for the analysis of more complex negotiation environment.

How good strategy can be learned by multi-agent deep reinforcement learning in multi-agent environment (concurrent negotiation)?

This question is the key question of this thesis. Due to the successful application of DRL in complex game environments, exploring its significance in complex negotiation environments will accelerate the research progress in the field of automatic negotiation. Beside the value of research, definitively, it also has the economic value. By comparing with other benchmark negotiators or agents, the importance of DRL can be evaluated.

What is the difference between deep reinforcement learningDRL strategies and other heuristic strategies?

Researchers has proposed many heuristic strategies, such as time-based and behavior-based negotiation strategies, which will be introduced in chapter related work 3.1. Comparing these strategies can help achieve better strategies, and agents based on these strategies can be used as opponents of RL agents. The results of the evaluation will help to understand the reasons for using deep reinforcement learning.

In order to obtain and analyze the results of the above four questions, it is necessary to understand the simulation logic of the simulator NegMAS and SCML as a prerequisite for the experiment.

1.0.2. Outline of this Work

In the following, the other chapters of the work of thesis are listed and their content briefly presented.

Chapter 2: Background: This chapter contains basic knowledge and concepts that are necessary for understanding the work of this thesis. First, some concepts from game theory are introduced. These mentioned concepts are often discussed and used in autonomous negotiation. Second, utility function and several important negotiation mechanisms are described in the section autonomous negotiation. In addition, the basics and the historical development of artificial intelligence are presented in two individual sections. The focus of these sections are basic knowledge of reinforcement learning. Then, NegMAS and SCML are introduced relatively clearly, which are two important packages (simulator) that simulate autonomous negotiation and supply chain world, respectively. Finally, this chapter contains the brief introduction of some tools (OpenAI Gym) and training tools (Ray) used to create a training environment.

Chapter 3: Related Works In this chapter, some published matter which technically relates to the proposed work in this thesis will be discussed. These publication will be divided as three categories: Heuristic Negotiation Strategies for Autonomous Negotiation, Reinforcement Learning used in Autonomous Negotiations and Challenges in Deep Reinforcement Learning. Heuristic negotiation strategies, such as time-based and behavior-based, will be discussed first. In the second part, some deep reinforcement learning frameworks for autonomous negotiation will be introduced. Finally, it is about typical challenges (e.g., design of reward function) in the field of reinforcement learning. Some solutions will be given in this section.

Chapter 4: Analysis The task of the thesis is studied in detail in the chapter "Analysis". The main content of this chapter is to explain how to design two custom training environments: single-agent (bilateral negotiation) and multi-agent (concurrent negotiation) environment. In addition to the model of the training environment, this chapter will also introduce the design of the observation space, action space and reward function in the algorithm. This means that the characteristics of the algorithm need to be analyzed in advance.

Chapter 5: Methods and Experiments The detailed configuration of the experimental environment is explained in this chapter. The specific hyperparameters and training process of the algorithm will also be explained. Finally, the experimental results are presented and evaluated, and compared with other algorithms.

Chapter 6: Conclusions and Future Work In the last chapter, the work of this thesis will be summarized and the areas for improvement will be pointed out. Finally, that chapter will provide some directions for future work.

2. Background

2.1. Game Theory

Game theory is the study of the ways in which interacting choices of economic agents produce outcomes with respect to the preferences (or utility) of those agents, where the outcomes in question might have been intended by none of the agents[Ros19]. The key pioneers of game theory were mathematician John von Neumann and economist Oskar Morgenstern in the 1940s [NMR44]. A few terms are commonly used in the study of game theory: **Game**, **Payoff**, **Players**, **Strategy**, **Equilibrium**.

2.1.1. Nash Equilibrium

The concept of a Nash equilibrium plays a central role in game theory. The definition in a simple setting of finite number of players is described as follow. Let as $k = i, \ldots, K$ to be agent index. S_k denotes the set of strategies, and s_k indicates a the member of the set. A strategy profile, named $s = (s_1, ..., s_K)$, is a vector of strategies for the individual players. Hence, all strategy profiles can be written as S for $\Pi_{k=1}^{K} S_k$. We write $s \mid s'_k$ for the strategy $\left(s_1, \ldots, s_{k-1}, s'_{k+1}, \ldots, s_K\right)$ means a strategy of agent k changed from s_k to s'_k, in which a strategy profile s is $s = (s_1, ..., s_K)$ and a strategy of agent k is $s'_k \in S_k$. The excepted utility or payoff of each agent k is defined as $u_k(s)$, when agents select strategy profile s[Kre89].

Proposition 2.1 (Nash Equilibrium) *For a strategy profile s^*, Nash equilibrium can be described by a set of mathematical inequalities: For each agent k and $s'_k \in S_k$, $u_k(s^*) \geqslant u_k\left(s^* \mid s'_k\right)$.*

In plain english, in a Nash equilibrium if other agents do not change their strategy, no single agent can obtain higher utility by changing its own.

2.1.2. Pareto Efficient

A Pareto Efficient state is also called as **Pareto Optimal** state and is defined as a state at which resources in a system are optimized in a way that one dimension cannot improve without a second worsening. We can consider an economic scenario, there are N agents and K goods. For an allocation state, formed as $x = \{x_1, \ldots, x_n\}$, where $x_i \in \mathbb{R}^K$. x_i represents the resource set allocated to each agent i. The utility of each agent i is formed as $u_i(x_i)$. Therefore, the Pareto optimal allocation is defined as follows.

Proposition 2.2 (Pareto Efficient) *There is no other feasible allocation $\{x'_1,...,x'_n\}$ where, for utility function u_i for each agent i, $u_i(x'_i) \geqslant u_i(x_i)$ for all $i \in \{1,...,n\}$ with $u_i(x'_i) > u_i(x_i)$ for some i [Whi95].*

The relationship between Pareto Efficiency and Nash Equilibrium has following two points:

- A certain Nash Equilibrium will implement a resource allocation, which may or may not be Pareto optimal.
- A certain Pareto Optimal resource allocation may or may not be obtained by the execution of the Nash Equilibrium of a complete information static game.

2.1.3. Markov Games

Games based on the **Markov Decision Process** (MDP) are called Markov Games. The term MDP has been proposed by Bellman in 1954[Bel54]. It models decision making in discrete, stochastic, sequential environments. The essence of the model is that a decision maker, or agent, inhabits an environment, which changes state randomly in response to action choices made by the decision makers.

Finite MDP model means MDP with finite time horizon. An MDP is fully characterised by its state space, action space, random transition law and reward function of the system. In the context of sequential decision-making under uncertainty, a stationary environment is an environment whose components do not evolve over time. Unfortunately, the stationarity condition does not hold in problems like stock market forecasting, multi-agent problems where agents learn simultaneously[Had15]. In non-stationary, some components, such as the transition and reward functions, will be evolve over time. Hence, a non-stationary MDP with horizon $N \in \mathbb{N}$ consists of a set of data $(S, A, D_n, Q_n, r_n, g_N)$ with the follwing meaning [BR10]:

- S is the state space, the elements (states) are denoted by $s \in S$
- A is the action space, the elements (actions) are denoted by $a \in A$.
- $D_n \subset S \times A$ denotes the set of admissible state-action pairs at time n.
- Q_n is a stochastic transition kernel from D_n to S. Q_n describes the transition law. The quantity $Q_n(s' \mid s, a)$ gives the probability that next state at time $n+1$ is s' if the current state is s and action is a.
- $r_n : D \longrightarrow \mathbb{R}$ is a mapping function. Hence, at the time n, the reward of the system can be denoted by $r_n(s, a)$, where the state is s and action is executed as a.
- $g_N : E \longrightarrow \mathbb{R}$ is a measurable mapping. $g_N(s)$ gives the discounted terminal reward of the system at the time N if the state is s.

Next, the definition of strategy (policy) will be introduced. A strategy is a mapping $\pi : S \longrightarrow A$, where $\pi(s)$ is the action an agent will perform in state s. In the case of a given MDP, the agent should adopt the best strategy, which should maximize the accumulated expected reward when performing the specified action. Since RL is based on the MDP process, how to find the optimal strategy and calculate the accumulated expected reward are important questions in the research field of RL. This will be discussed further introduced in section value functions 2.5.2.

Multi-Agent Markov Decision Processes(MDPs) Based on communication ability of agents, multi-agent extension of MDPs can be called partially or completely observable markov games. There are N players indexed by $n = 1,2,\ldots,N$. A markov game for N agents is defined by a set of states S describing the possible configuration of all agents. $A_1, \ldots, A_N$ and $O_1, \ldots, O_N$ are the set of actions and observations of individual agents, respectively. For each agent, a stochastic strategy $\boldsymbol{\pi}_{\theta_i}$ will be used to choose action, the mapping is $\boldsymbol{\pi}_{\theta_i} : O_i \times \mathcal{A}_i \mapsto [0,1]$. For the multi-agent MDPs, the transition function is defined as following mapping functions $\mathcal{T} : S \times A_1 \times \ldots \times A_N \mapsto \mathcal{S}$, which produces the next state. The term $r_i(s, a)$ means the reward of agent i if the state is s and action a is taken. The mapping of reward function is described as $r_i : S \times A_i \mapsto \mathbb{R}$. Each agent i receives the private observation correlated with the state $o_i : S \mapsto O_i$[Low+17]. There are many cases for multi-agent MDPs. The following three cases are the typical cases that are introduced by `Boutilier` in [Bou96].

- Complete communication
- Communication of actions, but not states
- No communication of actions or states

2.2. Autonomous Negotiaion

Negotiation is an important process in coordinating behavior and represents a principal topic in the field of multi-agent system research. Extensive research has been conducted in the field of autonomous negotiation[FSJ98; PB11; BHJ13; Zho+17; Ayd+17; LKK04; Lin+14].

Automated agents can be used side-by-side with a human negotiator embarking on an important negotiation task. They can alleviate some of the effort required of people during negotiations and also assist people who are less qualified in the negotiation process. There may even be situations in which automated negotiators can replace the human negotiators. Thus, success in developing an automated agent with negotiation capabilities has great advantages and implications[Baa+12].

The perfection of information is an important notion in game theory when considering sequential and simultaneous games. It refers to the fact that each player has the same information that would be available at the end of the game. This means, each palyer knows other player's information. Imperfect information appears when decisions have to be made simultaneously, and players need to balance all possible outcomes when making a decision. A good example of imperfect information games is autonomous negotiation game where each player (negotiator)'s utility function are hidden from the rest of the palyers (negotiators).

2.2.1. Utility Function (ufun)

Utility Function is an important concept in economics. It measures preferences for a set of goods and services. Utility Function can measure either single offer or set of offers.

Utility represents the satisfaction that consumers obtain when choosing and consuming products or services and is measured in units called utils, but calculating the benefit or satisfaction that consumers receive from is abstract and difficult to pinpoint[BLO19]. One typical utility function is the linear ufun:

- **linear utility function:** $u(x_1, x_2, \ldots, x_m) = w_1x_1 + w_2x_2 + \ldots w_mx_m$ or described as a vector $u(\vec{x}) = \vec{w} \cdot \vec{x}$, where m is the number of differen goods in the economy. The element x represents the amount of good i. The element w_i represents the relative value that the consumer assigns to good i.

For heuristic agents, the utility function is a keypoint to measure preferences. For reinforcement learning agents utility function conducts the behavior of learnable agents, and it represents the reward function, which significantly affects the design and evaluation of RL-agent.

2.2.2. Basic Notation in Negotiation Mechanism

Before introducing a specific negotiation mechanism, it is necessary to understand some basic symbols defined in the paper [Ayd+17].

Definition 2.3 (Round and Phase) *Round and Phase within rounds are used to structure the negotiation process.*

Definition 2.4 (Turn taking) *Protocols assign turns to the negotiating agents. Turns are taken according to a turn-taking sequence.*

2.2.3. Rubinstein Bargaining Mechanism

Rubinstein bargaining mechanism is widely cited for multi-round bilateral negotiation[Baa+14; Y+21; FWJ02]. Two agents join the mechanism which has an infinite time horizon and have to reach an agreement. In a turn, one agent proposes an offer, the other needs to decide either to accept it, or to reject it and continue the bargaining[Rub82]. The offer is about how to divide a pie of size 1. After the two agents have played indefinitely, they may get the corresponding nash equilibrium solution. For example, when the both utility functions are common knowledge, the Nash equilibrium is achievable in a single round.

2.2.4. Stacked Alternating Offers Mechanism (SAOM)

In the alternating offers protocol also called AOM, one of the agents start to propose an offer. The other can either accept or reject the given offer. If an agreement is reached, the negotiation is successful and ended. When rejecting the offers the other agent can either end the negotiation or give a counter offer.

SAOM is also named the stacked alternating offers protocol (SAOP)[Ayd+17]. Agents can only take their action when it is their turn. SAOM allows negotiating agents to evaluate only the most recent offer in their turn and accordingly they can take the following actions:

- Make a count offer (thus rerejecting and overriding the previous offer)
- Accept the offer
- Walk away (e.g. ending the negotiation without any agreement)

This negotiation process is repeated until a termination condition is met. The termination condition is met, if an agreement is reached, the deadline is reached or one agent walks away. When an agreement is reached, all agents need to accept the offer. If the deadline is reached with no agreement or any agent walks away, this negotiation is failed.

2.3. Artificial Intelligence

Artificial intelligence is a broad branch of computer science that is focused on a machine's capability to produce rational behavior from external inputs. The goal of AI is to create systems that can perform tasks that would otherwise require human intelligence. It is generally believed that the field of artificial intelligence began at a conference held at Dartmouth College in July 1956, when the term "artificial intelligence" was first used[BFF09].

There is a set of three related terms that sometimes are erroneously used interchangeably, namely artificial intelligence, machine learning, and deep learning. According to Encyclopaedia Britannica, AI defines the ability of a digital computer or computer-controlled robot to perform tasks commonly associated with intelligent beings. On the other hand, according to Arthur Samuel, one of the pioneers of the field, machine learning is a “field of study that gives computers the ability to learn without being explicitly programmed”[Sam59; Bha+17].

2.3.1. Sub-areas

Fig. 2.1 shows the relationship of artificial intelligence, machine learning and deep learning. AI is a program that can sense, reason, act and adapt. Machine learning is a set of methods, whose performance improve as they are exposed to more data over time. Deep learning is a subset of machine learning, in which multi-layered neural networks learn from vast amount of data.

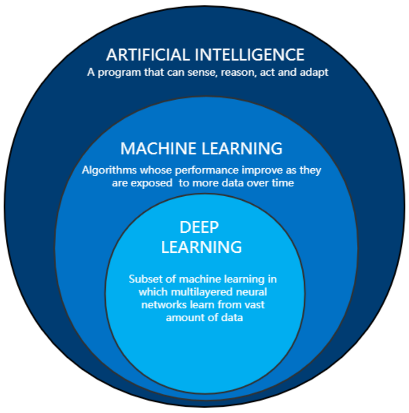
ARTIFICIAL INTELLIGENCE
A program that can sense, reason, act and adapt
MACHINE LEARNING
Algorithms whose performance improve as they are exposed to more data over time
DEEP LEARNING
Subset of machine learning in which multilayered neural networks learn from vast amount of data

Figure 2.1.: Sub-areas of artificial intelligence Source: Own illustration based on[SUM20]

2.3.1.1. Artificial Intelligence

Artificial intelligence (also called machine intelligence) can be understood as a type of intelligence, which is different from the natural intelligence displayed by humans and animals, which can be demonstrated by machines. It looks at methods for designing smart devices and systems that can creatively solve problems that are usually regarded as human privileges. Hence, AI means that the machine imitates human behavior in some way.

2.3.1.2. Machine Learning (ML)

Machine learning is an AI subset and consists of techniques that enable computers to improve their performance using data and supply AI applications. Different algorithms (e.g., neural networks) contribute to problem resolution in ML.

2.3.1.3. Deep Learning (DL)

Deep learning, often called deep neural learning or deep neural network, is a subset of machine learning that uses neural networks to evaluate various factors with a similar framework to a human neural system. The defining characteristic of the DL system is the use of a "deep" architecture with many sequential structural layers or equivalent structures. It has networks that can learn from unstructured or unlabeled data without supervision and has a powerful ability to extract complex features. DL has been successfully applied in many field: self-driving-cars, natrual language processing, visual recognition, automatic game playing, etc.

2.3.2. Types of ML problems

Supervised Learning Training data contains optimal outcomes (also known as inductive learning). Learning is tracked in this method. Some famous examples of supervised machine learning algorithms are Linear regression for regression problems.

Unsupervised Learning Training data contains no labels. Clustering is an example. It is impossible to know what is and what is not good learning. Recently, self-supervised DL was introduced as a way to extend existing SL methods[Car+21].

Semi-supervised Learning A few labels are included in the training data.

Reinforcement Learning Rewards are given after a sequence of actions. In a given case, it is a matter of taking appropriate steps to maximize compensation.

2.3.3. Application Field

AI researchers have paid a great deal of attention to automated negotiation over the past decade and a number of prominent models have been proposed in the literature. AI was applied in many scenarios, such as eCommerce, Logistics and Supply Chain and as research tools for computer science. In this section, some applications of autonomous negotiation based on AI will be introduced.

2.3.3.1. eCommerce

Rank in E-Commerce Search Engine In E-commerce platforms such as Amazon and TaoBao, ranking items in a search session is a typical multi-step decision-making problem. AI can learn the relation between different ranking steps, in the paper [Hu+18] authors use reinforcement learning (RL) to learn an optimal ranking policy, which maximizes the expected accumulative rewards in a search session. The more reasonable the ranking of commodities, the more frequent commodity transactions, and the greater the corresponding income.

Business-to-Business Negotiation Negotiation is an important challenge for B2B eCommerce. For B2B e-commerce, AI is making great strides and is being used in a variety of ways to improve and enhance business[Wei01]. AI-based algorithms and tools can help companies in various ways, from personalizing the shopping experience to improving supply chain management.

2.3.3.2. Logistics and Supply Chain

Contextual Intelligence has the ability to understand the limits of our knowledge and to adapt that knowledge to an environment different from the one in which it was developed. AI provides contextual intelligence for the supply chain, and they can use contextual intelligence to reduce operating costs and manage inventory. Contextual information can help them return to customers quickly.

Enhancing Productivity and Profits AI can analyze the performance of the supply chain and propose new factors that affect the same field. It can combine the capabilities of different technologies such as reinforcement learning, unsupervised learning and supervised learning to discover factors and problems that affect the performance of the supply chain and can make better contracts between different suppliers and consumers[Pnd19]. It can analyze the data related to the supplier like audits, in-full delivery performance, credit scoring, evaluations and based on that deliver information which can be used to make future decisions. This kind of step helps the company make better decisions as a supplier and work towards improving customer service. Autonomous negotiation by autonomous agent is an important technology that can be used in this field. Pactum, a successful platform that leverages AI to automatically negotiate supplier contracts for enterprises. Pocket negotiator is a negotiation support system, which contains 4 sub-projects: qualitative negotiation models, emotion and interaction in negotiation, man machine interaction and negotiation and negotiation strategies.

2.4. Artificial Neural Network (ANN)

Artificial neural network is a technology based on the study of the brain and nervous system[WC03]. ANNs are efficient data-driven modelling tools widely used for nonlinear systems dynamic modelling and identification, due to their universal approximation capabilities and flexible structure that allow to capture complex nonlinear behaviors [SE18].

2.4.1. Artificial Neuron

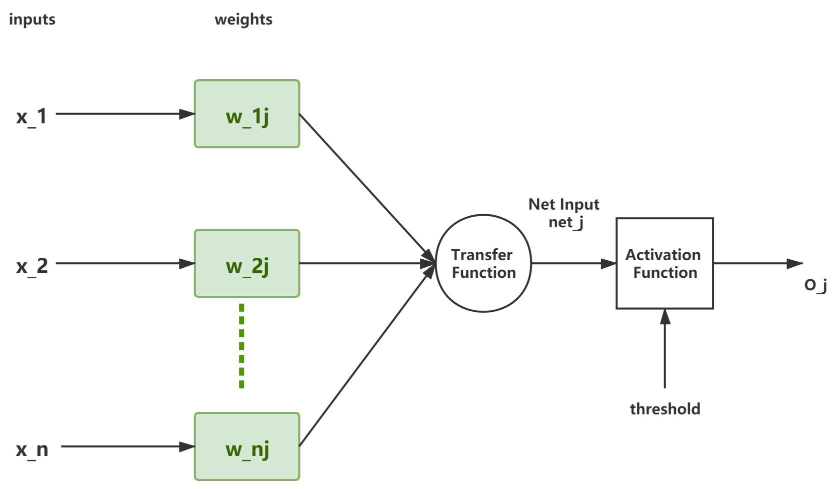

Figure 2.2.: Aritifical Neuron

Artificial neuron defines the core module of neural network, in addition to weighted input, it also contains transfer and activation functions. Figure 2.2 diagrams an aritificial neuron. The input and output of neuron are $x_1, x_2, \ldots, x_N$ and o_J, respectively. The output is obtained by calculation and processing of an activation function and a transfer function. The transfer function usually uses the weighted sum function defined below:

$$\text{net}_j = \sum_{i=1}^{n} w_{ij} x_i \tag{2.1}$$

Then the result of transfer function inputs to activation function. Based on different goal of application, related activation function will be used to calculate the output o_j, the general formula is as follows:

$$o_j = f(net_j) \tag{2.2}$$

There are many different activation functions such as sigmoid, softmax, relu and tanh. The corresponding activation function is used in specific application scenarios (e.g. classification or regression).

2.4.2. Multi-Layers Neural Network

In addition to the input and output layers, there are intermediate layers that do not interact with the external environment. Therefore, these intermediate layers are called hidden layers, and their nodes are called hidden nodes. The addition of hidden layers expands the ability of neural networks to solve nonlinear classification problems[BH00]. Figure 2.3 diagrams a multi-layers neural network.

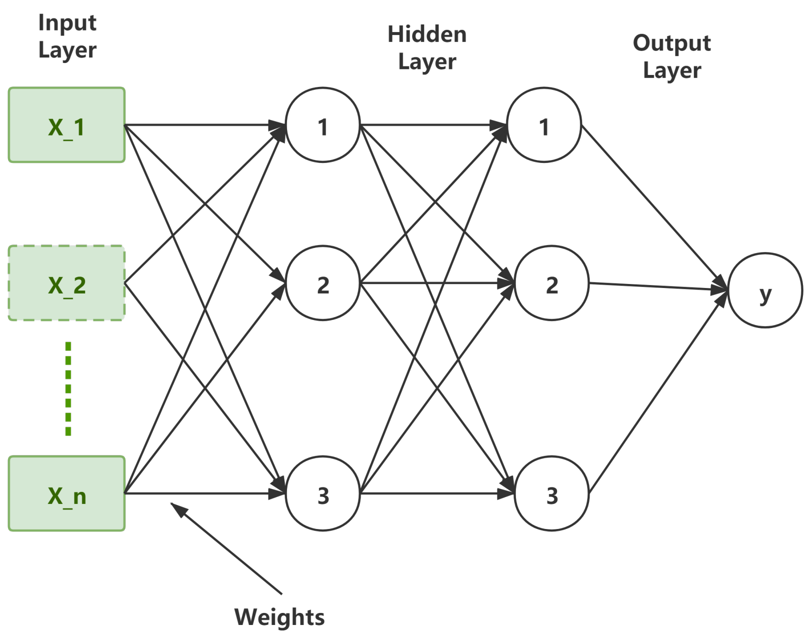

Figure 2.3.: Aritifical Multi-Layers Neural Network, x* are inputs. Source: Own illustration based on[Sai+19].

2.4.3. Recurrent Neural Networks (RNNS)

In a recurrent neural network, the output of some neurons are fed back to the same neurons or to neurons in the previous layers[BH00]. Information flows both forward and backward directions. These networks have an important ability to store information. The special algorithms for training recurrent networks were introduced in the book [Has95]. Figure 2.4 depicts a recurrent neural networks.

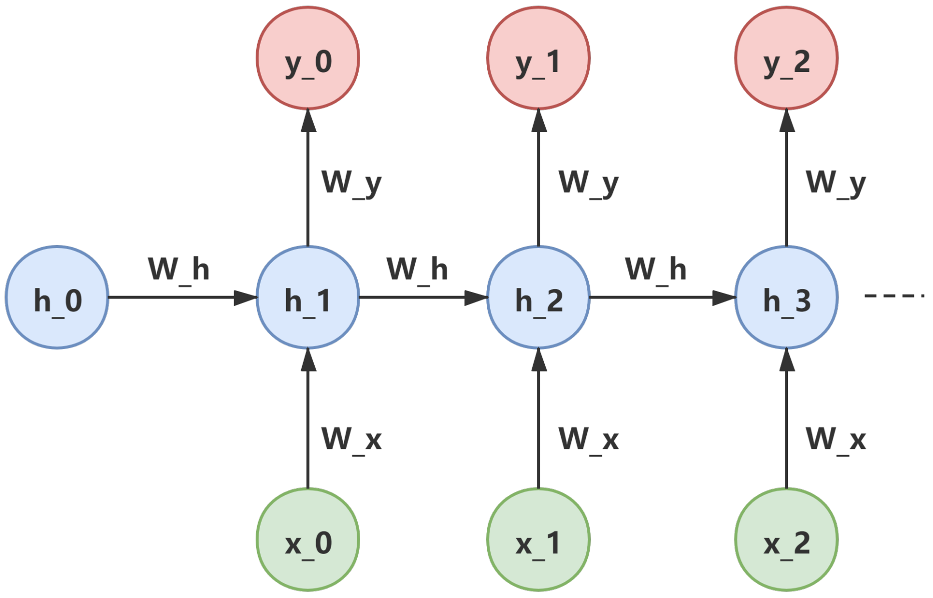

Figure 2.4.: Recurrent Neural Network where h* are outputs of hidden layers, x* are inputs, y* are outputs of network, w* are the weights corresponding to the layers.

2.5. Reinforcement Learning

2.5.1. The Agent–Environment Interface

The reinforcement learning problem is meant to be a straightforward framing of the problem of learning from interaction to achieve a goal. A learner and decision-maker is called an agent. The thing it interacts with, comprising everything outside the agent, is called the environment. These interact continually, the agent selecting actions and the environment responding to those actions and presenting new situations to the agent [SB18]. Figure 2.5 diagrams the agent–environment interaction.

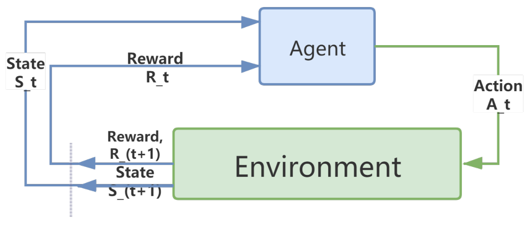

Figure 2.5.: The agent–environment interaction in reinforcement learning, Source: Own illustration based on[SB18].

The process is a typical single-agent interaction process. Single-agent reinforcement learning algorithms are based on this process. By extending this interactive process, multi-agent interactive process can be intuitively diagrammed In 2.6. The case based on multi-agent interaction process is called multi-agent reinforcement learning MARL. In the figure 2.6, an agent is splited as two parts: **perceptor** and **learner**. **Perceptor** observes the environment and sends the state to learner. **Learner** learns strategies based on the states, rewards and actions. There are many methods proposed over the past decade. Some MADRL methods (e.g., MADDPG) will be discussed in the following sections. According to the design requirements of the training environment, the structure of the interaction process is flexible.

2.5.2. Value Function

Just like the description of Markov games in the chapter background 2.1.3, the agent attempts to find a strategy that maximizes the expected cumulative future discounted reward. The definition of value function stems from this goal of the agent. Value Function can be referred as the state value function and state-action value function which are composed of fixed state or both state and action (state-action pair), respectively.

State Value Function (**V**) measures the goodness of each state. It is based on the return reward G following a policy π. In a formal way, the value of $V_\pi(s)$ is:

$$V_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s\right] = \mathbb{E}_\pi \left[\sum_{j=0}^{T-t-1} \gamma^j r_{t+j+1} \mid S_t = s\right] \tag{2.3}$$

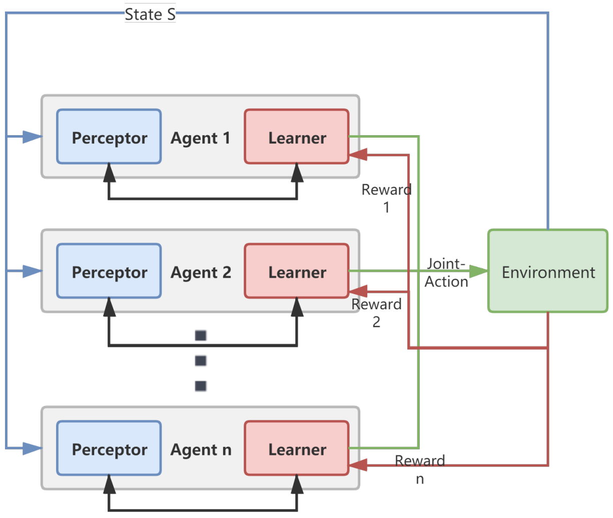

Figure 2.6.: The multi-agent–environment interaction in reinforcement learning, Source: Own illus- tration based on [Fan+20].

Compared with an infinite MDP the time horizon here is set as T. The discount is γ. At the time t, S_t denotes the state of agent.

State-Action Value Function (Q) which measures the goodness of each pair of state and action. Compared with state value function, an action is also determined. The meaning of G, S_t and γ are same as in the state value function. A_t indicates the action of agent at the time t.

$$Q_\pi(s,a) = \mathbb{E}_\pi\left[G_t \mid S_t = s, A_t = a\right] = \mathbb{E}_\pi\left[\sum_{j=0}^{T-t-1} \gamma^j r_{t+j+1} \mid S_t = s, A_t = a\right] \tag{2.4}$$

2.5.3. Bellman Function

In summary, bellman function decomposes the value function into two parts: **immediate reward** ($r(s,a)$) and **discounted future values** ($\gamma V_\pi(s')$). Equation 2.5 shows how to recursively define the Bellman equation for the state-value function:

$$V_\pi(s) = \sum_a \pi(a \mid s) \cdot \sum_{s'} P_{ss'}^a \left(r(s,a) + \gamma V_\pi(s')\right) \tag{2.5}$$

$\pi(a \mid s)$ is the probability that an agent chooses the action a in the state s. $P^{a}_{ss'}$ is the probability that state of an agent changes from s to s' after the agent chooses action a.

As same as bellman equation for the state value function, equation 2.6 indicates how to recursively find the q value of a state-action pair following a policy π.

$$Q_{\pi}(s,a) = \sum_{s'} P^{a}_{ss'} \left(r(s,a) + \gamma V_{\pi}(s')\right) \tag{2.6}$$

2.5.4. Q-Learning

To maximize the total cumulative reward in the long sequence is the goal of RL-agent. The policy, which maximizes the total cumulative reward is called optimal policy formed as π^*. Optimal **State-Action-Value-Function** and optimal **State-Value-Function** are formed as $Q^*_{\pi}(s,a)$ and $V^*_{\pi}(s)$, respectively. The update rule of **State-Action-Value-Function** is shown below:

$$\begin{aligned} Q_t(s,a) &= Q_{t-1}(s,a) + \alpha TD_t(s,a) \\ &= Q_{t-1}(s,a) + \alpha \left(R(s,a) + \gamma \max_{a'} Q(s',a') - Q_{t-1}(s,a) \right) \end{aligned} \tag{2.7}$$

TD is the abbreviation of **Temporal Error**. $R(s,a) + \gamma \max_{a'} Q(s',a')$ denotes the TD target(target q). Current state-action value is formed as $Q_{t-1}(s,a)$. $Q_t(s,a)$ is updated with the learning rate α.

2.5.5. Policy Gradient PG

Different from Q-Learning, policy gradient learns the strategy directly based on the gradient of reward function. The policy is usually modeled with a parameterized function respect to θ, $\pi_{\theta}(a \mid s)$. The gradient of reward function is shown below:

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{s \sim d^{\pi}, a \sim \pi_{\theta}} \left[\nabla_{\theta} \log \boldsymbol{\pi}_{\theta}(a \mid s) Q^{\boldsymbol{\pi}}(s,a) \right] \tag{2.8}$$

The derivation process is written in the appendix A.1.

2.5.6. Deep Reinforcement Learning (DRL)

2.5.6.1. Deep Q-Networks

DQN uses a Q-Netowrk to replace the Q-Table in normal Q-Learning. Additionally, the training process is supervised by the target network, and the target network is updated by duplicating the parameters of the current network after the fixed frequency training step. The target q is formed as $y_i = \begin{cases} r_i & \text{if episode terminates at step i + 1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(s_{i+1}, a'; \theta^-\right) & \text{otherwise} \end{cases}$.

The loss function of DQN is shown in the following equation:

$$\mathcal{L}(\theta) = \sum_{i=1}^{T} \left[(y_i - Q(s, a \mid \theta))^2 \right] \tag{2.9}$$

θ^- and θ are the parameters of target network and current network, respectively. The detailed information of the algorithm is described in appendix B.1.1.

2.5.6.2. Proximal Policy Optimization (PPO)

During the training process, PG will continuously update the probability distribution of actions. However, there is a problem: if the reward is always positive or negative, some possible actions will disappear and it is difficult to sample these actions. PPO uses some constraint tricks, such as clip of policy, to avoid this problem. It makes the probability distribution of actions more reasonable. The loss function based on PPO-clip is as follows:

$$L\left(s, a, \theta_k, \theta\right) = \min\left(\frac{\pi_\theta(a \mid s)}{\pi_{\theta_k}(a \mid s)} \hat{A}^{\pi_{\theta_k}}(s, a), \quad \operatorname{clip}\left(\frac{\pi_\theta(a \mid s)}{\pi_{\theta_k}(a \mid s)}, 1 - \varepsilon, 1 + \varepsilon \right) \hat{A}^{\pi_{\theta_k}}(s, a) \right) \tag{2.10}$$

The details of algorithm is introduced in appendix B.1.2.

2.5.6.3. DPG,DDPG, MADDPG

DPG: Deterministic policy gradient creates a μ function to determine the action instead sampling the action from probability distribution in PG.

DDPG: DDPG[Lil+15], abbreviation for Deep Deterministic Policy Gradient, is a model-free off-policy actor-critic algorithm, combining DPG with DQN. The original DQN works in discrete space, and DDPG extends it to continuous space with the actor-critic framework while learning a deterministic policy. Actor-Critic framework will create two networks named as actor network, which outputs the action, and critic network, which conducts the training and update of actor network.

MADDPG: Multi-Agent DDPG[Low+17] extends DDPG to an environment where multiple agents coordinate only local information to complete tasks. From the perspective of an agent, the environment is unstable because the policies of other agents will quickly escalate and remain unknown. MADDPG is a critical model of actors, which has been redesigned to deal with this ever-changing environment and the interaction between agents.

Actor update The gradient of reward of MADDPG is shown below:

$$\nabla_{\theta_i} J\left(\boldsymbol{\mu}_i\right)=\mathbb{E}_{\mathbf{x}, a \sim \mathcal{D}}\left[\left.\nabla_{\theta_i} \boldsymbol{\mu}_i\left(a_i \mid o_i\right) \nabla_{a_i} Q_i^\mu\left(\mathbf{x}, a_1, \ldots, a_N\right)\right|_{a_i=\boldsymbol{\mu}_i\left(o_i\right)}\right] \tag{2.11}$$

Where D is the memory buffer for experience replay, containing multiple episode samples.

Critic update Loss function is used for updating of critic network. The form of loss function is shown as follow:

$$\mathcal{L}\left(\theta_i\right)=\mathbb{E}_{\mathbf{x}, a, r, \mathbf{x}^{\prime}}\left[\left(Q_i^{\boldsymbol{\mu}}\left(\mathbf{x}, a_1, \ldots, a_N\right)-y\right)^2\right], \quad y=r_i+\gamma Q_i^{\boldsymbol{\mu}^{\prime}}\left.\left(\mathbf{x}^{\prime}, a_1^{\prime}, \ldots, a_N^{\prime}\right)\right|_{a_j^{\prime}=\boldsymbol{\mu}_j^{\prime}\left(o_j\right)} \tag{2.12}$$

Where μ' is the target network (policy) with delayed updated parameters.

The detailed information about the DPG, DDPG and MADDPG is introduced in appendix B and A.

2.5.6.4. Value Decomposition Networks (VDN)

For multi-agent deep reinforcement learning, there is a very natural idea, which is to learn concentrated state action values but perform action through local observation to solve the non-stational environmemt problem which exists in the scneario, where agents use independet learning methods. The key idea of MADDPG is the same. However, it has been proven that its convergence is problematic, and training is extremely difficult. This requires on policy learning, which is sample inefficiency, and when there are too many agents, it becomes impractical to train fully focused critics. VDN[Sun+17] takes an approach, which learns a centralised but factored Q_{tot}. Author represents Q_{tot} as a sum of individual value functions Q_a. Each agent selected actions greedily with respect to its Q_a. However, a simple summation operator limits the representation ability of centralised action-value function. More importantly, additional environmental states are not considered during the training process.

2.5.6.5. QMIX

QMIX is a method first proposed in the paper `Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning`[Ras+18]. It is used to solve the limitation of VDN. QMIX consists of two networks, one of which is called as agent network representing each Q_a, second network is a mixing network that combines all Q_a into Q_{tot}, not as a simple sum as in VDN. Mixing network uses complex non-linear way to ensure consistency between centralized and decentralized strategies. The detailed model will be introduced in the experiment when it is combined with the experimental environment 5.2.2.1.

2.6. Platform and Library

2.6.1. GENIUS

GENIUS: An integrated environment for supporting the design of generic automated negotiators [Lin+14]. It is considered the defacto standard of AN.

2.6.2. NegMAS

NegMAS is an opensource automated negotiation platform, which can model situated simultaneous negotiations such as SCM which will be discussed separately in detail in the next section 2.6.3. The purpose of this section is to provide an overview of the key components (e.g. Mechanism, World) of this platform.

NegMAS is a python library for developing autonomous negotiation agents embedded in simulation environments. The name negmas stands for either **NEGotiation MultiAgent System** or **NEGotiations Managed by Agent Simulations**. The main goal of NegMAS is to advance the state of the art in situated simultaneous negotiations. Nevertheless, it can; and is being used; for modeling simpler bilateral and multi-lateral negotiations, preference elicitation , etc [Moh+19].

NegMAS and Mechanism NegMAS has natively implemented five mechanisms, Stacked Algernating Offers Mechanism (SAOM), single-text negotiation mechanisms(st) [Rai82], multi-text mechanisms (mt), GA-based negotiation mechanisms[KT15] and chain negotiations mechasnim (chain of bilateral negotiations). Among them, SAOM is the negotiation mechanism that is discussed and used in the experiments of this thesis. It has been introduced in detail in section Autonomous Negotiation 2.2.4. At the same time, in the related negotiation mechanism packages, some negotiators, such as AspirationNegotiator in negmas.sao, are developed as key part of the packages. These negotiation negotiator will be used as the baseline negotiators in the following experiments.

NegMAS and World A simulation is an embedded domain in which agents behave. It is represented in NegMAS by a **World**. The world in NegMAS was designed to simplify the common tasks involved in constructing negotiation driven simulations[Moh+19]. The entire simulation includes multiple simulation steps which is different from the negotiation rounds definied in 2.2.2. A simulation step can have multiple negotiation rounds. In each step, agents can be allowed to take proactive actions by performing operations worldwide, reading their status from the world, or requesting/operating negotiations with other agents.

NegMAS and Negotiator Negotiator is NegMAS's way to represent automated negotiation agent in a negotiation mechanism. Several negotiators are natively implemented in

NegMAS. AspirationNegotiator, SimpleTitForTatNegotiator and PassThroughSAONegotiator are negotiators, which are developed for SAOM. However, most Genius agents are available in NegMAS.

An overview of the main components of a simulation in a NegMAS world is shown in Figure 2.7.

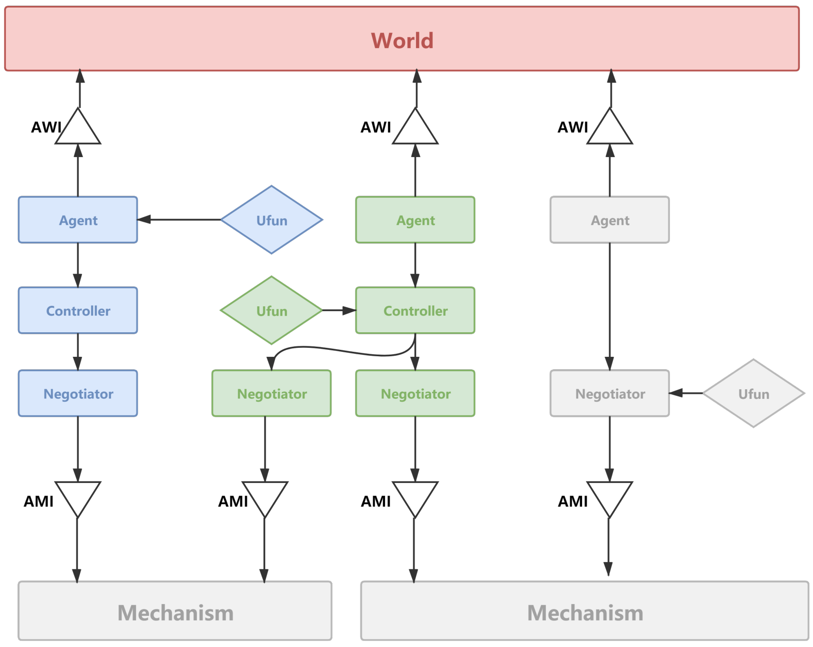

Figure 2.7.: Main components and interactive logic of a simulation in a NegMAS world, Source: Own illustration based on[Moh+19]

2.6.3. SCML

A supply chain is a sequence of processes by which raw materials are converted into finished goods. A supply chain is usually managed by multiple independent entities, whose coordination is called **supply chain management (SCM)**. SCM exemplifies situated negotiation. The SCM world was built on top of NegMAS to serve as a common benchmark environment for the study of situated negotiation [Moh+19]. This repository is the official platform for running ANAC Supply Chain Management Leagues. It will contain a package called scmlXXXX for the competition run in year XXXX. So far, there are three main different versions of SCML with different designs. In the following sections, some of the components of the three versions related to the work of this thesis will be introduced.

- SCML2020-OneShot: Agent Competition (ANAC) Supply Chain Management League OneShot trace (SCML-OneShot).
- SCML2020/2021: Standard Automated Negotiation Agent Competition (ANAC) Supply Chain Management League (SCML) 2020/2021.
- SCML2019: Standard Automated Negotiation Agent Competition (ANAC) Supply Chain Management League (SCML) 2019

2.6.3.1. SCML2020/2021 (standard scml)

SCML was originally developed as a part of NegMAS, from the version 0.4.0 it was splited as an indepedent project to research SCM. SCML realized a SCM world to simulate the SCM process. In this world, agent needs to buy input material through negotiation, manufacture them, then sell output products through negotiation[Y+21]. The strategy of an agent can be splitted as three parts: **Trading Strategy**, **Negotiation Control Strategy**, **Production Strategy**.

Trading Strategy determines the quantity (and price) bought and sold at each time step.

Negotiation Control Strategy this component is responsible for actively requesting negotiation, responding to negotiation requests and actually conducting concurrent negotiation.

Production Strategy decides what to produce at every time-step.

2.6.3.2. SCML2020-OneShot

In SCML-OneShot world, the simulation steps can be referred as days. There are multiple concurrent negotiations going on every day. Difference with the SCML2019 and SCML2020/2021, SCML-OneShot emphasizes negotiation and de-emphasizes operations (e.g. scheduling) which are also important in standard scml. The simulation in oneshot focuses on the research of concurrent negotiation. The design of SCML-OneShot ensures the following points [Y+21]:

- Agents (factory managers) consider only the negotiations in the current simulation step. Only the current concurrent negotiations can affect the agent's score. It means, regardless of the result of the negotiations, the concurrent negotiations will be ended at the end of the simulation step.
- Agents can learn over time about their negotiation partners (i.e. suppliers or consumers).

Figure 2.8 diagrams a World created by SCML-OneShot

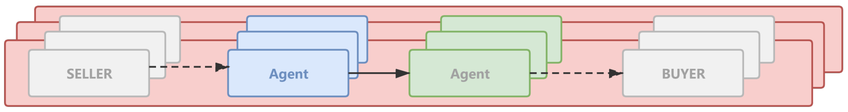

Figure 2.8.: Main running logic of a simulation in a SCML-OneShot world, Each red box represents a day. Broken lines represent exogenous contracts while connected lines are negotiations (about the intermediate product). All negotiations conducted in a given day are about deliveries on the same day. Source: Own illustration based on[Y+21]

Production Graph represents the products defined in the world and the process of converting between them. There are three product types: raw-material, intermediate-product and final-product and two manufacing processes for converting the raw materail into the intermediate product and for converting the intermediate product to the final product.

Factories are entities which in the SCM world convert input products into output products by running manufacturing process on their production lines. A special point is that all processes require zero time to complete.

Agents The agents in the SCM world function as factory managers controlling the negotiations between factories.

Conctracts include normal contracts which are signed agreements between agents or exogenous contracts, which are contracts signed between agent and system entities (BUYER and SELLER).

Utiliy Function The utility function of an agent specifies its preferences over possible outcomes of a negotiation. The utility function is well defined in SCML-OneShot. It is simply the total money it receives minus the money it pays for buying input product, production cost, storage and garbage collection costs and delivery penalties. It is called OneShotUfun in SCML-OneShot world. This utility can be set as a part of reward function of RL-agent. The complete information of the utility function will be covered in Chapter 4 "Analysis" 4.2.5.2.

Negotiation Mechanism The negotiation mechanism is a variant of the alternating offers protocol. It involves two agents, who take turns making offers for a finite number of rounds. One agent opens the negotiation with an offer, after which the other agent takes one of the actions (Accepts, Counter-offer, Walks away). All actions are also introduced in SAOM 2.2.4. This process repeats until an agreement or a deadline is reached. A key difference between the variant of the protocol used in the SCM world and other implementations of the alternating offers protocol is that in the first round of negotiations, both agents propose an offer, and then one of these offers is chosen at random to be the initial offer. This trick was designed to prevent the mechanism from degenerating into an ultimatum game.

Bulletin Board The world contains a public bulletin board. It contains both static and dynamic information about the game and all factories. The static information includes the game settings, product information, catalog prices and trading prices. The dynamical information includes a breach list and financial reports.

Simulation World Each simulation in SCM world runs for multiple days. Before the first day, each agent is assigned a private production cost (m_f). During each day[Y+21]:

- Conduct multiple rounds of negotiations between agents.
- Execute all contracts.
- Update the bulletin board to reflect new financial reports, transaction prices and external contract summaries.

2.6.4. Tensorflow

Tensorflow is an end-to-end open source platform for machine learning. It has a comprehensive and flexible ecosystem of tools, libraries and community resources, allowing researchers to promote the latest developments in ML, and allowing developers to easily build and deploy ML-supported applications[Mar+15]. In this thesis, it is used for creating neural networks (in MADDPG and stable-baselines) and summarizing the results.

2.6.5. PyTorch

PyTorch is an open source machine learning library and framework which performs immediate execution of dynamic tensor computations with automatic differentiation and GPU acceleration, and does so while maintaining performance comparable to the fastest current libraries for deep learning[Pas+19]. While considering performance, it is also easier to apply and debug. In this thesis, it is used for creating an easy-to-understand network (in QMIX).

2.6.6. OpenAI Gym

OpenAI Gym is a toolkit for developing and comparing reinforcement learning algorithms [Bro+16]. To create custom environments, they just need to follow its interface.

2.6.6.1. Environment

The core gym interface is `Env`, which is the unified environment interface. The following are the Env methods that developers should implement [Bro+16].

STEP Run one timestep of the environment's dynamics. When end of episode is reached, `reset()` is calld to reset this environment's state. Accepts an action and returns a tuple (observation, reward, done, info).

- observation (object): agent's observation of the current environment, such as frame of the game.

- reward (float): amount of reward returned after previous action, such as 1 when action is go to left.
- done (bool): whether the episode has ended, in which case further step() calls will return undefined results, such as agent is dead in game, as True.
- info (dict): contains auxiliary diagnostic information (helpful for debugging, and sometimes learning), such as goal of agent.

RESET Resets the environment to an initial state and returns an initial observation. This function should not reset the environment's random number generator(s). Random variables in the environment's state should be sampled independently between multiple calls to `reset()`. Each call of `reset()` should yield an environment suitable for a new episode, independent of previous episodes.

RENDER Define how to display the output of the environment. Multiple modes can be used:

- human: Render to the current display or terminal and return nothing. Usually for human consumption
- rgb_array: Return an numpy.ndarray with shape (x, y, 3), representing RGB values for an x-by-y pixel image, suitable for turning into a video.
- ansi: Return a string (str) or StringIO.StringIO containing a terminal-style text representation. The text can include newlines and ANSI escape sequences (e.g. for colors).

CLOSE Override close in the subclass to perform any necessary cleanup. Environments will automatically `close()` themselves when garbage collected or when the program exits. Save datas at the end of the program.

SEED Sets the seed for this env's random number generator(s). It is useful for reproducing the results.

```
ob0=env.reset() #sample environment state, return first observation
a0=agent.act(ob0) #agent chooses first action
ob1,rew0,done0,info0=env.step(a0) #environment returns observation,
#reward, and boolean flag indicating if the episode is complete.
a1=agent.act(ob1)
ob2,rew1,done1,info1=env.step(a1)
...
a99=agent.act(o99)
ob100,rew99,done99,info2=env.step(a99)
# done99 == True => terminal
```

Listing 2.1: Logic of OpenAI Gym Interaction

From Listing 2.1, user can get the logic of interaction in OpenAI Gym.

2.6.6.2. Stable Baselines

The **Stable Baselines** are developed in the project stable-baselines[Hil+18]. The project implemented a set of improved implementations of RL algorithms based on OpenAI Baselines[Dha+17]. All implemented algorithms with characteristic discrete/continuous actions are shown in 2.1 and would be tested on the experiments.

Name	**Box**	**Discrete**
A2C	Yes	Yes
ACER	No	Yes
ACKTR	Yes	Yes
DDPG	Yes	No
DQN	No	Yes
HER	Yes	Yes
GAIL	Yes	Yes
PPO1	Yes	Yes
PPO2	Yes	Yes
SAC	Yes	No
TD3	Yes	No
TRPO	Yes	Yes

Table 2.1.: stable baselines algorithms

2.6.7. Ray

An open source framework that provides a simple, universal API for building distributed applications. Ray is packaged with the following libraries for accelerating machine learning workloads[Mor+17]. Due to the complexity of training, use it to build a distributed training environment for parallel distributed training.

- Tune: Scalable Hyperparameter Tuning
- RLlib: Scalable Reinforcement Learning
- RaySGD: Distributed Training Wrappers
- Ray Serve: Scalable and Programmable Serving

3. Related Works

In this chapter, the topics related to the work of this thesis are introduced and discussed. The topics include mainly three parts: **Heuristic Negotiation Strategies for Autonomous Negotiation**, **Reinforcement Learning used in Autonomous Negotiation** and **Challenges in Deep Reinforcement Learning**. Several time-based, behavior-based and concurrent negotiation methods are discussed in the section heuristic negotiation strategies. The content of the second part is about the RL framework used in autonomous negotiation. Finally, some frequently appearing challenges when applying deep reinforcement learning are specified in the section challenges in deep reinforcement learning.

3.1. Heuristic Negotiation Strategies for Autonomous Negotiation

3.1.1. Time-based Strategy (Aspiration Negotiator)

Aspirations are the specific goals in a negotiation that a negotiator wishes to achieve as part of an agreement. Empirical evidence has shown that negotiators with higher aspirations tend to achieve better bargaining results.. First, aspiration of negotiator can help to determine the outer limit of what negotiator will request. Second, optimistic aspirations can make the negotiators to work harder[Sch04]. Many aspiration negotiators are time dependent, such as **Boulware** (with concession factor $e = 1/2$), **Linear** ($e = 1$) and **Conceder** ($e = 2$)[FSJ98]. Boulwarism is the tactic of making a "take-it-or-leave-it" offer in a negotiation, with no further concessions or discussion. It was named after General Electric's former vice president Lemuel Boulware, who promoted the strategy[Pet91].

At every round, the agent (negotiator) calculates their decision utility which determines

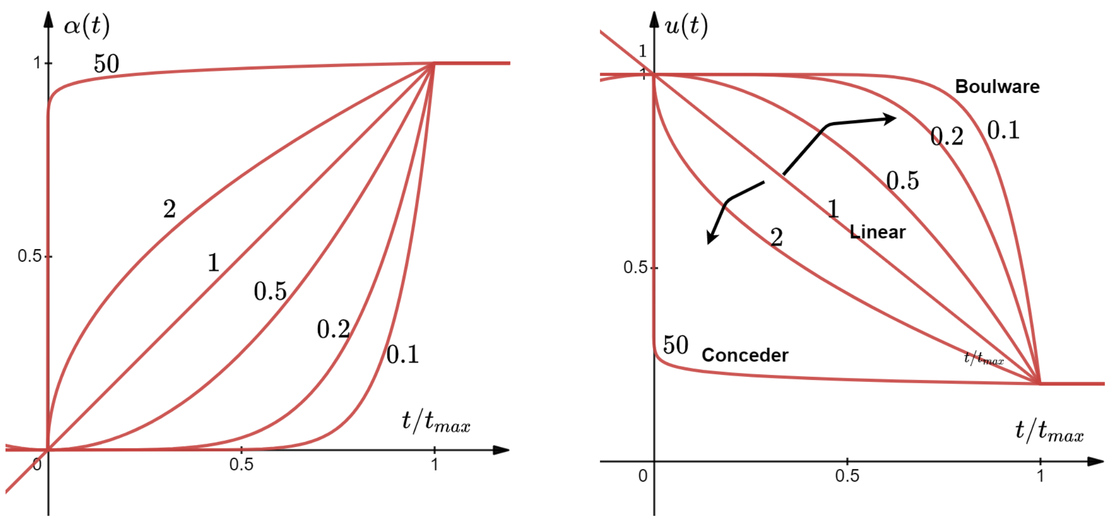

Figure 3.1.: Convexity degree of $\alpha(t)$ (left) and $u(t)$ (right) with concession factors $e = [0.1, 0.2, 0.5, 1, 2, 50]$.

whether they accept an offer or not. For time-dependent agent, the utility is:

$$u(t) = P_{min} + (P_{max} - P_{min})(1 - \alpha(t)) \tag{3.1}$$

P_{max} and $P_{min} \in [0, 1]$ means the minimum utility and the maximum utility, respectively. Frequently, $\alpha(t)$ is parametrized as a time-depedent polynomial function[Cha20; FSJ98]:

$$\alpha(t) = \kappa + (1 - \kappa) \left(\min\left(t, t_{\max}\right) / t_{\max}\right)^{1/e} \tag{3.2}$$

Where e is the concession factor. For simplicity, k is often set to 0. Figure 3.1 diagrams the convexity degree of $\alpha(t)$and utility value $u(t)$ based on the $\alpha(t)$.

In addition, it is obvious, for single issue, the value of issue j as the offer at time t sent by agent a to agent b can be formed as follows:

$$x^t_{a\to b}[j] = \begin{cases} \min^a_j + \alpha^a_j(t) \left(\max^a_j - \min^a_j\right) & \text{if } u^a_j \text{ is decreasing} \\ \min^a_j + \left(1 - \alpha^a_j(t)\right) \left(\max^a_j - \min^a_j\right) & \text{if } u^a_j \text{ is increasing.} \end{cases} \tag{3.3}$$

Where u_j denotes utility value of issue j. The minimum and maximum value of issue j are represented by $\min_j$ and $\max_j$, respectively.

The offer will always be within the value range ($[min_j, max_j]$), the initial constant will be given at the beginning, and before the deadline is reached, the strategy will suggest an outcome with the reserved value, which is usually the value with minimum utility.

3.1.2. Behavior-based Strategies

Behavior-based and imitation bidding strategies observe the opponent's behavior and decide what to offer and accept. A well-known behavior-based strategy is **tit-for-tat**. This strategy is to act cooperatively first, and then mirrors what other players did in the previous round. It is a very robust strategy and has three main following features[BHJ13; Cha20]:

- It is never the first to defect (i.e. As long as the opponent also plays well, it will play well).
- The opponent's defection may lead to retaliation.
- It can forgive after retaliation.

There are two different actions from the opponent that could be considered "nice": The opponent concedes according to its utility function or offers more utility to the agent, according to the agent's utility function. Agent can also choose one of this two options. Therefore, there are four different ways for tit-for-tat agent to reciprocate. When agent reciprocate according to the agent's own utility function and opponent makes a bid that is better for the agent, in return, the agent will produce a bid of lower utility for itself. This is a cooperative behavior. But, agent also considers the utility of its opponent. Bayesian learning is used to construct an opponent model. For nice-tit-for-tat, it uses the opponent model to make an estimate of the location of the Nash point of the negotiation scenario. It uses improved moving ratio to avoid far too nice strategy. For example, if the opponent has made an offer that is 0.7 on the way to the Nash point, the agent will respond in kind by approaching from other side, making an offer that is 0.3 away from the Nash point[BHJ13]. In contrast, if opponent makes a bid that is worse for the agent, in return, the agent will produce a bid of higher utility for itself. It means, opponent is defect, and agent retaliates against the opponent.

3.1.3. Concurrent Negotiation Strategy (CNS)

In a concurrent negotiation environment, an agent will negotiate with many opponents at the same time (one-to-many). One issue is how to coordinate all these negotiations. The author of the paper [Wil+12] designed an intuitive model with two key parts, namely the **Coordinator** and **Negotiation Thread** to deal with this problem.

Negotiation Threads The strategy of each negotiation thread is an extension of a principled, adaptive bilateral negotiation agent. This agent was designed to be used in a similarly complex environment, but only for negotiations against a single opponent. In this thread, agent i performs gaussian process regression to predict the future concession of its opponent. With the probability distribution $p_{i,t}(u_i)$ over the utility, prediction of the future concession of opponent can be captured (see [Wil+11] for details). The probability distribution is then passed to the coordinator.

Coordinator The role of the coordinator is to calculate the best time t_i to reach agreement and utility value u_i at that time, for each thread. To do so, it uses the probability distribution received from the individual threads, which predict future utilities offered by the opponents.

Related components and data flow are diagrammed in the figure 3.2.

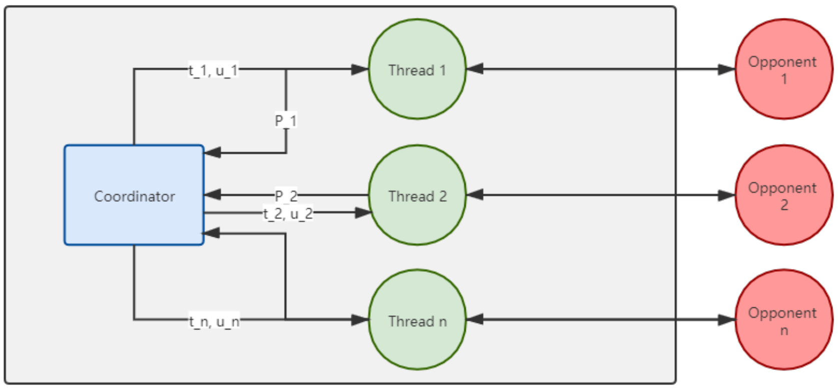

Figure 3.2.: Architecture of the concurrent negotiation agent, best time: t_i and utility value: u_i, probability distributions: P. Own illustration based on[Wil+12].

3.1.4. Conclusion

From the analysis of the heuristic negotiation strategy in automated negotiation, we can extract some important parameters, such as time and the opponent's offer, these parameters can be regarded as important information that affects the negotiation process.

3.2. Reinforcement Learning used in Autonomous Negotiation

RLBOA Bak et. al. proposed [Bak+19], a modular reinforcement learning framework for autonomous negotiating agents. This framwork implemented an agent that used tabular Q-Learning on the compressed state and action space to learning bidding strategy which is one of modules BOA proposed in the paper [Baa+14]. According the utility function of agent, the outcome space is discretized into evenly spaced bins. The utility bin can be described as part of the state (i.e., state $s_t = \{ub(w_A^t), ub(w_B^t), ub(w_A^{t-1}, ub(w_B^{t-1}), t\}$, $ub(w_A)$ denotes the utility bin of agent for its own bid, $ub(w_B)$ denotes the utility bin of agent for the opponent's bid). Negotiation strategy was split into three modules: **bidding strategy**, **opponent model**, and **acceptance strategy**. RLBOA maps the multi-dimensional contract space to the utility axis, which enables compact and universal descriptions of states and actions. Hence, the action space of this framework is discrete. The model is diagrammed in the figure 3.3.

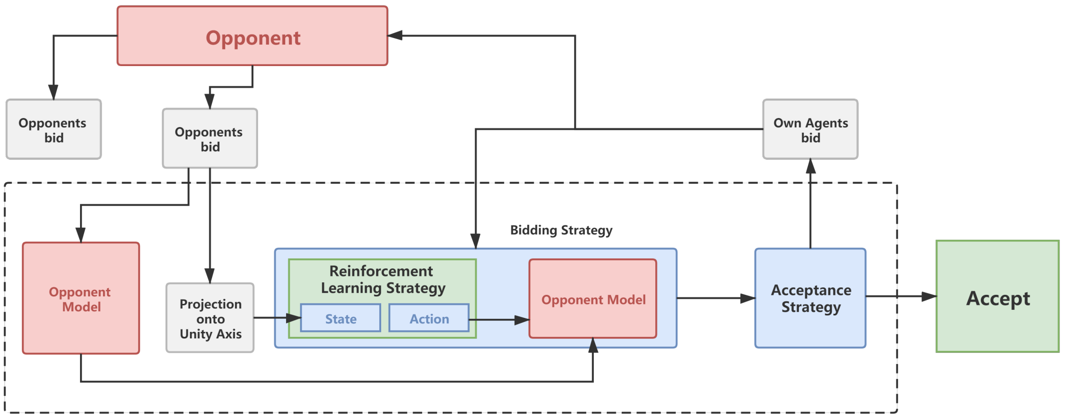

Figure 3.3.: A schematic overview of the RLBOA-framework (within the dashed box), Source: Own illustration based on[Bak+19].

ANEGMA[Bag+20]: Bag et. al. proposed a novel DRL-inspired agent model called ANEGMA, which allows the buyer to develop an adaptive strategy to effectively use against its opponents (which use fixed-but-unknown strategies) during concurrent negotiations in an environment with incomplete information. The architecture of ANEGMA is shown in Figure 3.4. The agent uses an actor-critic architecture with model-free reinforcement learning. The method supervision from synthetic market data is adopted to pre-train the strategy. Especially, the proposed automated agents that can adapt to different settings without the need to be pre-programmed.

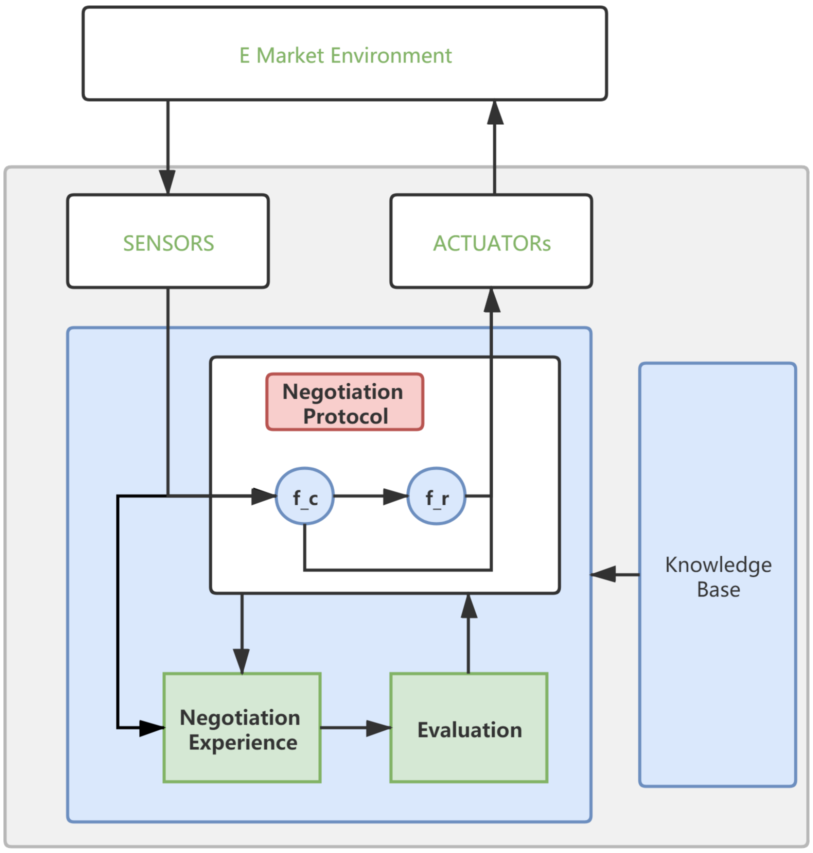

Figure 3.4.: The Architecture of ANEGMA. Source: Own illustration based on[Bag+20].

3.2.1. Conclusion

A lot of work has focused on the application of RL in autonomous negotiation. The method of **decoupling strategy** can simplify the training process and make the behavior of the agent more clearly analyzed. The method **supervision from synthetic market data** is an impressive idea that can speed up the training of agents and reduce the exploration time required for learning during the negotiation process.

3.3. Challenges in Deep Reinforcement Learning

3.3.1. Sparse Reward

For financial problem, the reward is usually profit and an appropriate reinforcement learning signal should be based on it. However, when an agent learns the strategy just based on profit, the reward is too sparse. Hence, the reward function is needed to provide more frequent feedback. Methods **Reward Shaping**, **Curiosity Driven** and **Imitation Learning** proposed to solve sparse reward problem, will be introduced in this section.

Reward Shaping[Wie10] is a method for engineering a reward function in order to provide more frequent feedback on appropriate behaviors. Providing feedback during early learning is crucial, so try promising behaviors as early as possible. This is necessary in large domains, where reinforcement signals may be few and far apart. If a method added shaping rewards in a way, it need to guarantees the optimal policy maintains its optimality. Ng et al. proposed a method with a new concept potential function $\Phi()$ to guarantee it. Hence, reward shpaing f is described as $f(s, s') = \gamma\Phi(s') - \Phi(s)$ over the stats[NHR99]. The form means shaping reward f for transitioning from state s' to s is defined as the discounted change in this state potential. Based on the value function mentioned in the section 2.5.2, the augmented value function is closely related to the original and is described as $V'(s) = V(s) - \Phi(s)$. It is obvious to set potential function $\Phi(s) \approx V(s)$. This intuition is strengthened by results presented by Wiewiora in the paper [Wie03].

Curiosity Driven [Pat+17] The author designed a new intrinsic reward signal that describes the agent's familiarity with the environment. The policy outputs a sequence of actions to maximize that intrinsic reward signal. In addition to intrinsic rewards, the agent optionally may also receive some extrinsic reward from the environment. Intrinsic rewards encourage agents to actively explore the environment, instead of staying in place due to lack of reward signals.

Imitation Learning[Hes+17] Inverse reinforcement learning (IRL) is a different approach of imitation learning, where the main idea is to learn the reward function of the environment based on the expert's demonstrations, and then find the optimal policy.

3.3.2. Non-stationary environment

Traditional RL research assumes that environment dynamical (i.e., MDP parameters) are always fixed (i.e., stationary). However, this assumption is not realistic in many real-world environment. In SCM world, for instance, factories' can change their negotiation strategy during the simulation. The difficult question is how to deal with non-stationary rewards and non-stationary transition probabilities between system stats. Centralized training decentralized execution[Low+17] is a method for multi-agent learning under non-stationary environment. Besides, a method called **Context Q-learning**[PKB20] first detects the changes of environment with a detection algorithm. Using the results of the detection, this method can estimate the strategy of the new environment model, or if the environment model has been previously experienced, the learned strategy can be improved.

3.3.3. Huge size of action space

In many real-world environment, the tasks involve large numbers of discrete actions. Traditional RL methods are difficult or even often impossible to apply to solve the problem. One proposed approach in the paper [Dul+16] embed the large discrete actions in a continous space with prior information about the actions. Then, the action can be selected using approximate nearest-neighbor method. The lookup complexity is logarithmic-time relative to the number of actions.

Learning algorithms, which used to solve continuous action space do not need to calculate Q value for every state-action pairs. It maintains just a policy π and can output continuous action. One method is to replace the large discrete action space by continuous action space.

Additionally, discrete actions can be determined based on the results.

3.3.4. Conclusion

There are many challenges in deep reinforcement learning. Although many solutions have been proposed, they are still difficult to apply, but they still have certain experimental significance. In this section, the work related to the three typical challenges helped understand and solve many of the problems in the work of this thesis. For example, in oder to solve the challenge of non-stationary environment, method centralised training decentralised execution is adapted in the training algorithm of multi-agent. For challenge of huge size of action space, one solution is that algorithms for continuous action space are used. For the design of the reward function, reward shaping helps to understand the shaping reward theoretically. This is a way to solve the sparse reward challenge.

4. Analysis

Two environments are developed for comparing the DRL algorithms used in this thesis: **single-agent bilateral negotiation environment** (SBE) and multi-agent **concurrent bilateral negotiation environment** (MCBE). The details are described in section 4.1.3 and 4.2.3. In addition to these environments, some methods have been implemented to make the training logic clearer, such as `Game` in section 4.1.4 and `Scenario` in section 4.2.4.

4.1. NegMAS with OpenAI Gym

NegMAS has implemented some negotiation mechanisms and specific simulated world, such as SAOM and SCML (now as an independent project). In order to compare the algorithms in a specific simulated world more easily, an interface is needed to connect NegMAS and RL algorithms. This interface and all algorithms can be rewritten from scratch, but it is very time-consuming and not ideal. The second option is to implement some interfaces of an existing RL framework, which will reduce the required work. OpenAI realizes the environmental standardization and comparison of algorithms with the help of toolkit OpenAI Gym[Bro+16]. Although OpenAI Gym is not enough to complete the work in this thesis, the baseline algorithms and the environmental interface in the package greatly speed up the work. In this section, the implementation of training environment and assisted methods used in bilateral negotiation will be presented.

With the help of OpenAI Gym, a bilateral negotiation environment can be developed on top of SAOM from NegMAS to study reinforcement learning algorithms. The package **Stable Baseline** 2.6.6.2 implemented many baseline algorithms, which can be easily tested in a custom environment.

4.1.1. Configuration

4.1.1.1. Negotiation Issues

NegMAS provoides some classes and methods to design issues flexibly. In SBE, following issues are used:

- **PRICE:** Integer between two values, such as (10, 20)
- **QUANTITY:** Integer between two values, such as (1, 10)
- **TIME:** Relative step between zero and maximal step.

In the section "Experiment" 5.1.2 of Chapter "Methods and Experiments", the configuration of the training environment will be listed in detail.

4.1.2. Model

The model consists of five parts, **environment SBE**, **negotiation game**, **negotiation mechanism**, **negotiator** and **RL trainer**. Except for the negotiation mechanism mentioned in sections 2.2 and 2.6.2, others parts will be introduced step by step in the following sections.

First, we give a brief introduction of the five parts in this section. **Environment SBE** inherits from `gym.env` and implements the interfaces, mainly the step function. **Negotiation Game** controls the logic and several properties of the negotiation (e.g. negotiaton issues, type of learning strategies)and provides the functions and parameters required by training algorithms. **Negotiation Mechanism** is realized in the simulator NegMAS. **Negotiator** is a general class of negotiators, which can execute negotiation behavior and have learning ability in SBE. **Trainer** is the true learnable agent corresponding to the reinforcement learning algorithm, which receives observation, state from SBE. After training and feedward calculation it sends the action to SBE. Then, the interactive agent in the environment will execute this action. The entire model is shown in 4.1.

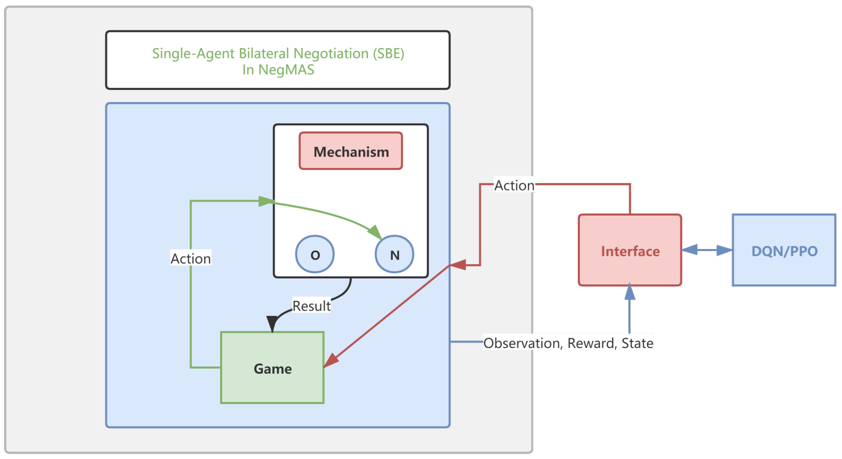
Single-Agent Bilateral Negotiation (SBE)
In NegMAS
Mechanism
Action
O
N
Result
Game
Action
Interface
DQN/PPO
Observation, Reward, State

Figure 4.1.: Model for single agent bilateral negotiation based on NegMAS

4.1.3. Single-Agent Environment

Since the default interface of the OpenAI Gym environment was designed for single-agent as standard, we only need to examine how the SBE can be represented via this interface and controlled by the controller. The interface methods of SBE are therefore defined explicit below.

STEP First, sets up but does not perform the action received from trainer for the negotiator. Then, runs the negotiation mechanism, such as SAOM, for one step. All actions will be performed by the negotiation mechanism. Finally, this function returns four parameters.

- Observation: Offer proposed by opponent and current relative time.
- Reward: Utility value of the current offer and extra reward when an agreement is reached.
- Done: Reaches the final state or there is no agreement within the maximum running time.
- Info: State of the negotiation mechanism, extra info (i.e., state from negotiation mechanism (e.g, SAOMechanism)) used for evaluation.

RESET resets the environment to an initial state and returns an initial observation, which contains negotiators' initial observation and other information relative to the design of the training environment. It will reset some environmental parameters at the same time, such as time and current step and creates a new negotiation mechanism session.

RENDER This application is not required because there is no visual output.

CLOSE Save debugging information (e.g., historical offer) about the training process.

SEED Sets the seed for the env's random number generator(s), such as random generator in negotiation mechanism.

4.1.4. Game

In addition to implementing the official OpenAI Gym `Env` interface, class **Game** is designed to control the entire negotiation mechanism. The purpose of this design is to reduce the modification of negotiatioin mechanism in NegMAS. In this class, there are some parameters, which are received from the mechanism in NegMAS and passed to the RL algorithms as additional information (e.g., goal of learning). The two main methods are defined below.

STEP Checks the state of `Game`, runs the negotiation mechanism for one step.

STEP_FORWARD realizes the key logic for the running of the negotiation mechanism, because negotiator can learn different strategies (acceptance and offer strategy) in SBE.

4.1.5. Challenges of the environment

Although OpenAI Gym provides a unified interface for custom environments, it has some problems, which cannot be directly solved by the interface. These problems occurred during environmental design and will be listed and discussed in the following sections.

4.1.5.1. Design of Action Space and Observation Space

One relevant consideration is related to RL In [Bak+19], the author study a modular RL based on BOA (Bidding strategy, Opponent model and Acceptance condition) framwork which is an extension of the work done in [Bak+19]. This framework of RLBOA implements an agent that uses tabular Q-learning to learn the bidding strategy by discretizing the continuous state/action space (not an optimal solution for large state/action spaces as it may lead to curse of dimensionality and cause the loss of relevant information about the state/action domain structure too) [Bag+20]. Compared with tabular Q-learning, deep reinforcement learning algorithms use neural networks to solve this problem.

There are at least two possible approaches to implementing deep reinforcement learning for this learning case:

The first method: The output size of neural network is directly related to the size of the action space, in other words, it is equal to the size of the outcome space.

The second method: Discrete action space is replaced by continuous action space. Before applying the action, filter invalid actions and scale valid actions.

The state in this experiment is opponent's offer (PRICE, QUANTITY, TIME). When learning different strategies, the actions are different. For acceptance strategy, actions are ACCEPT OFFER, REJECT OFFER, END NEGOTIATION and NO RESPONSE. For offer strategy, actions are outcomes.

4.1.5.2. Design of Reward Function

The design of the reward function is the key point in the realization of RL algorithm. This is very easy to understand, learners learn by evaluating the value of actions. Therefore, the reward function will directly affect the learning effect, and it is very necessary to design a good reward function. In SBE, the utility function defined by `Negotiator` can be used as a calculation tool for obtaining the current offer reward, which can be intuitively set as part of the reward function. In order to encourage negotiators to sign the contract, an extra reward will be given when the contract is successfully signed. Hence, the reward function is defined as follows:

$$R_t = \begin{cases} 1 - d(u(a), u(o)) + e & \text{accept offer} \\ 1 - d(u(p), u(o)) & \text{reject, propose meaningful offer} \\ -1 & \text{otherwise} \end{cases} \tag{4.1}$$

Where $d(u(a), u(o))$ denotes the distance between of utility of an agreement and utility of optimal offer. $d(u(p), u(o))$ denotes the distance between of utility of a proposal offer and utility of optimal offer. When at the last step, agent reachs an agreement, a extra reward (e) will be added into the reward.

4.1.6. Analysis of the reinforcement learning algorithms

4.1.6.1. Policy-based vs. Value-based

Policy-based (e.g., PG, DPG and PPO) methods use a policy $\pi : s \longrightarrow a$ to output action based on state and keep the parameters in the memory. Value-based (e.g., tabular q-learning, DQN and SARSA) methods do not explicitly store any policy, only a value function or value tabular. The policy can be implicitly derived from the value function (e.g., greedy selection). A well-known RL framework A2C combined both policy-based and value-based parts.

4.1.6.2. Model-based vs. Model-free

One problem when applying the RL is whenever you are in state s and make an action a you might not know the next state s'.

For model-based approach learner has access to the model (i.e., environment or world) and knows some parameters, such as transition probability between states. Generally, if learner can predict the next state s' or reward r after learning, these approaches of this learner are model-based. In model-free learner will collect some experience and derive optimal policy. It is not given any explicit information of model.

4.1.7. Conclusion

With the help of OpenAI Gym and NegMAS, the design of SBE and the training of negotiators are not too complicated. The purpose of this part is to explore the implementation possibilities of deep reinforcement learning negotiators.

4.2. SCML with OpenAI Gym

4.2.1. Configuration

4.2.1.1. Negotiation Issues

Standard SCML Negotiation issues are multi-issues, QUANTITY, TIME and PRICE.

SCML-OneShot Negotiation issues are multi-issues, QUANTITY and PRICE. Time is not important in this simulated world. All contracts will be executed at the same step in which agents reach agreements.

4.2.2. Model

The model consists of six parts: environment `MCBE`, Scenario, World, Agent, Interfaces and MADRL algorithms (trainer). All six parts are needed to be rewritten according to SCML and OpenAI Gym. **Environment `MCBE`** is a new universal environment for multi-agent learning. The key functions have the same names as functions in SBE, such as `step`. It provides the pre-definition action spaces and observation spaces. Additionally, the function `run` is used to run the complete simulation process. **Scenario** is the same as the class `Game` designed in SBE. However, it does not consider the detailed logic of the game. The configuration of SCM world and the interactive logic of the agent with environment will be set and implemented by class scenario. **World** is the key class which simulates the SCM world. Negotiation and manufacturing process are executed by it. **Agent** is a general abstract class of agents (factory manager) which can run in MCBE. **Interfaces** are a type of classes and functions, which control communication between environment and algorithms. **MADRL algorithms (trainer)** are deep reinforcement learning algorithms (trainer) used for training multi-agent. Entire model is shown in 4.2. Using MADRL algorithms can avoid the challenge of non-stationary environment and train multi-agent together to maximize the profit of the same type of agent.

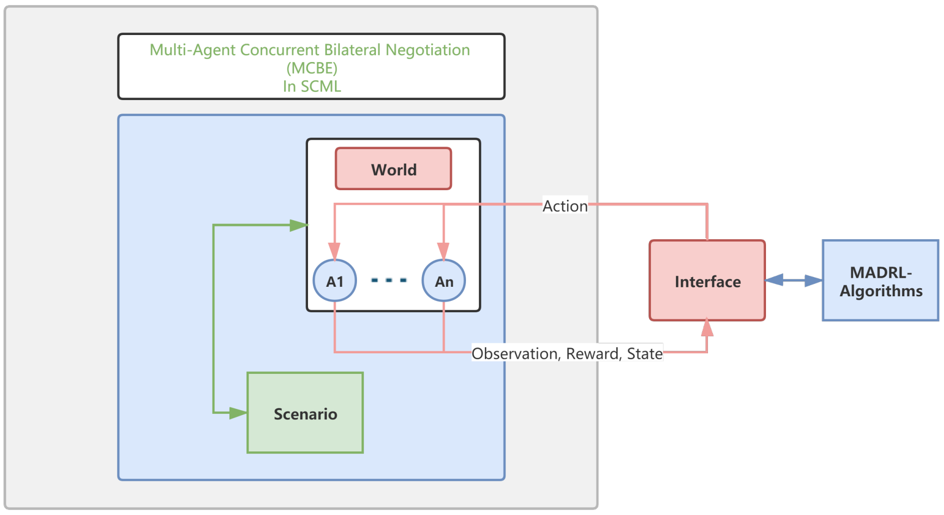
Multi-Agent Concurrent Bilateral Negotiation (MCBE) In SCML
World
Action
A1
An
Interface
MADRL-Algorithms
Observation, Reward, State
Scenario

Figure 4.2.: Model for Multi-Agent Concurrent Bilateral Negotiation based on SCML

4.2.3. Multi-Agent Environment

In order to be able to realize deep reinforcement learning for multi-agent with an OpenAI Gym Environment, the interface would have to be expanded. In the following, alternative possibilities for using an OpenAI Gym Environment for MARL are discussed. In addition to implementing the OpenAI Gym env interface methods, MCBE added a new method called `run` to execute the entire episode.

STEP Runs the simulated world for one step.

RESET Resets the environment (MCBE) and other related parameters to an initial state after every episode and returns an initial observation.

RENDER This application is not required because there is no visual output.

CLOSE This application is not needed because there is no need to save the data created by the environment.

SEED Sets the seed for this env's random number generator(s)

RUN Runs entire episode. After a negotiation step, the rewards, observations, actions, etc. are stored in the memory buffer.

- Observation: Current offer in negotiation mechanism. The observations of all agents are combined in one list. Agent can only access to its local observation during decentralised execution.
- Reward: Sum reward of all learnable agents. Cumulative reward of a single agent (4.2.5.2) is the sum of utility value of the current offer after one negotiation round, utility value of agent after one simulation step and profit of agent after the completed simulation.
- Done: Reaches the final state (last step of simulation world) or the maximum running time.
- State: State of environment. It can be replaced by `Observation`.

4.2.4. Scenario

Scenario describes the structure of simulation world. It is similar as the assisted method `Game` in SBE and provides functions for generating and resetting the world. With the help of **Scenario**, many scenarios can be created without changing of MCBE. Figure 4.3 diagrams a simple scenario.

Interface of **Scenario**contains three normal functions and four callback functions passed to MCBE.

MAKE_WORLD Creates instance of game or training world.

RESET_WORLD Sets the world to the initial state.

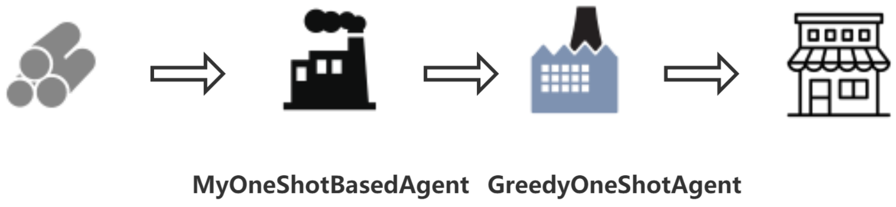

Figure 4.3.: MyOneShotBasedAgent vs. GreedyOneShotAgent

RESET_AGENT Resets agent, returns initial observation.

CALLBACK OBSERVATION, REWARD, DONE and INFO

4.2.5. Challenges of the environment

4.2.5.1. Combination with SCML

Compared with SBE, MCBE can not directly call the functions designed in official SCML. `Step` in SCML-OneShot means one simulation step. In one simulation step, many negotiation rounds will be performed. The action of the agent is a `proposal`, so it needs to be meticulous to control every round of the negotiation mechanism in the simulation world. The class `TrainWorld` inherited from SCMLOneShotWorld achieves this goal.

4.2.5.2. Design of Reward Function

In SCM world, the goal of the agent is to maximize profit at the end of the simulation. It is difficult to train an agent based on this reward signal alone. With the concept of reward shape 3.3.1 the reward signal can be split as three parts: utility value of current offer after every negotiation round, utility value of agent after every simulated step and profit at the end of simulation. The reward function of single agent a_k is defined as follows:

$$R_t^{a_k} = \begin{cases} \sum_{j=0}^{N} \left(1 - d_{n_j}(u(p), u(o))\right) & \text{in negotiation} \\ u_{a_k}\left(C^i, C^o\right) & \text{end of one negotiation} \\ p_{a_k} & \text{end of simulation, profit} \end{cases} \tag{4.2}$$

Where, for negotiation n_j, $d_{n_j}(u(p), u(o))$ denotes the distance between utility of proposal offer $(u(p))$ and utility of optimal offer $(u(p))$. N represents the maximum number of concurrent negotiations of the agent a_k. The utility function of agent (i.e., $u_a\left(C^i, C^o\right)$) is defined in the equation 4.3 from [Y+21]. It represents the **expected profit** of a factory.

$$u_a\left(C^i, C^o\right) = \sum_{c \in \bar{C}^o} (p_c q_c) - \sum_{c \in C^i} (p_c q_c) - m_a P_a - \gamma_a \text{tp}\left(p_a^i, s\right) Q_a^{i+} - \alpha_a \text{tp}\left(p_a^o, s\right) Q_a^{o+} \tag{4.3}$$

Where C^i denotes the set of input contracts plus the exogenous input contracts. C^o denotes the set of output contracts plus the exogenous output contracts. Price and quantity of the contract c are formed by p_c and q_c, respectively. Because the agent can only sell what it can produce, the set of satisfiable output contracts can be defined as $\bar{C}^o$. Q^{i+} is simply the quantity to be bought according to C^i but is never sold. p_a^i and p_a^o are the input and output products, and $tp(p, s)$ is the current trading price of product p at step s. The meaning of each term is listed below:

- $\sum_{c \in \bar{C}^o} (p_c q_c)$ The total money it earns by selling its produced outputs.
- $\sum_{c \in C^i} (p_c q_c)$ The total money it pays for buying its inputs.
- $m_a P_a$ The cost of producing the product.
- $\gamma_a \text{tp}\left(p_a^i, s\right) Q_a^{i+}$ The loss of buying too many inputs without using them immediately.
- $\alpha_a \text{tp}\left(p_a^o, s\right) Q_a^{o+}$ The penalty for failed delivery of production products.

4.2.6. Analysis of the reinforcement learning algorithms

4.2.6.1. Independent Learning vs. Centralized Learning

Non-stationary environment Traditional reinforcement learning approaches such as Q-Learning or policy gradient are poorly suited to multi-agent environments[Low+17; Ras+18; Fan+20]. One issue is that each agent's policy is changing as training progresses, and the environment becomes non-stationary from the perspective of any individual agent (in a way that is not explainable by changes in the agent's own policy). Due to the non-stationary environment, a method called centralized learning and decentralized execution is proposed to train multi-agents.

Cooperative and Competitive Independent learning agent consider just the goal of itself and can not deal with the cooperative and competitive problem. The centralized learning framework is an intuitive idea to solve this problem by changing the design of the reward function in different scenarios. Each agent is allowed to have self-defined reward function. For competitive scenario, agents have conflicting goals, the agents are rewarded while the adversary is penalized. For cooperative scenario, all agents must maximize a shared return.

4.2.6.2. MADDPG vs. QMIX

Centralised learning of joint actions (**MADDPG**) can naturally handle coordination problems and avoids non-stationarity, but is hard to scale, as the joint action space grows exponentially with the number of agents. In [Ras+18], author proposed a neural network (**QMIX**) to transform the centralised state into the weights of another neural network. This second neural network is constrained to be monotonic with respect to its inputs by keeping its weights positive. This feature makes it possible to learn when there are many agents.

4.2.7. Conclusion

Compared with SBE, MCBE consider not just single learnable RL agent. Multi-agent is the basic setting of this environment. Hence, the number of agent, the observation space and action space of single agent, the design reward are needed to be considered carefully. With the help of MCBE developers just need to focus on the configuration of training scenario. MCBE decouples well the algorithms and environment.

5. Methods and Experiments

5.1. Single-Agent Bilateral Negotiation Environment (SBE)

In this environment, the agent is a negotiator in a negotiation mechanism.

5.1.1. Independent Negotiator in NegMAS

SBE has just single learnable DRL negotiator. All RL algorithms with the correct type of action space and observation space can be tested in this specific environment. In the experiment of this thesis, some algorithms, such as DQN, PPO, A2C, from stable-baselines 2.6.6.2 are tested in four learning cases:

- single issue, acceptance strategy
- single issue, offer strategy
- multi-issues, acceptance strategy
- multi-issues, offer startegy

The training logic of DQN is shown in the figure 5.1, and the detailed description of this algorithm is shown in the appendix B.1.1.

5.1.2. Experiment

Figure 5.2 depicts a game composed of two negotiators: **RL negotiator** and **Opponent negotiator** (AspirationNegotiator). The type of opponent negotiator can be changed in the settings file. All negotiators, which inherit the base abstract negotiator class in NegMAS can

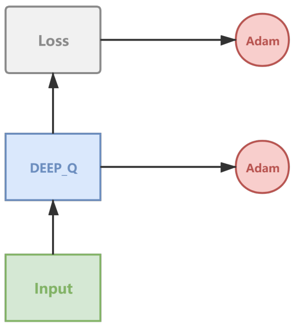

Figure 5.1.: Training logic of DQN

be configured in the experiment. AspirationNegotiator is selected as the baseline negotiator in this experiment.

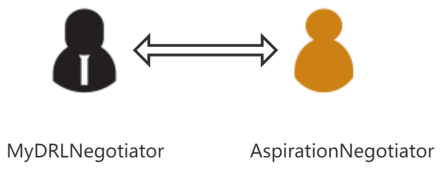

Figure 5.2.: Bilateral Negotiation Game in SBE, My Deep Reinforcement Learning Negotiator vs. Aspiration Negotiator

The negotiation mechanism is SAOM, the RL negotiator can learn two strategies called **acceptance strategy** and **offer strategy**, which will be described in detail in the following paragraphs.

Acceptance strategy The variables observed by the agent are current offer of opponent and current time (running time, or relative round of negotiation mechanism).

Offer strategy Actions of the negotiator are the set of outcomes in the outcome space. The observation space is the same as in the acceptance strategy. There is a special requirement when learning this strategy: Before feeding variables into the algorithm, observation is normalized.

The equation of normalization is defined as follows:

$$o' = (rng_{max} - rng_{min})\frac{o - o_{min}}{o_{max} - o_{min}} + rng_{min} \tag{5.1}$$

Where rng_{max} (1) and rng_{min} (-1) are the boundary value of normalization. o_{min} and o_{max} represent the boundary value of the observation value. o' and o are the normalized observation value and original observation value, respectively.

Two different categories of agents are used for the experiments.

RL agent was trained in the training environment for acceptance strategy and for offer strategy. Each strategy was learned under single issue (Table 5.1) and under multi-issues (Table 5.2) cases.

Heuristic agents (e.g., AspirationNegotiator) Negotiators were implemented in the package scml. In this experiment, the baseline negotiator is **ApsirationNegotiator** 3.1, which is a time-based heuristic negotiator.

5.1.2.1. single issue

The training environment is based on the SBE and sets concrete limits and attributes for it, which are defined in table 5.1.

Attributes	Value
Name	negotiation_env_ac_s, negotiation_env_of_s
Negotiation Mechanism	SAOMechanism
Max_Steps	100
Issue	Price(300, 550)
Competitors	[MyDRLNegotiator, AspirationNegotiator]
Utility Functions	[LinearUtility(-0.35), LinearUtility(0.25)]
Actions	[ACCEPT, REJCT, END, NO RESPONSE], Outcomes

Table 5.1.: Attributes of the training environment (sbe), single-issue

Algorithms DQN, PPO, ACER[Wan+16], A2C and DDPG are tested in the case of single issue. The mean episode reward of all algorithms for **single issue, acceptance strategy** and **single**

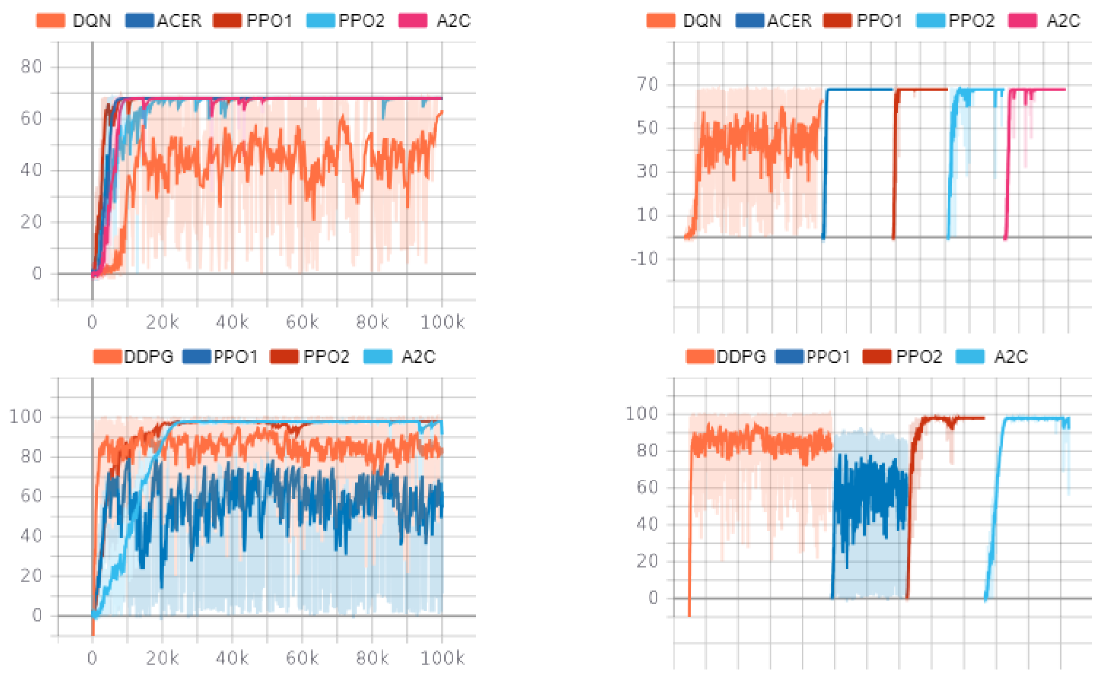

Figure 5.3.: Mean reward of **Acceptance Strategy** (top left(step), top right (wall)) and of **Offer Strategy** (bottom left(step), bottom right (wall)) under single issue negotiation

issue, offer strategy are shown in 5.3. DQN, ACER, PPO and A2C support discrete action space. Hence, these algorithms are used for training acceptance strategy, its action space is discrete. Additionally, DDPG, PPO and A2C can be used for training offer strategy, its action spaces can be considered as continuous. Mean episode reward is represented in two forms:

- step: combines mean episode reward of all algorithms into one diagram.
- wall: splits the mean episode reward of all algorithms as horizontal independente diagram.

Evaluation When training independent negotiator with SAOMechanism in the environment SBE, almost all well-known DRL algorithms have the ability to learn. ACER, PPO and A2C have learned very good acceptance strategy, the mean reward curve converged to around 70 (i.e., 68.9). However, the performance of DQN is not particularly good. For learning offer strategy, PPO2 (improved version of PPO) and A2C perform best. The reward curve converged to around 100. The result of DDPG are also valuable. Although the reward curve does not converge, it oscillates around a better strategy. Overall, normal version PPO does not perform well here.

5.1.2.2. multi issues

The training environment is almost the same as the training environment for single issue. It is based on SBE and sets concrete limits and attributes for it, which are defined in table 5.2.

Attributes	Value
Name	negotiation_env_ac_s, negotiation_env_of_s
Negotiation Mechanism	SAOMechanism
Max_Steps	100
Issue	[Quantity(0, 100), Time(0, 100), Price(10, 100)]
Competitors	[MyDRLNegotiator, AspirationNegotiator]
Utility Functions	[LinearUtility((0, -0.25, -0.6)), LinearUtility((0, 0.25, 1))]
Actions	[ACCEPT, REJCT, END, NO RESPONSE], Outcomes

Table 5.2.: Attributes of the training environment (sbe), multi-issues

In the single issue cases, the curve of mean episode reward of **multi-issues, acceptance strategy** and **multi -issues, offer strategy** are shown in 5.4.

Evaluation The most characteristics of the mean episode reward curve are the same as in single-issue cases. All algoritms (DQN, ACER, PPO1, PPO2, A2C) can learn better acceptance strategy than random (best reward of around 70). It's just that the performance of DQN is not as good as others. For learning the offer strategy, DDPG and PPO2 converged to about 80 and 100, respectively. After increasing the action space, the training time increased rapidly. From the figure, we can find the curve of A2C is very strange, after 28 thousand time steps, the mean episode reward is suddenly decreasing to zero. The reason needs to be analyzed, not particularly clear.

5.2. Multi-Agent Concurrent Bilateral Negotiation Environment (MCBE)

The agent interacting with environment have many related trainable agents (e.g. one seller, one buyer, named as trainer) as the part of learner in the model. Each seller and buyer control

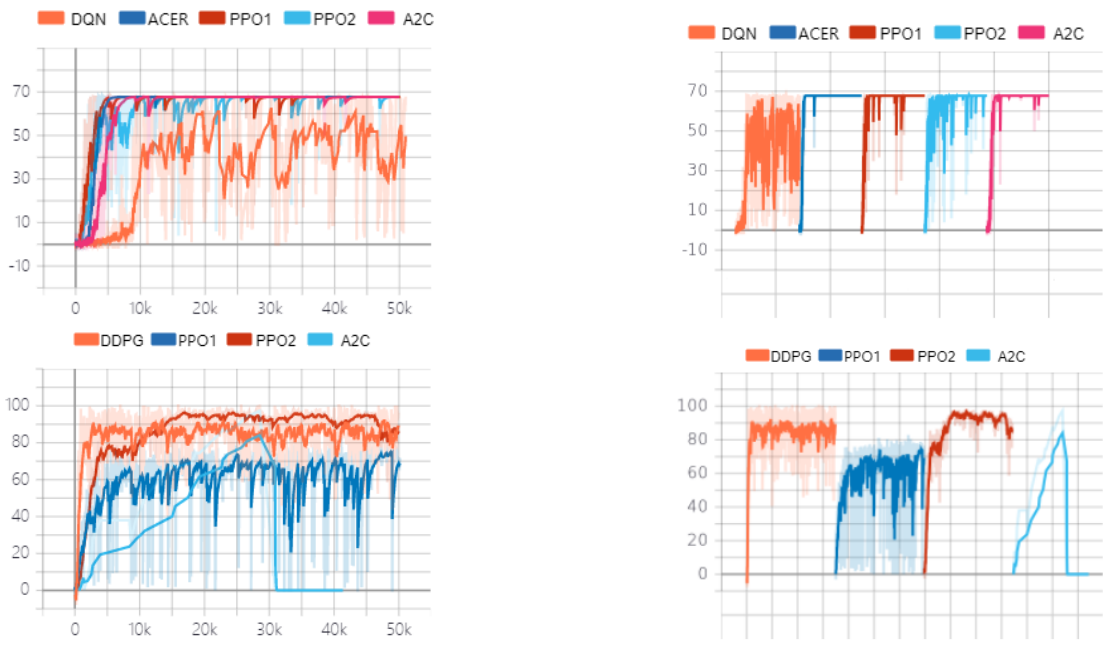

Figure 5.4.: Mean reward of **Acceptance Strategy** (top left(step), top right (wall)) and of **Offer Strategy** (bottom left(step), bottom right (wall)) under multi-issues negotiation

multiple negotiation sessions. The detail of interactive logic is shown below in 5.5

Where environment contains six factories and two system entities (SELLER and BUYER). Three RL agents (A1, A2, An) are located at two production positions. Each RL agent contains three parts:

- **Perceptor** Receives state, reward from environment and send these to Learner.
- **Actuator** Receives action from Learner and execute it in the environment.
- **Learner** In addition to connecting with **Perceptor** and **Actuator**, it manages the multiple trainers. Because for an agent, there are many concurrent negotiation sessions. A trainer is responsible for the part of the concurrent negotiations.

Based on different algorithms, the internal components of the trainer are different. In general, the trainer will handle the training logic of concurrent negotiation. It will be introduced in the specific algorithm, diagramed in figures 5.6 and 5.9.

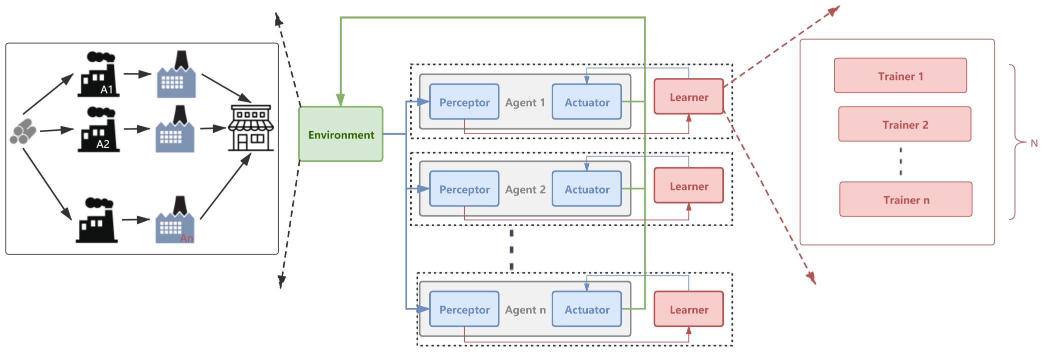

Figure 5.5.: Interactive logic based on the perspective of SCML. N: The maximum number of concurrent negotiations for a single agent

5.2.1. In Standard SCML

In this environment, the agent represents a factory manager.

5.2.1.1. MADDPG in SCML

In the standard scml environment, one problem have to be solved by MADDPG.

Problem: Dynamic Range Of Negotiation Issues At the beginning of every negotiation, the agent will determine the range which constraints values for each negotiation issue. In the experiment, the negotiation issues are **QUANTITY**, **PRICE** and **TIME**. After creating the simulated world, simulator determines the minimum and maximum values for each negotiation issue taken by the entire simulation episode, such as value of **QUANTITY** between (1, 10), **PRICE** between (0, 100) and **TIME** between (0, 100). However, for every negotiation session created inside the entire simulation episode, it has its dynamic range of negotiation issues which is determined by the agent. This question was raised based on such a situation.

From an algorithm perspective, the data flow of the model is shown in 5.6. The basic concepts of MADDPG are introduced in chapter background 2.5.6.3. Two trainers are created in MADDPG for each agent. One trainer (trainer_seller) manages sell negotiations and another (trainer_buyer) manages buy negotiations.

In the model 5.6, policy output action as input to related agent interacting with the environment,

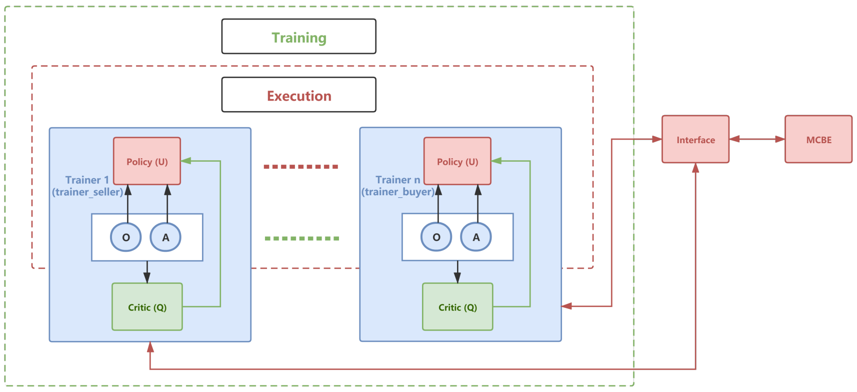

Figure 5.6.: MADDPG used in MCBE

which outputs the observation and reward as the inputs to related trainer. Two trainers are created in the experiment:

- **trainer_seller** controls all sale negotiations. The action of the trainer is the boundary value of all sale negotiation issues.
- **trainer_buyer** controls all buy negotiations. The action of the trainer is the boundary value of all buy negotiation issues.

The equation 5.2 defines the actions of two trainers (i.e., trainer_seller and trainer_buyer). The observation is defined in the equation 5.3.

$$\begin{aligned} a_{trainer\_seller} &= \prod_{i=0}^{N_{sell}} [p_{i,l}, p_{i,r}, q_{i,l}, q_{i,r}, t_{i,l}, t_{i,r}] \\ a_{trainer\_buyer} &= \prod_{i=0}^{N_{buy}} [p_{i,l}, p_{i,r}, q_{i,l}, q_{i,r}, t_{i,l}, t_{i,r}] \end{aligned} \tag{5.2}$$

Where the $p_{i,l}$, $q_{i,l}$ and $t_{i,l}$ denote the left boundary value of negotiation issues (**PRICE**, **QUANTITY**, **TIME**) of agent i. $p_{i,r}$, $q_{i,r}$ and $t_{i,r}$ represent the right boundary value of negotiation issues of agent i. N_* denotes the number of negotiations. Details of the algorithm are described in the appendix B.2.1.

$$
\begin{aligned}
o_{trainer\_seller} &= \prod_{i=0}^{N_{sell}} [e_i, m_i, c_i, n_{i,sell}, t_i] \\
o_{trainer\_buyer} &= \prod_{i=0}^{N_{buy}} [e_i, m_i, c_i, n_{i,buy}, t_i]
\end{aligned}
\tag{5.3}
$$

Where e_i denotes the economic profit of agent i, m_i represents the production costs, catalog prices of products are denoted by c_i. $n_{i,sell}$ and $n_{i,buy}$ represent the number of ongoing buying and selling negotiations of the agent i, respectively. Time is represented by t_i.

5.2.1.2. Experiment

Standard SCML is a complex simulation world, which contains various parts with specific functions. The brief description of this simulation is introduced in chapter Background 2.6.3. The experiment of this thesis focus on only the Negotiation Manager (Negotiation Control Strategy) of Decision-Maker Agent. The above mentioned method maddpg 5.2.1.1 is used in this experiment. Scenario is diagrammed in Figure 5.7.

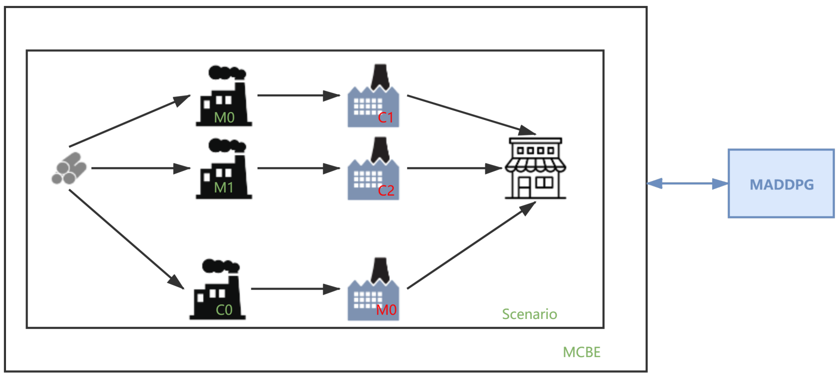

Figure 5.7.: M* represent My Component Based Agent with learner MADDPG, C* represent Opponent Agents, such as IndDecentralizingAgent

Two different categories of agents are used for the experiments:

RL agents were trained with twenty-five thousand time episodes in the training environment for the algorithms maddpg. The Rl-Agents are titled as M* in the scenario image 5.7. All RL agents have been trained together. After training, they reload the strategy from disk and run it in the standard scml world.

Heuristic agent (e.g., IndDecentralizingAgent) acts according to the strategy implemented in the scml package. The heuristic agent is named as C* in the scenario image 5.7. There are many heuristic agents, such as IndDecentralzingAgent, BuyCheapSellExpensiveAgent, MarketAwareBuyCheapSellExpensiveAgent, etc. In this experiment, IndDecentralizingAgent is used for baseline agent.

The training environment is based on the MCBE and sets concrete limits and attributes for it, which are defined in table 5.3. The simulated world is SCML2020World designed in the package standard scml. Negotiation mechanism SAOMechanism is the default mechanism of simulated world. The environment sets the negotiation speed (negotiation rate as 1) equal to the world speed. Because the negotiation speed has a self-running speed independent of the world speed. In the default world settings, the same negotiation partner can conduct concurrent negotiations with the same partner. In this experiment, in order to fix the size of the action space, this is not allowed, which means that the maximum number of concurrent negotiations is prior knowledge. RL agents can determine the best range of negotiation issues (i.e, actions of agents). Negotiation issues are the quantity, time, and price commonly used in the SCM world.

Attributes	Value
Name	scml
World	SCML2020World
Neogitation Mechanism	SAOMechanism
Max Negotiation Steps	10
Max Simulation Steps	10
Issue	[Quantity(0, 100), Time(0, 100), Price(10, 100)]
Competitors	[MyBasedAgent, DecentralizingAgent]
Negotiated Rate	1
DNSP	True
Actions	Range of Negotiation Issues

Table 5.3.: Attributes of the training environment (mcbe), DNSP: Disallow Negotiation with Same Partners, standard scml

The following paragraphs will evaulate the results based on the two questions and method maddpg.

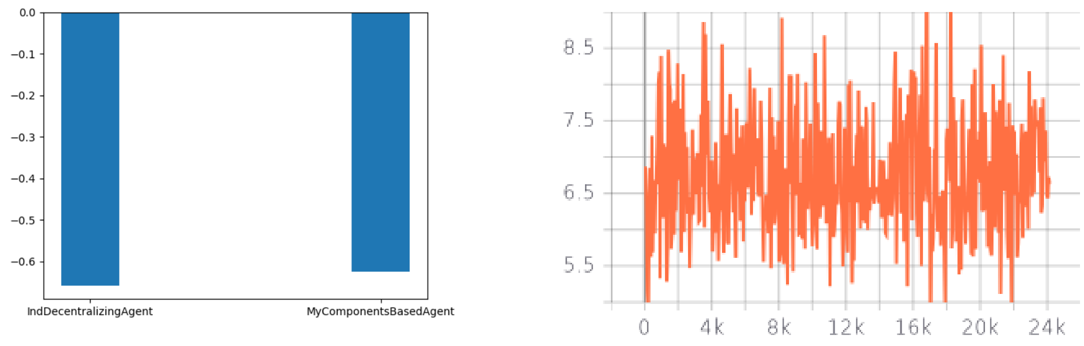

Figure 5.8.: Scores (left) of agents running in simulation world after training, Mean Episode Reward (right)

Evaluation: The results after training are shown in figure 5.8. From the mean episode reward curve (right in the figure 5.8), we can see that the agent has not learned a valuable strategy in this situation. The mean episode reward always oscillates around 7. In a sense, this is the best strategy that our RL agent can obtain. For this type of situation, only range of negotiation issues can be controlled by the RL agents, but others part, such as offering strategy is a normal heuristic strategy.

It means, merely changing the dynamic range of negotiation issues cannot effectively improve strategy of RL agents. Compared with baseline agent IndDecentralizingAgent (left in figure 5.8), the score of MyComponentsBasedAgent is almost the same.

5.2.2. In SCML OneShot

In this environment, the agent represents a factory manager.

5.2.2.1. QMIX in SCML-OneShot

The world created by SCML-OneShot is described in detail in the chapter background 2.6.3.

Question: The Offer For Every Step Unlike in standard scml **Dynamical Range of Negotiation Issues** is not controlled by agents. Hence, question in the standard SCML does not need to be discussed here. Although the design of oneshot world is very different with the standard SCML, the key question is how to find the optimal sequence action (offer for every negotiation round). When the offer of the opponent is the same as the offer of the RL-agent's negotiator, the negotiator will accept this offer.

In the current version QMIX, which is used in the experiment, one trainable agent is create for each negotiation session. When the agents are located in different locations in the scml world, the agents have different numbers of concurrent negotiation issues. For example, since the agent **A1** shown in Figure 5.5 has three consumers, the maximum number of concurrent negotiations for it is 3. Based on this value, we need to create three trainable agents (trainers) in QMIX, and each trainable agent (trainer) controls one negotiation session of interactive agent.

The data flow is shown in fig 5.9, the total number of trainers is equal to the sum of the number of concurrent negotiations of all agents. Additionally, unlike in MADDPG, in QMIX, there is only one global learner, which can control all trainers together.

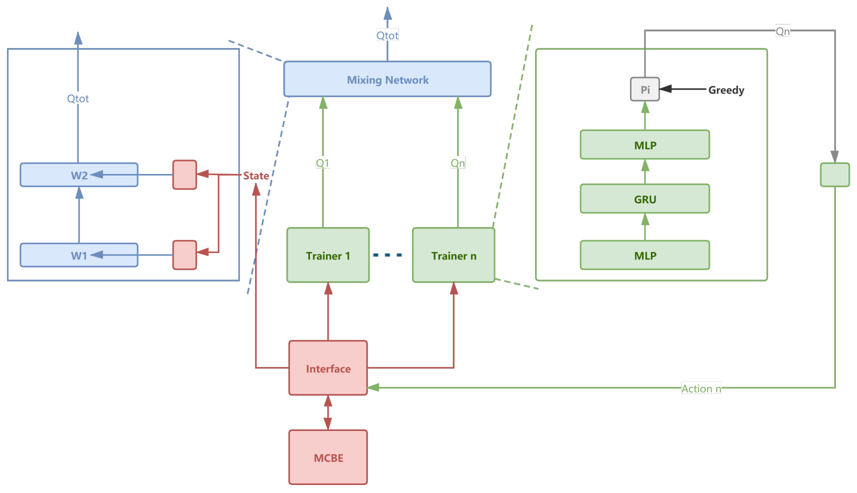

Figure 5.9.: QMIX used in MCBE

5.2.2.2. Experiment

In this experment, first, the training scenario is defined and how the various agents play is described. Secondly, the configuration of training environment is introuduced in the setting table. At the end of each experiment, the results (i.e., scores or mean episode award) will be evaluated and compared.

World created in scml-oneshot is a new simpler world which only cares about concurrent negotiation in supply chain management, and the agents used in this world can be easily transferred to standard scml. In other words, agent focus only the negotiation control strategy, which is one of the three components of a standard SCML agent2.6.3.1.

A brief description of this simulated world is introduced in chapter Background 2.6.3. This part of the experiment only focuses on negotiation. The above mentioned method qmix 5.2.2.1 is used in this experiment. Two scenarios are created: **self-play** and **play-with-others**.

self-play Scenario is diagrammed in Figure 5.10.

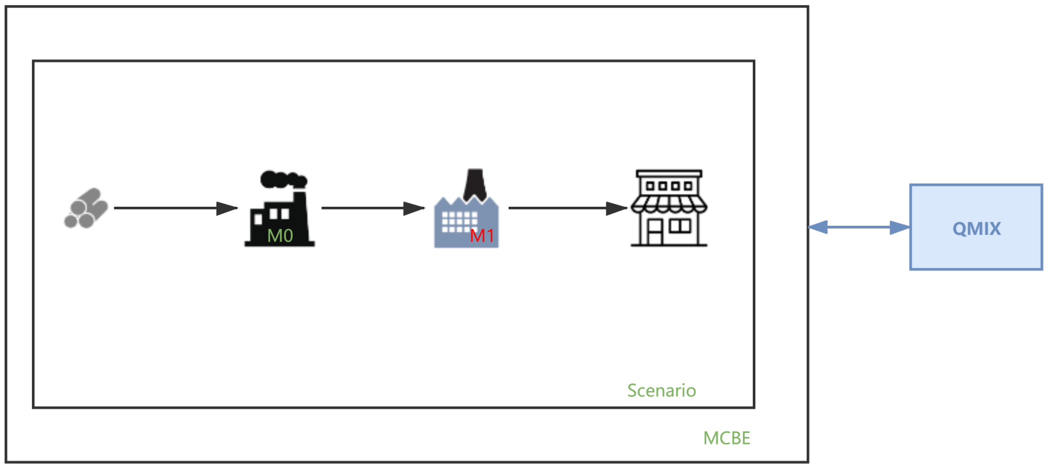

Figure 5.10.: M* represent My Component Based Agent with learner QMIX

In this scenario, just only one category of agent, which is RL agent. Because it is the self-play scenario, no other heuristic agents join in.

RL agents were trained with 10 thousands time steps in the training environment for QMIX. The RL-agents are named as M* in the scenario image 5.10. The reward is sum of reward of each agent. Because it is a cooperative game, RL agents try to maximize profit at the end of simulation.

Although the attributes of the scml-oneshot world are very different from the standard scml world, we only consider the negotiation part. Therefore, the negotiation control attributes are almost the same, such as negotiation mechanism, max negotiation steps, max simulation steps, issues, etc. The main difference is that we do not need to set the negotiation speed multiplier, because all negotiations will be finished inside each world (simulation) step. The name of simulated world is SCML2020OneShotWorld. The actions of each agent is outcomes (QUANTITY, PRICE) defined in the equation 5.4. All attributes are defined in table 5.4.

$$a_i = [q_i, p_i] \tag{5.4}$$

Attributes	Value
Name	scml-oneshot-concurrent-negotiation
World	SCML2020OneShotWorld
Neogitation Mechanism	SAOMechanism
Max Negotiation Steps	20
Max Simulation Steps	100
Issue	[Quantity(0, 100), Time(0, 100), Price(10, 100)]
Competitors	[MyAgent, MyAgent] (self-play)
Actions	Joint-Outcomes

Table 5.4.: Attributes of the training environment (mcbe), scml-oneshot, self-play

play with other agent In this scenario, the opponent agent is a heuristic agent from the scml packages, such as GreedyAgent. Scenario is diagrammed in Figure 5.11.

Two different categories of agents are used for the experiment:

RL agents The key part is same as RL agents mentioned in self-play. In addition to the normal negotiation ability, the agent can also determine whether the negotiator is his teammate. RL agents were trained with 10 thousands time steps in the training environment with QMIX. The RL-agents are named M* in the scenario image 5.11.

Baseline agents There are many baseline agents realized in scml-oneshot package, such as GreedyAgent, which uses the greedy strategy to act the negotiation action. Overall, it is a simple strategy. In the future, in order to evaluate the performance of this algorithm, more sophisticated agents will need to be tested in this environment.

The setting of this scenario is the same as in the scenario self-play. Only the competitors

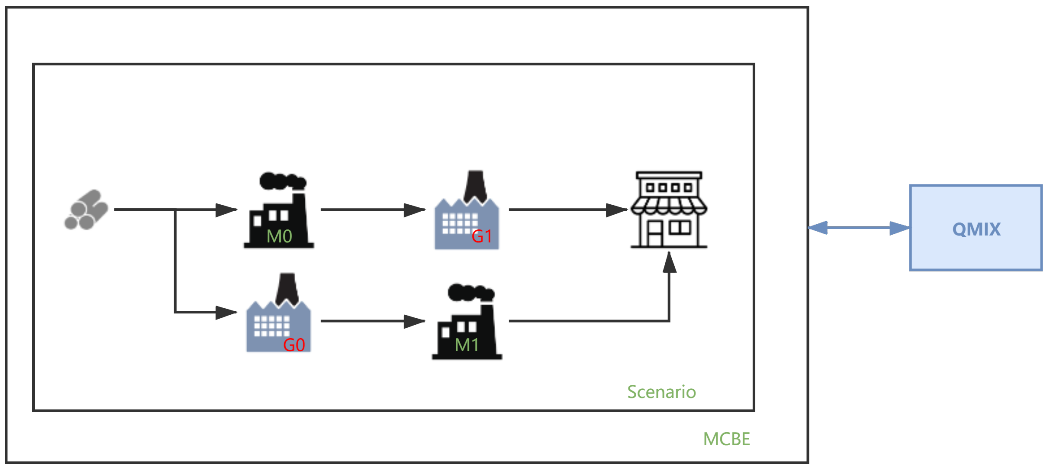

Figure 5.11.: M* represent My Component Based Agent with learner QMIX, G* represent Greedy-OneShotAgent

are changed as MyAgent vs. GreedyAgent. The negotiation mechanism is an alternative rubinstein negotiation mechanism. Negotiation issues are QUANTITY and PRICE. Based on the characteristic (all negotiations finished in each world step) of the world, time is not necessary to be considered here (i.e., set the issue time equal to the simulation step in the offer). This characteristic is briefly introduced in the section 2.6.3.2. All parameters are defined in table 5.5.

Attributes	Value
Name	scml-oneshot-concurrent-negotiation
World	SCML2020OneShotWorld
Neogitation Mechanism	Alternative Rubinstein Mechanism
Max Negotiation Steps	20
Max Simulation Steps	100
Issue	[Quantity(0, 100), Price(10, 100)]
Competitors	[MyAgent, GreedyOneShotAgent] (play-with-others)
Actions	Joint-Outcomes

Table 5.5.: Attributes of the training environment (mcbe), scml-oneshot, play-with-others

In the following paragraphs the results will be evaulate based on the two scenarios **self-paly**, **play-with-others** and method qmix raised in section 5.2.2.1.

Evaluation of self-play Curve of mean episode reward is shown in the figure 5.12. After 5 thousands time steps, the curve converges around 0. In the self-play scenario, the definition of reward is the sum reward of each agent. Therefore, while one agent gains more profits, the other agent loses more. The reward will eventually converge to a certain value, and it can be seen that the value is close to zero. The interpretation of the final convergence value will be discussed in future work. The most important question is whether this state is pareto optimal. The second question is whether RL agents can evolve on their own when the reward function is designed according to the competitive situation. This situation is very useful for training general RL agents. It means that the agent can learn more unpredictable excellent strategies through self-play.

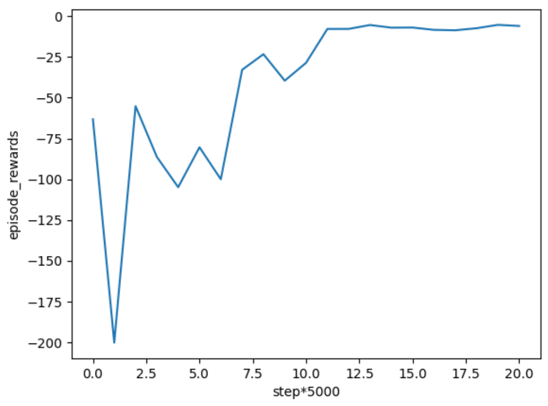

Figure 5.12.: Mean episode reward of self-play in SCML OneShot

Evaluation of play-with-others Mean episode reward curve is shown in 5.13. When the reward is only minus 400 at the beginning (random strategy), the reward curve finally converges to around 300. Obviously, RL agents have learned a good strategy. In addition, the training time is also very short. Thus, the results are very meaningful.

In the scenario **self-play** and **play with other agent**, agents learned better strategy than random. It means, method QMIX is valid in scml-oneshot world.

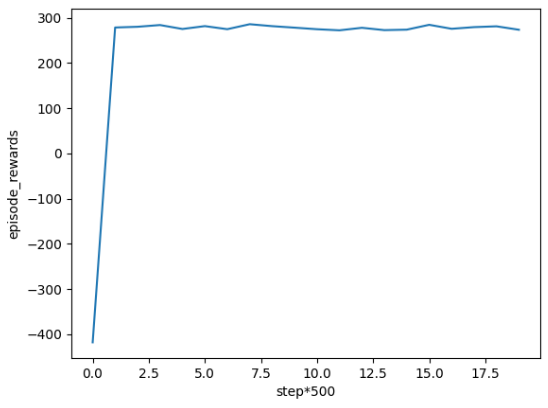

Figure 5.13.: Mean episode reward of my agent vs GreedyOneShotAgent under SCML OneShot

5.3. Conclusion

The result of training of single agent (negotiator) is not bad. For multi-agent concurrent negotiations, it is not easy to implemente. The work of this thesis focuses on two algortims MADDPG and QMIX. The performance and results of QMIX have certain reference value. In the future many work and algorithms are needed to be finished and implemented in the environment MCBE. In the future, one more experiment will be needed: In order to check the generalization ability, let trained RL agent against untrained strategy.

6. Conclusions and Future Work

This thesis has three main contribution. First, two custom training environments (i.e., SBE and MCBE) were developed for training agents in the environment of bilateral negotiation and concurrent bilateral negotiations, respectively. Secondly, it testes baseline RL algorithms (scuh as, DQN, PPO) in SBE and implements two multi-agent drl algorithms (i.e., maddpg and qmix) in MCBE. Finally, the results (e.g., mean episode reward) are presented and evaluated. There are many parts that can be analyzed and improved in the future.

6.1. Other goals

In the SCM league, profit-maximization is the goal of all agents. In a game theoretic analysis we could define many other goals, such as welfare-maximization, pareto optimality. How to achieve these goals with RL-methods based on the developed environments SBE and MCBE is a kez question for future research. The key part is the design of the reward function. For pareto optimal, one idea is the distance between the utility of the current offer and the utility of the pareto optimal offer. The details of the reward function design for other goals can be regarded as the main question in the future.

6.2. Evaluation

The evaluation work of this thesis focuses on two metrics: mean episode reward and score. It is inadequate for evaluating multiple agents. There are many metrics for the evaluation of multi-agent system, not only for the RL multi-agent, but also for the environment. These metrics and methods can speed up the development of available RL multi-agent.

Many characteristics, such as **Complexity of environment**, **Rationality of agent**, **Auton-**

omy, **Reactivity** and **Adaptability**, first were proposed in the paper[P+10].

Complexity of environment: Many parameters were proposed in the paper to describe the complexity:

- **Inaccessibility** It expresses the difficulty in gaining complete access to the resources in its environment.
- **Instability** It expresses the way the environment evolves. In other words, the difficulty in perceiving changes in the environment. The faster and more unpredictably the environment changes, the more complex it is.

The future direction of work may be to find suitable methods to qualify and quantify these metrics.

6.3. Design of reward function

The reward function is an important part of realizing RL-Agent. The future work can focus on the design of more effective reward functions. Inverse reinforcement learning is a powerful method through which an appropriate reward function can be deduced based on expert demonstration. This deduced reward function can be set as a conventional reward function used in traditional RL algorithms. In the future, the effectiveness and implementation process of this method will be explored.

6.4. Complex environment

Currently, it is only possible to train multi-agents in the very simple SCM world. How to effectively train multi-agents in a complex environment is another question for the future. We can do some exploratory work in this area in the future.

6.5. Huge scale high performance learning

Although the experiment in this thesis has achieved negotiators under multiple issues negotiation game, the scale of multiple issues is only 3. The first question in this environment is the large scale action space. The second question is the adaptability of strategies when the DRL negotiators faces different types of opponent negotiators. Future work will focus on these questions.

In addition to improving algorithms, there are many engineering methods that help train the agents in large-scale environments and speed up the learning process of the agents. DeepMind developed a tool called **Reverb**[Cas+21], which is an efficient and easy-to-use data storage and transport system designed for machine learning research. Reverb is primarily used as an experience replay system for distributed reinforcement learning algorithms. In the experiment of this thesis, due to the storage of a lot of experience replay, very large memory usage is a problem. Reverb can be used as an engineering tool to solve this problem. There are some experiments that can be carried out in the future.

Appendices

In order to keep clean, some derivation processes were moved to the appendix. Nevertheless, for completeness, detailed algorithms training pseudo-code were provided after the derivation.

A. Derivation Process

A.1. Proof of Policy Gradient Theorem

The policy is usually modeled with a parameterized function respect to θ, $\pi_\theta(a|s)$. The value of the reward (objective) function depends on this policy. The reward function is defined as:

$$J(\theta) = \sum_{s\in\mathcal{S}} d^\pi(s)V^\pi(s) = \sum_{s\in\mathcal{S}} d^\pi(s) \sum_{a\in\mathcal{A}} \pi_\theta(a \mid s)Q^\pi(s,a) \tag{A.1}$$

We first start with the derivative of the state value function:

$$\begin{aligned}
&\nabla_\theta V^\pi(s) \\
=&\nabla_\theta\Big(\sum_{a\in\mathcal{A}} \pi_\theta(a \mid s)Q^\pi(s,a)\Big) \\
=&\sum_{a\in\mathcal{A}} \left(\nabla_\theta\pi_\theta(a \mid s)Q^\pi(s,a) + \pi_\theta(a \mid s)\nabla_\theta Q^\pi(s,a)\right) \\
=&\sum_{a\in\mathcal{A}} \Big(\nabla_\theta\pi_\theta(a \mid s)Q^\pi(s,a) + \pi_\theta(a \mid s)\nabla_\theta \sum_{s',r} P\left(s',r \mid s,a\right)\left(r + V^\pi\left(s'\right)\right)\Big) \\
=&\sum_{a\in\mathcal{A}} \Big(\nabla_\theta\pi_\theta(a \mid s)Q^\pi(s,a) + \pi_\theta(a \mid s) \sum_{s'} P\left(s' \mid s,a\right)\nabla_\theta V^\pi\left(s'\right)\Big) \\
=&\varphi(s) + \sum_{s'}\sum_{a} \pi_\theta(a \mid s)P\left(s' \mid s,a\right)\nabla_\theta V^\pi\left(s'\right) \qquad\qquad (A.2)\\
=&\varphi(s) + \sum \rho^\pi\left(s \to s',1\right)\nabla_\theta V^\pi\left(s'\right) \\
=&\varphi(s) + \sum \rho^\pi\left(s \to s',1\right)\left[\varphi\left(s'\right) + \sum \rho^\pi\left(s' \to s'',1\right)\nabla_\theta V^\pi\left(s''\right)\right] \\
=&\varphi(s) + \sum_{1} \rho^\pi\left(s \to s',1\right)\varphi\left(s'\right) + \sum_{1\prime} \rho^\pi\left(s \to s'',2\right)\nabla_\theta V^\pi\left(s''\right) \\
=&\text{- .; Repeatedly unrolling the part of}\nabla_\theta V^\pi(.) \\
=&\sum_{x\in\mathcal{S}}\sum_{k=0}^{\infty} \rho^\pi(s \to x,k)\varphi(x)
\end{aligned}$$

By plugging above into the objective function $J(\theta)$, we are getting the following:

$$\begin{aligned}
\nabla_\theta J(\theta) =& \nabla_\theta V^\pi\left(s_0\right) \\
=& \sum_s \sum_{k=0}^{\infty} \rho^\pi\left(s_0 \rightarrow s, k\right) \varphi(s) \\
=& \sum_s \eta(s) \varphi(s) \\
=& \left(\sum_s \eta(s)\right) \sum_s \frac{\eta(s)}{\sum_s \eta(s)} \varphi(s) \\
\propto& \sum_s \frac{\eta(s)}{\sum_s \eta(s)} \varphi(s) \\
=& \sum_s d^\pi(s) \sum_a \nabla_\theta \pi_\theta(a \mid s) Q^\pi(s, a) \\
=& \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \pi_\theta(a \mid s) Q^\pi(s, a) \frac{\nabla_\theta \pi_\theta(a \mid s)}{\pi_\theta(a \mid s)} \\
=& \mathbb{E}_{s \sim d^\pi, a \sim \pi_\theta}\left[Q^\pi(s, a) \nabla_\theta \log \pi_\theta(a \mid s)\right]
\end{aligned} \tag{A.3}$$

Where $d^\pi(s)=\frac{\eta(s)}{\sum_s \eta(s)}$ is stationary distribution. $\sum_s \eta(s)$ is a constant. The derivation process is carried out through the proof in the book [SB18], from which we can understand why the policy gradient theorem is correct.

A.2. Advatage Function

$\hat{A}_t=-V\left(s_t\right)+r_t+\gamma r_{t+1}+\cdots+\gamma^{T-t+1} r_{T-1}+\gamma^{T-t} V\left(s_T\right)$

B. Algorithms

B.1. Single-Agent Reinforcement Learning

B.1.1. DQN

For completeness, the DQN algorithm used in single-agent bilateral negotiation

Algorithmus 1 : Deep Q-learning with experience replay

```
1  Initialize replay buffer $D$ to capacity $N$;
2  Initialize action-value function $Q$ with random weights $\theta$;
3  Initialize target action-value function $\hat{Q}$ with weight $\theta^- = \theta$;
4  for episode = 1, $M$ do
5      Receive state from simulator $s_1 = \{x_1\}$ and preprocessed state $\varphi_1 = \varphi(s_1)$;
6      for $t$ = 1, $T$ do
7          With probability $\omega$ select a random action $a_t$ (first step);
8          otherwise select $a_t = \text{argmax}_a Q(\varphi(s_t), a; \theta)$;
9          Execute action $a_t$ in siumulator and observe reward $r_t$ and new state $x_{t+1}$;
10         Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\varphi_{t+1} = \varphi(s_{t+1})$ ;
11         Store transitions $(\varphi_j, a_j, r_j, \varphi_{t+1})$ in $D$ ;
12         Sample random minibatch of transitions $(\varphi_j, a_j, r_j, \varphi_{j+1})$ from $D$;
13         Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j + 1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\varphi_{j+1}, a'; \theta^-\right) & \text{otherwise} \end{cases}$ ;
14         Perform a gradient descent step on $\left(y_j - Q\left(\varphi_j, a_j; \theta\right)\right)^2$ with respect to the
             network parameters;
15         Every $C$ steps reset $\hat{Q} = Q$;
16     end
17 end
```

B.1.2. PPO

Algorithmus 2 : PPO, Actor-Critic Style [Sch+17]

```
for iteration = 1 to N do
    for actor = 1 to N do
        Run policy $\pi_{\theta_{\text{old}}}$ in evironment for $T$ timesteps ;
        Compute advantage estimates $\hat{A}_1 \ldots \hat{A}_T$ ;
    end
    Optimize surrogate $L$ wrt $\theta$, with $K$ epochs and minibatch size $M \leq NT$ ;
    $\theta_{\text{old}} \leftarrow \theta$ ;
end
```

B.1.3. DDPG

Algorithmus 3 : Deep Deterministic Policy Gradient(DDPG) algorithm[Wen18]

```
1  Randomly initialize critic network $Q(s, a \mid \theta)$ and actor $\mu(s \mid \omega)$ ;
2  Initialize target network $Q'$ and $\mu'$ with weights $\theta' \leftarrow \theta$, $\omega' \leftarrow \omega$ ;
3  Initialize replay buffer *mathcalD*;
4  **for** *episode = 1 to M* **do**
5      Initialize *a* random process $\mathcal{N}$ for action exploration ;
6      Receive initial observation state $s_1$ ;
7      **for** *t = 1 to T* **do**
8          Select action $a_t = \mu(s_t \mid \omega) + \mathcal{N}_t$ according to the current policy and exploration
             noise ;
9          Execute action $a_t$ and observe reward $r_t$ and observe new state $s_{t+1}$ ;
10         Store transition $(s_t, a_t, r_t, s_t t + 1)$ in $\mathcal{D}$ ;
11         Sample a random minibatch of $N$ transitions $(s_i, a_i, r_i, s_{i+1})$ from $\mathcal{N}$ ;
12         Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1} \mid \omega') \mid \theta')$ ;
13         Update critic by minimizing the loss: $L = \frac{1}{N} \sum_i (y_i - Q(s_i, a_i \mid \theta))^2$ ;
14         Update the actor policy using the sampled policy gradient:
               $$\nabla_\omega J \approx \frac{1}{N} \sum_i \nabla_\theta Q(s, a \mid \theta) \Big|_{s=s_i, a=\mu(s_i)} \nabla_\omega \mu(s \mid \omega) \Big|_{s_i} \quad (B.1)$$
             Update the target networks:
               $$\theta' \leftarrow \tau\theta + (1 - \tau)\theta' \qquad \omega' \leftarrow \tau\omega + (1 - \tau)\omega' \quad (B.2)$$
15     **end**
16 **end**
```

B.2. Multi-Agent Reinforcement Learning

B.2.1. MADDPG

For completeness, the MADDPG algorithm used in SCML is provided below.

Algorithmus 4 : Multi-Agent Deep Deterministic Policy Gradient for N agents

Data : State comes from simulator SCML

Result : action sequence, proposal offer or set dynamical range of negotiation issues

```
1  for episode = 1 to M do
2      Initialize a random process $\mathcal{N}$ for action exploration;
3      Receive the intial state from the Simulator;
4      for t = 1 to max-episode-length do
5          for each agent $i$, select action $a_i = \boldsymbol{\mu}_{\theta_i}(o_i) + \mathcal{N}_t$ w.r.t. the current policy and
             exploration.;
6          Execute joint actions $a = (a_1, \ldots, a_N)$ and get the reward $r$ and new state $\mathbf{s}'$;
7          Store $(\mathbf{s}, a, r, \mathbf{s}')$ in replay buffer $\mathcal{D}$;
8          $\mathbf{s} \leftarrow \mathbf{s}'$;
9          for agent $i = 1$ to $N$ do
10             Sample a random minibatch of samples $\mathcal{B}\left(\mathbf{s}^j, a^j, r^j, \mathbf{s}'^j\right)$ from $\mathcal{D}$;
11             Set $y^j = r_i^j + \gamma Q_i^{\mu'}\left(\mathbf{s}'^j, a_1', \ldots, a_N'\right)\Big|_{a_k' = \mu_k'\left(o_k^j\right)}$;
12             Update critic by minimizing the loss $\mathcal{L}(\theta_i) = \frac{1}{B}\sum_j \left(y^j - Q_i^{\boldsymbol{\mu}}\left(\mathbf{s}^j, a_1^j, \ldots, a_N^j\right)\right)^2$;
13             Update actor using the sampled policy gradient:
                   $$\nabla_{\theta_i} J \approx \frac{1}{B}\sum_j \nabla_{\theta_i}\boldsymbol{\mu}_i\left(o_i^j\right)\nabla_{a_i} Q_i^{\mu}\left(\mathbf{s}^j, a_1^j, \ldots, a_i, \ldots, a_N^j\right)\Big|_{a_i = \mu_i\left(o_i^j\right)} \quad \text{(B.3)}$$
14         end
15         Update target network parameters for each agent i: $\theta_i' \leftarrow \tau\theta_i + (1 - \tau)\theta_i'$
16     end
17 end
```

B.2.2. QMIX

Algorithmus 5 : QMIX Algorithm[Ras+20]

```
1  Initialise $\theta$, the parameters of mixing network, agent networks and hypernetwork;
2  Set the learning rate $\alpha$ and replay buffer $\mathcal{D}$;
3  step = 0, $\theta^{-} = \theta$;
4  for step = 0 to $step_{max}$ do
5      $s_0$ = initial state while $s_t \neq$ terminal and $t <$ episode limit do
6          for each agent a do
7              $\tau_t^a = \tau_{t-1}^a \cup \{(o_t, u_{t-1})\}$ ;
8              $\varepsilon =$ epsilon-schedule ( step ) ;
9              $u_t^a = \begin{cases} \text{argmax}_{u_t^a} Q\left(\tau_t^a, u_t^a\right) & \text{with probability } 1-\varepsilon \\ \text{randint}(1,|U|) & \text{with probability } \varepsilon \end{cases}$ ;
10         end
11         Get reward $r_t$ and next state $s_{t+1}$ ;
12         $\mathcal{D} = \mathcal{D} \cup \{(s_t, \mathbf{u}_t, r_t, s_{t+1})\}$ ;
13         $t = t+1, step = step+1$;
14     end
15     if $|\mathcal{D}|>$ batch-size then
16         b $\leftarrow$ random batch of episodes from $\mathcal{D}$ ;
17         for each timestep t in each episode in batch b do
18             $Q_{tot} =$ Mixing-network $\left(Q_1\left(\tau_t^1, u_t^1\right), \ldots, Q_n\left(\tau_t^n, u_t^n\right);\ \text{Hypernetwork}\ (s_t; \theta)\right)$ ;
19             Calculate target $Q_{tot}$ using Mixing-network with Hypernetwork $(s_t; \theta^{-}))$ ;
20         end
21         $\Delta Q_{tot} = y^{tot} - Q_{tot}$ ;
22         $\Delta\theta = \nabla_{\theta}(\Delta Q_{tot})^2$ ;
23         $\theta = \theta - \alpha\Delta\theta$ ;
24     end
25     if update-interval steps have passed then
26         $\theta^{-} = \theta$;
27     end
28 end
```

Bibliography

[Ayd+17] Reyhan Aydoğan et al. “Alternating Offers Protocols for Multilateral Negotiation.” In: *Modern Approaches to Agent-based Complex Automated Negotiation*. Ed. by Katsuhide Fujita et al. Cham: Springer International Publishing, 2017, pp. 153–167. URL: https://doi.org/10.1007/978-3-319-51563-2_10.

[Baa+12] Tim Baarslag et al. “The First Automated Negotiating Agents Competition (ANAC 2010).” In: vol. 383. Jan. 2012, pp. 113–135.

[Baa+14] Tim Baarslag et al. “Decoupling Negotiating Agents to Explore the Space of Negotiation Strategies.” In: vol. 535. Jan. 2014, pp. 61–83.

[Bag+20] Pallavi Bagga et al. *A Deep Reinforcement Learning Approach to Concurrent Bilateral Negotiation*. 2020. arXiv: 2001.11785 [cs.MA].

[Bak+19] Jasper Bakker et al. “RLBOA: A Modular Reinforcement Learning Framework for Autonomous Negotiating Agents.” In: *AAMAS*. 2019.

[BBB12] Ezzeddine Benaissa, Abdellatif BenAbdelhafid, and Mounir Benaissa. “An Agent-based framework for cooperation in Supply Chain.” In: *CoRR* abs/1210.3375 (2012). arXiv: 1210.3375. URL: http://arxiv.org/abs/1210.3375.

[Bel54] Richard Ernest Bellman. *The Theory of Dynamic Programming*. Santa Monica, CA: RAND Corporation, 1954.

[Ber+19] Christopher Berner et al. “Dota 2 with Large Scale Deep Reinforcement Learning.” In: *CoRR* abs/1912.06680 (2019). arXiv: 1912.06680. URL: http://arxiv.org/abs/1912.06680.

[BFF09] Emma Brunette, Rory Flemmer, and Claire Flemmer. “A review of artificial intelligence.” In: Feb. 2009, pp. 385–392.

[BH00] I.A Basheer and M Hajmeer. “Artificial neural networks: fundamentals, computing, design, and application.” In: *Journal of Microbiological Methods* 43.1 (2000). Neural Computting in Micrbiology, pp. 3–31. URL: https://www.sciencedirect.com/science/article/pii/S0167701200002013.

[Bha+17] Parth Bhavsar et al. “Machine Learning in Transportation Data Analytics.” In: Dec. 2017, pp. 283–307.

[BHJ13] Tim Baarslag, Koen Hindriks, and Catholijn Jonker. “A Tit for Tat Negotiation Strategy for Real-Time Bilateral Negotiations.” In: vol. 435. Jan. 2013, pp. 229–233.

[BLO19] ANDRIY BLOKHIN. *What is the Utility Function and How is it Calculated?* 2019. URL: `https://www.investopedia.com/ask/answers/072915/what-utility-function-and-how-it-calculated.asp`.

[BM07] Constanta Nicoleta BODEA and Radu Ioan MOGOS. “An Electronic Market Space Architecture Based On Intelligent Agents And Data Mining Technologies.” In: *Informatica Economica* 0.4 (2007), pp. 115–118. URL: `https://ideas.repec.org/a/aes/infoec/vxiy2007i4p115-118.html`.

[Bou96] Craig Boutilier. “Planning, Learning and Coordination in Multiagent Decision Processes.” In: *TARK*. 1996.

[BR10] Nicole Bäuerle and Ulrich Rieder. “Markov Decision Processes.” In: *Jahresbericht der Deutschen Mathematiker-Vereinigung* 112 (Dec. 2010), pp. 217–243.

[Bro+16] Greg Brockman et al. *OpenAI Gym*. 2016. arXiv: `1606.01540 [cs.LG]`.

[Car+21] Mathilde Caron et al. *Emerging Properties in Self-Supervised Vision Transformers*. 2021. arXiv: `2104.14294 [cs.CV]`.

[Cas+21] Albin Cassirer et al. *Reverb: A Framework For Experience Replay*. 2021. arXiv: `2102.04736 [cs.LG]`.

[Cha20] Ho-Chun Herbert Chang. *Multi-Issue Bargaining With Deep Reinforcement Learning*. 2020. arXiv: `2002.07788 [cs.MA]`.

[Che+99] Ye Chen et al. “A negotiation-based Multi-agent System for Supply Chain Management.” In: 1999.

[Dha+17] Prafulla Dhariwal et al. *OpenAI Baselines*. `https://github.com/openai/baselines`. 2017.

[Dul+16] Gabriel Dulac-Arnold et al. *Deep Reinforcement Learning in Large Discrete Action Spaces*. 2016. arXiv: `1512.07679 [cs.AI]`.

[Fan+20] Xiaohan Fang et al. “Multi-Agent Reinforcement Learning Approach for Residential Microgrid Energy Scheduling.” In: *Energies* 13.1 (2020). URL: `https://www.mdpi.com/1996-1073/13/1/123`.

[Fet19] Fetch.ai. *Introducing Autonomous Economic Agents (AEAs).* 2019. URL: https://medium.com/fetch-ai/introducing-autonomous-economic-agents-aeas-a6290c2092ac.

[FSJ98] Peyman Faratin, Carles Sierra, and Nick R. Jennings. "Negotiation decision functions for autonomous agents." In: *Robotics and Autonomous Systems* 24.3 (1998). Multi-Agent Rationality, pp. 159–182. URL: https://www.sciencedirect.com/science/article/pii/S0921889098000293.

[FWJ02] S. Shaheen Fatima, Michael Wooldridge, and Nicholas R. Jennings. "Optimal Negotiation Strategies for Agents with Incomplete Information." In: *Intelligent Agents VIII.* Ed. by John-Jules Ch. Meyer and Milind Tambe. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 377–392.

[Had15] Emmanuel Hadoux. "Markovian sequential decision-making in non-stationary environments : application to argumentative debates." Theses. Université Pierre et Marie Curie - Paris VI, Nov. 2015. URL: https://tel.archives-ouvertes.fr/tel-01259918.

[Has95] Mohamad Hassoun. *Fundamentals of Artificial Neural Networks.* 1995. URL: https://mitpress.mit.edu/books/fundamentals-artificial-neural-networks.

[Hes+17] Todd Hester et al. "Learning from Demonstrations for Real World Reinforcement Learning." In: *CoRR* abs/1704.03732 (2017). arXiv: 1704.03732. URL: http://arxiv.org/abs/1704.03732.

[Hil+18] Ashley Hill et al. *Stable Baselines.* https://github.com/hill-a/stable-baselines. 2018.

[Hu+18] Yujing Hu et al. "Reinforcement Learning to Rank in E-Commerce Search Engine: Formalization, Analysis, and Application." In: *CoRR* abs/1803.00710 (2018). arXiv: 1803.00710. URL: http://arxiv.org/abs/1803.00710.

[Kre89] David M. Kreps. "Nash Equilibrium." In: *Game Theory.* Ed. by John Eatwell, Murray Milgate, and Peter Newman. London: Palgrave Macmillan UK, 1989, pp. 167–177. URL: https://doi.org/10.1007/978-1-349-20181-5_19.

[KT15] Hara K. and Ito T. "Effects of GA Based Mediation Protocol for Utilities that Change Over Time." In: *In: Fujita K., Ito T., Zhang M., Robu V. (eds) Next Frontier in Agent-Based Complex Automated Negotiation. Studies in Computational Intelligence, vol* 596. (2015).

[Lil+15] Timothy Lillicrap et al. "Continuous control with deep reinforcement learning." In: *CoRR* (Sept. 2015).

[Lin+14] Raz Lin et al. "Genius: An Integrated Environment for Supporting the Design of Generic Automated Negotiators." In: *Computational Intelligence* 30 (Feb. 2014), pp. 48–70.

[LKK04] Keonsoo Lee, Wonil Kim, and Minkoo Kim. "Supply Chain Management using Multi-agent System." In: Sept. 2004, pp. 215–225.

[Low+17] Ryan Lowe et al. "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments." In: *CoRR* abs/1706.02275 (2017). arXiv: `1706.02275`. URL: `http://arxiv.org/abs/1706.02275`.

[Mar+15] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.* Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[Moh+19] Yasser Mohammad et al. "Supply Chain Management World." In: Oct. 2019, pp. 153–169.

[Mor+17] Philipp Moritz et al. "Ray: A Distributed Framework for Emerging AI Applications." In: *CoRR* abs/1712.05889 (2017). arXiv: `1712.05889`. URL: `http://arxiv.org/abs/1712.05889`.

[NHR99] A. Ng, D. Harada, and Stuart J. Russell. "Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping." In: *ICML*. 1999.

[NMR44] John von Neumann, Oskar Morgenstern, and Ariel Rubinstein. *Theory of Games and Economic Behavior (60th Anniversary Commemorative Edition).* Princeton University Press, 1944. URL: `http://www.jstor.org/stable/j.ctt1r2gkx`.

[P+10] Di Bitonto P. et al. *An Evaluation Method for Multi-Agent Systems.* Springer Berlin Heidelberg, 2010. URL: `https://link.springer.com/chapter/10.1007/978-3-642-13480-7_5#citeas`.

[Pas+19] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In: *Advances in Neural Information Processing Systems.* Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: `https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf`.

[Pat+17] Deepak Pathak et al. "Curiosity-driven Exploration by Self-supervised Prediction." In: *ICML*. 2017.

[PB11] Darja Plinere and Arkādijs Borisovs. *A Negotiation-Based Multi-Agent System for Supply Chain Management*. RTU scientific journal, 2011. URL: https://link.springer.com/chapter/10.1007/978-3-642-13480-7_5#citeas.

[Pet91] William H. Peterson. *Boulwarism: Ideas Have Consequences*. Apr. 1991. URL: https://fee.org/articles/boulwarism-ideas-have-consequences/.

[PKB20] Sindhu Padakandla, Prabuchandran K. J., and Shalabh Bhatnagar. "Reinforcement learning algorithm for non-stationary environments." In: *Applied Intelligence* 50.11 (June 2020), pp. 3590–3606. URL: http://dx.doi.org/10.1007/s10489-020-01758-5.

[Pnd19] Kritika Pndey. *How AI is Revolutionizing Global Logistics and Supply Chain Management*. 2019. URL: https://readwrite.com/2019/04/15/how-ai-is-revolutionizing-global-logistics-and-supply-chain-management/.

[Rai82] H. Raiffa. *The Art and Science of Negotiation*. Harvard University Press, 1982. URL: https://books.google.de/books?id=y-4T88h3ntAC.

[Ras+18] Tabish Rashid et al. "QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning." In: (Mar. 2018).

[Ras+20] Tabish Rashid et al. "Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning." In: *CoRR* abs/2003.08839 (2020). arXiv: 2003.08839. URL: https://arxiv.org/abs/2003.08839.

[Ros19] Don Ross. "Game Theory." In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta. Winter 2019. Metaphysics Research Lab, Stanford University, 2019.

[Rub82] Ariel Rubinstein. "Perfect Equilibrium in A Bargaining Model." In: *Econometrica* 50 (Feb. 1982), pp. 97–109.

[Sai+19] N.J. Sairamya et al. "Chapter 12 - Hybrid Approach for Classification of Electroencephalographic Signals Using Time–Frequency Images With Wavelets and Texture Features." In: *Intelligent Data Analysis for Biomedical Applications*. Ed. by D. Jude Hemanth, Deepak Gupta, and Valentina Emilia Balas. Intelligent Data-Centric Systems. Academic Press, 2019, pp. 253–273. URL: https://www.sciencedirect.com/science/article/pii/B9780128155530000136.

[Sam59] A. L. Samuel. "Some Studies in Machine Learning Using the Game of Checkers." In: *IBM Journal of Research and Development* 3.3 (1959), pp. 210–229.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction, Second Edition*. MIT Press Cambridge MA, 2018. URL: https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf.

[Sch+17] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].

[Sch04] Andrea Schneider. "Aspirations in Negotiation." In: (Jan. 2004).

[SE18] Ahmed Shokry and Antonio Espuña. "The Ordinary Kriging in Multivariate Dynamic Modelling and Multistep-Ahead Prediction." In: *28th European Symposium on Computer Aided Process Engineering*. Ed. by Anton Friedl et al. Vol. 43. Computer Aided Chemical Engineering. Elsevier, 2018, pp. 265–270. URL: https://www.sciencedirect.com/science/article/pii/B9780444642356500474.

[Sil+17] David Silver et al. "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm." In: *CoRR* abs/1712.01815 (2017). arXiv: 1712.01815. URL: http://arxiv.org/abs/1712.01815.

[SUM20] SUMAN. *Is machine learning required for deep learning?* 2020. URL: https://ai.stackexchange.com/questions/15859/is-machine-learning-required-for-deep-learning.

[Sun+17] Peter Sunehag et al. "Value-Decomposition Networks For Cooperative Multi-Agent Learning." In: *CoRR* abs/1706.05296 (2017). arXiv: 1706.05296. URL: http://arxiv.org/abs/1706.05296.

[Wan+16] Ziyu Wang et al. "Sample Efficient Actor-Critic with Experience Replay." In: *CoRR* abs/1611.01224 (2016). arXiv: 1611.01224. URL: http://arxiv.org/abs/1611.01224.

[WC03] Steven Walczak and Narciso Cerpa. "Artificial Neural Networks." In: *Encyclopedia of Physical Science and Technology (Third Edition)*. Ed. by Robert A. Meyers. Third Edition. New York: Academic Press, 2003, pp. 631–645. URL: https://www.sciencedirect.com/science/article/pii/B0122274105008371.

[Wei01] Hans Weigand. "The Communicative Logic of Negotiation in B2B e-Commerce." In: vol. 74. Jan. 2001, pp. 523–236.

[Wen18] Lilian Weng. *Policy Gradient Algorithms*. 2018. URL: https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html.

[Whi95] Jerry R. Whinston Michael D.; Green. *Equilibrium and its Basic Welfare Properties, Microeconomic Theory*. Oxford University Press, 1995.

[Wie03] Eric Wiewiora. "Potential-Based Shaping and Q-Value Initialization are Equivalent." In: (Oct. 2003).

[Wie10] Eric Wiewiora. "Reward Shaping." In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 863–865. URL: https://doi.org/10.1007/978-0-387-30164-8_731.

[Wil+11] Colin Williams et al. "Using Gaussian Processes to Optimise Concession in Complex Negotiations against Unknown Opponents." In: *IJCAI International Joint Conference on Artificial Intelligence* (Jan. 2011).

[Wil+12] Colin Williams et al. "Negotiating Concurrently with Unknown Opponents in Complex, Real-Time Domains." In: May 2012.

[Y+21] Mohammed Y et al. "Supply Chain Management League (OneShot)." In: Mar. 2021. URL: http://www.yasserm.com/scml/scml2021oneshot.pdf.

[Zho+17] L. Zhou et al. "Multiagent Reinforcement Learning With Sparse Interactions by Negotiation and Knowledge Transfer." In: *IEEE Transactions on Cybernetics* 47.5 (May 2017), pp. 1238–1250.

List of Tables

List of Figures

List of Theorems

Listings

Glossary

ACER ACER 58

ANAC The International Automated Negotiating Agents Competition (ANAC) is an annual event, held in conjunction with the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), or the International Joint Conference on Artificial Intelligence (IJCAI). The ANAC competition brings together researchers from the negotiation community and provides unique benchmarks for evaluating practical negotiation strategies in multi-issue domains. The competitions have spawned novel research in AI in the field of autonomous agent design which are available to the wider research community. 25, 26

ANEGMA ANEGMA 38, 92

ANN Artificial Neural Network ix, 15

BOA the bidding strategy, the opponent model, and the acceptance strategy 37

CNS Concurrent negotiation strategy. x, 36

DDPG Deep Deterministic Policy Gradient 22, 58

DPG Deterministic Policy Gradient 22, 47

DQN Deep Q-Value Network 21, 47, 57, 58

DRL Deep reinforcement learning. x, 3, 21, 42, 55, 58, 74

GreedyOneShotAgent A greedy agent based on OneShotAgent 51, 93

IndDecentralizingAgent Independent Centralizing Agent, implemented in standard scml 63, 93

MADDPG Multi Agent Deep Deterministic Policy Gradient iv, v, xi, 18, 22, 23, 53, 61–63, 71, 82, 93

MADRL Multi-Agent Deep reinforcement learning. 18, 48, 49

MARL Multi-Agent Reinforcement Learning. 18, 49

MAS Multi-Agent system 1

MCBE Multi-agent concurrent bilateral negotiation environment. xi, 42, 48–51, 53, 54, 59, 62, 64, 66, 71, 72, 93

MDP Markov decision process. 7, 8, 19, 40

MDPs Markov decision process. 8

MyOneShotBasedAgent My Deep Reinforcement Learning Agent in SCM-ONESHOT 51, 93

NegMAS NEGotiation MultiAgent System x, 1–4, 24–26, 42, 43, 45, 48, 55, 92

NegoSI negotiation-based MARL with sparse interactions (NegoSI)

OpenAI Gym OpenAI Gym is a toolkit for reinforcement learning research. It includes a growing collection of benchmark problems that expose a common interface, and a website where people can share their results and compare the performance of algorithms. x, 4, 29, 31, 42, 44, 45, 48, 49, 95

PG Policy Gradient ix, 20–22, 47

PPO Proximal Policy Optimization 21, 47, 57, 58

PyTorch Python machine learning framework, developed by... x, 29

QMIX Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning iv, v, xi, 23, 53, 65–67, 69–71, 83, 93

Ray Ray provides a simple, universal API for building distributed applications. x, 32

RL Reinforcement learning. 3, 8, 31, 33, 39, 40, 42, 43, 45–47, 53, 72

RLBOA RLBOA 37, 46

SAOM Stacked Alternating Offers Protocol, namely in SCML also as Stacked Alternating Offers Mechanism. ix, 11, 24, 25, 42, 44, 56

SARSA State-action-reward-state-action 47

SBE Single-agent bilateral negotiation environment. xi, 42–46, 48, 50, 51, 53, 55–59, 72, 93

SCM Supply Chain Management 1, 2, 24–28, 40, 48, 51, 73

SCML Supply Chain Management League one of ANAC 2020 and 2021 leagues @ IJCAI 2020 and 2021. x, 1–4, 25, 26, 42, 48, 49, 51, 61, 63, 82, 92, 93

SCML-OneShot OneShot World in SCML. 26–28, 48, 51, 65, 92

TD Temporal Error. 20

VDN Value Decomposition Networks 23