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. habil. Jürgen Beyerer
Betreuende Mitarbeiter: Dr.-Ing. Tim Zander
Prof. Dr.-Ing. Yasser Mohammad(Extern)

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)

Abstract

Consider an agent that can cooperate with others and autonomously negotiate, to reach an agreement. These agents could achieve great results, such as presenting the profitability of 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

Falls die Abschlussarbeit auf Deutsch geschrieben wird, genügt die deutsche Kurzfassung.

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

$\alpha, \ldots, \omega$	Skalare
$a, \ldots, z$	Skalar, Vektor, Funktionssymbol (oder Realisierung einer Zufallsvariablen)
$\boldsymbol{a}, \ldots, \boldsymbol{z}$	Zufallsvariable (skalar bzw. vektoriell)
$\hat{\boldsymbol{a}}, \ldots, \hat{\boldsymbol{z}}$	Schätzer für jeweilige Variable als Zufallsgröße
$\hat{a}, \ldots, \hat{z}$	Realisierter Schätzer für jeweilige Variable
$A, \ldots, Z$	Matrix
$\boldsymbol{A}, \ldots, \boldsymbol{Z}$	Matrix als Zufallsgröße
$\mathcal{A}, \ldots, \mathcal{Z}$	Menge
$\mathfrak{A}, \ldots, \mathfrak{Z}$	Mengensystem

Special identifier

s	State
a	action
$\mathcal{S}$	set of nonterminal states
$\mathcal{A}(s)$	set of actions possible in state s
$\mathcal{R}$	set of possible rewards
t	discrete time step
T	final time step of an episode
S_t	state at t
A_t	action at t
R_t	reward at t
G_t	return (cumulative discounted reward) following t
π	policy, decision-making rule
$\pi(s)$	action taken in state s under `deterministic` policy π
$\pi(a\|s)$	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
$V_(s)$	esimate (a random variable) of $v_\pi(s)$ or $v_*(s)$
$Q_t(s,a)$	

General quantities

$\mathbb{C}$	Menge der komplexen Zahlen
$\mathbb{H}$	Poincaré Halbebene
$\mathbb{N}$	Menge der natürliche Zahlen (ohne Null)
$\mathbb{N}_0$	Menge der natürliche Zahlen mit Null
$\mathbb{Q}$	Menge der rationalen Zahlen
$\mathbb{Q}^{>0}$, $\mathbb{Q}^{<0}$	Menge der positiven bzw. negative rationalen Zahlen
$\mathbb{R}$	Menge der reellen Zahlen
$\mathbb{R}^{>0}$, $\mathbb{R}^{<0}$	Menge der positiven bzw. negative reellen Zahlen
$\mathbb{Z}$	Menge der ganzen Zahlen

Special symbols

$\mathfrak{N}(\mu, \sigma^2)$	Normalverteilung mit Erwartungswert μ und Varianz σ
$\mathfrak{F}_{r,s}$	Fisher-Verteilung mir r Zähler- und s Nennerfreiheitsgraden
$\mathfrak{t}_s$	Student-t-Verteilung mit s Freiheitsgraden
δ_ξ	Ein-Punkt-Maß an der Stelle ξ
χ^2_s	χ^2-Verteilung mit s Freiheitsgraden

Contents

1. Introduction

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

In economic, 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 agents 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 could be considered as agents in the Multi-agent systems (MAS). Multi-agent System(MAS) are suitable the domains that involve interactions between different people or organizations with different (possibly conflicting) goals and proprietary information, there are many approaches are proposed in order to solve the problem in the supply chain mangement system, such as negotiation-based Multi-agent System[].

The Supply Chain Management (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 [Moh+19]. It is characterized by profit-maximizing agents that inhabit a complex, dynamic, negotiation environment[]. The

introduction of all definitions and the work by `Yasser Mohammad` will be shown in Background 2. Because all subsequent work in this thesis will be carried out around this platform. Before discussing the details of the work, we need to talk about the specific motivation of 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 []. 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 []. Autonomous negotiation agent is meaningful in many realistic environment, such as all mentioned economic value in industries. The development of current machine learning algorithms and increased hardware resource make it possible, model the realistic environment to evaluate the problem with computer system. According to the modeled realistic environment, it will be easier to find more possible solutions with the help of machine learning technology.

In this work, some modeled negotiation environments, such as single agent environment (bilateral negotiation), are developed for analyzing whether deep reinforcement learning can be used to let agent learns some strategies autonomously. In contrast to single agent environment, in the supply chain environment, there are many agents with the same goal. After analyzing the simple environment, the situation is needed to be explored whether multi-agent deep reinforcement learning 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 environment, and all multi-agent reinforcement learning methods have great practical significance here.

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

Before testing deep reinforcement learning in a multi-agent environment, single-agent bilateral negotiation is a better simple test environment. The result will help to analyze the effect of algorithm in autonomous negotiation game. It has a certain practical significance.

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

This is the key question of this thesis. By comparing with other benchmark negotiators or agents, the results can be evaluated.

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

In the evaluation process, it 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 this work are listed and their content briefly presented.

Chapter 2: Background: This chapter contains basic knowledge and concepts that are necessary to understand the thesis. Firstly, some concepts from game theory are listed. These concepts are often discussed and used in autonomous negotiation. Secondly, utility function, some negotiation mechanisms are described in the section on autonomous negotiation. In addition, the basics and the historical development of artificial intelligence are presented. The focus of this chapter is on reinforcement learning.

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: Negotiation Strategies for Autonomous Negotiation, Reinforcement Learning used in Autonomous Negotiations and Challenges in Deep Reinforcement Learning. In the section `Challenges in Deep Reinforcement Learning`, some related algorithms except RL. will be discussed.

Chapter 4: Analysis The task of this thesis is studied in detail in the Analysis chapter. The main content of this chapter is to show how to use the ideas of OpenAI Gym and related reference materials to develop the two custom training environments. The second part is to analyze the characteristics of the algorithm in advance.

Chapter 5: Methods and Experiments In this chapter, The configuration and parameters of the experimental environment are introduced in detail. The specific hyperparameters and training process of the algorithm will also be explained. At the end, the experimental results are displayed and evaluated, and compared with other algorithms.

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

2. Background

2.1. Game Theory

2.1.1. Nash Equilibrium

The concept of a Nash Equilibrium plays a central role in game theory. The definition in simple setting of a finite player is described as follow with mathematical form. Form indexes K agents as $k = i, \ldots, K$. There are total N_k pure strategies. From N_k agent k choose a strategy called s_k. S_k denotes the set of strategies, and s_k as 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 formed 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 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 terms of words description, the definition of Nash Equilibrium is that if other agents do not change its strategy, then no single agent can obtain higher utility.

2.1.2. Pareto Efficient

Pareto Efficient is also named as Pareto Optimal which is 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 economy 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 Efficient 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 completely information static game.

2.1.3. Markov Games

These Games based on the Markov Decision Process are called Markov Game. The term **Markov Decision Process** has been proposed by Bellman in 1954[Bel54]. It can described as a system that can be controlled by a sequential decisions.

Finite Markov Decision Processes Markov Decision Processes with finite time horizon will be considered in this section. The model mainly includes the state space, action space, random transition law and reward function of the system. Hence, a non-stationary Markov Decision Model 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 th e actin 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.

- $r_n : D \rightarrow \mathbb{R}$ 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.
- $g_N : E \rightarrow \mathbb{R}$ is a measurable mapping. $g_N(s)$ gives the discounted terminal reward of the system at the time N if the state is x.

Next step the definition of strategy or policy is neccessary to be introduced. A policy is a mapping $\pi : S \rightarrow A$, where $\pi(s)$ means 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. Hence, how to find the optimal strategy and calculate the accumulated expected reward are important questions in the research field of reinforcement learning. All details will be further introduced in section value functions 2.5.2.

Multi-Agent Markov Decision Processes Based on communication ability of agents, Multi-Agent extension of Markov Decision Processes(MDPs) can be called partially or complete 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 individuale agents, respectively. For each agent, a stochastic strategy $\boldsymbol{\pi}_{\theta_i}$ will be used to choose action, the mapping is $\boldsymbol{\pi}_{\theta_i} : \boldsymbol{O}_i \times \mathcal{A}_i \mapsto [0,1]$. For the multi-agent MDP, the trainstion function is defined as following mapping functions $\mathcal{T} : \mathcal{S} \times \mathcal{A}_1 \times \ldots \times \mathcal{A}_N \mapsto \mathcal{S}$, which produces the next state. $r_i(s, a)$ 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 obsevation correlated with the state $o_i : S \mapsto O_i$[Low+17]. There are many cases for Multi-Agent MDP. Following three items are the typical cases that are introduced by `Boutilier` in paper[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. There has been extensive research in the area of automated negotiating agents.

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].

Through the negotiation agents, many problems that arise in real or simulated domain can be solved. In industrial domains and in commerical domains, the Supply Chain Management System (SCMS) functionality is implemented through agent-based negotiation environment, in which contracts can be singed through negotiation between agents. Many papers describe ongoing effort in developing a Multi-agent System (MAS) for supply chain management, such as work in paper [LKK04].

In game domains, bilateral negotiation in [GENIUS]

2.2.1. Utility Function

Utility Function is an important concept in economics. It measures preferences for a set of goods and services. Utility represents the satisfaction that consumers obtain when choosing and consuming products or services[BLO19]. Utility Function can measure either single offer or set of offers.

Utility 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 briefly listed and introduced below:

- **linear utility function:** $u\left(x_1, x_2, \ldots, x_m\right) = w_1x_1 + w_2x_2 + \ldots w_mx_m$ 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.

It is a important point for designing a new Agent in autonomous negotiation environments. For heuristic agents utility function is a keypoint to measure preferences. For reinforment learning agents utiliy function conducts the behavior of learnable agents, it can be used as a part of reward function, significantly affect the design and evaluation of RL-Agent.

2.2.2. Basic Notation in Negotiation Mechanism

Before going into the specific negotiation mechanism. We need to understand some basic notations defined in the paper [Ayd+17]. When discussing specific negotiation mechanisms, the following definitions will be used.

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

Definition 2.4 (Turn taking) *Alternating Offer Protocols assign turns to the negoiating 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. Two agents in the mechanism which has an infinite time horizon and have to reach an agreement. In a turn, one agent propose an offer, the other need to decide either to accept it, or to reject it and continue the bargaining[Rub82]. The offer is about how to divide the pie of size 1.

After the two agents have played indefinitely, they may get the corresponding Nash Equilibrium solution.

2.2.4. Stacked Alternating Offers Mechanism(SAOM)

In the Alternating Offers Protocol, one of the agent start to proposal 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 offer the other agent can either end the negotiation or give a count offer.

SAOM is also named as stacked alternating offers protocol. 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 condisiton is met, if an agreement is reached or the deadline is reached. When an agreement is reached, all agents need to accept the offer. If at the deadline no agreement is reached, 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 items that sometimes are erroneously used interchangeably, namely artificial intelligence, machine learning, and neural networks. 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.

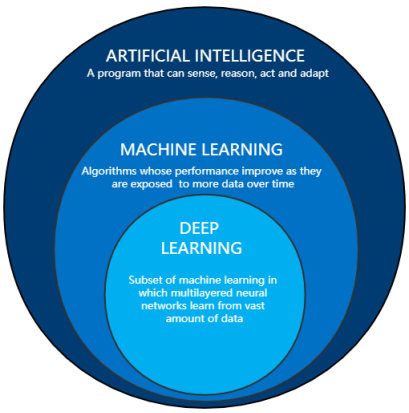

Figure 2.1.: Sub-areas of artificial intelligence Source: Own illustration based on[SUM20]

2.3.1.1. Artificial Intelligence

Artificial intelligence (also called machine intelligence) can be understood through 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. Therefore, AI means that the machine imitates human behavior in some way.

2.3.1.2. Machine Learning

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

2.3.1.3. Deep Learning

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. It has networks that can learn from unstructured or unlabeled data without supervision.

2.3.2. Methods

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 There are not the desired outputs in the training results. Clustering is an example. It is impossible to know what is and what is not good learning.

Semi-supervised Learning A few desired outputs 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. It is the most ambitious method of learning in AI.

2.3.3. Application Field

Artificial intelligence (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. Autonomous Agent is an important concept of AI. Artificial intelligence is a big concept with a wide range of applications. It provides support for many scenarios, such as eCommerce, Logistics and Supply Chain and as research tools for computer science.

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, artificial intelligence 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 Artificial intelligence 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 Artificial intelligence 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 techonology that can be used in this field.

2.3.3.3. Tools for computer science

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

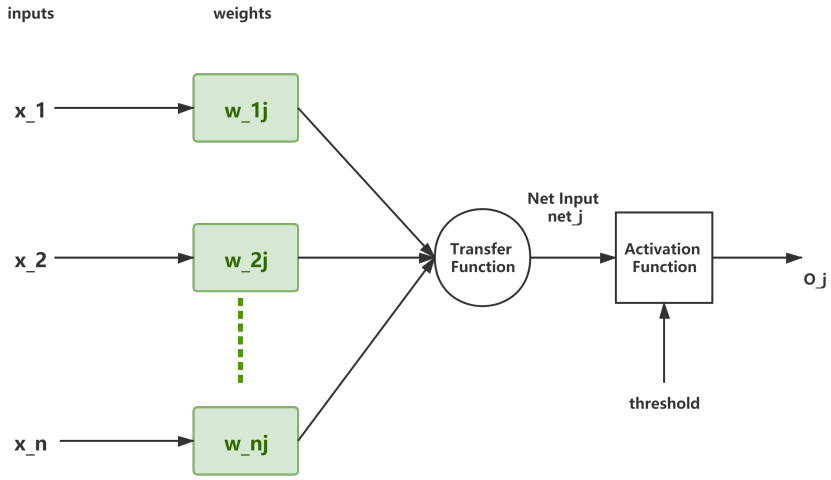

Figure 2.2.: Aritifical Neuron

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 a 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 form is shown as follows:

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

There are many diffrent 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 the hidden layers extend the ability of the neural networks to solve nonlinear classification problems[BHo0]. Figure 2.3 diagrams a multi-layers neural network.

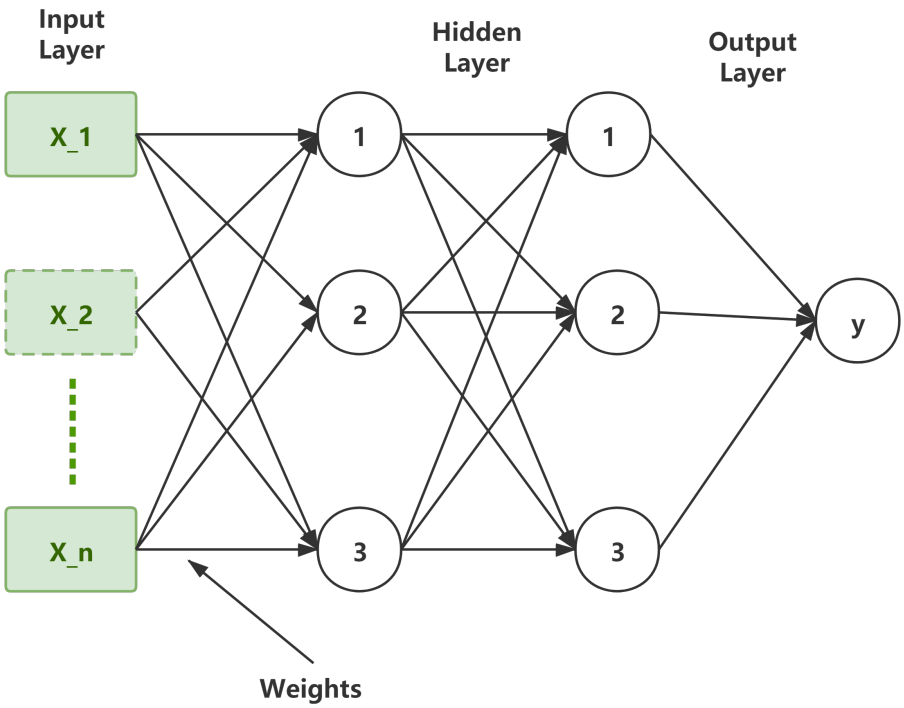

Figure 2.3.: Aritifical Multi-Layers Neural Network, x* are inputs. Source: Own illustration based on[Sai+19].

2.4.3. Recurrent Neural Networks (RNNS)

In a recurrent neural netowrks, the output of some neurons are fed back to the same neurons or to neurons in preceding layers[BHo0]. 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 diagrams the

recurrent neural networks.

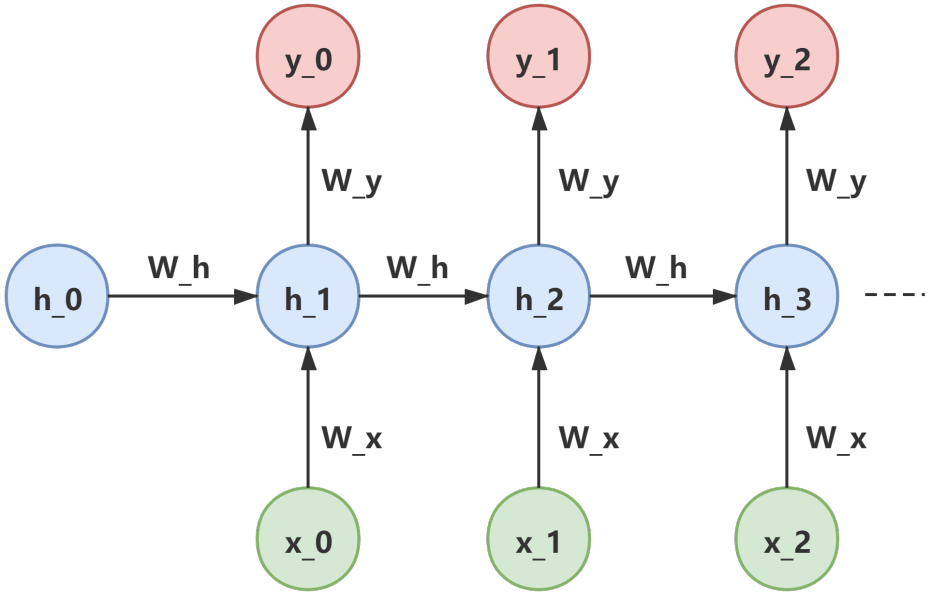

Figure 2.4.: Recurrent Neural Network where h* are outputs of hidden layers, x* are inputs, y* are outputs of network, w* are the weights corresponding to the layers.

2.4.4. Backpropagation

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. The learner and decision-maker is called the 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.

Above process is a typical single-agent interaction process. Single agent reinforcement learning

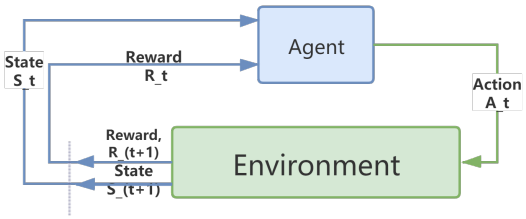

Figure 2.5.: The agent–environment interaction in reinforcement learning, Source: Own illustration based on[SB18].

algorithms are based on this process. By extending this interactive process, multi-agent interactive process can be intuitively diagrammed In 2.6. These learning cases are called Multi-Agent-Reinforcement Learning MARL. In the figure 2.6 agent is splited as two parts perceptor and learner. Perceptors observe the environment and send the state to learners. Learners learn strategies based on the states, rewards and actions. There are many methods proposed in these years. Some MADRL methods will be discussed in following sections. According to the design requirements of the training environment, the structure of the interaction process is flexible.

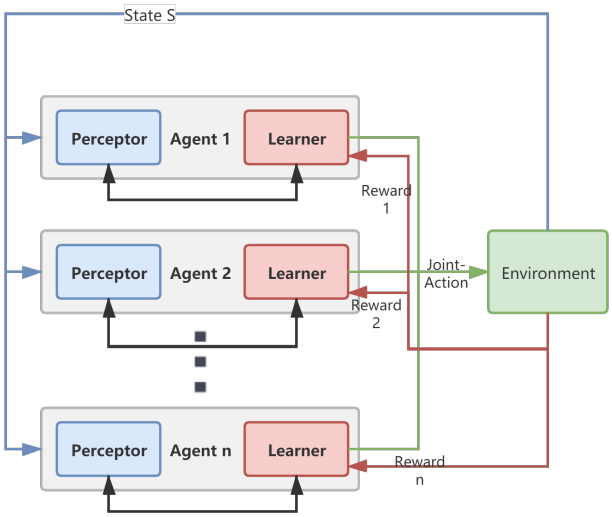

Figure 2.6.: The multi-agent–environment interaction in reinforcement learning, diagrammed In [Fan+20].

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 discount accumulated expected 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 consists fixed state or both state and action, 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:

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

Compared with a 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, action. Compared with state value function 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.

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

2.5.3. Bellman Functions

In summary, Bellman Functions decomposes the value function into two parts, the immediate reward plus the discounted future values. Equation 2.5 show how to recursively the Bellman equation is defined for the state-value function:

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

$\pi(a \mid s)$ 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 tells us how to find

recursively the value of a state-action pair following a policy π.

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

2.5.4. Q-Learning

To maximize the total cumulative reward in the long sequence is the goal of 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-actin value function is shown below:

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

TD is the abbreviation of Temporal Error. $R(s,a) + \gamma \max_{a'} Q\left(s',a'\right)$ denotes the TD target. Current state-action value is fromed 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, $\pi_\theta\left(a \mid s\right)$. The gradient of reward function is shown below:

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

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 use a Q-Netowrk to replace the Q-Table in Q-Learning. In addition, 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:

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

θ^- and θ are the parameters of target network and current network, respectively. The details of the algorithm are 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, but there will be a problem. If the reward is always positive or negative, some possible actions will disappear and it is difficult to sample these actions. PPO use some contraint tricks, such as clip of policy, to avoid the problem. It makes the probability distribution of actions more reasonable. The loss function based on PPO-clip is as follow.

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

The details of derivation process and algorithm are introduced in appendix A.2 and B.1.2.

2.5.6.3. DPG,DDPG, MADDPG

DPG: Deterministic policy gradient creates a μ function to determine the action instead the sample in PG.

DDPG: DDPG[Lil+15], short 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 ouputs 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 an agent's point of view, 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:

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

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:

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

Where μ' is are the target networks (target policies) with delayed updated parameters.

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 observations 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 learn 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, extra states of environment are not considered during the training.

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 networks 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.

2.6. Platform and Library

2.6.1. GENIUS

GENIUS: An integrated environment for supporting the design of generic automated negotiators [Lin+14].

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 the 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 mechanism, Stacked Algernating Offers Mechanism (SAOM), single-text negotiation mechanisms (st) **??**, multi-text mechanisms (mt) **??**, GA-based negotiation mechanisms**??** and chain negotiations mechasnim**??**. 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 negotiatioin 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 simply the common tasks involved in constructing negotiation driven simulations[Moh+19]. The entire simulation includes multiple simulation steps which is different with 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 an entity in a negotiation mechanism. Several negotiators are natively implemented in NegMAS. AspirationNegotiator, SimpleTitForTatNegotiator and PassThroughSAONegotiator are negotiators, which are developed for SAOM.

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

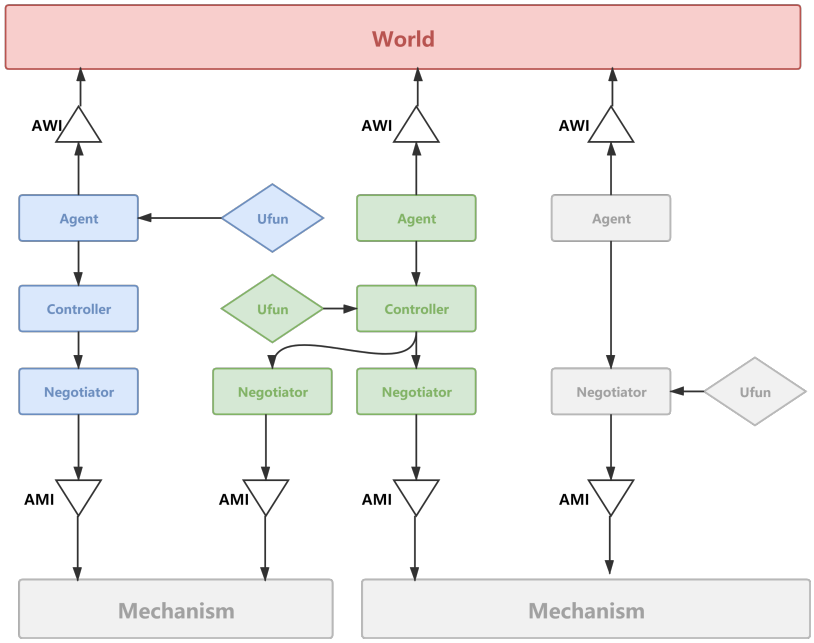

Figure 2.7.: Main components and interactive logic of a simulation in a NegMAS world, Source: Own illustration based on[Moh+19]

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. For example scml2019 will contain all files related to the 2019's version of the competition **??**. There are three main different versions of SCML, which have different designs. In the following sections, the similarities and differences 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

SCML was originally developed as a part of NegMAS, from the version ? it was splited as an indepedent project to research SCM. SCML realized a SCM world to simuate 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 srategies of agent can be splited as three parts: Trading Strategy, Negotiation Control Strategy, Production Strategy.

Trading Strategy Determine 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 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 in SCML-OneShot

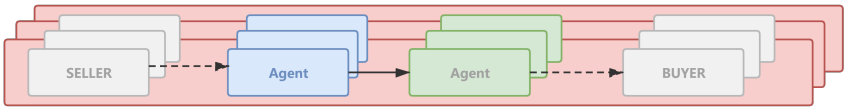

Figure 2.8.: Main running logic of a simulation in a SCML-OneShot world, Each red box represents a day. Broken lines represent exogenous contracts while connected lines are negotiations (about the intermediate product). All negotiations conducted in a given day are about deliveries on the same day. Source: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

Production Graph The production diagram defines 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 manufacting processes for converting the raw materail into the intermediate product and for converting the intermediate product to the final product.

Factories is an entity which in the SCM world convert input products into output products by running manufacturing process on their production lines. All processes take 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 bewtween agents or exogenous contracts, which are contracts 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. 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.

Negotiation Mechanism The negotiation mechanism adopted by SCML2020 is a variant of Rubinstein's alternating offers protocol. It involves two agents, who take turns making offers for a finite number of rounds. One agent open the negotiation with an offer, after which the other agent takes one of the actions(Accepts, Counteroffer, Walks away). All actions are also described in SAOM. This process repeats util 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.

Bulletin Board The world contains a world-readable bulletin board. It contains both static and dynamcial 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 a 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:

- Agents can engage in multiple rounds of negotiations with their negotiation partners.
- All contracts are executed.
- The bulletin-board is updated to reflects new finacial reports, trading prices, and exogenous contract summaries.

2.6.4. 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.

2.6.5. OpenAI Gym

OpenAI Gym is a toolkit for developing and comparing reinforcement learning algorithms [Bro+16].

2.6.5.1. 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.

2.6.5.2. Stable Baselines

The stable baselines developed in the project stable-baselines [Hil+18]. All implemented algorithms with characteristic discrete/continuous actions are shown in 2.1.

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

2.6.6. Ray

Ray is packaged with the following libraries for accelerating machine learning workloads.

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

3. Related Works

In this chapter, related work on the relevant topics of this work is presented and discussed. The topics include autonomous negotiation, multi-agent reinforcement learning. At the last section, the work of reinforcement learning used in autonomous negotiation is presented.

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 negoitator wishes to achieve as part of an agreement. Empricial 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[Scho4]. Many aspiration negotiators are time dependent, such as Boulware(with concession factor $e = 1/2$), Hardliner($e = 0$), Conceder 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 calculates their decision utility which determines whether they accept an offer or not. For time-dependent agents, this is:

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

P_{max} and $P_{min} \in [0, 1]$, Frequently, $\alpha(t)$ is parametrized as a polynomial function: $\alpha(t) = \kappa + (1 - \kappa)\left(\min\left(t, t_{\max}\right)/t_{\max}\right)^{1/e}$.

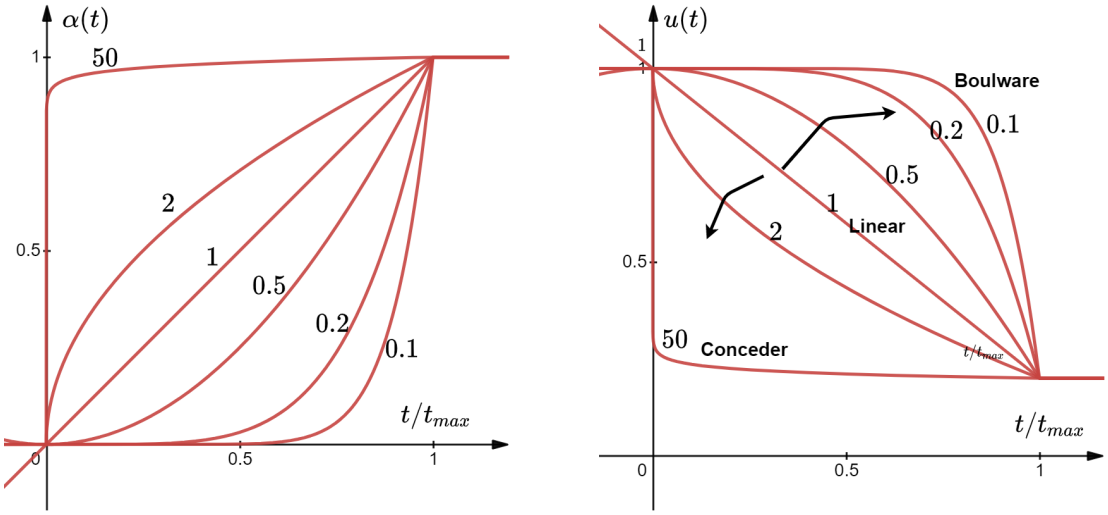

Figure 3.1.: Convexity degree of $\alpha(t)$(left) and $u(t)$(right) with concession factors $e = [0.1, 0.2, 0.5, 1, 2, 50]$.

Where e is the concession factor. k is often set to 0 for simplicity. 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:

$$x^t_{a\to b}[j] = \begin{cases} \min^a_j + \alpha^a_j(t)\left(\max^a_j - \min^a_j\right) & \text{if } u^a_j \text{ is decreasing} \\ \min^a_j + \left(1 - \alpha^a_j(t)\right)\left(\max^a_j - \min^a_j\right) & \text{if } u^a_j \text{ is increasing.} \end{cases} \tag{3.2}$$

Where u_j denotes utility value of issue j set as the offer, $\min_j$ and $\max_j$ mean the minimum and maximum value of issue j, respectively.

The offer will always between 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 to provide a reserved value.

3.1.2. Behavior-based Strategies

Behavior-based and imitation bidding strategies observe the opponent's behavior and decide what to offer and accept. The well-known behavior-based strategy is `tit for tat`. The

strategy is to act cooperatively first, and then mirror 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.

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.

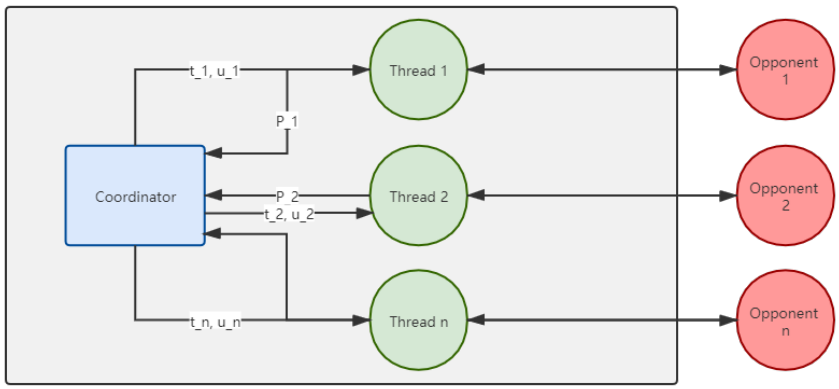

Figure 3.2.: Architecture of the concurrent negotiation agent, best time: t_i and utility value: u_i, probability distributions: P [Wil+12].

Negotiation Threads: The strategy of each negotiation thread is an extension of a recently published, 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.

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

3.1.4. Conclusion

From the analysis of the heuristic negotiation strategy in a specific field, We can get some important parameters, such as time, 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

NegoSI: A novel algorithm named negotiation-based MARL with sparse interactions (NegoSI) is presented by Luowei Zhou. In contrast to traditional sparse-interaction based MARL algorithms, NegoSI adopts the equilibrium concept and makes it possible for agents to select the non-strict Equilibrium Dominating Strategy Profile (non-strict EDSP) or Meta equilibrium for their joint actions [Zho+17].

RLBOA: From the paper [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]. Negotiation strategy was split into three modules: bidding strategy, the opponent model, and the 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.

ANEGMA: Work by [Bag+20]. 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.

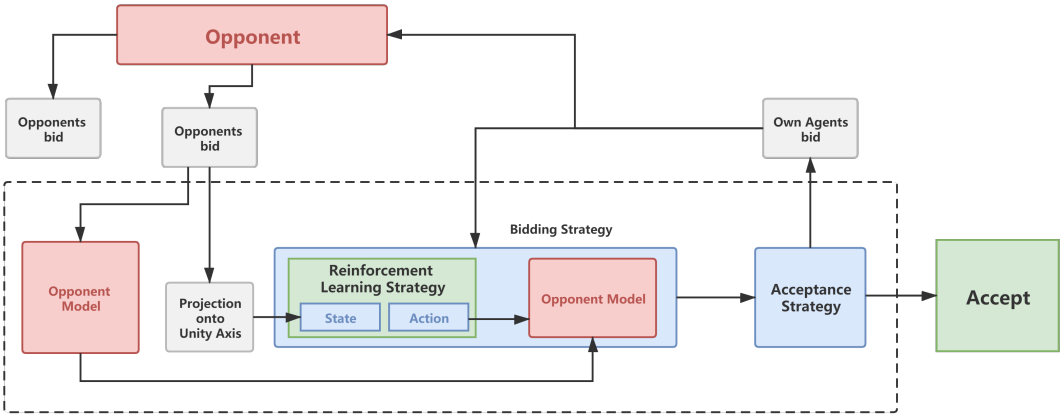

Figure 3.3.: A schematic overview of the RLBOA-framework (within the dashed box), Source: Own illustration based on[Bak+19].

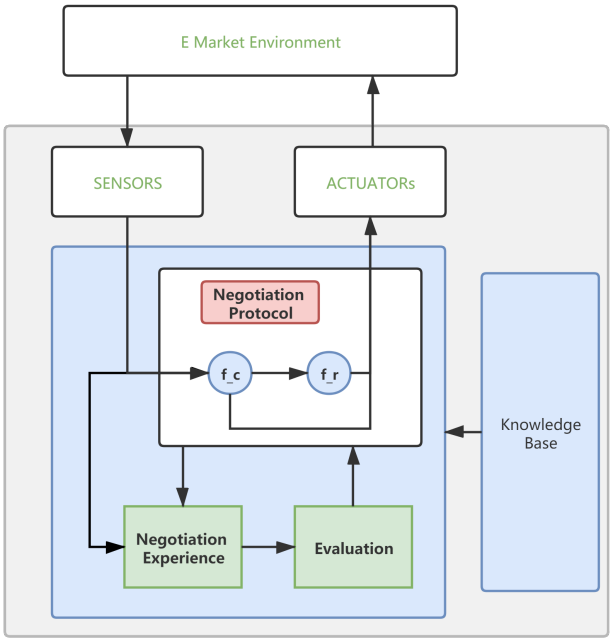

Figure 3.4.: The Architecture of ANEGMA. Source: Own illustration based on[Bag+20].

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. When an agent learns the strategy just with it, the reward is too sparse. Hence, the reward function is needed to provide more frequent feedback or redesign the learning model. 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 necesary 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]. The derivation process can be found in appendix.

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 negotiaton strategy during the simulation. The difficult question is how to deal with non-stationary rewards and non-stationary transition probabilties 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, the 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 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] embes 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.

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 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 RL framework interfaces, which will reduce a lot of work. OpenAI realize 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 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 the top of SAOM to research reinforcement learning algorithms. OpenAI Gym implements 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` 5, the configuration of the negotiation mechanism will be listed in detail.

4.1.2. Model

The model consists of five parts, environment `SBE`, negotiation game, negotiation mechanism, negotiator and reinforcement learning algorithms. Except for the negotiation mechanism mentioned and implemented in sections 2.2 and 2.6.2, others parts will be introduced step by step in following sections.

First, we give the 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 of the negotiation game(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 and detaily introduced in chapter 2. **Negotiator** is a general class of negotiators that can be run in SBE. **Algorithms** are deep reinforcement learning algorithms which receives observation, state from SBE, after training and feedward calculation send the action to SBE.

Entire model is shown in 4.1.

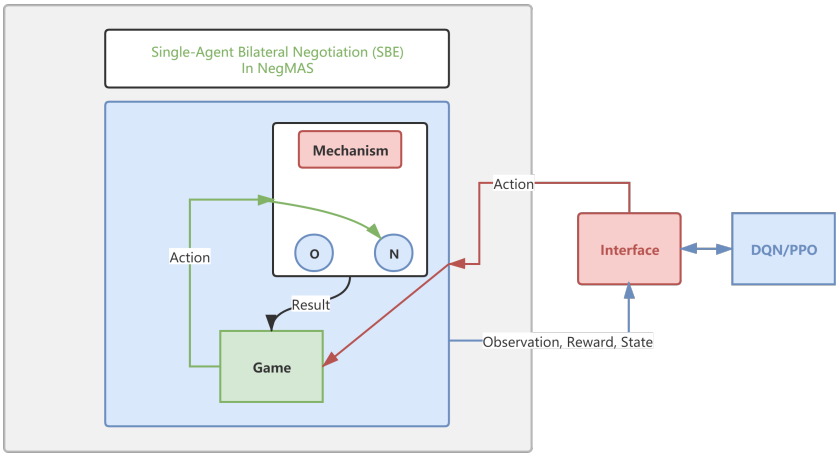

Figure 4.1.: Model for single agent bilateral negotiation based on NegMAS

4.1.3. Single-Agent Environment

The interface of the OpenAI Gym Environments is designed for one agent as standard. Nevertheless, it must be examined how the SBE can be represented via this interface and controlled by the controller. The methods of the interface for the SBE are therefore defined below.

STEP Firstly, sets up but not performs the action received from RL. algorithms for the negotiator. Then, run the negotiation mechanism (such as SAOM) for one step. All actions will be perfomed by the negotiation mechanism. Finally, the 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: Reach the final state or there is no agreement within the maximum running time.
- Info: State of the negotiation mechanism, extra info used for evaluation.

RESET Resets the environment to an initial state and returns an initial observation, initial observation contains negotiators' intial observation and other information relative to the definition of observation space. Reset the time and current step. Create a new negotiation

mechanism session.

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), such as 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 of NegMAS. In this class, there are some parameters, which are received from the mechanism of NegMAS and passed to the RL. algorithms as additional information. The two main methods are defined below.

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

STEP_FORWARD Sets the key logic for the running of the negotiation mechanism, because negotiator can learn different strategy in SBE.

4.1.5. Challenges of the environment

OpenAI Gym provides an unified interface for custome environment. But 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 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 related to the size of negotiation issues.

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

4.1.5.2. Design of Reward Function

The reward function is the focus of the implementation of the RL. algorithm. It is easy to understand, RL. learns startegies by evaluating the value of actions. Therefore, 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 to get the current offer reward, which can be intuitively set as part of the reward function.

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 $\pi : s \rightarrow a$ 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 implictly 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. The approach of this learner is 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

Generally, the design of SBE and training of negotiators are not too complex with the help of OpenAI Gym and NegMAS. The goal of this part is to explore the implementation probability of deep reinforcement learning negotiator. Using the result it can analyse the features of different deep reinforcement learning algorithms used in autonomous negotiation. Although the experiment in this article has achieved negotiators under multiple issues negotiation game, the scale of multiple issues is only 3. First question in this environment is the large scale action space. Second question is the adaptation of strategies when DRL negotiators facing different type of opponent negotiators. Future work may focus on these.

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 simulation 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. 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 are named same as functions in SBE, such as `step`. It provides the pre-definition action spaces and observation spaces. In addition, the function `run` is used to run a complete simulation process. **Scenario** is same as the class `Game` designed in SBE. However, it do not consider the detailed logic of game. The configuration of SCM world and the interactive logic of agent with environment will be set and implemented here. **World** is the key class which simulates the SCM world. Negotiation and manufacturing process are executed by it. **Agent** is a general class of agents(factory manager) that can be run in MCBE. **Interfaces** are a type of classes, which control communication between training environment and training algorithms. **MADRL algorithms** are deep reinforcement learning algorithms used for training multi-agent.

Entire model is shown in 4.2

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. Since MCBE realizes OpenAI Gym `env` interface methods, a new method named `run` is added to execute entire episode.

STEP Runs the simulated world for one step. Not important in this case.

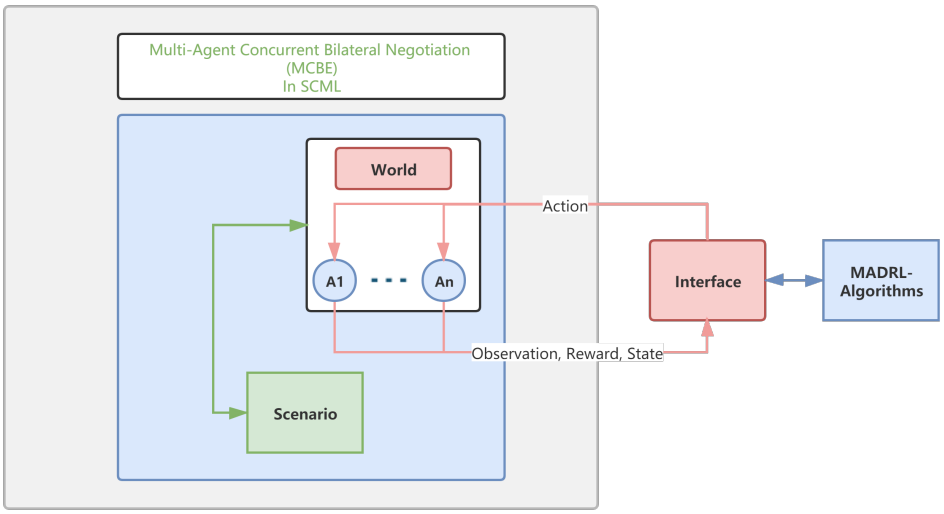

Figure 4.2.: Model for Multi-Agent Concurrent Bilateral Negotiation based on SCML

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. Accumulative reward of single agent 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 simulation is completed.

- 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 logic 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.

Figure 4.3.: Example of supply chain scenario, MyOneShotBasedAgent vs. GreedyOneShotAgent

Scenario Interface consists 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.

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 function designed in official SCML. `Step` in SCML-OneShot is one simulation step. In one simulation step, many negotiation steps will be performed. The action of the agent is a `proposal`, so it needs to be meticulous to control every step 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 splited 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 utility function of agent is defined in the equation 4.1 from [Y+21]. It represents the `excepted profit` of a factory.

$$u_a\left(C^i, C^o\right) = \sum_{c \in \bar{C}^o}\left(p_c q_c\right) - \sum_{c \in C^i}\left(p_c q_c\right) - m_a P_a - \gamma_a \mathrm{tp}\left(p_a^i, s\right) Q_a^{i+} - \alpha_a \mathrm{tp}\left(p_a^o, s\right) Q_a^{o+} \tag{4.1}$$

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:

- $\sum_{c \in \bar{C}^o}\left(p_c q_c\right)$ The total money it earns by selling its produced outputs.
- $\sum_{c \in C^i}\left(p_c q_c\right)$ The total money it pays for buying its inputs.
- $m_a P_a$ The cost of producing the product.

- $\gamma_a \mathrm{tp}\left(p_a^i, s\right) Q_a^{i+}$ The loss of buying too many inputs without using them immediately.
- $\alpha_a \mathrm{tp}\left(p_a^o, s\right) Q_a^{o+}$ 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. 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). Because of non-stationary environment one trick named independent learning insteaded by centralized learning is proposed to train multi-agent.

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 situations.

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 in 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 researches just need to focus on the configuration of training scenario. MCBE decouples well the training environment and running environment.

5. Methods and Experiments

5.1. Single-Agent Bilateral Negotiation Environment (SBE)

In this environment, agent represents the negotiator in negotiation mechanism.

5.1.1. Independent Negotiator in NegMAS

In the environment 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 2.6.5.2 were 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 some RL algorithms is shown in the figure 5.1, and the detailed description of the algorithm is shown in the appendix B.1.1.

5.1.2. Experiment

Figure 5.2 diagrams the Game which consistes of two negotiators: RL negotiator and Opponent negotiator(AspirationNegotiator). Opponent Negotiator can be set in the configuration. All negotiators, which inherit the base abstract negotiator class in NegMAS can be configured in

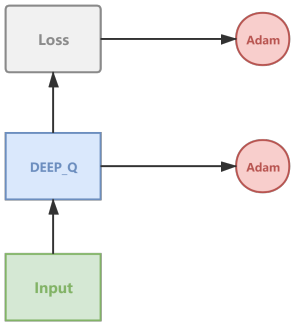

Figure 5.1.: Training logic of DQN

the experiment. AspirationNegotiator is selected as the baseline negoitator in this experiment.

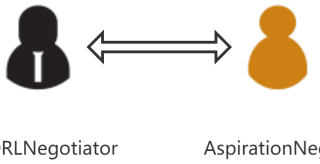

Figure 5.2.: Bilateral Negotiation Game in SBE, My Deep Reinforcement Learning Negotiator vs. Aspiration Negotiator

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

Acceptance strategy actions of agent are `Accept offer`, `Wait` and `Reject offer`. 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 negotiator are set of outcome in negotiation mechanism. The observation is same as observation defined in the acceptance strategy. Before feeding variables into the algorithm, action and observation are normalized.

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, WAIT, END], 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 cases of single issue. The curve of mean episode reward of case **single issue, acceptance strategy** and **single issue, offer strategy** are shown in 5.3. DQN, ACER, PPO and A2C support the 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. The curve of mean epsidoe reward is represented as two type:

- step: combine mean episode reward of all algorithms into one figure
- wall: split the mean episode reward of all algorithms as horizontal independente figure.

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.

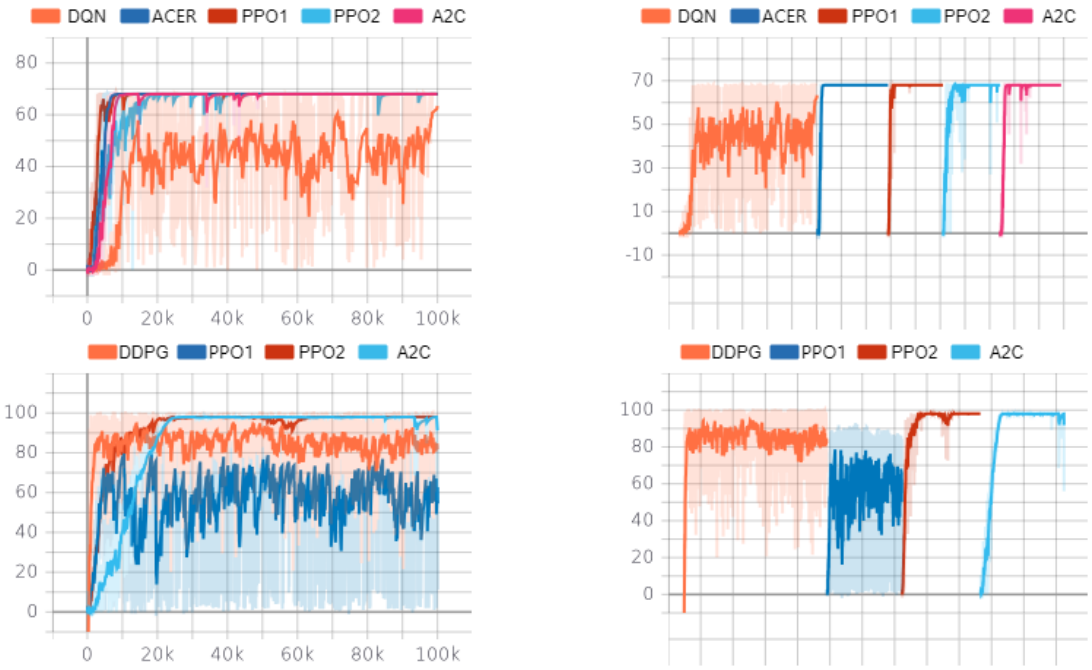

Figure 5.3.: Mean reward of **Acceptance Strategy**(top left(step), top right(wall)) and of **Offer Strategy**(bottom left(step), bottom right(wall)) under single issue negotiation

5.1.2.2. multi issues

The training environment is almost same as the training environment for single issue. It is based on the 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, WAIT, END], Outcomes

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

As same as under single issue cases, the curve of mean episode reward of case **multi-issues, acceptance strategy** and **multi -issues, offer strategy** are shown in 5.4.

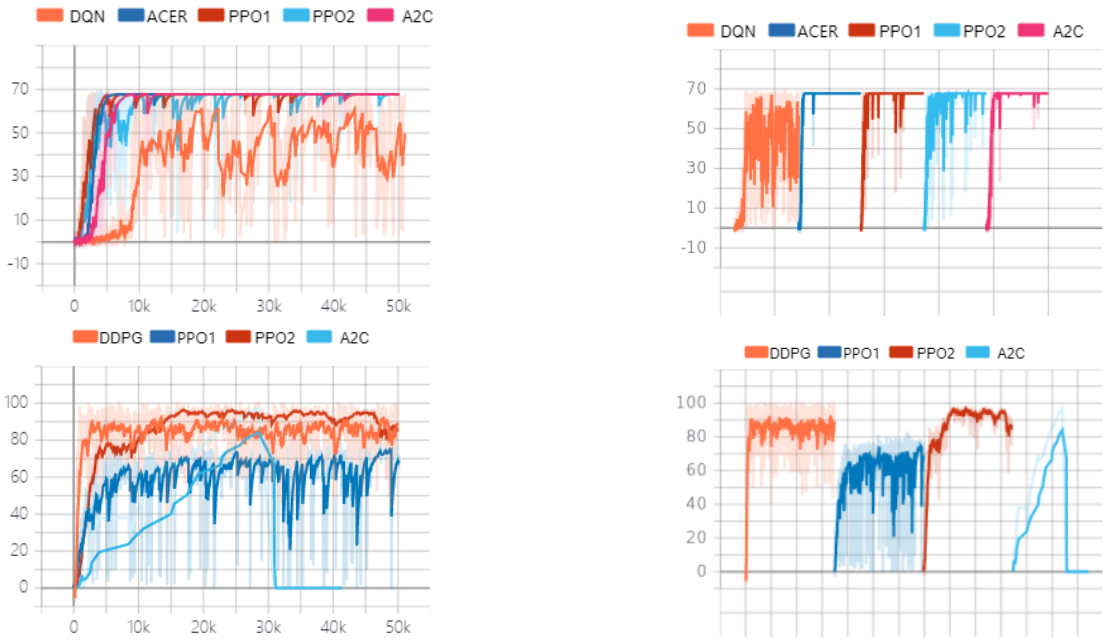

Figure 5.4.: Mean reward of **Acceptance Strategy**(top left(step), top right(wall)) and of **Offer Strategy**(bottom left(step), bottom right(wall)) under multi-issues negotiation

Evaluation The characteristics of the mean episode reward curve are the same as in single-issue cases. After increasing the action space, the training time increased rapidly.

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

In this environment, agent represents the factory manager and negotiation controller in standard SCML and SCML OneShot, respectively.

The agent interacting with environment may be 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 controls 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:

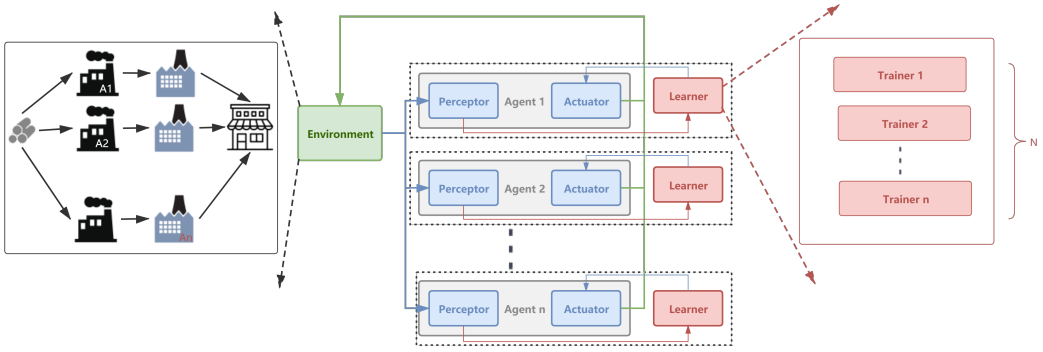

Figure 5.5.: Interactive logic based on the perspective of SCML. N: The maximum number of concurrent negotiations for a single agent

- **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 trainer.

Based on different algorithms, the internal components of the trainer are different. In general, the trainer will handle the training and execution logic of concurrent negotiation. It will be introduced in the specific algorithm, diagramed in figures 5.6 and 5.7.

5.2.1. MADDPG in SCML

In the standard scml environment, two questions are tried to be fixed with maddpg.

Question 1: Dynamical Range Of Negotiation Issues At the beginning of every negotiation in simulator, agent will determine the range which constraints value interval for negotiation issues. In the experiment, the negotiation issues are **QUANTITY**, **PRICE** and **TIME**. After creating the simulation 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 mechanism created inside the entire simulation episode, it has its dynamic range of negotiation issues which affect the negotiation process. This question was raised

based on such a situation.

Question 2: The Offer For Every Round From the description of question 1, we can find, action obtained by algorithm influence only finite the state of environment. Agent(Factory Manager) can not control the function **proposal** of every negotiation round. Every negotiation round has always been controlled by heuristic negotiation strategy. Intuitively, the main influence comes from the joint action of each round of the negotiation. Hence, question 2 **The Offer For Every Round** is proposed naturally. After the basic problem is determined, how to design becomes the current problem.

From an algorithm perspective, the data flow of the model is shown in 5.6. MADDPG used in SCML, one trainable agent(trainer) defined in MADDPG is not equal to the agent defined in SCML. It create D process for action exploration, in this environment, **Dynamical Range Of Negotiation Issues**(Question 1) and **The Offer For Every Round**(Question 2) are needed to be explored. The basic concepts of MADDPG are introduced in chapter background 2.5.6.3.

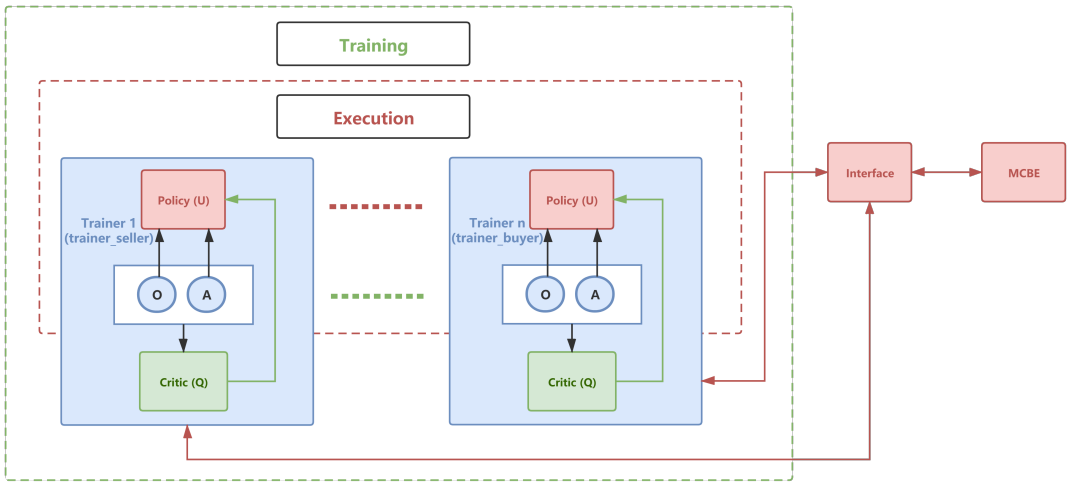

Figure 5.6.: MADDPG used in MCBE

In the model 5.6, policy output action as input to related agent interacting with the environment, 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 size of the action space has a linear relationship with the size of the largest concurrent sale negotiations.

- **trainer_buyer** controls all buy negotiations, the size of the action space has a linear relationship with the size of the largest concurrent purchase negotiations.

Details of the algorithm are described in the appendix B.2.1.

5.2.2. QMIX in SCML-OneShot

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

Question: The Offer For Every Step Unlike in standard scml **Dynamical Range of Negotiation Issues** is controlled by agents, the system takes over the related control and access authority in scml oneshot. Hence, question 1 in the standard library does not need to be discussed here. Although the design of oneshot world is very different with the standard libray, the key question is also how to find the optimal sequence action(offer for every negotiation step).

In the current version QMIX, which is used in the experiment, one trainable agent is related to one negotiation session. When the agents are located in different locations in the scml world, the agents have different concurrent negotiation maximums. Since the agent **A1** shown in Figure 5.5 has three consumers, the maximum value of concurrent negotiations of the agent **A1** is 3. Based on this value, we need to create three trainable agents in algorithm, and each trainable agent control one negotiation session of interative agent.

Data flow is shown in 5.7, the total number of trainers is equal to the sum of the most simultaneous negotiations of all agents. Additionally, unlike in maddpg, in qmix, there is only one global learner, which can control all trainers together.

5.2.3. Experiment

In this section the experiments of the concurrent negotiations within the scml simulated world and their results are presented and discussed. It contains mainly two sub-sections: concurrent negotiations in standard scml world and in scml-oneshot world. For every sub-section, first, the training scenario is defined and how the various agents play is described. Second, 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.

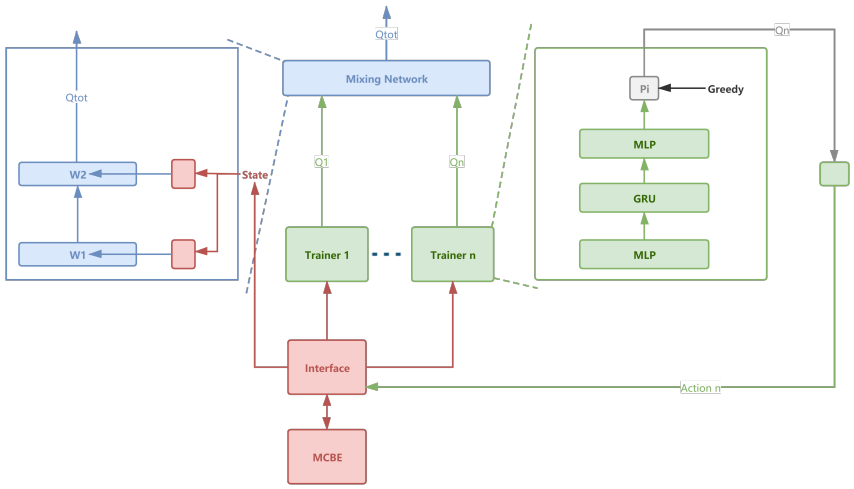

Figure 5.7.: QMIX used in MCBE

5.2.3.1. Concurrent Neogtiations in standard SCML

Standard SCML is a complex simulation world, which contains various parts with specific functions. The breif 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 is used in this experiment. Scenario is diagramed in Figure 5.8.

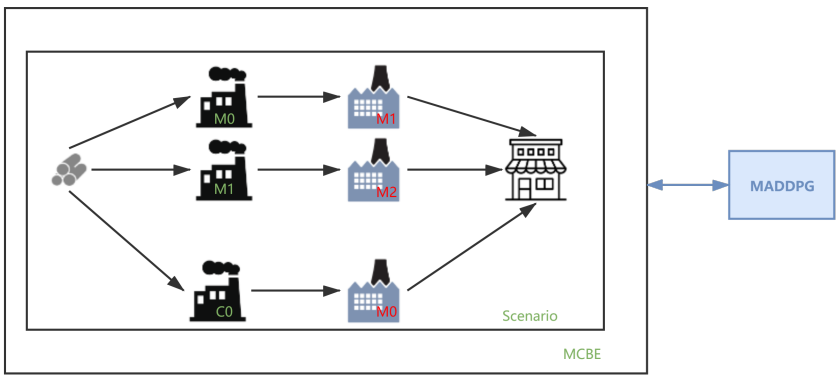

Figure 5.8.: M* represent My Component Based Agent with learner MADDPG, C* represent Opponent Agents, such as IndDecentralizingAgent

Two different categories of agents are used for the experiments:

RL agents were trained with 25000 time episodes in the training environment for the algorithms maddpg. The Rl-Agents are titled as M* in the scneario image 5.8. All RL agents have been trained together. After training, they reloaded the strategy from disk and ran 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 5.8. There are many heuristic agents were developed. In the future work, these agents can be tested and compared.

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 which realized in the 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 setting, the same negotiation partner can conduct concurrent negotiations. 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 better range of negotiation issues(i.e, actions of agents) by training. 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 raised in section 5.2.1.

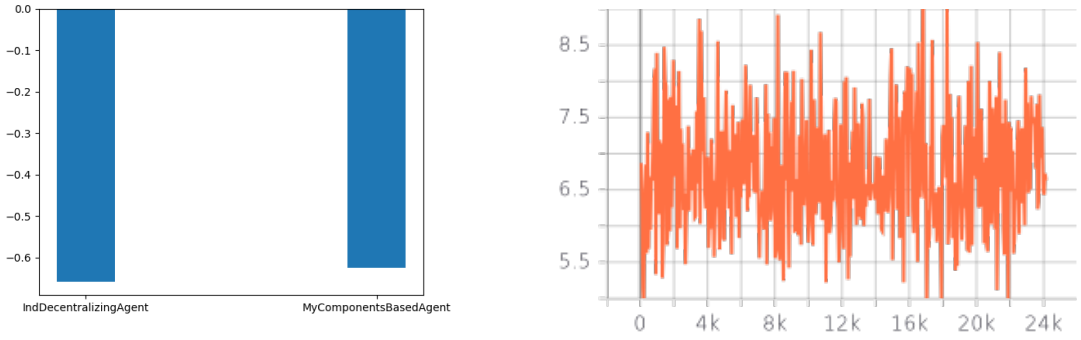

Figure 5.9.: Scores(left) of agents running in simulation world after training, Mean Episode Reward(right)

Evaluation of Question 1: The results after training are shown in figure 5.9. From the mean episode reward curve(right in the figure 5.9), we can know 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 RL agent can obtain. For this type of situation, only range of negotiation issues can be controlled by the RL agents, but others parts, such as offer strategy is a normal heuristic strategy. As a result, the RL agents can perform better than random agent, but can not improve the strategy more.

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 5.9), the score of MyComponentsBasedAgent is not obvious better.

Evaluation of Question 2: Overall, idea from maddpg is very constructive, but it is difficult to learn good strategies with maddpg. Training consumes a lot of time and hardware resources.

5.2.3.2. Concurrent Negotiations in OneShot SCML

scml-oneshot world is a new simpler world from standard scml. This world 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 strategy of three strategies mentioned in section 2.6.3.1.

The breif 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 is used in this experiment. There are two scenarios are creted: self-play and play-with-others.

self-play Scenario is diagramed in Figure 5.10.

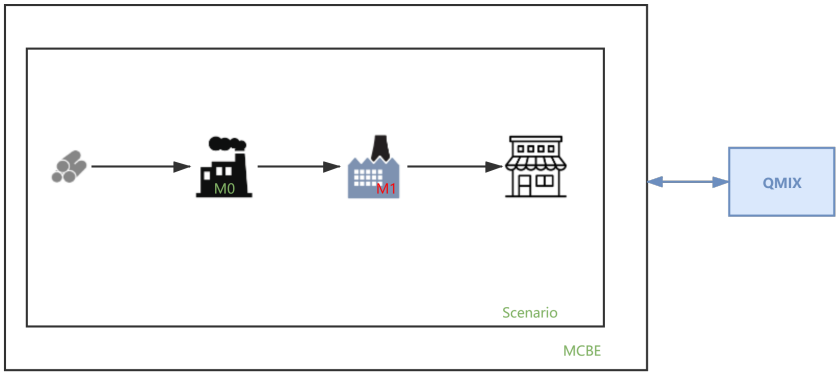

Figure 5.10.: M* represent My Component Based Agent with learner QMIX

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

RL agents were trained with 10000 time steps in the training environment for algorithm qmix. The RL-agents are named as M* in the scenario imamge 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 that need to be set have almost not changed, such as negotiation mechanism, max negotiation steps, max simulation steps, issues, usw. The mainly different is we do not need to set the negotiation rate, because all negotiations will be finished inside each world(simulation) step. The name of simulated world is SCML2020OneShotWorld. The actions of agent is joint outcomes. All attributes are defined in table 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 diagramed in Figure 5.11.

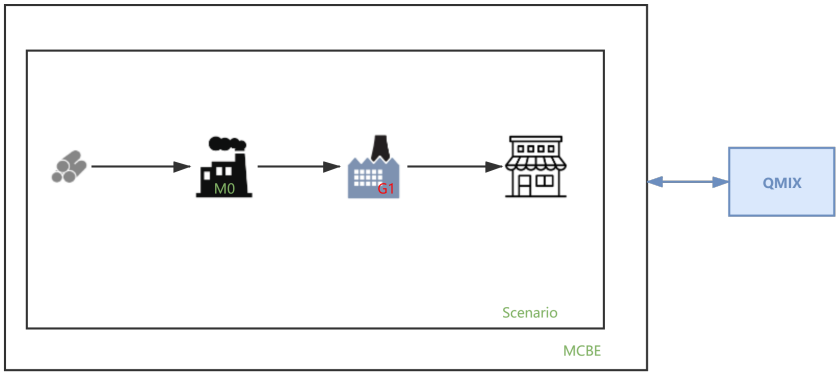

Figure 5.11.: M* represent My Component Based Agent with learner QMIX, G* represent GreedyOneShotAgent

Two different categories of agents are used for the experiments:

RL agents The key parts is same as RL agents mentioned in self-play. In addition to the normal negotiating ability, the agent can also determine whether the negotiator is his teammate. RL agents were trained with 10000 time steps in the training environment for algorithm qmix. The RL-agents are named as 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 same as in self-play scenario. Only the competitors are changed as MyAgent vs. GreedyAgent. The negotiation mechanism is a 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. 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, GreedyAgent] (play-with-others)
Actions	Joint-Outcomes

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

The following paragraphs will evaulate the results based on the two scenarios **self-paly**, **play-with-others** and method qmix raised in section 5.2.2.

Evaluation of self-play Episode mean reward curve is shown in 5.12. After 5000 time steps, the curve converges around 0. In the self-play scenario, the definition of reward is the sum reward of each agent. Therefore, one agent will make more profits, and the other agent will lose more. The reward will eventually converge to a certain value, and it can be seen that the value is close to zero. The explanation of this value will be discussed in the 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 is very useful for training general RL agents. It means that the agent can learn more unpredictable excellent strategies through self-game.

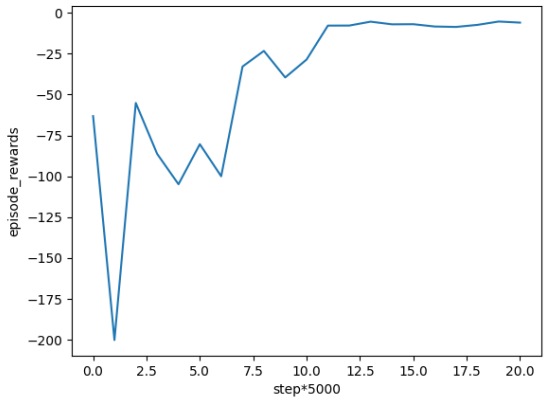

Figure 5.12.: Episode mean reward of self paly under SCML OneShot

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

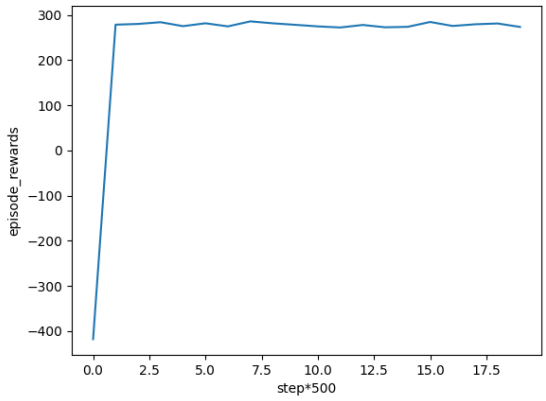

Figure 5.13.: Episode mean reward of my agent vs GreedyOneShotAgent under SCML OneShot

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.

5.3. Conclusion

The training effect of single agent(negotiator) is not bad. But for multi-agent concurrent negotiations it is not easy to implemente. The work of this thesis focus on just 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.

6. Conclusions and Future Work

6.1. Others goal

In the SCM league, profit-maximizing is the goal of RL-agent, in game theory we could get many goals, such as welfare-maximization, pareto optimality. How to achieve these goals with RL-methods based on the developed environments SBE and MCBE? The key part is the design of reward function. For pareto optimality, the reward can be set as the distance between the current utility of offer and the utility of pareto optimal offer. The details of the design of the reward function for other goals can be seen as a major issue in the future.

6.2. Evaluation

The evaluation work of this thesis focuses on two parameters: mean episode reward and score running in the SCML world. It is very inadequate for evaluating multiple agents. There are many metrics for the evaluation of multi-agent system, not only for the RL. multi-agent, 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**, **Autonomy**, **Reactivity** and **Adaptability**, first proposed in the paper[P+10].

Complexity of environment: Many parameters 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

Reward function is an important part of realizing of RL-Agent. The future work can focus on the design of a more effective reward function. Imitation reinforcement learning is a meaningful method through which an appropriate reward function can be deduced based on expert knowledge. This type of 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 very important question.

6.5. Huge scale high performance learning

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. Reverb is Ray and reverb

Appendices

In the appendices, many detailed information are listed, such as algortihms of reinforcement learning.

A. Derivation Process

A.1. Gradient of Reward of PG

The complete derivation process of gradient of reward of PG is step-by-step shown as follows. The reward function is defined as :

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

A.2. Loss Function of PPO

B. Algorithms

B.1. Single-Agent Reinforcement Learning

B.1.1. DQN

For completeness, the DQN algorithm used in Bilateral Negotiation Mechanism from NegMAS is provided below.

Algorithmus 1 : Deep Q-learning with experience replay

Data :

Result :

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

B.1.2. PPO

B.1.3. DPG

B.1.4. DDPG

B.2. Multi-Agent Reinforcement Learning

B.2.1. MADDPG

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

Algorithmus 2 : 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

for *episode = 1 to M* **do**
  Initialize a random process $\mathcal{N}$ for action exploration;
  Receive the intial state from the Simulator;
  for *t = 1 to max-episode-length* **do**
    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.;
    Execute joint actions $a = (a_1, \ldots, a_N)$ and get the reward r and new state $\mathbf{s}'$;
    Store $(\mathbf{s}, a, r, \mathbf{s}')$ in replay buffer $\mathcal{D}$;
    $\mathbf{s} \leftarrow \mathbf{s}'$;
    for *agent i = 1 to N* **do**
      Sample a random minibatch of samples $\mathcal{B}\left(\mathbf{s}^j, a^j, r^j, \mathbf{s}'^j\right)$ from $\mathcal{D}$;
      Set $y^j = r_i^j + \gamma Q_i^{\mu'}\left(\mathbf{s}'^j, a'_1, \ldots, a'_N\right)\Big|_{a'_k = \boldsymbol{\mu}'_k\left(o_k^j\right)}$;
      Update critic by minimizing the loss $\mathcal{L}(\theta_i) = \frac{1}{B}\sum_j \left(y^j - Q_i^{\boldsymbol{\mu}}\left(\mathbf{s}^j, a_1^j, \ldots, a_N^j\right)\right)^2$;
      Update actor using the sampled policy gradient:

$$\nabla_{\theta_i} J \approx \frac{1}{B}\sum_j \nabla_{\theta_i}\boldsymbol{\mu}_i\left(o_i^j\right)\nabla_{a_i} Q_i^{\mu}\left(\mathbf{s}^j, a_1^j, \ldots, a_i, \ldots, a_N^j\right)\Big|_{a_i = \boldsymbol{\mu}_i\left(o_i^j\right)} \tag{B.1}$$

    end
    Update target network parameters for each agent i: $\theta_i' \leftarrow \tau\theta_i + (1-\tau)\theta_i'$
  end
end

B.2.2. QMIX

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.

[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.

[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].

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

[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.

[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.

[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.

[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.

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

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

[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.

[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/.

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

[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.

[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.

[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+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 52

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. 24

ANEGMA ANEGMA 34, 35

ANN Artificial Neural Network viii, 13

BOA the bidding strategy, the opponent model, and the acceptance strategy 34

CNS Concurrent negotiation strategy. ix, 33

DDPG Deep Deterministic Policy Gradient 21, 52

DPG Deterministic Policy Gradient 21, 42

DQN Deep Q-Value Network 20, 42, 52

DRL Deep reinforcement learning. ix, 20, 38, 43, 50, 52

GreedyOneShotAgent A greedy agent based on OneShotAgent 46

IndDecentralizingAgent Independent Centralizing Agent, implemented in standard scml 58

MADDPG Multi Agent Deep Deterministic Policy Gradient iii, x, 21, 22, 48, 55, 56, 58, 64, 73

MADRL Multi-Agent Deep reinforcement learning. 17, 44

MARL Multi-Agent Reinforcement Learning. 17, 44

MCBE Multi-agent concurrent bilateral negotiation environment. x, 38, 44, 46–49, 54, 56, 58, 59, 64, 66

MDP Markov decision process. 7, 18, 37

MDPs Markov decision process. 7

MyOneShotBasedAgent My Deep Reinforcement Learning Agent in SCM-ONESHOT 46

NegMAS NEGotiation MultiAgent System ix, 1, 3, 23–25, 38, 39, 41, 43, 50, 71

NegoSI negotiation-based MARL with sparse interactions (NegoSI) 34

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. ix, 4, 28, 29, 38, 41, 43, 44

PCA Principal Component Analysis.

PDF Portable Document Format.

PG Policy Gradient ix, 19–21, 42, 70

PPO Proximal Policy Optimization 20, 42, 52, 70

PyTorch Python machine learning framework, developed by... ix, 27

QMIX Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning iii, x, 22, 48, 57, 58, 61, 62, 64, 74

Ray Ray provides a simple, universal API for building distributed applications. ix, 30

RL. Reinforcement learning. 4, 37, 38, 40–43, 48, 66

RLBOA RLBOA 34, 42

SAOM Stacked Alternating Offers Protocol, namely in SCML also as Stacked Alternating Offers Mechanism. viii, 9, 10, 23, 38, 40, 51

SARSA State-action-reward-state-action 42

SBE Single-agent bilateral negotiation environment. x, 38–44, 46–48, 50–53, 66

SCM Supply Chain Management 1, 23–27, 37, 44, 47, 67

SCML Supply Chain Management League one of ANAC 2020 and 2021 leagues @ IJCAI 2020 and 2021. ix, x, 1, 3, 24, 25, 38, 43–45, 47, 54–56, 58, 60, 66, 73, 74

SCML-OneShot OneShot World in SCML. x, 24–27, 44, 47, 57

TD Temporal Error. 19

VDN Value Decomposition Networks 22