Multi-Objective Optimization using Reinforcement Learning Framework

Submitted April 2022, in partial fulfillment of
the conditions for the award of the degree **BSc Computer Science.**

Enze REN
20127138

Supervised by Tianxiang CUI

School of Computer Science
University of Nottingham

I hereby declare that this dissertation is all my own work, except as indicated in the text:

Signature _____

Date _____ / _____ / _____

I hereby declare that I have all necessary rights and consents to publicly distribute this dissertation via the University of Nottingham's e-dissertation archive.

Public access to this dissertation is restricted until: DD/MM/YYYY

Abstract

Many real-world problems involve multiple optimizations. Multi-objective optimization reinforcement learning is an extension of reinforcement learning that has multiple reward signals, one reward for each object. All we have to do is optimize multiple rewards to achieve optimization. In this paper, we use deep reinforcement learning methods to train and optimize OpenAI games. Our goal is to utilize a framework to optimize multiple rewards in OpenAI games. This paper describes an algorithm for abandoning the Q-table. The algorithm improves the difficulty of Actor-Critic convergence through Deep Deterministic Policy Gradients based on policy gradients. Based on Deep Deterministic Policy Gradient and Hindsight Experience Replay, multi-objective optimization is realized by combining two kinds of rewards.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Many problems in life are composed of multiple conflicting and influential goals. Optimization is a widely used technique in operations research and has been applied in a series of applications[28]. People often encounter the optimization problem of making multiple goals in a given area as optimal as possible at the same time, which is also called a multi-objective optimization problem(MOP). An MOP can be defined as follows:

$$\begin{aligned} & min_i \quad f(x) = (f_1(x), f_2(x), ..., f_M(x)) \\ & s.t. \quad x \in X, \end{aligned} \tag{1.1}$$

where $f(x)$ consists of M different objective functions and $X \in R_D$ is the decision space[16]. In MOP, many multi-objective optimization problems have been studied in recent years. The typical problem is the multi-object travel problem, which is an NP-hard problem. In the past few decades, evolutionary algorithms have been an effective method to solve MOP. Two popular method are NSGA-II[7] and MOEA/D[40], which can solve many real world problems.

At present, high-efficient methods rely on evolutionary computation and Particle Swarm Optimization[12]. But in the cases like non-deterministic and uncertain environments, the traditional methods may lose their competitiveness. In last decade, Deep Reinforcement

Learning (DRL)[18] has continuously attracted enormous attention from the operation research communities.

Reinforcement learning (RL) is about the interaction between the agent and the environment, through trial and error to learn the best strategy, in order to solve the problem of sequential decision-making in a wide range of natural sciences, social sciences and engineering[33]. The reinforcement learning has been combined with the deep learning in the last several years.The powerful data expression ability of deep neural network is mainly used in reinforcement learning. Value function can be approximated by neural network to achieve end-to-end optimization learning.

Reinforcement learning is acting on feedback from the environment. Reinforcement learning is about constant interaction with the environment, trial and error, and ultimately achieving a specific goal or maximizing the benefits of an overall action. Reinforcement learning does not need the label of training data, but it needs the feedback given by the action environment in each step, whether it is reward or punishment. The feedback can be quantified and the behavior of the training object can be constantly adjusted based on the feedback.

The main difference between reinforcement learning and other machine learning methods is that the environment needs to give feedback and corresponding specific feedback value during reinforcement learning and training. It is not a sorting task, how to distinguish fraudulent customers from normal customers in the financial anti-fraud scenario. Reinforcement learning mainly instructs the trainees how to make decisions at each step and what actions can be adopted to achieve specific goals or maximize benefits. For example,AlphaGo plays Go. AlphaGo is the training object of reinforcement learning. Every step taken by AlphaGo is not right or wrong, but "good or bad". This is a good move. This is a good move. Play "bad", this is a bad move. The training basis of reinforcement learning is that each action environment of AlphaGo can give clear feedback, is it "good"

or "bad"? "Good" and "bad" can be quantified. Reinforcement learning In the AlphaGo scenario, the ultimate purpose of training is to let the pieces occupy more areas on the board and win the final victory.

1.2 Motivation

Neurevolution and population-based algorithms are very mature in solving classical optimization problems and have become an alternative to standard reinforcement learning methods. However, evolutionary algorithms often do not make full use of the collected state and value experience. If you consider reinforcement learning for real-world problems with large resource costs, sample efficiency is crucial. Therefore, an empirical development method is needed to enhance evolutionary algorithms and is expected to provide valuable insights.

Although reinforcement learning still has various thorny problems at present, the industry has begun to try to apply reinforcement learning to practical scenarios. At present, Baidu in China uses a certain reinforcement learning algorithm in the field of automatic driving. However, because reinforcement learning requires interactive trial and error with the environment, this cost is too high in the real world. Therefore, safety officers are required to intervene in real training to timely correct the wrong behaviors of agents. The game can be said to be the most extensive application of reinforcement learning. At present, some MOBA (Multiplayer Online Battle Arena) games on the market basically have the AI of reinforcement learning version in it, the most famous is Arena of Valor AI [39]. The game environment is free to interact, free to try and error, with no real cost. Meanwhile, rewards are relatively easy to set and have obvious Reward mechanisms. At present, some Internet giants are also trying to add reinforcement learning to their recommendation systems, such as Baidu and Meituan. Reinforcement learning is used to improve the diversity of recommendation results, which is complementary to the traditional collaborative filtering CTR prediction model.

Reinforcement learning has made remarkable achievements in recent years, and its application in combinatorial optimization has also made great progress. In reinforcement learning, strategies for combinatorial optimization problems are selected through state changes in order to maximize long-term cumulative rewards. In recent years, a series of remarkable progress has been made in this direction, such as multi-objective optimization problem and online and offline combination optimization. Therefore, applying reinforcement learning to combinatorial optimization has great advantages.

Deep reinforcement learning has recently achieved remarkable results in solving complex combinatorial optimization problems. When these problems are extended to multi-objective problems, the existing DRL methods have slow training speed, difficult to flexibly and effectively apply and deal with multiple sub-problems determined by target weight decomposition[41]. Therefore, it is worth investigating that how to solve the multi-objective optimization problems by deep reinforcement learning in an efficient way.

1.3 Aims and Objectives

The overall structure of the project is investigating the related researches firstly and analysing the feasibility of the research direction, including two aspect: supporting examples and efficiency analysis. The project would then concentrate on the achieving the multiple objectives based on the DDPG (Deep Deterministic Policy Gradient). Two rewards and rewards function will be created to optimize multiple objectives and rewards. Next, the new framework will be trained and the related methods mentioned in the paper will be used in this framework. The framework will also be compared with the typical methods, such as MOEA/D, NSGA-II, in efficiency and be improved according to the performance.

The key objectives of this project are:

1. Studying the existing projects to find a appropriate algorithm and framework used

in reinforcement learning.

2. Scrutinizing the previous researches of multi-objectives reinforcement learning.

3. Investigating the recent researches on Actor-Critic and Deep Deterministic Policy Gradient.

4. Optimizing the reward function to satisfy multiple rewards.

5. Proposing a method to solve multi-objective optimization problem based on the Deep Deterministic Policy Gradient.

6. Comparing the performances between the proposed method and the established methods using the OpenAI-gym as the environment.

7. Applying the evaluate method to test the results and improve the arguments, such as weights.

Chapter 2

Related Work

This chapter mainly involves literature review and presents an overview containing a background of the studies in this field, including two aspects: the method of multi-objectives optimization by reinforcement learning and the theory of DDPG (Deep Deterministic Policy Gradient). Meanwhile, retrospective analysis the primary research methods in this field to enhance the understanding of the project.

2.1 Reinforcement Learning on Multi-Objectives

In recent research, reinforcement learning has been used to solve multi-objective optimization problems. For example, in the power system, the optimal power flow problem includes several different goals: reducing fuel costs, improving voltage stability, and so on. Researchers have implemented the framework for multi-objectives optimization reinforcement learning[19]. In Multi-objectives Optimization Reinforcement Learning (MORL), the dimensions are divided into cells in order to perform search operations by moving states from one cell to another to avoid unmanageable memory of an unlimited number of states. Therefore, in order to solve the multi-objective optimization problem here, we only need to discretize the space into a set of cells and decide how to traverse such a space. After that, we need to establish a standard to assign instant rewards to different states[19].

Some frameworks can reduce training time when dealing with multiple tasks[26]. Most

importantly, they are universal, highly modular, and can adapt to different DRL algorithms, such as DQN, Dueling DQN, asynchronous advantage actor-critic (A3C), Double DQN[25]. An example framework of MORL is created by the H.L. Liao [19], which is fully compared with the promising MOEA/D[40]. It shows that the MORL has the better accuracy and efficiency than the previous algorithms. According to Sejin Kim[13], the MORL can find Pareto front in specific space. For instance, the DeepMind has achieved the reinforcement learning to StarCraft (A game with multiple objectives and rewards)[36].

2.2 Markov Decision Processes

Markov Decision Process (MDP) is a mathematical model of sequential Decision, which is used to simulate the stochastic strategies and rewards that can be realized by an agent in a Markov environment. The theoretical basis of MDP is Markov chain, so it is also considered as a markov model with action [32]. A Markov decision process consists of a quintuple,

$$(S, A, \{P_{sa}\}, \gamma, R) \tag{2.1}$$

The S stands for states, for example, in an automated helicopter system, the coordinates of the helicopter's current position form a state set. The A represents A group of actions, for example, use the joystick to steer the helicopter forward, backward, etc. The P_{sa} is the probability of state transition. The transition from one state of S to another requires the participation of A. P_{sa} represents the probability distribution of other states that will be transferred to after the action $a \in A$ in the current state $s \in S$ (The current state may jump to many states after the execution of A). $\gamma \in [0, 1)$ is the discount factor. When $\gamma = 0$, it is equivalent to taking immediate returns into account, while when $\gamma = 1$, long-term returns are considered as important as immediate returns. $R : S \times A \to R$, R is a reward function, which is often written as a function of S (only related to S), in which case R is rewritten as a $R : S \to R$.

2.3 Bellman Equation

The Bellman Equation is also known as the Dynamic Programming Equation. The Behrman equation is a necessary condition for the optimization of these mathematical optimization methods in Dynamic Programming. This equation expresses "what is the value of the decision problem at a given time" in terms of "the reward ratio from the initial choice is the value of the decision problem derived from the initial choice" [3]. In this way, the dynamic optimization problem is reduced to simple subproblems that obey the "optimization return principle" derived from Behrman.

Policy Function is a Function whose input is s and output is a, which is expressed as $\pi(s)$, where s represents the state and a represents the action. The meaning of Policy Function is the action a that should be selected under the state s. The core problem of reinforcement learning is to optimize the strategy function to maximize the value function introduced later.

The core problem of reinforcement learning is to optimize the strategy function, so how to evaluate the strategy function is optimal? State value function is one of the criteria for evaluating the merits and disadvantages of strategy function $\pi(s)$. In each state s ($s \in S$, S is the set of all states), multiple actions a can be selected ($a \in A$, A is the set of all actions). System will be moved to another state (state sometimes have more than one possible, each state has a probability is transferred to below $\sum_{s'} P^a_{ss'}$), how to ensure that all the action can make the system the global optimal value to define the function, the meaning of value function of the system begins with the current state to a final state system of the cumulative returns expectation, The next state is selected according to the policy function (different actions a will cause the system to move to different states) [34]. So the state value function of the system is related to two factors, one is the current state s, and the other is the strategy $\pi(s)$. Starting from different states, the value may be different, starting from the same state using different strategies, the final value may also be different. Therefore, the established state value function must be established under

different strategies and initial state conditions. The specific form of state value function is as follows:

$$V^{\pi}(s) = E_{\pi}[R_t|s_t = s] \tag{2.2}$$

Where $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$, where r_{t+1} represents the return obtained from the transfer from s to s_{t+1}, γ is the discount factor and its value is $0 \sim 1$. The form of the above state value function can be expressed recursively:

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}[R_t|s_t = s] \\ &= E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t = s\} \\ &= E_{\pi}\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2}|s_t = s\} \\ &= \sum_{a} \pi(s,a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma E_{\pi}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+2}|s_{t+1} = s'\}] \\ &= \sum_{a} \pi(s,a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^{\pi}(s')] \end{aligned} \tag{2.3}$$

Where $P^a_{ss'}$ represents the probability that the state will be transferred from s to s' when action a is selected.

2.4 Deep Q-learning

Markov decision process (MDP) is usually used to formalise a reinforcement learning[31]. An MDP can be described as follows. Let $S = \{s_1, ..., s_N\}$ be the state space and $A = \{a_1, ..., a_r\}$ the action set available to the learning agent. Each combination of current state s, action choice $a \in A$ and next state s has an associated transition probability $T(s|s,a)$ and expected immediate reward $R(s,a)$. The goal is to learn a deterministic stationary policy π, which maps each state to an action, such that the value function of a state s, i.e., its expected return received from time step t and onwards, is maximized. The state-dependent value function of a policy in a state s is then

$$V^{\pi}(s) = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^{k} r_{t+k+1} | s_t = s\}, \tag{2.4}$$

where $\gamma \in [0, 1]$ is the discount factor[37]. The value of taking an action in a state under policy π is represented by a $Q^{\pi}(s, a)$-value which stores the expected return starting from state s, taking action a, and thereafter following π again. The optimal Q-values are defined as

$$Q(s, a) = R(s, a) + \gamma \sum_{s'} T(s^{'}|s, a) maxQ(s^{'}, a^{'}) \tag{2.5}$$

Therefore, the update rule of Q-value is

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma max_{a'} Q(s^{'}, a^{'}) - Q(s, a)] \tag{2.6}$$

Deep Q-learning uses a deep convolutional neural network with a multi-layer tiled convolution filter to simulate the effects of the receptive field. When a nonlinear function approximator such as a neural network is used to represent Q, reinforcement learning is unstable or divergent. This instability comes from the correlation in the observation sequence. In fact, a small update to Q may significantly change the agent's strategy and data distribution, as well as the correlation between Q and the target value[21]. The algorithm of Deep Q-Learning is defined as follow:

The training model based on the Deep Q-Learning outperforms better than the previous methods[20] in the Atari games. According to the Brendan[27], the method implemented on the deep Q-learning achieves performance exceeding the Q-leaning.

2.5 Policy Gradient

In the DQN series reinforcement learning algorithm mentioned above, it mainly approximates the value function and learns based on value. This Value Based reinforcement learning method has been well applied in many fields, but Value Based reinforcement learning method also has many limitations, so other methods are needed in some sce-

Algorithm 1 Deep Q-learning with Experience Replay

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

narios, such as Policy Gradient discussed in this part. It is a policy-based reinforcement learning method, which is Based on strategy.

There are three main problems of DQN series reinforcement learning algorithms. The first is the inability to process continuous action. Methods like DQN generally deal only with discrete actions, not continuous actions. There are workarounds like NAF and DQN, but they are not elegant. Take the classic PuckWorld reinforcement learning problem (Figure 2.1). The environment consists of a square area representing the hockey field, with the large circle representing the individual players and the small circle representing the target puck. In this square environment, the small circle will randomly change its position on the field at regular intervals, while the task of the large circle representing the individual is to approach the puck as quickly as possible. The operation of the great circle is to change the speed of the great circle by applying a time multiplier in four directions, horizontal and vertical. If the magnitude and direction of the force can be flexibly selected, then it is difficult to use ordinary algorithms such as DQN. Because the strategy is a force with a specific value and a direction, one way to do this is to split the force horizontally and vertically. Then the force is composed of two continuous vectors. This strategy is difficult to express in the discrete way, but it is easy to model with the Policy Based reinforcement

learning method.

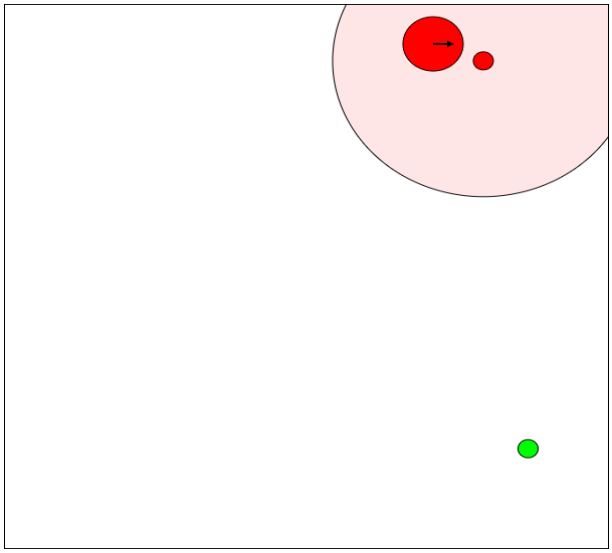


Figure 2.1: PuckWorld

The second point is the inability to deal with problems in the constrained state. When features are used to describe a certain state in the state space, it is possible that due to the limitations of individual observation or modeling, the original two states in the real environment have the same feature description after modeling, which may lead to the failure of our value Based method to obtain the optimal solution. In this case, the Policy Based reinforcement learning method is also effective [34]. The third is that you can't solve the random strategy problem. The optimal strategy corresponding to Value Based reinforcement learning method is usually deterministic strategy, because it selects a behavior with the maximum Value from numerous behavioral values, while the optimal strategy of some problems is random strategy, in this case, it cannot be solved by Value Based learning. At this time, the Policy Based reinforcement learning method can also be considered. Due to the above reasons, the Value Based reinforcement learning method cannot cover all scenarios. New methods are needed to solve the above problems, such as policy-based reinforcement learning.

Assume that the value function is approximately expressed in the Alue Based reinforcement learning method and an action value function $\hat{q}$ is introduced, which is described by parameter w and accepts state s and action a as input. The estimated action value can

be obtained after calculation, that is,

$$\hat{q}(s, a, w) \approx q_\pi(s, a) \tag{2.7}$$

Under the Policy Based reinforcement learning method, similar ideas are sampled, but the strategy is approximated at this time. Here the policy π can be described as a function containing the parameter θ, that is,

$$\pi_\theta(a, s) = P(a|s, \theta) \approx \pi(a|s) \tag{2.8}$$

After the strategy is expressed as a continuous function, the optimization method of continuous function can be used to find the optimal strategy. The most common method is gradient ascent.The first step is to find a functional target that can be optimized. The simplest optimization goal is the expectation of initial state harvest, that is, the optimization goal is,

$$J_1(\theta) = V_{\pi_0}(s_1) = \mathbb{E}_{\pi_0}(G_1) \tag{2.9}$$

However, some problems have no clear initial state, so the optimization goal can define the average value, namely,

$$J_{avV}(\theta) = \sum_s d_{\pi_0}(s) V_{\pi_0}(s) \tag{2.10}$$

Where $d_{\pi_0}(s)$ is the static distribution of markov chain states generated based on policy π_0. Or defined as the average reward for each time step,

$$J_{avR}(\theta) = \sum_s d_{\pi_0}(s) \sum_a \pi_0(s, a) R_s^a \tag{2.11}$$

Whether using J_1, J_{avV} or J_{avR} to represent the optimization goal, finally to θ derivation of the gradient can be represented as,

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_0}[\nabla_\theta log \pi_\theta(s, a) Q_\pi(s, a)] \tag{2.12}$$

Policy Gradient Method usually converges to the limit, and evaluation strategies are

inefficient and highly varied. However, It has better convergence properties. Value-Based method needs to update the value function before it can be reflected in the policy. However, some small changes in the value function may lead to big changes in the policy, resulting in poor convergence. Of course, if the epsilon of exploration factor is set at 1/k, the Monte-Carlo Contorl obtained meets the GLIE condition, and the function number corresponding to this method will converge to the optimal value function. The poor convergence of value-based method means that it is easy to oscillate and difficult to converge, while the latter means that it will converge to the optimal Value function "eventually" [23]. It is not known how long the "final" will be, so policy-based has an advantage on this issue. In addition policy gradient method is highly efficient at high latitudes and in continuous motion space. After all, value-based methods need to compute $\underset{a}{max}\, Q(s, a)$. If the set of actions is large, then the Max operation is very computationally intensive, whereas the policy-based RL method does not have this problem.

2.6 Actor-Critic

Actor-Critic is equivalent to the combination of Deep Q-learning and Policy gradient. Actor is a neural network. Critic is also a neural network, they are different neural networks, Actor is used to predict the probability of behavior, Critic is to predict the value in this state. Combined with the method of Policy Gradient (Actor) and Function Approximation (Critic), Actor selects behavior based on probability, and Critic (can be q-learning or value-based) estimates the Value of each state. Subtract the value of the next state from the value of this state, (TD-error), and the Critic tells the actor that the next action should be updated more. If TD-error is positive, the next action should be updated more[14]. If it is negative, the actor should be updated less. Criric judges the score of the behavior based on the Actor's behavior, and Actor modifies the probability of the selected behavior according to the score of Critic. In conclusion, Actors are the strategy functions $\pi_\theta(a|s)$, learning a strategy to get the highest possible reward. A Critic is a value function $V_\pi(s)$, an estimate of the value function of the current strategy, that is, an evaluation of the quality of the actor. With the help of value functions, the actor-critic

algorithm can update parameters in a single step, without waiting until the end of the turn.

In order to solve the contradiction between High Variance and High bias, they can be combined together to make use of the advantages of value based and Policy based methods and supplement their shortcomings. The result is the omnipresent actor-Critic class method. Specifically, it is to construct an all-purpose agent, which can directly output policies and evaluate the current policies in real time through value function. So you need two networks, an Actor responsible for generating the policy and an Critic responsible for evaluating the policy. This is a bit like an actor performing while a critic corrects his performance, both of which are constantly being updated. This complementary training can be more effective than a separate strategy network or value function network[10]. Actor-based Critic is composed of Policy Gradients and value-based Critic. The Critic can see the potential reward in the state it is in through the relationship between learning environment and reward and punishment, so it points the Actor to update the Actor at every step. With Policy Gradients, actors can only wait one turn to update. Therefore, it can be updated in a single step, faster than the traditional Policy Gradient[2]. However, Actor-Critic involves two neural networks, each time updating parameters in a continuous state, each parameter update before and after the correlation, resulting in the neural network can only one-sided view of the problem, or even learn nothing. In addition, the behavior of actors depends on the Value of Critic, but because Critic is inherently difficult to converge, it is even more difficult to converge when updated with actors (DDPG is later proposed to solve the convergence problem).

In actor-Critic algorithm, two groups of approximations need to be made. The first group is the approximation of the policy function,

$$\pi_\theta(a, s) = P(a|s, \theta) \approx \pi(a|s) \tag{2.13}$$

The second group is the approximation of value function. For state value and action value function, they are,

$$\begin{aligned} \hat{v}(s,w) &\approx v_\pi(s) \\ \hat{q}(s,a,w) &\approx q_\pi(s,a) \end{aligned} \tag{2.14}$$

In the Monte Carlo strategy gradient Reinforce algorithm, the parameter update formula for the policy is,

$$\theta = \theta + \alpha \nabla_\theta log \pi_\theta(s_t, a_t) v_t \tag{2.15}$$

In general, the Critic calculates the optimal value of the state vt through the Q network, while the Actor uses the optimal value vt to iterate and update the parameter θ of the policy function, and then selects the action and gets feedback and new state. The Critic uses feedback and the new state update Q network parameter w, after which the Critic uses the new network parameter w to help Actor calculate the optimal value of the state, vt. Evaluation points are based on TD error, Critic uses neural network to calculate TD error and update network parameters, Actor also uses neural network to update network parameters.

To avoid the positive number trap, make the Actor's update weights both positive and negative. So let's subtract the Q value from their mean V: $Q(s,a) - V(s)$. To avoid the need to estimate V and Q, unify Q and V. Since $Q(s,a) = \gamma V(s^{'}) + r - V(s)$, $TD\text{-}error = \gamma V(s^{'}) + r - V(s)$ was get. $TD\text{-}error$ is the weight value in Actor updated policy with weight. Now the Critic no longer needs to estimate Q, but rather V. According to The Marklov chain, $TD\text{-}error$ is the loss required by the Critic network, that is to say, the Critic function needs to minimize $TD\text{-}error$. The algorithm of the Actor-Critic is defined as follow:

2.7 Deep Deterministic Policy Gradient

When the dimensions of the action space become higher, even highly successful DQNS become difficult to learn, especially when faced with continuous action Spaces. For ex-

Algorithm 2 Actor-Critic with Eligibility Traces (episodic), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\theta)$
Input: a differentiable state-value function parameterization $\hat{v}(s,w)$
Algorithm parameters: trace-decay rates $\lambda^\theta \in [0,1]$, $\lambda^w \in [0,1]$; step size $\alpha^\theta > 0$, $\alpha^w > 0$
Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ and state-value weights $w \in \mathbb{R}^d$ (e.g., to 0)
Loop forever (for each episode)
    Initialize S (first state of episode)
    $z^\theta \leftarrow 0$ (d'-component eligibility trace vector)
    $z^w \leftarrow 0$ (d'-component eligibility trace vector)
    $I \leftarrow 1$
    Loop while S is not terminal (for each time step):
        $A \sim \pi(\cdot|S,\theta)$
        Take action A, observe S', R
        $\delta \leftarrow R + \gamma\hat{v}(S',w) - \hat{v}(S,w)$ (if S' is terminal, then $\hat{v}(S',w) \doteq 0$)
        $z^w \leftarrow \gamma\lambda^w z^w + I\nabla_w \hat{v}(S,w)$
        $z^\theta \leftarrow \gamma\lambda^\theta z^\theta + I\nabla_\theta ln\pi(A|S,\theta)$
        $w \leftarrow w + \alpha^w \delta z^w$
        $\theta \leftarrow \theta + \alpha^\theta \delta z^\theta$
        $I \leftarrow \gamma I$
        $S \leftarrow S'$

ample, in the robot arm control scenario, the rotation Angle is a continuous variable and the robot arm has multiple degrees of freedom. In other words, DQN is a method based on value functions, which is difficult to deal with large action space, especially continuous action. Because it's hard for a network to have that many outputs, and it's hard to find the maximum Q value in those outputs[5]. Therefore, deterministic strategy gradient algorithm can be used before deep learning. The depth deterministic strategy gradient is based on the above actor criticism method. In terms of action output, network is adopted to fit strategy function and output action directly, which can cope with the output of continuous action and large action space.

Deep Deterministic Policy Gradient (DDPG) is an algorithm which concurrently learns a Q-function and a policy. It uses off-policy data and the Bellman equation to learn the Q-function, and uses the Q-function to learn the policy. This approach is closely connected to Q-learning, and is motivated the same way: the optimal action $a^*(s)$can be found by solving

$$a^*(s) = arg\max_a Q^*(s,a) \tag{2.16}$$

DDPG interleaves learning an approximator to $Q^*(s,a)$ with learning an approximator to $a^*(s)$, and it does so in a way which is specifically adapted for environments with continuous action spaces. When the number of discrete actions is limited, the maximum is fine because the Q value of each action was calculate separately and compared directly. (This also immediately gives the action to maximize Q). However, when the action space is continuous, it is not possible to evaluate the space in detail, and solving the optimization problem is very important. Using normal optimization algorithms would make calculating $max_a Q^*(s,a)$ a very expensive subroutine[17]. And since it needs to be run every time the generation wants to take action in the environment, this is unacceptable. Because the action space is continuous, the function $Q^*(s,a)$ is presumed to be differentiable with respect to the action argument. This allows the establishment of an efficient, gradient-based learning rule for policy $\mu(s)$ that takes advantage of this fact. Then, instead of running an expensive optimization subroutine each time, the $max_a Q(s,a)$ is computed, it can be approximated with $max_a Q(s,a) \approx Q(s,\mu(s))$.

As a random strategy, in the same strategy, in the same state, the action adopted is based on a probability distribution, that is, it is uncertain. The deterministic strategy is simpler. Although the probabilities of the actions are different in the same state, there is only one maximum probability. If you only take the actions with the maximum probability and remove the probability distribution, it will be much simpler. That is, as a deterministic strategy, the same strategy, in the same state, the action is uniquely deterministic, that is, the strategy becomes,

$$\pi_\theta(s) = a \tag{2.17}$$

The gradient calculation formula of random strategy gradient based on Q value is as follows,

$$\nabla_\theta J(\pi_\theta) = E_{s\sim\rho^{pi},a\sim\pi_\theta}[\nabla_\theta log\pi_\theta(s,a)Q_\pi(s,a)] \tag{2.18}$$

Status of the sample space for ρ^π, $\nabla_\theta log\pi_\theta(s,a)$ is a function score, visible random strategy gradient needed in the space of whole action ϕ_θ samples. The gradient calculation formula

of deterministic strategy gradient of DPG based on Q value is as follows,

$$\nabla_\theta J(\pi_\theta) = E_{s\sim\rho^{pi}}[\nabla_\theta \pi_\theta(s,a)\nabla_a Q_\pi(s,a)|_{a=\pi_\theta(s)}] \tag{2.19}$$

Compared with the formula of random strategy gradient, there is less integration of action and more derivative of return Q function of action.

DDPG has 4 networks the current Q network of DDQN is responsible for selecting action A for the current state S by using εGreedy method, performing action A to obtain new state $S^{'}$ and reward R. Place the sample into the empirical playback pool, select action $S^{'}$ for the next state $R^{'}$ sampled in the empirical playback pool using greedy method for target Q network to calculate target Q value[29]. When the destination Q network calculates the destination Q value, the current Q network updates the network parameters and periodically copies the latest network parameters to the destination Q network. The target Q network of DDQN calculates the target Q value based on the experience playback pool and provides it to the current Q network. The target Q network periodically copies the latest network parameters from the current Q network. As DDPG, Critic current network, Critic target network and DDQN's current Q network, target Q network function positioning is basically similar, but it has its own Actor strategy network, so it does not need εGreedy method such a selection method, This part of DDQN function to DDPG can be done in Actor current network. The greedy method is used to select the action $A^{'}$ for the next state $S^{'}$ of sampling in the empirical playback pool. This part of work can be put into the Actor target network because it is used to estimate the target Q value. A part of the target Q value is calculated based on the experience playback pool and $S^{'}$, $A^{'}$ provided by the target Actor network, which is still placed in the target network because it is an evaluation[11]. After the Critic network calculates a part of the target Q value, the current Critic network calculates the target Q value, updates the network parameters, and periodically copies the network parameters to the Critic network. In addition, the Actor's current network updates the network parameters based on the target Q-value calculated by the current Critic network, and periodically copies the network parameters

to the Actor's target network.

DDPG uses soft updates, where parameters are updated a little bit at a time, i.e. ,

$$\begin{aligned} w^{'} &\leftarrow \tau w + (1-\tau) w^{'} \\ \theta^{'} &\leftarrow \tau \theta + (1-\tau) \theta^{'} \end{aligned} \tag{2.20}$$

Where τ is the update coefficient, which is usually small, such as 0.1 or 0.01. At the same time, in order to add some randomness to the learning process and increase the coverage of learning, DDPG will add a certain amount of noise $\mathcal{N}$ to the selected action A, that is, the expression of the final interaction action A with the environment is,

$$A = \pi_\theta(S) + \mathcal{N} \tag{2.21}$$

For Critic current network, DDPG loss function is similar to DQN, both are mean square error,

$$J(w) = \frac{1}{m} \sum_{j=1}^{m} (y_j - Q(\phi(S_j), A_j, w))^2 \tag{2.22}$$

For the current network of Actor, the loss function is different from that of PG,

$$\nabla_J(\theta) = \frac{1}{m} \sum_{j=1}^{m} [\nabla_a Q(s_i, a_i, w)|_{s=s_t, a=\pi_\theta(s)} \nabla_\theta \pi_{\theta(s)}|_{s=s_t}] \tag{2.23}$$

If for the same state, two different actions a_1 and a_2 are output, and two feedback Q values are obtained from the current Critic network, namely Q_1 and Q_2. Assuming that $Q_1 > Q_2$, more rewards can be obtained by taking action 1, then the idea of strategic gradient is to increase the probability of a_1 and reduce the probability of a_2, in other words, Actor wants to get as large a Q value as possible. Therefore, the loss of Actor can be simply understood as the larger the feedback Q value is, the smaller the loss is, and the smaller the feedback Q value is, the greater the loss is. Therefore, it is only necessary

to take a negative sign for the Q value returned by the state estimation network,

$$J(\theta) = -\frac{1}{m}\sum_{j=1}^{m} Q(s_i, a_i, w) \tag{2.24}$$

The structure of DDPG is similar to actor-Critic. DDPG can be divided into two large networks: policy network and value network. DDPG continues DQN's idea of fixed target networks, and each network is subdivided into target networks and real networks. But there are some differences in the target network updates. Let's start with policy networks, or actors[38]. The output of Actor is a deterministic action, which is generated by the network defined as $a = \mu_\theta(s)$. In the past, Policy gradient adopted a random strategy, and each acquisition action required sampling of the distribution of the current optimal strategy, while DDPG adopted a deterministic strategy, which was directly determined by the function μ. The estimator network of Actor is the $\mu_\theta(s)$, where θ is the parameter of the neural network. The estimator network is used to output real-time actions. In addition, Actor has a target network of the same structure but different parameters, which is used to update the value network Critic. Both networks output action. Look at the value network, which is Critic. Its function is to fit the value function $Q_w(s, a)$. There is also an estimation network and a target network. The two networks both output the current state value $Q - value$ on the output side, but differ on the input side. The target network input of Critic has two parameters, which are the observation value of the current state and the action output by the Actor target network[38]. The estimated network input of the Critic is the estimated network output action of the current Actor. The target network is used to calculate Q_{target}. The algorithm of the DDPG is defined as follow,

Algorithm 3 Deep Deterministic Policy Gradient

Input: Initialize policy parameters θ, Q-function parameters ϕ, empty replay buffer D
Set target parameters equal to main parameters $\theta_{targ} \leftarrow \theta$, $\phi_{targ} \leftarrow \phi$
repeat
Observe state s and select action $a = clip(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
Execute a in the environment
Observe next state $s^{'}$, reward r, and done signal d to indicate whether $s^{'}$ is terminal
Store $(s, a, r, s^{'}, d)$ in replay buffer D
If s is terminal, reset environment state.
if it's time to update **then**
for however many updates **do**
Randomly sample a batch of transitions, $B = (s, a, r, s^{'}, d)$
Compute targets

$$y(r, s^{'}, d) = r + \gamma(1 - d)Q_{\phi_{targ}}(s^{'}, \mu_{\theta_{targ}}(s^{'})) \tag{2.25}$$

Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s^{'},d) \in B} (Q_\phi(s, a) - y(r, s^{'}, d))^2 \tag{2.26}$$

Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s)) \tag{2.27}$$

Update target networks with

$$\begin{aligned} \phi_{targ} &\leftarrow \rho\phi_{targ} + (1 - \rho)\phi \\ \theta_{targ} &\leftarrow \rho\theta_{targ} + (1 - \rho)\theta \end{aligned} \tag{2.28}$$

end for
end if
until convergence

2.8 Hindsight Experience Replay

HER (Hindsight Experience Replay) is a data structure proposed by Open AI to solve the problem of feedback and reward sparse stored samples. It adopts the progressive learning method, adjusts the difficulty of the task to make the model gradually learn and continuously enhance the ability of strategy. In this paper, the replay buffer is stored in the unit of sequence, and the future sampling mode is adopted. Sample B sequences from replay buffer, select a time from b sequences to get B samples, each sample has a certain probability to set $achieved_goal$ to the state at any time at the current time[1]. Sparse rewards are a problem in many reinforcement learning environments, such as a robotic arm grasping an object giving a positive reward when it grasps an object and zero the rest of the time. Another example is the environment composed of N coins, each coin has two sides as $0, 1$, the action is to choose one of the coins to flip, there is a target coin state combination (eg. $0, 1, 1, 1, 0...$), a reward is given when the state is reached. The experience pool buffer stores a large number of transitions of the same reward in the reinforcement learning algorithm based on these environments, which is very challenging for network learning. One solution is reward shaping[9]. For example, in writing the environment of glute-eating snakes, the original reward is given when the snake eats food and the negative reward is given when the snake does not eat food instead of the reward is given when the snake does not eat food, so as to guide the snake to the food and facilitate training.

HER adopted an ingenious idea to solve the above problems. Let me give you an example to illustrate this problem. A football player kicked a penalty kick. The m missed and shot to the right of the goal. For RL algorithms that do not use HER technology, this reward is of little significance and can learn little information. For the RL algorithm using HER technology, it learned from this error that its current kicking Angle and force can shoot the ball away from the goal. That is, when the Agent does not reach the expected target state, but reaches another state S, it will make a self-judgment[1]. If the goal of the environment is your current state, then your previous transitions are correct. If

you still use the above example of shooting is: if the goal is to the right, their current kicking Angle and power is more appropriate. HER is specific to the off-policy algorithm. HER needs a certain understanding of the environment. It is necessary to know the mapping relationship from the state S to the goal in advance (take the mechanical arm for example, the state may be the angles of multiple joints, and the target is the coordinate of a point in three-dimensional space. If the state is known, the coordinate of the end of the mechanical arm in space can also be calculated). At the same time, a new reward calculation mechanism needs to be established, which depends on the goal and the state S. Generally, when the $goal^{'}$ mapped by the state S is close to the $goal$, the reward will be given. A list of records for each episode transition is also required, which serves as a post-experience replay after each episode ends[24]. In addition, compared with the original dimension, the status dimension accepted by RL algorithm adds the dimension of goal, namely RL acceptance,

$$obs = \{state||goal\} \tag{2.29}$$

HER has a good performance in multi-objective reinforcement learning. The algorithm of Hindsight Experience Replay is defined as follow,

Algorithm 4 Hindsight Experience Replay (HER)

Given:

- an off-policy RL algorithm $\mathbb{A}$, ▷ e.g. DQN, DDPG, NAF, SDQN
- a strategy $\mathbb{S}$ for sampling goals for replay, ▷ e.g. $\mathbb{S}(s_0, ..., s_T) = m(s_T)$
- a reward function $r : S \times \mathcal{A} \times \mathcal{G} \rightarrow \mathbb{R}$. ▷ e.g. $r(s, a, g) = -[f_g(s) = 0]$

Initialize $\mathbb{A}$ ▷ e.g. initialize neutral networks
Initialize replay buffer $\mathcal{R}$
for eposide $= 1, M$ **do**
  Sample a goal g and an initial state s_0.
  for $t = 0, T - 1$ **do**
    Sample an action a_t using the behavioural policy from $\mathbb{A}$:
    $a_t \leftarrow \pi_b(s_t||g)$ ▷|| denotes concatenation
    Execute the action a_t and observe a new state s_{t+1}
  end for
  for $t = 0, T - 1$ **do**
    $r_t := r(s_t, a_t, g)$
    Store the transition $(S_t||g, a_t, r_t, s_{t+1}||g)$ in R ▷ standard experience replay
    Sample a set of additional goals for replay $G := \mathbb{S}$(current episode)
    for $g^{'} \in \mathbb{G}$ **do**
      $r^{'} := r(s_t, a_t, g^{'})$
      Store the transition $(s_t||g^{'}, a_t, r^{'}, s_{t+1}||g^{'})$ in $\mathbb{R}$ ▷ HER
    end for
  end for
  for t=1, N **do**
    Sample a minibatch B from the replay buffer R
    Perform one step of optimization using $\mathbb{A}$ and minibatch B
  end for
end for

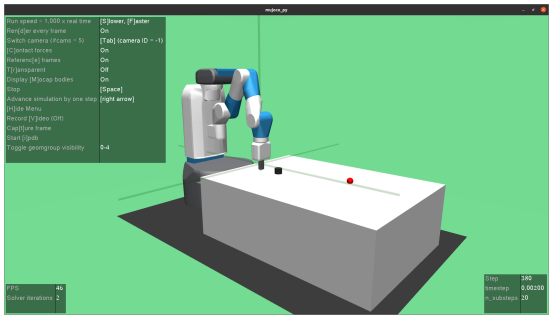
Chapter 3

Design

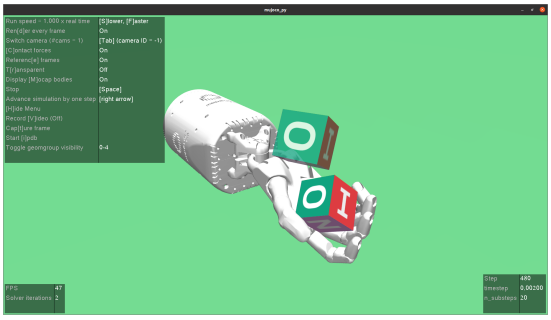
The topic of this project is to use reinforcement learning to solve the problem of multi-objective optimization. At present, reinforcement learning is used to solve multi-objective optimization problems, including A3C and DQN combined with Actor-Critic. Therefore, the DDPG selected in this project solves the defects in Q-learning and integrates PG and AC network structure. The design of this algorithm is based on the Deep Deterministic Policy Gradient, which was proposed by the Deepmind. On this basis, the Project introduced the Hindsight Experience Replay (HER) data structure. The algorithm model is gradually learned to enhance the capability of the strategy. In this project, this paper sets up three replay strategy modes, none, final and random. The users can select one of them to train the model.

To idealize the learning of strategies, the project references a strategy learning. Its function is to learn and update the strategies generated by the reinforcement learning algorithm. So the output of the training model is a constantly updated policy file. The project also has a tree model algorithm. It is a nonparametric supervised learning method that summarizes decision rules from a set of feature and labeled data and presents these rules in a tree diagram structure to solve classification and regression problems. Decision tree algorithms are easy to understand and can be applied to any type of data. In solving various problems, tree models, especially as the core of various ensemble algorithms, perform well. The project also defines a boundary loss algorithm that maps decisions

to their associated costs. Minimize the error of each training sample. Since the focus of this problem is multi-objective optimization, there is a special algorithm to solve the multi-objective optimization problem. It sets up two rewards and then combines them to get a reward. The project also defines a boundary loss algorithm that maps decisions to their associated costs. Minimize the error of each training sample. Since the focus of this problem is multi-objective optimization, there is a special algorithm to solve the multi-objective optimization problem. It sets up two rewards and then combines them to get a reward. In terms of exploration methods, random exploration and noise exploration are used in this paper to avoid the trap of exploration.



(a) Fetch



(b) Hand

Figure 3.1: Two Examples of Robotics

The game, which name is robotics from OpenAI-Gym are selected to test the algorithm. The robotics that based on the MuJoco includes eight games (environments). They are *FetchReach*, *FetchSlide*, *FetchPush*, *FetchPickAndPlace*, *HandManipulateBlock*, *HandReach*, *HandMainpulateEgg* and *HandMainpulatePen* (see Figure 3.1). The project only choose six of them as the environments. Finally, the project produces a csv file and users can create results' figures according this. The brief architecture of project is,

- Application Layer: OpenAI-Robotics
- Algorithm Layer: DDPG

- Network Layer: Actor-Critic

- Back-end Framework: Tensorflow

In the construction process of this project, each module plays its own role, and the modules are effectively connected into a whole. There are parts to speed up data processing and data calculation. There are also modules that are responsible for multithreading, enabling parallel computation during project training. In terms of final testing, there are also powerful modeling tools for programmers to demonstrate the results of training. Here are some other additions that are used in algorithm design

3.1 Message Queue Library

ZeroMQ (ZMQ for short) is a message queue-based multithreaded network library that abstracts the low-level details of socket types, connection processing, frames, and even routing to provide sockets across multiple transport protocols. ZMQ is a new layer in network communication, between the application layer and the transport layer (divided by TCP/IP). It is a scalable layer that can run in parallel and spread across distributed systems[15]. ZMQ is not a separate service, but an embedded library, it encapsulates the network communication, message queue, thread scheduling and other functions, to the upper layer to provide a concise API, applications by loading library files, call API functions to achieve high-performance network communication. The ZeroMQ API provides sockets (a kind of generalization over the traditional IP and Unix domain sockets), each of which can represent a many-to-many connection between endpoints. Operating with a message-wise granularity, they require that a messaging pattern be used, and are particularly optimized for that kind of pattern. The ZMQ ensure the communication between the python and the C++, because some parts of the project is compiled by C++.

3.2 Parallel Computing

To save training time, the project uses MPI. MPI is a messaging interface, a standardized and portable messaging system designed to work with a variety of parallel computers. This standard defines the syntax and semantics of library routines and allows users to write portable programs in major scientific programming languages (Fortran, C, or C++). Since its publication, the MPI specification has become the leading standard for messaging libraries on parallel computers. High performance computer vendors and well-known open source projects such as MPICH and OpenMPI provide implementations.

It can start a group of processes at the same time. In the same communication domain, different processes have different numbers. Programmers can use the interface provided by MPI to assign different tasks to different processes and help them communicate with each other and finally complete the same task. It's like a contractor gives workers a job number and then comes up with a plan to assign tasks to different numbers of workers and get them to communicate with each other to complete the task.

Mpi4py is a Python library built on top of MPI, written primarily in Cython. Mpi4py makes It easy to pass Python data structures across multiple processes[6]. Mpi4py is a powerful library that implements many of the interfaces of the MPI standard, including point-to-point communication, intra-group collection communication, non-blocking communication, repetitive non-blocking communication, inter-group communication, and so on. Not only Python objects, mpi4py also has good support for Numpy and is very efficient to deliver. It also provides SWIG and F2PY interfaces to enable Fortran or C/C++ programs packaged in Python to use mpi4py objects and interfaces for parallel processing.

Also, joblib helps with this. Joblib is a set of tools for providing lightweight pipeline processing in Python. Functional disk caching and delayed re-evaluation (pattern memory) can be performed transparently, enabling simple and straightforward parallel computation. Joblib has been optimized to be fast and robust, especially for large amounts of

data, with certain optimizations for numpy arrays. Joblib solves two problems with as few code and flow control changes as possible. The first is to count the same thing twice. For example, during prototyping for computationally intensive work (such as scientific development), code is often re-executed multiple times, but manual solutions to mitigate this problem are error-prone and error-prone. Often leads to unreproducible results. The second is to make it transparent to disk. It is difficult to efficiently persist any object that contains a lot of data. Using joblib's caching mechanism avoids hand-written persistence and implicitly links the file on disk to the execution context of the original Python object. Therefore, joblib persistence is good for restarting application state and computing jobs, for example after a crash.

3.3 Physical Modeling

Because the environments used in this paper are the Robotics in OpenAI, the MuJoCo is essential for this paper to demonstrative the training model. It is convenient for user to observe how phisical model working. MuJoCo stands for Multi-Joint Dynamics with Contact. MuJoCo is a C/C library with A C API for researchers and developers. The runtime simulation module is tuned for maximum performance and operates on low-level data structures preassigned by the built-in XML parser and compiler. Users define models using the native MJCF Scenario Description Language – an XML file format designed to be as human-readable and editable as possible[35]. You can also load URDF model files. The library includes interactive visualizations with native GUIs, rendered in OpenGL. MuJoCo further disclosed a large number of utility functions used to calculate quantities related to physics. MuJoCo can be used for model-based computation such as control synthesis, state estimation, system identification, mechanism design, inverse dynamics data analysis, and parallel sampling for machine learning applications. It can also be used as a more traditional game emulator and interactive virtual environment.

MuJoCo combines advanced contact mechanics with generalized coordinate systems.
Game engines (ODE, Bullet, Physx) typically handle joint constraints through numerical

optimization, resulting in unstable and inaccurate multi-rigid body systems. MuJoCo uses generalized coordinate system and contact mechanics method based on optimization method. Meanwhile, MuJoCo can resolve reversible contact mechanics. While most modern physics engines solve linear complementarity problems to handle constraints, MuJoCo allows software contact and other constraints, and includes a unique inverse dynamics model to do data analysis[8]. A new friction model is proposed to support the simulation of rolling friction and torsional friction. In addition, MuJoCo supports 3D tendon models, which are related to OpenSim's muscle models, but have been optimized to support practical features such as artificial muscle simulation and wire-driven dexterous hand simulation. MuJoCo, on the other hand, uses [formulas] to separate model and runtime data, facilitating parallelism. The model can be compiled into MJB binary file format to improve the loading speed.

3.4 Replay Buffer

Replay buffer in this paper is an improved construct for Replay buffers. In the most traditional Deep Q-network[30], when a sample is collected from the replay buffer, each sample is sampled with the same probability. In other words, each sample will be studied at the same frequency. But in reality, each sample is different in its difficulty, and each sample will yield different results.

In this project, the replay buffer solves this problem nicely. It gives a certain weight to the sample according to how the model performs on the current sample, and the probability of the sample being sampled is related to this weight. The worse the interaction performance is, the higher the corresponding weight is and the higher the sampling probability is. Conversely, the better the interaction, the lower the weight and the lower the sampling probability. During training, the training was randomly sampled from the Replay Buffer. The Replay Buffer is where many transitions are stored. A transition is a set of data (state, Action, reward, next-state).

If you don't use historical data you have to throw away the previous data every time you update it. New data is then generated to update network parameters. If this is the case, it will consume resources and waste resources to face a slightly more complex problem. At the same time, the old data generated by the old parameter network is not completely unavailable for updating the new network parameters, such as empirical playback, and ppo derivation process proved by mathematics. So there's no need to completely waste old data, because if you waste useful information, it should be algorithmically optimized. In addition, the iteration of the algorithm is very similar to that of traditional data structure algorithms. For example, the Fibonacci sequence problem can be solved intuitively with recursion, but it is obvious that there is a lot of room for optimization with full recursion, and there is a lot of repeated calculation. Later, dynamic programming algorithm is used to calculate Fibonacci sequence, which can avoid a lot of repeated calculation. This example is analogous to many machine learning algorithms that are very resource-intensive and clearly have a lot of room for optimization. Because when people study a class of problems, they always think of the simplest and most intuitive solution first. For example, many algorithms of the operating system are iterated from simple and inefficient to complex and efficient. Because intuitive algorithms do have a lot of potential for optimization.

3.5 Numerical Calculation

SciPy is a collection of mathematical algorithms and useful functions built on top of Python's NumPy extension. It adds power to interactive Python sessions by providing users with high-level commands and classes for manipulating and visualizing data. With SciPy, an interactive Python session becomes a data processing and system prototyping environment comparable to systems such as MATLAB, IDL, Octave, R-Lab, and SciLab. Another benefit of making SciPy based on Python is that it also provides a powerful programming language that can be used to develop advanced programs and specialized applications. Scientific applications using SciPy benefit from add-on modules developed by developers around the world in numerous areas of the software domain. Everything

from parallel programming to web and database subroutines and classes is available to Python programmers[4]. All of these functions are available in addition to SciPy's math library. In this project, SciPy will help you with scientific computing.

3.6 Building Environment

All projects are built in Ubuntu20.04. The coding, training and testing procedures are all run in the Linux system. There are many reasons. Linux, the open source gene, has many advantages over Windows and Mac. Linus who is the inventor of the Linux famously said, Talk is cheap, Show me the code. Simply, for all software on GNU/Linux, the source code is naked to the user, not a black box. Open source makes it possible for programmer worldwide to research and contribute code to Linux. Linux, meanwhile, is highly stable even on low-cost hardware. It can also use terminal tools to free up hands and improve execution efficiency. Standalone deep learning development environment relies on GPU and Linux configuration is extremely fast. In deep learning networks, all kinds of matrix operations, often thousands of dimensions of large matrix operations, and Nvidia GPUs can accelerate these operations, compared with CPU, can bring ten times or even higher speed in convolutional neural networks. For tensorFlow used in this project, Linux will be easier to install.

Chapter 4

Implementation

4.1 Reinforcement Learning

It thinks that the agent is interacting with the environment. Suppose the environment is fully observable, including the state set $\mathcal{S}$, the action set $\mathcal{A}$, the initial state distribution $p(s0)$, the transition probability $p(s_{t+1}|s_t, a_t)$, and the reward function $r : S \times A \to \mathbb{R}$ discount coefficient $\gamma \in [0, 1]$. For continuous control tasks, Deep Deterministic Policy Gradient (DDPG) shows good performance. This is essentially the practice of a non-policy actor critic. For multipurpose continuous control tasks, DDPG can be extended with Universal Value Function Approximation (UVFA). UVFA basically generalizes the Q-function to multiple target states. The Q value is related to goals and state-action pairs. For robotic tasks, if the purpose is difficult and the reward is sparse, the agent can work well for a long time before it can learn anything. Hindsight Experience Replay encourages agents to learn from their target state. She challenges robotic tasks by redesigning goals. This allows for possible training, random swaps between the target and the real target.

4.2 Multi-Objective

The objective g is the desired position and orientation of the object. Specifically, we use g^e and e to represent the environment. It shows the actual objective as input to the environment to distinguish it from the objective used in the hindsight setup. In this paper,

we will consider when states can be used to express objectives. This leads to the concept of achieving the objective state g^s. The state s consists of two sub-vectors, the realized goal state s^g representing the position and orientation of the manipulated object, and the context state s^c, or $s = (s^g||s^c)$. where $||$ represents concatenation. In this example, we define $g^s = s^g$ to represent the achieved objective, which has the same dimensions as the actual objective g^e in the environment. The context state s^c contains the remaining state information, such as the linear and angular velocities of all robot joints and objects. The actual objective g^e can be replaced by the achieved objective g^s to facilitate learning. Orbitals consisting only of target states are denoted as τ. Use τ to denote all objectives achieved in orbital τ. That is, $\tau_g = (g_0^s, ..., g_T^s)$.

Therefore, consider the sparse reward r. There is a tolerance between what is expected and what is achieved. If the object is not within the tolerance of the actual objective, the agent receives a reward signal of -1 for each transition. Otherwise, the reward signal received by the agent is 0. In a multi-objective configuration, the agent receives an environment objective g^e and a state input $s = (s^g||s^c)$. I want to train objectives-conditioned policies to effectively generalize their behavior to different environmental objective g^e.

Multi-objective RL is considered as strategy learning with objective conditions. The random variable is displayed according to the random variable value corresponding to the capital letter. For example, $V(X)$ represents the set of effective values of random variable x, and $P(x)$ represents the probability function of random variable x. The objective receiving an objective $g^e \in V(G^e)$ is at the beginning of this group. The agent interacts with the environment in the T times step. In each step t, the agent observes the state $s_t \in V(S_t)$ and performs an action to obtain $a_t \in V(A_t)$ compensation for the specified by the input target $r(s_t, g^e) \in R$. Using $\tau = s_1, a_1, s_2, a_2, ..., s_{T-1}, a_{T-1}, s_T$ to represent the track, where $\tau \in V(\tau)$. The orbit of the objective g^e is specified τ probability $p(\tau|g^e, \theta)$

and parametric policy $\theta \in V(\Theta)$.

$$p(\tau|g^e,\theta) = p(s_1)\prod_{t=1}^{T-1} p(a_t|s_t,g^e,\theta)p(s_{t+1}|s_t,a_t) \tag{4.1}$$

The transition probability $p(s_{t+1}|s_t,a_t)$ means that the probability of a state transition for a given action is independent of the goal, expressed as $S_{t+1} \perp G^e|S^t,A^t...$ .. for all τ, g^e and θ, assuming that $p(\tau|g^e,\theta)$ is not zero. The expected return of a policy parameterized by θ is as follows:

$$\begin{aligned}\eta(\theta) &= \mathbb{E}[\sum_{t=1}^{T} r(S_t,G^e)|\theta] \\ &= \sum_{g^e} p(g^e)\sum_{\tau} p(\tau|g^e,)\sum_{t=1}^{T} r(s_t,g^e)\end{aligned} \tag{4.2}$$

Off-policy reinforcement learning methods use experience replay to exploit bias towards variance and potentially improve sample efficiency. If off-policy, the desired expression (4.2) looks like this:

$$\eta^{\mathcal{R}}(\theta) = \sum_{\tau,g^e} p_{\mathcal{R}}(\tau,g^e|\theta)\sum_{t=1}^{T} r(s_t,g^e) \tag{4.3}$$

where R represents the replay buffer. Typically, the trajectory τ is randomly sampled from the buffer. However, the trajectories in the playback buffer are often unbalanced relative to the achieved objective *tau*. In multi-objective reinforcement learning, we want to encourage the agent to follow trajectories of different objective states while maximizing the expected return. It's like maximizing the delegation of agents trying to achieve multiple objectives. Propose a reward-weighted entropy objective for multi-objective reinforcement learning. is given as follows:

$$\begin{aligned}\eta^{\mathcal{H}}() &= \mathcal{H}_p^w(\tau^g) \\ &= \mathbb{E}_p[log\frac{1}{p(\tau^g)}\sum_{t=1}^{T} r(S_t,G^e)|\theta]\end{aligned} \tag{4.4}$$

For simplicity, use $p(\tau^g)$ to represent $\sum_{g^e} p_{\mathcal{R}}(\tau^g,g^e|\theta)$ that is the probability of the objective state trajectory τ^g appearing. Since the expected value is also calculated based on $p(\tau^g)$, the proposed objective is the weighted entropy of τ^g, which is $\mathcal{H}_p^w(\tau^g)$, where

weight w is cumulative reward $\sum_{t=1}^{T} r(s_t, g^e)$ in our example.

There are two interpretations of the objective function equation (4.4). The first explanation is to maximize the weighted expected return when the rare orbit weight is high. Note that this weighting mechanism has no effect if all trajectories are displayed uniformly. The second explanation is to maximize reward-weighted entropy. In this case, the higher the reward, the higher the weight. This purpose encourages the agent to learn how to achieve different goal states and maximize the expected reward.

In equation (4.4), the weight $log(\frac{1}{p(\tau^g)})$ is not restricted, which makes the training of general function approximation unstable. Therefore, we propose a safe alternative objective $\eta^{\mathcal{L}}$ which is essentially a lower bound on the original purpose.

For the purpose of establishing a safe alternative, we sample tracks from the regeneration buffer using the proposed distribution $q(\tau^g) = \frac{1}{Z} p(\tau^g)(1 - p(\tau^g))$. $p(\tau^g)$ represents the distribution of the target track in the playback buffer. The alternative $\eta^{\mathcal{L}}(\theta)$ is the lower bound of the objective function $\eta^{\mathcal{H}}(\theta)$. That is, $\eta^{\mathcal{L}}(\theta) < \eta^{\mathcal{H}}(\theta)$.

$$\begin{aligned} \eta^{\mathcal{H}}(\theta) &= \mathcal{H}_p^w(\tau^g) \\ &= \mathbb{E}_p[log\frac{1}{p(\tau^g)} \sum_{t=1}^{T} r(S_t, G^e)|\theta] \end{aligned} \tag{4.5}$$

$$\eta^{\mathcal{L}}(\theta) = Z \cdot \mathbb{E}_q[\sum_{t=1}^{T} r(S_t, G^e)|\theta] \tag{4.6}$$

$$q(\tau^g) = \frac{1}{Z} p(\tau^g)(1 - p(\tau^g)) \tag{4.7}$$

where Z is the normalization coefficient of $q(\tau^g)$. $\mathcal{H}_p^w(\tau^g)$ is the weighted entropy, in this case the weight is the cumulative reward $\sum_{t=1}^{T} r(S_t, G^e)$.

4.3 Prioritization

The optimization process is projected onto the preferred sampling frame to optimize expression (4.6), which is the alternative objective. In each iteration, we first create a proposed distribution $q(\tau^g)$ with higher entropy than $p(\tau^g)$. This allows the agent to learn from a wider distribution of objective states. The entropy of $q(\tau^g)$ is higher than that of $p(\tau^g)$. Let it be the objective probability density function in the playback buffer,

$$p(\tau^g), where\ p(\tau_i^g) \in (0,1) and \sum_{i=1}^{N} p(\tau_i^g) = 1 \tag{4.8}$$

The proposed probability density function is defined as,

$$q(\tau_i^g) = \frac{1}{Z} p(\tau_i^g)(1 - p(\tau_i^g)), where \sum_{i=1}^{N} q(\tau_i^g) = 1 \tag{4.9}$$

The proposed objective distribution has equal or greater entropy,

$$\mathcal{H}_q(\tau^g) - \mathcal{H}_p(\tau^g) \geq 0 \tag{4.10}$$

For the purpose of optimizing the alternative objective using preferential sampling, we need to know the probability distribution of the objective state trajectory $p(\tau^g)$. Because LVM is suitable for modeling complex distributions, we use a Latent Variable Model (LVM)[22] to model the underlying distribution of $p(\tau^g)$. Specifically, we use $p(\tau^g|z_k)$ to show the orbital distribution of the objective state conditioned by latent variables. Suppose this is a Gaussian distribution. z_k is the k-th latent variable, where $k \in \{1, ..., K\}$ and K are the number of latent variables. The resulting model is mathematically a mixture of Gaussian distributions (MoG).

$$p(\tau^g|\phi) = \frac{1}{Z} \sum_{i=k}^{K} c_k \mathcal{N}(\tau^g|\mu_k, \Sigma_k) \tag{4.11}$$

where each Gaussian distribution $\mathcal{N}(\tau^g|\mu_k, \Sigma_k)$ with its own mean μ_k and covariance Σ_k. c_k stands for the mixing factor and Z is the partition function. The model parameter ϕ includes all means μ, covariance Σ_i and mixing factor c_k. Priority sampling uses the complementary predicted density of the objective state trajectory τ^g as the priority, which is given as follows,

$$\bar{p}(\tau^g|\phi) \propto 1 - p(\tau^g|\phi) \tag{4.12}$$

The complementary density represents the likelihood that the objective state trajectory τ^g will appear in the regeneration buffer. The high complementarity density corresponds to the rare occurrence of the objective orbit. To increase the entropy of the training distribution, we wish to over-sample the trajectories of these rare target states during regeneration. Therefore, we use the complement density to construct the proposed distribution as a joint distribution,

$$\begin{aligned} q(\tau^g) &\propto \bar{p}(\tau^g|\phi)p(\tau^g) \\ &\propto (1 - p(\tau^g|\phi))p(\tau^g) \\ &\approx p(\tau^g) - p(\tau^g)^2 \end{aligned} \tag{4.13}$$

Using preferential sampling, the agent learns to maximize returns from a more diverse objective distribution. When the agent plays samples, the agent first ranks all trajectories with respect to the proposed distribution $q(\tau^g)$, and then directly uses the ranking number as the sampling probability. This means that rare objectives have a higher rank and have an equally high priority. Here we use rank instead of density. The reason is that the rank-based variants are more robust as they are not affected by outliers or the magnitude of the density. In addition, its heavy hem guarantees a wide variety of samples. Mathematically, the probability that the orbit will be regenerated after prioritization is,

$$q(\tau_i^g) = \frac{rank(q(\tau_i^g))}{\sum_{n=1}^{N} rank(q(\tau_n^g))} \tag{4.14}$$

N is the total number of tracks in the playback buffer, and $rank(\cdot)$ is the ranking function.

The complete training algorithm is summarized in equations (4.5), (4.6), (4.7) and Algorithm 5. In short, we propose a multi-objective Reinforcement learning algorithm (Section 4.1) so that reinforcement learning agents can learn multi-objective tasks more efficiently. Integrate the target entropy term into the normal expected return target. For the maximization objective, equation (4.4), we derive lower bounds for the alternative purpose of equations (4.5), (4.6) and (4.7), i.e. the original objective. Each iteration uses priority sampling based on a higher entropy proposal distribution and leverages an off-policy reinforcement learning approach to maximize expected rewards.

Algorithm 5 Multi-objective Reinforcement Learning

repeat
  Sample goal $g^e \sim p(g^e)$ and initial state $s_0 \sim p(s_0)$
  for *step_per_epoch* **do**
    for *step_per_episode* **do**
      Sample action $a_t \sim p(a_t|s_t, g^e, \theta)$ from behavior policy
      Step environment: $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
      Update replay buffer $\mathcal{R}$
      Construct prioritized sampling distribution $q(\tau^g) \propto (1 - p(\tau^g|\phi))p(\tau^g)$ with higher $\mathcal{H}_q(\tau^g)$
      Sample trajectories $\tau \sim q(\tau^g|phi)$
      Update policy (θ) to max $\mathbb{E}_q[r(S, G)]$ via DDPG, HER
    end for
    Update density model (ϕ)
  end for
until convergence

Chapter 5

Evaluation

5.1 Experiments

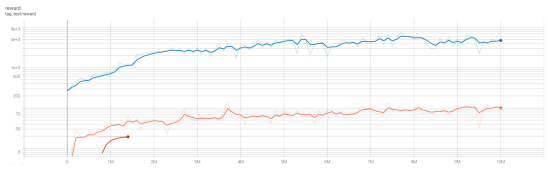
The proposed methods are tested on various simulated robotic tasks described in Chapter 3 and compared with strong baselines such as "DDPG" and "HER". we first calculate the success rate of the algorithm according to the csv output file. Then we show the time complexity of different methods and games. This paper shows that it improves performance in a shorter computation time. Furthermore, the motivation of this article is different from some other methods for multi-objective optimization problem. This paper regularizes the objective function based on entropy. For all experiments, the test environment is 8 CPUs and train the agent for 200 epochs in ideally. The test machine is the Ubuntu 20.04 system and the CPU of this machine is i7-9700K, which is 8 cores and 8 threads. However, because of the lack of the computer performance. I only use the 1 thread to train and test the model.

5.2 Performance

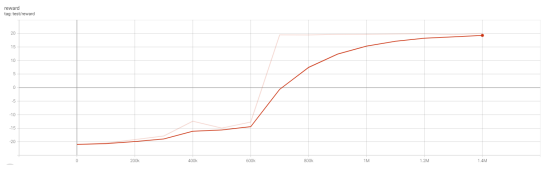
5.2.1 Previous Results

At the start of the project, the plan is to use DQN to achieve the multi-objectives. However,as the project progressed, we found that DQN was not suitable for solving multi-

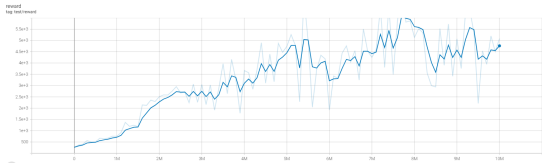
objective optimization problems. Because DQN is used to deal with discrete action problems, it is not suitable for solving multi-objective optimization problems. Multi-objective optimization problem belongs to continuous action problem. The difference of the solution algorithm leads to the difference of the training environment and results and is not comparable. But in this article, he still has some reference. Previously, the training environment for the project was Atari. But the training environment was a single-objective environment. Here, we present the previous results. Preliminary results are training the three simple Atari games, *PongNoFrameskip*, *AssaultNoFrameskip* and *BreakoutNoFrameskip* based on DQN. Because of the drawback of the equipment, each game is trained with 100 epoch and 100,000 environment step per epoch. Therefore, the total is 10 million. The results clearly demonstrate that for some simple games like *Pong*, it's rewards are not are not too different from other method and it terminates after 1.5 millions training. But the DQN doesn't perform well in some games, such as *Breakout*. According to the Rainbow, the rewards of *AssaultNoFrameskip* can reach 500, but the DQN only reaches 100. The following figures shows the results.



(a) Rewards



(b) PongNoFrameskip-v4 Rewards



(c) AssaultNoFrameskip-v4 Rewards

(d) BreakoutNoFrameskip-v4 Rewards

Figure 5.1: Previous DQN Results

5.2.2 Latest Results

Testing includes performance differences between environments and whether or not to use HER. Experiments were conducted in three robotic environments. Since the robotic arm environment is relatively simple, we use DDPG as the baseline here. In a more

difficult robotic hand environment, DDPG+HER is used as a baseline method to test performance. Compare the average success rates. Each experiment was performed using 5 random seeds, and the shaded area represents the standard deviation. The learning curves for training epochs are shown in Figure 5.2 and Figure 5.3. All experiments use 19 CPUs and train the agent with 200 epochs. After training, evaluate using the best learned policy and test it in your environment. The test results are the average success rate. Table 1 shows the comparison of performance and training time.

The mean success rate is calculate by the two rewards with two weightshu. Because in this paper, the multi-objective problem is translated to the two rewards problem as mentioned before. The two rewards are given a weight separately. The users can choose the weights because the project only output an csv file including the data about two rewards.

First we give 0.5 weight to each reward. The mean success rate is shown in Figure 5.2. As you can see from the Figure 5.2, DDPG increases the success rate as the number of sessions increases for all six environments. In general, DDPG are optimized for learning in multi-objective environments. Although the training effect was not very good in both (b) and (c) environments in Figure 5.2, there was no obvious sign of improving the success rate. This is probably due to errors in setting the two environment weights. It could also have been caused by machines during training. However, in the other four environments in Figure 5.2, we can obviously see that with the increase of the number of epochs, the average success rate of the algorithm keeps improving and gradually reaches stability. Meanwhile, the performance of the algorithm in (d), (e) and (f) in Figure 5.2 are obviously better than that in (a), (b) and (c), but this is not reasonable. This is a big deviation from the expected result. Normally, the average success rate in (a), (b) and (c) should be significantly better than (d), (e) and (f) in Figure 5.2. The main reason for this phenomenon in this project is. In the course of training the (d), (e) and (f) environment, the machine used 19 cpus at the same time. But only a single CPU is used in (a), (b) and (c). This leads to poor training results in the three (a), (b) and (c) environments.

In addition, we compare the DDPG algorithm with the DDPG algorithm combined with HER algorithm. From the Figure 5.2, we can see that the DDPG algorithm combined with HER performs significantly better than the algorithm using ONLY DDPG in all six environments. This proves the correctness of incorporating HER into this project. The use of HER can make the strategy learning of the whole DDPG algorithm more excellent. In continuous training to optimize the choice of strategies, so as to improve the average success rate. The results shows that for $FetchSlide$, $FetchPickAndPlace$ and $FetchPush$ these three environments, the success rate can reach approximate 0.45, while the success will be 0.01 higher if using DDPG+HER. For another three environments, the success rate can reach approximate 0.65, while the success will be 0.02 higher if using DDPG+HER.

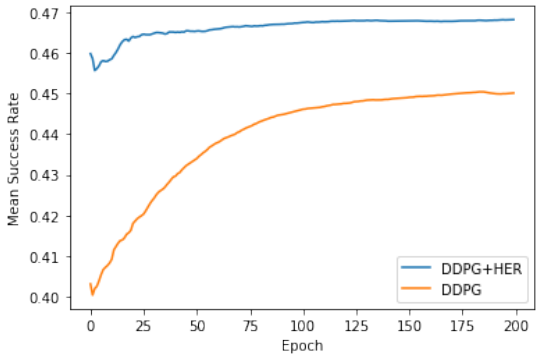
Secondly, we give 0.7 weight to one reward and 0.3 to thes other. The reason is that during the training, we found that one reward had a great influence on the final success rate. So we updated the weights and retrained. We give a greater weight to the reward, which has a greater impact on the outcome. As we can see from the Figure 5.3, the average success rate of all training environments improved after the weights were updated. This also proves that the adjustment of the weight can affect the final success rate. In addition, in the process of use, we can constantly change the weight and observe the results, so as to get an optimal training result according to their own actual situation. For multi-rewards reinforcement learning, different rewards have different effects on the results. In this project, two rewards, O and G, hold different weights respectively, which affects the final training results.

The agent improved its performance at the end of training, as shown in Table 1. You can also see in Table 1 that the training time consumes less time in different environments. This algorithm is known to be modest, especially if the memory size "N" is very large. Our experiments use an efficient implementation based on a "Tree" data structure. It can

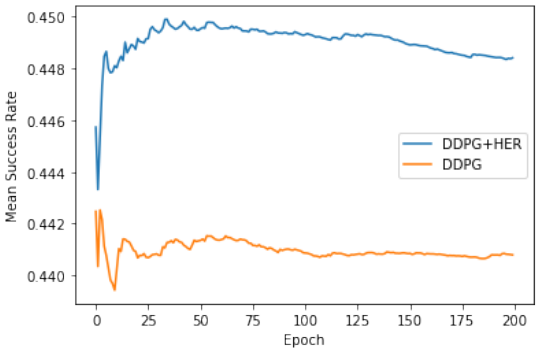
be updated and sampled relatively efficiently. More specifically, our algorithm consumes much less computation time. Table 1 shows that the *FetchPickAndPlace*, *FetchPush* and *FetchSlide* environments only need approximately six hours to train, while only need seven hours if using the DDPG+HER. This is very faster than some reinforcement learning algorithms. Normally, for the other three environments, training takes more time. This is also reflected in this project. And the use of HER will cause time consumption. Therefore, if using the algorithm proposed in this project, try to use multiple CPUS for training. This will help reduce time consumption.

Table 5.1: Training Time (hour) for all six environments

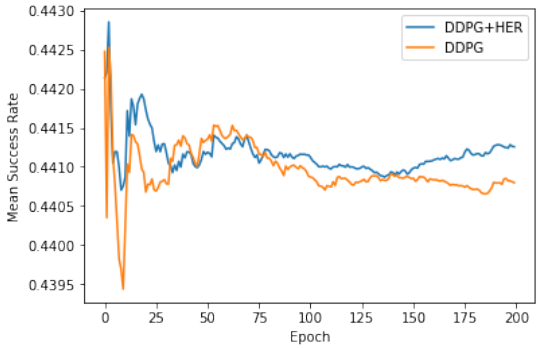
Method	**PickAndPlace**	**Push**	**Slide**
DDPG	6.17h	6.03h	5.98h
DDPG+HER	7.04h	6.77h	6.47h
Method	**Egg**	**Pen**	**Block**
DDPG	16.87h	18.06h	20.86h
DDPG+HER	22.26h	25.14h	26.51h



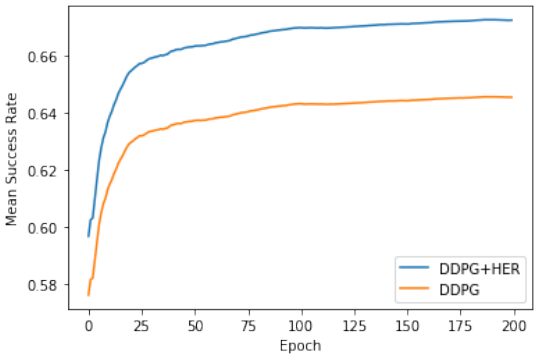
(a) FetchSlide



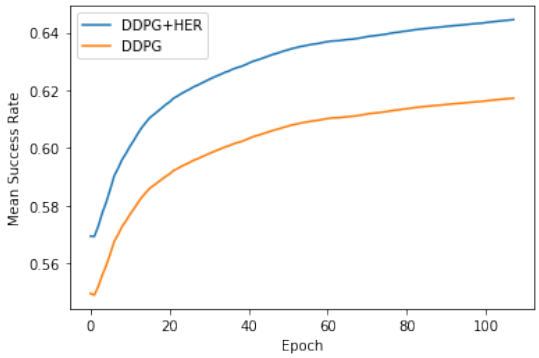
(b) FetchPickAndPlace



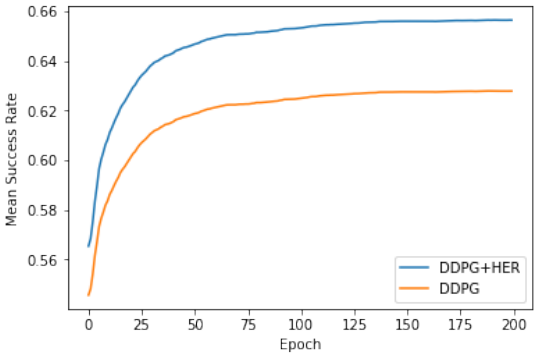
(c) FetchPush



(d) HandManipulateBlockFull

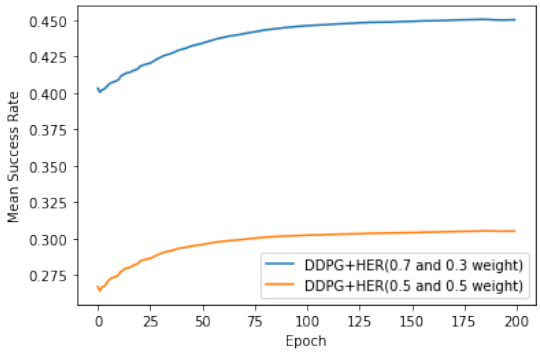


(e) HandManipulateEggFull

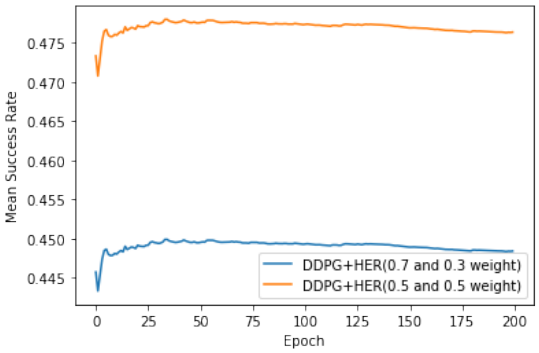


(f) HandManipulatePenRotate

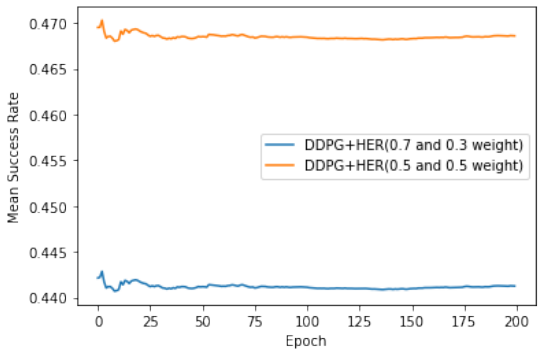
Figure 5.2: Mean success rate in all six robot environments (0.5 weight each)



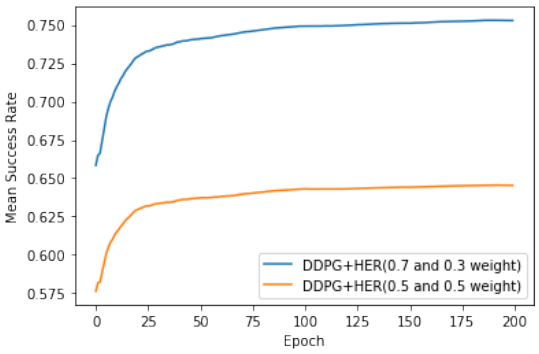
(a) FetchSlide



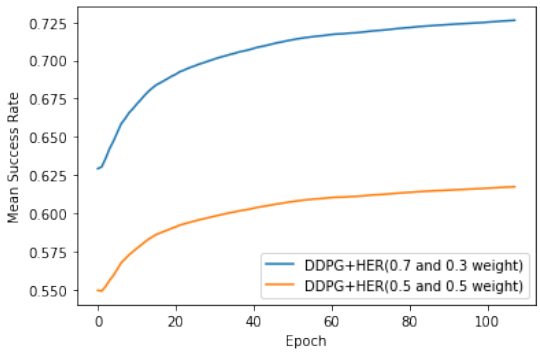
(b) FetchPickAndPlace



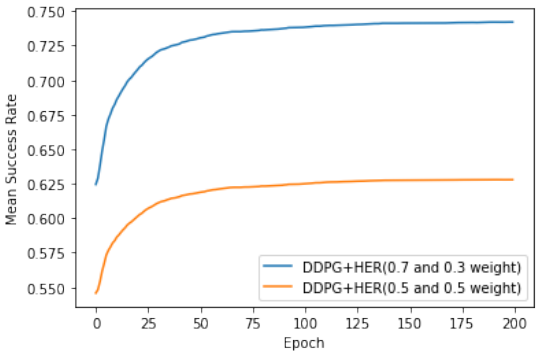
(c) FetchPush



(d) HandManipulateBlockFull



(e) HandManipulateEggFull



(f) HandManipulatePenRotate

Figure 5.3: Mean success rate in all six robot environments (0.7 and 0.3 weight)

Chapter 6

Summary and Reflections

This paper contributes well. In less than a year, we implemented reinforcement learning to solve a multi-objective optimization problem. Although the final goal is not the same as the original vision of the project, in general, this paper uses a better method than before to achieve the original goal. First, we implement multi-objective optimization based on reinforcement learning. Then, to achieve stable optimization, we derive a safe proxy objective, which is a lower bound on the original objective. Overall, our approach encourages agents to achieve diversification goals while maximizing expected returns. We evaluate the method in a multipurpose robot simulation. Experimental results show that our method improves surrogate performance and sampling efficiency while controlling computation time. More precisely, the results show that our method has excellent performance and a sample efficiency of 9 percentage points.

6.1 Project Management

At the beginning of the project, the main idea was to conduct multi-objective optimization training for Atari games through reinforcement learning framework. Meanwhile, Pytorch is used as reinforcement learning framework and DQN algorithm is used to complete the algorithm framework of this project. However, after learning related work, we found that DQN could not perform well in multi-objective optimization problems. Because DQN is better at solving discrete problems. At the same time, there is no obvious multi-objective

environment in Atari games. After deliberation and discussion, we decided to abandon the previous idea and switch to the robotics environment, a more obvious multi-objective model. And through the combination of DDPG and HER to solve the optimization problem. In the later stage of the project, the model is continuously tested and the weight allocation is improved to achieve the project objectives.

Figure 6.1 shows the timeline of the entire project. In the fall semester, due to the wrong research direction, a learning framework based on DQN and Pytorch was made, but it could only solve the target problem. In the process of improving the multi-objective problem, we found that the method based on DQN could not meet our requirements. So in the Spring Festival of this year, we decided to change the direction, abandon most of the previous framework, and integrate actor-Critic on the basis of DQN. Use DDPG as a new framework to accomplish project objectives.

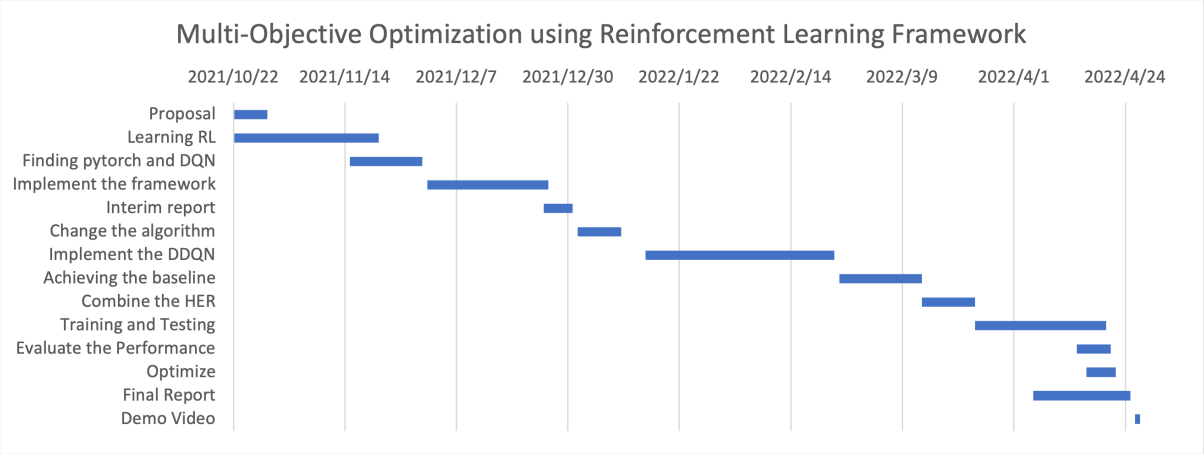


Figure 6.1: Gantt chart for the project

In the spring semester, we implemented the basic DDPG framework based on DeepMind in one month. Then I added a multi-rewards algorithm on this basis to solve the multi-objective optimization problem. In March, I found that using HER could better optimize

the choice of strategies. So it took a few days to integrate HER algorithm. Because HER algorithm can be well combined with off-policy algorithm. A lot of time was wasted due to the wrong research direction. So we ended up training the test model in less than a month. And we didn't have enough time to optimize our algorithm. Finally, about half a month before the deadline, the final report was written.

6.2 Contributions and reflections

In this project, I used DDPG as the basic framework to implement multi-objective reinforcement learning. At the same time, the network structure is implemented using AC. Because DDPG has a better ability to solve such multi-objective optimization problems and continuous actions than DQN. In my opinion, the key idea of this project is to transform multi-objective into multi-reward problem. The two reward parameters are calculated simultaneously to transform into a single reward for reinforcement learning. In addition, the training of the entire algorithm is constantly looking for a suitable strategy. Therefore, after each training, the project outputs many policy files and the optimal policy. After the introduction of HER, the entire algorithm framework tends to be more perfect for the selection of strategies, which is of great help to the fitting of training. On the whole, the project takes literature review as the idea, combines some existing frameworks on the Internet, and solves the multi-objective optimization problem in a relatively simple way.

But there are also many deficiencies in this project that need to be resolved. The first is that the management of the project cycle is not very good. The first 50% of the project cycle and the last 50% of the assumptions have a large deviation, which leads to a lot of lost time. The reason for this problem is that this aspect of multi-objective optimization, especially in reinforcement learning, was not well understood at first. This led to the initial selection of DQNs that were not suitable for solving multi-objective optimization problems. Although the problem was discovered and corrected later, and the implementation of the code and the configuration of the environment were completed as quickly as

possible, this resulted in not much time for extensive testing. At the same time, due to the wrong direction at the beginning, there is not much time to compare and analyze with other multi-objective optimization algorithms in the end. This makes the experimental results and the performance of the algorithm less convincing. Secondly, this algorithm cannot generate the chart by itself, and the user also needs to draw the csv file into a chart to see the real effect.

On the other hand, the speed of the entire project is not particularly efficient. In some environments, the training time for the experimental machine took one to two days. At the same time, the project requires strict environment configuration and strict adherence to specific versions of related libraries. It is more troublesome to configure the environment. In addition, in the final environment configuration stage, we did not implement the algorithm to train on the windows system. In other words, the project can only be trained on linux systems. In the end, although there are still many problems with this project, it is a reliable way to solve multi-objective optimization problems. This project solves a multi-objective optimization problem using reinforcement learning in a specific environment.

In the futurethere is still room for improvement in this project. First of all, in terms of environment, we hope to extend the robotics environment to most of the gym environments. Individual environment cannot reflect the performance level of the algorithm. At the same time, extending to a larger environment also helps to optimize the algorithm. On the other hand, the algorithm can also visualize the training results rather than just output a bunch of data. Although this will affect the weight allocation, it will make the training results more intuitive.

Bibliography

[1] Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Pieter Abbeel, O., and Zaremba, W. Hindsight experience replay. *Advances in neural information processing systems 30* (2017).

[2] Bahdanau, D., Brakel, P., Xu, K., Goyal, A., Lowe, R., Pineau, J., Courville, A., and Bengio, Y. An actor-critic algorithm for sequence prediction. *arXiv preprint arXiv:1607.07086* (2016).

[3] Barron, E., and Ishii, H. The bellman equation for minimizing the maximum cost. *Nonlinear Analysis: Theory, Methods & Applications 13*, 9 (1989), 1067–1090.

[4] Bressert, E. Scipy and numpy: an overview for developers.

[5] Casas, N. Deep deterministic policy gradient for urban traffic light control. *arXiv preprint arXiv:1703.09035* (2017).

[6] Dalcin, L., and Fang, Y.-L. L. Mpi4py: Status update after 12 years of development. *Computing in Science & Engineering 23*, 4 (2021), 47–54.

[7] Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation 6*, 2 (2002), 182–197.

[8] Erez, T., Tassa, Y., and Todorov, E. Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx. In *2015 IEEE international conference on robotics and automation (ICRA)* (2015), IEEE, pp. 4397–4404.

[9] Fang, M., Zhou, C., Shi, B., Gong, B., Xu, J., and Zhang, T. Dher: Hindsight experience replay for dynamic goals. In *International Conference on Learning Representations* (2018).

[10] Grondman, I., Busoniu, L., Lopes, G. A., and Babuska, R. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 42*, 6 (2012), 1291–1307.

[11] Jesus, J. C., Bottega, J. A., Cuadros, M. A., and Gamarra, D. F. Deep deterministic policy gradient for navigation of mobile robots in simulated environments. In *2019 19th International Conference on Advanced Robotics (ICAR)* (2019), IEEE, pp. 362–367.

[12] Kennedy, J., and Eberhart, R. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks* (1995), vol. 4, IEEE, pp. 1942–1948.

[13] Kim, S., Kim, I., and You, D. Multi-condition multi-objective optimization using deep reinforcement learning. *arXiv preprint arXiv:2110.05945* (2021).

[14] Konda, V., and Tsitsiklis, J. Actor-critic algorithms. *Advances in neural information processing systems 12* (1999).

[15] Lauener, J., Sliwinski, W., and CERN, G. How to design & implement a modern communication middleware based on zeromq. In *Proc of ICALEPCS* (2017), vol. 17, pp. 45–51.

[16] Li, K., Zhang, T., and Wang, R. Deep reinforcement learning for multiobjective optimization. *IEEE transactions on cybernetics 51*, 6 (2020), 3103–3114.

[17] Li, S., Wu, Y., Cui, X., Dong, H., Fang, F., and Russell, S. Robust multi-agent reinforcement learning via minimax deep deterministic policy gradient. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2019), vol. 33, pp. 4213–4220.

[18] Li, Y. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274* (2017).

[19] Liao, H., Wu, Q., and Jiang, L. Multi-objective optimization by reinforcement learning for power system dispatch and voltage stability. In *2010 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT Europe)* (2010), IEEE, pp. 1–8.

[20] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[21] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature 518*, 7540 (2015), 529–533.

[22] Murphy, K. P. A probabilistic perspective. *Text book* (2012).

[23] Nachum, O., Norouzi, M., Xu, K., and Schuurmans, D. Bridging the gap between value and policy based reinforcement learning. *Advances in neural information processing systems 30* (2017).

[24] Nguyen, H., La, H. M., and Deans, M. Hindsight experience replay with experience ranking. In *2019 Joint IEEE 9th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)* (2019), IEEE, pp. 1–6.

[25] Nguyen, N. D., Nguyen, T., and Nahavandi, S. System design perspective for human-level agents using deep reinforcement learning: A survey. *IEEE Access 5* (2017), 27091–27102.

[26] Nguyen, T. T., Nguyen, N. D., Vamplew, P., Nahavandi, S., Dazeley, R., and Lim, C. P. A multi-objective deep reinforcement learning framework. *Engineering Applications of Artificial Intelligence 96* (2020), 103915.

[27] O'Donoghue, B., Munos, R., Kavukcuoglu, K., and Mnih, V. Combining policy gradient and q-learning. *arXiv preprint arXiv:1611.01626* (2016).

[28] Pike-Burke, C. Multi-objective optimization, 2019.

[29] Qiu, C., Hu, Y., Chen, Y., and Zeng, B. Deep deterministic policy gradient (ddpg)-based energy harvesting wireless communications. *IEEE Internet of Things Journal 6*, 5 (2019), 8577–8588.

[30] Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. *arXiv preprint arXiv:1511.05952* (2015).

[31] Sutton, B., and Powell, B. P. Notation for stochastic dynamic programming (markov decision processes, approximate dynamic program-ming, reinforcement learning).

[32] Sutton, R. S., and Barto, A. G. Reinforcement learning: An introduction. *Robotica 17*, 2 (1999), 229–235.

[33] Sutton, R. S., Barto, A. G., et al. *Introduction to reinforcement learning*, vol. 135. MIT press Cambridge, 1998.

[34] Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems 12* (1999).

[35] Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems* (2012), IEEE, pp. 5026–5033.

[36] Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., Makhzani, A., Küttler, H., Agapiou, J., Schrittwieser, J., et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782* (2017).

[37] Watkins, C. J. C. H. Learning from delayed rewards.

[38] Wu, D., Dong, X., Shen, J., and Hoi, S. C. Reducing estimation bias via triplet-average deep deterministic policy gradient. *IEEE transactions on neural networks and learning systems 31*, 11 (2020), 4933–4945.

[39] Ye, D., Chen, G., Zhang, W., Chen, S., Yuan, B., Liu, B., Chen, J., Liu, Z., Qiu, F., Yu, H., et al. Towards playing full moba games with deep reinforcement learning. *Advances in Neural Information Processing Systems 33* (2020), 621–632.

[40] Zhang, Q., and Li, H. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation 11*, 6 (2007), 712–731.

[41] Zhang, Z., Wu, Z., and Wang, J. Meta-learning-based deep reinforcement learning for multiobjective optimization problems. *arXiv preprint arXiv:2105.02741* (2021).