저작자표시-비영리-변경금지 **2.0** 대한민국

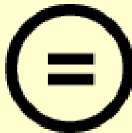
이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

공학박사학위논문

Exploratory Hybrid Search in Hierarchical Reinforcement Learning

계층 강화 학습에서의 탐험적 혼합 탐색

2020년 8월

서울대학교 대학원

전기·컴퓨터공학부

이 상 엽

Exploratory Hybrid Search in Hierarchical Reinforcement Learning

지도교수 문병로

이 논문을 공학박사 학위논문으로 제출함

2020년 5월

서울대학교 대학원

전기·컴퓨터공학부

이 상 엽

이 상 엽의 공학박사 학위논문을 인준함

2020년 6월

위 원 장	신 영 길	(인)
부위원장	문 병 로	(인)
위 원	오 일 석	(인)
위 원	정 순 철	(인)
위 원	김 용 혁	(인)

Abstract

Balancing exploitation and exploration is a great challenge in many optimization problems. Evolutionary algorithms, such as evolutionary strategies and genetic algorithms, are algorithms inspired by biological evolution. They have been used for various optimization problems, such as combinatorial optimization and continuous optimization. However, evolutionary algorithms lack fine-tuning near local optima; in other words, they lack exploitation power. This drawback can be overcome by hybridization. Hybrid genetic algorithms, or memetic algorithms, are successful examples of hybridization. Although the solution space is exponentially vast in some optimization problems, these algorithms successfully find satisfactory solutions.

In the deep learning era, the problem of exploitation and exploration has been relatively neglected. In deep reinforcement learning problems, however, balancing exploitation and exploration is more crucial than that in problems with supervision. Many environments in the real world have an exponentially wide state space that must be explored by agents. Without sufficient exploration power, agents only reveal a small portion of the state space and end up with seeking only instant rewards.

In this thesis, a hybridization method is proposed which contains both gradient-based policy optimization with strong exploitation power and evolutionary policy optimization with strong exploration power. First, the gradient-based policy optimization and evolutionary policy optimization are analyzed in various environments. The results demonstrate that evolutionary policy optimization is robust for sparse rewards but weak for instant rewards, whereas gradient-based policy optimization is effective for instant rewards but weak for sparse rewards. This difference between the two optimizations reveals the potential of hybridization in policy optimization. Then, a

hybrid search is suggested in the framework of hierarchical reinforcement learning. The results demonstrate that the hybrid search finds an effective agent for complex environments with sparse rewards thanks to its balanced exploitation and exploration.

Keywords : Deep reinforcement learning, Evolutionary computation, hierarchical reinforcement learning, Neuroevolution

Student Number : 2013-20845

Contents

Contents . . . . **iv**

List of Figures . . . . **v**

List of Tables . . . . **vi**

I. Introduction **1**

II. Background **6**

2.1 Evolutionary Computations 6

2.1.1 Hybrid Genetic Algorithm 7

2.1.2 Evolutionary Strategy 9

2.2 Hybrid Genetic Algorithm Example: Brick Layout Problem 10

2.2.1 Problem Statement 11

2.2.2 Hybrid Genetic Algorithm 11

2.2.3 Experimental Results 14

2.2.4 Discussion 15

2.3 Reinforcement Learning 16

2.3.1 Policy Optimization 19

2.3.2 Proximal Policy Optimization 21

2.4 Neuroevolution for Reinforcement Learning 23

2.5 Hierarchical Reinforcement Learning 25

2.5.1 Option-based HRL 26

2.5.2 Goal-based HRL 27

2.5.3 Exploitation versus Exploration 27

III. Understanding Features of Evolutionary Policy Optimizations . . . 29
3.1 Experimental Setup 31
3.2 Feature Analysis 32
3.2.1 Convolution Filter Inspection 32
3.2.2 Saliency Map 36
3.3 Discussion 40
3.3.1 Behavioral Characteristics 40
3.3.2 ES Agent without Inputs 42

IV. Hybrid Search for Hierarchical Reinforcement Learning 44
4.1 Method 45
4.2 Experimental Setup 47
4.2.1 Environment 47
4.2.2 Network Architectures 50
4.2.3 Training 50
4.3 Results 51
4.3.1 Comparison 51
4.3.2 Experimental Results 53
4.3.3 Behavior of Low-Level Policy 54
4.4 Conclusion 55

V. Conclusion 56
5.1 Summary 56
5.2 Future Work 57

Bibliography 58

List of Figures

Figure 1. Exploitation and exploration 2

Figure 2. Exploitation and exploration in reinforcement learning 3

Figure 3. Crossover and mutation in genetic algorithms 8

Figure 4. Assembling a 3D object with the LEGO® 10

Figure 5. Example of MDP 16

Figure 6. Basic framework of hierarchical reinforcement learning 25

Figure 7. First convolutional layer filter visualizations 33

Figure 8. Saliency maps for different actions and states trained by A2C . 37

Figure 9. Saliency maps for evolutionary strategy 39

Figure 10. KL-divergence plots of agent trained by ES 40

Figure 11. Four environments in Atari 2600 41

Figure 12. Direction-masking network 46

Figure 13. AntMaze and AntGather environments 47

Figure 14. Wall readings for Maze environment 48

Figure 15. Cycle movement for four directions 54

List of Tables

Table 1. Comparison of different mutations in the brick layout problem . 14
Table 2. Convolutional network architecture 30
Table 3. Performance of agents trained by each algorithm 32
Table 4. The number of filters that passed Kolmogorov-Smirnov test . . . 34
Table 5. The number of filter pairs that passed two sample Kolmogorov-Smirnov test 35
Table 6. The number of feature map pairs that passed two sample Kolmogorov-Smirnov test 35
Table 7. Performance of agents trained by evolutionary strategy (ES) without inputs 43
Table 8. Results of hybrid hierarchical reinforcement learning 52

Chapter 1

Introduction

Machine learning has achieved remarkable success with developments in hardware and various algorithms. One of the most successful approaches is deep reinforcement learning, and a number of different approaches have been proposed for reinforcement learning with neural networks. Deep Q-learning [31], the policy gradient method [30] and trust region method [38] are examples of successful algorithms.

However, there has always been a significant problem that involves balancing the algorithm's exploitation and exploration in reinforcement learning. Exploitation and exploration are two criteria in searching for a solution in the problem space. Exploitation consists of probing a limited region, usually neighborhoods or promising area, of the given search space. It is also called a local search of the problem space. Exploration, in contrast, consists of probing an even wider region of the problem space. It tends to find a solution from a region not yet discovered; however, the region is not sufficiently refined. The concept of the exploitation exploration trade-off is used in optimization. Figure 1 presents a visual example to illustrate exploitation and exploration. In this figure, the curve represents the graph of function $f(x)$, while circle and square points represent inputs. The red circle points are moved toward neighboring points that have smaller values. By repeating these operations, a point is moved to a region in which all neighbors have larger function values, that is, a local optimum. This criterion refers to exploitation of optimization. However, as displayed in Figure 1, there may exist promising areas other than the local optimum. Searching with a local search never leads to the discovery of the areas, however, because it only

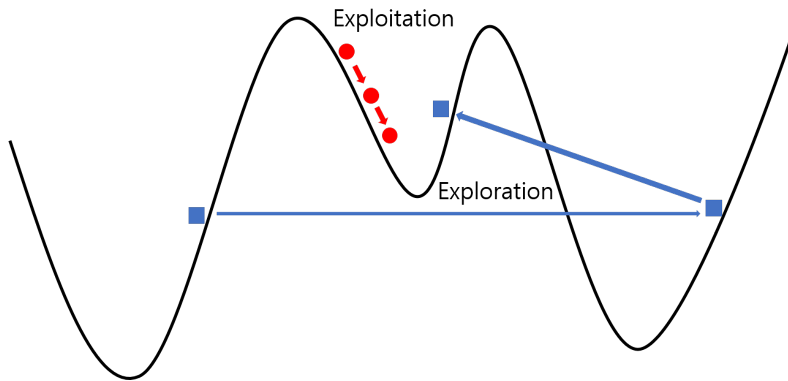
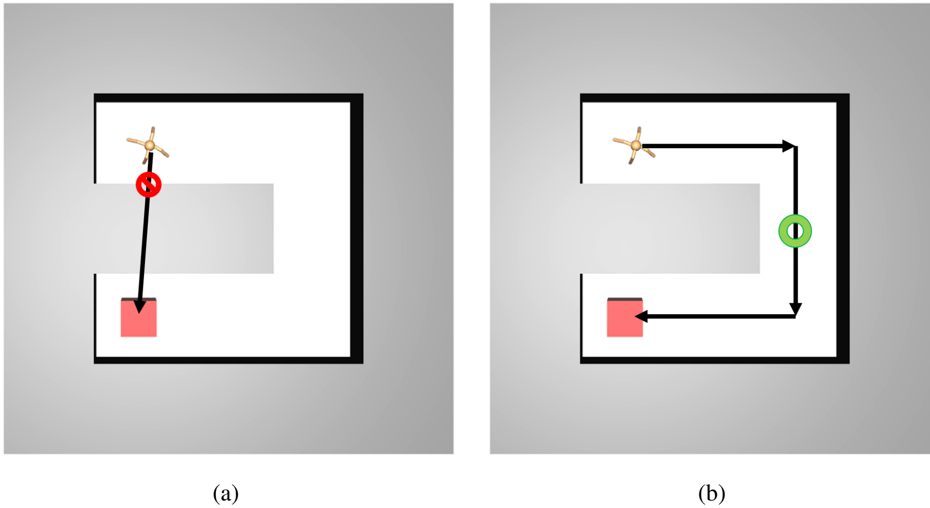


Figure 1: Exploitation and exploration

moves the probe to a better neighboring area, called hill-climbing. Blue square points, in contrast, are moved to an entirely new area without considering their original location. This criterion refers to exploration of optimization. Exploration may cause a point to escape from the well of the function, leading to areas with better optima. However, the points searched by exploration are rarely close to local optima, as they are not yet sufficiently tuned. There is also no guarantee that repeating exploration searches moves points to local optima. Therefore, to find the global optimum or a satisfactory local optimum, balancing exploitation and exploration is critical.

Many metaheuristic algorithms are challenged by the problem of balancing exploitation and exploration. One method to overcome this problem involves mixing, or hybridizing, two optimization algorithms that have different strategies. In the example in Figure 1, if an algorithm performs both exploration and exploitation, the global optimum can be found in a reasonable time. A hybrid genetic algorithm (GA), or a memetic algorithm [32], is an example of a hybrid optimization algorithm. Many hybrid GAs successfully produce good solutions for both benchmark problems [5] and real-world problems [26].



(a) (b)

Figure 2: Exploitation and exploration in reinforcement learning

In reinforcement learning, exploitation and exploration are used to describe a strategy for changing how an agent reacts to the given environment, namely the state space. If all states of the environment are not known in advance, an agent must determine the states from experience. One strategy is to use an agent who acts with a small difference from the previous agent to maximize the rewards to exploit the state space. An agent that follows immediate rewards can be found by the exploitation strategy, but will attempt to avoid the uncertainty of unseen states. However, in many environments, there exist sparse rewards that cannot be found by slight changes in the current policy. Instead, they can only be discovered through new policies that differ significantly from the current policies. Unseen states can thus only be visited with an entirely new policy. Figure 2 presents an example. Suppose that an agent is rewarded when it approaches the goal (red square). Its best choice is to move the red arrow in the left figure, which is achieved by exploitation. However, there exists a wall that prevents the agent from reaching the goal. To reach the goal, the agent must take a detour, such as the green arrow in the right figure. The agent with only exploitation

never finds this detour; it may find the detour with exploration. Many experience-based (i.e., model-free) reinforcement learning algorithms are challenged by the exploration problem. Different types of methods are adopted, such as ε-greedy [48] and rewarding for uncertainty [12]. However, especially in the deep learning era, the majority of algorithms tend to use gradient-based optimization. Gradient descent is an algorithm to find a local minimum when a sufficiently small step size is given. Therefore, when properly used, gradient descent has strong exploitation power; however, it lacks exploration power. Gradient-based optimization for reinforcement learning usually has stronger exploitation power than exploration power. Occasionally, improving exploitation power improves the agent [40], especially for immediate rewards. However, exploration becomes crucial in environments with sparse rewards.

Recent studies have suggested that some evolutionary approaches rival gradient-based optimization algorithms [8, 44]. GAs and evolutionary strategies display better performance in many reinforcement tasks. Some studies have also suggested that even a simple random search performs better than other policy optimization algorithms in some tasks. Interestingly, while gradient-based policy optimization algorithms demonstrate excellent performance in some tasks, they demonstrate poor performance in other tasks for which evolutionary algorithms perform well. These tasks are notorious for their reward sparsity [25]. This suggests that gradient-based optimizations are not effective for sparse rewards, whereas evolutionary algorithms, which are known for their exploration ability, are more effective in these environments.

The goal of this thesis is to propose an effective hybridization method for balancing exploitation and exploration in reinforcement learning. In Chapter 2, background knowledge of evolutionary algorithms and reinforcement learning is introduced. A real-world application of hybrid search is also presented to evaluate the importance

of domain knowledge in balancing exploitation and exploration. Then, in Chapter 3, experiments are proposed to examine the effects of the characteristics of optimization on the exploitation and exploration of environments. Due to the complexity of reinforcement learning problems, methods for visual inspection and supplementary experiments are suggested. Finally, in Chapter 4, a hybrid hierarchical reinforcement learning (HRL) algorithm is suggested that is effective in environments that require a strategy with both strong exploitation and exploration to obtain sufficient rewards.

Chapter 2

Background

2.1 Evolutionary Computations

Evolutionary computation (EC) algorithms are a family of optimization algorithms inspired by the evolution of life. All individuals produce offspring that inherit genetic information from their parents. Some individuals reproduce on their own, while other individuals mate with others to produce offspring. While producing offspring, some genes may undergo irregular changes due to internal or external causes, called mutation. With these mechanisms, all living beings compete with each other according to the law of the survival of the fittest, which is a fundamental law of evolution.

EC methods perform processes similar to those of biological evolution to optimize their solutions. There are many types of EC algorithms, but all have two common aspects, as described below.

- Population-based: EC algorithms maintain multiple (at least two) solutions and manipulate them for optimization.
- Stochastic optimization: EC algorithms produce offspring in a stochastic way; thus, updates of solutions become stochastic.

Some algorithms, such as the ant colony optimization and particle swarm optimization, do not mimic evolution, but rather, other natural phenomena. However, these algorithms are outside the scope of this thesis, whose focus is evolution-inspired algorithms.

2.1.1 Hybrid Genetic Algorithm

Algorithm 1 Hybrid GA

In: number of chromosome n
In: Operators, $select, crossover, mutate$
In: Local optimization opt
Initialize the population p
repeat$parents \leftarrow select(p)$ $children \leftarrow crossover(parents)$ $children \leftarrow mutate(children)$ $p \leftarrow opt(children)$
until stop condition
return the best in p

GAs are algorithms that are remarkably similar to evolution in nature. These algorithms use some evolutionary processes, called operators, to imitate the mechanisms of biological evolution. GAs select and use some of these operators for various purposes. Basic operators include selection, crossover, and mutation. The selection operator is used to select individuals from a population for reproduction or survival, thus generating the selection pressure of the algorithms. If the selection pressure is too high, the algorithm may experience premature convergence, resulting in a suboptimal solution. However, if the selection pressure is too low, progress is slower than necessary. Using an appropriate selection pressure is critical for balancing exploration and exploitation. Crossover is inspired by the crossover of chromosomes, or genetic material in nature. This operator takes two or more solutions and combines them to form new solutions. The purpose of crossover is to create a new solution while maintaining the good traits, called schema, of parents. In terms of the problem space, crossover usually limits the search space. For example, geometric crossover always produces offspring on the line segment defined between two parents. Mutation is a method that changes genes in a stochastic way, and it is the component that determines the exploration power of algorithms. Sufficient mutation power is essential for maintaining the genetic diversity of a population. Mutation should allow an algorithm to avoid local

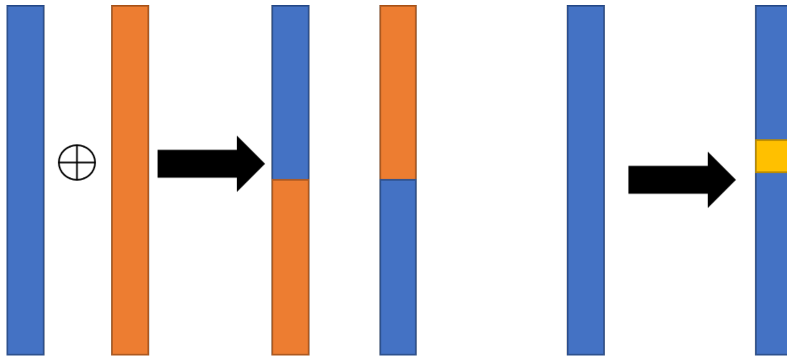


Figure 3: Crossover and mutation in genetic algorithms

minima by preventing individuals in the population from becoming too similar. The performance of GAs is determined by the representation of solutions and operator selections. Because there is no panacea for all problems according to the No Free Lunch theorem [52], it is crucial to select the most appropriate representations and operators.

Although GAs have proven to be a versatile approach for global optimization, they do not perform well in some situations. Most GAs generate new solutions in a stochastic way and are blind to the locality of the solutions. In terms of the problem space search, GA has weak exploitation power. Therefore, various methods of hybridization have been proposed; that combine both GA and other exploitation algorithms. One of the most common forms of hybrid GAs involves incorporating local optimization in the GA loop [15], called memetic algorithms. Algorithm 1 describes the basic framework of this type of hybrid GA. After performing mutation to offspring, a local optimization algorithm optimizes the offspring into a local optimum [36]. With two algorithms, one effective in exploitation and the other effective in exploration, the memetic algorithm performs well in various optimization tasks.

Algorithm 2 Simple (1+Λ) ES

In: number of children λ
In: Sample deviation σ
Initialize μ
repeat
  for $i \leftarrow 1$ to λ **do**
    Sample $\varepsilon_i \sim N(\mu, \sigma)$
    $F_i \leftarrow$ fitness(ε_i)
  end for
  Normalize F_i
  $\mu \leftarrow \sum_i^n F_i \varepsilon_i$
until stop condition
return μ

2.1.2 Evolutionary Strategy

The evolutionary strategy (ES) is another EC algorithm, which has a different mechanism from that of the GA. The ES is also a population-based algorithm that maintains multiple solutions at the same time. However, unlike the GA, it does not directly sample offspring from parents with genetic operators. The ES usually encodes parents into parameters of a distribution, where child solutions are sampled. The ES is often represented as $(\frac{\mu}{\sigma}+,\lambda)-ES$, which signifies that the algorithm maintains μ individuals, selects σ parents, and generates λ children. The plus (+) sign indicates that a new population is generated by both the original and generated solutions, while the comma (,) indicates that a new population is generated only by new solutions. Algorithm 2 describes the basic framework of simple Gaussian $(1+\lambda)-ES$ with fixed variance. It starts with one parent that has the mean of the distribution. Then, it samples λ solutions from the distribution and evaluates their fitness. Using their fitness, μ is updated to a new center, which may generate fitter samples. The ES does not identify a single solution with good fitness; instead, it finds a distribution with better expected fitness. This principle is similar to the concept of the search gra-

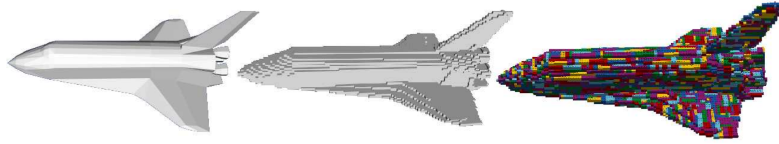


Figure 4: Assembling a 3D object with the LEGO®

dient [4], which solves the gradient of distribution by sampling; however, it does not use the gradient. Although there are many promising types of ESs, neuroevolution is specifically examined in Chapter 2.4.

2.2 Hybrid Genetic Algorithm Example: Brick Layout Problem

In this section, A real-world problem, LEGO® brick layout problem, is suggested that can be solved by a hybrid GA to elucidate the balancing exploration and exploitation in optimization. LEGO® is a line of brick-shaped toys manufactured by the LEGO® Group, and consists of bricks of a regular size and shape. The bricks can be attached to one another to assemble three-dimensional objects. Determining a way to assemble a 3D object with the LEGO® bricks can be divided into two steps. The first step is to transform a given three-dimensional object into a voxelized model. The second step, called the brick layout problem, involves assembling a voxelized object with real bricks, which is the focus of this section.

Some studies have formulated this problem as a combinatorial optimization problem [17, 35]. However, this optimization is nontrivial. The brick layout problem has an exponentially large solution space and also involves balancing exploitation and exploration.

2.2.1 Problem Statement

The brick layout problem is formulated as a combinatorial optimization problem. The input of the problem is the voxelized model that is represented as a 3D binary array, A solution to the problem corresponds to a brick layout, and the problem is to minimize the penalty function. The penalty function is defined as the number of connected components formed by the layout, and in case of a tie, the number of used bricks is compared. The first goal is to increase the connectivity, and the second goal is to increase the efficiency of the layout. The sizes of regular bricks are $1 \times n$ and $2 \times n$, where n is either 1, 2, 4, 6, or 8. A brick could be rotated, but cannot be placed diagonally. The layout is regarded as a graph to calculate the number of connected components.

2.2.2 Hybrid Genetic Algorithm

Local Optimization

A domain-based greedy heuristic algorithm is used for local optimization. For each layer, it first chooses a voxel and places a brick on that voxel in a greedy manner. All types of bricks and all feasible arrangements are considered, and the one with the largest score is selected. The algorithm repeats this step for every voxel in a layer, and repeat the entire process for each layer. The score is defined in such a way that maximizing the score of the arrangements may lead to minimizing the penalty of the entire layout. The score is a weighted sum of three factors, and the factors used are as follows.

- **Cover factor** evaluates the number of covered bricks and the score is doubled when bricks are perpendicular.

- **Size factor** encourages the algorithm to use a larger brick providing more chance of connections to the bricks in the above and below layers.

- **Isolation factor** is to minimize the number of isolated voxels.

After a significant number of experiments, the weights of the cover, size, and isolation factors were 10, 1, and 20, respectively.

Hybrid Genetic Algorithm

Algorithm 3 Genetic algorithm

Initialize population
repeat
  Select two parents $parent1, parent2$
  $offspring \leftarrow Crossover(parent1, parent2)$
  $Mutation(offspring)$
  $Repair(offspring)$
  Replace one chromosome with $offspring$
until stop condition

A hybrid genetic algorithm is proposed that consists of merge-split model and boundary split mutation. Merge-split model is to merging bricks into a larger one, or to split a brick into smaller ones This process is used to modify the solutions. The merge operator merges two or more bricks into one big brick if possible, and the split operator split a single brick into several 1×1 bricks. With a brick layout, it is possible to create several new solutions by splitting some bricks and merging them again with various orders and various combinations. . The basic framework of the GA is depicted in Algorithm 3. The hybrid GA is implemented with a 3000-generation steady-state method with the following operators:

- **Population**: 128 chromosomes.

- **Selection**: a rank based roulette-wheel-selection.

- **Crossover**: one-line crossover [22] and RectCrossover [35, 43].
- **Mutation**: boundary split mutation.
- **Repair and Optimization**: greedy heuristic in local optimization.

Boundary split mutation is the core concept in this algorithm that uses domain knowledge of brick layout problem. Instead of splitting the bricks blindly, it is possible to guide the operator to split only the bricks, which require modification. One way is to split blocks near the boundary of the connected components. If a solution is not connected and there exists more than one connected component, the space that divides the bricks into multiple connected components is defined as the boundary of the components. To connect the divided parts, adjacent bricks that are from the different components need to be merged into a single brick. Since these set of bricks resides near the boundary, choosing the bricks to split from this area might be promising to improve connectivity.

Algorithm 4 Largest boundary split

```
 1: function LargestBoundarySplit(layer L, rate r)
 2:     B ← set of all pairs of different connected components in L
 3:     Bricks ← emptylist
 4:     for all (b1, b2) in B do
 5:         S ← Bricks adjacent to the boundary between b1 and b2
 6:         Add the largest brick in S to Bricks
 7:     end for
 8:     Choose bricks in Bricks at rate r
 9:     Split the chosen bricks
10: end function
```

With this idea, Four different mutation algorithm were developed, namely largest boundary split(LB), random boundary split(RB), largest boundary with neighbor split(LBN), and random boundary with neighbor split(RBN). Algorithm 4 illustrates the mechanism of the largest boundary split. It gathers the largest bricks from the boundaries

Table 1: Comparison of different mutations in the brick layout problem

Model	Rand	LB	RB	LBN	RBN
lamp	1.00 1533.5	1.00 1516.7	1.00 1516.5	1.00 1523.8	**1.00** **1513.9**
dragon	1.07 1733.7	1.00 1741.8	**1.00** **1726.3**	1.00 1737.7	1.00 1733.4
lucy	1.27 800.3	1.06 802.2	**1.00** **798.37**	1.00 818.4	1.00 810.0
airboat	**107.26** **2517.1**	123.27 2627.9	118.73 2584.8	152.77 2639.0	150.13 2623.4
shuttle	102.27 2322.8	79.00 2413.6	**51.30** **2336.7**	138.10 2422.0	130.07 2400.3

and splits some of them with a given probability. Random boundary split works in a similar way but gathers random bricks from the boundaries instead of the largest one.

2.2.3 Experimental Results

The dataset consists of 17 different voxelized models. Five of them (*armadillo*, *bunny*, *dragon*, *happy*, and *lucy*) are from the Stanford 3D scanning repository[1] and the rest are from data files maintained by John Burkardt[2].

Interestingly, different crossovers and mutations demonstrate different performance. RectCrossover outperformed one-line crossover in general. In the case of the brick layout problem, one-line crossover, which includes zigzag random walks, is suitable for perturbing a solution, but it is vulnerable to loss of a schema. In contrast, RectCrossover will preserve the solution compared to the one-line crossover, because it is less likely that straight lines of the rectangle cross huge bricks. The excellence of RectCrossover over one-line crossover reveals that it is much more essential to

[1] `http://graphics.stanford.edu/data/3Dscanrep/`
[2] `http://people.sc.fsu.edu/~jburkardt/data/obj/obj.html`

preserve useful schemata for problem space search, which in turn better leads the creation of new attractive schemata.

Five types of mutations are also compared. One selects bricks to split randomly, and the others use LB, RB, LBN, and RBN mutation operators, respectively. Table 1 presents the result of the experiments. The upper denotes the number of connected components, and the lower indicates the number of bricks. RandomBoundarySplit showed the best performance over other mutations. Since other mutation operators choose bricks to split in a deterministic policy, these operators tend to split similar bricks, which in turn generate solutions similar to their parents. It is interesting that splitting the neighbors of boundary bricks did never improve the performance. Instead, it dropped the fitness even worse than the random mutation. This result points out that it is enough to split only boundary bricks to connect the components. In fact, this result is consistent with the result of the experiment on crossover operators. Splitting neighbors of boundary bricks will perturb the solution even more than LB or RB, end up destroying much of proper schema.

With the best operators of GA, RectCrossover, and RB mutation, and the best parameters for all models, overall experiments were conducted. For some models which are hard to assemble, an algorithm that increases the thickness of the voxel structure was adopted. After careful selection of operators, the hybrid GA managed to assemble all voxelized models into single connected components.

2.2.4 Discussion

The hybrid algorithm of both greedy-based heuristic and a genetic algorithm could assemble even the hardest models into one component without using an excessive number of bricks nor expensive time resources. The optimization for the brick layout problem shows that balancing exploitation-exploration in a problem needs a deep

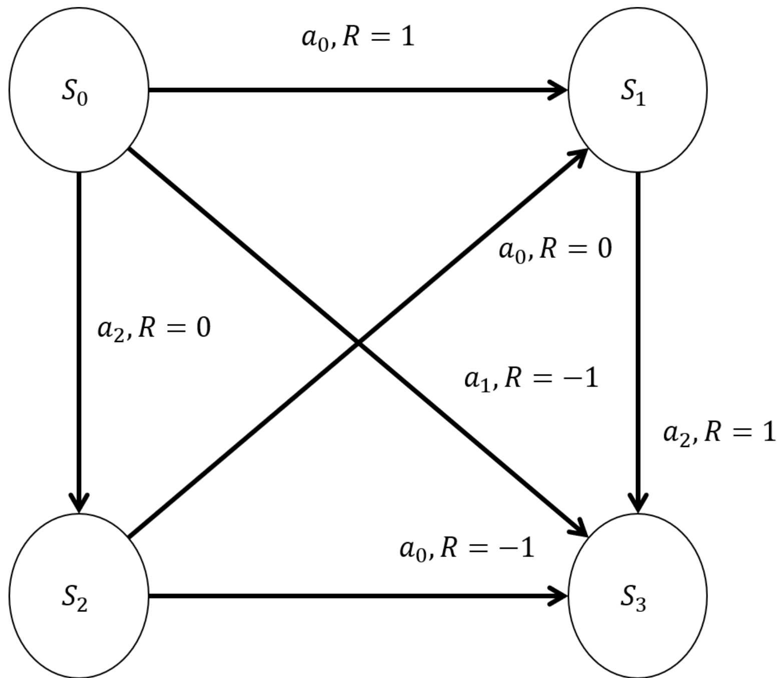


Figure 5: Example of MDP

understanding of domain knowledge. In solving the brick layout problem, domain-specific operators and local optimization work critical parts.

2.3 Reinforcement Learning

Reinforcement learning is an area of machine learning that focuses on developing an autonomous agent that takes actions in a given environment to maximize cumulative rewards. Unlike supervised learning, input-output data pairs are not available, and the agent attempts to learn the best strategy from information from the environment or experience. Reinforcement learning can be applied to many difficult real-world problems, such as gaming, robot controls, and self-driving vehicles. In reinforcement learning, a given environment is modeled as a mathematical framework called a Markov decision process (MDP).

Some terminology and notation in MDP are introduced as follows. A state s in a state space S is defined as a variable that captures all relevant information from both the present and past. If a state s_t is given at time t, past information is not required because it is included in s_t. The action space A is the set of all possible actions. We then have the state transition probability matrix $P_a(s,s')$ as the probability of the next state s after taking action a in the current state s. The reward function $R_a(s)$ denotes the expected reward of taking action a in state s. The discount factor $\gamma \in [0,1]$ determines the present value of future rewards. It is adopted because delayed rewards must be underestimated, considering the uncertainty. The MDP is defined by the following five parameters, S, A, P, R, and γ. Figure 5 presents a graphical example of the MDP. In an MDP, a policy $\pi(a|s)$ is defined as the probability of taking an action a in a given state s. The policy π generates a trajectory τ, which is a sequence of states and actions $(s_0, a_0, s_1, a_1, ...)$. Trajectory τ follows the distribution $p(\tau)$:

$$p(\tau) = p_0(s_0) \prod_{t=1}^{T-1} P_{a_t}(s_t, s_{t+1}) \pi(a_t|s_t)) \tag{2.1}$$

A partial trajectory from time t is denoted tau_t. Given trajectory τ we can calculate the cumulative reward over it. However, we generally use discounted cumulative rewards, or returns G, instead of original cumulative rewards:

$$G(\tau_t) = \sum_{k=0}^{T-1} \gamma^{t+k} R_{a_{t+k}}(s_{t+k}, s_{t+k+1}) \tag{2.2}$$

With policy π and trajectory τ_t, the value function $V_\pi(s)$ is defined as the expected return in state s given policy π:

$$V_\pi(s_t) = \mathbb{E}_{\tau_t \sim p(\tau_t)}[G(\tau_t)|s_t] \tag{2.3}$$

Similarly, the action value function $Q_\pi(s,a)$ is defined as the expected return of taking action a in state s and following π:

$$Q_\pi(s_t,a_t) = \mathbb{E}_{\tau_t \sim p(\tau)}[G(\tau_t)|s_t,a_t] \tag{2.4}$$

The goal of reinforcement learning is to predict the value function $V_\pi(s)$ or to find the policy $\pi(a|s)$ that maximizes the expected return $E[G]$. There are two types of reinforcement learning algorithms that differ based on the knowledge about the given MDP; model-based reinforcement learning and model-free reinforcement learning. In model-based reinforcement learning, all parameters of the MDP, that is $\langle S,A,P,R,\gamma \rangle$, are revealed. Because all information of the MDP is known, model-based reinforcement learning can be solved with the Bellman equation as follows:

$$V_\pi(s_t) = \mathbb{E}_{a_t \sim \pi, s_{t+1}\ P}[r(s,a_t) + \gamma V_\pi(s_{t+1})] \tag{2.5}$$

Some dynamic programming algorithms, such as value iteration, Q-value iteration, and policy iteration, use Bellman equation to determine the optimal policy π^* of the MDP. In contrast, in model-free reinforcement learning, the parameters of the MDP are unknown. Estimating the value function can only be achieved from experience. The focus of this thesis is on model-free reinforcement learning because it is difficult to define a proper MDP for many problems, and it is often necessary to learn from experience.

Reinforcement learning algorithms can also be categorized depending on the use of the value function. Some algorithms, called value-based reinforcement learning algorithms, estimate the value function to improve the policy. These algorithms usually estimate the action-value function $Q(s,a)$ and update the policy, which takes action with a maximum $Q(s,a)$ using greedy-based methods. Other algorithms, called policy

optimization algorithms, do not estimate the value function but optimize the policy directly. Value-based reinforcement learning is outside the scope of this thesis, whose focus is on various policy optimizations.

2.3.1 Policy Optimization

A policy optimization method is a class of reinforcement learning methods that do not maintain value models but directly search for the optimal policy. Let a policy be parameterized by θ and denoted $\pi(a|s;\theta)$. Policy search methods search for the optimal parameter θ^* to maximize the expectation of returns:

$$\theta^* = \underset{\theta}{\operatorname{argmax}} J(\theta) = \mathbb{E}_{\tau\sim p_\theta(\tau)}[G(\tau)] \tag{2.6}$$

The simplest approach for policy optimization is the policy gradient method [46]. The policy gradient method is a technique for deriving the gradient of the expected return with respect to the model parameter θ, called the policy gradient, for gradient ascent optimization. The policy gradient with respect to θ can be expressed as follows:

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{\tau\sim p_\theta(\tau)}[G(\tau)] \tag{2.7}$$

$$= \nabla_\theta \int_\tau p_\theta G(\tau) d\tau \tag{2.8}$$

$$= \int_\tau \nabla_\theta p_\theta G(\tau) d\tau \tag{2.9}$$

Using log derivative properties, $\nabla_\theta p_\theta(\tau)$ can be represented as follows:

$$\nabla_\theta p_\theta(\tau) = \frac{p_\theta(\tau)}{p_\theta(\tau)} \nabla_\theta p_\theta(\tau) \tag{2.10}$$

$$= p_\theta(\tau) \nabla_\theta log\, p_\theta(\tau) \tag{2.11}$$

The log function changes products in $p_\theta(\tau)$ into a summation as follows:

$$\nabla_\theta \, log \, p_\theta(\tau) = \nabla_\theta \, log(p_0(s_0) \prod_{t=1}^{T-1} P_{a_t}(s_t, s_{t+1}) \pi_\theta(a_t|s_t)) \tag{2.12}$$

$$= \nabla_\theta \sum_{t=1}^{T-1} (log \, \pi_\theta(a_t|s_t)) \tag{2.13}$$

$$= \sum_{t=1}^{T-1} \nabla_\theta log \, \pi_\theta(a_t|s_t) \tag{2.14}$$

The policy gradient can now be represented as follows:

$$\nabla_\theta J(\theta) = \int_\tau p_\theta(\tau) (\sum_{t=1}^{T-1} \nabla_\theta log \, \pi_\theta(a_t|s_t)) G(\tau) d\tau \tag{2.15}$$

$$= \mathbb{E}_{\tau \sim p_\theta(\tau)} [(\sum_{t=1}^{T-1} \nabla_\theta log \, \pi_\theta(a_t|s_t)) G(\tau)] \tag{2.16}$$

$$= \mathbb{E}_{\pi_\theta} [(\sum_{t=1}^{T-1} \nabla_\theta log \, \pi_\theta(a_t|s_t)) G(\tau)] \tag{2.17}$$

The expectation over trajectories in (2.17) can be approximated with the Monte-Carlo method with empirical trajectories. This Monte-Carlo policy gradient method is called REINFORCE. Equation (2.17) implies that as the reward increases, gradient ascent updates the parameters so that the probability of the action increases.

The Monte Carlo policy gradient relies on empirical returns, which have a high variance. To reduce the variance in empirical returns, we use a method called the actor-critic algorithm [23]. Assume that $G(\tau_t)$ can be represented as the action-value function $Q_{\pi_\theta}(s_t, a_t)$. In the actor-critic algorithm, the action-value function is approximated with $Q(s, a|w)$ where w represents the model parameters. This can reduce the high variance of empirical return.

Another way to reduce the variance is by adopting a baseline $b(s)$. Baseline $b(s)$ is a function that is independent of the policy π_θ; therefore, its expectation becomes

zero:

$$\mathbb{E}[(\sum_{t=1}^{T-1}\nabla_\theta log\,\pi_\theta(a_t|s_t))b(s_t)] = 0 \tag{2.18}$$

Subtracting the baseline from $Q_{\pi_\theta}(s_t, a_t)$ in the policy gradient does not change the expectation:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[(\sum_{t=1}^{T-1}\nabla_\theta log\,\pi_\theta(a_t|s_t))(Q_{\pi_\theta}(s_t, a_t) - b(s_t))] \tag{2.19}$$

One popular baseline function is the state value function $V_{\pi_\theta}(s_t)$. The advantage function $A_{\pi_\theta}(s_t, a_t)$ can be defined as follows:

$$A_{\pi_\theta}(s_t, a_t) = Q_{\pi_\theta}(s_t, a_t) - V_{\pi_\theta}(s_t) \tag{2.20}$$

Using the advantage function instead of G_τ retains the expectation in the policy gradient while reducing the variance.

An algorithm that uses Q or V as an advantage function is called the advantage actor-critic algorithm. The advantage actor-critic algorithm is implemented in several ways, including asynchronous advantage actor-critic (A3C) [30] and synchronous advantage actor-critic (A2C).

2.3.2 Proximal Policy Optimization

The policy gradient updates parameters with the gradient ascent method with expected returns. The gradient ascent method takes small steps in the parameter space, aiming to achieve small changes in the corresponding policy. However, determining the exact size of a small step is not a trivial problem. Because the policy space is not a

simple Euclidean space, small steps in the parameter space may lead to large changes in the policy space.

Trust region policy optimization (TRPO) offers a safe policy update algorithm that guarantees monotonic policy improvements [38]. The principle idea of TRPO is to optimize the surrogate function in a Kullback-Leibler (KL) divergence constraint.

$$\begin{aligned} &\underset{\theta}{\operatorname{argmax}} \, \mathbb{E}_{\tau \sim \pi_{\theta_{old}}} [\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} A_{\pi_{\theta_{old}}}(s_t, a_t)] \\ &s.t. \ \mathbb{E}_{\tau \sim \pi_{\theta_{old}}}[KL[\pi_{\theta_{old}}, \pi_\theta]] \leq \delta \end{aligned} \tag{2.21}$$

In (2.21) inside expectation, the surrogate function is an approximation for obtaining the current advantages with trajectories sampled from the old policy $\pi_{\theta_{old}}$ by importance sampling.

The constraint term can be changed into a penalty function by the Lagrange multiplier method to solve unconstrained optimization:

$$\underset{\theta}{\operatorname{argmax}} \, \mathbb{E}_{\tau \sim \pi_{\theta_{old}}} [\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} A_{\pi_{\theta_{old}}}(s_t, a_t) - \beta KL[\pi_{\theta_{old}}, \pi_\theta]] \tag{2.22}$$

The Lagrange multiplier method ensures that there exists a proper β such that both optimizations have the same optima. However, in practice, an appropriate β cannot be easily determined, and experiments have demonstrated that using a fixed β is insufficient. Therefore, TRPO solves constrained optimization; however, it has a complex structure and is difficult to implement.

Proximal policy optimization (PPO) is a simple version of TRPO that retains the core principle of TRPO [40]. It also uses the surrogate function in TRPO:

$$L(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta_{old}}}[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}A_t] \tag{2.23}$$

$$= \mathbb{E}_{\tau \sim \pi_{\theta_{old}}}[r_t(\theta)A_t] \tag{2.24}$$

Instead of solving constrained optimization, PPO simply clip the surrogate function with a given hyperparameter as follows:

$$L^{CLIP}(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta_{old}}}[min(r_t(\theta)A_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)A_t)] \tag{2.25}$$

In practice, PPO uses not only surrogate function clipping, but also value function clipping when training the baseline state value function:

$$L^V = (V_\theta - V_{target})^2 \tag{2.26}$$

$$L^{V,CLIP} = min[L^V, (clip(V_\theta, V_{\theta_{old}} - \varepsilon, V_{\theta_{old}} + \varepsilon) - V_{target})^2] \tag{2.27}$$

With this simple concept, PPO maintains the gradient step in a safe range without complex methods. PPO can be easily implemented because it has no constraint and does not require second-order derivatives. In addition, it displays better performance than that of other policy optimization methods, including the original TRPO.

2.4 Neuroevolution for Reinforcement Learning

Neuroevolution (NE) is a research domain in which EC is used to train neural networks [44]. In the past, many NE algorithms focused on learning the topology of neural networks, as back-propagation is highly effective in weight optimization.

Therefore, NE algorithms focused on encoding the network topologies into genes. Neuroevolution through Augmenting Topologies (NEAT) is a famous topology encoding. With the NEAT algorithm, the topology of neural networks can be evolved with traditional GA operators, such as crossover and mutation. There have been many variations of NEAT that demonstrated impressive results [44]. However, the algorithms involved very small neural networks compared to those used in the modern deep learning era. Specifically, they only used hundreds or thousands of connections, while deep learning uses millions of connections.

In modern NE, scalability has become crucial for competing with other deep learning algorithms, and studies have demonstrated that the NE algorithm has better performance when they are sufficiently scaled. In many supervised tasks, such as vision, speech, and language models, gradient descent with back-propagation works exceptionally well and it is difficult for NE to outperform. This is because in supervised tasks, there is a smaller need to explore the search space, as the loss function is fixed, dense, and its local optima display sufficiently good performance [7]. Gradient descent, a type of hill-climbing algorithm, can efficiently find any local optima because it has strong good exploitation power.

However, in reinforcement learning, there is no direct supervision. Furthermore, the environments are sometimes stochastic and sparse. A major challenge in reinforcement learning is to explore the environment to attempt to discover rewards and find optimal actions. As mentioned in Section 2.3, many gradient-based deep reinforcement learning algorithms have been proposed; however, they often suffer from sparse and stochastic environments. In contrast, most EC algorithms have stronger exploration power than that of hill-climbing algorithms.

One notable development in modern NE is the ES algorithm proposed by Salimans et al. [37], who used a new form of ES, called natural ES (NES) to optimize the

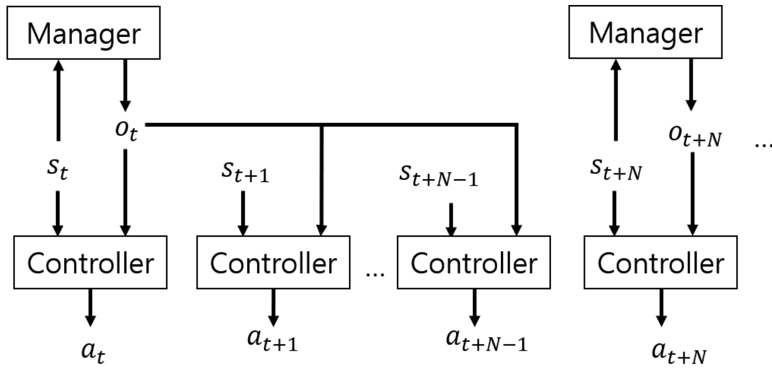


Figure 6: Basic framework of hierarchical reinforcement learning

network weights. This algorithm performs competitively with the most effective deep reinforcement learning algorithms, including deep Q-network (DQN) [31] and A3C in Atari games.

Whereas NES involves the fundamental concept of gradient descent, another NE algorithm based on the GA was proposed [45]. This algorithm uses a simple GA with top-k selection, elitism, no crossover, and Gaussian mutation. It also demonstrates results similar to those of DQN and A2C in Atari games. Furthermore, EC can be more easily parallelized than gradient-based policy optimization.

However, it is notable that in Atari games, NEs outperformed gradient-based policy optimization in some games, while gradient-based policy optimization outperformed NEs in other games. It appears that the two algorithms have a different method of optimization in terms of exploitation versus exploration. This topic is explored further in Chapter 3.

2.5 Hierarchical Reinforcement Learning

Many reinforcement learning algorithms have achieved success on various benchmark tasks. However, real-world problems involve large-scale planning, which re-

quires various forms of abstraction. Abstraction allows systems to focus on relevant tasks while micro-controllers handle details. Hierarchical reinforcement learning (HRL) is based on this concept of abstraction [2]. HRL decomposes a reinforcement learning problem into several subproblems, each of which is solved by a different agent. One agent, called the low-level agent, directly takes actions in the environment, while other agents, called high-level agents, work as macro-operators. From the viewpoint of balancing exploitation and exploration, the lowest-level agent exploits the environment for dense and immediate rewards, while high-level agents explore the environment for sparse and delayed rewards. In terms of the MDP, high-level agents consider both environmental observations and the lowest-level agent as a virtual environment. High-level agents take actions to the low level receives signals from both the lowest level agent and the environment. Figure 6 presents the basic framework of HRL. Based on the characteristics of high-level actions, HRL is divided into two categories; option-based HRL and goal-based HRL.

2.5.1 Option-based HRL

Sutton et al. [47] formalized HRL to include activities of reinforcement learning with options. One straightforward concept of an option is to select the lowest-level policy with a high-level agent's actions. Suppose that we have k different policies, $\pi_1, ..., \pi_k$. The action space of the high-level agent is a k dimensional categorical space. If the action of the high-level agent is i, π_i acts. In this case, the high-level agent and low-level agent share the same state space.

However, some algorithms do not use options as a policy selector but rather, as a low-level observation. If the high-level action is o, the low-level policy uses both the environmental observations and options as input, $\mu(s,o)$. When there are different policies and the option must select one of them, the option must be categorical.

However, when the options are inputs to the lower-level policy, they can be in any representation: categorical [49], multinoulli [34], or continuous valued [9, 20]. When an option is used as a lower level input, it is also called a modulation signal.

2.5.2 Goal-based HRL

When a modulation signal has special meaning in the observation space or feature space, it is called a goal signal. Nachum et al. [33] proposed a high-level policy that has the action space of (x, y) coordinates for the environment and provides the (x, y) location of the agent in the observation. The lower-level agent is given penalties in proportion to its distance to the goal location, causing the agent to move toward the goal. Because the modulation signal is in the same space as a subset of the environmental space, the concept from multi-goal reinforcement learning algorithms can be used [1]. In contrast, some algorithms use goal in the latent space [51] for the environment with the explicit location in the observation.

2.5.3 Exploitation versus Exploration

The structure of HRL is designed for the division of task into macro-management and micro-control. However, the structure itself does not necessarily improve the exploration power of the agent [11]. Balancing exploitation and exploration remains a significant challenge in HRL. In some algorithms, both agents are fixated on the micro-control and rarely explore, while other algorithms have difficulty training a low-level policy to perform micro-control. Pretraining the agent step by step may be effective for some environment, but fails in environments with sparse rewards.

Many HRL methods use reward function design to address the exploration problem. Some studies have used virtual rewards, called intrinsic rewards or motivations, to encourage an agent to explore [34, 6]. These methods usually evaluate current

states according to their novelty. An agent receives small motivations if the current state has been visited frequently, but receives large motivations when the current state has not yet been visited. Another study has proposed the advantage estimation of a high-level policy to a low-level policy as rewards, called auxiliary reward [28], to improve the exploitation power of the low-level policy.

Most algorithms have attempted to solve the exploitation exploration balancing problem while maintaining the framework of gradient-based policy optimization and modifying other aspects of the problem. This problem is addressed with hybridization of optimization in Chapter 4.

Chapter 3

Understanding Features of Evolutionary Policy Optimizations

Deep learning has achieved impressive performance thanks to powerful hardware and various algorithms [16]. By stacking multiple neural-network layers into specific architectures, deep neural networks can efficiently represent inductive biases for high-dimensional data [3]. However, understanding how deep neural networks work is nontrivial. Deep neural networks usually have a large number of parameters, causing high complexity. Many methods have been suggested to figure out the essential aspects of neural networks, especially for computer vision tasks. Various saliency methods have been used to examine the logical or abstract relations between input and hidden or output nodes. Data gradient methods [42], class activation maps [54, 41], and DeConvNet [53] have been used to provide useful analysis of convolution neural networks (ConvNets) for image data.

Recent advances in deep learning have also had a huge impact on reinforcement learning studies. Reinforcement learning with deep neural networks (i.e. deep reinforcement learning) can help overcome the curse of dimensionality. Policy gradient methods [30] and trust region-based methods [40] are some successful examples. In addition, recent studies have also suggested that some evolutionary methods may rival gradient-based optimization algorithms [37, 45]. It is well-known that evolutionary methods have significantly different properties from gradient-based optimization. They also demonstrate different behavior in various machine learning tasks [37].

Table 2: Convolutional network architecture

Layer	Input	Output
Conv 8×8 stride 4	$84 \times 84 \times 4$	$20 \times 20 \times 32$
Conv 4×4 stride 2	$20 \times 20 \times 32$	$9 \times 9 \times 64$
Conv 3×3 stride 1	$9 \times 9 \times 64$	$7 \times 7 \times 64$
Flatten	$7 \times 7 \times 64$	3136
Fully connected	3136	# of actions

However, not much information is known on the type of features learned by deep reinforcement learning methods. Several studies have used visual inspection to provide qualitative information on features in reinforcement learning [18]. There has also been a lack of studies examining evolutionary algorithms for reinforcement learning. Although evolutionary algorithms demonstrate competitive performance, how they function in complex reinforcement learning tasks remains largely unknown.

In this chapter, a sequence of experiments is described to examine the features of deep reinforcement learning networks. Visual analysis of the networks with various feature explanation methods is performed, including filter visualization, activation visualization, and data-gradient saliency mapping. The way how networks work in various situations is explained using several supplemental experiments. Agents obtained by both gradient-based policy optimization and evolutionary policy optimization are evaluated, and additional inspection of evolutionary policy optimization is performed with several experiments and behavioral analysis. The primary goal of this chapter is to reveal the reinforcement learning properties of evolutionary policy optimization. The results suggest that evolutionary policy optimization works differently from gradient-based policy optimization in terms of features and search space.

Algorithm 5 (1+Λ) ES for NE

In: number of children λ
In: Sample deviation σ
In: Learning Rate α
Initialize network parameters μ
repeat
  for $i \leftarrow 1$ to λ **do**
    Sample $\varepsilon_i \sim N(\mu, \sigma)$
    $F_i \leftarrow$ fitness(ε_i)
  end for
  Normalize F_i
  Adjust learning rate $alpha$ by Rectified Adam
  $\mu \leftarrow \mu + \frac{\alpha}{\sigma\lambda} \sum_i^n F_i \varepsilon_i$
until stop condition
return μ

3.1 Experimental Setup

Deep neural networks trained for tasks in Atari 2600 benchmarks were gathered. Considering the computation resource limits, eight different tasks were selected. The networks were trained with A2C and ES to compare the gradient-based method and evolutionary algorithm. The same ConvNet as the network used in a previous study [30] was used, as depicted in Table 2. For A2C, mini-batch training with a batch size of 128 and the RMSProp optimizer [50] were used. For ES, some modifications were applied to implement natural ES as in [37]. This process is described in Algorithm 5. ES sampled $n = 100$ solutions with a noise level of $\sigma = 0.1$, and parameters were updated with a learning rate $\alpha = 0.01$. Each solution was evaluated eight times, and the average results were used as fitness. Both algorithms were trained for 10,000,000 environment steps for a fair comparison. The performance of the networks in the given tasks is presented in Table 3. While networks trained by A2C demonstrated better performance in some tasks, they were outperformed by networks trained by ES in other tasks.

Table 3: Performance of agents trained by each algorithm

Environment	A2C	ES
Assault	1753.62	575.88
Asteroids	596.25	725.55
BattleZone	375.00	10 375.00
BeamRider	4930.00	671.00
Centipede	3041.25	8108.75
Gravitar	0.00	243.75
Frostbite	300.50	585.30
Seaquest	1830.00	427.50
SpaceInvaders	881.88	358.75

3.2 Feature Analysis

3.2.1 Convolution Filter Inspection

Because convolution filter weights in a ConvNet contain feature information trained from data, the filter weights can be visualized to examine the features of the network. It can be useful to plot the filter weights of the convolution filters in ConvNets. Figure 7 depicts some of the first layer filters of ConvNets trained by both A2C and ES for two tasks; SpaceInvaders and Centipede. On visual inspection, the difference between the two algorithms is clear. In the filters trained by A2C, sparsity and patterns appear. Some filters appear to have intuitive features such as edge detectors. In contrast, in filters trained by ES, noticeable patterns are not easily observed. It is thus difficult to distinguish filters trained by ES from pure random noises.

Visual assessment, however, is not sufficient for a clear analysis and can be misleading. Therefore, further statistical analysis is required to find out whether weights in filters have regular patterns or are random. Here, completely random noise is considered a normal distribution because a normal distribution is the maximum entropy distribution given finite mean and variance. A simple test was performed to compare

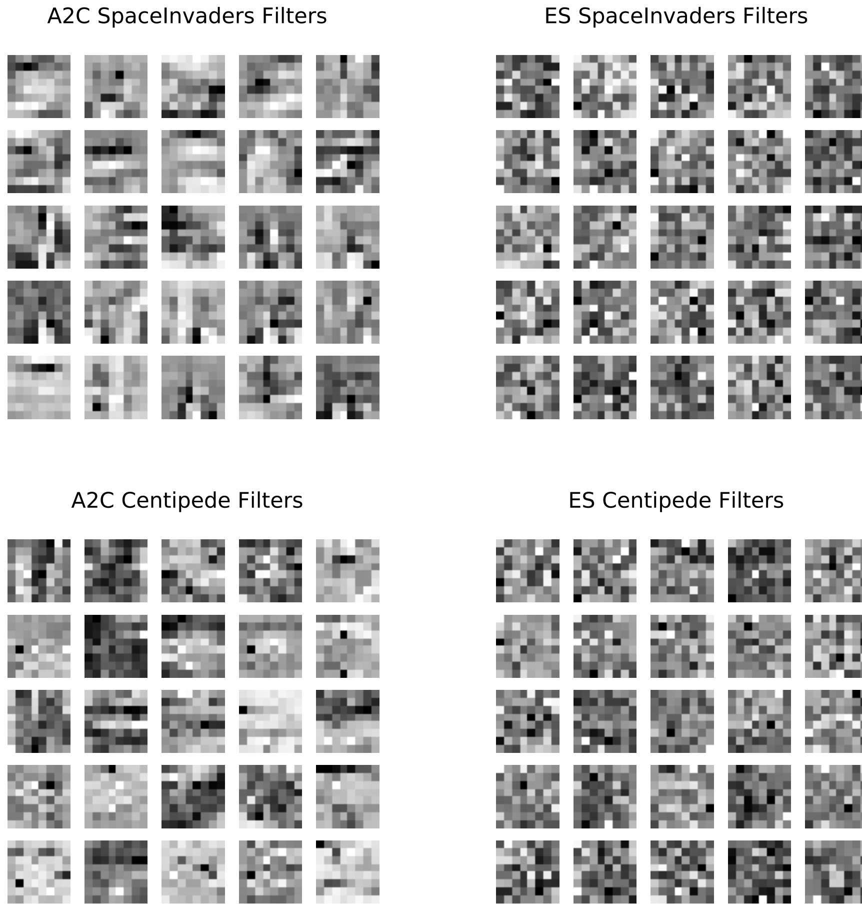


Figure 7: First convolutional layer filter visualizations

filters trained by A2C and ES. For each filter in the first convolutional layer, which consisted of $4 \times 8 \times 8 = 256$ weights, the weights were standardized by subtracting their mean and dividing by their standard deviations. Then, a Kolmogorov-Smirnov test was applied to determine whether the weights can be sampled from a normal distribution. If the test rejects the null hypothesis, it can be concluded that the weights are not entirely random. However, if the test accepts the null hypothesis, it is difficult to claim that the weights are not random. If the test with filter weights rejects

Table 4: The number of filters that passed Kolmogorov-Smirnov test

Algorithm	A2C		ES	
Layer	1	2	1	2
Assault	8	2	0	0
Asteroids	22	13	0	0
BattleZone	4	29	0	0
BeamRider	4	1	0	0
Centipede	16	5	0	0
Gravitar	5	7	0	0
Frostbite	4	61	0	0
Seaquest	7	8	0	0
SpaceInvaders	27	5	0	0

the null hypothesis, it can be stated that the filter passes the randomness test. Table 4 presents the result of the experiments, displaying the number of filters that passed the randomness test. The first convolutional layers trained by A2C have filters that passed the test. Especially for several tasks, more than half of the filters were not entirely random. In contrast, no filters trained by ES passed the test. Even in a task in which networks trained by ES performed better than networks trained by A2C, the filters trained by ES did not pass the test, whereas some filters trained by A2C did pass the test. The filters trained by ES were thus relatively random compared to those trained by A2C. The second layer filter weights, which consisted of $32 \times 4 \times 4 = 512$ weights, were also tested by the Kolmogorov-Smirnov test, which is presented in Table 4. While many filters trained by A2C passed the test, no filters trained ES passed the second layer's test.

Another approach is to compare each filter with another and test whether those filters have different distributions. If each filter learned different features, their weights must have different distributions. Kolmogorov-Smirnov tests were conducted on 496 pairs from 32 filters in the first convolutional layer and counted the number of pairs that passed the test. If a pair of filters pass the test, the two filters have different dis-

Table 5: The number of filter pairs that passed two sample Kolmogorov-Smirnov test

	A2C	ES
Assault	217	55
Asteroids	276	67
BattleZone	280	61
BeamRider	262	57
Centipede	288	56
Gravitar	171	44
Frostbite	299	47
Seaquest	192	53
SpaceInvaders	123	48

Table 6: The number of feature map pairs that passed two sample Kolmogorov-Smirnov test

	A2C			ES		
Layer	1	2	3	1	2	3
Assault	258.1	563.4	18.8	427.7	1496.0	264.4
Asteroids	222.2	238.6	207.9	417.6	1354.4	174.6
BattleZone	175.5	568.1	0.0	420.6	1258.3	250.1
BeamRider	377.9	482.6	9.0	479.1	1238.5	254.0
Centipede	292.2	409.1	113.5	421.3	1124.6	266.3
Gravitar	287.2	608.7	112.0	386.2	1237.7	277.8
Frostbite	132.0	670.3	0.0	451.0	1603.0	384.1
Seaquest	165.8	390.3	23.0	430.3	1289.3	296.7
SpaceInvaders	204.7	181.2	34.1	462.3	1242.6	298.4

tributions, so they have learned different features. Table 5 demonstrates the result of this analysis. At least 40 pairs passed the test for filters trained by ES. This result implies that some filters have different weight distribution than other filters and therefore learned different features. However, many more A2C trained filter pairs passed the test, while relatively fewer ES trained filter pairs passed the test.

The pairwise Kolmogorov-Smirnov test for the activation map was also conducted to make sure that different filters generated different feature maps. If feature maps are similar, they will never pass the test. For example, four consecutive frames of

observation cannot pass the test because they have the same value in general. However, if feature maps have different features, they can pass the test. These experiments were conducted in 10 different timesteps, and their average is reported in Table 6. Interestingly, unlike filter pair tests, the result of feature map pair tests presented that the much more feature map pairs of the network trained by ES passed the Kolmogorov-Smirnov test. Especially in the third convolutional layer, feature maps of the ES trained network were even more diverse than A2C trained networks. This result seems consistent with the assumption about the exploitation and exploration. A2C, a gradient-based optimization, tends to learn important features by exploitation. Therefore, there is a possibility that several channels learn the same feature. These characteristics lead to less diverse feature maps in many environments. In some environments, no feature map pairs passed the Kolmogorov-Smirnov test in the third convolutional layer This implies that there were small diversities in feature maps. However, ES tends to train weights by exploration, resulting in diverse feature maps. It seems that ES was effective in several environments, thanks to these diversities.

3.2.2 Saliency Map

A saliency map using data gradient was generated, as described in this section. Data gradient is a way to visualize saliency maps to explain the spatial information of the feature usage in a given image in a ConvNet. Suppose that image I_0, a target class c, and a class score function S_c of a ConvNet are given. The gradient of S_c with respect to I given an image I_0 can be calculated as follows.

$$w_c = \left. \frac{\partial S_c}{\partial I} \right|_{I_0}.$$

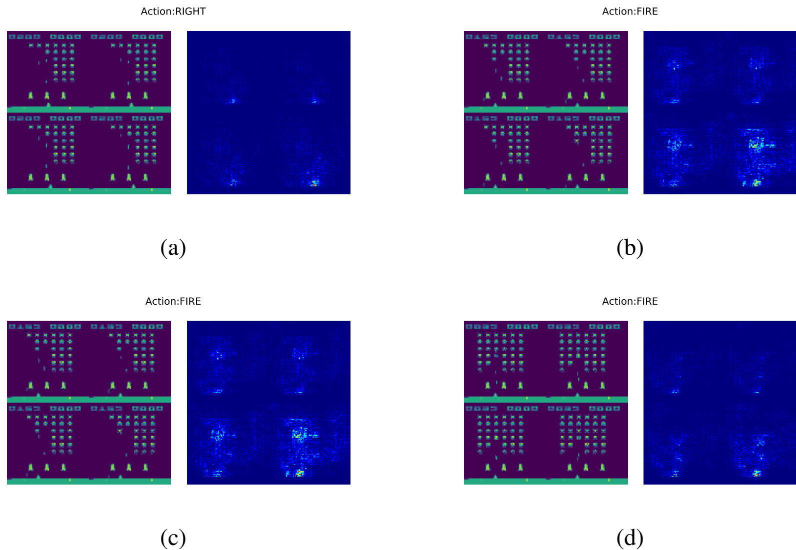


Figure 8: Saliency maps for different actions and states trained by A2C

This shows how much class score changes given small changes in pixels. As it is easy to calculate gradient with back-propagation in neural networks, saliency map can be applied to various networks without limitations. The data gradient saliency map was plotted for the given a series of states and actions. In Atari 2600 environments, a channel of images represents a time sequence of images, not a color encoding; therefore, the channel-wise maximum magnitude was not used. Instead, four images were placed into a 2×2 shape.

In the agents trained by A2C, interpretable features were not observed by their saliency maps. Let us use a SpaceInvaders agent as an example. Figure 8 presents input images, corresponding actions, and saliency maps. The agents generally focused more on the latest frame among the four frames. This is a natural phenomenon, considering that the latest frame plays a more decisive role in the impending action and return. In addition, depending on the task, an agent could represent saliency according to the given states and actions. The SpaceInvaders task had six different actions,

NOOP, FIRE, LEFT, RIGHT, LEFTFIRE, and RIGHTFIRE. When the agent took action RIGHT as in Figure 8a, it focused heavily on the position of the player but focused less on other pixels. In contrast, when it took action FIRE as in Figure 8b, it focused not only on the position of the player but also on the shooting target position. Moreover, the agent appeared to understand the semantic difference between actions given in different states. In SpaceInvaders, the action FIRE does not function the same all the time. After firing a bullet, FIRE does not fire another bullet for a short time (16 frames in the current environment). For a short time, FIRE works as NOOP, LEFTFIRE works as LEFT, and RIGHTFIRE works as RIGHT. The saliency maps demonstrate that the agent was able to understand this mechanism. When FIRE was available, as in Figure 8c, the agent focused on both the player and the target; however, when FIRE was not available, as in Figure 8d, the agent focused only on the player.

The saliency maps of agents trained by ES had different characteristics, however. Figure 9 presents the saliency maps of agents for Centipede trained by ES. On visual inspection, no patterns in the saliency maps of the agents were found. Even in some tasks in which agents trained by ES worked better, such as Centipede, their saliency maps appeared random. However, as in the filter weight analysis, visual inspection is not sufficiently precise and can be misleading. To verify whether the saliency maps of ES agents are random, a quantitative experiment was conducted. Let us assume that one cannot find out any pattern in the maps due to harsh noise. If it is possible to find some signals over noise in some areas, those areas may have meaningful features. Suppose that there are small patches (5×5) of the given image. The *episodic saliency distribution* of the patch can be defined as the distribution of data-gradients of the patch through the entire episode. Let the *frame saliency distribution* of the patch be the distribution of data gradients of the patch in a specific frame. For a given time step,

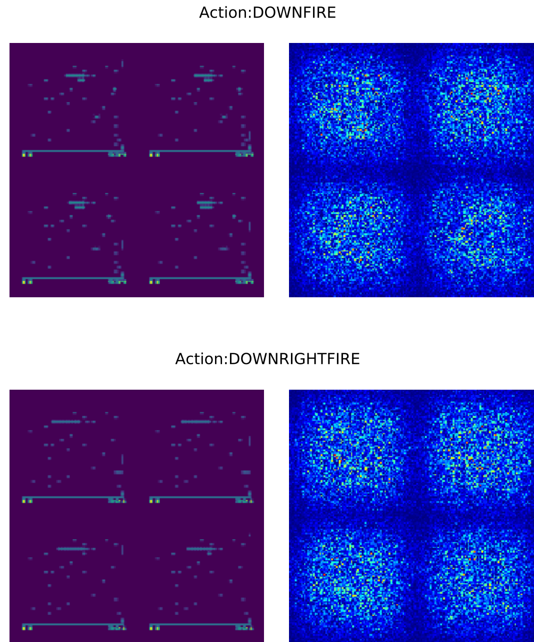
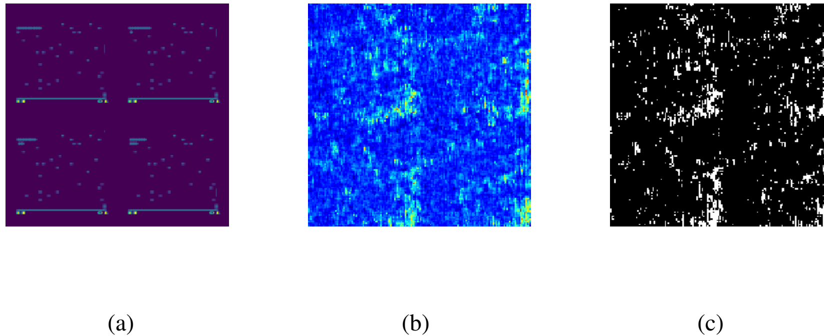


Figure 9: Saliency maps for evolutionary strategy

if the frame saliency distribution of a patch is different from the episodic distribution of the patch, which can be measured by the Kullback–Leibler (KL) divergence [24], the patch is considered to have meaningful features.

The KL divergence between the frame saliency distribution and the episodic distribution was plotted, sliding a 5×5 patch at a given time step. Figure 10 presents the plots of the input image, the KL divergence, and the KL divergence larger than 1.0 in a specific time step. The agent was trained for Centipede trained by ES. The plot reveals that the lower-right part of the image has an entirely different saliency distribution than that of the episodic distribution, which is the location of the player. This result reveals that although the original saliency maps appear completely random, there may be patterns in the noise that are related to some meaningful features.



(a) (b) (c)

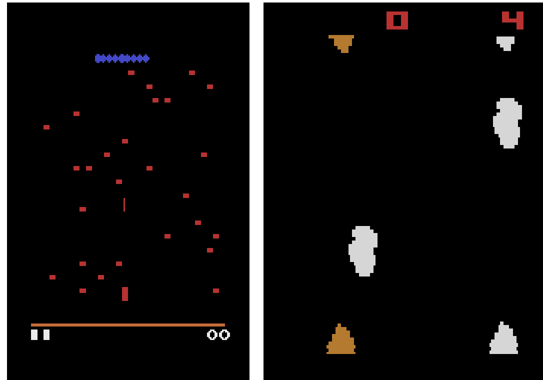
Figure 10: KL-divergence plots of agent trained by ES

3.3 Discussion

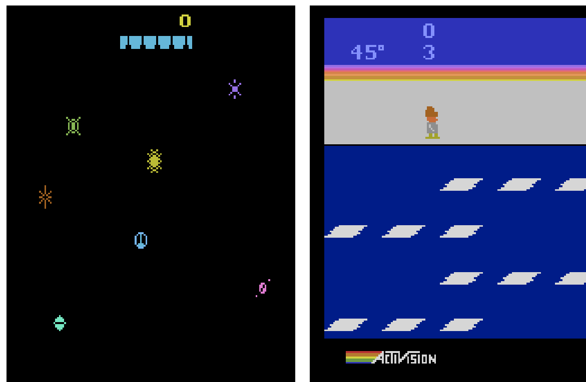
3.3.1 Behavioral Characteristics

To gain further insights, the behavioral characteristics of the agent in each environment were examined.

Centipede Agents trained by ES attempted to move to the lower-right corner and continued to fire by performing the action RIGHTDOWNFIRE, while agents trained by A2C attempted to track the movement of the centipede. A centipede (long blue object in Figure 11a) moves in a certain way; it sweeps the area from left to right or from right to left and moves down when it confronts obstacles or the right or the left edge of the playable area. If there are not too many obstacles, the centipede tends to visit the right or left edge frequently. Moreover, when a centipede is hit by bullets, it splits into two centipedes that move separately. Therefore, if an agent does not know the location of the centipedes or the player, it is unwise to move around in an attempt to track the centipedes. As all centipedes tend to move toward edges, it is reasonable to move toward the right or left side and continue firing, by performing RIGHTDOWNFIRE. The A2C agents attempted to follow the centipedes



(a) Centipede (b) Asteroids



(c) Gravitar (d) Frostbite

Figure 11: Four environments in Atari 2600

and did not have high scores. In contrast, the ES agents, which continued performing RIGHTDOWNFIRE and a few other movements, had much higher scores.

Asteroids In the Asteroids task, many asteroids (white and brown objects in Figure 11b) appear on the left or right side of the player and move from top to bottom or bottom to top. The environment never gives the location of asteroids and the player at the same time. It only gives visual information of the player for a time and then it only gives visual information of asteroids. If an agent does not know the location of the asteroids or the player, the best strategy is to continue turning around and fir-

ing. While the A2C agents clumsily attempted to follow the asteroids, the ES agents, which performed 56% of UPLEFTFIRE and other firing actions, scored more than the A2C agents.

Gravitar In the Gravitar environment, an agent must first move to one of planets (small objects in Figure 11c) to start a game. Before reaching the start point, no rewards are given. The A2C agents could not learn anything because rewards are extremely sparse. The ES agents, however, managed to find the starting point and gain scores. It appears that strong exploration power of ES achieved better performance.

Frostbite In the Frostbite environment, an agent first has to jump on ice planks (white objects in Figure 11d) on the water. Every jump gives the agent small rewards. After several jumps, an igloo is built and the agent must enter it to gain large rewards. When igloo building is finished, jumping on ice planks does not give any rewards. The A2C agent continued to jumping across ice planks even after the igloo is built. Because of exploitation nature of gradient-based optimization, A2C cannot find the state of entering the igloo. In contrast, the ES agents occasionally entered the igloo and were rewarded with huge scores. It also appears that strong exploration power of ES achieved better performance.

3.3.2 ES Agent without Inputs

In some environments, such as Centipede and Asteroids, the ES agents rarely reacted to the states but obtained much more scores. A simple additional experiment was conducted to figure out the reason for this result. In four different tasks in which ES agents performed better, I also trained an agent by ES without inputs. The agent was provided not the exact state images, but an image tiled with the mean values of the original states; therefore, their outputs were always the same. Table 7 presents the

Table 7: Performance of agents trained by evolutionary strategy (ES) without inputs

Environment	A2C	ES	ES without inputs
Asteroids	596.25	725.55	813.75
BattleZone	375.00	10375.00	5875.00
Centipede	3041.25	8108.75	4516.63
Gravitar	0.00	243.75	187.50
Frostbite	300.50	585.30	127.70

results of ES without inputs. Without inputs, agents trained by ES displayed worse performance than the agent trained with proper inputs. However, in all four tasks, the ES without inputs performed better than A2C. Especially in the Asteroids task, ES without inputs performed better than ES with inputs.

This discovery offers new knowledge about the way ES trains agents. These agents make use of macro-knowledge of tasks to learn useful action rather than learning detailed understanding of the way to respond to corresponding states. ES leads to only slightly better performance than the agent, which can be trained even without detailed inputs. Agents trained by ES do not seem to respond to states because macro-knowledge dominates their policies. ES does not work well in tasks in which macro-knowledge is not crucial, such as in Assault, SpaceInvaders, and BeamRider. Instead, A2C demonstrated superior performance on those tasks by learning proper features.

Chapter 4

Hybrid Search for Hierarchical Reinforcement Learning

For reinforcement learning, learning policies over a long-time with delayed rewards is a major challenge. Hierarchical reinforcement learning (HRL) can be used to solve these challenges. However, hierarchical policies do not necessarily help the exploration of skills [11]. Even with the HRL framework, gradient-based policy optimization cannot overcome the sparsity of rewards. Many different algorithms have been proposed, such as careful goal design [33], dense reward design [28, 34], the lower level pretraining [14], task dividing [20] and entropy-based exploration [11].

Most HRL algorithms still function in the paradigm of gradient-based policy optimization. However, as mentioned in Chapter 3, evolutionary policy optimizations perform better in sparse-reward environments. If they are supported by suitable exploitation methods, they can be useful in various complex tasks. It appears that the hybridization of policy optimization can result in an algorithm with balanced exploration and exploitation. Therefore, in this chapter, a hybrid search algorithm is proposed for the HRL framework. Based on the deep neural network HRL framework, different optimization strategies are applied to the low-level policy network and high-level policy network. The high level, which requires strong exploration power, is optimized by an evolutionary strategy while the low level, which requires strong exploitation power, is optimized by gradient-based optimization, PPO. A technique called a direction-masking network is used to improve low-level control performance. The experimental results demonstrate that the hybridization of policy optimizations

generated an effective agent performing both macro-management and micro-control without pretraining or goal designing.

4.1 Method

Hybrid Optimization for HRL The HRL framework for hybrid search consists of two levels, the high-level policy and low-level policy. The low-level policy deals with the micro-control for the agent's actions, while the high-level policy suggests the movement directions of the agent. There are eight directions; up, down, left, right, and an additional four directions in between. The high-level policy produces categorical signals with a dimension of 8. Then, the low-level policy is trained to move in that direction.

It is clear that the high-level policy requires strong exploration power, while the low-level policy requires strong exploration power. Therefore, a different optimization algorithm for each policy level is proposed. For the low-level policy which requires intensive exploitation, a state-of-the-art gradient-based policy optimization (PPO) is performed to train micro-control of the agent. In contrast, the high-level policy, which requires exploration, is trained with evolutionary algorithms. Two different algorithms are used, ES and GA, that have different exploitation and exploration characteristics. ES tends to have more exploitation power than the GA, while the GA has more exploration power than ES. In this study, only simple ES are used because complex ES implementation needs larger time complexity. For example, covariance matrix adaptation evolutionary strategy (CMA-ES) [19] needs $O(n^2)$ complexity given the number of parameter n. In deep neural network, the number n is usually large, more than 60k in this chapter. Performing an $O(n^2)$ algorithm for this size is impractical. In contrast, simple ES only needs $O(n)$ complexity so it can be performed in efficient time.

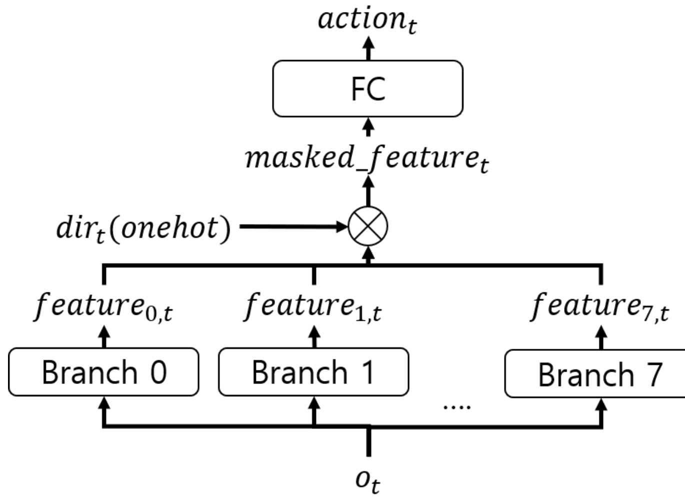
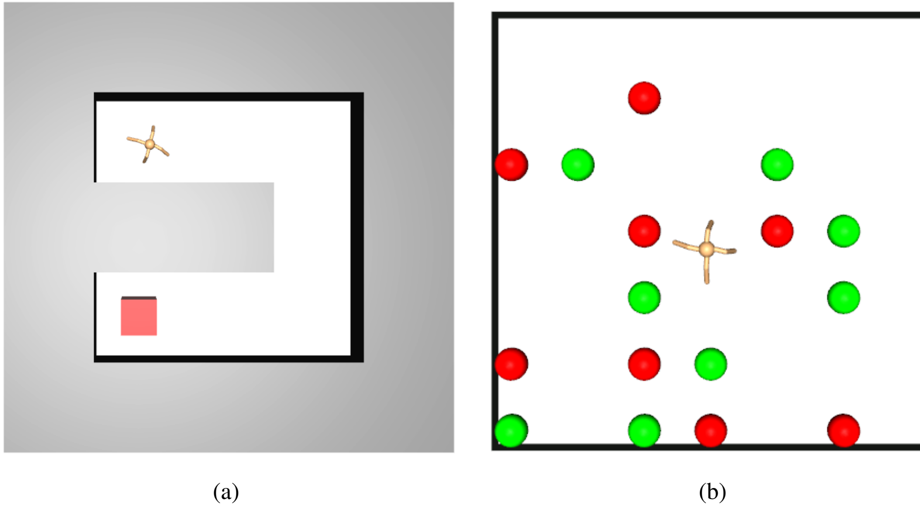


Figure 12: Direction-masking network

Direction-Masking Network Training a policy that moves in a given direction is also a nontrivial task. PPO with a simple multi-layer perceptron cannot handle this task, and various sophisticated algorithms, such as meta-learning[13] and reward function evolution [21], are used instead. In this chapter, this problem is overcome with a new architecture design, called a direction-masking network for the low level. The direction-masking network uses robot observations and directions and consists of eight branch networks. Each branch network is a feature-extracting multi-layer perceptron for each direction with robot observation inputs. The eight feature vectors are masked with the target direction inputs. Only features corresponding to the given direction survive while the remaining features are changed to zeros. The surviving features are used to determine the actions. Figure 12 illustrates the mechanism of the direction-masking network. The direction-masking network allows the low level policy to solve a challenging task without multiple policy networks or complex training algorithms.



(a) (b)

Figure 13: AntMaze and AntGather environments

Adaptive Mutation for Exploration There exist some environments with extremely sparse rewards that even an evolutionary algorithm cannot handle. Therefore, evolutionary algorithms are modified to increase the exploration power when the rewards are too sparse. When all children of evolutionary algorithms have no rewards in an episode, the mutation power is increased to produce children to search a broader space in the next generation. However, if all children receive some rewards in an episode, the mutation power is decreased to search a narrower space in the next generation. With this adaption method, the algorithm can search the problem space with sufficient exploration power.

4.2 Experimental Setup

4.2.1 Environment

To ensure that both macro-management and micro-control are performed well, a complex environment is required. Rllab [10] is known to have various environments

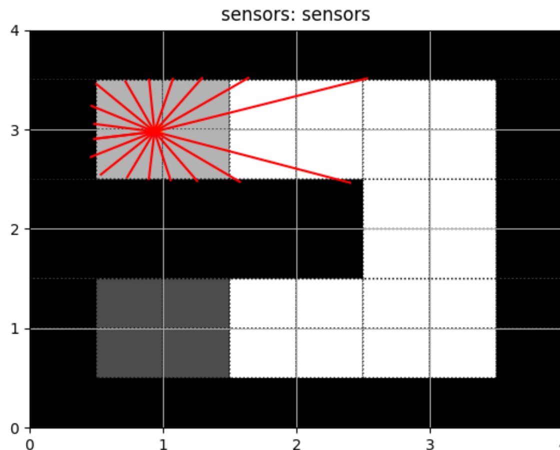


Figure 14: Wall readings for Maze environment

with high complexity. The environments used were AntMaze and AntGather. In these environments, agents must control their joints to continue moving. In addition, they must manage a long-term strategy to overcome obstacles and approach the appropriate targets. In AntMaze, there is a large wall between the agent and destination, as illustrated in Figure 13a. Therefore, the agent must use a ⊃-shaped detour to reach its destination. There is a scaling parameter that determines the size of the maze, and a width of 4 was used for the experiments. An agent is considered successful when it is sufficiently close to the destination. The success rate of the agent is assessed for the evaluation. An ant has four legs attached to its torso, and each leg has two joints, one on its hip and one on its knee.

Unlike most other experiments [33, 27], the agent does not have access to its absolute coordinates. Instead, it uses a sensor with 20 different rays originating from its agent, which is the default observation given in [10]. Assume that the agent has a ray with a length of 10.0. If there is a wall in the ray's direction and its distance is 7.0, the wall reading for the ray becomes $7.0/10.0 = 0.7$. If there is no wall within 10.0 in

the ray direction, the wall reading is 0.0. The goal readings work similarly to the wall readings, except that the target is the goal, not the walls. In addition to the agent's locomotion observations, 20 wall readings and 20 goal readings are given to the agent as observations. In the experiments the sensor range 10.0 was used, and the sensor span was 2π (the agent shot 20 rays in 360°). Figure 14 presents the operation of the sensor in the Maze environments. In AntGather, an ant must gather green ball-shaped objects, or apples, while avoiding red ball-shaped objects, or bombs as illustrated in Figure 13b. The agent in the AntGather environment uses a sensor similar to that of AntMaze agent. It shoots 20 rays originating from the agent in all directions, and gathers apple readings and bomb readings. With the ant's locomotion observations, 20 apple readings and 20 bomb readings are given.

The high-level policy and low-level policy are given with different observations and rewards. The low-level policy is given only with the locomotion observations of the agent. In the Ant environments, AntMaze and AntGather, it is given 15 positional, 14 velocity, and 84 contract force observations. The low level is rewarded with its forward velocity reward to the direction given by the high level. The high level, in contrast, is given both locomotion observations and sensor observations. In Ant environments, the agent is given 113 locomotion observations and 40 sensor observations. It is rewarded with the actual environmental rewards; a goal arrival reward for Maze, and +1 for apples, and -1 for bombs in Gather. In addition, to encourage exploration, the high level is also rewarded with the moving distance of agents in a cycle. An action is represented by eight continuous values in the range $[-150, 150]$ for the joint controls of the ant

4.2.2 Network Architectures

Multilayer perceptron networks parameterize both high-level and low-level policies. A high-level network consists of two hidden layers with 256 nodes activated by the *tanh* function. The output of the high-level policy is eight logits for a categorical distribution. Each category denotes the direction that the agent should follow.

The low-level network consists of eight branch networks with two 64-node hidden layers and *tanh* activation function. The features of these networks are masked by the direction given by the high level network. If the given direction is zero, only one branch network's feature is used, and features from other networks become zero. Then, with the 64 nodes (64×7 zeros), the action is calculated. The action values are sampled from independent normal distributions whose means are the output of the network, and standard deviations are learnable parameters initialized to one.

4.2.3 Training

The low-level policy was optimized with the PPO [40] algorithm, which is known for its performance in simple locomotion tasks. The algorithm sampled 4000 steps of (s, a, r)s and their returns were calculated with a discount $\gamma = 0.99$. In addition, the advantages were calculated with generalized advantage estimation of $\lambda = 0.95$ [39]. Then, the algorithm trained the advantages in mini-batches with a size of 128 for 10 mini-epochs. Both surrogate functions and value predictions were clipped into the $\varepsilon = 0.2$ range. The gradient was optimized with the Rectified Adam [29] optimizer with an initial learning rate of 0.003.

Meanwhile, the high level was optimized with the same ES as in Algorithm 5 and the GA in [45]. Both algorithms generated $\lambda = 10$ children with a mutation power of $sigma = 0.001$, with adaptive modification. In the case of ES, instead of sampling 10 random steps, ES sampled five random steps and their negative values as children.

Then, it updated μ with the learning rate of α was 0.0001 with the episodic fitness. It used the rank of children normalized into $[-0.5, 0.5]$; the fitness of the best child became 0.5, while the fitness of the worst child became -0.5. Selecting the best learning rate in ES is also a large challenge. ES updated μ with the Rectified Adam optimizer assuming that the evolving step $\sigma\lambda\sum_i^n F_i\varepsilon_i$ was the actual gradient of the point μ. With this technique, the learning rate of α was selected adaptively. In contrast, in the GA, top three children were selected and used for reproduction. The elite (best-performing) individual was always kept intact while training. No crossover was used in this GA.

The networks were trained for 10,000,000 steps in the environments, divided into 2,000-step episodes, and repeated five times. Each episode contained 20 sub-episodes, or cycles, of a step length of 100. The genes were updated every 500 steps. Therefore, there were 500 generations in both ES and the GA.

4.3 Results

4.3.1 Comparison

For comparison, four different algorithms were used for training in the same environment. Two algorithms were used as baseline algorithms to verify the effectiveness of hybrid search, while the other two algorithms were alternative.

PPO only A network was trained with only a PPO, which is the low-level part of the networks, without the HRL framework. Because the environments do not provide rewards from directional velocity without the HRL framework, merely using raw rewards is unfair. Therefore, raw rewards and directional velocity rewards were scalarized into a single value.

Table 8: Results of hybrid hierarchical reinforcement learning

Task	AntGather	AntMaze
Random walk	0.14 ± 1.07	0.00 ± 0.00
Hybrid HRL (ES)	2.20 ± 1.38	0.35 ± 0.47
Hybrid HRL (GA)	0.45 ± 1.44	0.00 ± 0.00
PPO only	0.04 ± 0.90	0.00 ± 0.00
PPO+Random Move HL	-0.10 ± 1.54	0.00 ± 0.00
PPO+Random Search HL	0.76 ± 1.70	0.00 ± 0.00
SNN4HRL	1.92 ± 0.52	0.00 ± 0.00

Random Move In the hybrid algorithm, the contribution of ES in the high level was uncertain because the low level was trained with a PPO. An algorithm was used in which random one-hot vector generators replaced the high level. This experiment clarified the contribution of the ES-optimized high level.

Random Search Some studies have suggested that a simple random search may be as good as evolutionary algorithms [45]. Therefore, a random search was used for high-level policy optimization. This experiment can reveal the difference between a simple random search and evolutionary search.

SNN4HRL SNN4HRL, a stochastic neural network for hierarchical reinforcement learning [14], which uses a similar categorical modulation signal, was used for comparison to the gradient-based HRL algorithm. The results were taken from the report by Nachum et al. [33] as the original paper did not provide the results for the current environments.

4.3.2 Experimental Results

The experimental results are depicted in Table 8. Every result is reported by 100 evaluations with the trained model. The first value denotes the mean of the episodic rewards, where $\pm$ indicates the standard deviation.

The hybrid HRL with ES produced favorable results in both AntGather and AntMaze. Compared to PPO only, the hybrid HRL had superior performance in both environments. The PPO-only agent could not reach the goal in AntMaze at all, whereas the hybrid HRL managed to reach the goal. In the AntGather environment, the results of the PPO-only agent were not better than those of the random walk, whereas the hybrid HRL produced better results. It is evident the agent benefitted from the hierarchical structure. The comparison between hybrid HRL and PPO plus random move revealed the contribution of ES to the high level policy. PPO plus random move agent performed as well as the random walk agent. This result indicates that ES did not find a good policy by chance; instead, it optimized the policy according to the rewards. The hybrid HRL with ES also outperformed SNN4HRL, a gradient-based HRL algorithm. SNN4HRL did not reach the goal at all in AntMaze, whereas hybrid HRL reached the goal in nearly 50% of trials. In addition, it is notable that the hybrid HRL used no pretraining at all, whereas SNN4HRL used pretraining for the low level.

In contrast, hybrid HRL with GA did not produce impressive results. Its performance was worse than SNN4HRL in AntGather. It was also unable to find any good agents in AntMaze. Interestingly, the random search high level agent produced better results than hybrid HRL with GA in AntGather; however, the results were not better than those of hybrid HRL with ES. It thus appears that the optimization power of the GA is not better than that of a random search, whereas ES has excellent optimization power in HRL.

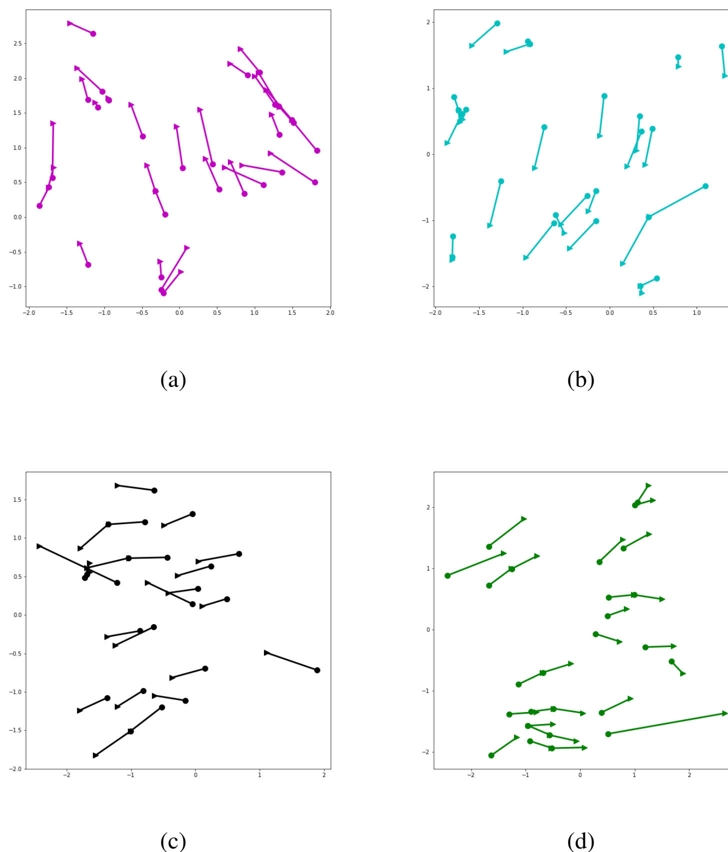


Figure 15: Cycle movement for four directions

Meanwhile, the hybrid HRL had a large variance in both environments, while SNN4HRL had a much smaller variance. It thus appears that the high level has difficulty fine-tuning the high level policy, resulting in a high variance in performance.

4.3.3 Behavior of Low-Level Policy

The function of the low-level policy is to accurately walk toward the direction. To ensure that the low-level policy is trained as designed, the cyclic movement of the agent was plotted given an action. Figure 15 illustrates the cyclic movement of the

ant agent in the AntGather environment. Figure 15a, 15b, 15c, and 15d depicts the cyclic movement of agents for the up, down, left and right directions, respectively. The agents moved from round shaped markers to triangle-shaped markers. The low level moved toward the given direction quite well, but with some errors. It appears that the low level policy was trained as designed with the hybrid HRL without any pretraining.

4.4 Conclusion

In this chapter, I proposed a hybrid HRL algorithm to generate successful agents for complex tasks. Several techniques were used to overcome the sparsity of rewards and the complexity of tasks. The direction-masking network was used for the agent to learn how to move in a given direction without complex learning algorithms Adaptive mutation power was used to overcome the sparsity of rewards by enlarging or reducing the search space. The results indicate that the hybrid HRL was more effective than simple gradient-based reinforcement learning and had better performance than a gradient-based HRL algorithm. In some environments in which exploration is crucial, the hybrid HRL produced better results than gradient-based HRL. Furthermore, the hybrid HRL used no pretraining techniques, while the gradient-based HRL used a pretraining technique. However, in hybrid HRL, there was a large variance in the results, while the results of gradient-based HRL had a small variance. The variance of hybrid HRL was even larger than that of the random walk agent, which may be a critical drawback. This result was likely due to the lack of fine-tuning in the high level policy which was trained with an evolutionary algorithm. These results demonstrate that even for the high level policy, a proper balance between exploitation and exploration is critical. Therefore, this algorithm can be further improved by hybridizing the high level optimization itself.

Chapter 5

Conclusion

5.1 Summary

In this thesis, a hybrid search algorithm was proposed for complex reinforcement learning tasks. In balancing exploitation and exploration in the optimization problem, domain knowledge is one of the most critical factors in designing optimization algorithms.

Many reinforcement learning algorithms have been suggested to overcome the exploration problem. But most of them do not consider other optimization methods than gradient-based optimizations. Gradient-based optimization is a powerful local optimizer, but it lacks exploration power. Most studies use modification other than changing the optimization algorithm, which is critical to the exploration power. No free lunch theorem is also applied to the field of optimization. In Chapter 3, both gradient-based policy optimization and evolutionary policy optimization were analyzed with various environments with different characteristics. The results demonstrate that while gradient-based optimizations are good at following immediate rewards, it fails to find the big picture of the environment. However, the evolutionary policy optimizations are good at finding the macro-knowledge of the environments, even with lesser observation.

A hybrid hierarchical reinforcement learning algorithm is proposed in Chapter 4. In the hybrid HRL, an evolutionary algorithm, which is strong for finding the macro-knowledge of the environment, is used to train the high-level macro-management

agent and a gradient-based algorithm, which is strong for following immediate rewards, is used to train the low-level micro-controller. The result showed that the hybrid HRL worked better than the gradient-only HRL algorithm. Even without any pretraining, the hybrid HRL trained the low level as it was designed. The high level also had enough exploration power to overcome extremely sparse rewards.

5.2 Future Work

While showing a good result, more improvement for the hybrid algorithm can be made. In the hybrid HRL, the high-level policy trained by an evolutionary algorithm suffers from huge variance, which is not found in gradient-based HRL algorithms. It seems that even for the macro-manager in HRL, fine-tuning the policy is still needed to reduce the variance of policy. Hybridizing the high level itself may be a solution to the problem. Another topic is to design a domain-specific evolutionary optimization. In this thesis, only a typical evolutionary strategy was used for high-level optimization in HRL. However, as it is cleared in this thesis, domain knowledge is always critical to the optimization algorithm design. The adaptive mutation was used in the hybrid HRL, but it is still room to improve the optimization with more domain knowledge.

Bibliography

[1] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba. "Hindsight experience replay". In: *Advances in neural information processing systems*. 2017, pp. 5048–5058.

[2] A. G. Barto and S. Mahadevan. "Recent advances in hierarchical reinforcement learning". In: *Discrete event dynamic systems* 13.1-2 (2003), pp. 41–77.

[3] Y. Bengio, A. Courville, and P. Vincent. "Representation learning: A review and new perspectives". In: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), pp. 1798–1828.

[4] A. Berny. "Selection and reinforcement learning for combinatorial optimization". In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2000, pp. 601–610.

[5] T. N. Bui and B. R. Moon. "Genetic algorithm and graph partitioning". In: *IEEE Transactions on computers* 45.7 (1996), pp. 841–855.

[6] Y. Burda, H. Edwards, A. Storkey, and O. Klimov. "Exploration by random network distillation". In: *arXiv preprint arXiv: 1810.12894* (2018).

[7] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun. "The loss surfaces of multilayer networks". In: *Artificial Intelligence and Statistics*. 2015, pp. 192–204.

[8] A. Darwish, A. E. Hassanien, and S. Das. "A survey of swarm and evolutionary computing approaches for deep learning". In: *Artificial Intelligence Review* (2019), pp. 1–46.

[9] P. Dayan and G. E. Hinton. “Feudal reinforcement learning”. In: *Advances in neural information processing systems*. 1993, pp. 271–278.

[10] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. “Benchmarking deep reinforcement learning for continuous control”. In: *International Conference on Machine Learning*. 2016, pp. 1329–1338.

[11] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine. “Diversity is all you need: Learning skills without a reward function”. In: *Proceedings of International Conference on Learning Representations*. 2019.

[12] S. Filippi, O. Cappé, and A. Garivier. “Optimism in reinforcement learning and Kullback-Leibler divergence”. In: *2010 48th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE. 2010, pp. 115–122.

[13] C. Finn, P. Abbeel, and S. Levine. “Model-agnostic meta-learning for fast adaptation of deep networks”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 1126–1135.

[14] C. Florensa, Y. Duan, and P. Abbeel. “Stochastic neural networks for hierarchical reinforcement learning”. In: *Proceedings of International Conference on Learning Representations*. 2017.

[15] M. Gen and R. Cheng. “Genetic algorithms and engineering design. 1997”. In: *John Wily and Sons, New York* (1997).

[16] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*. Vol. 1. MIT Press, 2016.

[17] R. Gower, A. Heydtmann, and H. Petersen. “Lego: Automated model construction”. In: (1998).

[18] S. Greydanus, A. Koul, J. Dodge, and A. Fern. "Visualizing and understanding atari agents". In: *International Conference on Machine Learning*. 2018, pp. 1792–1801.

[19] N. Hansen, S. D. Müller, and P. Koumoutsakos. "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES)". In: *Evolutionary computation* 11.1 (2003), pp. 1–18.

[20] N. Heess, G. Wayne, Y. Tassa, T. Lillicrap, M. Riedmiller, and D. Silver. "Learning and transfer of modulated locomotor controllers". In: *arXiv preprint arXiv: 1610.05182* (2016).

[21] R. Houthooft, Y. Chen, P. Isola, B. Stadie, F. Wolski, O. J. Ho, and P. Abbeel. "Evolved policy gradients". In: *Advances in Neural Information Processing Systems*. 2018, pp. 5400–5409.

[22] A. B. Kahng and B. R. Moon. "Toward More Powerful Recombinations." In: *ICGA*. 1995, pp. 96–103.

[23] V. R. Konda and J. N. Tsitsiklis. "Actor-critic algorithms". In: *Advances in neural information processing systems*. 2000, pp. 1008–1014.

[24] S. Kullback and R. A. Leibler. "On information and sufficiency". In: *The annals of mathematical statistics* 22.1 (1951), pp. 79–86.

[25] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman. "Building machines that learn and think like people". In: *Behavioral and Brain Sciences* 40 (2017).

[26] S. Lee, J. Kim, J. W. Kim, and B.-R. Moon. "Finding an optimal LEGO® brick layout of voxelized 3D object using a genetic algorithm". In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. 2015, pp. 1215–1222.

[27] A. Levy, G. Konidaris, R. Platt, and K. Saenko. "Learning multi-level hierarchies with hindsight". In: *Proceedings of International Conference on Learning Representations*. 2019.

[28] S. Li, R. Wang, M. Tang, and C. Zhang. "Hierarchical Reinforcement Learning with Advantage-Based Auxiliary Rewards". In: *Advances in Neural Information Processing Systems*. 2019, pp. 1407–1417.

[29] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han. "On the Variance of the Adaptive Learning Rate and Beyond". In: (2020).

[30] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. "Asynchronous methods for deep reinforcement learning". In: *International Conference on Machine Learning*. 2016, pp. 1928–1937.

[31] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), p. 529.

[32] P. Moscato et al. "On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms". In: *Caltech concurrent computation program, C3P Report* 826 (1989), p. 1989.

[33] O. Nachum, S. S. Gu, H. Lee, and S. Levine. "Data-efficient hierarchical reinforcement learning". In: *Advances in Neural Information Processing Systems*. 2018, pp. 3303–3313.

[34] A. Pashevich, D. Hafner, J. Davidson, R. Sukthankar, and C. Schmid. "Modulated policy hierarchies". In: *arXiv preprint arxiv: 1812.00025* (2018).

[35] P. Petrovic. “Solving lego brick layout problem using evolutionary algorithms”. In: *Proceedings to Norwegian Conference on Computer Science*. 2001.

[36] I. Rechenberg. “Evolutionsstrategien”. In: *Simulationsmethoden in der Medizin und Biologie*. Springer, 1978, pp. 83–114.

[37] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever. “Evolution strategies as a scalable alternative to reinforcement learning”. In: *arXiv preprint arxiv: 1703.03864* (2017).

[38] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. “Trust region policy optimization”. In: *International conference on machine learning*. 2015, pp. 1889–1897.

[39] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. “High-dimensional continuous control using generalized advantage estimation”. In: *arXiv preprint arxiv: 1506.02438* (2015).

[40] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv: 1707.06347* (2017).

[41] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. “Grad-cam: Visual explanations from deep networks via gradient-based localization”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 618–626.

[42] K. Simonyan, A. Vedaldi, and A. Zisserman. “Deep inside convolutional networks: Visualising image classification models and saliency maps”. In: *arXiv preprint arxiv: 1312.6034* (2013).

[43] E. Smal. “Automated brick sculpture construction”. PhD thesis. Stellenbosch: Stellenbosch University, 2008.

[44] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen. “Designing neural networks through neuroevolution”. In: *Nature Machine Intelligence* 1.1 (2019), pp. 24–35.

[45] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune. “Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning”. In: *arXiv preprint arXiv: 1712.06567* (2017).

[46] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. “Policy gradient methods for reinforcement learning with function approximation”. In: *Advances in neural information processing systems*. 2000, pp. 1057–1063.

[47] R. S. Sutton, D. Precup, and S. Singh. “Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning”. In: *Artificial intelligence* 112.1-2 (1999), pp. 181–211.

[48] R. S. Sutton, A. G. Barto, et al. *Introduction to reinforcement learning*. Vol. 2. 4. MIT press Cambridge, 1998.

[49] C. Tessler, S. Givony, T. Zahavy, D. J. Mankowitz, and S. Mannor. “A deep hierarchical approach to lifelong learning in minecraft”. In: *Thirty-First AAAI Conference on Artificial Intelligence*. 2017.

[50] T. Tieleman and G. Hinton. “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude”. In: *COURSERA: Neural networks for machine learning* 4.2 (2012), pp. 26–31.

[51] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu. “Feudal networks for hierarchical reinforcement learning”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 3540–3549.

[52] D. H. Wolpert and W. G. Macready. “No free lunch theorems for optimization”. In: *IEEE transactions on evolutionary computation* 1.1 (1997), pp. 67–82.

[53] M. D. Zeiler and R. Fergus. “Visualizing and understanding convolutional networks”. In: *European conference on computer vision*. Springer. 2014, pp. 818–833.

[54] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. “Learning deep features for discriminative localization”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2921–2929.

국문 초록

많은 최적화 문제에서 탐사와 탐험의 균형을 맞추는 것은 매우 중요한 문제이다. 진화 전략과 유전 알고리즘과 같은 진화 알고리즘은 자연에서의 진화에서 영감을 얻은 메타휴리스틱 알고리즘이다. 이들은 조합 최적화, 연속 최적화와 같은 다양한 최적화 문제를 풀기 위해 사용되었다. 하지만 진화 알고리즘은 지역 최적해 근처에서의 미세 조정, 즉 탐사에 약한 특성이 있다. 이러한 결점함은 혼합화를 통해 극복할 수 있다. 혼합 유전 알고리즘, 혹은 미미틱 알고리즘이 성공적인 혼합화의 사례이다. 이러한 알고리즘은 최적화 문제의 해 공간이 기하급수적으로 넓더라도 성공적으로 만족스러운 해를 찾아낸다.

한편 심층 학습의 시대에서, 탐사와 탐험의 균형을 맞추는 문제는 종종 무시되었다. 하지만 심층 강화학습에서는 탐사와 탐험의 균형을 맞추는 일은 지도학습에서보다 훨씬 더 중요하다. 많은 실제 세계의 환경은 기하급수적으로 큰 상태 공간을 가지고 있고 에이전트는 이를 탐험해야만 한다. 충분한 탐험 능력이 없으면 에이전트는 상태 공간의 극히 일부만을 밝혀내어 결국 즉각적인 보상만 탐하게 될 것이다.

본 학위논문에서는 강한 탐사 능력을 가진 그레디언트 기반 정책 최적화와 강한 탐험 능력을 가진 진화적 정책 최적화를 혼합하는 기법을 제시할 것이다. 우선 그레디언트 기반 정책 최적화와 진화적 정책 최적화를 다양한 환경에서 분석한다. 결과적으로 그레디언트 기반 정책 최적화는 즉각적 보상에 효과적이지만 보상의 밀도가 낮을때 취약한 반면 진화적 정책 최적화가 밀도가 낮은 보상에 대해 강하지만 즉각적인 보상에 대해 취약하다는 것을 알 수 있다. 두 가지 최적화의 특징상 차이점이 혼합적 정책 최적화의 가능성을 보여준다. 그리고 계층적 강화 학습 프레임워크에서의 혼합 탐색 기법을 제시한다. 그 결과 혼합 탐색 기법이 균형잡힌

탐사와 탐험 덕분에 밀도가 낮은 보상을 주는 복잡한 환경에서 효과적인 에이전트를 찾아낸 다는 것을 보여준다.

감사의 글

관악에서 학문과 연구를 하며 많은 분의 도움을 받았습니다. 이 모든 분의 지도와 격려, 도움으로 무사히 학위를 마치게 되어 이 자리를 빌어 감사의 말씀을 드립니다. 우선, 저를 지도해주신 문병로 교수님께 감사의 말씀을 드립니다. 지도 교수님의 현명한 지도 덕분에 무사히 학업을 마치고 이렇게 학위를 받을 수 있었습니다. 또한 학위 논문 심사를 맡아주신 신영길 교수님, 오일석 교수님, 김용혁 교수님, 그리고 정순철 박사님께도 감사드립니다. 또한 저와 여러 연구와 논문 작성을 함께했던 김진현 박사님, 하성주 박사님, 하명훈 박사님께도 감사의 인사를 드립니다.

오랜 연구실 생활을 하며 많은 동료들을 만났고 즐거운 일도 있었고 함께 어려움을 헤쳐나간 일도 있었습니다. 연구실 초기에 학업을 함께했던 조승현, 권지훈의 동료들 덕에 빠르게 연구실에 적응할 수 있었습니다. 하성주 박사님, 하명훈 박사님, 그리고 윤한상 선배님과 같은 뛰어난 선배님들과 학문적으로 교류하며 연구자로서의 마음가짐을 다지게 되었습니다. 또한 프로젝트 진행을 함께하며 고생했던 엄승현, 육지은, 지승근, 장보규, 김창겸, 그리고 황순용 후배님들 덕에 많은 경험을 쌓고 어려움을 헤쳐나갈 능력을 기를 수 있었습니다. 그 외에도 연구실을 지나갔던 많은 선후배 동료들 덕에 이렇게 무사히 학업을 마칠 수 있었습니다. 또 오랜 학업기간동안 함께 어울리던 연구실 밖의 오랜 친구들에게도 감사를 드립니다.

또한 오랜 기간 저의 학업을 지원해주시고 믿어주신 저의 부모님께 깊은 감사를 드립니다. 부모님 덕분에 이렇게 학위를 받게 되었고 이 기쁨을 함께 나누었으면 합니다.