Policy Direct Search for Effective Reinforcement Learning

by

Yiming Peng

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Doctor of Philosophy
in Computer Science.

Victoria University of Wellington
2019

Abstract

Reinforcement Learning (RL) problems appear in diverse real-world applications and are gaining substantial attention in academia and industry. Policy Direct Search (PDS) is widely recognized as an effective approach to RL problems. However, existing PDS algorithms have some major limitations. First, many step-wise Policy Gradient Search (PGS) algorithms cannot effectively utilize informative historical gradients to accurately estimate policy gradients. Second, although evolutionary PDS algorithms do not rely on accurate policy gradient estimations and can explore learning environments effectively, they are not sample efficient at learning policies in the form of deep neural networks. Third, existing PGS algorithms often diverge easily due to the lack of reliable and flexible techniques for value function learning. Fourth, existing PGS algorithms have not provided suitable mechanisms to learn proper state features automatically.

To address these limitations, the overall goal of this thesis is to develop ***effective policy direct search algorithms*** for tackling challenging RL problems through technical innovations in four key areas. First, the thesis aims to improve the accuracy of policy gradient estimation by utilizing historical gradients through a Primal-Dual Approximation technique. Second, the thesis targets on surpassing the state-of-the-art performance by properly balancing the exploration-exploitation trade-off via Covariance Matrix Adaption Evolutionary Strategy (CMA-ES) and Proximal Policy Optimization (PPO). Third, the thesis seeks to stabilize value function learning via a self-organized Sandpile Model (SM) meanwhile generalize the compatible condition to support flexible value function learning. Fourth, the thesis endeavors to develop innovative evolutionary feature learning techniques that are capable of automatically extracting useful state features so as to enhance various cutting-edge PGS algorithms.

In the thesis, we explore the four key technical areas by studying policies with increasing complexity. First of all, we start the research from a simple linear policy representation, and then proceed to a complex neural network based policy representation. Next, we consider a more complicated situation where policy learning is coupled with a value function learning. Subsequently, we consider policies modeled as a concatenation of two interrelated networks, one for feature learning and one for action selection.

To achieve the first goal, this thesis proposes a new policy gradient learning framework where a series of historical gradients are jointly exploited to obtain accurate policy gradient estimations via the Primal-Dual Approximation technique. Under the framework, three new PGS algorithms for step-wise policy training have been derived from three widely used PGS algorithms; meanwhile, the convergence properties of these new algorithms have been theoretically analyzed. The empirical results on several benchmark control problems further show that the newly proposed algorithms can significantly outperform their base algorithms.

To achieve the second goal, this thesis develops a new sample efficient evolutionary deep policy optimization algorithm based on CMA-ES and PPO. The algorithm has a layer-wise learning mechanism to improve computational efficiency in comparison to CMA-ES. Additionally, it uses a performance lower bound based surrogate model for fitness evaluation to significantly reduce the sample cost to the state-of-the-art level. More importantly, the best policy found by CMA-ES at every generation is further improved by PPO to properly balance exploration and exploitation. The experimental results confirm that the proposed algorithm outperforms various cutting-edge algorithms on many benchmark continuous control problems.

To achieve the third goal, this thesis develops new value function learning methods that are both reliable and flexible so as to further enhance the effectiveness of policy gradient search. Two Actor-Critic (AC) algorithms have been successfully developed from a commonly-used PGS algorithm, i.e., Regular Actor-Critic (RAC). The first algorithm adopts SM to stabilize value function learning, and the second algorithm generalizes the logarithm function used by the compatible condition to provide a flexible family of new compatible functions. The experimental results show that, with the help of reliable and flexible value function learning, the newly developed algorithms are more effective than RAC on several benchmark control problems.

To achieve the fourth goal, this thesis develops innovative NeuroEvolution algorithms for automated feature learning to enhance various cutting-edge PGS algorithms. The newly developed algorithms not only can extract useful state features but also learn good policies. The experimental analysis demonstrates that the newly proposed algorithms can achieve better performance on large-scale RL problems in comparison to both well-known PGS algorithms and NeuroEvolution techniques. Our experiments also confirm that the state features learned by NeuroEvolution on one RL task can be easily transferred to boost learning performance on similar but different tasks.

List of Publications

The publications completed during my PhD period are listed below in chronological order.

1. **Y. Peng**, G. Chen, M. Zhang, and S. Pang. "Generalized Compatible Function Approximation for Policy Gradient Search," in *The 23rd International Conference on Neural Information Processing (ICONIP 2016)*, 2016.

2. **Y. Peng**, G. Chen, S. Holdaway, Y. Mei, and M. Zhang. "Automated State Feature Learning for Actor-Critic Reinforcement Learning through NEAT," in *The Genetic and Evolutionary Computation Conference (GECCO Companion 2017)*, 2017.

3. **Y. Peng**, G. Chen, M. Zhang, and Y. Mei. "Effective Policy Gradient Search for Reinforcement Learning through NEAT based Feature Extraction," in *Simulated Evolution and Learning - 11th International Conference (SEAL 2017)*, 2017.

4. W. Hardwick-Smith, **Y. Peng**, G. Chen, Y. Mei and M. Zhang. "Evolving Transferable Artificial Neural Networks for Gameplay Tasks via NEAT with Phased Searching," in *Australasian Conference on Artificial Intelligence 2017 (AI 2017)*, 2017.

5. **Y. Peng**, G. Chen, M. Zhang, and S. Pang. "A Sandpile Model for Reliable Actor-Critic Reinforcement Learning," *2017 International Joint Conference on Neural Networks (IJCNN 2017)*, 2017.

6. G. Chen, **Y. Peng**, and M. Zhang. "Constrained Expectation-Maximization Methods for Effective Reinforcement Learning," *2018 International Joint*

Conference on Neural Networks (IJCNN 2018), 2018.

7. **Y. Peng**, G. Chen, H. Singh, and M. Zhang. "NEAT for Large-Scale Reinforcement Learning through Evolutionary Feature Learning and Policy Gradient Search," in *The Genetic and Evolutionary Computation Conference (GECCO 2018)*, 2018.

8. **Y. Peng**, G. Chen, and M. Zhang. "Proximal Evolutionary Strategies: Achieving State-of-the-art Deep Reinforcement Learning through Evolutionary Policy Optimization," 2018. (Submitted for review to the journal "IEEE Transaction on Evolutionary Computation")

9. **Y. Peng**, G. Chen, and M. Zhang, "Primal-dual Sub-Gradient Approximation based Policy Gradient Search," 2019. (To be submitted for review to the journal "Machine Learning")

10. G. Chen, **Y. Peng**, and M. Zhang. "An Adaptive Clipping Approach for Proximal Policy Optimization," arXiv:1804.06461, 2018.

11. G. Chen, **Y. Peng**, and M. Zhang. "Effective Exploration for Deep Reinforcement Learning via Bootstrapped Q-Ensembles with Tsallis Entropy Regularization," arXiv:1809.00403, 2018.

Acknowledgments

There are several people I would like to thank because without them this thesis would have gone nowhere and it would not have turned out the same.

First and foremost, my deepest gratitude goes to my supervisors Dr. Aaron Chen and Prof. Mengjie Zhang. Dr. Aaron Chen has supported me throughout my studies in many ways. Without his encouragement and help, realizing this work would have been much harder, if not impossible. He kept me motivated and focused, patiently answered many my naive questions, suggested various significant improvements or viable alternatives to my works. Aaron's wealth of knowledge in many aspects of machine learning and his excellent eye for details have helped shape this research into something far beyond what I could have ever done alone. My Mom alway says that Aaron acts more like an elder brother to me rather than a supervisor, which I can't agree more. Prof. Mengjie Zhang is a source of priceless advice and inspiration, who always provided substantial support to me in various ways during my Ph.D. journey. Thanks for his helpful comments and sound technical advice whenever they were necessary. Notably, his endless patience in correcting my less-than-perfect English as well as valuable suggestions for improving the thesis was worth his weight in gold.

Furthermore, I am deeply grateful to my beautiful and lovely wife, Dr. Bing Xue, for willing to marry me. She is always a constant source of love, support, amusement, and motivations. Sometimes, she was just like an extra supervisor who guided me towards the final step of the thesis and, more importantly, indicated always the right direction for my life journey. Words are just powerless at this moment to express how important to me she is, I just love her $\texttt{very}$ ($\texttt{very} \rightarrow \infty$) much. Also, I would like to immensely thank my parents as well as my brother's family whose unconditional support, love and understanding

throughout my life have always motivated me to work hard and pursue my goals. I also wish to send my thanks to my aunt in law, my parents in law and my brothers in law for being so supportive to me, more importantly being such an excellent family to let Bing grow up.

I also want to thank my dear friends for their friendship, their support, and a generally great time to relax me from the stressful Ph.D. life. I especially thank Jason Hu for his yummy dishes and warm encouragement which supported me getting through the difficult and lonely time. Also, I am grateful to Qi Chen, Jianyu Zhang, Shirley Chen, Yi Mei, Xiaohan Chen, Guanghui Li, Sharon Gao, Linda Zhang, Ivy Liu, and Sandy Liu for support, sharing ideas, thought-provoking discussions, delicious food and funny moments when being with them. I also wish to thank my lab buddies Tony Zhang, Chen Wang, Victoria Huang, and Hang Yu for the useful discussions and the interesting chitchats. Another big thank goes to my Auckland friends Lin Yuan, Ian Yu, Peter Zhang, Angelina Chai-Rodgers, Lei Zhu, Lei Song, Echo Peng, and Li You who have ever helped and supported me in no matter what way along this journey. In addition, I would love to thank all the friends and colleagues in the ECRG group for the great suggestions and useful comments for refining my research during the past three years.

Last but not least, I want to thank our school ECS, particularly the ECRG group, for being such an awesome environment for research and study. In particular, I want to thank our subgroups, ECCO and FASLIP, for the fruitful research discussions, informative knowledge sharing, and brilliant brainstorms which inspired me a lot to complete this thesis. Additionally, I wish to thank the High Performance Computing facilities provided New Zealand eScience Infrastructure (NeSI), which ensure the success of those computationally expensive experiments required by the development of this thesis.

Finally, I acknowledge the financial support toward the completion of this Ph.D. from Victoria Doctoral Scholarship and Thesis Submission Scholarship provided by Victoria University of Wellington.

Contents

1 Introduction **1**

1.1 Background 1
1.2 Motivations 5
1.3 Goals 10
1.4 Major Contributions 11
1.5 Organization of Thesis 19

2 Literature Review **23**

2.1 Background 24
2.1.1 Machine Learning 24
2.1.2 Reinforcement Learning 25
2.1.3 Evolutionary Computation 31
2.1.4 Feature Learning 36
2.2 Reinforcement Learning Methods 37
2.2.1 Reinforcement Learning Methods Taxonomy 38
2.2.2 Value Function Indirect Search 38
2.2.3 Discussion on Value Function Indirect Search 44
2.2.4 Policy Direct Search 47
2.2.5 Model-free Gradient-based Policy Direct Search 50
2.2.6 Model-free Gradient-free Policy Direct Search 61
2.2.7 Discussion on Policy Direct Search 65
2.2.8 Feature Learning in Reinforcement Learning 68
2.2.9 Discussion on Feature Learning in Reinforcement Learning 70
2.3 Related Work 72

2.3.1 Effective Policy Direct Search through Primal Dual Approximation . . . 72
2.3.2 Proximal Evolutionary Strategies for Sample Efficient Policy Direct Search . . . 74
2.3.3 Reliable and Flexible Value Function Learning for Policy Direct Search . . . 75
2.3.4 Enhancing Policy Direct Search via Automated Evolutionary Feature Learning . . . 77
2.4 Chapter Summary . . . 79

3 Experimental Methodology 83
3.1 Benchmark Problems . . . 83
3.2 Statistical Treatment . . . 87
3.2.1 General Experiment Setup . . . 87
3.2.2 Statistical Methods . . . 88

4 Effective Policy Direct Search through Primal-Dual Approximation 91
4.1 Introduction . . . 92
4.1.1 Chapter Goals . . . 93
4.1.2 Chapter Organization . . . 94
4.2 Preliminaries — A General Primal-Dual Approximation Method 94
4.3 The Proposed Algorithms . . . 96
4.3.1 General Dual Formulation for Policy Gradient Search . . . 97
4.3.2 Dual Regular Gradient Actor Critic Algorithm . . . 99
4.3.3 Dual Natural Gradient Actor Critic with Fisher Information Matrix . . . 103
4.3.4 Dual Natural Gradient Actor Critic with Advantage Parameters . . . 105
4.4 Theoretical Analysis . . . 107
4.4.1 Learning as Multi Time-Scale Stochastic Approximation . 108
4.4.2 Convergence Analysis . . . 111
4.5 Design of Experiments . . . 114
4.5.1 Experiment Setup . . . 114

4.5.2 Experiment Design 117
4.6 Results and Discussion 119
4.6.1 Discussion on Results of Bipedal Walker 119
4.6.2 Discussion on Results of Lunar Lander 122
4.6.3 Discussion on Results of Mountain Car Continuous 122
4.6.4 Discussion on Results of Inverted Pendulum 124
4.6.5 Discussion on Results of Inverted Double Pendulum . . . 125
4.6.6 Discussion on Results of Inverted Pendulum Swingup . . 125
4.6.7 Result Summary 125
4.7 Chapter Summary 128

5 Proximal Evolutionary Strategies for Sample Efficient Policy Direct Search 131
5.1 Introduction 132
5.1.1 Chapter Goals 133
5.1.2 Chapter Organization 135
5.2 The Proposed Algorithm — Proximal Evolutionary Strategy . . . 136
5.2.1 PES-S1: Layer-wise Learning 136
5.2.2 PES-S2: Surrogate Model Based Learning 142
5.2.3 PES-S3: Local Search Enhanced Learning 147
5.2.4 Key Characteristics of Proximal Evolutionary Strategy . . 150
5.3 Design of Experiments 152
5.3.1 Experiment Setup 153
5.3.2 Network Architecture 153
5.3.3 Hyper-Parameter Configurations 154
5.3.4 Evaluation Criteria 156
5.3.5 Experiment Design 156
5.4 Results and Discussion 157
5.4.1 Results of Experiment (A1) 157
5.4.2 Results of Experiment (A2) 159
5.4.3 Results of Experiment (A3) 162
5.5 Chapter Summary 165

6 Reliable and Flexible Value Function Learning for Policy Direct Search 167
6.1 Introduction . . . 169
6.1.1 Chapter Goals . . . 170
6.1.2 Chapter Organization . . . 171
6.2 Preliminaries . . . 171
6.2.1 Regular Actor-Critic Algorithm . . . 171
6.2.2 Sandpile Model . . . 172
6.3 The Proposed Algorithms — SM-RAC and GCFA-RAC . . . 174
6.3.1 Sandpile Model based Regular Actor-Critic (SM-RAC) . . . 174
6.3.2 Generalized Compatible Function Approximation base Regular Actor-Critic (GCFA-RAC) . . . 178
6.4 Design of Experiments . . . 179
6.4.1 Experiments on SM-RAC . . . 180
6.4.2 Experiment on GCFA-RAC . . . 184
6.5 Results and Discussion . . . 185
6.5.1 Discussion on Results of SM-RAC . . . 186
6.5.2 Discussion on Results of GCFA-RAC . . . 191
6.6 Chapter Summary . . . 194

7 Enhancing Policy Direct Search via Automated Evolutionary Feature Learning 197
7.1 Introduction . . . 199
7.1.1 Chapter Goals . . . 199
7.1.2 Chapter Organization . . . 200
7.2 The Proposed Algorithms — NEAT+RAC and NEAT+PGS . . . 201
7.2.1 NEAT based Feature Learning enhanced Regular Actor-Critic (NEAT+RAC) . . . 201
7.2.2 NEAT based Feature Learning enhanced Policy Gradient Search (NEAT+PGS) . . . 205
7.3 Design of Experiments . . . 211
7.3.1 Experiments on NEAT+RAC . . . 212
7.3.2 Experiments on NEAT+PGS . . . 214
7.4 Results and Discussion . . . 218

7.4.1 Discussion on Results of NEAT+RAC 218
7.4.2 Discussion on Results of NEAT+PGS 222
7.5 Chapter Summary 225

8 Conclusions **229**
8.1 Major Conclusions 230
8.1.1 Effective Policy Direct Search through Primal-Dual Approximation 231
8.1.2 Proximal Evolutionary Strategy for Sample Efficient Policy Direct Search 232
8.1.3 Reliable and Flexible Value Function Learning for Policy Gradient Search 232
8.1.4 Enhancing Policy Direct Search via Automated Evolutionary Feature Learning 233
8.2 Limitations 234
8.2.1 Manual Network Architecture 235
8.2.2 Trial-and-Error based Hyper-parameter Tunning 235
8.2.3 Pre-defined Reward Settings 235
8.3 Future Work 236
8.3.1 Combining Improvements for Policy Direct Search 236
8.3.2 Model-based vs. Model-free 237
8.3.3 Transfer Learning for Policy Direct Search 238
8.3.4 Automated Network Architecture Design 239
8.3.5 Automated Hyper-parameter Tunning 240
8.3.6 Cooperative Co-evolution for Policy Direct Search 240

List of Tables

4.1 The Hyper-parameter settings of all algorithms including RAC, NACF, NACA, NACAF, Dual-RAC, Dual-NACF, Dual-NACA, ARS and PPO-Linear used for all problems. 118

4.2 The final episode performance comparison of nine algorithms (i.e., ARS, Dual-RAC, Dual-NACF, Dual-NACA, RAC, NACF, NACA, NACAF, and PPO-Linear) on six benchmark problems (i.e., Bipedal Walker, Inverted Double Pendulum, Inverted Pendulum, Inverted Pendulum Swingup, Lunar Lander Continuous, and Mountain Car Continuous). 128

5.1 Hyper-parameter configurations of all candidate algorithms: CMA-ES, PES-S1, PES-S2, PES, OpenAI-ES, Uber-GA, TPRO, PPO, and ACKTR. 155

5.2 The final episode performance comparison of two algorithms (i.e., CMA-ES and PES-S1) on ten benchmark problems (i.e., Bipedal Walker, BipedalWalkerHardcore, HalfCheetah, Hopper, Inverted Double Pendulum, Inverted Pendulum, Inverted Pendulum Swingup, Lunar Lander Continuous, Reacher, and Walker2D). 159

5.3 The final episode performance comparison of four algorithms (i.e., CMA-ES, OpenAI-ES, Uber-GA and PES-S2) on ten benchmark problems (i.e., Bipedal Walker, BipedalWalkerHardcore, HalfCheetah, Hopper, Inverted Double Pendulum, Inverted Pendulum, Inverted Pendulum Swingup, Lunar Lander Continuous, Reacher, and Walker2D). 161

5.4 The final episode performance comparison of seven algorithms (i.e., PES, PES-S2, OpenAI-ES, Uber-GA, PPO, TRPO, and ACKTR) on ten benchmark problems (i.e., Bipedal Walker, BipedalWalkerHardcore, HalfCheetah, Hopper, Inverted Double Pendulum, Inverted Pendulum, Inverted Pendulum Swingup, Lunar Lander Continuous, Reacher, and Walker2D). 163

6.1 The hyper-parameter configurations for experiments of RAC and SM-RAC on the Puddle World problem and the Mountain Car problem. 183
6.2 The hyper-parameter configurations for experiments of RAC on the Puddle World problem, the Mountain Car problem, the Cart Pole problem, and the Heating Coil problem. 184
6.3 Experiment Common Settings for One Trial. 185
6.4 The final episode performance comparison of different ν values (i.e., 0.5, 0.7, 0.9, 1.0, 1.1, 1.5, 2.0) on three benchmark problems (i.e., Cart Pole, Puddle World and Heating Coil). 195

7.1 The meta-parameter settings for NEAT+RAC across all experiments. 213
7.2 Topology Setups for Policy Networks and Feature Networks. . . . 215
7.3 Hyper-parameter Configurations for PGS algorithms. 216
7.4 Hyper-parameter Configurations for NEAT. 216
7.5 The final episode performance comparison of two algorithms (i.e., NEAT, and NEAT-RAC-PGS) on two benchmark problems (i.e., Cart Pole and Mountain Car). 218
7.6 The final episode performance comparison of two algorithms (i.e., NEAT, and NEAT-RAC-PGS) on two benchmark problems (i.e., Cart Pole and Mountain Car). 220
7.7 The final episode performance comparison of seven algorithms (i.e., NEAT, NEAT+A2C, NEAT+POWER, NEAT+TRPO, A2C, POWER and TRPO) on six Atari games (i.e., Asteroids, Breakout, Freeway, Seaquest, SpaceInvaders, and TimePilot). 222

7.8 Sample Efficiency comparison of NEAT+PGS against NEAT, A2C, POWER and TRPO. 224

List of Figures

2.1 A Reinforcement Learning problem is formally modeled as an MDP, drawn based on Figure 3.1 in [212]. 26

2.2 Value Function Indirect Search Framework, drawn based on Figure 3 in [193]. 39

2.3 A generic feed-forward NN with four input units, two output units, and two hidden layers, adapted from Figure 9.14 [212]. . . . 42

2.4 Categorization for Policy Direct Search, adapted from Figure 1.1 [52]. 47

2.5 Model-free Policy Direct Search Framework, prepared based on Figure 4 in [193]. 48

2.6 Model-based Policy Direct Search Framework, prepared based on Figure 4 in [193], Figure 3.1 in [52] and Algorithm 1 in [51]. 49

2.7 Actor-Critic Architecture, adapted from Figure 6.15 in [212]. . . . 54

4.1 The Primal-Dual Approximation based Actor-Critic Algorithms. . 98

4.2 An example of the biased learning following (4.11) when t is very large. 102

4.3 The Architecture of NN for representing Value Function for Dual-RAC, Dual-NACF, Dual-NACA, RAC, NACF, NACA, NACAF, and PPO-Linear. 116

4.4 A performance comparison of the proposed dual algorithms including Dual-RAC, Dual-NACF, Dual-NACA against the competing algorithms including RAC, NACF, NACA, NACAF, ARS [146] and PPO-Linear [191] on the **Bipedal Walker** problem. 120

4.5 A performance comparison of the proposed dual algorithms including Dual-RAC, Dual-NACF, Dual-NACA against the competing algorithms including RAC, NACF, NACA, NACAF, ARS [146] and PPO-Linear [191] on the **Lunar Lander** problem. . 121

4.6 A performance comparison of the proposed dual algorithms including Dual-RAC, Dual-NACF, Dual-NACA against the competing algorithms including RAC, NACF, NACA, NACAF, ARS [146] and PPO-Linear [191] on the **Mountain Car Continuous** problem. 123

4.7 A performance comparison of the proposed dual algorithms including Dual-RAC, Dual-NACF, Dual-NACA against the competing algorithms including RAC, NACF, NACA, NACAF, ARS [146] and PPO-Linear [191] on the **Inverted Pendulum** problem. 124

4.8 A performance comparison of the proposed dual algorithms including Dual-RAC, Dual-NACF, Dual-NACA against the competing algorithms including RAC, NACF, NACA, NACAF, ARS [146] and PPO-Linear [191] on the **Inverted Double Pendulum** problem. 126

4.9 A performance comparison of the proposed dual algorithms including Dual-RAC, Dual-NACF, Dual-NACA against the competing algorithms including RAC, NACF, NACA, NACAF, ARS [146] and PPO-Linear [191] on the **Inverted Pendulum Swingup** problem. 127

5.1 The layer-wise training process via CMA-ES for a three-layer policy network, where all weights and biases for one layer are included in the process. 138

5.2 The layer-wise training process via CMA-ES for a three-layer policy network, where only a proportion of weights and biases uniformly selected at random are included in the process. 140

5.3 The surrogate model based learning process for a three-layer value function network and a three-layer policy network, where value function network is trained via gradient ascent and policy network is layer-wisely trained via CMA-ES (see Section 5.2.1). . . 143

5.4 The local search enhanced learning process for a three-layer value function network and a three-layer policy network, where a value function resilience re-training process and a PPO driven local search process are added on the top of the surrogate model based learning developed in Section 5.2.2. 148

5.5 The Architecture of DNN for CMA-ES, PES-S1, OpenAI-ES, and Uber-GA. 154

5.6 The Architecture of DNN for PES-S2, PES, PPO, TRPO, and ACKTR. 155

5.7 A comparison of average total rewards over running time (in seconds) obtained by PES-S1 and CMA-ES [88] on the ten benchmark control problems. 158

5.8 A comparison of average total rewards per 10,000 samples (5,000,000 samples in total) obtained by PES-S2, CMA-ES [88], OpenAI-ES [186], and Uber-GA [210] on the ten benchmark control problems. 160

5.9 A comparison of average total rewards per 10,000 samples obtained by PES (i.e., PES-S3), PES-S2, OpenAI-ES [186], Uber-GA [210], TRPO [189], ACKTR [240] and PPO [191] on ten control problems over total 5,000,000 samples. 162

6.1 The triangle basis function used for defining one single dimension of the state input. 182

6.2 Average of the absolute values generated from the value function learned by RAC and SM-RAC at every 50 episodes on the Puddle World problem. 187

6.3 Average cumulative rewards obtained by RAC and SM-RAC at every 50 episodes on the Puddle World problem. 187

6.4 Correlation between the learning effectiveness and the learning reliability of RAC on the Puddle World problem. 188

6.5 Average of the absolute values generated from the value function learned by RAC and SM-RAC at every 50 episodes on the Mountain Car problem. 189

6.6 Average value function learned by RAC and SM-RAC at every 50 episodes on the Mountain Car problem. 190

6.7 Correlation between the learning effectiveness and the learning reliability of RAC on the Mountain Car problem. 190

6.8 Average steps to the goal region of the Puddle World problem ($\nu = 1.0$ is the original RAC). 192

6.9 Average ξ on the Cart Pole problem ($\nu = 1.0$ is the original RAC). 192

6.10 Average balancing steps on the Cart Pole problem ($\nu = 1.0$ is the original RAC). 193

6.11 The average error (i.e., average deviation) between the output temperature T_{ao} and the target temperature T_d ($\nu = 1.0$ is the original RAC) on the Heating-Coil problem. 194

7.1 The overall design of NEAT+RAC. 202

7.2 An Overview on the NN Architecture of NEAT+PGS. 207

7.3 The comparison of learning performance of NEAT+RAC and NEAT on two benchmark problems: (a) displays the averaging steps to reach the goal region on Mountain Car (the smaller the better), (b) displays the averaging steps to balance the pole to the upright position on Cart Pole (the larger the better). 219

7.4 The comparison of learning performance of RAC with two different feature extractors (an evolved NN feature extractor and a predefined discretized feature extractor) on the two related problems (see Section 7.3.1): (a) displays learning performances obtained on the standard Cart Pole problem, (b) displays learning performances obtained on the modified Cart Pole problem. 220

7.5 Average rewards per 10,000 steps obtained by NEAT, NEAT+A2C, NEAT+POWER, NEAT+TRPO, A2C, POWER and TRPO on six Atari games, including Asteroids, Breakout, Freeway, Seaquest, SpaceInvaders, and TimePilot. As being highlighted with red color, for NEAT+PGS, the NEAT based feature learning stage stops at two million steps (i.e., 2,000,000 samples). 223

List of Algorithms

4.3.1 Dual-RAC Algorithm 103
4.3.2 Dual-NACF Algorithm 104
4.3.3 Dual-NACA Algorithm 106
5.2.1 Population Initialization in PES 141
5.2.2 Proximal Evolutionary Strategy 149
5.2.3 Experience Sampling 150
5.2.4 Rollout Function 151
6.3.1 Sandpile Model based Regular Actor-Critic (SM-RAC) 177
6.3.2 Generalized Compatible Function Approximation based Regular Actor-Critic Algorithm (GCFA-RAC) 180
7.2.1 NEAT+RAC Initialization 203
7.2.2 NEAT+RAC Evolution 204
7.2.3 NEAT+RAC Evaluation 206
7.2.4 NEAT+RAC Algorithm 207
7.2.5 NEAT+PGS 210
7.2.6 Policy Gradient Search 210
7.2.7 NEAT Feature Learning 211

Chapter 1

Introduction

The chapter starts with an introduction to the general background of Reinforcement Learning, then proceeds to the motivations, the research goals, the major contributions and the organization of the thesis.

1.1 Background

Sequential decision making plays a vital role in various real-world problems, such as budget setting [77, 156], baseball pitching [196], network monitoring [93], playing board games [197, 199], intelligent gameplay [155, 242], robot control [80, 209], self-driving cars [187], dynamic tasks scheduling [72], operational research [1, 177], and human-computer interactions [104, 201]. An important abstraction of these problems is known widely as the ***Reinforcement Learning (RL) problem*** [136, 110, 153, 212, 236]. In general, an RL problem is described through a scenario where an autonomous agent acts sequentially within an unknown environment. The action performed by the agent influences the state of the environment. Meanwhile, the agent receives feedback in terms of both immediate reward and the total rewards as a consequence of its action. The goal of the agent is to wisely choose actions to maximize the total rewards in a long term [136, 153, 212, 236]. It is challenging to build ***effective*** algorithms to guide the agent to achieve the goal as discussed in the literature [136, 153, 212, 236].

Generally, there are two main categories of approaches to addressing the RL

problems, i.e., *search and planning* [41, 24, 28, 35, 102, 148] and *Machine Learning (ML)* [136, 212, 236]. For the search and planning techniques to be successful, the environment model (i.e., system dynamics) must be known in advance [136, 41, 24]. However, it is often impractical to obtain a precise environment model in advance [136, 212, 236]. In addition, search and planning are also vulnerable when system dynamics change or uncertainties exist in the environment [212, 236].

In contrast to search and planning, ML avoids the necessity of knowing the environment model in advance [212, 153, 236, 185]. Also, uncertainties can be addressed by learning directly from trial-and-error interactions [212, 236]. Generally, ML is categorized into three major paradigms, namely *Supervised Learning*, *Unsupervised Learning*, and *Reinforcement Learning*. The suitability of each paradigm for solving RL problems is discussed below.

Supervised learning seeks to identify important relations among given inputs sampled from a fixed distribution [153, 185, 195, 147]. This paradigm is suitable for problems like classification and regression, but not applicable to RL problems where agents have to learn from interactions. In such interactive problems, it is often impractical to obtain examples labeled with desired behaviors. Given this, an agent must be capable of learning from its own experience.

Unsupervised learning aims to find certain patterns that frequently occur among the given data inputs [153, 185, 147, 3]. Clustering or visualization are primary applications of unsupervised learning algorithms [153]. They are inappropriate for solving RL problems because they do not receive any feedback from the environment which is essential for decision making in interactive problems.

Different from supervised learning and unsupervised learning, RL agents interact directly with an unknown environment through observing states and then taking actions; after taking each action, the learner shall receive a resulting reward. Hereby, the goal of RL is to learn a policy [1] by using the feedback from the environment, so as to maximize the long-term pay-off defined in the

[1] A policy defines the learning agent's way of behaving at a given time. More specifically, a policy is a mapping from observed states of the environment to actions to be taken when in those states [212]. Please refer to Section 2.1.2 for details.

RL problems [136, 212, 153, 185, 195, 147]. In this sense, ***Reinforcement learning is naturally suitable for learning to make correct decisions from direct interactions*** [136, 212]. Several important reasons are given below:

(1) Different from search and planning, an RL agent is capable of learning to accommodate uncertainties via interactions in an unknown environment as demonstrated in many practical applications [212, 236, 185, 195, 147].

(2) Different from unsupervised learning, RL is capable of using the feedback from the environment to learn to solve challenging RL problems [236, 147].

(3) Different from supervised learning, an RL agent does not rely on any external instructors (the environment is the only source of experiences). Moreover, supervised learning treats the decisions as completely independent events, whereas RL treats them as dependent events. So RL is more suitable to solve problems such as playing chess or robot navigation, as success depends on a sequence of interrelated decisions [212, 236, 195].

To date a great number of algorithms have been proposed to solve complex RL problems. We can largely divide them into two types [236, 218]: *Value Function Indirect Search* (VIS) [212, 217, 184, 214, 213, 155, 225, 230, 24, 28, 15, 116] and *Policy Direct Search* (PDS) [21, 2, 173, 215, 31, 238, 192, 117, 119, 168, 20, 189, 191, 154, 229, 198, 49, 50, 53]. The following will present high-level introduction of the two types respectively.

VIS algorithms rely on value functions which are used to measure the expected long-term pay-off obtainable by an agent who starts its journey from any given state. Through value functions, the agent can **indirectly** define policies that guide action selection in the environment [195]. Typical VIS algorithms mostly fall into the *Temporal Difference learning* scheme [212], including TD(λ) learning [212, 217], Q-learning [231], State-Action-Reward-State-Action (SARSA) [184], Gradient Temporal Difference (GTD) [214], GTD2 [213], and several modern variations of Q-Learning [155, 225, 230].

Although these indirect learning algorithms contribute significantly to RL research, there are still many challenges: **(1)** VIS algorithms, in comparison to PDS algorithms, may be less suitable on problems where policy is simpler to

learn than value functions [24, 28, 15, 116]. For example, as reported by Simsek et al. [200], one can play well in Tetris game without knowing any evaluation functions but by simply choosing from all the available actions. In such a case, PDS algorithms can learn faster and yield better policies than VIS algorithms [200, 212]. **(2)** The majority of VIS algorithms are designed to cope only with discrete states and discrete actions. However many real-world RL problems, in particular, control and robot locomotion problems, demand for an RL algorithm that is capable of handling continuous states and actions effectively [52, 215]. **(3)** VIS algorithms have no theoretical convergence guarantee for implicit represented policies [212, 121]. For example, when being used with VIS algorithms, the ϵ-greedy action selection mechanism may dramatically change the action probabilities arbitrarily for a small change in value function estimations [212]. **(4)** Most of VIS algorithms can only learn in Markovian environments due to the problem of "incomplete perception" [212, 15].

PDS algorithms aim to **directly** search the optimal policy to solve RL problems. For this purpose, the policy is often presented as a parametric model. Moreover, the value function can be explicitly used to evaluate the policy [183, 21, 125, 89]. Either by using the gradient-based or gradient-free techniques, we can search in the policy space to maximize the expected long-term pay-off [212, 236, 195, 21]. So far, various successful PDS methods have been brought onto the stage [21, 2, 173, 215, 31, 238, 192, 117, 119, 168, 20, 189, 191, 154, 229, 198, 49, 50, 53].

PDS algorithms are considered a promising approach to overcoming the limitations faced by VIS [183, 21, 125, 89, 215, 31]. Firstly, they can scale well and perform effectively in large state and action space, because the search is directly performed on the parametric policy space whose dimensionality can be much smaller [173, 31, 25, 247, 168]. Secondly, they are naturally suitable for continuous state and action space [173, 31, 25, 247, 168]. Thirdly, the evidence of strong convergence guarantees has already been shown in many works [122, 215, 31], and such guarantees can also be made available to some algorithms developed in this thesis. Last but not least, they are suitable for the non-Markovian environments, since we can effectively learn the policy by using only partially observable information [173, 31, 160]. Furthermore, they can make better use of

every single sample collected by interacting with an environment to achieve a certain level of effectiveness in comparison to VIS algorithms [229, 212].

Due to the key advantages mentioned above, PDS algorithms for RL become a very hot research topic recently, as many new works are being conducted to constantly push state-of-the-art performance to the next level [173, 31, 25, 247, 168, 183, 21, 125, 89, 215, 189, 191, 229, 135, 45]. However, there are still many open challenges in PDS for RL, such as (1) how to obtain more accurate policy gradient estimations for effective Policy Gradient Search (PGS), (2) how to improve learning effectiveness with reasonable sample efficiency for evolutionary PDS algorithms, (3) how to stabilize value function learning to facilitate policy learning for PGS algorithms, and (4) how to automatically learn useful state features to improve learning effectiveness of PDS algorithms. All these challenges are worthy of substantial studies, which will be the focus of this thesis.

1.2 Motivations

Most of existing RL applications are either extremely computational costly or highly dependent on VIS approaches that require a large number of samples. For example, the distributed version of AlphaGo requires 1202 CPUs, 176 GPUs, more than 100 human experts, and 30 million distinct positions sampled from different games to become the world champaign in Go game [197]. Moreover, Deep Q Network (DQN) [155] and many of its variations can achieve comparable human-level performances on playing 49 challenging Atari games. However, they require many weeks of intensive Neural Network (NN) training and millions of environment samples (e.g., video frames collected from Atari games) [155]. On the other hand, in comparison to VIS algorithms, PDS algorithms have shown higher performance and less sample cost on problems such as Atari games or continuous control tasks as evidenced in many works [189, 229, 80, 191, 240]. However, the practical use of PDS demands for significant improvements in both the learning effectiveness and the sample efficiency [52, 12, 189, 229, 80, 191, 240].

In view of this understanding, we aim at conducting studies to improve

effectiveness both practically and theoretically for PDS to address challenging RL problems. Driven by this goal, the principal research question that set the central theme of this thesis is:

> **How can the effectiveness of policy direct search algorithms be significantly improved in order to tackle difficult Reinforcement Learning problems?**

In this thesis, the definition of **effectiveness** under the context of RL is specified below:

*The **effectiveness** refers to the agent's ability to gain the maximum long-term payoff by consuming fewer environment samples.*

Our principal research question is further decomposed into four subquestions. We arrive at these questions detailed below through extensive review of numerous related works. According to our research, they are vital for PDS to solve many RL problems *effectively*.

To ensure a smooth start of the research for this thesis, we decide to focus on step-wise PGS [2] algorithms for training linear policies. This is because step learning strategy [3](i.e., step-wise learning) is the most common and long-lived learning strategy for PDS where learning occurs at every time step immediately after observing an environment sample. Also, a linear policy is the most basic form for policy representation.

In this context, to enhance the learning effectiveness of PGS, a possible way is to improve the accuracy of policy gradient estimation by utilizing multiple historical gradients for existing PGS algorithms. The majorities of step learning strategy based PGS algorithms [21, 2, 173, 215, 31, 238, 192, 117, 168, 20] do not preserve the gradients after every step of learning, which means that the current step gradient will be simply discarded with the assumption that it is no longer useful. However, the information of previous gradients can be better utilized to enhance the effectiveness of the learning process [62, 158]. For example, the

[2]PGS is one typical branch of PDS algorithms where gradient descent technique is applied for policy search.

[3]Please refer to Section 2.2.4 for details.

Adagrad algorithm [62] accumulates a series of consecutive historical gradients to accurately estimate gradients required for future learning. Driven by this understanding, we have formulated the first research question as in ***Q(1)*** below, which also leads to the first research objective ***O(1)*** in Section 1.3.

Q(1)*: How can we estimate policy gradient accurately so as to improve the effectiveness of step learning strategy based policy gradient search by using historical gradients*?

While conducting research toward answering **Q(1)**, we have found that stepwise PGS algorithms with linear policies have difficulties to learn effectively on several complicated continuous control problems, such as Hopper [189], HalfCheetah [220], and Walker2D [220]. The control signals for these problems are often high-dimensional which require more sophisticated policy representations, such as NNs [189]. In addition, gradient-based methods [4] perform less effective exploration in comparison to gradient-free methods (e.g., Evolutionary Algorithms (EAs)), and hence may be easily trapped to local optima [210, 186, 45]. Driven by this understanding, we have decided to place our second research focus on gradient-free PDS algorithms with deep structured policies.

Particularly, we have explored a key issue that is vital for improving existing EAs based PDS algorithms. The gist is to properly balance the exploration-exploitation trade-off by combining and unleashing the advantages of both EAs (suitable for exploration) and PGS (suitable for exploitation) for Deep Reinforcement Learning (DRL). Recent researches found that EAs can be competitive alternatives to cutting-edge PGS algorithms, because EAs can explore more effectively [186, 210, 101, 45]. On the other hand, PGS algorithms show clear advantages in exploitation. In practice, existing EAs for DRL remain distant from state-of-the-art performance in comparison to cutting-edge PGS algorithms, such as Trust Region Policy Optimization (TRPO) [189], Proximal Policy Optimization (PPO) [191], and Actor-Critic using Kronecker-Factored Trust Region (ACKTR) [240]. Motivated by this understanding, we set up the second

[4] Please refer to Section 2.2.4 for more details.

research question ***Q(2)*** below as the basis of the second research objective ***O(2)*** in Section 1.3.

Q(2)*: How can we surpass state-of-the-art performance in Evolutionary Algorithms by properly balancing the exploration-exploitation trade-off through adopting and improving off-policy and gradient-based training techniques developed by PDS algorithms*?

When conducting research to answer **Q(2)**, we have discovered that the learning effectiveness of Actor-Critic (AC) algorithms (i.e., an important family of PGS algorithms) relies heavily on value function learning, which is consistent with the findings reported in the literature [215, 190, 212]. In line with this finding, we performed studies on the third possible way to achieve effective PGS through developing innovative techniques for reliable and flexible learning of value functions in AC algorithms.

Note that classical PGS algorithms, such as REINFORCE [238], use the empirical total rewards to approximate policy gradients, which often leads to high approximation variances [143, 52]. A good way [122, 69, 52] to reduce estimation variances (or errors) is to learn a value function to replace empirical total rewards in REINFORCE. As a result, value function learning and policy learning are tightly coupled together. Thus, the reliability and accuracy of value function learning can have a significant impact on the performance of the policy learning. Several existing works have attempted to achieve reliable value function learning, but none has been satisfactory for wide adoption. For example, several methods (e.g., GTD [214] and GTD2 [213]) have adopted off-policy training to achieve reliable value function learning, but the learning process can still diverge in reality [214, 213, 49]. More specifically, we find that the reliability of critic learning will deteriorate abruptly whenever the predicted rewards by the value function fall outside the maximum/minimum possible cumulative rewards obtainable from any state in a learning environment. Such maximum/minimum possible cumulative rewards are problem-specific but can often be determined easily. Inspired by this finding, we decide to stabilize value function learning based on a Sandpile Model (SM) [18] with a self-organizing property. With the help of the property, we can self-organize value function

learning, effectively preventing learned value functions from diverging. Furthermore, inspired by the Policy Gradient Theorem (PGT) [5] [215], researchers have studied different forms of value functions that are compatible with policy parametrization. Nevertheless, to the best of our knowledge, very few works have studied the possible generalization of the compatible condition introduced in [215]. Such generalization can bring extra degrees of freedom for the compatible function (i.e., action-value function approximation [6]), which can help achieve more accurate value function learning. In view of this understanding, we form the third research question ***Q(3)***, which gives rise to the third objective ***O(3)*** in Section 1.3.

Q(3): *How can we enable a more reliable and flexible value function learning to enhance policy gradient search for tackling difficult RL problems via SandPile model and the generalization of compatible conditions?*

In-depth research in the thesis showed that the state features play a paramount role for effective RL. Because in PGS, both value function and policy are represented as parametric functions with respect to the state features [153, 26]. As a result, the effectiveness of PGS is heavily dependent on the quality of state features.

In line with this understanding, we have identified the fourth possible way to develop innovative feature-learning techniques to extract useful state/environment information required for effective PDS. Traditionally, state features that form the input to a policy are determined by human experts through a purely manual process, which has two problems. Firstly, it is tedious, error-prone and even impossible for human experts to determine all required state features [70]. For example, a RAM-based Atari game playing task has a 128-dimensional input (each dimension is an integer number), but not all dimensions are useful as reported in [91]. However, it is extremely difficult for human experts to manually determine suitable features on such a problem. Secondly, once the features for a problem have been determined, they can no longer be adapted according to the changing requirements of the problem do-

[5] Please refer to Section 2.2.5 for more technical details.

[6] Please refer to Section 2.2.5 for more technical details.

main [70, 246, 120, 150]. To address these problems, we formed another important research question ***Q(4)*** (addressed in ***O(4)*** in Section 1.3),

Q(4): *How can we develop innovative evolutionary feature learning techniques capable of automatically discovering useful state features to facilitate policy direct search on large-scale RL problems*?

1.3 Goals

The ultimate goal of the thesis is to develop ***effective policy direct search algorithms*** for tackling challenging RL problems via several important techniques. First, the thesis aims to improve the accuracy of policy gradient estimation by utilizing informative historical gradients through the Primal-Dual Approximation (PDA) technique. Second, the thesis intends to achieve the state-of-the-art performance by properly balancing the exploration-exploitation trade-off via joint application of Covariance Matrix Adaption Evolutionary Strategy (CMA-ES) and Proximal Policy Optimization (PPO). Third, the thesis seeks to stabilize value function learning via Sandpile Model (SM) meanwhile generalizing the compatible condition to support flexible value function learning via q-logarithm. Fourth, the thesis endeavors to enhance various PGS algorithms by extracting useful features via newly developed NeuroEvolution techniques for automated feature learning. To achieve this goal, four specific research objectives O(1) to O(4) have been identified to address the four key research questions Q(1) to Q(4) described in Section 1.2.

O(1) Develop a new general policy gradient learning framework by using the PDA technique. Algorithms developed under the framework are expected to estimate policy gradients more accurately than the traditional policy gradient estimation methods. The framework firstly converts the complex primal problem to a simpler dual problem and then utilizes weighted historical consecutive gradients to obtain a more accurate policy update. Moreover, the algorithms are expected to perform empirically better in

comparison to other PGS algorithms. In addition, the theoretical guarantee of convergence of the three newly proposed dual algorithms has been proven.

O(2) Develop a new sample efficient evolutionary deep policy optimization algorithm on CMA-ES. The algorithm is expected to achieve state-of-the-art performance in the DRL domain in terms of computational efficiency, sample complexity, and learning effectiveness, especially while comparing to cutting-edge PGS algorithms and advanced evolutionary DRL algorithms.

O(3) Develop new policy gradient search algorithms by improving the reliability and flexibility of value function learning. The algorithm is expected to achieve reliable and accurate learning of value functions by either applying the SM or generalized compatible function approximation. As a result, we expect to further achieve more effective policy learning in comparisons to the PGS algorithms without adopting SM and generalized compatible function approximation.

O(4) Develop a new NeuroEvolution (NE) based automated feature learning based policy learning scheme. The scheme is expected to enable seamless integration between automated feature learning and effective policy gradient search, and achieve state-of-the-art performance on large-scale RL problems in comparison to both well-known PGS methods and NE methods.

1.4 Major Contributions

The thesis makes four major contributions to the filed of RL.

1. Effective Policy Gradient Search through Primal-Dual Approximation
To improve the effectiveness of PGS, one key challenge is how to obtain accurate policy gradient estimations as the analytical expression of such gradients is not available. The situation becomes more challenging

in Actor-Critic (AC) learning framework because two learning processes (value function learning and policy learning) are both realized via gradient based updating. In such a way, even if the estimation noise for one learning process can be reduced to a reasonably low level, the strongly interconnected learning on two separate parametric functions can lead to error propagation and hence substantially affect the learning effectiveness.

Most PGS algorithms have been proposed with the focus on improving the precision of policy gradient estimations, such as [202, 119, 238, 20, 21, 31, 112, 172, 14, 173, 30, 173, 31, 53, 131, 69]. However, these algorithms neglect the importance of the historical gradients.

In the thesis, we show how this challenge can be addressed via the PDA technique. In particular, we study possible ways of converting the challenging primal problem (i.e., the original policy optimization problem defined in PGS) to simpler linear dual problems through averaged historical gradients accompanied with a strongly convex regularization term. The dual problems can, therefore, be treated as a locally linear estimation of the original primal problems and can often be solved analytically. As a result of the conversion between primal and dual problems, historical gradients obtained in primal spaces can be naturally maintained and used to reduce gradient estimation error. Following the idea, we have developed a general policy learning framework based on PDA. With the framework, we have developed three new Actor-Critic Algorithms, i.e., Dual Regular Actor-Critic (Dual-RAC), Dual Natural Actor-Critic with Advantage Parameters (Dual-NACA), and Dual Natural Actor-Critic with Fisher Matrix (Dual-NACF), on the basis of three classical PGS algorithms including Regular Actor-Critic (RAC), Natural Actor-Critic with Advantage Parameters (NACA) and Natural Actor-Critic with Fisher Matrix (NACF). We also theoretically prove that the proposed algorithms under the new learning framework can converge under suitable conditions.

We experimentally evaluate the proposed algorithms on six benchmark control tasks including Mountain Car, Inverted Pendulum, Inverted Double Pendulum, Inverted Pendulum Swing Up, Lunar Lander, and Bipedal

Walker. The results show that our algorithms not only are easily convergent but also are more effective than respective base algorithms.

To sum up, with the newly developed algorithms, we can finally solve difficult benchmark problems effectively by using step-wise PDS algorithms with linear policies. In addition, the new algorithms can even outperform batch-based PDS algorithms, such as PPO-Linear, debunking the myth that step-wise PDS is outdated. We, therefore, made the contribution to the literature since existing step-wise PGS algorithms with linear policy cannot solve those benchmark problems effectively and reliably.

Part of the contribution is formed as a journal article to be submitted:

- Y. Peng, G. Chen, and M. Zhang, "Primal-Dual Sub-Gradient Approximation based Policy Gradient Search," *Machine Learning*, 2018. (Targeting on the journal "Machine Learning")

2. Proximal Evolutionary Strategies for Sample Efficient Policy Direct Search

To solve complex RL problems, another promising alternative to PGS is Evolutionary Algorithms (EAs), such as Genetic Algorithms (GAs) [210] and Evolutionary Strategy (ES) [186]. Because EAs are naturally suitable for exploration which provides more opportunities to find better solutions. However, these EAs remain far away from the state-of-the-art performance when being applied to training policies represented as Deep Neural Networks (DNNs) since existing EAs focus mainly on exploration and may converge much slowly (hence require much more samples) in comparison to non-EA RL techniques. More specifically, these EAs are facing three issues: low time efficiency, high sample complexity, and low learning effectiveness. Thus, to achieve the state-of-the-art performance, it is paramount to develop new EA methods that properly balance exploration and exploitation.

Existing algorithms, such as Uber-GAs [210] or OpenAI-ES [186], have made considerable efforts to outperform state-of-the-art non-EA methods with some success. However, these techniques remain more sample

complex compared to traditional PGS. Because EAs evaluate each individual through simulations involving sampling a long sequence of new state transition samples. For example, as reported in [186], OpenAI-ES have used 3x and 10x as many samples to perform well on most Atari game playing tasks compared to A3C [154] and TRPO [189]. In addition, these works require a considerable amount of computational resource as they usually require large population sizes.

In the thesis, we have shown that CMA-ES, a typical EA, can be improved in terms of time efficiency by using a layer-wise training mechanism where only a portion of parameters of a DNN are trained each time. Moreover, we have shown that, with the help of a performance lower bound based surrogate model, the sample complexity can be further reduced while using CMA-ES for DRL. Lastly, to achieve the state-of-the-art learning performance, the PGS can be utilized to promote local exploitation to fine tune the best policy evolved by CMA-ES. By incorporating all three improvements, we successfully develop a new ES-based algorithm for DRL – Proximal Evolutionary Strategy (PES), which can achieve the state-of-the-art performance regarding time efficiency, sample complexity, and learning effectiveness.

The empirical experiments demonstrate that our proposed PES algorithm can achieve competitive and sometimes better performance than TRPO [189], PPO [191], and ACKTR [240] on ten continuous control benchmarks, including Lunar Lander, Bipedal Walker, Bipedal Walker Hardcore, Inverted Pendulum, Inverted Pendulum Swing Up, Inverted Double Pendulum, HalfCheetah, Hopper, Walker2D and Reacher.

In summary,we made contribution to DRL research by developing the first time in the literature an EA that can outperform cutting-edge DRL algorithms in terms of both effectiveness and sample efficiency. Notably, the key technical novelty of the newly proposed algorithm is the seamless integration of the EA based global search and the PGS based local search for properly balancing the exploration-exploitation trade-off, which is essential for effective DRL.

Part of the contribution is formed as a large journal paper:

- Y. Peng, G. Chen, and M. Zhang. "Proximal Evolutionary Strategies: Achieving State-of-the-art Deep Reinforcement Learning through Evolutionary Policy Optimization," submitted for review to *IEEE Transaction on Evolutionary Computation*, 2018.

3. Reliable and Flexible Value Function Learning for Policy Direct Search

The effectiveness of PGS often relies heavily on the quality of value function learning. In particular, AC algorithms, an important PGS approach, are usually composed of two distinct learning processes, namely actor (a.k.a, policy) learning and critic (a.k.a, value function) learning. Specifically for effective critic learning, the critic is usually represented as a parametric value function. This function is made up of two important components, i.e., value function parameters and state features. They follow a linear/nonlinear relationship to approximate the expected total rewards. Similarly, the actor learning is responsible for learning a policy which is often represented as a parametric function governed by a set of policy parameters. Accordingly, the gradient descent technique is adopted for the actor learning, where the gradients of the expected cumulative rewards with respect to policy parameters are known as policy gradients. In practice, unbiased estimation of the policy gradients is usually determined by the learned critic. The unreliable and inaccurate critic learning can lead to divergence to policy learning which can deteriorate the learning effectiveness [214, 213].

Several algorithms have been developed to address the issue above, but none has been satisfactory. For example, several methods, such as GTD [214], GTD2 [213] and Least Square Temporal Difference (LSTD) [49], have adopted off-policy training to achieve reliable value function learning, but the learning can still diverge eventually. Several second-order methods, such as LSTD [37, 36], can guarantee the reliability but with high computational complexity $O(n^2)$, where n is the number of state features. In addition, it is fairly crucial for PGS to determine a proper form of value function for estimating policy gradients. Because many PGS algorithms

rely on the PGS [7] [215], a suitable action-value function for estimating policy gradients must be a compatible function [8]. However, most of the existing works [215, 190] focus on studying different approximations of action-value function, such as Q function and Advantage function, which remain compatible with policy parameterization. Nevertheless, to the best of our knowledge, very few works have investigated the generalization of the compatible condition introduced in [215], which can bring more flexible and accurate policy gradient estimations for effective policy learning.

In the thesis, we firstly have shown that value functions can be learned reliably in a simple and straightforward manner by applying a self-organizing mechanism to the learning process. To do so, we choose the SM which has been frequently shown to drive self-organized behavior in many systems [18, 74, 55]. In this way, critic learning can adjust itself whenever it becomes unreliable based on our reliability measurement. Moreover, as value function learning becomes reliable, the effectiveness of policy learning in PGS can also be improved. Next, we have shown that flexible value function learning can be achieved by generalizing the compatible function representation. For this purpose, we adopt q-logarithm to formulate a general new family of the compatible functions which is used for accurately approximating gradients. With the generalization, we can introduce an extra degree of freedom to the value function learning, by adjusting which we may achieve more accurate value function learning that leads to more effective policy learning.

Empirical findings have demonstrated that learning effectiveness can be significantly improved on benchmark Cart Pole and Puddle World problems, and also revealed the fact that value function learning reliability and learning effectiveness are strongly correlated. Moreover, with the new flexible family of compatible functions, the learning effectiveness of PGS can also be improved.

To sum up, we have made contributions to the literature in two ways. We

[7] Please refer to Section 2.2.5 for more details.
[8] Please refer to Section 2.2.5 for more details.

first developed a new effective PGS algorithm with reliable value function learning enabled by the SM. Afterward, we proposed a new flexible family of compatible functions via the generalization of the compatible condition to achieve flexible value function learning. With such flexibility, we developed a new PGS algorithm with the well-demonstrated effectiveness on many benchmark problems. Furthermore, our research also opened a new direction to improve the effectiveness of PGS further.

Part of the contribution is evidenced in two conference publications:

- Y. Peng, G. Chen, M. Zhang, and S. Pang. "Generalized Compatible Function Approximation for Policy Gradient Search," in *The 23rd International Conference on Neural Information Processing (ICONIP 2016)*, 2016.
- Y. Peng., G. Chen., M. Zhang, and S. Pang. "A Sandpile Model for Reliable Actor-Critic Reinforcement Learning," *2017 International Joint Conference on Neural Networks (IJCNN 2017)*, 2017.

4. Enhancing Policy Direct Search via Automated Evolutionary Feature Learning

Our study of existing RL algorithms clearly shows that the effectiveness of PDS is highly sensitive to state features, which are often designed manually based on raw environment inputs. However manual feature design requires tremendous efforts from domain experts. If important state features were overlooked, the learning performance could be seriously affected.

Many researches have considered to automate the feature learning process [165, 57, 234] by optimizing/learning a parametric feature base functions (e.g., Radial Basis Function (RBF) Network [165]). However, these methods aim to accurately approximate the value function of given policies without attempting to learn effective policies. Some evolutionary approaches, such as NeuroEvolution of Augmenting Topology (NEAT) [206] and Hypercube-based NEAT (Hyper-NEAT) [205], search directly in the policy space by treating each policy represented as an NN as an action

selector. In such a setting, the state inputs are used directly by NNs to produce action outputs. Hence the feature learning process is mingled with the policy search process. However, these methods may completely fail on large-scale problems as reported in [90, 152].

In this thesis, we have shown how the issues mentioned above can be solved by seamlessly integrating NeuroEvolution based feature learning with PGS based policy learning. First, unlike other methods that mingled feature and policy learning into a single process, the proposed NEAT+PGS scheme realizes a clear separation between the two processes to avert interferences while learning them together. In our design, a fixed pre-trained policy is used for evaluating the NNs evolved by NEAT for extracting useful features. Afterward, the extracted good features are used to further search good policies with the help of PGS algorithms. Second, only a single policy is required to be retained and be improved across consecutive learning stages via PGS, and hence the number of training samples is significantly reduced. Lastly, NEAT+PGS provides a general scheme to accommodate various cutting-edge PGS algorithms such as TRPO, Policy learning by weighting exploration with the returns (POWER) [117] and Advantage Actor-Critic (A2C) [56].

The proposed algorithms have been evaluated on two types of problems, 1) control problems including Cart Pole and Mountain Car, and 2) Ram-based Atari game playing tasks including Asteroids, Breakout, Freeway, Seaquest, SpaceInvaders, and TimePilot. The evaluation results have confirmed the generalities, sample efficiency, and effectiveness, of the proposed algorithms. Besides, the results of a specially designed experiment have confirmed that the state features evolved from one task can be transferred to further facilitate the PGS algorithm in another similar but different task.

With the development of NEAT+RAC and NEAT+PGS, we can summarize our contribution to the literature from two aspects. First, we have developed a novel NE based feature learning technique to construct a new learning scheme that can improve the effectiveness of various PGS

algorithms. Second, with the newly developed learning scheme, we have, for the first time, proposed to clearly separate the feature learning from the policy learning to enable effective knowledge sharing among multiple agents in parallel for effective RL.

Part of the contribution is evidenced in three conference publications:

- Y. Peng, G. Chen, S. Holdaway, Y. Mei, and M. Zhang. "Automated State Feature Learning for Actor-Critic Reinforcement Learning through NEAT," in *The Genetic and Evolutionary Computation Conference (GECCO Companion 2017)*, 2017.
- Y. Peng, G. Chen, M. Zhang, and Y. Mei. "Effective Policy Gradient Search for Reinforcement Learning through NEAT based Feature Extraction," in *Simulated Evolution and Learning - 11th International Conference (SEAL 2017)*, 2017.
- Y. Peng, G. Chen, H. Singh, and M. Zhang. "NEAT for Large-Scale Reinforcement Learning through Evolutionary Feature Learning and Policy Gradient Search," in *The Genetic and Evolutionary Computation Conference (GECCO 2018)*, 2018.

1.5 Organization of Thesis

The remainder of the thesis is structured as follows. Chapter 2 provides a literature survey on the background of RL and followed by a discussion of related works to motivate the research of the thesis. In Chapters 3-6, the primary contributions of this thesis are presented, which correspond to each research contribution stated above. The thesis concludes and proposes future research directions in Chapter 8.

Chapter 2 presents a general background review for RL and a discussion on related works to motivate the research of the thesis. It first introduces three fundamental ML paradigms, i.e., Supervised Learning (SL), Unsupervised Learning (UL) and Reinforcement Learning (RL). The main focus is to introduce the RL framework with formal descriptions of the RL problem. Next, it describes

the essential techniques used in the thesis, including Evolutionary Computation (EC) with focus on NeuroEvolution (NE), Transfer Learning (TL) and Feature Learning (FL). Then it proceeds to a discussion on different approaches to solving RL problems, mainly it discusses related works with respect to each research objective introduced in Section 1.3 to support the motivations of the thesis.

Chapter 4 proposes a new effective policy gradient search framework by applying the primal-dual sub-gradient approximation based optimization. It firstly discusses the issues that may be brought by traditional policy gradient learning which occur in the sophisticated primal problem space. Then it proposes a new learning scheme where a complex primal problem can be converted to a simpler dual problem via the primal-dual sub-gradient approximate technique. In such dual space, more effective gradient updates can be achieved. Following the learning scheme, three new PGS algorithms are developed, i.e., Dual-RAC, Dual-NACA, and Dual-NACF, from the three existing PGS algorithms including RAC, NACA, and NACF. The proposed three new algorithms are examined on two benchmark problems in comparison to RAC, NACA, NACF and two state-of-the-art algorithms, i.e., Augmented Random Search (ARS) and an adapted PPO algorithm with linear policy. Additionally, the chapter provides theoretical analysis to analyze the convergence of all proposed algorithms.

Chapter 5 proposes a new evolutionary deep policy optimization algorithm that achieves state-of-the-art performance in terms of time efficiency, sample complexity, and learning effectiveness. The chapter firstly discusses cutting-edge research on DRL and the positions of evolutionary algorithms in the domain. Then it proposes a proximal evolutionary strategy algorithm to fulfill the potential of EAs to achieve competitive performance in terms of both effectiveness and sample efficiency in comparison to several state-of-the-art DRL algorithms such as PPO, TRPO, and ACKTR. This is achieved by incorporating three new improvements: a CMA-ES based layer-wised training for improving time efficiency, a proximal performance lower bound based surrogate model for improving sample complexity, and a gradient-based local search method to further boost learning effectiveness. PES is examined on nine continuous

benchmark problems with comparisons to three cutting-edge PGS algorithms.

Chapter 6 proposes new policy gradient search algorithms by improving the reliability of value function learning and generalizing approximation for compatible function. The chapter introduces the actor-critic scheme for policy gradient search. Next, it highlights the fact that the effectiveness of policy learning relies heavily on reliable and accurate value function learning. Driven by the understanding, the chapter develops two complementary algorithms with two improvements. One algorithm aims to enhance the reliability of value function learning with the help of a Sandpile model. The other algorithm targets to generalize the compatible function condition to obtain a flexible family of new compatible functions to improve the accuracy of policy gradient estimations eventually. The two algorithms are all derived from the RAC algorithm and are evaluated on two benchmark control problems. Experimental results show the significant improvement in the effectiveness of the two proposed algorithms compared to RAC, and more importantly, shed new light on how to enhance policy learning through reliable and accurate learning of value functions.

Chapter 7 proposes a new policy gradient learning scheme, in which automated feature learning via NEAT is seamlessly integrated with various effective policy gradient search algorithms. The chapter emphasizes on the importance of feature learning for effective RL. Driven by the main research goal, it develops a NEAT based three-stage learning scheme for effective feature learning, which can be seamlessly integrated with different cutting-edge PGS algorithms. The learning scheme has been deployed onto one classical PGS algorithm, i.e., RAC, and three state-of-the-art PGS algorithms including TRPO, PoWER, and A2C. Our experimental results show that the proposed algorithms outperform NEAT and the four PGS algorithms on six benchmark Atari games.

Chapter 8 gives a summary of all works proposed in this thesis and concludes the thesis. The discussions are around the contributions of the thesis. Also, possible future research opportunities are presented.

Chapter 2

Literature Review

The chapter presents the research background of the thesis and followed by a discussion of related works that supports the motivations of the thesis. It starts with a general introduction across different machine learning paradigms in Section 2.1.1 including Supervised Learning (SL) , Unsupervised Learning (UL), and Reinforcement Learning (RL), with a strong focus on the reinforcement learning framework by introducing the formal description of the problem in Section 2.1.2. Following that, the chapter discusses the background knowledge about Evolutionary Computation (EC) including particularly NeuroEvolution (NE) in Section 2.1.3, Transfer Learning (TL) in Section **??** and Feature Learning (FL) in Section 2.1.4. The chapter then proceeds to discuss different approaches to solving RL problems in Section 2.2. Subsequently, it discusses related works in Section 2.3 with respect to the four research objectives introduced in Section 1.3 to motivate the research works reported in the thesis. To the end, the chapter summarizes the discussions on related works in Section 2.4 to make connections to the subsequent contribution chapters (i.e., Chapters 3-6).

2.1 Background

2.1.1 Machine Learning

In the recent years, Machine Learning (ML) has become one of main driving forces of modern technology, which significantly changes our daily life with massive useful applications from providing intellectual recommendations of movies for your tastes [6] to defeating human-beings in the difficult GO game [199]. Owing to the enormous economic and research potential, ML has gained more and more popularity not only among industrial practitioners but also among academic researchers.

A typical ML system is expected to optimize some performance measurement for some task by learning from example data or past experiences [3, 153]. According to Mitchel [153], a formal definition of ML is given below:

> "computer program is said to learn from experience **E** with respect to some class of tasks **T** and performance measure **P**, if its performance at tasks in **T**, as measured by **P**, improves with experience **E**"

Generally, ML systems can be classified into three broad paradigms according to whether or not there are supervisors involved [153]. The paradigms are Supervised Learning (SL), Unsupervised Learning (UL), and Reinforcement Learning (RL) [153], which are introduced below respectively.

Supervised Learning

An SL system is to learn from examples containing desired outputs (i.e., labels) provided by a supervisor [153, 3, 22]. The goal is to find a mapping relationship from inputs to outputs that can be generalized to unseen inputs to make correct predictions on unseen inputs. Two typical SL tasks are Classification and Regression. In classification tasks, the learning system is given a set of instances (i.e., input and output pairs) and is expected to learn a model that can make predictions on categorical membership of each instance [153, 3, 22]. The spam filter is a typical example of classification task where the agent is trained to classify new emails to spam or ham [67]. A regression learning agent is to predict

a target numeric value based on the given inputs [153, 3, 22]. House pricing prediction for a specific area based on historical data with many features, such as neighborhood, location, number of bathrooms, etc., is a good example of regression [67].

Unsupervised Learning

A typical UL system is supplied with solely input data without any desired outputs pre-determined by human supervisors [153, 3, 22]. UL aims to find regularities, for instance, regular patterns that occur more often than others, from the input data. Typical UL methods include Clustering and Association Rule Learning. Clustering is such a task of finding clusters or groupings within the given input. Association Rule Learning is a task to discover associated relations or patterns between feature within the input data. A typical example of UL is to apply a clustering algorithm to detect groups of similar visitors to a particular website, where none of group information is given to the algorithm and it finds connections by itself [67].

Reinforcement Learning

An RL system takes observations from the environment that it interacts with as inputs, and then produces a sequence of actions as outputs to be performed in the environment. Meanwhile, it receives instant rewards from the environment. Its goal is to find a policy that can generate a series of correlated actions that can enable the RL system to obtain the maximum total rewards. In this scenario, there is no guidance of supervisors involved. RL is further discussed in Section 2.1.2 below.

2.1.2 Reinforcement Learning

Reinforcement Learning is defined as a process where actions are drawn by an agent via its iterative interactions with an unknown environment [153, 3, 22, 212]. The environment provides responsive feedbacks (i.e., reward) to the actions it receives from the agent. The agent learns from the interactions with the

environment in order to maximize the long-term pay-off. Fig. 2.1 illustrates the typical RL problem [212], which can be formally modeled as a Markov Decision Process (MDP) (see Section 2.1.2)

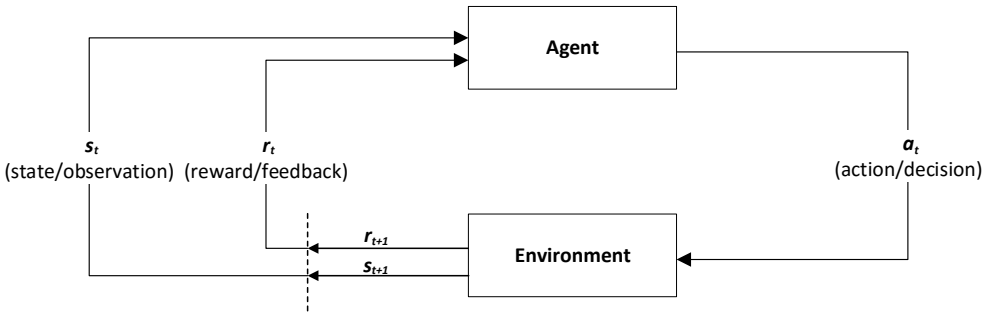


Figure 2.1: A Reinforcement Learning problem is formally modeled as an MDP, drawn based on Figure 3.1 in [212].

Markov Decision Process

An MDP [136, 212, 153, 236, 148, 185, 218] is modeled as a 5-tuple discrete-time process at a discrete time interval, namely $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, in which:

- $\mathcal{S}$ is a state space, where $\vec{s}_t \in \mathcal{S}$ is an observed state at time step t, and $n \in \mathbb{N}$.
  - For the continuous case, the state space is an uncountable set, such that $\mathcal{S} \subseteq \mathbb{R}^n$.
  - For the discrete case, the state space is a countable non-empty set containing all possible state.
- $\mathcal{A}$ is an action space. For any state $\vec{s} \in \mathcal{S}$, we have $a_t \in \mathcal{A}(\vec{s}_t)$ which represents the action taken by the agent at time step t.
  - For the continuous case, the action space is an uncountable set, such that $\mathcal{A}(\vec{s}) \subseteq \mathbb{R}$.

  - For the discrete case, the action space is a countable non-empty set containing all available action for the given state.

- $\mathcal{P}$ is a *transition function* defined as,

$$\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1], \tag{2.1}$$

  where $\mathcal{P}(\vec{s}_t, a_t, \vec{s}_{t+1})$ denotes $Pr(\vec{s}_{t+1}|\vec{s}_t, a_t)$, i.e., a transition probability from a state $\vec{s}_t$ taking an action a_t to a state $\vec{s}_{t+1}$.

- $\mathcal{R}$ is a *reward function* defined as,

$$\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}, \tag{2.2}$$

  where $\mathcal{R}(\vec{s}_t, a_t, \vec{s}_{t+1})$ denotes the immediate reward distribution, and r is used to represent an immediate scalar reward sampled from the reward distribution, i.e., $r_{t+1} \sim \mathcal{R}(\vec{s}_t, a_t, \vec{s}_{t+1})$.

- $\gamma \in [0, 1)$ is a discounted factor used for decaying the future reward.

The MDP model must satisfy the Markov Property below,

$$Pr(\vec{s}_{t+1}|\vec{s}_t, a_t, \vec{s}_{t-1}, a_{t-1}, \ldots, \vec{s}_0, a_0) = Pr(\vec{s}_{t+1}|\vec{s}_t, a_t) = \mathcal{P}(\vec{s}_t, a_t, \vec{s}_{t+1}). \tag{2.3}$$

According to (2.3), the probability of transiting to state $\vec{s}_{t+1}$ only depends on the action a_t taken in the state $\vec{s}_t$, which does not depend on any previous actions and states. The property implies that when the agent makes a decision of performing the action a_t at the state $\vec{s}_t$, it only needs to consider the current state $\vec{s}_t$ without looking back to the history $\mathcal{H}_t = \{\vec{s}_0, a_0, \ldots, \vec{s}_{t-1}, a_{t-1}\}$.

Foundational Concepts of Reinforcement Learning

By learning from its interactions with an unknown MDP, an RL agent attempts to find an optimal policy, which guides the agent to obtain the maximum long-term pay-off [212, 136, 236]. To better understand RL, we next introduce several fundamental concepts in RL, including state features, value function, policy and the optimality criteria respectively.

State Feature. In RL, state features usually refer to high-level representations of the raw state input (e.g. sensory input in a pole balancing control problem) that will be used by an RL system to effectively learn its policy [212]. In RL systems, it is beneficial to use such high-level representations because of two reasons. First, it maps the raw inputs to a high-level feature space where the relationships among attributes of the input may be extracted to promote the reinforcement learning [212, 234]. Second, carefully designed features can contain prior domain knowledge, which can be essential to improve the effectiveness of the RL system [212, 234, 70].

The extraction of state features is achieved through basis functions [218], which can be defined as,

$$\vec{\phi}(\vec{s}) = [\vec{\phi}_1(\vec{s}), \ldots, \phi_m(\vec{s})], \tag{2.4}$$

where $\vec{\phi}_i \in \mathbb{R}$ for $i = 1, \ldots, m$, and m is the dimension for the state features. For a given state space $\mathcal{S}$, one can construct the state features $\vec{\phi}(\vec{s})$ where $\vec{s} \in \mathcal{S}$ by using many different means, for example, discretization [212], tile coding [212], Radial Basis Function (RBF) networks [92], Neural Networks (NNs) [212], and so forth.

Value Function. For one specific state, Value Function (VF) specifies the expected long-term pay-off [1] that can be obtained by the agent in the future upon starting its journey from any given state. It indicates the long-term desirability of the state if the agent starts from the state following a specific policy, which helps the agent make and evaluate decisions. However, the value function cannot often be analytically determined. As a matter of fact, to achieve efficient and precise estimation of value function brings a great many of challenges to practical reinforcement learning.

Formally, value functions are defined as functions mapping states or state-action pairs to the expected long-term pay-off when following a particular policy. There are two types of value functions, i.e., the *state value function* V^π

[1] The *expected long-term pay-off* also refers to the *expected total rewards*, and are often used interchangeably in the literature.

(see (2.5)) and the *action value function* Q^π (see (2.6)). The state value function is defined as below,

$$\begin{aligned} V^\pi(\vec{s}) &= \mathcal{J}(\pi) \\ &= \mathbf{E}[\textstyle\sum_{t=0}^{\infty} \gamma^t r_{t+1} | \vec{s}_0 = \vec{s}, \pi], \end{aligned} \tag{2.5}$$

where $\vec{s}_0$ represents the starting state arbitrarily initialized by any state $\vec{s} \in \mathcal{S}$. This equation means that, the expected long-term pay-off is $V^\pi(\vec{s})$ whenever an agent starts from any state $\vec{s} \in \mathcal{S}$ under a specific policy π.

Also, we can have the action value function defined as,

$$Q^\pi(\vec{s}, a) = \mathbf{E}[\textstyle\sum_{t=0}^{\infty} \gamma^t r_{t+1} | \vec{s}_0 = \vec{s}, a_0 = a, \pi]. \tag{2.6}$$

Different from the state value function in (2.5), the action value function in (2.6) indicates that, the expected long-term pay-off $Q^\pi(\vec{s}, a)$ is given when initiating from any state $\vec{s} \in \mathcal{S}$ and taking an action $a \in \mathcal{A}(\vec{s})$, thereafter following the policy π. The relationship between V^π and Q^π is given below,

$$V^\pi(\vec{s}) = \int_{a \in \mathcal{A}(\vec{s})} \pi(a|\vec{s}) Q^\pi(\vec{s}, a) da. \tag{2.7}$$

Intuitively, the optimal value function is,

$$V^*(\vec{s}) = \max_\pi V^\pi(\vec{s}). \tag{2.8}$$

Thus, we can obtain the optimal policy as,

$$\pi^* = \operatorname*{argmax}_\pi V^\pi(\vec{s}). \tag{2.9}$$

Policy. A *Policy* is a core concept of RL. Formally, the policy is a function outputting an action $a \in \mathcal{A}(\vec{s})$ for each state $\vec{s} \in \mathcal{S}$ [212, 236]. There are two types of policy, i.e., *deterministic* and *stochastic*.

The deterministic policy is formulated as,

$$\pi : \mathcal{S} \to \mathcal{A}. \tag{2.10}$$

On the other hand, the stochastic policy [2] is formulated as,

$$\pi : \mathcal{S} \times \mathcal{A} \to [0, 1], \tag{2.11}$$

[2]Note that, a stochastic policy no longer outputs actions but probabilities of choosing actions at the state.

where $\pi(a|\vec{s}) \geq 0$ and $\forall \vec{s}, \int_{a \in \mathcal{A}(\vec{s})} \pi(a|\vec{s}) da = 1$.

The stochastic policy in (2.11) actually defines a distribution over all possible actions in a given state. Such a definition gives us a flexibility to consider both stochastic and deterministic policies. There, we will only use the definition of the stochastic policy in (2.11) throughout the thesis.

Optimality Criterion. In RL, the optimality criterion model is derived from the expected long-term pay-off while the agent follows a particular policy [212, 236, 218]. Two common optimality model are the *infinite horizon discounted reward model* and the *average reward model*. The former model is formulated as,

$$\mathcal{J}(\pi) = \lim_{h \to \infty} \mathbf{E}[\frac{1}{h} \sum_{t=0}^{h} r_{t+1} | \pi], \tag{2.12}$$

and the latter model is,

$$\mathcal{J}(\pi) = \mathbf{E}[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | \pi]. \tag{2.13}$$

In this thesis, we employ the infinite horizon discounted model, as it is more commonly studied in the literature [212]. In this model, later rewards are discounted more than earlier rewards. Moreover, the discount factor γ determines that only finite long-term pay-off by the agent even in an infinite horizon. Note that, when $\gamma = 0$, it means that the agent only concerns about immediate rewards.

In summary, the goal of an RL agent is to learn a policy π that maximizes the expected long-term pay-off $\mathcal{J}$. Such a policy is said to be ***optimal***. Thus, we can have the ***optimal policy*** as,

$$\begin{aligned} \pi^* &= \text{argmax}_\pi \, \mathcal{J}(\pi) \\ &= \text{argmax}_\pi \, \mathbf{E}[\textstyle\sum_{t=0}^{\infty} \gamma^t r_{t+1} | \pi]. \end{aligned} \tag{2.14}$$

Note that, if the RL problem satisfies the Markov Property in (2.3), there will exist at least one deterministic optimal policy.

2.1.3 Evolutionary Computation

In this subsection, we start with a general introduction to EC concerning its applicability to RL, then proceed to briefly introduce several conventional EC techniques including Evolutionary Algorithms (EA), Swarm Intelligence (SI), and NeuroEvolution (NE). This subsection mainly focuses on the basic ideas and techniques of NE. Also the relevant research of using NE for RL will be discussed in detail in Section 2.2.6.

Evolutionary Computation

Evolutionary Computation (EC) is a family of computational algorithms inspired by biological evolution principles, which is widely used as optimization or search techniques [244, 64]. Typical EC techniques can be generally categorized as Evolutionary Algorithms (EAs), Swarm Intelligence (SI), and others such as NeuroEvolution (NE), which are briefly discussed below. Particularly, Evolutionary Strategies and NeuroEvolution, as the main EC techniques used in this thesis, are discussed with more details.

Evolutionary Algorithms

Evolutionary Algorithms are methods of conducting a stochastic search for a near-optimal solution to a given problem [64, 7]. The search process generally consists of several key components, including solution representation (chromosome encoding), fitness function, population initialization, selection operators, and reproduction operators (recombination and mutation). A typical EA starts by initializing a group of encoded solution individuals as an initial population. Afterward, based on the evaluation of each individual by the fitness function, the EA utilizes selection and reproduction operations to produce a new population of individuals for the next generation. The process repeats for many generations until a near-optimal solution is located. Here, we briefly discuss three important EAs, including Genetic Algorithm (GA) [7], Genetic Programming (GP) [128], and Evolutionary Strategies (ES) [13].

Genetic Algorithms. Genetic algorithms (GAs) [7] are one of the earliest algorithms developed to mimic biological systems. The search process of GAs is based on natural selection (i.e., the principle of biological evolution). Each individual in standard GAs is expressed as a genotype encoded by a fixed length of bits (bit-strings) or floating points. During the evolution process, GAs repeatedly adapts the population of individuals, which makes the entire population evolve towards an optimal solution. The main operators in GAs are (1) selection operator that decides which individuals to survive to the next generation, (2) recombination operator that combines selected parents to produce offspring for the next generation, and (3) mutation operator that introduces new variations into an existing individual for the next generation. Although GAs, as a global search mechanism, may less likely be trapped by local optima compared to gradient descent methods, it can often be very computationally expensive especially when individuals are encoded as large dimensional bit-strings [64].

Genetic Programming. Genetic programming (GP) [128] encodes computer programs as a set of variable-length genes, and then evolves them via EAs such as GAs. Each individual in GP is a computer program. For every generation, each individual is directly evaluated based on a specific problem domain to determine its fitness for that program. One advantage of GP is that the solutions evolved by itself are interpretable, unlike other oracles such as Neural Networks. However, the computational cost of using GP is often high [64].

Evolutionary Strategy. Evolution Strategies (ESs), different from GAs or GP, consider both genotypic and phenotypic evolution [3], which is considered a specialization of EAs [13, 64, 179]. Thus, ES repeats a similar evolutionary process as a typical EA to produce new individuals repeatedly by adopting selection, recombination and mutation operators. However, ESs are distinguished in two key design principles, namely *Unbiasedness* and *self-adaptive control of strategy parameters* [85, 88].

[3] A genotypes refers to the genetic composition of an individual inherited from its parents, whereas a phenotype is the expressed behavioral traits of an individual in a specific environment.

- *Unbiasedness*: In ESs, new information introduced by mutation or recombination to individuals are unbiased, but selection may bias the information towards the direction of better fitness. Such bias can lead to premature convergence, which can be addressed by adopting fitness independent recombination and environmental selection that are expected to be unbiased operators [85, 88]. By doing so, ESs can achieve maximum exploration together with unbiasedness.

- *Self-adaptive control of strategy parameters*: In ESs, each individual is encoded with decision parameters as well as strategy parameters such as step size [13]. Thus, these strategy parameters (e.g., step size) are self-adapted so as to achieve most effective search along each possible direction.

Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES)

Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) is a notable member of the family of ES algorithms. It is a purposefully designed stochastic search method for continuous optimization of non-linear and non-convex functions [88]. In CMA-ES, it adapts the covariance matrix to learn a second-oder model of the underlying objective function similar to the approximation of the inverse Hessian matrix in the Quasi-Newton method in classical optimization [88, 86]. Thus, CMA-ES has been widely shown to be more effective in contrast to many first-order gradient descent methods as well as other EAs [86, 88].

CMA-ES is implemented in four steps. First, CMA-ES samples a population of new individuals from a multivariate normal distribution. For each generation $g = 0, 1, 2, \ldots$, this step is defined as,

$$\vec{x}_k^{g+1} \sim \vec{m}^g + \sigma^g \mathcal{N}(\vec{0}, \vec{C}^g), \tag{2.15}$$

where $\mathcal{N}(\vec{0}, \vec{C}^g)$ is the multivariate normal distribution with zero mean and covariance matrix $\vec{C}^g$, $\vec{x}_k^{g+1}$ denotes the k-th individuals of the generation $g+1$, $\vec{m}^g$ and σ^g represent the mean of search distribution and step-size at generation g respectively. Second, it selects $\mu < \lambda$ out of λ individuals by truncated selection, and then recombines these individuals with crossover (e.g., weighted

intermediate recombination). This step can be represented as,

$$\vec{m}^{g+1} \sim \vec{m}^{g} + c_m \sum_{i=1} \mu\omega_i(\vec{x}_{i:\lambda}^{g+1} - \vec{m}^{g}), \tag{2.16}$$

where $c_m \leq 1$ is a learning rate usually set to 1. Third, it updates the covariance matrix C by different means, such as estimating it from scratch, rank-μ-update, rank-one-update, and combining rank-μ-update and cumulation method. The last option is generally adopted in literature, which can represented as,

$$\vec{C}^{g+1} = (1 - c_1 - c_\mu \sum \omega_j)\vec{C}^{g} + c_1 p_c^{g+1} p_c^{g+1T} + c_\mu \sum_{i=1}^{\lambda} \omega_i \vec{y}_{i:\lambda}^{g+1}(\vec{y}_{i:\lambda}^{g+1T}) \tag{2.17}$$

where $c_1 \approx \frac{2}{n^2}$, $c_\mu \approx \min(\frac{\mu}{n^2}, 1 - c_1)$, $y_{i:\lambda}^{g+1} = \frac{\vec{x}_{i:\lambda}^{g+1} - \vec{m}^{g}}{\sigma^{g}}$, $\sum \omega_j = \sum_{i=1}^{\lambda} \omega_i \approx \frac{-c_1}{c_\mu}$, p_c^{g+1} is the evolution path [4] that reintroduces the sign information. Fourth, CMA-ES also explicitly increases or decreases of the scale for covariance matrix adaptation by tweaking the overall step size σ. Additionally, the step size is adapted according to (2.18) below,

$$\sigma^{g+1} = \sigma^{g} e^{\frac{c_\sigma}{d_\sigma}\left(\frac{||p_\sigma^{g+1}||}{\mathbb{E}||\mathcal{N}(\vec{0},\vec{I})||} - 1\right)} \tag{2.18}$$

where p_σ^{g} is the evolution path for σ.

Swarm Intelligence

Swarm intelligence (SI) is the collective intelligence of using decentralized control and self-organization [114]. Typical SI systems consists of a population of simple individuals interacting with each other and their environment, they aim to optimize global objectives through collaborative search of the space. There is no centralized control structure to regulate the behaviors of each individual. These individuals follow simple rules to interact and move towards a center of mass in the population on important dimensions with a general stochastic (or chaotic) tendency. Such interaction leads to an "intelligent" global search behavior that converges to an optimum [114]. Two main techniques in swarm intelligence are Particle Swarm Optimization (PSO) [115] and Ant Colony Optimization [58].

[4] The evolution path accumulates historical search directions in successive generations and acts as a momentum to guide the search [134].

NeuroEvolution

Inspired by the evolution of a biological nervous system in nature, NeuroEvolution (NE) is an important EC technique where EAs are applied to directly construct/evolve artificial Neural Networks (NNs) [243]. In NE, each individual is represented as an NN, and the algorithm tries to evolve a population of NNs with varied weights and topologies. In each generation of NE, each network is evaluated on a given problem. Based on the results of the evaluation, the best performing networks are selected via different selection methods, such as rank-based selection, roulette wheel selection, or tournament selection [243, 64]. Afterward, these selected networks are used to reproduce new networks for the next generation via crossover and mutation.

While solving an RL problem, individual NNs evolved by a NeuroEvolution algorithm represents a policy where the input nodes correspond to raw state inputs or state features, and the output nodes are either actions or the probability of selecting each action. Each policy is evaluated by conducting multiple roll-outs in a given MDP, and its fitness is determined by the average total rewards observed from the roll-outs. Two typical examples of NeuroEvolution algorithms are NeuroEvolution of Augmenting Topology (NEAT) [206] and its variation Hypercube-based NEAT [205], which are introduced below.

NeuroEvolution of Augmenting Topology (NEAT). NEAT is an evolutionary approach towards learning flexible NNs for various machine learning problems including RL [207]. It has gained prominent successes on classic RL tasks such as double pole balancing [206] and robotic control [208]. In comparison to other Neural Evolution methods, NEAT possesses the following technical strengths:

- *Topology evolution*: NEAT starts with a population of simplest networks where no hidden neurons or connections are given. The topology is incrementally augmented via two mutation operators, adding nodes and adding links. In this way, NEAT tends to find an NN with a minimum number of weights and a suitably complex topology for a given problem.
- *Innovation number*: NEAT encodes its genome (i.e., an NN) in a direct way where a group of connection genes defines each genome. Each connection

gene is specified by one in-node gene, one out-node gene, the weight of the connection, an enable indicator that determines whether the connection gene is expressed and an innovation number that helps identify corresponding genes for crossover. By using the simple innovation number, the crossover operation can be efficiently performed without expensive comparisons on topological similarities across different NNs.

- *Speciation*: NEAT also speciates its population so that an individual mainly competes within its speciation rather than within the entire population. In doing so, NEAT can prevent any topological innovations generated by individuals within their niches from being overridden by other niches of the population.

Hypercube-based NEAT. The Hypercube-based NEAT (HyperNEAT) was introduced by Stanley et.al [205] to extend NEAT by using indirect encodings for NNs. It is based on compositional pattern producing networks (CPPNs), which are networks used to describes complex patterns such as images. The CPPN can also be evolved by NEAT to address problems with complex input such as image-based tasks. In this thesis, we focus more on NEAT over HyperNEAT because of two reasons. First, HyperNEAT is more suitable for problems where geometric knowledge is important [205], which are not typical settings of RL problems. Second, HyperNEAT actually suffer from the "fractured" problems [5] severer than NEAT on some complicated problems as reported in [140, 118].

2.1.4 Feature Learning

The performance of almost all ML methods heavily relies on how the features (i.e., data representation) are constructed. Traditionally, to properly represent the data for effective ML, task related features must be carefully designed via feature engineering with the support of knowledgeable domain experts [26].

[5] In a "Fractured" problem, the correct decision for the agent changes abruptly and often rather than slowly and continuously, when the agent transits between states [140, 118]. Thus, the agent may perform ineffectively in such problems.

Despite the usefulness of feature engineering, it is still labor intensive. To address this, Feature Learning (FL) is a way that can automatically learn general features from the data so as to effectively extract useful information for ML tasks.

It is also imperative to learn useful features for RL algorithms [212]. A known limitation for raw state input (i.e., linear feature) is that it does not take correlations among different state input dimensions into consideration which sometimes are very useful for effective learning [212]. For example, in Cart-Pole problem, it is difficult to judge the goodness of high angular velocity (i.e., the fourth dimension of the raw state input) since it depends closely on the angle (the third dimension of the raw state input). If the angle is large, then high angular velocity indicates a dangerous situation where the pole is falling.

In RL, there are many methods developed to learn/extract features from raw state input, which can be generally categorized as four different ways. One common approach is to use supervised learning techniques to learn basis functions, such as [70, 109]. The second common approach is to apply traditional unsupervised learning techniques to construct features before applying any RL methods [127]. The third approach to directly apply NeuroEvolution to evolve NNs that combine feature extraction and action selection together in the form of high-performing policies [206]. The final approach, which is very popular recently, is to combine deep learning with reinforcement learning autonomously extract high-level features from raw state inputs, such as [81, 117, 132].

2.2 Reinforcement Learning Methods

In this section, we will focus on technical aspects of different methods for solving the difficult RL problems which are closely related to the thesis. The chapter starts from a general taxonomy of RL methods, then proceeds to describe technical details of each method. It finally discusses several methods for feature learning to improve the effectiveness of RL algorithms.

2.2.1 Reinforcement Learning Methods Taxonomy

In literature, a massive number of algorithms have been proposed to address the RL problem. In a broad sense, they can be simply separated into two major categories, i.e., value function indirect search (VIS) and policy direct search (PDS). VIS aims at learning directly the expected long-term pay-off by following the optimal policy. Once the value function is determined, RL agent can select actions under the guidance of the learned value function (without explicit representation of the optimal policy). On the other hand, in PDS methods, policies are explicitly represented and agents search directly in the space of policy representations. We will introduce the two categories of methods respectively in the following subsections.

2.2.2 Value Function Indirect Search

VIS [212, 195, 211, 231, 184, 213] is designed to find an optimal policy by indirectly searching the optimal value function first, and then obtaining the optimal policy from that optimal value function[6] (see (2.8)) [236, 195]. However, value function is often approximated since it is infeasible to determine it analytically.

In literature, there are three common ways to learn value functions based on three different representations [212], i.e., tabular value function methods, linear combination based value function approximation methods and neural network based value function approximation. Tabular value function methods are the simplest form to approximate value functions, where value functions are represented as arrays or tables [212]. These methods often give exactly the optimal value function and the optimal policy, which contrasts to the other two approximation ways where only approximated solutions can be found. However, tabular methods assume the state and action spaces are sufficiently small (finite MDP) to be represented as tables, which are impractical in reality [7] [236, 234]. To cope with this issue, one can represent value functions through parametric

[6]Note that VIS assumes that a greedy policy is always deployed, so if the value function is optimal (i.e., the maximum long-term pay-off), then the policy will also be optimal.

[7]Due to its simplicity and unpopularity, we do not cover any technical details about these tabular methods in the thesis. The technical details can be found in [212]

functions (a.k.a, function approximation) [212, 236, 218]. Commonly, two types of parameterization can be applied, linear or nonlinear, which will be detailed in Subsection 2.2.2 and Subsection 2.2.2 respectively.

Figure 2.2 illustrates a general architecture followed by most VIS algorithms. The value functions V^π or Q^π are learned via an iterative process. When the algorithm converges, the V^π or Q^π is expected to closely approximate V^* or Q^*, i.e., the optimal value functions are obtained. Then, the optimal policy π^* can be extracted from the optimal value functions.

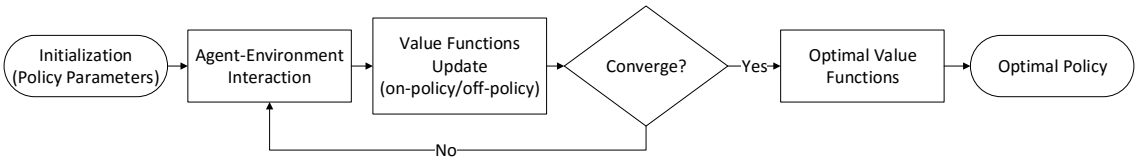


Figure 2.2: Value Function Indirect Search Framework, drawn based on Figure 3 in [193].

Linear Value Function Approximation Methods

A linear parametric value function $\tilde{V}^\pi(\vec{s})$ (i.e., linear state features explained in Subsection 2.1.2) is defined as,

$$V^\pi(\vec{s}) \approx \tilde{V}^\pi_\upsilon(\vec{s}) = \upsilon^{\pi T} \cdot \vec{\phi}(\vec{s}), \tag{2.19}$$

where $\upsilon^\pi \in \Upsilon \subseteq \mathbb{R}^m$ represents the value function parameter vectors, and $\vec{\phi}(\vec{s})$ is the m-dimensional state features (explained below). The linear value function representation $\tilde{V}^\pi(\vec{s})$ is linear to its parameters υ^π, but its state features $\vec{\phi}(\vec{s})$ may be nonlinear. The usefulness of linear value function representation relies largely on the choices of proper state features [212, 218, 70, 246, 150].

Temporal Difference (λ). The TD(λ) algorithm is frequently utilized to learn state value functions (represented in a tabular form). "Eligibility Trace" [212] is integrated in the algorithm, which works as temporary records for the occurrence of events, such as visiting a state or taking an action. More specifically,

the trace decides the eligibility of events for the undergoing learning process by λ, hence only eligible records can be given temporal credits. When $\lambda = 0$, the credit will only be given to last state $\vec{s}_{t-1}$. On the other hand, when $\lambda = 1$, it means all visited states will be credited.

Q-learning and State-Action-Reward-State-Action (SARSA). Different from TD(λ), Q-Learning [231] and SARSA [184] are designed to learn the action value function Q^π. The difference between Q-Learning and SARSA is that, the former is off-policy and the latter is on-policy.

Off-policy Q-learning can directly learn the Q function of the optimal policy. This is achieved by following the updating rule below at any time step t,

$$Q^\pi(\vec{s}_t, a_t) \leftarrow Q^\pi(\vec{s}_t, a_t) + \alpha[r_{t+1} + \gamma \max_{a_{t+1}} Q^\pi(\vec{s}_{t+1}, a_{t+1}) - Q^\pi(\vec{s}_t, a_t)]. \tag{2.20}$$

On the other hand, on-policy SARSA is developed to learn the Q-function of a specific policy known to the RL system. The corresponding updating rule at any time step t is given below,

$$Q^\pi(\vec{s}_t, a_t) \leftarrow Q^\pi(\vec{s}_t, a_t) + \alpha[r_{t+1} + \gamma Q^\pi(\vec{s}_{t+1}, a_{t+1}) - Q^\pi(\vec{s}_t, a_t)]. \tag{2.21}$$

The algorithms discussed above are the most representative VIS methods only support tabular representation of value functions. In fact, many practical problems require not only continuous state space but also continuous action space, but these mentioned algorithms all appear to be incompetent. Next, we will discuss an algorithm that enables traditional TD learning to handle continuous problems, which is called Gradient Temporal-Difference (GTD) algorithm.

Gradient Temporal Difference Learning. Unlike other TD algorithms, GTD uses the objective function as L_2 form of the following vector:

$$\mathcal{J}(\vec{v}) = \mathbf{E}[\delta\vec{\phi}]^T\mathbf{E}[\delta\vec{\phi}] \tag{2.22}$$

where δ is the TD error defined in (2.38) and $\vec{\phi}$ is the state feature vector defined in (2.4). Its minimum value of 0 can be achieved when $\mathbf{E}[\delta\vec{\phi}] = 0$. Also, the gradient of the objective function is,

$$\nabla_{\vec{v}}\mathcal{J}(\vec{v}) = -2\mathbf{E}[\vec{\phi}(\vec{\phi} - \gamma\vec{\phi}')^T]^T\mathbf{E}[\delta\vec{\phi}] \tag{2.23}$$

Following this, GTD forms the matrix $A = \mathbf{E}[\vec{\phi}(\vec{\phi}-\gamma\vec{\phi}')^T]$ and it can be estimated from t samples as,

$$A_t = \frac{1}{k}\sum_{i=1}^{t}\vec{\phi}_i(\vec{\phi}_i - \gamma\vec{\phi}_i{}')^T. \tag{2.24}$$

Accordingly, we can have the value function parameter updating rule:

$$\vec{v}_{t+1} = \vec{v}_t + \alpha_t A_t^T \delta_t \vec{\phi}_t. \tag{2.25}$$

Neural Networks based Value Function Approximation Methods

In ML domain, NNs are widely utilized to approximate nonlinear value functions. There are many value function approximation methods that have adopted NN as representations for value function [212]. Based on the difference of training methods, these NN based Value Function Approximation methods can be divided into two groups, namely Back-Propagation training based methods and NeuroEvolution based methods.

Figure 2.3 shows a generic feed-forward NN [8]. As can be seen from the figure, the network consists of one output layer with two output neurons, two hidden layers with eight hidden neurons, and one input layer with four input neurons. In addition, weights (real values) are assigned to each link which are the core of a network.

Back-Propagation Training based Value Function Approximation. One of the most successful algorithms to train NNs is the back-propagation algorithm, which enables both forward and backward passes through the network. In one forward pass, signals from inputs nodes are passed layer by layer to the output nodes, meanwhile they are computed with weights along connections and are activated by the activation functions of hidden nodes. After each forward pass, a backward pass starts by computing the error from the output, and then proceeds to distribute the obtained error back to input nodes by computing par-

[8] In this thesis, we are only interested in feed-forward networks, because generally recurrent networks are used for solving partially observable MDP which is out of our research scope.

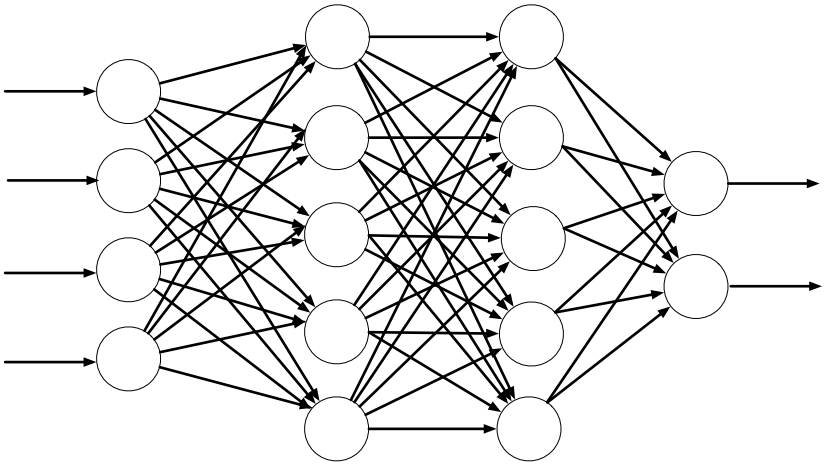


Figure 2.3: A generic feed-forward NN with four input units, two output units, and two hidden layers, adapted from Figure 9.14 [212].

tial derivatives of weights with respect to the error. In this way, an NN can be trained towards the direction of error decrease.

With recent advancements in training deep structured NNs via back-propagation (i.e., Deep Learning (DL)), NNs have become the most promising function approximators for RL algorithms to solve challenging problems [212]. The use of Deep NNs (DNNs) in RL gives rise to many cutting-edge Deep Reinforcement Learning (DRL) algorithms [189, 191, 229, 240], meanwhile has shown many impressive results on solving very difficult problems such as playing Atari games [12]. Note that, DNNs can be used not only to approximate value functions, but also policy as well. In this subsection, we only focus on those algorithms using DNNs for value function approximation, and deep policy search methods are described in Section 2.2.5. Next, we will discuss the most popular two VIS algorithms that uses DNN as value function representations, i.e., Deep Q Network (DQN) [155] and Double Deep Q Network (DDQN) [225].

- *Deep Q Network (DQN)*: DQN is the seminal work that successfully used deep representation to solve complex problems [155]. In comparison to traditional Q learning stated in Section 2.2.2, DQN has several technical advancements. First, it uses experience replay where one step experience may be used for multiple rounds of weights updates thereby increasing

sample efficiency. Second, learning is conducted on randomly selected samples from experience repository that breaks the strong correlations between samples for improved learning reliability.

- *Double Deep Q Network (DDQN)*: DDQN is proposed by [225] to tackle the over-estimation problem in traditional Q learning as well as DQN, because in Q learning and DQN, agent tends to select actions based on over-estimated Q values. In DDQN, a separate target Q network is proposed to estimate Q values, while another online Q network is being trained through temporal difference. DDQN was shown to outperform DQN [225].

- *Other DQN extensions*: Besides DQN itself and DDQN, there have been many extensions proposed to further improve learning performance of VIS. For example, Dueling DQN [230] uses a new network architecture (Dueling Architecture) to estimate both the state value function (i.e., $V(\vec{s})$ function) and the advantage function (i.e., $A(\vec{s}, a)$ function) to construct an estimation of the action value function $Q(\vec{s}, a)$. Average-DQN [10] aims to reduce variances and instability by averaging previously obtained Q-value estimations. Accelerated-DQN [99] uses a constrained optimization technique (i.e., optimality tightening) to accelerate DQN with faster reward propagation meanwhile achieving higher accuracy than the original DQN.

NeuroEvolution Based Value Function Approximation. Another effective way to train value functions in the form of NNs is the NeuroEvolution algorithms introduced in Section 2.1.3. Here, we are interested on NeuroEvolution for value function approximation methods, and leave the NeuroEvolution based policy search in Section 2.2.6. In the literature, a typical NeuroEvolution based VIS methods is NEAT based Q Learning (NEAT+Q) proposed by Whiteson and Stone [232].

- *NEAT based Q learning (NEAT+Q)*: NEAT+Q is also known as evolutionary function Approximation [232], which aims to combine evolutionary

computation technique and temporal-difference methods into a single method. The key idea is to use NEAT to directly evolve value functions instead of action selectors (policies). The value functions are updated by NEAT for topology adaptation and Q learning for weights adaptation. Like most hybrid methods, NEAT+Q enjoys the advantages from both EC and TD algorithms. In particular, it allows the agent to explore useful topologies to enable effective value function approximation. By adapting th topologies, NEAT+Q can automatically discover effective representations for the NN based function approximators which are potentially improve the learning effectiveness. In NEAT+Q, each network (individual) represents a value function which takes state features and actions as input and outputs a corresponding single scalar as the Q value.

2.2.3 Discussion on Value Function Indirect Search

This section gives a brief review of VIS algorithms. It begins with a general introduction to the three categorizations of VIS, and then it shows a general architecture followed by most VIS algorithms. Subsequently, the chapters describes in details on two typical categories of VIS, i.e., Linear Value Function Approximation methods and Neural Network based Value Function Approximation methods. In linear methods, several well-known algorithms are presented including Temporal-Difference methods, Q Learning and SARSA, as well as GTD. In NN based methods, the popular DQN and DDQN have been briefly introduced from a technical view of point.

VIS algorithms are still very popular in the RL research field owing to its simplicity and intuitiveness [212]. Some long-standing open questions have been gradually addressed. For instance, one of these questions claimed by Sutton in [212, 215] is that VIS with linear value function representation lacks of theoretical proofs on convergence, but as we discussed, some VIS algorithms (e.g., GTD and GTD2) have already been proven to converge under suitable conditions [213, 149, 17]. However, there are still several challenging issues for VIS [168, 52].

- *Deterministic Policy vs. Stochastic Policy* VIS seeks only for a determinis-

tic policy [212]. In practice, there may be multiple deterministic policies satisfying (2.14). In contrast, the stochastic policy has an ability to represent multiple deterministic policy trajectories. In other words, one optimal stochastic policy can capture simultaneously multiple optimal deterministic policies. So it may be more desirable to learn a stochastic policy. Moreover, such a policy can often cope with a partially observable environment more gracefully than a deterministic alternative.

In fact, one can straightforwardly learn stochastic policies, this gives rise to the popularity of PDS algorithms. Many recent PDS algorithms show high-level effectiveness in comparison to VIS algorithms [191, 189, 52]. We will systematically discuss PDS in Section 2.2.4

Specifically, in this thesis, we mainly focus on learning stochastic policy. Thus all policies discussed in the following contribution chapters (i.e., Chapter 3- 6) are stochastic.

- *Implicit Policy vs. Explicit Policy*

  VIS aims to learn a deterministic policy, but the policy is only implicitly represented. However, after learning the value function (i.e., the value function converges), it is difficult to recreate the policy to achieve the learned long-term pay-off. Many emerging research works [215, 218] proposed to explicitly represent the policy and directly search in the policy space to address the issue, which appears to be straightforward and effective.

  By explicitly representing the policy associated with value function, we can learn both the policy and the value function, eventually we can obtain an optimal policy as well as an optimal value function. So the reconstruction issue stated above can be avoided directly. In fact, this is actually the so-called Actor-Critic (AC) architecture falling into the PDS family. The architecture will be depicted in Section 2.2.5.

  Particularly, in this thesis, our major research works for developing effective PDS algorithms presented in Chapters 3-6 are all based on AC algorithms.

- *Improper Representation vs. Proper Representation*

  Value function representation are the key to the success of VIS methods. Improper representations may still guarantee the value function converge, but a big gap highly likely appears in-between the true value function and the learned value function, which may lead to "less optimal" performance (the learned policy may converge to local optima). The improperness is highly likely to happen whenever inadequate state features were used. This urges us to develop mechanisms to learn proper state features.

  In fact, with the help of feature learning, proper state features can be learned automatically for constructing proper value function presentations. The success of DRL is a good example of using DL to extract useful features from high-dimensional raw state inputs to enable effective reinforcement learning, which will be discussed in Section 2.3.

  In this thesis, we specifically investigate how to enhance PGS algorithms via evolutionary feature learning in Chapter 7.

- *Stable Learning vs. Unstable Learning*

  Though value function provides a clear signal for learning and evaluating policies in the RL framework, many VIS algorithms have difficulties stabilizing the learning process. An unstable learning on the linear value function can lead to the divergence on specially designed environments [16] if there are no proper corrections [142]. In fact, the problem is faced not only by linear methods but also by DNN based methods. DQN is shown to diverge due to the sparse reward distribution [44] or over-estimated Q values [225].

  The instable value function learning has been a long-standing issue. For linear methods, though a recent work [214] has proposed a clear solution of the issue, the unstable learning can still occur resulting in the divergence in value function learning with improper settings on the hyper-parameters [213]. For DNN based methods, many techniques have been proposed to stabilize value function learning, such as experience replay with priority [155] and dueling networks[230]. Nevertheless, they still

cannot guarantee the stability of learning on value functions, which remains a big room to be studied.

In Chapter 6 of the thesis, we particularly are interested to study approaches to stabilizing the value function learning and hence eventually improving policy learning.

2.2.4 Policy Direct Search

Very different from VIS, PDS searches directly in the policy space towards the optimal policy in (2.14). PDS can address some limitations of VIS as discussed in Section 2.2.3. In the past decades, massive PDS algorithms have been proposed [15, 121, 183, 21, 2, 170, 215, 31, 192, 117, 119, 168, 20, 160, 143, 198, 53, 169, 194, 63, 8], and they have been successfully applied to solving many challenging RL problems (e.g., robotics [52, 170, 53], locomotion tasks and intelligent game-play [189, 161, 190, 191]).

Figure 2.4 shows a taxonomic categorization for PDS algorithms [52]. As seen from the figure, the PDS methodology is classified into two groups, namely model-free and model-based policy direct search. Distinguished by the policy updating process, we can further divide each group into gradient-based search or gradient-free search.

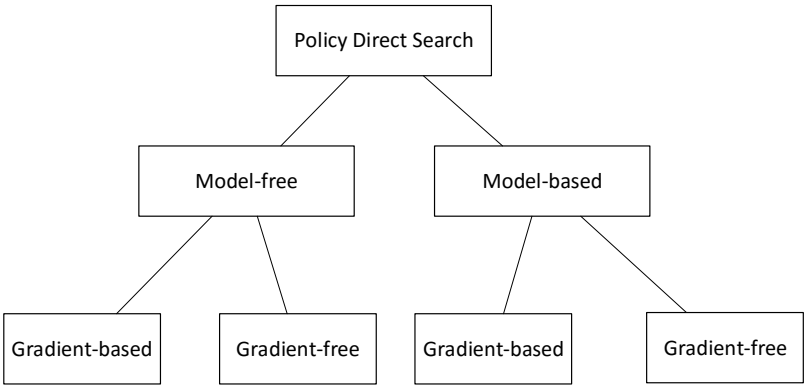


Figure 2.4: Categorization for Policy Direct Search, adapted from Figure 1.1 [52].

Model-free vs. Model-based

Model-free PDS learning is driven completely by the samples (i.e., sampled trajectories) obtained by an agent from interacting with an environment. A general model-free policy search framework [52, 193] is given in Figure 2.5. Following the trial-and-error process, policy improvement is achieved from either explicitly estimating gradients for gradient-descent policy update in policy space or conducting population-based random search.

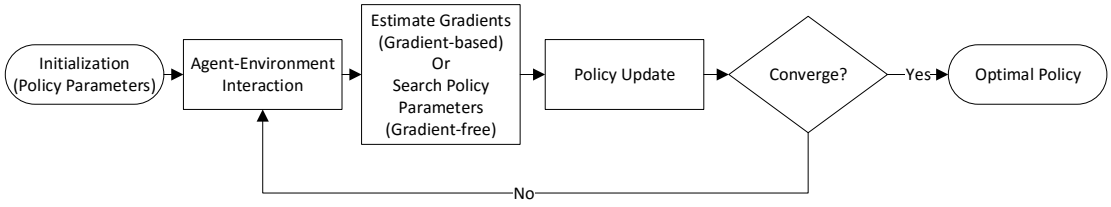


Figure 2.5: Model-free Policy Direct Search Framework, prepared based on Figure 4 in [193].

On the other hand, model-based PDS [52, 51] attempts to learn a predicted environment model first, then to derive the optimal policy directly from the learned model. Figure 2.6 gives a general framework for model-based PDS. The model learning requires to train a model based on observed data to forecast the environment behaviors. Subsequently through model-based internal simulation, the algorithm evaluates policies with the learned model, and improves the policy by following a similar approach to model-free PDS. The gradients here can be computed analytically from the model. When the algorithm converges, the learned policy will guide new rounds of environment sampling to obtain more data to learn a more precise model. The process terminates until the problem is solved.

In the thesis, we are only interested in model-free PDS due to several reasons. First, it is usually very impractical to obtain an environmental model because of the uncertainties of the environment and extremely high computational cost. Second, an inaccurate model with modeling bias may easily trap the

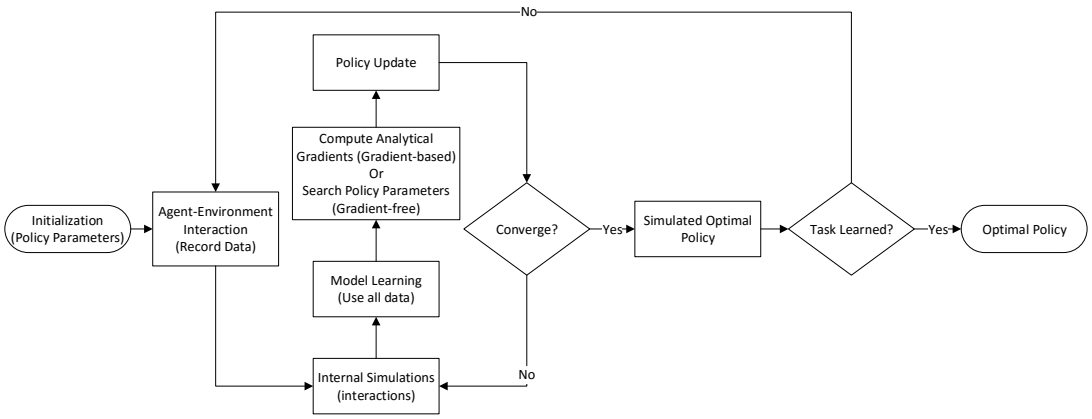


Figure 2.6: Model-based Policy Direct Search Framework, prepared based on Figure 4 in [193], Figure 3.1 in [52] and Algorithm 1 in [51].

learning process to poor local optima. In consequence, in the following, we will only concentrate on model-free policy search techniques.

Gradient-based vs. Gradient-free

Gradient-based algorithms require to directly compute/estimate gradients of the expected long-term pay-off with respect to the policy parameters. In contrast, gradient-free algorithms do not rely on the policy gradients. Many methods fall into the gradient-free category, such as cross-entropy optimization based policy search [194] and evolutionary computation based policy search [232].

Episode Learning vs. Step Learning

Another way to classify the PDS methods is to divide PDS according to its training strategy, i.e., Episode Learning and Step Learning [212]. In episode tasks, the goal is to take the agent from a starting state to a goal state, thus the term of episode refers to a fixed length of sequence of state transitions [212, 236]. Episode learning means that, after one episode (or multiple episodes) terminates, the value function or policy learning is conducted based on the samples obtained from the episode (or multiple episodes) [236]. On the other hand, Step

learning refers to a learning strategy where the algorithm learns the policy (or value functions) at every n steps (commonly $n = 1$) [236]. Note that, if n equals to the length of an episode, the step learning essentially converts to the episode learning. In this thesis, we will focus on step learning strategy in Chapter 4, Chapter 6. Also, we will investigate episode learning strategy in Chapter 5. In Chapter 7, both strategies are used for policy learning.

2.2.5 Model-free Gradient-based Policy Direct Search

Model-free gradient-based PDS is a widely-used effective framework for searching the optimal policy parameter that satisfies (2.29) [52, 215]. The framework is also known as *PGS* [9] [52, 215, 31].

In the following, we will introduce the concepts and examples of PGS algorithms. We will first introduce the two popular representations of policy in the literature which are also used in this thesis. After that, we will introduce both the general PGS framework,as well as the specific Actor-Critic (AC) framework for Direct Policy Optimization (DPO). Following the concepts of PGS, we will introduce the technical details of several representative PGS algorithms.

Policy Representations

The effectiveness of PGS depends on the choices of policy representation. There are three typical representations, linear representations, nonlinear representations, and dynamic movement primitives [52]. In this thesis, we primarily focus on linear representations and Neural Networks based nonlinear representations, which will be introduced in the following two subsections.

Linear Representation. Similar to value function representation, the policy representation is said to be linear in its parameter space, but may be nonlinear in the state features. The policy function with a linear policy representation

[9] In the following text, we just use PGS to represent Model-free Gradient-based PDS for expressive convenience.

can be described as below,

$$\begin{aligned} a &\sim \pi_{\vec{\theta}}(a|\vec{s}) \\ \mathbf{E}[a|\pi] &= \vec{\theta} \cdot \vec{\phi}(\vec{s}) \end{aligned} \tag{2.26}$$

where $\vec{\theta} \in \Theta$ denotes the policy parameter, and the policy π is only linearly dependent on $\vec{\theta}$. Additionally $\vec{\phi}(\vec{s})$ is defined as the state features in (2.4).

Neural Network based Representation. In contrast to linear representation, NN based nonlinear representation is the dominant representation in the RL field owing to its great power on solving very complex problems. Very similar to the policy represented linearly, a NN based policy representation can be described as below,

$$\begin{aligned} a &\sim \pi_{\vec{\theta}}(a|\vec{s}) \\ \mathbf{E}[a|\pi] &= f(\vec{s}, \vec{\theta}) \end{aligned} \tag{2.27}$$

where $\vec{\theta} \in \Theta$ denotes the policy parameter, and the policy π is a function f described by a NN over state inputs $\vec{s}$ with respect to its parameters $\vec{\theta}$.

The Goal of PDS. Based on the policy representations (either linear or nonlinear), we can reformulate the expected long-term pay-off for PDS as,

$$\mathcal{J}(\vec{\theta}) = \mathbf{E}[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | \pi_{\vec{\theta}}]. \tag{2.28}$$

The goal of PDS is to find the optimal policy parameter [10] such that,

$$\vec{\theta}^{\star} = \underset{\vec{\theta}}{\operatorname{argmax}}\, \mathcal{J}(\vec{\theta}). \tag{2.29}$$

In what follows, we use $\mathcal{J}(\pi_{\vec{\theta}})$ and $\mathcal{J}(\vec{\theta})$ interchangeably, which represents the expected long-term pay-off when the policy $\pi_{\vec{\theta}}$ parameterized by $\vec{\theta}$ has been followed.

[10] Different from $*$ used previously for "*true-optimal*", we use $\star$ to denote "*near-optimal*". This is because, after representing π with a set of parameters $\vec{\theta}$, we may only obtain a "*near-optimal*" policy sharing the similar performances (e.g., value functions) with the "*true-optimal*" policy. However, it is **NOT** "*true-optimal*" at all, thus for the precision of expression, we distinguish them with different symbols.

General Policy Gradient Search Framework

The general PGS framework follows a basic idea,

$$\Delta\vec{\theta} \propto \nabla_{\vec{\theta}}\mathcal{J}(\vec{\theta}), \tag{2.30}$$

which means that the policy parameters updating is proportional to the gradients of the expected long-term pay-off with respect to policy parameters. Note that, $\nabla_{\vec{\theta}}\mathcal{J}(\vec{\theta})$ is not directly computable, but we can use sampled trajectories to construct unbiased estimators of it.

To conduct the PGS, firstly let us explicitly present the expected long-term pay-off by following (2.28) as,

$$\begin{aligned}\mathcal{J}(\vec{\theta}) &= V^{\pi}(\vec{s}_0) \\ &= \mathbf{E}[\textstyle\sum_{t=0}^{\infty}\gamma^t r_{t+1}|\vec{s}_0 = s, \pi_{\vec{\theta}}(a|\vec{s})] \\ &= \textstyle\int_{\vec{s}\in\mathcal{S}} p^{\pi}(\vec{s}) \int_{a\in\mathcal{A}(\vec{s})} \pi_{\vec{\theta}}(a|\vec{s})\mathcal{R}(\vec{s},a,\vec{s}')dad\vec{s},\end{aligned} \tag{2.31}$$

where $p^{\pi}(\vec{s}) = \sum_{t=0}^{\infty}\gamma^t Pr(\vec{s}_t = \vec{s}|\vec{s}_0, \pi_{\vec{\theta}}(a|\vec{s}))$ is a discounted weighting of encountering states (a.k.a, stationary state distribution) initiating from $\vec{s}_0$ following a certain policy π.

Next, to search the policy by PGS, we are required to obtain unbiased estimations of $\nabla_{\vec{\theta}}\mathcal{J}(\vec{\theta})$. One of the most influential way to construct this estimation is the so-called *Policy Gradient Theorem* proposed by [215]. Most traditional PGS algorithms or recent state-of-the-art PGS are based on or extended from *Policy Gradient Theorem* (PGT), which will be introduced in details below.

Policy Gradient Theorem (PGT). *Policy Gradient Theorem* proposed by Sutton [215] is given below,

$$\begin{aligned}\frac{\partial\mathcal{J}(\vec{\theta})}{\partial\vec{\theta}} &= \textstyle\int_{\vec{s}\in\mathcal{S}} p^{\pi}(\vec{s}) \int_{a\in\mathcal{A}(\vec{s})} \frac{\partial\pi_{\vec{\theta}}(a|\vec{s})}{\partial\vec{\theta}}\mathcal{R}(\vec{s},a,\vec{s}')dad\vec{s} \\ &= \textstyle\int_{\vec{s}\in\mathcal{S}} p^{\pi}(\vec{s}) \int_{a\in\mathcal{A}(\vec{s})} \frac{\partial\pi_{\vec{\theta}}(a|\vec{s})}{\partial\vec{\theta}}Q^{\pi}(\vec{s},a)dad\vec{s}\end{aligned} \tag{2.32}$$

where $\int_{a\in\mathcal{A}(\vec{s})} \frac{\partial\pi_{\vec{\theta}}(a|\vec{s})}{\partial\vec{\theta}}Q^{\pi}(\vec{s},a)da$ is an unbiased estimate of $\frac{\partial\mathcal{J}(\vec{\theta})}{\partial\vec{\theta}}$ when $\vec{s}$ comes from sampled trajectories of π.

Obviously, $Q^{\pi}(\vec{s},a)$ is unknown and must be estimated. In the same work [215], Sutton also proposed *Compatible Function Approximation Theorem* as a suitable way to estimate $Q^{\pi}(\vec{s},a)$.

Compatible Function Approximation Theorem. Before discussing the Compatible Function Approximation theorem, we need to define a function approximator for Q^π. According to (2.19), we can linearly approximate Q^π as,

$$Q^\pi(\vec{s}, a) \approx \tilde{Q}^\pi_{\vec{\omega}}(\vec{s}, a) = \vec{\omega}^{\pi T} \cdot \Phi(\vec{s}, a), \tag{2.33}$$

where $\Phi(\vec{s}, a)$ is compatible features defined below in (2.37).

Following (2.33), the Compatible Function Approximation theorem [215] is presented below:

Given two conditions,

(1) *Compatibility Condition*:

$$\frac{\partial \tilde{Q}^\pi_{\vec{\omega}}(\vec{s}, a)}{\partial \vec{\omega}} = \frac{\partial \ln \pi_{\vec{\theta}}(a|\vec{s})}{\partial \vec{\theta}}, \tag{2.34}$$

(2) *Mean Squared Error (MSE) Minimization Condition*:

$$\varepsilon(\vec{\omega}) = \mathbf{E}[(Q^\pi(a|\vec{s}) - \tilde{Q}^\pi_{\vec{\omega}}(\vec{s}, a))^2|\vec{\omega}], \tag{2.35}$$

where we require to find $\vec{\omega}^\star = \text{argmin}_{\vec{\omega}}\, \varepsilon(\vec{\omega})$,

if both conditions are satisfied, policy gradient can be precisely determined as,

$$\frac{\partial \mathcal{J}(\vec{\theta})}{\partial \vec{\theta}} = \int_{\vec{s} \in \mathcal{S}} p^\pi(\vec{s}) \int_{a \in \mathcal{A}(\vec{s})} \frac{\partial \pi_{\vec{\theta}}(a|\vec{s})}{\partial \vec{\theta}} \tilde{Q}^\pi_{\vec{w}^\star}(\vec{s}, a) da d\vec{s}, \tag{2.36}$$

and we also have,

$$\Phi(\vec{s}, a) = \frac{\partial \ln \pi_{\vec{\theta}}(a|\vec{s})}{\partial \vec{\theta}}, \tag{2.37}$$

Consequently, based on (2.36), many PGS algorithms have been developed. We will further investigate several representative works in Section 2.2.5.

Actor-Critic Architecture

The AC architecture is widely recognized as a sub-branch of PGS framework [212, 31, 52]. An AC algorithm maintains an explicit policy (i.e., actor) separately from the value function (i.e., critic). The value function is used for policy

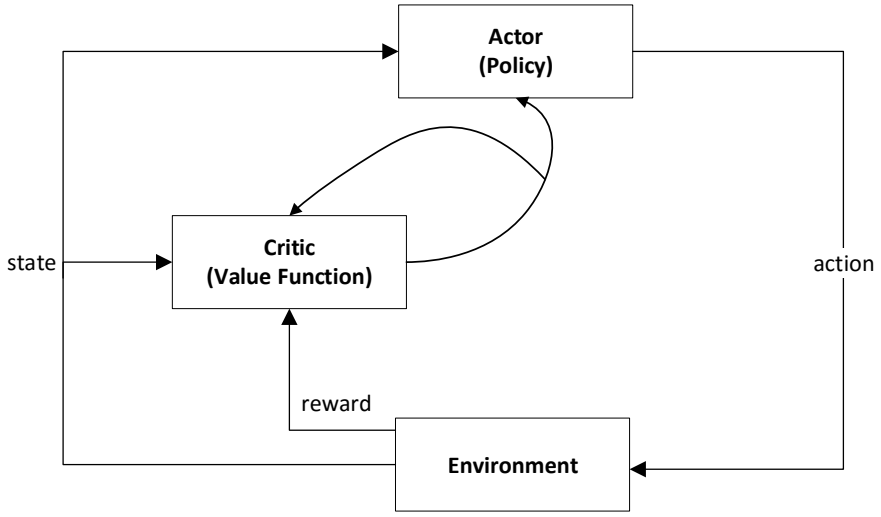


Figure 2.7: Actor-Critic Architecture, adapted from Figure 6.15 in [212].

evaluation. The agent learns both the value function and the policy. Figure 2.7 shows a typical AC architecture. As seen from Figure 2.7, the general actor-critic architecture is using state value function V^π to criticize the action chosen by policy, normally the criticizing credit is defined as the TD error formulated below,

$$\delta_t^\pi = \mathcal{R}(\vec{s}_t, a_t, \vec{s}_{t+1}) + \gamma V^\pi(\vec{s}_{t+1}) - V^\pi(\vec{s}_t), \tag{2.38}$$

where $V^\pi(\vec{s}_t)$ is the expected long-term pay-off for current state $\vec{s}_t$, and $V^\pi(\vec{s}_{t+1})$ is for next state $\vec{s}_{t+1}$ observed when following the policy π to take action a_t at state $\vec{s}_t$.

Based on δ_t^π in (2.38), if the critic is approximated linearly, it can be learned by following an incremental updating process,

$$\vec{v}_{t+1} = \vec{v}_t + \alpha_t \delta_t^\pi \vec{\phi}(\vec{s}_t), \tag{2.39}$$

with the aim of minimizing

$$\varepsilon(\vec{v}) = ||r_{t+1} + \gamma \vec{v} \cdot \vec{\phi}(\vec{s}_{t+1}) - \vec{v} \cdot \vec{\phi}(\vec{s}_t)||^2. \tag{2.40}$$

Simultaneously, actor learning in AC algorithms is realized in general by following the direction of critic improvements, which essentially estimates the gradient of cumulative rewards with respect to all policy parameters, also known

as the policy gradient, i.e., $\nabla_{\vec{\theta}}\mathcal{J}(\vec{\theta}_t)$ in (2.36),

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \beta_t \nabla_{\vec{\theta}}\mathcal{J}(\vec{\theta}_t), \tag{2.41}$$

so as to maximize (2.31).

AC methods combine the advantages of both VIS and PDS as mentioned in Section 2.2.3. It can learn stochastic policies but may require less samples comparing to PDS algorithm without explicitly maintaining value functions (e.g., REINFORCE [238]). Thus, AC architecture has become one of the most popular RL framework recently [189].

Direct Policy Optimization

Besides the PGT, another popular way to estimate policy gradients is called Direct Policy Optimization (DPO) [173, 111]. In DPO, the performance of a policy π with respect to $\vec{\theta}$ on a trajectory [11] is defined as,

$$J(\vec{\theta}) = \int_{\xi} \pi_{\vec{\theta}}(\xi) R(\xi) \mathrm{d}\xi \tag{2.42}$$

where $R(\xi)$ is the observed total rewards for the given trajectory ξ.

The ultimate goal for RL is hence to identify the optimal policy parameters $\vec{\theta}^*$ so as to achieve the maximum learning performance in (2.42). Driven by this goal, some RL algorithms choose to optimize a performance lower bound $L_{\vec{\theta}}(\vec{\theta'})$ as derived below [117, 111],

$$\begin{aligned} \log J(\vec{\theta'}) &= \log \int_{\xi} \frac{Pr_{\vec{\theta}}(\xi)}{Pr_{\vec{\theta}}(\xi)} Pr_{\vec{\theta'}}(\xi) R(\xi) \mathrm{d}\xi \\ &\geq \int_{\xi} Pr_{\vec{\theta}}(\xi) R(\xi) \log \frac{Pr_{\vec{\theta'}}(\xi)}{Pr_{\vec{\theta}}(\xi)} \mathrm{d}\xi + \mathbb{C} \\ &\propto \int_{\xi} Pr_{\vec{\theta}}(\xi) R(\xi) \log Pr_{\vec{\theta'}}(\xi) \mathrm{d}\xi = L_{\vec{\theta}}(\vec{\theta'}) \end{aligned} \tag{2.43}$$

where $\vec{\theta}$ refers to the policy parameters before update and $\vec{\theta'}$ stands for the updated policy parameters. Given fixed $\vec{\theta}$, the aim is to find suitable $\vec{\theta'}$ that maximize $L_{\vec{\theta}}(\vec{\theta'})$. Thus, the DPO algorithms have the goal to optimize directly the

[11] In RL, a trajectory ξ is a sequence of state transitions over a set of contiguous timestamps from a single episode.

performance lower bound $L_{\vec{\theta}}(\vec{\theta'})$. Many popular algorithms are developed following this framework, such as Policy learning by weighting exploration with the returns (PoWER) [117] and Trust Region Policy Optimization (TRPO) [189], which will be introduced in Section 2.2.5.

Policy Gradient Search Methods

In this subsection, we will introduce the technical details about several typical PGS algorithms, starting from the earlies PGS (i.e., REINFORCE) to the latest DPO based algorithms such as TRPO and Proximal Policy Optimization (PPO) [191].

REINFORCE. REINFORCE proposed by Williams [238] is the first policy gradient methods, which can be considered as a special case of PGT [215]. In REINFORCE, it still uses stochastic gradient descent to update the policy parameters and following the PGT as stated in Section 2.2.5. However, it does not construct the Q approximator but uses the sampled total return as an unbiased estimation of $Q^{\pi_{\vec{\theta}}}(\vec{s}_t, a_t)$. The gradient estimation in REINFORCE is defined as,

$$\nabla_{\vec{\theta}} \mathcal{J}(\vec{\theta}) = \nabla_{\vec{\theta}} \ln \pi_{\vec{\theta}(\vec{s}_t, a_t)} R_t. \tag{2.44}$$

where R_t is the actual return. Although REINFORCE enjoys the benefits of low biases, its learned policy actually yields very high variances.

Actor-Critic Algorithms. To address the high-variance problem, AC methods have been developed to balance the bias-variance trade-off. AC methods are to learn from total rewards and/or TD errors to improve value functions so as to search good policies. Due to this popularity, a great number of AC algorithms are proposed, such as Regular Actor-Critic (RAC) [31], Natural Actor-Critic (NAC) [31, 170], Asynchronous Advantage Actor-Critic (A3C) [154], Advantage Actor-Critic(A2C) [56], Actor-Critic using Kronecker-factored Trust Region(ACKTR) [240] and so forth.

- *Regular Actor-Critic (RAC)*: Following the general AC architecture stated

in above, RAC derives the TD-error as,

$$\delta_t^{\pi} = \mathcal{R}(\vec{s}_t, a_t, \vec{s}_{t+1}) + \gamma \upsilon_t^{\pi T} \cdot \vec{\phi}(\vec{s}_{t+1}) - \upsilon_t^{\pi T} \cdot \vec{\phi}(\vec{s}_t), \tag{2.45}$$

where υ_t^{π} denotes the state value function parameter at time t, and $\upsilon^{\pi T} \cdot \vec{\phi}(\vec{s})$ represents an estimated value function as shown in (2.19).

Based on the TD error derived above, RAC estimates the policy gradient as

$$\frac{\partial \mathcal{J}(\vec{\theta})}{\partial \vec{\theta}} = \mathbf{E}[\delta \Phi^{\pi}(\vec{s}, a) | \vec{\theta}]. \tag{2.46}$$

- *Natural Actor-Critic (NAC)*: In NAC algorithms, the policy gradients are transformed to the natural gradients proposed by Amari [5], which give more reliable estimation of policy gradients. NAC first computes the natural-gradient estimation from the inverse of a Fisher Information Matrix G , i.e.,

$$\frac{\partial \mathcal{J}(\vec{\theta})}{\partial \vec{\theta}} = G^{-1}(\vec{\theta}) \mathbf{E}[\delta \Phi^{\pi}(\vec{s}, a) | \vec{\theta}]. \tag{2.47}$$

. where the Fisher Information matrix G is computed by,

$$\begin{aligned} G(\vec{\theta}) &= \mathbf{E}[\Phi(\vec{s}, a) \Phi(\vec{s}, a)^T | \vec{\theta}] \\ &= \mathbf{E}[\nabla_{\vec{\theta}} \ln \pi_{\vec{\theta}}(a|\vec{s}) \nabla_{\vec{\theta}} \ln \pi_{\vec{\theta}}(a|\vec{s})^T | \vec{\theta}]. \end{aligned} \tag{2.48}$$

However, the computation of $G^{-1}(\vec{\theta})$ is often extremely complicated and error-prone. Following the idea of NAC, several algorithms are proposed to use different ways to estimate the fisher information matrix G. For example, Bhatnagar et.al. [31] proposed to use Sherman-Morrison to incrementally update G, where the inverse of the fisher information matrix is initialized as $G_0^{-1}(\vec{\theta}) = kI$ where $k \in \mathbb{R}$ as a tunable meta parameter. Further, G_t^{-1} is incrementally updated within the learning process by following the Sherman-Morrison matrix inversion lemma as,

$$G_t^{-1} = \frac{1}{1-\alpha}[G_{t-1}^{-1} - \alpha \frac{(G_{t-1}^{-1} \Phi(\vec{s}, a))(G_{t-1}^{-1} \Phi(\vec{s}, a))^T}{1 - \alpha_t + \alpha_t \Phi(\vec{s}, a)^T G_{t-1}^{-1} \Phi(\vec{s}, a)}]. \tag{2.49}$$

Moreover, Peter [170] computes the inverse directly by least square methods. Recently, ACKTR [240] adopts Kronecker-factored approximation

to estimate the natural gradients. With help of trust region optimization technique to restrict the policy updates in a reasonable level, ACKTR have shown very prominent results on a number of benchmark problems, such as Atari games. Another interesting work under the AC framework has been proposed by Rajeswaran et.al, [178]. In their work, a new NAC algorithm with a linear policy has been developed to solve many difficult RL tasks which previously was supposed to be only solvable by the agent with deeply structured policy.

- *Asynchronous Advantage Actor-Critic(A3C)/Advantage Actor-Critic(A2C)*: A3C [154] is one of the cutting-edge AC algorithms, whereas A2C [56] is just a synchronous and deterministic variant of A3C. Both algorithms have achieved state-of-the-art performance on playing Atari games and controlling locomotion tasks. Here, we will describe A3C as an example.

  A3C has three key characteristics, *Asynchronous*, *Actor-Critic*, and *Advantage*. Referring to *Asynchronous*, unlike other DRL algorithms, A3C uses multiple agents (i.e., multiple neural networks) to learn from multiple environments. There is a global network, also each individual agent has an independent network to interact with its own environment instance. The advantages of this design are, (1) parallel execution greatly speeds up the learning, and (2) experiences obtained by multiple independent agents increase the diversity of learning. In addition, A3C also benefits from training the advantage function below that determines the preference of selecting any action in comparison to the average (expected).

$$A^{\pi_{\vec{\theta}}}(\vec{s}, a) = Q^{\pi_{\vec{\theta}}}(\vec{s}, a) - V^{\pi_{\vec{\theta}}}(\vec{s}) \tag{2.50}$$

Direct Policy Optimization Algorithms. Here, we explain the technical details of three typical DPO algorithms, including PoWER, TRPO and PPO.

- *Expectation Maximization based PGS*: The key idea of Expectation Maximization based Policy Gradient Search (EM-PGS) is to construct lower bound for the policy, and instead of maximizing $\mathcal{J}$ in (2.42), the lower bound is expected to be optimized during learning [52]. In line with this

idea, EM-PGS derives a lower bound shown below,

$$\ln \mathcal{J}(\vec{\theta}) \geq \mathcal{L}^{\vec{\theta}}(\vec{\theta'}) = \mathrm{E}_{p^{\vec{\theta'}}(\tau)}[R(\tau) \ln p^{\vec{\theta}}(\tau)], \tag{2.51}$$

where τ represents any trajectory, $\vec{\theta'}$ is the old policy parameter (i.e., sampling policy) whereas $\vec{\theta'}$ is the new policy parameter, $p^{\vec{\theta}}(\tau)$ and $p^{\vec{\theta'}}(\tau)$ are the probabilities of generating the trajectory τ following the new policy $\pi^{\vec{\theta}}$ and $\pi^{\vec{\theta'}}$ respectively.

Based on (2.51), if $\vec{\theta'}$ and $\vec{\theta}$ are close enough, we can estimate the policy gradient estimation according to

$$\nabla_{\vec{\theta}} J(\vec{\theta}) = \lim_{\vec{\theta} \to \vec{\theta'}} \nabla_{\vec{\theta}} \mathcal{L}^{\vec{\theta}}(\vec{\theta'}) = \mathrm{E}_{p^{\vec{\theta'}}}[\sum_{t=0}^{T-1} Q_t^{\pi}(\vec{s}_t, a_t) \nabla_{\vec{\theta}} \ln \pi^{\vec{\theta}}(\vec{s}_t, a_t)] \tag{2.52}$$

In this thesis, we consider a typical EM-PGS algorithm (i.e., PoWER), because it has shown proven effectiveness on many practical applications like robotic controls [117].

- *Trust Region Policy Optimization*: Similar to EM-PGS, TRPO [189] is to optimize a lower bound on learning performance according to a surrogate objective function $\mathcal{L}^{\vec{\theta'}}(\vec{\theta})$ subject to some behavioral constraints as described below,

$$\begin{aligned} &\text{maximize}_{\vec{\theta}}[\mathcal{L}^{\vec{\theta'}}(\vec{\theta})] \\ &\text{subject to } \mathrm{D}_{KL}(\pi^{\vec{\theta'}} || \pi^{\vec{\theta}}) \leq \delta \end{aligned} \tag{2.53}$$

where

$$\mathcal{L}^{\vec{\theta'}}(\vec{\theta}) = \eta(\vec{\theta'}) + \mathrm{E}_{\rho^{\vec{\theta}}(\vec{s})} \mathrm{E}_{a \sim \pi^{\vec{s}}} A_{\vec{\theta'}}(\vec{s}, a), \tag{2.54}$$

with $\mathrm{D}_{KL}(\pi^{\vec{\theta'}} || \pi^{\vec{\theta}})]$ is the expected KL divergence between the old policy parameter $\vec{\theta'}$ and the new policy parameter $\vec{\theta}$, $\rho^{\vec{\theta}}(\vec{s})$ is state visiting frequency which is approximated by using $\rho^{\vec{\theta'}}(\vec{s}_0)$. Note, $\mathrm{D}_{KL}(\pi^{\vec{\theta'}} || \pi^{\vec{\theta}})]$ is also known as the Trust Region which guarantees policy parameter updating to be reasonably small (i.e., "be trusted") so as to stabilize policy learning. TRPO is a prominent PGS approach that has been successfully applied to many difficult RL problems like playing Atari games [189].

- *Proximal Policy Optimization*: As discussed previously, TRPO has already shown great effectiveness on many difficult RL tasks. However, it is tricky to handle the TRPO constraint given in (2.54), no matter whether to use it as a hard constraint as what TRPO practically does, or to use it as a penalty with a fixed coefficient β [191]. To solve the constraint issue, PPO was proposed. In PPO, a simple approach has been adopted to handle the constraint by using a clipped probability ratio $\nu_t(\vec{\theta})$ as an importance weight.

  In particular, PPO defines a different learning objective from TRPO as,

$$J^{\pi} = \mathbf{E}_t\Big[\mathrm{L}_t^{\mathrm{CLIP}}(\vec{\theta}) - c_1\mathrm{L}_t^{VF}(\vec{\theta}) + c_2\mathrm{S}(\pi_{\vec{\theta}}(\vec{s}_t))\Big] \tag{2.55}$$

  where

$$\mathrm{L}_t^{\mathrm{CLIP}}(\vec{\theta}) = \mathbf{E}_t\Big[\min\Big(\nu_t(\vec{\theta})A^{\pi}(\vec{s}_t,a_t), \mathrm{clip}\big(\nu_t(\vec{\theta}), 1-\varepsilon, 1+\varepsilon\big)A^{\pi}(\vec{s}_t,a_t)\Big)\Big], \tag{2.56}$$

  and

$$\mathrm{L}_t^{\mathrm{VF}}(\vec{\theta}) = \mathbf{E}_t\Big[||V^{\vec{\phi}_{old}}(\vec{s}_t) - \hat{V}_t||^2], \tag{2.57}$$

  additionally, c_1, c_2 are coefficients, S is the entropy bonus.

  By optimizing the surrogate policy learning objective $\mathrm{L}_t^{\mathrm{CLIP}}(\vec{\theta})$, PPO is able to control the policy update reliably and adaptively. If $A^{\pi}(\vec{s}_t,a_t) > 0$, the ν_t will be increased to encourage to select current action a_t. This is because that a_t is the better-than-average action according to the definition of the advantage function. On the other hand, if $A^{\pi}(\vec{s}_t,a_t) < 0$, it implies that a worse-than-average action a_t has been selected, thus ν_t will be decreased to discourage to select the action. Note that, the clip function limits the increase/decrease on ν_t within the range $[1-\varepsilon, 1+\varepsilon]$.

  In practice, PPO optimizes the entire objective (2.55) to seek for optimal policy. By doing so, accurate value functions, in particular advantage function, can be obtained. Meanwhile the entropy term encourages continued exploration [191].

2.2.6 Model-free Gradient-free Policy Direct Search

Under the model-free framework, another important method to conduct PDS is using gradient-free techniques [236], including Cross-Entropy (CE) Optimization, Random Search Optimization (RSO), and EC. Each approach is discussed briefly below

Cross-Entropy Optimization based Policy Direct Search

CE optimization method is considered as a gradient-free optimization technique in the optimization domain [48], which has been widely used to solve combinatorial and multi-extremal continuous optimization problem. Let us consider the following general maximization problem as below,

$$S(\vec{x}^*) = \Gamma^* = \max_{\vec{x} \in \mathcal{X}} S(\vec{x}), \tag{2.58}$$

where $\mathcal{X}$ is a finite set of states, S is real-valued performance function on $\mathcal{X}$, and the maximum is denoted as Γ^*. Next, the CE method is to associated an estimation problem to (2.58). To do so, one firstly define a collection of indicator functions $I_{\{S(\vec{x}) \geq \Gamma\}}$ on $\mathcal{X}$ with various thresholds $\Gamma \in \mathbb{R}$. Subsequently, let $\{f(\vec{\cdot}, \vec{v}, \vec{v} \in \mathcal{V})\}$ be a family of discrete Probability Density Functions (PDFs) over $\mathcal{X}$ with parameters of $\vec{v}$. Then, for a certain $\vec{u} \in \mathcal{V}$, (2.58) can be associated with a problem of estimating the number

$$l(\Gamma) = \mathbb{P}_{\vec{u}}(S(\mathbf{X}) \geq \Gamma) = \sum_{\vec{x}} I_{\{S(\vec{x}) \geq \Gamma\}} f(\vec{x}; \vec{u}) = \mathbf{E}_{\vec{u}} I_{\{S(\mathbf{X}) \geq \Gamma\}}, \tag{2.59}$$

where $\mathbb{P}_{\vec{u}}$ is the probability measure under which the random state $\mathbf{X}$ has a pdf $f(\vec{\cdot}; \vec{u})$. The estimation problem of (2.59) is called *associated stochastic problem* (ASP). After transforming (2.58) to simpler (2.59), CE first generates a random sample from the data distribution and then randomly updates the parameters of the data distribution to produce a better sample. The process repeats until the random sequence of solutions converges to the optimal or near-optimal. Following the idea of CE, there are some PGS methods proposed [194, 40].

We will brief the key ideas of the first CE based PDS proposed in [194]. In this work, CE first generates N random trajectories $\mathcal{T} = \{\xi_0, \xi_1, \ldots, \xi_N\}$ using a

parametric policy $\pi_{\vec{\theta}}$ and meanwhile compute the performance (i.e., the average total reward obtained by the policy). Afterwards, it updates the parameters $\vec{\theta}$ on the basis of those collected trajectories by solving the stochastic program below,

$$\vec{\theta} = \underset{\vec{\theta}}{\operatorname{argmax}} \frac{1}{N} \sum_{i=1}^{N} \vec{I}_{S(\xi_i) \geq \gamma_t} \ln \pi(\xi_i | \vec{\theta}), \tag{2.60}$$

where I stands for the indicator function, and γ_t is predefined threshold. By iteratively conduct these two steps, the algorithm is able to constantly improve policy parameters.

Random Search Optimization based Policy Direct Search

Random Search (RS) is one of the simplest gradient-free optimization techniques [248], which randomly selects a direction with the uniform distribution in the parameter space and then updates the parameters along the direction. RS normally are considered slow in convergence speed, which hence is seldom used in RL domain until very recent. Mania et al. [146] shows that a simple random search algorithm with only linear policy can be competitive to a number of state-of-the-art PGS or EC algorithms equipped with deep-layered policy.

The algorithm is named as Augmented Random Search (ARS), which is developed on the basis of Basic Random Search (BRS) [248]. In BRS, the algorithm approximates a finite difference along the random direction as follows,

$$\frac{r(\pi_{\vec{\theta}+\nu\vec{\delta}}, \epsilon_1) - r(\pi_{\vec{\theta}-\nu\vec{\delta}}, \epsilon_2)}{\nu}, \tag{2.61}$$

where $r(\pi_{\vec{\theta},\xi})$ is the total reward achieved on one trajectory ξ generated by the policy $\pi_{\vec{\theta}}$,ϵ_1 and ϵ_2 are two i.i.d random variables, ν is a positive real number, and $\vec{\delta} \sim \mathcal{N}(\vec{0}, \vec{I})$. Next, BRS uses a line search to take one step updating in the direction.

ARS follows a similar strategy of BRS, it starts with the first iteration $j = 0$ and samples noises $\delta_1, \delta_2, \ldots, \delta_N$ of the same size as the policy parameters $\vec{\theta}_j$ with i.i.d standard normal entries. Next, ARS collects $2N$ trajectories and corresponding rewards following the policies, $\pi_{j,k,+}(\vec{\theta}_j) = \pi_{\vec{\theta}_j} + \nu\vec{\delta}_k$ and $\pi_{j,k,-}(\vec{\theta}_j) =$

$\pi_{\vec{\theta}_j} - \nu\vec{\delta}_k$, where $k \in \{1, 2, \ldots, N\}$. Afterward, ARS updates the policy parameters by

$$\vec{\theta}_{j+1} = \vec{\theta}_j + \frac{\alpha}{N}\sum_{k=1}^{N}[r(\pi_{j,k,+}) - r(\pi_{j,k,-})]\delta_k. \tag{2.62}$$

Lastly, ARS enters the next iteration by $j = j + 1$.

Evolutionary Computation based Policy Direct Search

Evolutionary Computation (EC) has been proven to be a suitable technique for solving RL problems without relying on estimated policy gradients, because of its ability of automatically finding proper representations, handling continuous spaces, etc [236, 234, 232, 206]. Here, we introduce a few typical EC based RL algorithms.

NEAT based Policy Direct Search. NEAT was frequently utilized to evolve a direct action selector, i.e., policy networks. In this setting, NEAT starts with a population of individuals (i.e., policy networks) that are initiated with the simplest form where there are only fully connected input and output nodes. Afterwards, NEAT attempts to use specially designed crossover and mutation operators to reproduce new networks for next generation. Each evolved NN represents a policy network, which takes raw state input and produce the action to be selected for the encountered state. NEAT was previously shown its effectiveness on solving some simple control tasks, for example, pole balancing problem.

Evolutionary Strategies based Policy Direct Search. Evolution Strategies (ES), as a class of black box optimization algorithms, have already shown their great power on heuristically searching optimal solutions for continuous optimization problems [179, 13, 151]. Based on the differences of how the population is represented and how the evolutionary operators perform, ES have many different but related variations, such as 1+1 ES [179], μ/λ-ES [179], CMA-ES [85], and so forth. For solving RL problems, CMA-ES has already been widely used [226, 14]. A more recent work is reported by Salimans et.al [186] of Open

AI labs to use a simple variant of the basic ES to solve very challenging RL problems with highly competitive results compared to most of cutting-edge DRL algorithms. We use OpenAI-ES to represent the algorithm for convenience of expression.

- *CMA-ES based PDS*: CMA-ES have already been widely used as a policy search algorithm for solving RL problems, such as [103, 228, 82, 83, 219], etc. The key principle behind these work is to directly apply CMA-ES to optimizing policy parameters to improve its performance on different RL tasks. The evolution process of CMA-ES can be found in Section 2.1.3. However, as it is a second-order method, CMA-ES becomes more difficult when being applied to large-scale RL problems in terms of high computational cost [14, 85].

- *OpenAI-ES*: OpenAI-ES belongs to the family of Natural Evolution Strategies (NES) [237]. In general, given an objective function $\mathcal{J}$ with respect to $\vec{\theta}$. The population of NES is represented as a distribution over $\vec{\theta}$, i.e., $p_{\vec{\psi}}(\vec{\theta})$ parametrized by $\vec{\psi}$. NES proceeds to maximize $\mathbf{E}_{\vec{\theta}\sim p_{\vec{\psi}}} F(\vec{\theta})$ over the population with gradient descent. Moreover, NES conducts gradient updates with the estimator below,

$$\nabla_{\vec{\psi}} \mathbf{E}_{\vec{\theta}\sim p_{\vec{\psi}}} F(\vec{\theta}) = \mathbf{E}_{\vec{\theta}\sim p_{\vec{\psi}}}[F(\vec{\theta}) \nabla_{\vec{\psi}} \ln(p_{\vec{\psi}}(\vec{\theta}))] \tag{2.63}$$

  In (2.63), $F(\cdot)$ is the total rewards from the environment, and $\vec{\theta}$ is the policy parameter. The population is initialized with mean $\vec{\psi}$ and fixed covariance $\sigma^2 \vec{I}$. More specifically, in OpenAI-ES, the gradient estimation based on (2.63) is approximated by samples, which is described as,

$$\nabla_{\vec{\theta}} \mathbf{E}_{\vec{\epsilon}\sim\mathcal{N}(\vec{0},\vec{I})} F(\vec{\theta} + \sigma\vec{\epsilon}) = \frac{1}{\sigma} \mathbf{E}_{\vec{\epsilon}\sim\mathcal{N}(\vec{0},\vec{I})}[F(\vec{\theta} + \sigma\vec{\epsilon})\vec{\epsilon}] \tag{2.64}$$

  Accordingly, we can have the updating rule for the policy parameters as,

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} F_i \vec{\epsilon}_i, \tag{2.65}$$

  where n is the number of individuals.

Evolutionary Algorithms based Policy Direct Search. In literature, there have already been attempts of using Evolutionary Algorithms to solve simple RL tasks as reported in [157, 236]. A very recent successful application of GA algorithm for DRL is reported by Such et.al. [210] from Uber AI group. In the following, we use Uber-GA to represent the algorithm for convenience.

- *Uber-GA*: Uber-GA is a variant of the traditional Genetic Algorithm proposed in [7] where a population of N individuals are evolved. In Uber-GA, the individuals are parameter vectors of the policy network. The average total reward over multiple trajectories collected by one individual ($\pi_{\vec{\theta}}$) is used as a fitness score for evaluation. A truncation selection is adopted to select the top-ranked T individuals as parents to produce offspring for next generation. The evolution in Uber-GA repeats the following process: A parent is selected uniformly at random and then is mutated by introducing a Gaussian noise as follows,

$$\vec{\theta}^{g+1} = \vec{\theta}^{g} + \sigma\vec{\epsilon}, \tag{2.66}$$

  where $\vec{\epsilon} \sim \mathcal{N}(\vec{0}, \vec{I})$, σ is the step-size which is determined empirically for different experiment. In addition, *elitism* is also adopted to maintain n individuals where $n << N$. Note that, Uber-GA does not include the crossover operator and hence functions more like an ES algorithm. After new population is generated, it then will be evaluated. The process repeats for g generations or until other stopping criteria are met.

2.2.7 Discussion on Policy Direct Search

In the above subsection, we have elaborated many representative model-free PDS algorithms. In particular, we have described both gradient-based and gradient-free PDS techniques. In this subsection, we will discuss the advantages and challenges of these methods in comparison to VIS algorithms.

PDS enjoys many advantages in comparison to VIS algorithms, more importantly, these advantages help PDS address several challenges stated in Section 2.2.3. Firstly, PGS can straightforwardly handle problems with a continuous action space (see (2.26)). Secondly, PDS can directly search for both de-

terministic and stochastic policies. Thirdly, PGS can achieve higher sample efficiency than VIS algorithms [12]. Fourthly, some gradient-free methods can be effectively exploited to balance the exploration and exploitation during the learning process. This is because PDS they directly perturbs in the policy parameter space whereas VIS can only perturb in action space. The latter is empirically found less efficient than the former [45].

However, we also have identified several challenges after reviewing state-of-the-art PDS algorithms:

- *Linear Function Approximation vs. Nonlinear Function Approximation*

  Linear function approximation is straightforward but lacks accuracy. In view of this understanding, nonlinear function approximation may be more appropriate for addressing complicated problems. On the other hand, nonlinear approximation sometimes over-complexify the problem, and produce overcomplicated model that is overfitted by a specific tasks. This may cause the agent performing badly in environments with high uncertainty. It is a challenge to find proper approximation for solutions on specific problems.

  Particularly, in this thesis, we investigate how to improve step learning based PGS with linear policy representation in Chapter 4. Next, we study the nonlinear policy representations (i.e., NN based representation) in Chapter 5. In addition, to better examine the reliability of value function learning in Chapter 6 and the usefulness of feature learning in Chapter 7, we must ensure the minimum impact of policy learning, therefore we also adopt linear policy representation in these chapters.

- *Accurate Gradient Estimation vs. Inaccurate Gradient Estimation*

  A critical factor for the success of PGS is to obtain accurate policy gradient estimations. Almost all PGS algorithms discussed so far are designed to obtain more accurate estimations of the policy gradients than before so as to eventually enable effective reinforcement learning. In particular, step learning based PGS algorithms estimate the gradients based on one step sample, then follow the stochastic gradient descent to update the policy

parameters. In this way, the estimation can vary significantly from step to step, which results in unstable learning. This is because step learning based PGS algorithms disregard all historical gradients with an assumption that the past gradients are no longer useful, which is not always true [61]. In view of this, it raises another challenge of how to effectively use these historical gradients.

More specifically, we aim to answer this question by obtaining more accurate policy gradient estimations for several representative step learning based PGS algorithms with historical gradients in Chapter 4.

- *Sample Efficiency vs. Learning Effectiveness*

  The key of all model free methods is to learn from samples. For example, PGS uses Monte Carlo methods to estimate the expected return. Gradient-free methods require policies to be evaluated by actual simulations in the environment. However, it is very difficult to collect millions of samples in a real-world application. For instance, it is almost impossible to conduct thousands of real Lunar Lander experiments. This raises a new question how can we use less samples to achieve reasonable effectiveness (i.e., improve sample efficiency). It is clear that using less samples may degrade the learning effectiveness. Consequently, it is truly challenging to find a trade-off between these two factors (sample efficiency and learning effectiveness).

  To particularly answer these questions, we investigate the improvements of sample efficiency on evolutionary policy optimization with the help of a surrogate model in Chapter 5. Also, we study to enhance sample efficiency of NEAT by integrating NEAT based automated feature learning with cutting-edge PGS algorithms in Chapter 7.

- *Exploration vs. Exploitation*

  The exploration-exploitation trade-off is a long standing dilemma for all RL methods. The dilemma is about, at a time point, whether the agent needs to behave greedily with the best action found so far or to explore

the environment for better actions. It is widely accepted that gradient-based methods are good at exploitation, as they always follow the direction of improving the best policy found so far [45]. In contrast, gradient-free methods, especially EC based PDS, possess clear strengthen on exploration. In fact EC methods are population-based which greatly increases the policy diversity to encourage more exploration. However, gradient-free methods are more sample inefficient than gradient-based methods. Thus, it is a challenge to balance the exploration-exploitation trade-off while developing new policy direct search algorithms.

In this thesis, we specifically to investigate balancing the exploration and exploitation in PGS by adopting an seamless integration of EC based global search and PGS based local search in Chapter 5.

2.2.8 Feature Learning in Reinforcement Learning

As introduced in Section 2.1.4, Feature learning (FL) plays an important role in RL. FL generally aims to achieve two goals by incorporating feature learning with RL algorithms [71, 236]: first, FL can extract useful features or reduce the dimensionality of raw state inputs; second, with the learned (extracted) features, FL can realize more effective RL in comparison to learning from raw state. There are four common approaches developed to achieve feature learning so as to improve RL performance as shown in Section 2.1.4, which will be discussed in details below.

Basis Function Adaptation

For methods that search linear policy, the state features are often considered as basis functions. Many researchers have considered to implement self-adaptive basis functions to assist RL. These methods usually combine supervised learning approach into RL for training the basis functions, which is normally achieved through two steps. First, a parametric function is chosen carefully as a feature base, including Radial Basis Function Network [165], Fourier Basis Function [123] and other types of bases [212]. Next, the feature function

parameters are learned by optimizing carefully-designed score functions such as the Bellman Error [150, 165], and Mean Squared Error [57].

Unsupervised Feature Learning

UL can help grouping similar features, eliminating useless features, or transforming low-level features to high-level features so as to improve the general learning performance for its subsequent tasks particularly RL [26, 71, 132, 239]. Two typical examples of using unsupervised feature learning in conjunction with reinforcement learning are given below:

HORDE. HORDE was proposed by Sutton et. al [216], where general value function, policy, termination function, reward function, and terminal reward function are all represented as parametric functions (i.e., so-called knowledge). Essentially, HORDE is a scalable real-time architecture consisting of many independent sub-agents and a base-agent. Each sub-agent is responsible for learning one small piece of knowledge about the base-agent's interaction with the environment. By using HORDE, one can learn to predict the sensor inputs, build general value functions, and identify policies to maximize those sensor values. It follows essentially off-policy learning where it learns in real-time while following some behavior policy, and updates value functions with gradient-based TD learning methods.

Unsupervised Auxiliary Learning. UNsupervised REinforcement and Auxiliary Learning (UNREAL) is an algorithm proposed by Jaderberg et al. [106] to improve learning efficiency by maximizing pseudo-reward functions associated with the usual cumulative reward. The pseudo-reward functions and the usual cumulative reward share a common representation. Long-Short-Term-Memory based Recurrent Neural Network (LSTM-RNN) is used to compose of the learning agent in UNREAL. It has technical features such as pixel control, reward prediction, and value function replay. The RL agent is trained on-policy with A3C [154]. Experiences of state transitions are stored in a replay buffer, for auxiliary tasks. To maximize changes in pixel intensity of different regions of the

input images, the auxiliary policies use the base Convolution Neural Network (CNN) and LSTM-RNN, together with a deconvolutional network. To tackle the issue of reward sparsity, it also equips with a reward prediction module which is to predict short-term rewards in next frame by observing the last three frames. Value function replay is further adopted to train the value function to stabilize the value function learning. UNREAL outperforms A3C on Atari game playing tasks and a 3D Labyrinth game.

Deep Learning and NeuroEvolution

In fact, most of RL algorithms that adopt Deep Neural Networks as representations for either value function or policy have already embedded the feature learning through end-to-end training of NNs [12]. As discussed in Section 2.1.4, the multi-layered structures of NNs can be treated as two consecutive parts. The first several layers can be viewed as feature networks and the last layer is treated as the model for the value function or the policy. The training of both feature learning and reinforcement learning are coupled together as a whole, which can be done by either gradient-based back-propagation or gradient-free evolutionary operations.

NeuroEvolution based RL algorithms are similar to DRL algorithms [204], the feature learning is already mingled with policy search process as a couple. The only difference is about the topology representations and training methods [204, 152]. As introduced in Section 2.1.3, some NeuroEvolution algorithms are capable of adapting the topology as well as weights simultaneously. In addition, NeuroEvolution algorithms use evolutionary algorithms to adapt network parameters instead of gradient-descent techniques.

2.2.9 Discussion on Feature Learning in Reinforcement Learning

In the above subsections, we have introduced the research progress made in literature on FL for RL. To better understand the effect of FL in RL and motivate our research in the thesis, we next discuss the importance as well as the

challenges of FL in RL.

There are several obvious advantages of introducing FL into the RL algorithms. First, some FL algorithms are capable of largely reducing the dimensions of the raw state input which help improve the time efficiency of RL algorithms. Second, feature learning straightforwardly automates the feature extraction process which releases human experts from the labor-intensive feature engineering process. Third, self-adapted feature extractor obtained by FL algorithms can improve the generalization of RL algorithms on different problems. Fourth, the learned features, in comparison to the raw state inputs, can be more informative and less noisy which enables more effective learning.

Despite of these superiorities, the integration of feature learning and reinforcement learning may still encounter some challenges as below.

- *Complete Automation vs. Partial Automation*

  One of the objectives for feature learning is actually to automate the feature extraction process hereby to avoid human interferences during the learning process. However, current existing feature learning approaches, when being applied to RL domain, still require domain knowledges. In other words, these feature learning methods have achieved only partial automation. For example, DRL algorithms require to pre-determine the network topology which is usually a very challenging task. For training the self-adaptive basis functions, one must determine the error function as well as target values in advance before applying the supervised learning techniques. Thus, complete automation on feature learning remains a big challenge for us at current stage.

- *Reusable Features vs. Problem Specific Features*

  Another key objective of feature learning is to extract useful features. In literature, the useful features normally refers to two aspects. First, the feature must promote the learning on current task. Second, it can be reused for the case when environments or the learning tasks change. Thus, the learned features that are strongly limited to the specific tasks cannot be treated as useful features. However, in literature, there are few attempts being made to improve the re-usability of features.

- *Coupling vs. Decoupling*

  For DRL or NeuroEvolution based RL algorithms, feature learning and reinforcement learning are coupled together. The upside of such a coupling approach is that training is viewed as a whole where no extra requirements on designing different training strategy for FL. However, the downside of coupling is also prominent. First, the couple of the two learning processes blend the feature output with final decisions (actions), resulting in learned features with poor generality. Second, it hinders effective knowledge (feature) sharing across agents for algorithms that use multiple learning agents. Therefore, this raises a challenge of how to effectively decouple the two learning processes so as to obtain useful features as well as find good policies.

In particular, we study how to achieve more effective PDS by adopting completely automated feature learning decoupled from policy learning to extract reusable state features in Chapter 7.

2.3 Related Work

In this section, we will introduce the research works that are closely related to this thesis regarding their pros and cons to better motivate the research of this thesis. The section is organized in correspondence to research objectives stated in Section 1.3.

2.3.1 Effective Policy Direct Search through Primal Dual Approximation

Recently, the challenge of using linear approximation or non-linear approximation stated in Section 2.2.7 has gained more attention [146, 178]. For instance, Rajeswaran et al. [178] and Mania et al. [146] have found that linear policy with simple search techniques can achieve competitive performance as opposed to state-of-the-art DRL algorithms with reasonable computational resources. They argued that the reasons for such findings are as follows. First, current DRL

systems have been found prone to over-fitting problems, especially when large data collection is unavailable [178]. Second, results of many current RL methods are difficult to be reproduced due to their high sensitivity to hyper-parameters, random seeds, or even the diversities of implementations [146]. Third, the engagement of DNN (e.g., Convolutional Neural Networks (CNN)) brings extra workload for engineers/researchers to carefully choose/design the network structure for hard problems [178]. In addition, the randomness and sparseness of reward distributions can significantly affect the effectiveness of RL methods, which is often addressed by carefully engineering the reward functions [146]. Fourth, extremely high demands of computational resources are common in the DL field, especially DRL [178]. Besides the heavy computations of back-propagation, effective learning also requires to process massive trial samples from simulations or real-world applications.

However, there remain some limitations with the two works of [178] and [146]. For the NAC algorithm proposed by Rajeswaran et al's [178], it actually learns from high-level state features extracted by an NN rather than from the raw state input. In addition, it focuses only on episode learning strategy but overlooks another training strategy, i.e., step learning. For ARS proposed by Mania et. al [146], it is also episode learning. Moreover it can be very sample inefficient, because as a random search technique, it normally requires much more samples for performance evaluation than those for PGS algorithms such as TRPO [189], PPO [191], ACKTR [240].

In view of this, we aim to study simple step-wise PGS algorithm also with linear policy. We choose is RAC and its variants proposed by [31] as base algorithms. This is because 1) RAC features a simple structure that are easy to be customized and extended, and 2) its effectiveness has already been witnessed in the literature [31].

Nonetheless, referring to step learning strategy, we have confronted with another challenge of how to obtain accurate estimations of policy gradients with lower variances as mentioned in Section 2.2.7. One possibility is to use the historical gradients to stabilize the gradient-based learning as suggested in [62, 241]. However, to our best knowledge, there are no existing studies about how to utilize the historical gradients on step-based PGS such as RAC.

Motivated by this understanding, we intend to develop new step learning based PGS algorithms by properly utilizing historical gradients via the Primal-Dual Approximation (PDA) technique. With the help of PDA, the algorithm is expected to achieve more accurate policy gradients resulting more effective policy updates. The development and evaluations of the new PGS algorithms are presented in Chapter 4.

2.3.2 Proximal Evolutionary Strategies for Sample Efficient Policy Direct Search

The applicability of EAs for RL to solve continuous control problems has been an enduring research topic for many decades. Among all different EAs for RL, NeuroEvolution (NE) plays a dominating role. This is because 1) Neural Networks (NN) provides flexible representations that are suitable proven suitable for solving RL problems, 2) NN provides a good platform where most optimization methods are directly applicable. NE approaches can be simply classified as two branches by whether or not the topology is changed. Typical representatives for the NE approaches where topologies and weights are co-evolved are NEAT and its variation HyperNEAT. They have been found to perform effectively on solving classic control problems, such as pole-balancing and mountain car [205]. Another type of NE is only to adapt weights leaving the topology unchanged, typical examples are ES based algorithms. The successful applications of these methods in RL are fruitful, such as [228, 82, 83, 219, 94, 227].

However, all the NE approaches discussed above are proven difficult to solve complex problems [186, 210]. In an attempt to evolve DNNs, a few Deep NE approaches have been proposed recently, such as OpenAI-ES [186] and Uber-GA [210]. They have shown outstanding performance on very complicated problems, such as Atari games or locomotion problems. Thanks to the efforts made in [186, 210], EAs have been widely accepted now as an alternative to the gradient-based DRL approach.

There are three issues faced by EAs when they are used for DRL, including (I1) *Low Time Efficiency*, (I2) *High Sample Complexity* and (I3) *Low Learning Effectiveness*, which are discussed below:

(I1) EAs can be very time inefficient while handling a large search space. For example, both OpenAI-ES [186] and Uber-GA [210] must maintain an immense population size to reasonably cover the search space better in order to solve complex RL problems. The time complexity in the situation can be significant because they require a large number of evaluations that increases linearly with respect to the population size. Although the issue can be mitigated by large-scale parallelization, it requires large computational facility.

(I2) EAs are more sample complex compared to traditional PGS. For example, as reported in [186], OpenAI-ES have used 3x and 10x as many samples to perform well in some environments compared to A3C [154] or TRPO [189].

(I3) EAs cannot challenge cutting-edge PGS algorithms in terms of learning performance on many benchmark problems. For example, OpenAI-ES has only achieved moderate performance on 23 games but worse on 28 games compared to A3C [186].

Aiming to address the above issues, we are urged to investigate new EA based DRL algorithm which can achieve state-of-the-art performance regarding high time efficiency, low sample complexity, and high learning effectiveness. Particularly, in Chapter 5, we aim to develop a sample efficient evolutionary deep policy optimization algorithm based on CMA-ES. The algorithm is expected to resolve the issues mentioned above while being compared to cutting-edge PGS algorithms (e.g., TRPO [189], PPO [191], ACKTR [240]) and advanced EAs (e.g., OpenAI-ES [186], Uber-GA [210]).

2.3.3 Reliable and Flexible Value Function Learning for Policy Direct Search

One of the critical challenges for VIS is how to stabilize the learning on value functions as stated in Section 2.2.3. It is also a challenge particularly for AC algorithms where value function must be learned to guide policy search. The overall

effectiveness of AC algorithms largely depends on critic learning. Traditionally, Aimed at improving reliability of critic learning, various research works have been proposed [122, 212, 213, 66, 49, 37, 36]. For example, Sutton and his colleagues [214] proposed Gradient Temporal Difference (GTD) to use off-policy gradient-based training techniques to stabilize the temporal difference learning, but the algorithm converges very slowly. A later version GTD2 was proposed in [213] by replacing the objective function from the common used Mean Squared Bellman Error (MSBE) [122, 212, 213, 66] to the mean-square projected Bellman error (MSPBE) to further enhance the learning reliability with a faster convergent pace compared to GTD. However, both GTD and GTD2 empirically exhibit the behavior of divergence on critic learning, calling into question its practical utility. Several second-order methods, such as Least Square Temporal Difference (LSTD) [37, 36], can guarantee the reliability but with high computational complexity $O(n^2)$, where n is the number of state features. The problems of the existing works motivate us to consider another possible solution to improve the reliability in critic learning in a fast manner without incurring extra learning complexity.

Another common challenge for PDS is to accurately estimate policy gradient estimations based on the value function learning process under the Policy Gradient Theorem [215]. Most existing PGS algorithms focus on constructing accurate policy gradients by replacing the other component, i.e., $Q^{\pi}(\vec{s}, a)$ as seen in (2.36). However, these algorithms overlooked the importance of another key component to form the policy gradient, i.e., $\Phi(\vec{s}, a) = \nabla_{\vec{\theta}} \ln \pi^{\vec{\theta}}(\vec{s}, a)$. To our best knowledge so far, existing works that study different forms of the compatible features $\Phi(\vec{s}, a)$ are very limited. This urges us to dive into this direction to investigate the effects of different form of compatible features to the learning effectiveness of PGS.

In line with this understanding, we intend to address the two challenges mentioned above in Chapter 6 by developing new PGS algorithms via different techniques to improve reliability and flexibility of value function learning. Specifically, the algorithms can stabilize value function learning via the SM or can generalize compatible features via q-logarithm to provide a new flexible family of compatible functions. The developed algorithms are expected to even-

tually achieve effective policy learning on benchmark RL problems.

2.3.4 Enhancing Policy Direct Search via Automated Evolutionary Feature Learning

The importance of Feature Learning (FL) has gained more and more notices in RL research domain, which has been considered as an effective auxiliary approach to promote reinforcement learning [132, 71, 138]. Traditionally, features are manually designed, it has been considered to be time-consuming and error-prone, because the design process normally requires special domain knowledge from human experts. In practice, the human experts are not often available, and such a manual process can also bring biases into the extracted features possibly producing unexpected results. Some early methods have been developed, such as [123, 212], by using fixed feature functions to transform low-level states to high-level features. These methods are problem-specific, as some features may only be suitable for specific tasks. In view of this, a group of methods have been proposed to implement self-adaptive basis functions (feature functions) [150, 165, 57]. These methods have adopted SL techniques to train the basis functions, but the SL techniques require human experts to carefully choose error functions and determine labels.

To address the issue, many methods adopted NNs or DNNs as representations for policy or value function where feature learning is considered coupled with the reinforcement learning. Such approaches face two main challenges, i.e., 1) completed automation vs. partial automation and 2) coupling vs. decoupling, as stated in Section 2.2.7.

To address these challenges, in comparison to DRL, NeuroEvolution has more potential for the complete automation on feature learning. First, NEAT can automatically evolve the topology without specifying it in advance. Second, NEAT gradually complexify the topology which can result in a reasonably suitable (i.e., less complicated than some pre-defined DNNs) representations [152]. Because of this, we consider that NEAT can be a good candidate for feature learning during RL. However, we are not the first to explore this idea. For example, FS-NEAT [233] and its variations [139] have been proposed to perform

feature selection but not feature extraction, where no high-level features are explicitly evolved. NEAT+Q [232] considers feature extraction, but it is a VIS algorithm which does not explicitly learn a policy. Besides, the features learned through NEAT+Q are embedded in value functions. They cannot be directly reused for other learning algorithms.

Another challenge of "decoupling vs. coupling" introduced in Section 2.2.7 becomes severe when NEAT is applied to solving large-scale RL problems. For NEAT, it must evolve a highly sophisticated NN, since the network by itself has to accomplish two tasks, i.e., feature extraction and policy search. Also, it must evolve the network from the simplest initial structure with no hidden neurons and connections. This makes the tasks even more difficult and sample inefficient. In addition, extensive investigations of HyperNEAT [205] suggest that HyperNEAT may perform much worse than NEAT on many large-scale problems including Atari games, especially in the presence of high-level features [90]. Besides, NEAT was found vulnerable to the fracture issue, i.e. the mapping from states to optimal actions is highly discontinuous [118]. The same issue is proven to be even more challenging for HyperNEAT in [224].

To address these important issues, we make a first a first attempt to split feature learning from policy learning, resulting in a new algorithm called NEAT+RAC in Chapter 7. Despite of clear performance advantages of NEAT+RAC on classic control problems, NEAT+RAC still has limited effectiveness for large-scale RL. This is because, NEAT+RAC relies on the traditional RAC which often fails to learn reliably with non-linear and more powerful policy networks. Moreover, feature learning and policy learning are heavily mingled in a single process, preventing easy sharing of learned knowledge (e.g. policy networks) across multiple agents. Further motivated by this, we study how to clearly separate the feature learning and policy learning, meanwhile to achieve policy improvements by using cutting-edge PGS algorithms suitable for training deeply-structured policy networks in the same Chapter(i.e., Chapter 7).

2.4 Chapter Summary

The Chapter provided a comprehensive literature review, including fundamental concepts of ML with focus on RL, the general principles and formulation of the RL framework, and several popular techniques widely used in the RL domain. Besides, in order to motivate the research of the thesis, the chapter analyzed and discussed advantages as well as challenges of related works with the focus on PDS approach.

Although PDS has become more and more popular and influential in the RL domain, there are still rooms to develop new policy direct search algorithms further. In comparison to existing work, the new algorithms ought to 1) be more effective, 2) be more sample efficient, 3) achieve better balance of the exploration-exploitation trade-off, 4) support seamlessly integration with a decoupled feature learning to extract useful features, and 5) be capable to stabilizing learning on value functions. Our understanding on the challenges of RL methods discussed in prior sections, form the fundamental motivations of the thesis.

Despite the general challenges, more specific limitations of existing works discussed in Section 2.3 are summarized below to motivate the research of the thesis further.

- Episode learning based PGS with linear policy has already been investigated in the literature. However, the existing works overlooked another important learning strategy, i.e., step-learning. Nevertheless, the step learning strategy conducts the policy updates based on a single step sample, which often yields high variances in the updated policy. Therefore, it is necessary to investigate techniques further to improve policy gradient estimation accuracy and stability to further improve the learning effectiveness for step learning based PGS.

- Some existing work has shown that EAs can be useful algorithms for solving RL problems when the solution dimensions are reasonably small. However, all existing EAs are still facing three critical issues, i.e., low time efficiency, high sample complexity and low learning effectiveness,

when they are applied to training deep structured policy networks. This makes EAs lagging behind cutting-edge PGS algorithms. Given this situation, it is worthy to investigate new EA based DRL algorithms to address the above three issues so that EAs can truly become state-of-the-art algorithms.

- Value function learning is vital for AC algorithms, as the learned value function is used to estimate the policy gradients. However, value functions learned from samples by Monte Carlo methods often exhibit high variances, resulting in the unstable and fluctuated value function learning, and hence deteriorated policy learning. Some existing works attempted to address the issue by introducing extra complexity into the objective function, which may lead to performance degradation as well. Hence, it is necessary to discover new methods to stabilize the value function learning without introducing any extra complexity. Moreover, most existing works overlooked the importance of compatible features proposed in [215] for accurately estimating policy gradients. To our best knowledge, there is no existing work studying this important problem that must be investigated further.

- Feature learning is another key factor for the success of reinforcement learning. In traditional RL, features are normally hand-crafted by human experts, which is considered time-consuming and error-prone. Many existing algorithms can naturally address these issues by naturally coupling the feature learning into policy learning. However, such a coupled learning process has limitations in terms of the effective knowledge sharing and easy splitting features from final decisions. Motivated by this, some other existing work proposed to decouple the feature learning and policy learning and automate the feature learning process. Nevertheless, these methods either have to use a fixed policy or have to adopt domain knowledge from human experts. Therefore, it is useful to study a fully automated feature learning process that is capable of extracting useful features and finding good policies.

Multiple chapters in the thesis is constructed will address the above issues.

Chapter 4 will develop three new AC algorithms for training linear policies based on a primal-dual approximation technique. Chapter 5 will develop a CMA-ES based deep policy optimization algorithm. Chapter 6 will develop two PGS algorithms. One algorithm is a new RAC algorithm with stabilized critic learning by self-organized SandPile Model. The other algorithm is a new RAC algorithm where the compatible function is generalized based on a generalized logarithm function. Chapter 7 will develop two new PGS algorithms. The first algorithm integrates NEAT based feature learning with the RAC algorithm for PGS called NEAT+RAC. The second algorithm presents a major improvement of NEAT+RAC by explicitly decoupling the NEAT based feature learning from PGS to enable effective knowledge sharing and by integrating with various state-of-the-art PGS algorithms.

Chapter 3

Experimental Methodology

The chapter presents the core experimental methodology of this thesis. It starts with the benchmark problems used throughout the thesis. At the end of the chapter, it summarizes statistical treatment used in this thesis including general experiment setup and the statistical methods for experimental results analysis.

3.1 Benchmark Problems

To evaluate RL algorithms, there are many applicable benchmark problems [212]. In this thesis, we have primarily focused on control problems, because of two reasons: (1) It is widely recognized challenging to RL agents in the literature [212], which can genuinely reflect the true capability of the agent. (2) It has good coverage on both continuous and discrete problems. In this thesis, we have adopted 16 commonly used benchmark continuous control problems. Four problems, i.e., Mountain Car Continuous, LunarLander, BipedalWalker, and BipedalWalkerHardcore, are provided by GYM environment [39]. The other six problems are simulated by using Bullet Physics Engine [220], including HalfCheetah, Hopper, Inverted Double Pendulum, Inverted Pendulum, Inverted Pendulum Swingup, and Walker2D. We briefly describe each problem used in experiments in the rest of the subsection, more detailed descriptions about these problems can be found in [65, 221, 95].

- *Mountain Car Continuous* is a continuous version of the classic Mountain

Car Problem. In the problem, a car is positioned between two "mountains" in a one-dimensional axis. The goal is to drive the car up to the mountain on the right side. To make the problem difficult, the car's engine is not strong enough to reach the goal in a single pass. The key to success is the use of the momentum produced by driving the car back and forth. Moreover, the instant reward is the distance from the current car position to the goal region, and a +10 is directly given once the car reaches the goal region. It is designed to have one continuous action in [-1.0, 1.0].

- *Lunar Lander* is to control an agent accepting 8-dimensional continuous sensor input to produce a two-dimensional continuous action ranging from -1.0 to 1.0. It aims to smoothly and accurately guide the lander robot to land on a target pad which is always set at the origin (0.0, 0.0). While moving from the top of the screen to the target pad with zero speed, the agent will be awarded a reward ranging from 100 to 140. However, it loses rewards due to it is moving away from the pad. As long as the lander crashes or comes to rest, it receives an additional -100 or 100, and the episode completes.

- *Bipedal Walker* is to drive a robot move along flat terrain. It is constituted of 24-dimensional continuous state space and 4-dimensional continuous action space. The agent is rewarded $+1$ point by moving forward, and a total of $+300$ points are given at the far end. The fallen of the robot will cause a -100 penalty; also motor torque costs a small number of points.

- *Inverted Pendulum*(i.e., *Cart Pole*) is the classical pole balancing problem, where a pole is attached by a joint to a cart moving horizontally. It aims to find a plot that balances the pole to the upright angle as long as possible. It is designed to have four state inputs and one continuous action in $[-1.0, 1.0]$. As long as the pole maintains upright, it receives a $+1$ reward.

- *Inverted Pendulum Swingup* is an analogy to Inverted Pendulum, but it requires additional swing the pole up to maintain its balance to the upright angle.

- *Inverted Double Pendulum* is a hardcore version of Inverted Pendulum, as it contains two joints connecting two poles to a fixed point. The controller actuates the joint to swing the end of the lower pole to a given height from the initial situation where both poles are hanging downwards.

- *Bipedal Walker Hardcore* is very similar to the Bipedal Walker. The only difference is that the robot travels on a tough terrain where ladders, stumps, and pitfalls are placed. These obstacles increase the time limit hence the difficulty of the problem.

- *Half Cheetah* is a planar locomotion task where the agent is required to control a cheetah-like robot to move along a flat plane as quickly as possible. It contains 26 state inputs and six action outputs; each action dimension is located in the range [-1.0, 1.0]. The rewards is given by the function $r = v_x + 0.1 * ||a||^2$, where v_x is the velocity along x-axis.

- *Hopper* aims to move one-legged robot travel forward as fast as possible on a 2D plane in a 3D environment. It consists of 15-dimensional state space and three-dimensional action space within the range [-1.0, 1.0]. The reward is given by the velocity position of x and y with an alive bonus 1 point. Note, there is a $0.001 * ||a||^2$ cost on control.

- *Walker2D* is like a two-legged Hopper with a larger state space and action space. It uses 24-dimensional inputs to generate six-dimensional continuous actions ranging from -1.0 to 1.0 for each dimension. The reward consists of x-velocity of the torso with penalties of quadratic control cost, distance between the torso and the target height, and the orientation.

- *Reacher* describes a problem where a robot with two links aiming at reaching a target position. It contains 11-dimensional state space and 2-dimensional action space. It receives rewards by using the negative distance from the end of the arm to the target. Each movement control costs $||a||^2$.

- *Puddle World* is a two-dimensional continuous environment (i.e., $[0, 1]^2$) in which round puddles are placed at $(0.2, 0.25)$ to $(0.55, 0.25)$ and $(0.45, 0.2)$

to $(0.45, 0.6)$ with a radius 0.1. A mobile agent initiates at a random position in the environment and learns to reach the goal region (i.e., $x+y \geq 1.9$) without entering the puddles. When reaching the goal region, the agent will receive an instant reward of $+40$. Otherwise, it will be penalized with -1 for its movement. In particular, when entering the puddle area, it receives a penalty computed by multiplying -400 with the agent's shortest distance to the border of the puddle [43]. In this problem, a learning episode is defined as the learning period from the agent's initial state to the moment when it arrives at the goal region.

- *Heating-Coil Problem* is one member of the challenging Heating, Ventilation, and Air Conditioning (HVAC) problem set [9, 84]. The problem has several dynamics (i.e., state dimensions), and theses dynamics are categorized as IPV and EPV respectively. IPV are variables directly influenced by the controller, whereas EPV is controller-independent variables solely reflecting the environment changes (i.e., environment disturbance). In order to simulate the environment disturbance, the EPVs (i.e., T_{ai}, T_{wi}, f_a) are changed by random walk every κ (benchmarked as 30 in [84]) time-steps within their own intervals. The goal of the problem is to make the output air temperature T_{ao} close enough to the target temperature of T_d. This can be achieved by adjusting the opening valve c to control the system [84].

In addition to control problems, we have also adopted six Atari gameplay tasks to evaluate the effectiveness of our algorithms in Chapter 6. This is because the proposed algorithms must be evaluated on large-scale reinforcement learning to fulfill our research goals. Hence Atari game playing tasks are considered as suitable benchmarks for our experiments. However, due to a limited computational resource, we consider mainly six widely-used/representative games, including Asteroids, Breakout, Freeway, Seaquest, SpaceInvaders, and TimePilot, implemented in Atari Learning Environment (ALE) [23]. In the RL literature, ALE is a set of well-known and highly challenging benchmark problems [23, 59, 91, 189, 154, 155]. ALE provides two types of Atari game playing tasks with the key difference on state representations: one is RAM-based games where states are represented as 128-bit integers stored in memory, the

other is IMAGE-based games where states are represented as video frames captured directly from the games. In Chapter 6, we choose RAM-based games, because they are more suitable for NeuroEvolution of Augmenting Topology (NEAT) [206] as 128 dimensions of raw state inputs are considered sufficiently large for our empirical study. Because when the number of samples allowed for training is limited at a certain number, NEAT has shown bad performances on the 128-dimensional ram-based Atari game playing tasks as reported in [90].

3.2 Statistical Treatment

In this section, we depict the statistical treatment used in this thesis to determine the performance significance among experiments results. We firstly describe our general experiment setup to explain how the experiments are performed. Then we explain different statistical methods used in this thesis, including Student's t-test, and Analysis of Variance (ANOVA) [4].

3.2.1 General Experiment Setup

To determine any performance significant differences in experiment results, we conduct a standard experiment setup for all experiments in this thesis. Firstly, we perform 30 independent runs for all algorithms on each continuous control task. At every 10,000 samples or the end of every 50 training episode, we conduct one independent testing episode with the learned policy (deterministic). The testing episode is performed in a separated testing environment with the same random seed as the one used for training. In doing so, we can also identify the true effectiveness of each algorithm. All these independent tests are carried out by using the best policy learned so far till the testing point (i.e., every 10,000 samples or every training episode). For all experiments, we have performed training on each algorithm for only 5,000,000 samples or only 10,000 training episodes due to the computational resource limitation.

3.2.2 Statistical Methods

In this thesis, we have used four different statistical methods for experiment analysis. To show the significant differences of algorithms visually, we have plotted the confidence intervals for each algorithm on the learning curve figures. To determine the performance significance among a group of algorithms, we have performed ANOVA. Specifically, when comparing two algorithms, we have performed a student's t-test on the performance. Lastly, we have performed correlation analysis particularly in Chapter 5 to examine the correlation between the stability of value function learning and the effectiveness of police learning. In the following subsection, we will brief these statistical methods.

Confidence Interval

One classic method to examine the statistical difference is to compute the confidence interval, which is an estimated range of values that likely includes an unknown population parameter [4]. Such an estimated range can be calculated from a given set of sample data [4].

In this thesis, we consider the confidence interval where the population of interest is not known, i.e., both the population mean and standard deviations are unknown. In addition, in this thesis, we have only a small portion of samples, so we consider t-distribution rather than the normal distribution. Thus, in this case, we use the following formula to compute the confidence interval,

$$\overline{X} \pm t^* \frac{S}{\sqrt{n}}, \tag{3.1}$$

where n is the sample size,t^* is the upper $\frac{(1-C)}{2}$ critical value for the t-distribution with $n-1$ degrees of freedom, i.e., $t(n-1)$.

Student's t-test

Student's t-test is a classical statistical method to test the null hypothesis of whether there is a significant difference between the means of two groups [4]. It can be generally used for checking the means estimated by two dependent samples differ significantly, i.e., the paired t-test.

In this thesis, we have adopted the paired t-test in the experiments when determining the significance between the performances of two algorithms. The formula we used for the paired t-test is,

$$\mathrm{t} = \frac{\left(\sum \mathrm{D}\right)/\mathrm{N}}{\sqrt{\frac{\Sigma \mathrm{D}^2 - \left(\frac{\left(\sum \mathrm{D}\right)^2}{\mathrm{N}}\right)}{(\mathrm{N}-1)(\mathrm{N})}}} \tag{3.2}$$

where D is the difference and N is the number of samples.

As every t-value has a p-value associated with it, so after obtaining the t-value, we can find the p-value in the t-table [4]. In this thesis, we define the alpha level as 0.05 (5%). Next, we can check whether the computed t-value is greater than the given table value at an alpha level of 0.05. If $p < 0.05$, we can reject the null hypothesis that there is no difference between the means.

ANOVA

However, in our thesis, we have cases where more than two algorithms are compared to determine the significance. In this case, the Student's t-test is not applicable. Thus, we consider ANOVA for analysis of such experiments.

ANOVA is to test if there is any significant difference between the means of two or more groups [4]. In this thesis, we consider one-way ANOVA, as there is only one independent variable to be taken into account. In fact, one way ANOVA is a generalisation of the two sample Student's t-test [4]. In particular, ANOVA compares the variability between the groups to the variability within the groups. The formula we use in the thesis is,

$$\begin{aligned} F &= \frac{MST}{MSE} \\ MST &= \frac{\sum_{i=1}^{k} \left(T_i^2/n_i\right) - G^2/n}{k-1} \\ MSE &= \frac{\sum_{i=1}^{k} \sum_{j=1}^{n_i} Y_{ij}^2 - \sum_{i=1}^{k} \left(T_i^2/n_i\right)}{n-k}, \end{aligned} \tag{3.3}$$

where F is the overall test variance ratio, MST is the mean square between groups, MSE is the mean square within groups, Y_{ij} is an observation, T_i is a group total, G is the grand total of all observations, n_i is the number in group i and n is the number of all observations.

Chapter 4

Effective Policy Direct Search through Primal-Dual Approximation

This chapter is developed to answer the research question Q(1) in Section 1.2 and achieve the research objective O(1) in Section 1.3. In particular, we develop a Primal-Dual Approximation (PDA) technique based step learning framework, where the primal policy optimization problem is converted to a simpler dual problem through averaging historical gradients accompanied with a strongly convex regularization term. This enables us to obtain a more precise policy gradient estimation for effective policy direct search. Based on this idea, we develop three new PGS algorithms on the basis of three existing conventional stepwise learning based PGS algorithms proposed in [31] (i.e., Regular Actor-Critic (RAC), Natural Actor-Critic with Fisher Matrix(NACF), Natural Actor-Critic with Advantage Parameters(NACA)), with the aim to achieve high-performing learning of linear parametric policies. With the development of the three algorithms, we have made two contributions in below:

1. With our formulation of PDA for PGS and the newly derived learning rules, we show that each of the three original PGS algorithms (i.e., RAC, NACF, NACA) can be treated, respectively, as special cases of our proposed PDA based PGS algorithms (i.e., Dual-RAC, Dual-NACF, Dual-NACA).

2. We theoretically prove that our PDA based PGS algorithms can eventually

converge under suitable conditions.

The evaluations are conducted in-between the three newly proposed dual algorithms (i.e., Dual-RAC, Dual-NACF, Dual-NACA) against six competitors. The six competitors can be split into two streams. One purely uses a step learning strategy including RAC, NACF, NACA, and Natural Actor-Critic with Advantage Parameters and Fisher Matrix (NACAF) [31], The other is based on episodic learning including Augmented Random Search (ARS) [146] and Proximal Policy Optimization with linear policies [191]. The testing benchmark problems include Mountain Car, Inverted Pendulum, Inverted Double Pendulum, Inverted Pendulum Swingup, Lunar Lander and Bipedal Walker. The obtained results have shown that

- Step-wise PGS [1] can be equivalently effective but more sample efficient in comparison to episodic learning based algorithms, and
- PDA based PGS algorithms perform significantly better than all other six competing algorithms.

4.1 Introduction

As discussed in Section 2.2.4, PDS searches an explicitly represented policy directly for solving RL problems. There are two common ways to represent the policies, namely linear parametrization and non-linear parameterization [212]. The former way is the simplest form where a policy is represented by a linear combination of a group of parameters, and the latter way is to use a complex non-linear parametric model, such as a Neural Network (NN), to represent the policy [212]. Despite the simplicity of linear parametrization, PGS algorithms with linear policies have three significant advantages over those with non-linear policies: (1) they are easier to be interpreted and understood [54]; (2) they are more efficient in terms of computational cost [164]; (3) they have strong theoretical guarantees of convergence [212, 215, 164]. However, traditional PGS with

[1] For simplicity, we use "Step-wise PGS" to represent the term of "PGS on linear parametric policies with step learning strategy" in the remainder of the chapter.

linear policies are believed difficult to solve difficult continuous RL problems, and hence NN based complex representations are often considered as the first choice in recent PDS algorithms such as [52, 191, 189, 135].

Till very recently, two original works [178, 146] have recreated the glory of PGS with linear parametric policies with empirical evidence of effectively solving the complicated RL continuous control problems. However, the two works both focus on the context of episodic learning and neglect a big family of algorithms that learn linear policies in a step-wise manner (i.e., step learning strategy).

In view of this understanding, in this chapter, we intend to build new PGS algorithms with linear policy in the context of step learning that is effective to tackle the complicated RL problems. To achieve this, we are required to address a critical technical challenge of step learning strategy. Currently, following the Stochastic Gradient Descent (SGD) technique, all current step learning methods use the current step policy gradient to update the policy, which is a volatile process and the resultant policies are often associated with high variances [78, 52, 170, 176].

Ideally, the issue can be tackled by cumulating historical gradients, but it remains a critical challenge of how to properly use the historical gradients to maintain the effectiveness and the convergence of the policy learning. To the best of our knowledge, the research towards addressing the challenge for step-wise PGS remains unveiled, which is worthwhile for further exploration.

4.1.1 Chapter Goals

The overall goal of the chapter is to develop a new PDA framework for building new linear policy search algorithms with stable step-wise learning, good learning performance as well as theoretical convergence guarantees. In particular, we intend to achieve three research objectives in this chapter:

1. To build a new PGS framework based on the general PDA technique, and under which to derive new dual problems with respect to three commonly-used step-wise PGS algorithms, i.e., RAC, NACA, and NACF.

2. To theoretically analyze the convergence behaviors of the PDA based PGS algorithms under suitable conditions.

3. To empirically evaluate the learning effectiveness of PDA based PGS (i.e., Dual-RAC, Dual-NACA, and Dual-NACF) against the original PGS (i.e., RAC, NACA, NACF) and two cutting-edge PGS algorithms (i.e., ARS and PPO-Linear) on six benchmark continuous control problems.

4.1.2 Chapter Organization

The chapter is structured as follows. Section 4.2 presents a preliminary knowledge of the general PDA. Next, Section 4.3 gradually builds the new PGS framework using the PDA technique, and under the framework, the section develops three new PGS algorithms (i.e., Dual-RAC, Dual-NACA, Dual-NACF). Subsequently, a theoretical analysis for convergence of the proposed algorithms are presented in Section 4.4. The design of experiments and the discussion on results are given in Section 4.5 and Section 4.6 respectively. The chapter is finally summarized in Section 4.7.

4.2 Preliminaries — A General Primal-Dual Approximation Method

This section introduces the preliminary background of the general PDA technique to pave the way for the development of new algorithms in Section 4.3. Moreover, we refer readers to the details about the general PGS framework and the typical algorithms such as RAC, NACA, NACF and NACAF in Section 2.2.5

The core notion of PDA [158] is to approximate a complicated primal problem for learning through a simpler linear dual problem that can be solved immediately. The work [158] showed that, although PDA works on a simplified linear dual problem with some loss of precision, the technique can still guarantee to solve the original learning problem accurately. A mathematical description of PDA is given below. Given a general optimization/learning problem

(i.e., ***the primal problem***)

$$f(\vec{x}^\star) = \max_{\vec{x}\in\mathcal{D}} f(\vec{x}), \tag{4.1}$$

where $\mathcal{D} \subseteq \mathbb{R}^m$ is a convex subset of the real vector space, and the convex scalar objective function $f : \mathcal{D} \to \mathbb{R}$ is Lipschitz continuous.

To solve (4.1), the work [158] reformulated (4.1) to a simple ***dual problem***. To construct the dual formulation, we assume that k ($k \geq 0$) candidate solutions $\{\vec{x}_i\}_{i=0}^k \subset \mathcal{D}$ are available in advance. Based on these candidate solutions, the dual problem is defined as

$$l_k(\vec{x}^\star) = \max_{\vec{x}\in\mathcal{D}}[\frac{1}{k+1}\sum_{i=0}^{k}[f(\vec{x}_i) + \nabla_{\vec{x}} f(\vec{x}_i)^T \cdot (\vec{x} - \vec{x}_i)] - \mu_k d(\vec{x}, \vec{x}_0)], \tag{4.2}$$

where $\mu_k = \frac{1}{2\beta(k+1)}$ is a scaling parameter, and $d(\vec{x}, \vec{x}_0)$ is an arbitrary distance measure between any two solutions, e.g., $\vec{x}$ and $\vec{x}_0$. In order to provide such k candidate solutions in (4.2), an iterative learning process is often adopted [158, 241]. Assuming that $d(\vec{x}, \vec{x}_0)$ gives the Euclidean distance, then during each learning iteration, (4.2) can be solved directly to produce the next candidate solution as

$$\begin{aligned} \vec{x}_{k+1} &= \vec{x}_0 + \beta\tfrac{1}{k+1}\textstyle\sum_{i=0}^{k} \nabla_{\vec{x}} f(\vec{x}_i) \\ &= \vec{x}_k + \beta\tfrac{1}{k+1}\nabla_{\vec{x}} f(\vec{x}_k), \end{aligned} \tag{4.3}$$

where β is the learning rate.

Comparing (4.3) to (2.39) or (2.41), they are essentially identical for both critic learning and actor learning. This implies that the dual formulation of (4.3) is not suitable to achieve our goal mentioned in Section 4.1. Therefore, we must investigate the usefulness of different reformulations from (4.2), which have not been investigated in-depth in the literature.

Recall the SGD method, (4.3) is fairly similar to the iterative process in SGD. The difference is that, for one iterative update, the gradient in SGD is computed based on the current sample, but the PDA approach uses an average of all k steps historical gradients. In other words, the iterative learning process of SGD is one-step based whereas that of PDA is k-steps based. The advantage of the PDA approach is that the error of parameter updates caused by noises of gradient estimations can be reduced. For example, if assuming the gradient estimation $\nabla_x f(x)$ and the estimation noise ε are identical for every step in k-step

learning, by following SGD updating, the x_k will have an accumulated noise of $k \times \varepsilon$. However, in PDA, the noises are averaged which results in x_k with a noise of only ε. In consequence, for PGS in RL, the effectiveness of policy gradient is highly dependent on the variance of gradient estimation [78]. The main concern of direct use of SGD onto RL algorithms relates to the high variance problem in policy gradient estimations [78, 68]. The drawback can be conquered by applying the low-variant PDA approach to policy-gradient based search algorithms.

In addition, though the work [158] has proposed a dual formulation for the general optimization, the true usefulness of such a formulation has not been assessed in RL ever before. Even similar techniques, such as mirror-descent, have been successfully applied to a single time-scale learning of VIS in [143], but its applicability to two time-scale learning in AC architecture still demands in-depth investigations. Moreover, owing to the complexity of the strongly coupled parametric learning processes in the AC framework, the suitability of the newly derived dual formulations requires further investigations.

Hence, it is worth exploring, whether there are any other suitable formulations (primal or dual), and whether these formulations can improve effectiveness for the PGS algorithms. To the utmost of our awareness, there are still very few research works in the literature focusing on investigating the usefulness of PDA from the challenging perceptive discussed above. Motivated by these understandings above, we intend to take the first step to adopt PDA for developing new PGS algorithms to solve difficult RL problems.

4.3 The Proposed Algorithms

In this section, we will gradually build up a general PDA framework for RL. We start with an introduction to a general formulation of the dual problem in PGS. Through various extensions of the dual problem formulations, we subsequently develop three different RL algorithms. Meanwhile, an analysis of the relationships with some important existing algorithms will also be highlighted.

4.3.1 General Dual Formulation for Policy Gradient Search

In this subsection, we study some general dual problem formulations based on the primal problem defined as,

$$\begin{aligned} \mathcal{J}(\vec{\theta}^{\star}) &= max_{\vec{\theta}\in\vec{\Theta}}\mathcal{J}(\vec{\theta}) \\ &= max_{\vec{\theta}\in\vec{\Theta}}[\int_{\vec{s}\in\mathcal{S}} p^{\pi}(\vec{s}) \int_{a\in\mathcal{A}(\vec{s})} \pi_{\vec{\theta}}(a|\vec{s})\mathcal{R}(\vec{s},a,\vec{s}')dad\vec{s}] \end{aligned} . \tag{4.4}$$

for actor learning as well as in (2.40) and (2.35) for critic learning. Instead of using the dual formulation given in (4.2), we will consider other useful problem formulations based on (4.2) in order to develop new and effective RL algorithms.

Without exploring all possible dual problem formulations, we concentrate on three main aspects to build new dual problems: (A1) changing the arithmetic averaging in (4.2) to the exponentially-weighted averaging in (4.5); (A2) adopting varied distance measurements for $d(\vec{x},\vec{x}_0)$ in (4.2); (A3) applying PDA to different primal problems for actor learning in (2.41) and critic learning in (2.35).

We consider mainly these three aspects because they enable us to construct different dual problems straightforwardly. They also cover the main factors and differences in formulating the dual problems. As seen from Figure 4.1, a few important dual problem formulations can be derived from (4.2). In fact, several possible aspects can be considered. For example, we can make the scaling parameter μ_k in the dual problem formulation self-adaptive. Also, we can use a different weighted averaging method by setting the norm of the gradients as the weights [158]. This implies that only the directions of the gradients will be considered during the learning process. However, we have not found empirical deviations in terms of effectiveness with the use of other possible aspects. Thus, they are not considered in this chapter. Moreover, to accurately evaluate the usefulness of adopting each specific aspect, in this study every aspect will be considered individually while developing any new algorithms.

Based on these aspects, we can build up three different dual problem formulations. These formulations give rise to three new AC algorithms as we summarize in Figure 4.1. Particularly, from Aspect (A1), we develop the dual problem

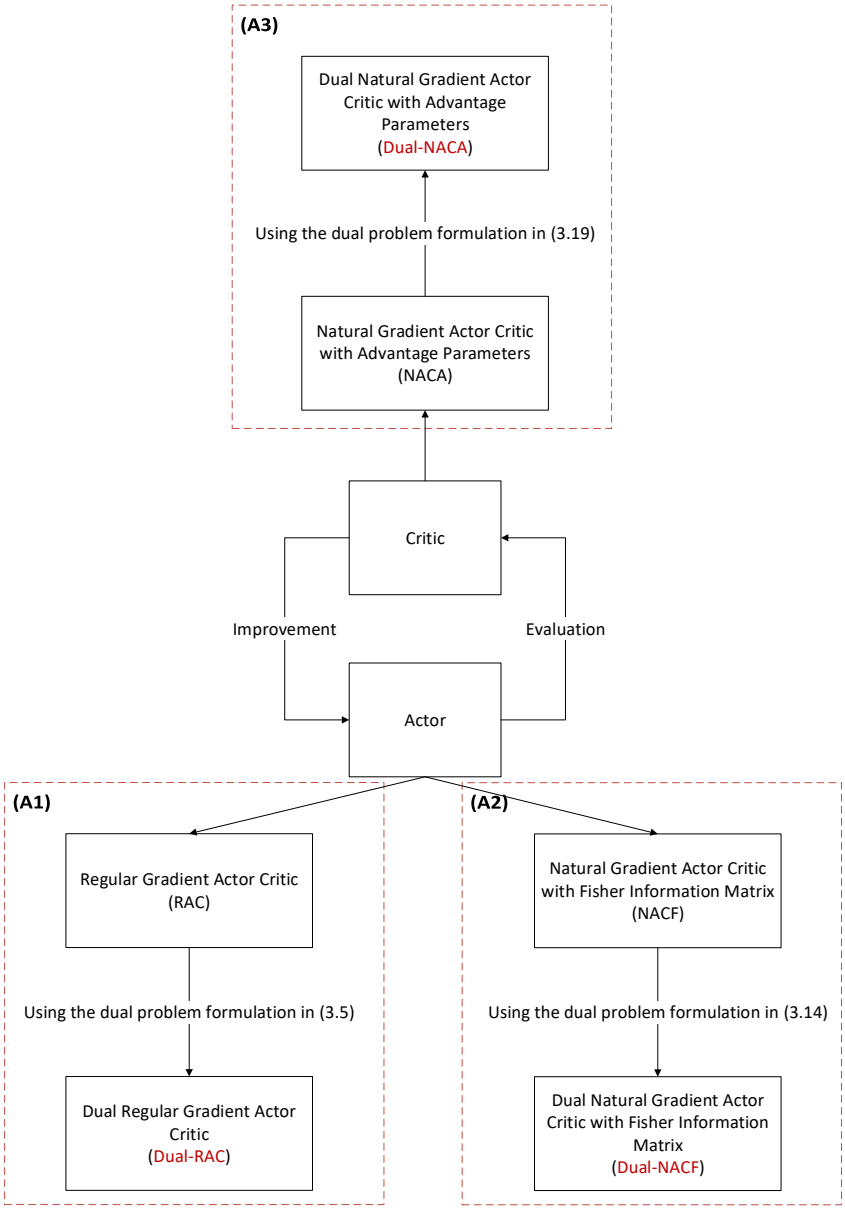


Figure 4.1: The Primal-Dual Approximation based Actor-Critic Algorithms.

formulation in (4.5) and generalize the RAC algorithm to the new **Dual Regular Gradient Actor-Critic (Dual-RAC)** algorithm. Meanwhile, following Aspect (A2), we propose another dual problem formulation (4.14), which serves as the basis for the development of the new **Dual Natural Gradient Actor-Critic with**

Fisher Information Matrix (Dual-NACF) algorithm. In line with Aspect (A3), we further derive the dual problem formulation in (4.19). Based on (4.19), **Dual Natural Gradient Actor-Critic with Advantage Parameters (Dual-NACA)** algorithm is obtained from the NACA algorithm. The technical details of each algorithm are presented in the following subsections.

4.3.2 Dual Regular Gradient Actor Critic Algorithm

Following Aspect (A1), the dual problem formulation for actor learning defined in (4.2) can be presented as

$$l(\vec{\theta}^{\star}) = \max_{\vec{\theta}\in\Theta}[\sum_{i=0}^{t}\rho^{t-i}[\mathcal{J}(\vec{\theta}_i) + \nabla_{\vec{\theta}}\mathcal{J}(\vec{\theta}_i)^T \cdot (\vec{\theta} - \vec{\theta}_i)] - \mu_t||\vec{\theta} - \vec{\theta}_0||_2^2]. \tag{4.5}$$

It is clear to see that, in this new formulation, we no longer use the arithmetic averaging method. Instead the exponentially-weighted averaging is applied. Particularly, $\rho \in (0, 1]$ is the weighting parameter given for measuring the importance of historical gradients (i.e., gradients obtained from the prior t time points) at an exponential scale. In the meantime, the regularization term can be arbitrary distance measures. For example, one can consider Kullback–Leibler divergence [170] which is normally difficult to be computed analytically. In our case, we consider the Euclidean distance for simplicity.

To analytically solve (4.5), we can simply compute the gradient of $l(\vec{\theta}^{\star})$ with respect to $\vec{\theta}$ as,

$$\begin{aligned}\frac{\partial l'_t(\vec{\theta})}{\partial\vec{\theta}} &= \frac{\partial[\frac{1}{t+1}\sum_{i=0}^{t}[f(\vec{\theta}_i)+(\frac{\partial\mathcal{J}(\vec{\theta}_i)}{\partial\vec{\theta}_i})^T\cdot(\vec{\theta}-\vec{\theta}_i)]+\mu_t d(\vec{\theta})]}{\partial\vec{\theta}} \\ &= (\tfrac{1}{t+1})\sum_{i=0}^{t}(\tfrac{\partial\mathcal{J}(\vec{\theta}_i)}{\partial\vec{\theta}_i}) + \mu_t\frac{\partial d(\vec{\theta})}{\partial\vec{\theta}}.\end{aligned} \tag{4.6}$$

Afterward, we just simply let (4.6) equal to $\vec{0}$,

$$(\frac{1}{t+1})\sum_{i=0}^{t}(\frac{\partial\mathcal{J}(\vec{\theta}_i)}{\partial\vec{\theta}_i}) + \mu_t\frac{\partial d(\vec{\theta})}{\partial\vec{\theta}} = \vec{0}, \tag{4.7}$$

Following (4.7), we can generalize the framework by introducing a tunable parameter $0 < \rho \leq 1$,

$$\frac{1}{(t+1)\sum_{i=0}^{t}\rho^{t-i}}\sum_{i=0}^{t}[\rho^{t-i}(\frac{\partial\mathcal{J}(\vec{\theta}_i)}{\partial\vec{\theta}_i})] = \mu_t\frac{\partial d(\vec{\theta})}{\partial\vec{\theta}}, \tag{4.8}$$

According to the definition of μ_t in [158] and (4.7), we can have the scaling parameter μ_t as,

$$\mu_t = \frac{\sum_{i=0}^{t} \rho^{t-i}}{2t(t+1)\sum_{i=0}^{t} \rho^{t-i}}, \tag{4.9}$$

Thereby, (4.8) can be reformulated as,

$$\sum_{i=0}^{t}[\rho^{t-i}(\frac{\partial \mathcal{J}(\vec{\theta}_i)}{\partial \vec{\theta}_i})] = (\frac{\sum_{i=0}^{t} \rho^{t-i}}{2t})\frac{\partial d(\vec{\theta})}{\partial \vec{\theta}}. \tag{4.10}$$

In such a way, we have actually generalize the typical stepwise gradient-based learning. If $\rho = 1$, the learning can downgrade to original average model shown in (4.7). Additionally, the scaling parameter is obtained as $\mu_t = \frac{\sum_{i=0}^{t} \rho^{t-i}}{2(t+1)}$.

Following the above derivations, we can obtain the policy parameter updating rule as

$$\vec{\theta}_{t+1} \leftarrow \vec{\theta}_0 + \beta_t(\frac{t+1}{\sum_{i=0}^{t} \rho^{t-i}}) \sum_{i=0}^{t}[\rho^{t-i}\nabla_{\vec{\theta}}\mathcal{J}(\vec{\theta}_i)], \tag{4.11}$$

where β_t is the learning rate for the t step.

By using the exponentially-weighted averaging method, we emphasize more on recently obtained policy gradients which are considered more important in a two time-scale learning process. Furthermore, in a two time-scale learning process, the change to policy parameter can be made to be reasonably small while learning the value function in the fast time-scale. Along with the entire learning process, the value function becomes better and better, which implies that the estimated policy gradients become more and more accurate. In fact, for a t-step learning, the gradients obtained in early steps (e.g., when $t = 0$) may not be as crucial as those obtained in later steps if the time elapsed to the t time is very long. However, in (4.2), the historical gradients are treated equally, they may not be as accurate as the gradients obtained recently. Thus we introduce the dual problem formulation in (4.5), where gradients obtained more recently are considered more important in contrast to those gradients obtained from candidates close to $\vec{\theta}_0$. Moreover, in the literature, numerous research works have shown that the adaptive changes of weights for each linear subproblem, i.e., the approximation of original problem at each candidate solution point, can potentially result in better convergence rat [130, 32]. We also performed several preliminary numerical studies on simple functions. Our study

consistently shows that the convergence rate of using exponentially-weighted averaging method is faster than that of using the arithmetic averaging method.

However, there is a key issue of directly using (4.11). The issue is that following the rule (4.11), if t is very large, the policy gradients obtained close to the time point $t = 0$ will become useless as they are not very important. This may lead to biased learning.

To demonstrate this issue further, we give a simple example in Figure 4.2. Here, we use a 2D contour graph to represent the policy parameter space. Each parameter value is represented as a black point at different time points. The red dashed vectors represent the normal updating trajectories from $\vec{\theta}_0$ to $\vec{\theta}_{t+1}$ generated by following regular gradients. Particularly, Figure 4.2 shows that, at the time point t, based on the parameter $\vec{\theta}_t$ and the gradient of $\nabla_{\vec{\theta}}\mathcal{J}(\vec{\theta}_t)$, we will get the updated parameter $\vec{\theta}_{t+1}$. On the other hand, considering the updating rule (4.11), its second part can be illustrated as the solid blue vector, i.e., $\frac{t+1}{\sum_{i=0}^{t}\rho^{t-i}}\sum_{i=0}^{t}\rho^{t-i}\nabla_{\vec{\theta}}\mathcal{J}(\vec{\theta}_t)$. This is because, when t is very large, the policy gradient (the dashed blue vector) is largely determined by the recent gradients, namely the two parts A and B shown in the figure. If still following the direction of the solid blue vector (i.e., the dashed blue vector) to conduct the updating from $\vec{\theta}_0$, the updated parameter $\vec{\theta}'_{t+1}$ will end up at an unideal position as shown in the figure. As seen clearly, $\vec{\theta}'_{t+1}$ and $\vec{\theta}_0$ are almost locating on the same contour line, resulting in no improvement in performance.

To address the issue demonstrated in Figure 4.2, we considered a periodical updating process for learning the policy parameters. This means that the policy parameters are updated every K (i.e.,the periodic interval) steps where $K > 0$ is a small constant. In other words, after every K steps, we will apply the updating rule (4.11), as a result, $\vec{\theta}_0$ becomes $\vec{\theta}_K$. Accordingly, we can rewrite (4.11) to a more general periodic updating rule as

$$\vec{\theta}_{(n+1)K} \leftarrow \vec{\theta}_{nK} + \beta_{nK}\left(\frac{nK}{\sum_{i=0}^{nK}\rho^{n(K-i)}}\right)\sum_{i=0}^{nK}[\rho^{n(K-i)}\nabla_{\vec{\theta}}\mathcal{J}(\vec{\theta}_i)], \tag{4.12}$$

where $n \in \mathbb{N}$. Additionally, within one single period, we have the step-updating rule as

$$\vec{\theta}_{t+1} \leftarrow \vec{\theta}_t + \beta_t\nabla_{\vec{\theta}}\mathcal{J}(\vec{\theta}_i). \tag{4.13}$$

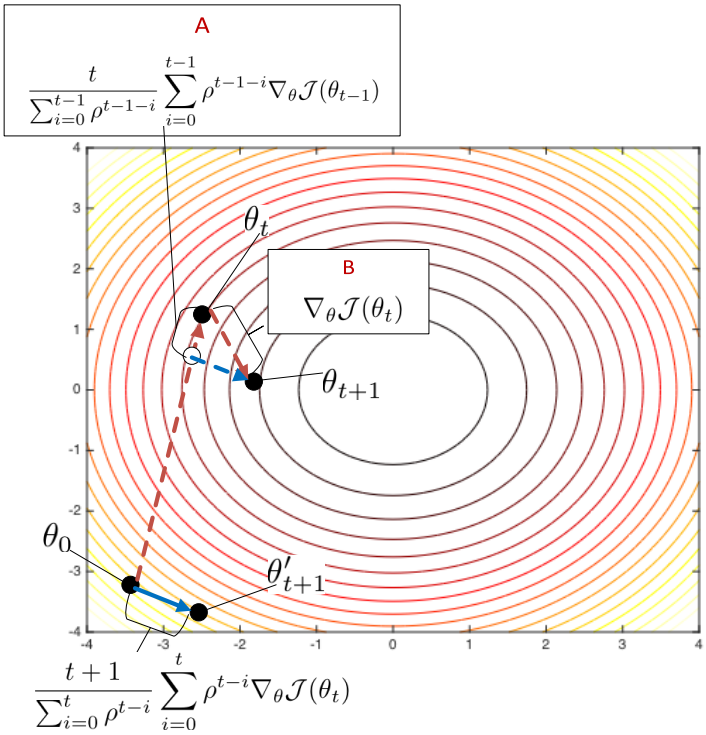


Figure 4.2: An example of the biased learning following (4.11) when t is very large.

where $0 \leq t < K$.

Note that, the RAC algorithm can be viewed as a special case of the Dual-RAC algorithm. This is because under the specific setting of $\rho = 1$, the proposed updating rules in (4.12) and (4.13) are equivalent to (2.41) adopted directly by the RAC algorithm. Hence Dual-RAC algorithm is a generalization of RAC.

Follow (4.12) and (4.13) for actor learning, we present the complete Dual-RAC algorithm in Algorithm 4.3.1. In addition, we obtain the regular gradient estimator at each learning step as $\nabla_{\vec{\theta}} \mathcal{J}(\vec{\theta}_i) = \mathbf{E}_{\vec{\theta}}[\delta_i^{\pi_i} \Phi(\vec{s}_i, a_i)]$, where $\delta_i^{\pi_i}$ and $\Phi(\vec{s}_i, a_i)$ can be obtained by (2.38) and (2.37) respectively.

Algorithm 4.3.1 Dual-RAC Algorithm

Require: an MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, the periodic step interval K.

Ensure: $\vec{\theta}$, $\vec{v}^{\pi}$

Initialization:

$\vec{\theta} \leftarrow \vec{\theta}_0$, $\vec{v}^{\pi} \leftarrow v_0^{\pi}$, $\vec{s}_t \leftarrow \vec{s}_0$, $\hat{\vec{g}} \leftarrow \vec{0}$, $\vec{g} \leftarrow \vec{0}$, $k \leftarrow 0$

Learning Process for one episode:

for $t = 0, 1, 2, ...$ **do**

  $a_t \sim \pi_{\vec{\theta}}(a|\vec{s}_t)$

  Take action a_t, observe reward r_{t+1} and new state $\vec{s}_{t+1}$

  $\delta_t^{\pi} \leftarrow r_{t+1} + \gamma \vec{v}_t^{\pi T} \cdot \vec{\phi}(\vec{s}_{t+1}) - \vec{v}_t^{\pi T} \cdot \vec{\phi}(\vec{s}_t)$

  $\vec{v}_{t+1}^{\pi} \leftarrow \vec{v}_t^{\pi} + \vec{\alpha} \delta_t^{\pi} \vec{\phi}(\vec{s}_t)$

  $\hat{\vec{g}} \leftarrow \hat{\vec{g}} + \rho \delta_t^{\pi} \vec{\Phi}(\vec{s}_t, a_t)$

  $k \leftarrow k + 1$

  $\vec{g} \leftarrow \frac{k}{\sum_{i=0}^{k} \rho^{k-i}} \hat{\vec{g}}$

  $\vec{\theta}_{t+1} \leftarrow \vec{\theta}_t + \beta \vec{\Phi}(\vec{s}_t, a_t)$

  if $k \geq K$ **then**

    $\vec{\theta}_{t+1} \leftarrow \vec{\theta}_0 + \beta \vec{g}$

    $\vec{\theta}_0 \leftarrow \vec{\theta}_{t+1}$

    $k \leftarrow 0$

  end if

end for

$\vec{\theta}_0 \leftarrow \vec{\theta}_{t+1}$, $\vec{v}^{\pi} \leftarrow v_0^{\pi}$, $\vec{s}_t \leftarrow \vec{s}_0$, $\hat{\vec{g}} \leftarrow \vec{0}$, $\vec{g} \leftarrow \vec{0}$, $k \leftarrow 0$

return $\vec{\theta}$, $\vec{v}^{\pi}$

4.3.3 Dual Natural Gradient Actor Critic with Fisher Information Matrix

Following Aspect (A2), we derive the new dual problem formulation for actor learning where the Riemannian Distance is used as the regularization term in (4.2),

$$l_t(\vec{\theta}^{\star}) = \max_{\vec{\theta} \in \Theta} [\frac{1}{t+1} \sum_{i=0}^{t} [\mathcal{J}(\vec{\theta}_i) + \nabla_{\vec{\theta}} \mathcal{J}(\vec{\theta}_i)^T \cdot (\vec{\theta} - \vec{\theta}_i)] - \mu_t (\vec{\theta} - \vec{\theta}_0)^T G(\vec{\theta}_t)(\vec{\theta} - \vec{\theta}_0)], \quad (4.14)$$

Algorithm 4.3.2 Dual-NACF Algorithm

Require: an MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$.

Ensure: $\vec{\theta}$, $\vec{v}^{\pi}$

Initialization:

$\vec{\theta} \leftarrow \vec{\theta}_0$, $G \leftarrow G_0$, $\vec{v}^{\pi} \leftarrow \vec{v}_0^{\pi}$, $\vec{s}_t \leftarrow \vec{s}_0$, $\vec{\hat{g}} \leftarrow \vec{0}$, $\vec{g} \leftarrow \vec{0}$,$k \leftarrow 0$

Learning Process:

for $k = 0, 1, 2, ...$ **do**

  for $t = 0, 1, 2, ...$ **do**

    $a_t \sim \pi_{\vec{\theta}}(a|\vec{s}_t)$

    Take action a_t, observe reward r_{t+1} and new state $\vec{s}_{t+1}$

    $\delta_t^{\pi} \leftarrow r_{t+1} + \gamma \vec{v}_t^{\pi T} \cdot \vec{\phi}(\vec{s}_{t+1}) - \vec{v}_t^{\pi T} \cdot \vec{\phi}(\vec{s}_t)$

    $\vec{v}_{t+1}^{\pi} \leftarrow \vec{v}_t^{\pi} + \alpha \delta_t^{\pi} \vec{\phi}(\vec{s}_t)$

    $\vec{\hat{g}} \leftarrow \vec{\hat{g}} + \delta_t^{\pi} \vec{\Phi}(\vec{s}_t, a_t)$

    $G_{t+1}^{-1} = \frac{1}{1-\alpha}[G_t^{-1} - \alpha \frac{(G_t^{-1}\vec{\Phi}(\vec{s}_t,a_t))(G_t^{-1}\vec{\Phi}(\vec{s}_t,a_t))^T}{1-\alpha+\alpha\vec{\Phi}(\vec{s}_t,a_t)^T G_t^{-1}\vec{\Phi}(\vec{s}_t,a_t)}]$

    $\vec{\theta}_{t+1} \leftarrow \vec{\theta}_t + \beta \delta_t^{\pi} \vec{\Phi}(\vec{s}_t, a_t)$

  end for

  $\vec{g} \leftarrow \frac{t(\rho-1)}{\rho^t - 1} G_{t+1}^{-1} \times \vec{\hat{g}}$

  $\vec{\theta}_k \leftarrow \vec{\theta}_0$

  $\vec{\theta}_{k+1} \leftarrow \vec{\theta}_k + \beta \vec{g}$

  $\vec{\theta}_0 \leftarrow \vec{\theta}_{kc+1}$, $\vec{\theta} \leftarrow \vec{\theta}_0$, $\vec{\hat{g}} \leftarrow \vec{0}$, $G \leftarrow G_0$, $\vec{v}^{\pi} \leftarrow \vec{v}_0^{\pi}$, $\vec{s}_t \leftarrow \vec{s}_0$

end for

return $\vec{\theta}$, $\vec{v}^{\pi}$

where $G(\vec{\theta}_t) = \int_{\vec{s} \in \mathcal{S}} d^{\pi}(s) \int_{a \in \mathcal{A}} \pi(\vec{s}, a) \nabla \ln \pi(\vec{s}, a) \nabla \ln \pi(\vec{s}, a)^T d\vec{s} da$ is the Fisher Information Matrix [172], here we use the same unbiased estimation $G(\vec{\theta}_t) \approx \frac{1}{t+1} \sum_{i=0}^{t} \vec{\Phi}(\vec{s}_i, a_i) \vec{\Phi}(\vec{s}_i, a_i)^T$ presented in [31] and the scaling factor $\mu_t = \frac{1}{2(t+1)}$ as in [158].

Similar to the derivation of Dual-RAC, following (4.14), we can analytically obtain the policy parameter updating rule by introducing a tunable parameter $0 \leq \rho < 1$ as

$$\vec{\theta}_{t+1} \leftarrow \vec{\theta}_0 + \beta_t \frac{t(\rho-1)}{\rho^{t+1} - 1} G(\vec{\theta}_t)^{-1} \sum_{i=0}^{t} \nabla_{\vec{\theta}} \rho^{t-i} \mathcal{J}(\vec{\theta}_i). \tag{4.15}$$

For the policy parameters updating, we can have two choices, namely step-by-step updating and periodic updating. Either of the two choices can be applied to our Dual-NACF algorithm. To have a better approximation of $G(\vec{\theta}_t)$, we choose the second choice, where the periodic interval K as explained in Section 4.3.2.

Accordingly, we can have an K interval updating rule for Dual-NACF represented as

$$\vec{\theta}_{(n+1)K} \leftarrow \vec{\theta}_{nK} + \beta_{nK} \frac{nK(\rho-1)}{\rho^{nK}-1} G(\vec{\theta}_{nK})^{-1} \sum_{i=0}^{nk} \nabla_{\vec{\theta}} \mathcal{J}(\vec{\theta}_i), \tag{4.16}$$

where $n \in \mathbb{N}$ denotes the total steps for the specified interval K. Moreover, in each specified interval, we can also have a step-based updating rule using the regular gradient as,

$$\vec{\theta}_{t+1} \leftarrow \vec{\theta}_t + \beta_t \nabla_{\vec{\theta}} \mathcal{J}(\vec{\theta}_t). \tag{4.17}$$

If setting $\rho = 0$, then the Dual-NACF algorithm becomes the NACF algorithm, which shows the latter is a special case of the former.

Moreover, based on (4.16) and (4.17), we have the pseudo codes of Dual-NACF is derived and given in Algorithm 4.3.2. Note that, we also use the Sherman-Morrison matrix inversion lemma given in (30) of NACF in [31] to update the $G(\vec{\theta})^{-1}$ incrementally in avoidance of the complex computation on $G(\vec{\theta})^{-1}$.

4.3.4 Dual Natural Gradient Actor Critic with Advantage Parameters

Following Aspect (A3), in this subsection, we concentrate on the primal problem in critic learning, i.e.,

$$\varepsilon(\vec{\omega}^\star) = \min_{\vec{\omega} \in \Omega} \mathbf{E}_{\vec{s} \sim d^\pi(\vec{s}), a \sim \pi} [(Q^\pi(\vec{s}, a) - \vec{\omega}^T \cdot \vec{\Phi}(\vec{s}, a))^2], \tag{4.18}$$

to derive the new dual problem from (4.2) as

$$l_t(\vec{\omega}^\star) = \min_{\vec{\omega} \in \Omega} [\sum_{i=0}^{t} \rho^{t-i} [\varepsilon(\vec{\omega}_i) + \nabla_{\vec{\omega}} \varepsilon(\vec{\omega}_i)^T \cdot (\vec{\omega} - \vec{\omega}_i)] + \mu_t ||\vec{\omega} - \vec{\omega}_0||_2^2], \tag{4.19}$$

Algorithm 4.3.3 Dual-NACA Algorithm

Require: an MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, the periodic step interval K.

Ensure: $\vec{\theta}$, $\vec{v}^{\pi}$

Initialization:
$\vec{\theta} \leftarrow \vec{\theta}_0$, $\vec{\omega}^{\pi} \leftarrow \vec{\omega}_0^{\pi}$, $\vec{v}^{\pi} \leftarrow \vec{v}_0^{\pi}$, $\vec{s}_t \leftarrow \vec{s}_0$, $\vec{\hat{g}} \leftarrow \vec{0}$,$\vec{g} \leftarrow \vec{0}$, $k \leftarrow 0$
Learning Process:
for $E = 0, 1, 2, ...$ **do**
  for $t = 0, 1, 2, ...$ **do**
    $a_t \sim \pi_{\vec{\theta}}(a|\vec{s}_t)$
    Take action a_t, observe reward r_{t+1} and new state $\vec{s}_{t+1}$
    $\delta_t^{\pi} \leftarrow r_{t+1} + \gamma \vec{v}_t^{\pi T} \cdot \vec{\phi}(\vec{s}_{t+1}) - \vec{v}_t^{\pi T} \cdot \vec{\phi}(\vec{s}_t)$
    $\vec{v}_{t+1}^{\pi} \leftarrow \vec{v}_t^{\pi} + \alpha \delta_t^{\pi} \vec{\phi}(\vec{s}_t)$
    $\vec{\hat{g}} \leftarrow \vec{\hat{g}} + [\vec{\Phi}(\vec{s}_t, a_t)\vec{\Phi}(\vec{s}_t, a_t)^T \vec{\omega}_t - \delta_t \vec{\Phi}(\vec{s}_t, a_t)]$
    $k \leftarrow k + 1$
    $\vec{g} \leftarrow \frac{k}{\sum_0^k \rho^k} \vec{\hat{g}}$
    if $k \geq K$ **then**
      $\vec{\omega}_0^{\pi} \leftarrow \vec{\omega}_t^{\pi}$
      $\vec{g} \leftarrow \sum_0^k \rho^k \vec{\hat{g}}$
      $\vec{\omega}_{t+1}^{\pi} \leftarrow \vec{\omega}_0^{\pi} - \alpha \vec{g}$
      $k \leftarrow 0$
    else
      $\vec{\omega}_{t+1}^{\pi} \leftarrow \vec{\omega}_0^{\pi} - \alpha[\vec{\Phi}(\vec{s}_t, a_t)\vec{\Phi}(\vec{s}_t, a_t)^T \vec{\omega}_t - \delta_t \vec{\Phi}(\vec{s}_t, a_t)]$
    end if
  end for
  $\vec{\omega}_0^{\pi} \leftarrow \vec{\omega}_{t+1}^{\pi}$
  $\vec{\theta}_{E+1} \leftarrow \vec{\theta}_E + \beta \vec{\omega}_0^{\pi}$,
  $\vec{\hat{g}} \leftarrow \vec{0}$, $k \leftarrow 0$
end for
return $\vec{\theta}$, $\vec{v}^{\pi}$

where the gradient of $\varepsilon(\vec{\omega}_i)$ is given and approximated in the original NACA algorithm, i.e.,

$$\begin{aligned} \nabla_{\vec{\omega}} \varepsilon(\vec{\omega}_i) &= 2 \int_{\vec{s} \in \mathcal{S}} d^{\pi}(\vec{s}) \int_{a \in \mathcal{A}} \pi(\vec{s}, a)[Q^{\pi}(\vec{s}, a) - \vec{\omega}^T \cdot \vec{\Phi}(\vec{s}, a)]\vec{\Phi}(\vec{s}, a) d\vec{s} da, \\ &\approx \vec{\Phi}(\vec{s}_i, a_i)\vec{\Phi}(\vec{s}_i, a_i)^T \vec{\omega}_i - \delta_i^{\pi_i} \vec{\Phi}(\vec{s}_i, a_i). \end{aligned} \tag{4.20}$$

Analogue to Dual-RAC and Dual-NACF, we can also obtain the critic parameter updating rule by analytically solving (4.19) with introducing the parameter $0 \leq \rho < 1$ and let $\mu_t = \frac{\sum_{i=0}^{t} \rho^{t-i}}{2(t+1)}$, i.e.,

$$\vec{\omega}_{t+1}^{\pi} \leftarrow \vec{\omega}_{0}^{\pi} - \alpha_t (\frac{t+1}{\sum_{i=0}^{t} \rho^{t-i}}) \sum_{i=0}^{t} [\rho^{t-i} \nabla_{\vec{\omega}} \varepsilon(\vec{\omega}_i)], \tag{4.21}$$

where $\rho \in (0, 1]$ and α_t is the learning rate for critic learning at step t.

The development of the Dual-NACA algorithm and the Dual-RAC algorithm follow the same principle. Particularly, the K step periodic learning for Dual-NACA can be defined as

$$\vec{\omega}_{(n+1)K} \leftarrow \vec{\omega}_{nK} - \alpha_{nK} (\frac{nK}{\sum_{i=0}^{nK} \rho^{nK-i}}) \sum_{i=0}^{nK} [\rho^{nK-i} \nabla_{\vec{\omega}} \varepsilon(\vec{\omega}_i)]. \tag{4.22}$$

Additionally, within one episode, we have the single step updating rule,

$$\vec{\omega}_{t+1} \leftarrow \vec{\omega}_t - \alpha_t \nabla_{\vec{\omega}} \varepsilon(\vec{\omega}_t), \tag{4.23}$$

where $0 \leq t < K$.

After K steps periodic learning, $\vec{\omega}_{(n+1)K}$ becomes an accurate approximation of the natural gradient as evidenced in [31]. Because of this, the policy parameter updating will be conducted at the end of each episode, i.e.,

$$\vec{\theta}_{E+1} \leftarrow \vec{\theta}_E + \beta \vec{\omega}_{(n+1)K}, \tag{4.24}$$

where $\vec{\omega}_{(n+1)*K}$ is the updated critic parameter obtained by following the periodic updating in (4.22) until the end of an episode E. The pseudo code of Dual-NACA is given in Algorithm 4.3.3.

Note that, following the discussion for Dual-RAC in Section 4.3.2, the Dual-NACA algorithm also generalizes NACA algorithm which adopts the setting of $\rho = 1$ and $K = 1$ in (4.22).

4.4 Theoretical Analysis

In this section, we present a theoretical convergence analysis for our algorithms. Similar to the analysis procedure in [31], our investigation starts at introducing

the learning as a multi-scale learning process, and later turns to view the ACRL algorithms as two time-scale learning process where the convergence under certain conditions can be guaranteed. Essential assumptions for the correctness of our proof have also been discussed in details. Lastly, we present the convergence analysis for each of our generalized dual AC algorithms.

4.4.1 Learning as Multi Time-Scale Stochastic Approximation

Learning as the two time-scale stochastic approximation is a process adhered to two coupled updating recursions. Each recursion has different decreasing step-sizes so that one recursion can learn faster than the other. In contrast, learning in the manner of single time-scale approximation involves solely one recursive learning process. Here, we first present the single time-scale learning process briefly, followed by a detailed description of two time-scale learning process, and finally discuss their applicabilities to convergence analysis for reinforcement learning algorithms.

Single Time-Scale Learning Process

Single time-scale learning process is also known as the classical stochastic approximation algorithm [129]. The algorithm has only a recursive updating a sequence along with a time interval $t \geq 0$ shown as,

$$X_{t+1} = X_t + \alpha_t(f(X_t) + N_{t+1}), \tag{4.25}$$

where $f : \mathbb{R}^d \to \mathbb{R}^d, d \geq 1$ is a Lipschitz continuous function, $\{\alpha_t\} \subset \{0, \infty\}$ satisfy $\sum_t \alpha_t = \infty$, $\sum_t \alpha_t < \infty$, as well as $\lim_{t\to\infty} \alpha_t \to 0$, and $\{N_t, \mathcal{F}_t\}$ is a martingale sequence for the σ-fields $\mathcal{F}_t = \sigma(X_n, N_n, n \leq t), t \geq 0$.

To analyze the asymptotic behavior of (4.25), one can consider the following Ordinary Differential Equation (ODE) as,

$$\dot{X}_t = f(X_t), \tag{4.26}$$

where we assume that 1) a unique solution exists for any initial conditions for all $t \geq 0$, 2) there is a globally asymptotically stable attractor $\Im$. Following that,

let $\Im^{\epsilon}$ denote the ϵ-neighborhood of $\Im$, i.e., $\Im^{\epsilon} = \{X | ||X - Y|| < \epsilon, Y \in \Im\}$ for a given $\epsilon > 0$.

Following the Lemma 6 presented in [31], the learning process can ensure X_t as well as a $(T, \triangle)$-perturbation of (4.26) converging to $\Im^{\epsilon}$ with probability one for given $\epsilon > 0, T > 0, \triangle > 0$. Sophisticated derivations and proofs can be found in works [31, 34].

Two Time-Scale Learning Process

Evidenced in literature, the aforementioned paradigm is inadequate for some recent applications [76, 121, 122, 34]. This is because RHS of (4.25) may require an extra recursion to evaluate in these applications. Motivated by this, the two time-scale learning process had been brought to the foreground and applied to many areas like reinforcement learning, signal processing, and admission control in communication networks [34, 113].

Two time-scale learning process can be viewed as a variant of the aforementioned single time-scale learning process. Following the same notations stated above, we can have the two time-scale learning process as two coupled iterations, i.e.,

$$\begin{aligned} X_{t+1} &= X_t + \alpha_t(f(X_t, Y_t) + N_{t+1}), \\ Y_{t+1} &= Y_t + \beta_t(g(X_t, Y_t) + N'_{t+1}), \end{aligned} \tag{4.27}$$

where $f : \mathbb{R}^{d+d'} \to \mathbb{R}^d$, $g : \mathbb{R}^{d+d'} \to \mathbb{R}^d$ are Lipschitz continuous; $\{\alpha_t\}$, $\{\beta_t\}$ are step-sizes subject to $\sum_t \alpha_t = \sum_t \beta_t = \infty$, $\sum_t \alpha_t^2, \sum_t \beta_t^2 < \infty$. $\{N_t, \bar{\mathcal{F}}_t\}$, $\{N'_t, \bar{\mathcal{F}}_t\}$ are two martingale sequences with respect to the σ-fields $\bar{\mathcal{F}}_t = \sigma\{X_n, Y_n, N_n, N'_n, n \leq t\}, t \geq 0$ satisfying $\sum_t \alpha_t N_t < \infty, \sum_t \beta_t N'_t < \infty$.

Owing to $\beta_t = o(\alpha_t)$, the schedule with step size α achieves faster convergence speed than the second schedule with rate β. Thus, to analyze the asymptotic behavior of (4.27), one can track the following ODE,

$$\begin{aligned} \dot{X}_t &= f(X_t, Y_t) \\ \dot{Y}_t &= 0. \end{aligned} \tag{4.28}$$

As $\dot{Y}_t = 0$, one may also consider the ODE,

$$\dot{X} = f(X_t, Y), \tag{4.29}$$

where Y is a constant.

Essential Assumptions

For the correctness of convergence analysis, six assumptions in the literature [33, 129, 31] are essential which are given in this subsection.

Assumption 1.

$$\sup_t ||X_t|| < \infty$$
$$\sup_t ||Y_t|| < \infty$$

Assumption 2. *The ODE* (4.29) *has a unique global asymptotically stable equilibrium* $\lambda(Y(t))$ *and* $\lambda : \mathbb{R}^d \to \mathbb{R}^{d'}$ *is a Lipschitz continuous function.*

Assumption 3. *The ODE*

$$\dot{Y} = g(\lambda(Y_t), Y_t) \tag{4.30}$$

also has a unique global asymptotically stable equilibrium Y^*.

Under the above assumptions, one can follow Theorem 1 presented in [31] to prove that the two time-scale learning process converges with probability one, i.e., $(X_t, Y_t) \to (\lambda(Y_t), Y_t)$ as $t \to \infty$. Related proofs can be found in works [31, 34, 129].

Assumption 4. *The sequence of states* $\{\vec{s}_t\}$, $t = 0, 1, 2, \ldots$ *(i.e., a Markov Chain) produced by an MDP is irreducible and aperiodic while following arbitrary policy.*

Assumption 5. *Every policy* $\pi(\vec{s}, a)$ *is continuously differentiable w.r.t its parameter* $\vec{\theta}$ *for any state-action pair* $(\vec{s}, a)$.

Assumption 6. *The state features (i.e., basis functions)* $\vec{\phi}$ *has full column rank, which means that* $\vec{\phi}(\vec{s}) = \{\phi_i(\vec{s})\}, i = 0, 1, 2, \ldots, k$ *(i.e., a* $d \times k$ *matrix) are linearly independent and* $k \leq d$. *Also, for every* $\upsilon \in R^k$, $\vec{\phi}\upsilon \neq e$ *where e is the* k*-dimensional vector with all entries equal to one.*

Additionally, we regard the total rewards maximization problem as a cost minimization problem associating with negative rewards. Followed the discussion above, the convergence of our dual algorithms has been presented below.

4.4.2 Convergence Analysis

In this section, we first provide Proposition 1 that guarantees the convergence of Dual-RAC followed by a complete theoretical proof. Next, we present the other two propositions (i.e., Proposition 2 and Proposition 3) for convergence guarantees of Dual-NACF and Dual-NACA respectively. The proofs for the latter two propositions are very similar to that of Proposition 1, which are hence not explicitly given in the chapter.

The convergence analysis of Dual-RAC follows and extends the analysis of RAC presented in [31]. The key idea of this analysis is to find a way to categorize Dual-RAC into the two time-scale learning process. Distinct from the single-step parameter updating process of RAC, Dual-RAC features a k-step parameter updating process where k is a pre-defined step-length. In what follows, we give a detailed analysis on the case $k = 2$ and explain how to transform the 2-step updating to its step counterpart. Afterward, we also discuss how to extend the analysis to $k \to \infty$.

Proposition 1. *Under Assumptions 1- 6, given some small $\eta > 0$* [2] *and $\epsilon > 0$, $\exists \delta > 0$ such that for $\vec{\theta}_t$, $t \geq 0$ obtained from Dual-RAC, if $\sum_{i=0}^{t} \sup_{\vec{\theta}_i} ||e^{\vec{\theta}_i}|| < t\delta$, also $\nabla_{\vec{\theta}} \mathcal{J}(\vec{\theta})$ and $\nabla^2_{\vec{\theta}} \mathcal{J}(\vec{\theta})$ are bounded, then $\vec{\theta}_t \to \Im^{\epsilon}$ as $t \to \infty$ with probability one.*

Proof. Assumptions 1- 6 are required here. Let us recall that the single-step policy parameter learning can be regarded as,

$$\vec{\theta}_{t+1} = \vec{\theta}_t - \beta_t \nabla_{\vec{\theta}_t} \mathcal{J}(\vec{\theta}_t). \tag{4.31}$$

Following (4.31), we can obtain a sequence of policy parameters over time, i.e.,

$$\vec{\theta}_0, \ldots, \vec{\theta}_t.$$

Our idea for the parameter updating in Dual-RAC is to treat the k-step as a single step shown as below,

$$\underbrace{\vec{\theta}_0, \ldots, \vec{\theta}_k}_{\hat{\vec{\theta}}_1}, \underbrace{\vec{\theta}_{k+1}, \ldots, \vec{\theta}_{k+k}}_{\hat{\vec{\theta}}_2}, \ldots, \underbrace{\vec{\theta}_{t-k}, \ldots, \vec{\theta}_t}_{\hat{\vec{\theta}}_t}.$$

[2] η is a new learning rate introduced to mitigate the affect of higher order terms, when expanding the first order gradient of $\mathcal{J}$, i.e., $\nabla_{\vec{\theta}} \mathcal{J}(\vec{\theta})$, with Taylor Series. Thus, the value must be assumed to be small enough.

Accordingly, we can have another variant single-step updating sequence of parameters, i.e.,

$$\hat{\vec{\theta}}_0, \ldots, \hat{\vec{\theta}}_t.$$

Thus, following the findings in previous research [31], we can easily understand the convergence of sequence $\hat{\vec{\theta}}_t$ if each $\hat{\vec{\theta}}_t$ is bounded.

Next, we need to further investigate the boundedness of each $\hat{\vec{\theta}}_t$. For simplicity, here we only consider the case while $k = 2$, i.e.,

$$\underbrace{\vec{\theta}_0, \vec{\theta}_1, \vec{\theta}_2}_{\hat{\vec{\theta}}_1}, \underbrace{\vec{\theta}_2, \vec{\theta}_3, \vec{\theta}_4}_{\hat{\vec{\theta}}_2}, \ldots, \underbrace{\vec{\theta}_{t-2}, \vec{\theta}_{t-1}, \vec{\theta}_t}_{\hat{\vec{\theta}}_t}.$$

Let us consider the objective function $\mathcal{J}(\vec{\theta})$ which expanded at $\vec{\theta}_0$, we can obtain,

$$\mathcal{J}(\vec{\theta}) \approx \hat{\mathcal{J}}(\vec{\theta}) = \mathcal{J}(\vec{\theta}_0) + \nabla_{\vec{\theta}_0}\mathcal{J}(\vec{\theta} - \vec{\theta}_0) + \frac{1}{2}\nabla^2_{\vec{\theta}_0}\mathcal{J}(\vec{\theta} - \vec{\theta}_0)^2 + \varepsilon(\vec{\theta} - \vec{\theta}_0)^3, \quad (4.32)$$

where we conventionally assume $\nabla_{\vec{\theta}}\mathcal{J}(\vec{\theta})$ and $\nabla^2_{\vec{\theta}}\mathcal{J}(\vec{\theta})$ are bounded, $\varepsilon(\vec{\theta} - \vec{\theta}_0)^3$ is the infinitesimal of higher order which can be viewed as a noise term N'. Using the first small sequence $\vec{\theta}_0, \vec{\theta}_1, \vec{\theta}_2$ as a starting point, based on (4.11), (4.27), (4.31) and (4.32), we can obtain the equation,

$$\vec{\theta}_2 = \vec{\theta}_0 - \beta_1[\frac{1+\rho}{\rho}[\rho\nabla_{\vec{\theta}_0}\mathcal{J}(\vec{\theta}_0) + \nabla_{\vec{\theta}_1}\mathcal{J}(\vec{\theta}_1) + \rho e^{\vec{\theta}_0}(\vec{s}_0) + e^{\vec{\theta}_1}(\vec{s}_1)] + N'], \quad (4.33)$$

where $\rho \in [0, 1]$, N' is the noise term. Here, we keep expanding the first order gradient $\nabla_{\vec{\theta}_1}\mathcal{J}(\vec{\theta}_1)$ with Taylor Series, afterwards, we introduce another learning rate η which is reasonably small to mitigate the impacts of higher order terms. Consequently, based on (4.32) we can have,

$$\nabla_{\vec{\theta}_1}\mathcal{J}(\vec{\theta}_1) = \nabla_{\vec{\theta}_0}\mathcal{J}(\vec{\theta}_0) + \eta\nabla_{\vec{\theta}_0}\mathcal{J}(\vec{\theta}_0)\nabla^2_{\vec{\theta}_0}\mathcal{J}(\vec{\theta}_0) + \eta^2\varepsilon[\nabla_{\vec{\theta}_0}\mathcal{J}(\vec{\theta}_0)]^2. \quad (4.34)$$

From (4.33) and (4.34), let $\beta_1\frac{1+\rho}{\rho} = A$, we can show that,

$$\begin{aligned}\vec{\theta}_2 \quad &= \vec{\theta}_0 - A[(\rho+1)\nabla_{\vec{\theta}_0}\mathcal{J}(\vec{\theta}_0) + \eta\nabla_{\vec{\theta}_0}\mathcal{J}(\vec{\theta}_0)\nabla^2_{\vec{\theta}_0}\mathcal{J}(\vec{\theta}_0) \\ &\quad +\eta^2\varepsilon[\nabla_{\vec{\theta}_0}\mathcal{J}(\vec{\theta}_0)]^2 + \rho e^{\vec{\theta}_0}(\vec{s}_0) + e^{\vec{\theta}_1}(\vec{s}_1)].\end{aligned} \quad (4.35)$$

Let $\rho+1=B$, $\eta\nabla^2_{\vec{\theta}_0}\mathcal{J}(\vec{\theta}_0)=C$, $\eta^2\varepsilon\nabla_{\vec{\theta}_0}\mathcal{J}(\vec{\theta}_0)=D$, then from (4.35) we can obtain,

$$\begin{aligned}\vec{\theta}_2 &= \vec{\theta}_0 - A[B\nabla_{\vec{\theta}_0}\mathcal{J}(\vec{\theta}_0) + C\nabla_{\vec{\theta}_0}\mathcal{J}(\vec{\theta}_0) \\ &\quad + D\nabla_{\vec{\theta}_0}\mathcal{J}(\vec{\theta}_0) + \rho e^{\vec{\theta}_0}(\vec{s}_0) + e^{\vec{\theta}_1}(\vec{s}_1)].\end{aligned} \tag{4.36}$$

Consequently, we can see that if η is reasonably small, the update to $\vec{\theta}_2$ can be bounded as,

$$\vec{\theta}_2 \leq \vec{\theta}_0 - A(B+C+D)\nabla_{\vec{\theta}_0}\mathcal{J}(\vec{\theta}_0) - A[\rho e^{\vec{\theta}_0}(\vec{s}_0) + e^{\vec{\theta}_1}(\vec{s}_1)]. \tag{4.37}$$

Note that, we assume that $\sup_{\pi_t}||e^{\pi_t}(\vec{s})|| < \delta$ for some small $\delta > 0$. Therefore

$$\sum_{i=0}^{t}\sup_{\pi_i}||e^{\pi_i}(\vec{s})|| < \sum_{i=0}^{t}\delta = t\delta, \tag{4.38}$$

where $t \to \infty$ gives $\delta \to 0$.

Following the above derivations, we can easily extend the conclusion of (4.37) to other cases when $k \geq 2$, i.e.,

$$\vec{\theta}_k \leq \vec{\theta}_0 - (k-1)A(B+C+D)\nabla_{\vec{\theta}_0}\mathcal{J}(\vec{\theta}_0) - (k-1)A[\rho e^{\vec{\theta}_0}(\vec{s}_0) + e^{\vec{\theta}_1}(\vec{s}_1) + \cdots + e^{\vec{\theta}_{k-1}}(\vec{s}_{k-1})]. \tag{4.39}$$

Up to now, we can consider the ODE,

$$\dot{\vec{\theta}} = \hat{\Gamma}(-\nabla_{\vec{\theta}_0}\mathcal{J}(\vec{\theta}_0) - e^{\vec{\theta}}), \tag{4.40}$$

where $e^{\vec{\theta}} = \rho\sum_{\vec{s}\in\mathcal{S}} d^{\vec{\theta}_0}(\vec{s})[\nabla\bar{V}^{\vec{\theta}_0}(\vec{s}) - \nabla\vec{v}^{\vec{\theta}_0}\vec{\phi}(\vec{s})] + \sum_{\vec{s}\in\mathcal{S}} d^{\vec{\theta}_1}(\vec{s})[\nabla\bar{V}^{\vec{\theta}_1}(\vec{s}) - \nabla\vec{v}^{\vec{\theta}_1 T}\vec{\phi}(\vec{s})]$, as following (4.38), $e^{\vec{\theta}}$ is bounded. Meanwhile, let us consider another ODE,

$$\dot{\vec{\theta}} = \hat{\Gamma}(-\nabla_{\vec{\theta}_0}\mathcal{J}(\vec{\theta}_0)), \tag{4.41}$$

Following similar steps in the proof for Theorem 2 in [31], we can let $\bar{\vec{\theta}}(z), z > 0$ be continuous linear interpolation of $\vec{\theta}_t$ over intervals $[n(t), n(t+1)]$, $t \geq 0$, i.e., $\bar{\vec{\theta}}(n(t)) = \vec{\theta}_t$. Thus, we can have for any $\triangle > 0$, $\exists z(\triangle) > 0$ such that $\bar{\vec{\theta}}(z(\triangle)+\cdot)$ is a $(T, \triangle)$-perturbation of ODE (4.40). As a consequence, we follow Lemma 6 proposed in [31], the learning process of $\vec{\theta}$ converges to a local equilibrium of (4.41) as (4.38) holds. □

Following the similar proof of Dual-RAC above and Theorem 3 in [31], we can also have the following convergence guarantee for Dual-NACF as in Proposition 2 below,

Proposition 2. *Under Assumptions 1- 4, given some small $\eta > 0$ and $\epsilon > 0$, $\exists \delta > 0$ such that for $\vec{\theta}_t$, $t \geq 0$ obtained from Dual-NACF, if $\sum_{i=0}^{t} \sup_{\vec{\theta}_i} ||e^{\vec{\theta}_i}|| < t\delta$, also $\nabla_{\vec{\theta}} \mathcal{J}(\vec{\theta})$ and $\nabla^2_{\vec{\theta}} \mathcal{J}(\vec{\theta})$ are bounded, then $\vec{\theta}_t \to \Im^{\epsilon}$ as $t \to \infty$ with probability one.*

Furthermore, with a similar proof manner in the proof of Proposition 1 and Theorem 4 in [31], we obtain the following Proposition 3.

Proposition 3. *Under Assumptions 1- 6, given some small $\eta > 0$ and $\epsilon > 0$, $\exists \delta > 0$ such that for $\vec{\theta}_t$, $t \geq 0$ obtained from Dual-NACA, if $\sum_{i=0}^{t} \sup_{\vec{\theta}_i} ||e^{\vec{\theta}_i}|| < t\delta$, also $\nabla_{\vec{\theta}} \mathcal{J}(\vec{\theta})$ and $\nabla^2_{\vec{\theta}} \mathcal{J}(\vec{\theta})$ are bounded, then $\vec{\theta}_t \to \Im^{\epsilon}$ as $t \to \infty$ with probability one.*

4.5 Design of Experiments

In this section, we investigate the usefulness of different dual problem formulations by evaluating the effectiveness of corresponding dual algorithms presented in Section 4.3. The evaluations have been conducted on six benchmarked continuous control tasks provided by Bullet Physical Engine [220] and GYM benchmark environments [39], including Bipedal Walker, LunarLander, Mountain Car Continuous, Pendulum, Inverted Double Pendulum, and Inverted Pendulum Swingup. The detailed descriptions of each benchmark problem can be found in Section 3.1. Subsequently, the detailed setups for experiments including competing algorithms, value function and policy representations, and hyper-parameter configurations. Lastly, the experiment design in line with our research goals stated in Section 4.1.1 is presented.

4.5.1 Experiment Setup

We describe the overall experimental setups in this subsection. The description includes: (1) the competing algorithms including ARS and PPO-Linear, (2) the stochastic policy distribution adopted by the agent to learn, i.e., the Gaussian

distribution; (3) the representations for value function (i.e., NNs) and policy (i.e., linear parametric representations) used in our experiments; (4) the hyper-parameter configurations for adjusting the behaviors of the algorithms, such as the learning rate and the discount factor.

Competing Algorithms

In our experiments, we consider six competing algorithms, including RAC, NACF, NACA, NACAF, ARS, and PPO-Linear, because of two major reasons. First, to evaluate the effectiveness of our dual algorithms, we need to compare the performance of newly proposed algorithms to that of their counterparts, i.e., RAC, NACF, and NACA. Second, to position our dual algorithms in the context of state-of-the-art algorithms, we compare with two cutting-edge algorithms closely related to our work in this chapter, i.e., ARS [146] and PPO-Linear [191]. As a matter fact, we have not included another related work [178], because it essentially is an episodic natural actor-critic with a linear policy which is very similar to the design of PPO. Also, PPO has been reported as the best-performing algorithms on challenging control problems in comparison to many state-of-the-art PGS algorithms [191]. Thus, we have decided to modify the PPO algorithm [178] to support linear policy representation for our experiments. Empirically, we have found that the adapted PPO (i.e., PPO-Linear) is still effective on most control continuous problems, which satisfies our experimental requirements. To ensure good performance for all competing algorithms, we rely on high-quality algorithm implementations provided by OpenAI Baselines [3] [56].

Value Function and Policy Representations

As discussed in Section 2.2.5, AC algorithms rely heavily on the quality of value function learning. To ensure the quality of value function learning, we decided to represent the value function by using an NN which has shown proven effectiveness in the literature [178, 212]. Thus, we present the NN architecture

[3] Our implementations of all algorithms can be found at https://github.com/yimingpeng/primal_dual_baseline

for value function representation first in this subsection. Following that, we describe the stochastic policy implementation where the policy is represented as a linear parametric function.

Value Function Representation

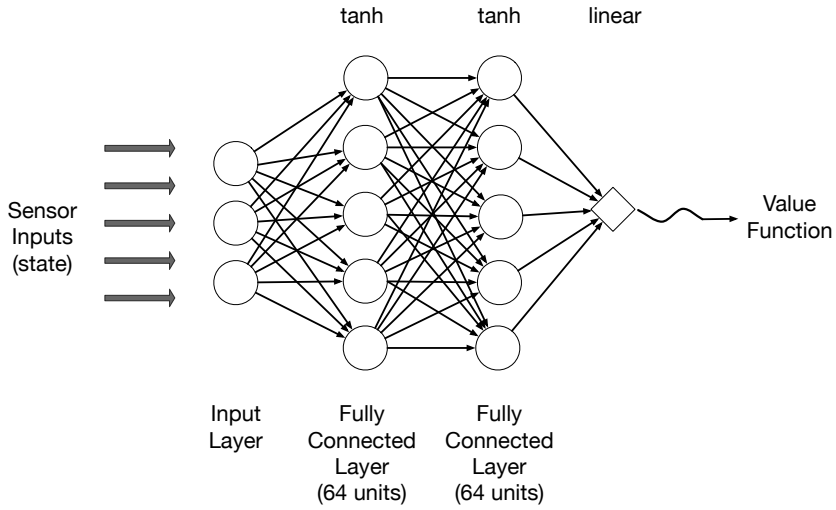


Figure 4.3: The Architecture of NN for representing Value Function for Dual-RAC, Dual-NACF, Dual-NACA, RAC, NACF, NACA, NACAF, and PPO-Linear.

For fair comparisons, we consistently use the same network architecture for all competing AC algorithms except for ARS which does not require a value function for policy search. In our experiments, we adopt the commonly-used network architecture when being applied to continuous control problems in the literature [191, 189, 56]. The architecture is illustrated in Figure 4.3.

Stochastic Policy Implementation

Two commonly used stochastic policy implementations in the literature are the Gaussian distribution for continuous problems or Gibbs distribution for discrete problems [168, 52]. As our research only is interested in the continuous

problem, we select the Gaussian policy for all experiments which is parameterized by $\vec{\theta}$ as

$$\pi_{\vec{\theta}}(a|\vec{s}) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(a-\mu)^2}{2\sigma^2}}, \tag{4.42}$$

where $\mu = \vec{\theta}^T \cdot \vec{\phi}(\vec{s})$ [4] is the mean action output from policy $\pi_{\vec{\theta}}$ in state $\vec{s}$, which can be adjusted by changing policy parameters $\vec{\theta}$. Following the common setup in the literature [171], the exploration meta-parameter (i.e., the standard deviation) is fixed for all experiments, i.e., $\sigma = 1.0$. Note that, π at the RHS of the (4.42) is the regular circumference ratio.

Hyper-parameter Configurations

Regarding the meta-parameters settings (e.g., learning rate, discount factor, etc.) algorithms' performance comparisons, we have adopted the hyper-parameter configurations differently according to the best reported settings for competing algorithms (i.e., ARS, PPO-Linear, RAC, NACF, NACA and NACAF) in the literature [146, 191, 31]. For our proposed dual algorithms, we followed the same settings to their counterparts. All the important configurations can be found in Table 4.1, we refer readers to relevant papers or our implementation in Github for more detailed configurations.

Based on Table 4.1, we briefly recapitulate the essence of these meta-parameters here: α and β are learning rates for critic and actor respectively; γ is the future reward discount factor; κ represents the coefficient for Fisher Information matrix; ρ controls the importance level of historical gradients while estimating new gradient in dual algorithms; K is a periodic interval window for Dual-RAC and Dual-NACA to conduct the periodic updating, σ is the standard deviations for noise in ARS.

4.5.2 Experiment Design

In the experiments, we evaluate our algorithms in terms of learning effectiveness. Following the standard setting in the literature [191, 56, 189], effectiveness

[4] Note that, the policy adopts a linear parametric representation.

Table 4.1: The Hyper-parameter settings of all algorithms including RAC, NACF, NACA, NACAF, Dual-RAC, Dual-NACF, Dual-NACA, ARS and PPO-Linear used for all problems.

Algorithms	Hyper-parameters						
	α	β	γ	κ	ρ	K	σ
RAC	3e-4	3e-5	0.99	N/A	N/A	N/A	N/A
NACF	3e-4	3e-5	0.99	1.0	N/A	N/A	N/A
NACA	3e-4	3e-5	0.99	N/A	N/A	N/A	N/A
NACAF	3e-4	3e-5	0.99	N/A	N/A	N/A	N/A
Dual-RAC	3e-4	3e-5	0.99	N/A	0.95	5	N/A
Dual-NACF	3e-4	3e-5	0.99	1.0	1.0	5	N/A
Dual-NACA	3e-4	3e-5	0.99	N/A	0.95	5	N/A
ARS	N/A	0.025	0.99	N/A	N/A	N/A	0.1
PPO-Linear	3e-4	3e-4	0.99	N/A	N/A	N/A	N/A

is defined as the average total rewards of the last 100 episodes [5].

To determine any performance significant differences in experiment results, we perform 30 independent runs for all algorithms on each continuous control task. At every 10,000 samples, we conduct one independent testing episode with a deterministic policy. The testing episode is performed in a separated testing environment with the same random seed as the one used for training. In doing so, we can also identify the true effectiveness as well as sample efficiency of each algorithm. All these independent tests are carried out by using the best policy learned so far till the testing point (i.e., every 10,000 samples). For all three experiments, we have performed training on each algorithm for only 5,000,000 samples (i.e., 5 million steps) due to the computational resource limitation.

[5]One episode indicates a sequence of interactions (i.e., state transitions) between an agent and an environment, which ends with some terminal conditions. For example, in the Cart Pole problem, one episode starts when the agent balances the pole and terminates when the pole falls down.

4.6 Results and Discussion

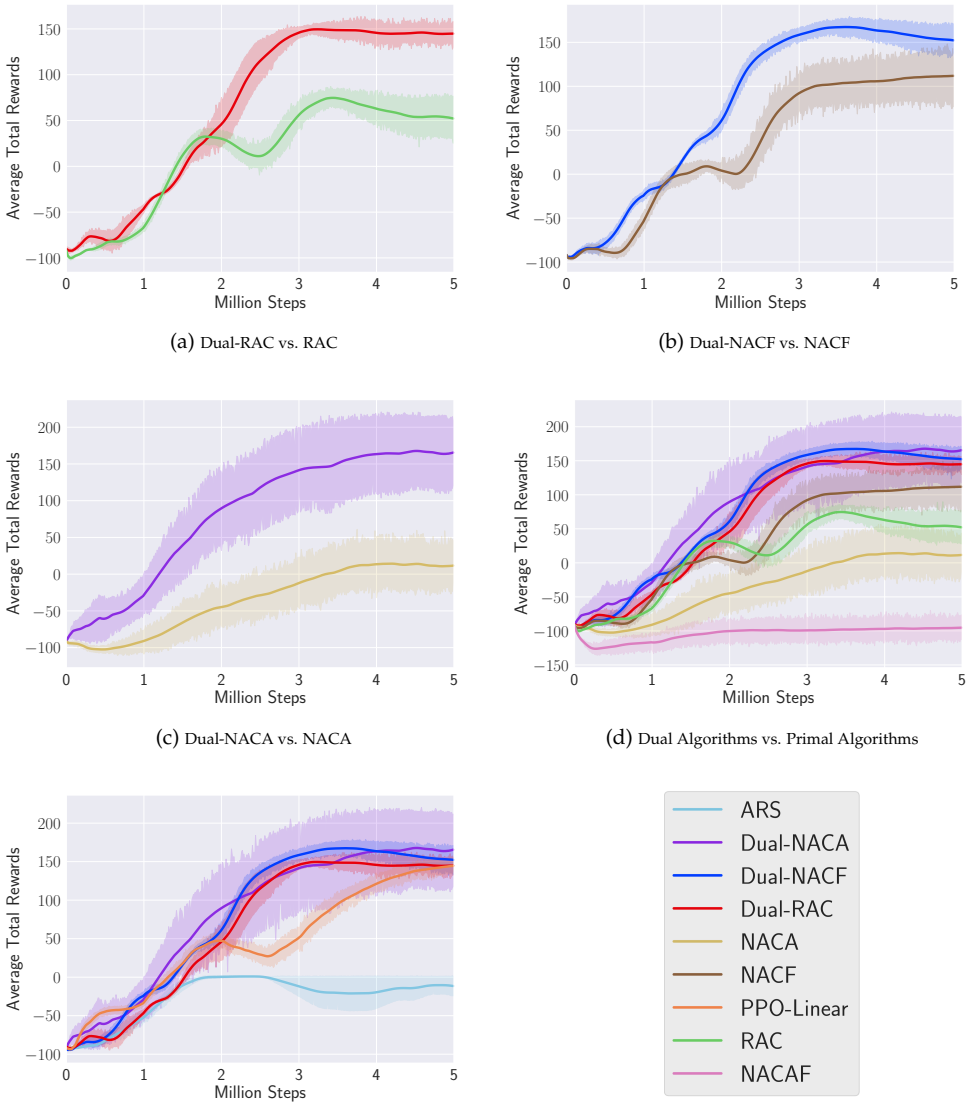
In what follows, we present experimental results with discussions of nine algorithms (i.e., Dual-RAC, Dual-NACF, Dual-NACA, RAC, NACF, NACA, NACAF, ARS, and PPO-Linear) on the six benchmark problems (i.e., Bipedal Walker, Lunar Lander, Mountain Car Continuous, Inverted Pendulum, Inverted Double Pendulum, and Inverted Pendulum Swingup). We present evaluations with respect to each problem to clearly show the performance difference, since the performance of the same algorithm may vary largely on different problems.

In the experiment on each problem, we first compare each newly proposed dual algorithm against its primal counterpart to assess the performance difference. Next, an overview of all dual algorithms in comparison to all their counterparts is presented. Lastly, we investigate the performance of all dual algorithms in comparison to the two cutting-edge algorithms, i.e., ARS and PPO-Linear.

4.6.1 Discussion on Results of Bipedal Walker

Figure 4.4 illustrates the learning curves of the proposed dual algorithms in comparison to the competing algorithms. As can be seen from Figure 4.4a- 4.4c, all three new dual algorithms (i.e., Dual-RAC, Dual-NACF and Dual-NACA) perform significantly better than their counterparts (i.e., RAC, NACF, NACA) on the Bipedal Walker problem. In addition, Figure 4.4d shows that there is no significant performance difference among all three dual algorithms. Lastly, compared to the cutting-edge algorithms, all three dual algorithms outperform ARS and achieve competitive performance to PPO-Linear.

An interesting finding is that ARS did not manage to achieve good performance on the problem. This is because fitness evaluation in ARS requires a large number of samples. In our case, 5,000,000 learning steps may not be sufficient for the algorithm to converge. In the original paper of ARS [146], the average number for solving one particular control task at minimum is 1,000,000 steps, but it searches the optimal policy with 100 CPU cores in parallel. This implies that, in total, it actually requires 100,000,000 samples which are far more what



(a) Dual-RAC vs. RAC

(b) Dual-NACF vs. NACF

(c) Dual-NACA vs. NACA

(d) Dual Algorithms vs. Primal Algorithms

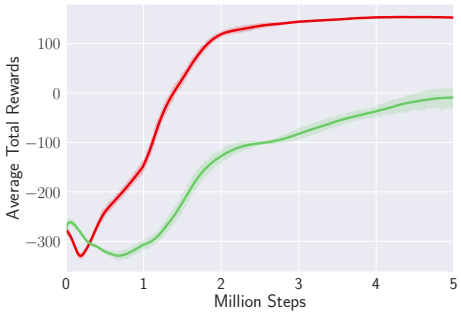
(e) Dual Algorithms vs. Cutting-edge Algorithms

Figure 4.4: A performance comparison of the proposed dual algorithms including Dual-RAC, Dual-NACF, Dual-NACA against the competing algorithms including RAC, NACF, NACA, NACAF, ARS [146] and PPO-Linear [191] on the **Bipedal Walker** problem.

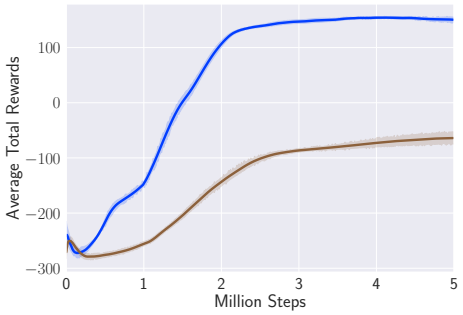
we can support in our experiments.

To sum up, on the Bipedal Walker problem, we can confirm that our dual

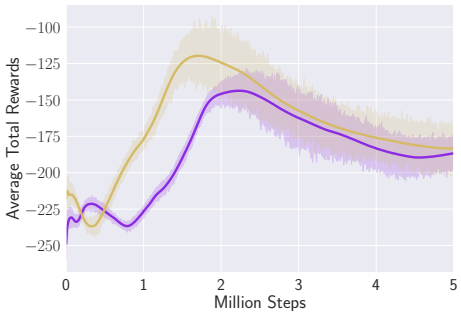
algorithms performed generally better than the competing algorithms in terms of both learning effectiveness and sample efficiency.



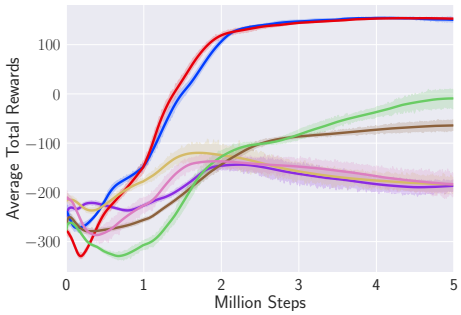
(a) Dual-RAC vs. RAC



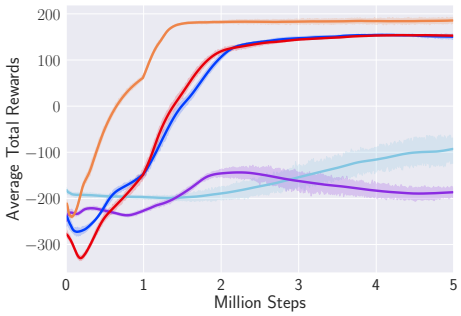
(b) Dual-NACF vs. NACF

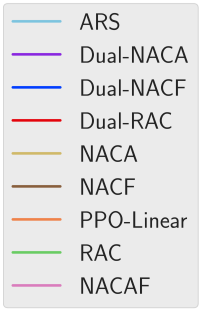


(c) Dual-NACA vs. NACA



(d) Dual Algorithms vs. Primal Algorithms





(e) Dual Algorithms vs. Cutting-edge Algorithms

Figure 4.5: A performance comparison of the proposed dual algorithms including Dual-RAC, Dual-NACF, Dual-NACA against the competing algorithms including RAC, NACF, NACA, NACAF, ARS [146] and PPO-Linear [191] on the **Lunar Lander** problem.

4.6.2 Discussion on Results of Lunar Lander

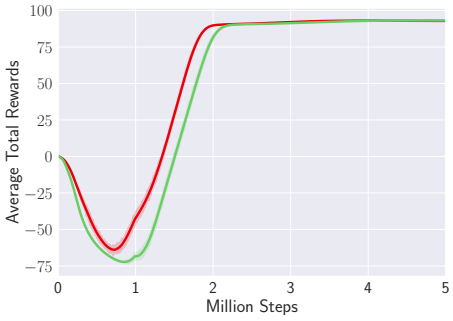
We present the performance comparisons of all algorithms on the Lunar Lander problem in Figure 4.5. In this experiment, we can clearly see significant performance difference between Dual-RAC and RAC in Figure 4.5a as well as between Dual-NACF and NACF in Figure 4.5b. Additionally, the performance advantage of Dual-RAC and Dual-NACF in comparison to all primal counterparts including RAC, NACF, NACA and NACAF can also be witnessed in Figure 4.5d.

Interestingly, a small performance gap can be seen between the dual algorithms (i.e., Dual-RAC and Dual-NACF) and PPO-Linear. In fact, the Lunar Lander problem is a complicated problem where the system performance is highly sensitive to precise control signals. This implies that the changes to the policy cannot be too large otherwise it may generate unexpected behaviors. PPO naturally can keep the policy changes properly bounded by a gradually reducing threshold. However, this ability may prevent PPO-Linear from exploring effectively on other problems such as Mountain Car Continuous and Inverted Double Pendulum.

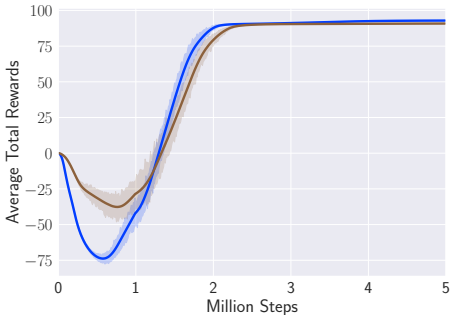
In consequence, we can also conclude the two dual algorithms (i.e., Dual-RAC and Dual-NACF) are highly competitive algorithms on the Lunar Lander problem.

4.6.3 Discussion on Results of Mountain Car Continuous

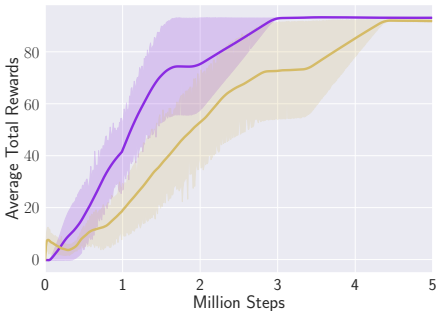
On the Mountain Car Continuous problem, all algorithms except for NACAF have completely solved the problem by reaching average rewards of 90 for 100 consecutive episodes as illustrated in Figure 4.6. In Figure 4.6a- 4.6c, the proposed dual algorithms including Dual-RAC, Dual-NACF, and Dual-NACA can learn much faster than their counterparts. In addition, Figure 4.6e shows that all of our dual algorithms are more sample efficient than PPO-Linear. For example, PPO-Linear requires 1,000,000 more steps to reach the competitive performance in comparison to all our dual algorithms.



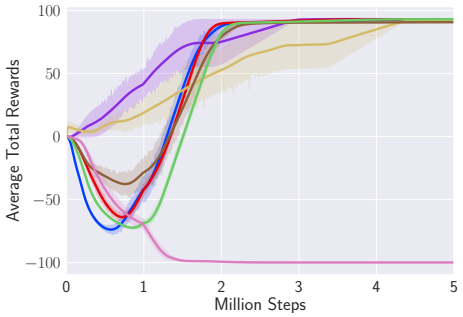
(a) Dual-RAC vs. RAC



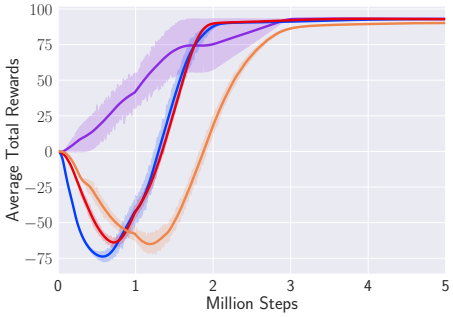
(b) Dual-NACF vs. NACF

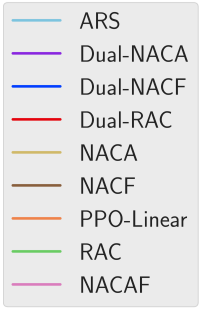


(c) Dual-NACA vs. NACA



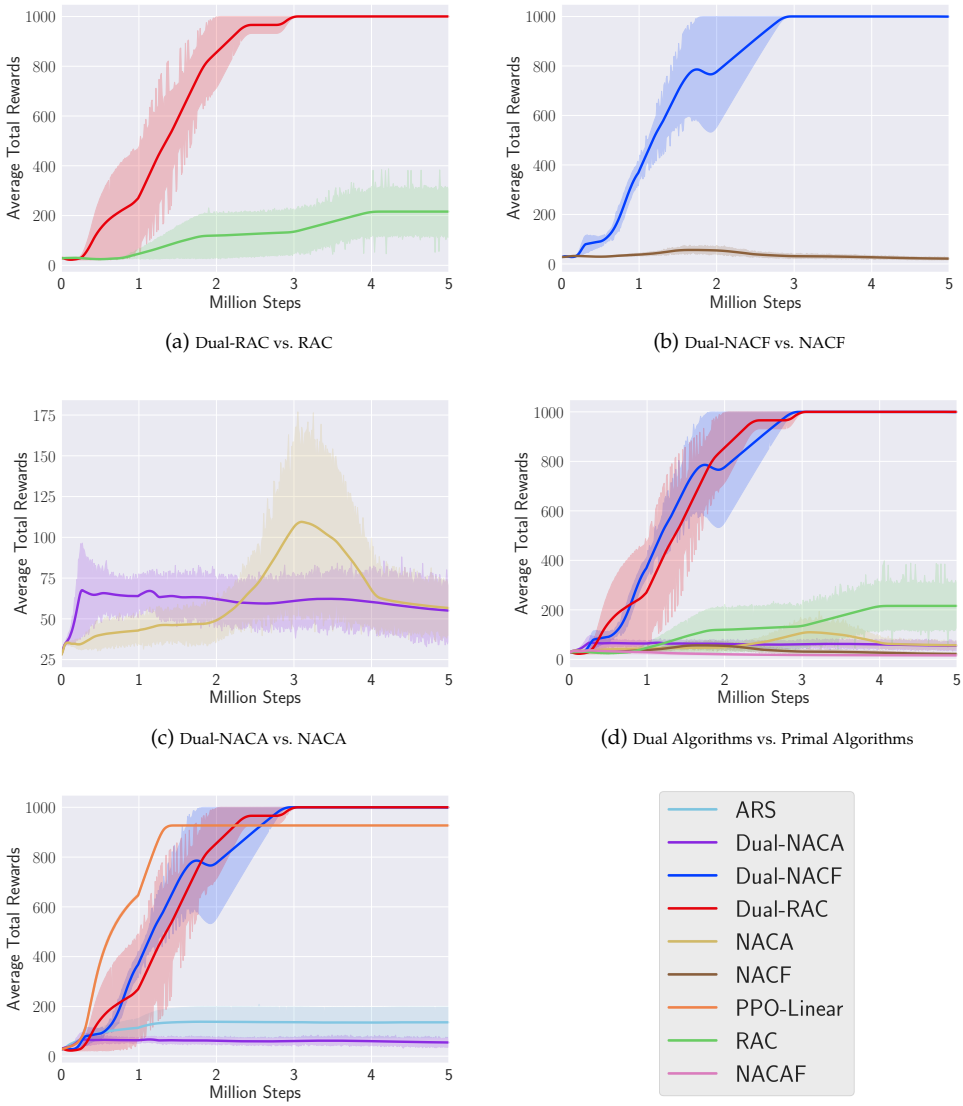
(d) Dual Algorithms vs. Primal Algorithms





(e) Dual Algorithms vs. Cutting-edge Algorithms

Figure 4.6: A performance comparison of the proposed dual algorithms including Dual-RAC, Dual-NACF, Dual-NACA against the competing algorithms including RAC, NACF, NACA, NACAF, ARS [146] and PPO-Linear [191] on the **Mountain Car Continuous** problem.



(a) Dual-RAC vs. RAC

(b) Dual-NACF vs. NACF

(c) Dual-NACA vs. NACA

(d) Dual Algorithms vs. Primal Algorithms

(e) Dual Algorithms vs. Cutting-edge Algorithms

Figure 4.7: A performance comparison of the proposed dual algorithms including Dual-RAC, Dual-NACF, Dual-NACA against the competing algorithms including RAC, NACF, NACA, NACAF, ARS [146] and PPO-Linear [191] on the **Inverted Pendulum** problem.

4.6.4 Discussion on Results of Inverted Pendulum

Similar to the results obtained on the Lunar Lander problem, two of our dual algorithms, Dual-RAC and Dual-NACF are significantly more effective than

RAC, NACF, NACA, NACAF, and ARS on Inverted Pendulum as shown in Figure 4.7. More importantly, as can be seen in Figure 4.7e, both algorithms achieved a significantly higher performance level eventually compared to PPO-Linear although they converge slightly slower than PPO-Linear at the early learning stage.

4.6.5 Discussion on Results of Inverted Double Pendulum

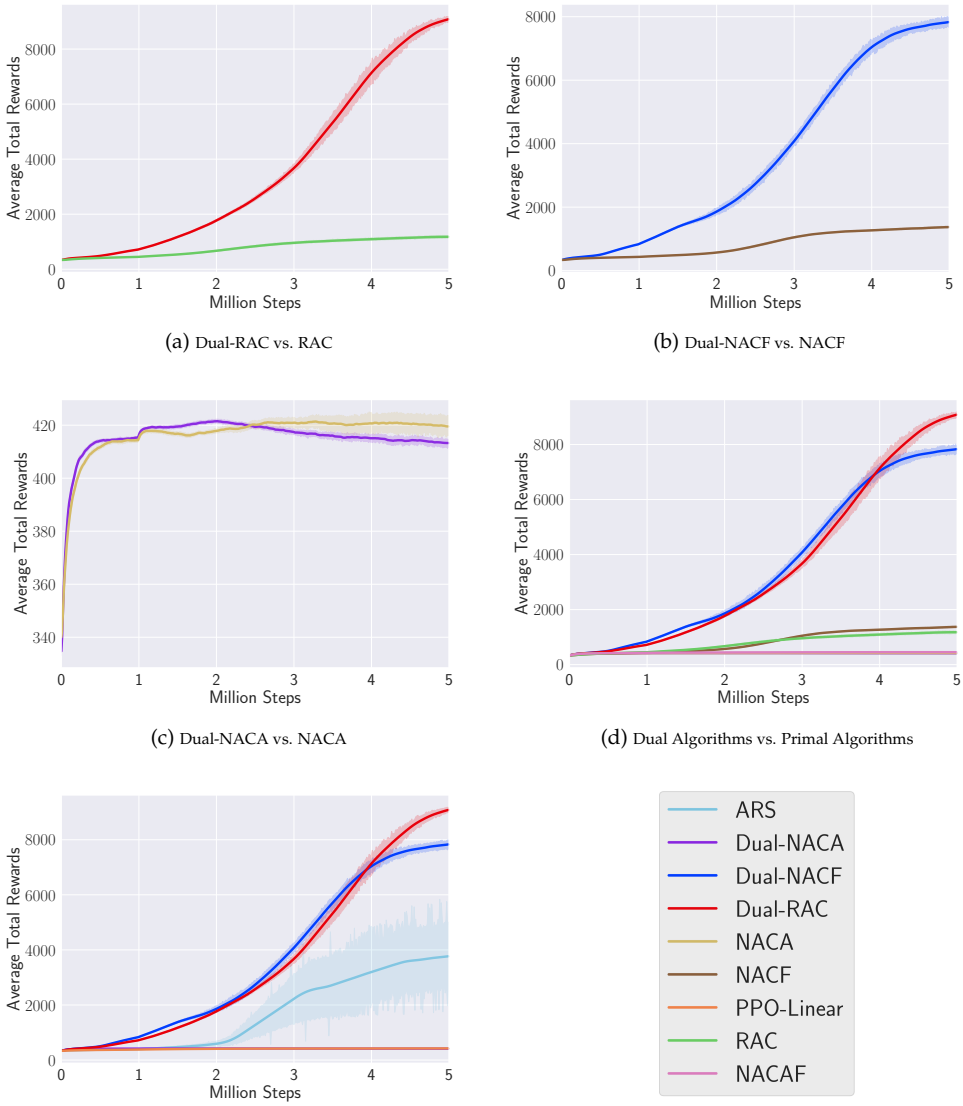
Figure 4.8 generally shows that Dual-RAC and Dual-NACF remain the leading algorithms on Inverted Double Pendulum in terms of effectiveness as well as sample efficiency against all other algorithms such as RAC, NACF, ARS and PPO-Linear.

4.6.6 Discussion on Results of Inverted Pendulum Swingup

As can been witnessed in Figure 4.9, all our proposed dual algorithms, including Dual-RAC, Dual-NACF and Dual-NACA, are much more sample efficient than ARS and PPO-Linear on the **Inverted Pendulum Swingup** problem. Moreover, they also significantly outperform ARS.

4.6.7 Result Summary

By conducting the experiments discussed above, we have successfully achieved the third research objective of this chapter in Section 4.1.1. Specifically, as can be seen in Table 4.2 the experimental results on all benchmark problems show that the proposed dual algorithms, i.e., Dual-RAC, Dual-NACF, Dual-NACA, are generally more effective as well as more sample efficient than their counterparts, i.e., RAC, NACF, and NACA. In particular, two out of three dual algorithms (i.e., Dual-RAC and Dual-NACF) significantly outperform all other algorithms on five out of six benchmark problems including Bipedal Walker, Lunar Lander, Mountain Car Continuous, Inverted Pendulum, and Inverted Double Pendulum. In addition, Dual-NACA algorithm achieved highly competitive



(a) Dual-RAC vs. RAC

(b) Dual-NACF vs. NACF

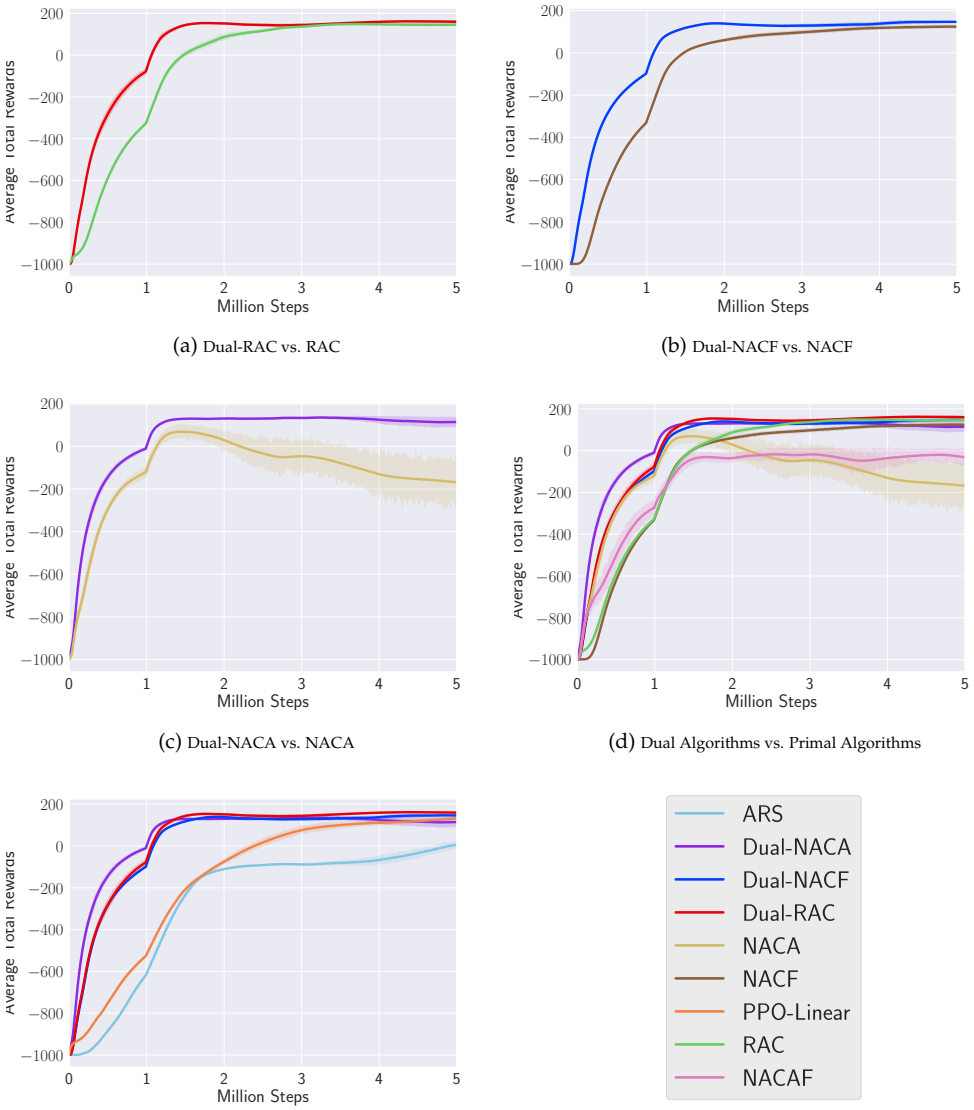
(c) Dual-NACA vs. NACA

(d) Dual Algorithms vs. Primal Algorithms

(e) Dual Algorithms vs. Cutting-edge Algorithms

Figure 4.8: A performance comparison of the proposed dual algorithms including Dual-RAC, Dual-NACF, Dual-NACA against the competing algorithms including RAC, NACF, NACA, NACAF, ARS [146] and PPO-Linear [191] on the **Inverted Double Pendulum** problem.

performances to Dual-RAC and Dual-NACF on Lunar Lander, Mountain Car Continuous, and Inverted Pendulum Swingup. Particularly, on the Inverted



(a) Dual-RAC vs. RAC

(b) Dual-NACF vs. NACF

(c) Dual-NACA vs. NACA

(d) Dual Algorithms vs. Primal Algorithms

(e) Dual Algorithms vs. Cutting-edge Algorithms

Figure 4.9: A performance comparison of the proposed dual algorithms including Dual-RAC, Dual-NACF, Dual-NACA against the competing algorithms including RAC, NACF, NACA, NACAF, ARS [146] and PPO-Linear [191] on the **Inverted Pendulum Swingup** problem.

Pendulum Swingup problem, all three of our dual algorithms also performed competitively to both their counterparts and ARS and PPO Linear.

Table 4.2: The final episode performance comparison of nine algorithms (i.e., ARS, Dual-RAC, Dual-NACF, Dual-NACA, RAC, NACF, NACA, NACAF, and PPO-Linear) on six benchmark problems (i.e., Bipedal Walker, Inverted Double Pendulum, Inverted Pendulum, Inverted Pendulum Swingup, Lunar Lander Continuous, and Mountain Car Continuous).

Algorithms/Problems	Bipedal Walker	Inverted Double Pendulum	Inverted Pendulum	Inverted Pendulum Swingup	Lunar Lander Continuous	Mountain Car Continuous
ARS	-27.78±51.43	4000.07±4017.15	135.63±231.02	19.30±25.84	-77.04±83.59	0.00±0.00
Dual-NACA	**187.08±44.91**	412.03±7.26	53.49±57.52	106.37±51.86	-179.04±28.20	**93.07±0.15**
Dual-NACF	141.59±57.52	7969.90±479.66	998.35±2.33	143.15±12.14	147.93±8.60	92.96±0.22
Dual-RAC	145.13±30.54	**9271.13±131.03**	**1000.00±0.00**	**156.14±7.54**	155.77±4.37	92.79±0.07
NACA	23.31±105.47	420.30±15.76	54.77±49.92	-194.63±324.26	-183.95±45.48	91.95±0.89
NACF	108.75±64.89	1376.44±102.71	20.41±15.60	116.56±8.84	-62.57±20.06	90.84±0.16
PPO-Linear	146.32±28.03	426.51±6.43	931.19±3.56	140.48±29.33	**188.04±20.80**	90.19±0.08
RAC	39.76±63.33	1184.29±116.89	215.41±356.46	149.52±14.31	-16.73±37.79	91.10±0.06
NACAF	-95.04±59.32	450.21±37.82	15.61±1.38	-65.40±146.56	-186.04±80.61	-99.90±0.00

4.7 Chapter Summary

The primary goal of this chapter was to adopt the Primal-Dual Approximation technique to simplify and generalize several typical PGS to improve step-wise learning on linear policies. We have successfully achieved the goal by deriving three dual problems from the primal problems given in the original PGS (i.e., RAC, NACA, and NACF). As a result, we have developed three new step-wise PGS algorithms, i.e., Dual-RAC, Dual-NACA, and Dual-NACF.

More specifically, in Dual-RAC, we have utilized weighted historical gradients to obtain more accurate policy gradient estimations for effective actor learning. Instead of treating all historical gradients equally, Dual-RAC mitigates the effects of gradients which are far away from and less useful for current step updating. For Dual-NACF, by using of Riemannian distance with Fisher Information Matrix on the regularizing components for actor learning, we extend the policy parameter space into more generalized Riemannian space rather than simple Euclidean space. The adaptation of Dual-NACA is to adopt the primal-dual approximation on critic learning rather than to apply it to actor learning. In such a way, we have shown a wide applicable range of our proposed dual generalization.

In addition, we have experimentally shown the superiorities on the effectiveness of applying dual algorithms to most benchmark cases in comparison to primal algorithms as well as other competing algorithms. Meanwhile, the

convergence of the dual algorithms is also theoretically guaranteed.

This work provides possibilities for future exploration. Firstly, in our work, the variance for the policy distribution is predefined and maintained the same throughout the experiments, which can be learned by the algorithms to improve the effectiveness as shown in [43]. Secondly, the algorithms we discussed here are all gradient-based learning, which often suffers from high variances on gradient estimations. Gradient-free methods, as exemplified in ARS, usually exhibit much more stable learning despite its poor sample efficiency. Lastly, this chapter develops the algorithms based on the linear policy which enjoys the benefits like proved convergence and efficient learning, but it may fail in more complex scenarios, such as continuous locomotion tasks with high-dimensional control signals. This can be addressed by generalizing the policy representation in a non-linear manner, for example, adopting Deep Neural Networks, which may also result in more improvements in terms of effectiveness. In view of these situations, in the next chapter, we will further study to develop new PGS algorithms based on an integration of both gradient-free global search and gradient-based local search for effectively finding good policies to tackle difficult RL problems.

Chapter 5

Proximal Evolutionary Strategies for Sample Efficient Policy Direct Search

This chapter presents the second core contribution of the thesis, which aims at addressing the research question Q(2) in Section 1.2 as well as the research objective O(2) in Section 1.3. In particular, we develop a new evolutionary deep policy optimization algorithm on the basis of Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES). The algorithm exploits the proximal (performance lower bound driven) policy optimization techniques [191], which is called Proximal Evolutionary Strategy (PES). PES has three key technical advancements.

1. We develop a layer-wise learning strategy for CMA-ES to improve its computational efficiency for training Deep Neural Networks (DNNs).

2. We establish a proximal performance lower bound based surrogate model for fitness evaluation to reduce the sample cost.

3. We introduce a gradient-based local search technique to significantly improve the performance of evolutionary policy optimization.

Our experiments on ten continuous control problems show that,

- PES with layer-wise training can be more computationally efficient in terms of running time to reach high performance in comparison to other EAs, such as CMA-ES [88], OpenAI-ES [186], Uber-GA [210]);
- PES with a surrogate model can substantially reduce the sample cost;
- Upon using the gradient-based local search technique, PES can achieve better effectiveness in comparison to three state-of-the-art PGS algorithms (i.e., Trust Region Policy Optimization (TRPO) [189], Actor-Critic using Kronecker-Factored Trust Region (ACKTR) [240], Proximal Policy Optimization(PPO) [191]) and two recently developed evolutionary algorithms (i.e., OpenAI-ES [186], Uber-GA [210]).

5.1 Introduction

Building intelligent agents to solve complex tasks in unknown environments effectively is the ultimate goal of reinforcement learning (RL). Recent significant breakthroughs in RL are marked by the engagement with deep learning that fuels the development of a series of prominent deep reinforcement learning (DRL) algorithms, such as Distributed Deterministic Policy Gradient (DDPG) [135], Trust Region Policy Optimization (TRPO) [189], Proximal Policy Optimization (PPO) [191], Actor-Critic with Experience Replay (ACER) [229], OpenAI Evolutionary Strategies (OpenAI-ES) [186] and Uber Genetic Algorithms (Uber-GA) [210]. By using DNNs to represent policies (i.e., policy direct search) directly, these algorithms are capable of directly processing raw state representations to solve complicated tasks without relying on any feature engineering. DRL so far has achieved outstanding success on many difficult problems, especially in the domain of automated control characterized by continuous and high-dimensional action spaces [240, 191, 189, 229, 135].

In general, there are two approaches of training DNNs in PGS algorithms, 1) back-propagation with policy gradients [215, 52], a.k.a., Policy Gradient Search (PGS) and 2) fitness guided evolutionary search, a.k.a, Evolutionary Algorithms (EAs) [186, 210]. PGS applies Stochastic Gradient Descent (SGD) to learn the

weights of policy networks (sometimes value function networks [155, 225, 230]). It conducts a search based on one policy (or one value function) by following the direction of its gradient with respect to some performance measure (e.g., the expected total rewards) in a more exploitative manner. Meanwhile, it requires to carefully select suitable exploration techniques for balancing exploration and exploitation, which is often a tricky part [45]. On the other hand, guided by a general fitness measure, EAs constantly evolve a population of DNNs in the hope of discovering the most suitable DNN for solving any given RL problems. Different from PGS, EAs focus more on exploration and are particularly reliable for tackling complex RL problems whose learning goals have many local optima. Hence they are widely considered as a promising alternative to PGS for effective DRL [186, 210, 45].

5.1.1 Chapter Goals

In this paper, we aim to develop a new EA-based DRL algorithm that can satisfactorily address all the three challenges above. As a result, our new algorithm is expected to achieve high sample efficiency and state-of-the-art performance, in comparison to many advanced DRL algorithms developed in recent years. Our algorithm will also demonstrate the potential of EAs as a strong competitor of PGS for DRL. It particularly encourages the fusion of EAs and PGS in the DRL research community. The development of PES enables us to achieve three main research goals as summarized below.

(G1) To address (C1), we propose to reduce computation efficiency by using the divide-and-conquer strategy. Specifically, while training DNNs, we can decompose the search space based on the layering structures of the DNNs, where each layer is treated as a smaller subspace to be trained separately. Such layer-wise training techniques have already been well investigated in the literature, such as [27]. Different from these existing works, within each layer of a DNN, our algorithm is designed to train a portion of the weights (parameters) in that layer which are selected uniformly at random during the evolutionary process. This is because, in CMA-ES, the dimensionality of a solution should be kept in a reasonable small level to

keep the computation cost in association with the processing of the covariance matrix at a reasonable level and achieve effective and efficient adaptation on the covariance matrix. However, this raises new questions regarding whether or when to retrain the weights in previous learning iterations and how to balance in between training new weights and retraining old weights. To answer these questions, we further develop an epsilon-greedy based weights selection method, following which we can ensure mathematically that a suitable proportion of weights will be retrained across consecutive iterations.

(G2) To tackle (C2), we propose to avoid a direct evaluation of candidate DNNs, instead the performance lower bound of any DNN is estimated through a surrogate model. The use of surrogate model has already been widely recognized as an effective way to reduce search cost for EAs in literature [249]. Inspired by years of development of RL technologies, several different performance lower bounds as surrogate models for evaluating DNNs as evidenced in [111, 189, 111, 191, 117]. In this paper, we are particularly interested in the proximal performance lower bound initially proposed by Kakade in [111] and subsequently improved for training deep neural networks in TRPO [189] and PPO [191]. By optimizing this performance lower bound, PES enjoys the theoretical advantage of consistently improving the learning performance without consuming substantial environment samples. Following the design of PPO, we use gradient descent techniques to train the surrogate model which is further utilized to evaluate the policy networks evolved by CMA-ES. In this way, we expect to significantly improve sample efficiency without noticeable impact on learning effectiveness.

(G3) To solve (C3), we propose to introduce gradient-based local search to enhance learning performance further. As discussed previously, EAs do not require to follow a gradient but conducting extensive exploration via a population of candidates searching in the solution space. However, it requires more computational efforts and often can only reach near optimal (i.e., the optima with less precision) [133]. This can be problematic

in the situations of dealing with RL problems. For instance, those locomotion tasks are very challenging because of their high sensitivity high-dimensional control signals. A small variation along with one degree of the freedom often can cause failure to the system fatally, sometimes such failure is by no means affordable (e.g., a real lunar landing mission). Apparently, it is essential for a controller (i.e., an RL solution) to produce precise actions for these problems. Gradient-based approaches seem to be able to locate such optimal solutions within a local vicinity precisely and quickly, but are often trapped by the first found optima and unable to have further improvements [133]. Therefore it can be seen that neither EAs nor gradient-based search alone is suitable for handling this situation. A better way is to take merits of both sides, namely to combine both EAs and gradient-based local searches to efficiently explore globally meanwhile effectively exploit locally [133, 11]. Motivated by this understanding, we incorporate the gradient-based policy search of PPO as an exploitative local search to locally fine-tune the fitted policy chosen by CMA-ES. By doing so, the learning performance is expected to be further boosted.

Driven by the proposed three goals, we develop a new algorithm named PES for DRL. With the aim for direct policy optimization, PES is particularly suitable for tackling challenging continuous control problems. The effectiveness of PES is further evidenced in promising experimental results obtained on ten benchmark control tasks [220], including Inverted Pendulum, Inverted Double Pendulum, Inverted Pendulum Swingup, Hopper, Walker2D, HalfCheetah, Lunar Lander, BipedalWalker, Bipedal Walker Hardcore and Reacher.

5.1.2 Chapter Organization

The remainder of the chapter is organized as follows. The next section, Section 5.2, describes the new algorithm, i.e., PES, which is followed by design of experiments in Section 5.3. Section 5.4 discusses experimental results with detailed analysis. Finally, the chapter draws conclusions and briefs future work in Section 5.5.

5.2 The Proposed Algorithm — Proximal Evolutionary Strategy

In this section, we propose a new evolutionary deep reinforcement learning algorithm, i.e., Proximal Evolutionary Strategy (PES), for tackling challenging continuous control tasks. In line with the three research goals presented in Section 5.1.1, we design PES progressively across three stages. The first stage aims at supporting layer-wise training of deep neural networks for improved computation efficiency. The resulting algorithm will be named as PES-S1 explained in Section 5.2.1. The second stage targets at sample efficient learning through evaluating evolved deep neural networks based on a proximal surrogate model. An improved algorithm called PES-S2 (see Section 5.2.2) is developed based on PES-S1. Finally, the third stage aims at further boosting learning effectiveness via effective incorporation of PPO's gradient-based local search with PES-S2. The final resulting algorithms is named as PES-S3 [1] to be presented in Section 5.2.3. A complete algorithmic description of PES is presented in Algorithm 5.2.2. The section summarizes the key characteristics of all these improvements against existing works to the end.

5.2.1 PES-S1: Layer-wise Learning

It is challenging to use CMA-ES directly for DRL. To address challenging RL problems, the NN models (i.e., value function network or policy network) in DRL must be sufficiently complex, and hence their parameter spaces are often with huge dimensionality. However, as being discussed previously, a critical issue of CMA-ES is its lower computational efficiency when the search space size becomes more significant. Particularly, CMA-ES requires to maintain the covariance matrix in a reasonable small level, for example, it is suggested that the dimension of the solution should be no more than hundreds [86].

To make CMA-ES effective for DRL, we overcome the limitations with by designing a layer-wise learning process with two steps of improvements.

[1] Note that, PES-S3 is essentially the overall algorithm, i.e., PES. In the following text, we will use the two acronyms interchangeably.

Layer-wise Training

Divide-and-conquer strategy is always a natural choice to tackle complex problems, by which a complicated problem can be decomposed into several simpler sub-problems such that the original complexity can be significantly reduced [180]. Owing to the layered hierarchical structure of DNNs, such decomposition for DNNs can be straightforwardly realized by treating the learning on each layer as a subproblem. Thus for each sub learning problem, we are only required to take care of a portion of weights of the entire network (i.e., the weights of one layer), which significantly reduces the impact of dimensionality in contrast to that of training the NN as a whole.

In this stage, as being illustrated in Figure 5.1, the layer-wise training process is designed to repeat over n_i learning iterations. Inside one iteration, we use CMA-ES to train a DNN one layer at a time, whereas other layers remain fixed. For each layer, CMA-ES evolves all weights as well as biases within the layer for n_g generations. The fitness of an individual can be determined either by averaging the total rewards obtained over n_e full-length simulations [2] (i.e., episodes) or by computing the performance bound from a surrogate model depicted in Section 5.2.2. Until all layers have been updated, the algorithm enters next iteration.

Epsilon-Greedy based Uniform Random Selection

In fact, the direct use of layer-wise learning has already been witnessed in the literature [27], where all weights within one layer are learned at a time for deep belief networks. However, even if layer-wise training to a large extent can reduce the complexity limitation faced by CMA-ES for DRL, the total number of weights in one layer can still be high, unsuitable for simultaneous and direct evolutionary search through CMA-ES. For example, some layer of a DNN with 64×64 hidden neurons can have 4160 parameters (i.e., weights and biases) in

[2] Here, a full-length simulation or an episode means one rollout where an agent interacts with an unknown environment repeatedly until some termination condition is reached. For example, Algorithm 5.2.4 defines a rollout function that conducts n_e full-length simulations.

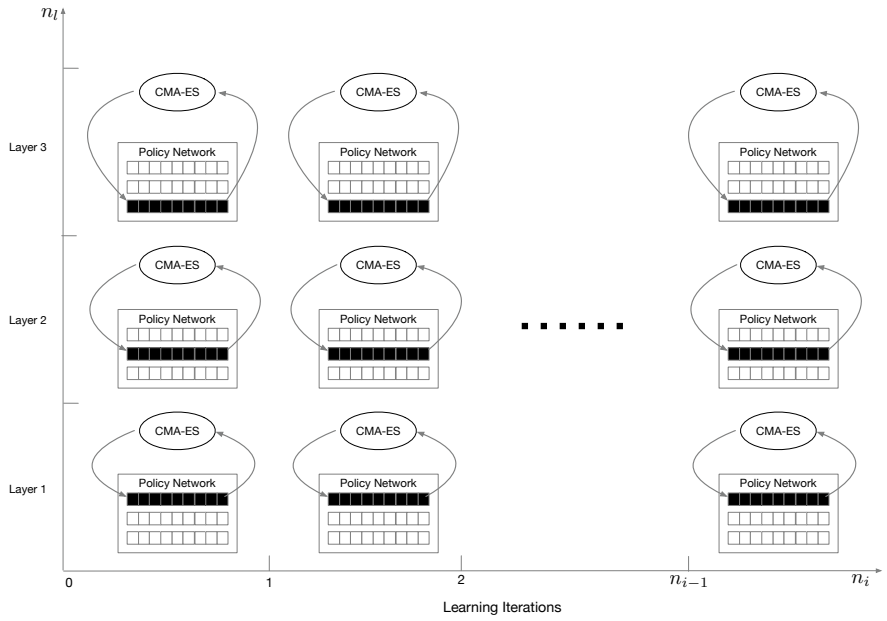


Figure 5.1: The layer-wise training process via CMA-ES for a three-layer policy network, where all weights and biases for one layer are included in the process.

total, which violates the design assumption of CMA-ES and makes the covariance matrix adaptation difficult and time-consuming.

To reduce computation complexity, we propose to select a suitable portion of parameters uniformly at random from one layer to conduct layer-wise learning as shown in Figure 5.2. In the literature, Nesterov [159] applied coordinate descent methods to perform only random updates on partial decision variables for huge-scale optimization problems, and numerical examinations demonstrate the computation efficiency and effectiveness. Unlike Nesterov's work [159] where the portion of parameters is randomly chosen from the entire set of parameters, we only focus on the parameters of one layer for smoothing integrating with layer-wised learning to further reduce the computational complexity.

By doing so, it raises yet a new question about how to balance between training new parameters (i.e., exploration) and re-training old parameters (i.e., exploitation), more specifically, whether or when to re-train parameters already trained in previous iterations. As can be seen in Figure 5.2, for one specific layer at two consecutive learning iterations, the parameters to be trained are

different since they are randomly selected at each iteration. Particularly, some parameters may be trained in one iteration but may be skipped in the next iteration, or even may never be trained again until the end of the entire learning process.

To address the issue, in this stage, we adopt the epsilon-greedy algorithm which is commonly used to balance exploration-exploitation trade-off [212]. Following the epsilon-greedy algorithm, the learning process alters between uniformly random selecting new parameters to train with a probability of ϵ and re-training old parameters already trained in the last iteration with a probability of $1 - \epsilon$ at the beginning of each iteration. Furthermore, it is suggested in [222] that the exploration ought to be encouraged at the early stages of learning whereas the exploitation should play more role with learning proceeding. Thus, ϵ value is expected to gradually reduced. For this purpose, we consider a linear decay schedule in this paper as,

$$\epsilon_t = \max(0, \epsilon_0 - \frac{t}{T}) \tag{5.1}$$

where ϵ_0 denotes the initial value of ϵ, t and T denote the number of current learning steps [3] and the maximumly allowed number of learning steps respectively.

When the epsilon-greedy mechanism chooses to explore a new proportion of weights for training, there are possibilities of which newly selected parameters at one iteration and old parameters from the previous iteration have an overlap. The overlap is important because it helps identify how much expected exploitation can be ensured for two consecutive iterations, which is also a key for balancing the exploration-exploitation trade-off. Suppose that, at the iteration, we have m parameters of one particular layer in total. Given an event $\mathcal{X}$ that there are k among m parameters overlapped between two consecutive iterations when the parameters are randomly re-selected for training, hence the

[3] One learning step indicates one sample interaction between the agent and the environment at one time point.

expectation of $\mathcal{X}$ can be represented as,

$$\begin{aligned} \mathbf{E}[\mathcal{X}] &= \sum_{i=1}^{k} \left[\frac{\binom{m}{i}\binom{m-k}{k-i}}{\binom{m}{k}} i \right] \\ &= \sum_{i=1}^{k} \left[\frac{(k!)^2((m-k)!)^2}{(i-1)!((m-i)!)^2 m!(i-2k+m)!} \right] \\ &= \frac{k^2}{m} \end{aligned} \tag{5.2}$$

Further, let $k = u \times m$ where u is a fixed ratio of parameters to be randomly selected from a layer at every iteration (e.g., $u = 0.5$ means 50% proportion of the layer weights are selected for re-training at random), then we can have $\mathbf{E}[\mathcal{X}] = u^2 \times m$. This expectation theoretically guarantees that there exist at least $u^2 \times m$ samples that can be still exploited between the two consecutive iterations. By adjusting u, we ensure a reasonable level of exploitation to ensure the learning stability without being affected much by too much exploration.

The key steps of PES-S1 are located in the function for initializing the CMA-ES population, accordingly, we present the algorithmic description in Algorithm 5.2.1.

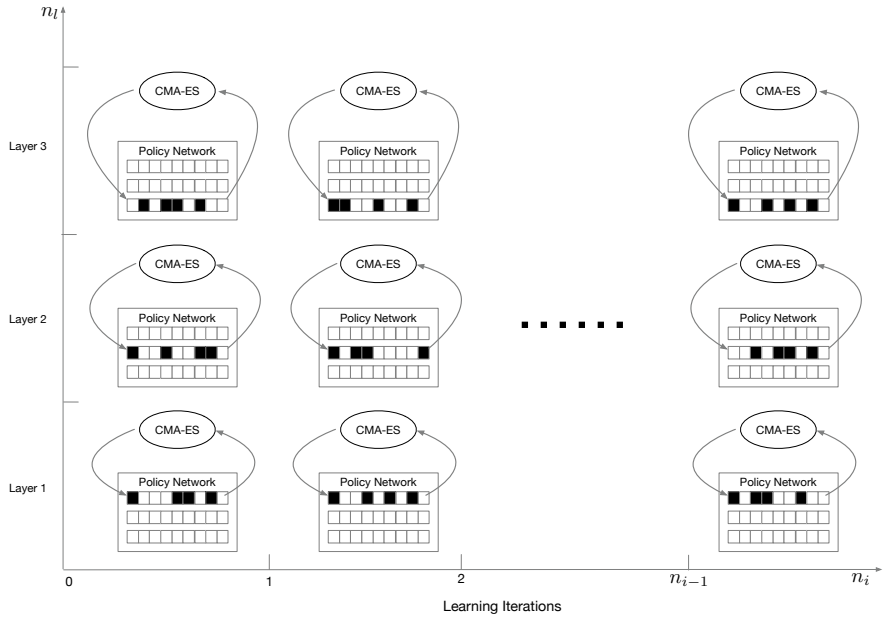


Figure 5.2: The layer-wise training process via CMA-ES for a three-layer policy network, where only a proportion of weights and biases uniformly selected at random are included in the process.

Algorithm 5.2.1 Population Initialization in PES

Require: p: population size, μ: parameters selection ratio, ϵ: exploration rate, $\vec{\Theta}[l]$: parameters of a specific layer l, σ: standard deviation for CMA-ES, $\vec{\tau}$: the indexes of trained parameter proportion from last learning iteration

Ensure: P: a population of partial parameters of the layer

1: **function** `INIT_POPULATION`(p, μ, ϵ, $\vec{\Theta}[l]$)
2: P[] $\leftarrow$ new array of size p
3: **if** `RANDOM`() $< \epsilon$ **then**
4: $\vec{\tau} =$ `UNIFORM_RANDOM_SELECT`$(\mu, \vec{\Theta}[l])$. ▷ Uniformly select μ proportion of parameters from $\vec{\Theta}[l]$ at random and return their indexes
5: **end if**
6: Initiate the covariance matrix $\vec{C_0}$ as an identify matrix
7: **for** $k = 1, 2, ..., p$ **do**
8: $P[k] \leftarrow \vec{\Theta}[l][\vec{\tau}] + \sigma\mathcal{N}(\vec{0}, \vec{C_0})$ ▷ See Section 2.1.3 for more details
9: **end for**
10: **return** P, $\vec{\tau}$
11: **end function**

Complexity Analysis of PES-S1

PES-S1 is designed to largely improve computational complexity for the original CMA-ES when training DNNs with a great number of parameters. To better show the improvement, we provide a theoretical time complexity analysis for PES-S1 in contrast to CMA-ES while they are presumably applied to the same problem. Moreover, the empirical results that are consistent to our theoretical findings are presented in Section 5.4.1.

For simplicity, we only analyze the computational complexity of covariance matrix adaptation. Because this complexity is the dominant factor of computational efficiency of CMA-ES [88, 87, 182], other factors are less critical and omitted, for example, the complexity of sampling a multivariate normally distributed random vector.

To optimize a h-layered NN with n parameters in total, CMA-ES searches a near optimal solution (i.e., a n-dimensional vector of all parameters of the NN)

by adapting a symmetric covariance matrix with $\frac{n^2+n}{2}$ elements [88, 87, 182]. Thus, it requires $\frac{n^2+n}{2}$ flops to complete one adaptation for optimizing the whole network. In contrast, when facing the same problem, PES-S1 optimizes only 50% (if $u = 0.5$) parameters of one layer at a time. Therefore, it holds a much smaller covariance matrix with only $\frac{n^2+hn}{2h^2}$ elements. However, to complete the optimization of the whole NN, it requires to repeat h times. Accordingly, it needs $\frac{n^2+hn}{2h}$ flops to optimize the whole NN.

Given $n >> 0$ and $h > 1$, we can have

$$\frac{n^2+n}{2} - \frac{n^2+hn}{2h} = \frac{(h-1)n^2}{2h} >> 0 \tag{5.3}$$

From (5.3), we can see that PES-S1 uses much less flops to complete one optimization of the entire network in comparison to CMA-ES, if the problem dimension n is large enough. This implies that PES-S1 can be more computationally efficient than CMA-ES.

5.2.2 PES-S2: Surrogate Model Based Learning

The conventional use of CMA-ES for DRL, as demonstrated in PES-S1, is vulnerable to high sample complexity in comparison to classical PGS for DRL. The primary cause is because each candidate neural network in a large population must be separately and independently evaluated based on a large number of direct simulated (or real/physical) interactions with the learning environment. This results in a considerable amount of samples to be consumed simply for evaluating one evolved neural network. Moreover, these samples will be discarded without reusing them for evaluating other evolved neural networks. On the other hand, PGS actually uses every sample it obtained to estimate policy gradients and accumulatively updates the policy. In other words, PGS can reuse the samples more effectively than CMA-ES.

To reduce the sample complexity of CMA-ES for DRL, we decide to adopt a surrogate model based learning process to address the mentioned issue above for improved sample reuse. With the help of the surrogate model for the fitness evaluation, PES (i.e., PES-S2) can achieve the same level of sample efficiency as other cutting-edge PGS algorithms.

In this paper, we develop three key steps to complete the surrogate model based learning process as illustrated in Figure 5.3.

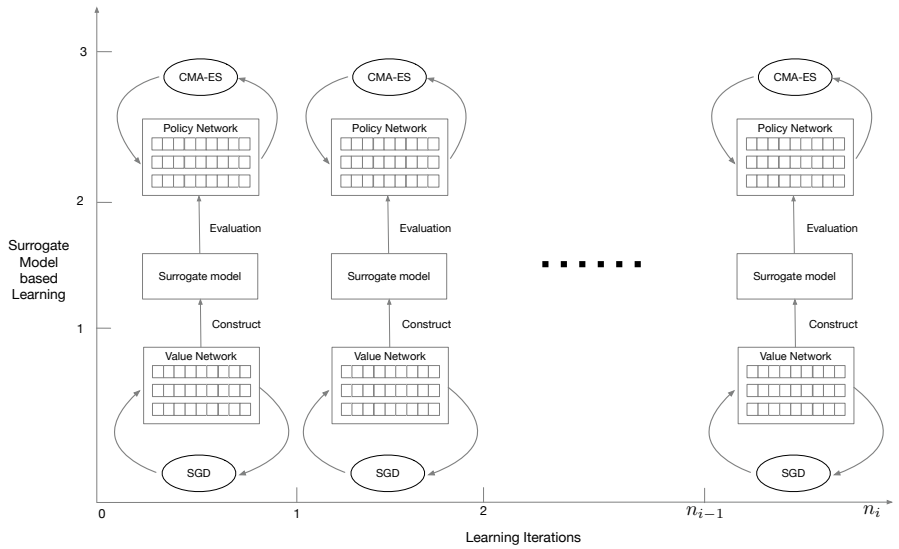


Figure 5.3: The surrogate model based learning process for a three-layer value function network and a three-layer policy network, where value function network is trained via gradient ascent and policy network is layer-wisely trained via CMA-ES (see Section 5.2.1).

Surrogate Model Construction

We propose to construct a surrogate model based on the performance lower bound formulated in (2.56) proposed by PPO [191]. The reason of doing so instead of using any learned value functions (e.g., Q functions) as a surrogate model is because that the learned value function can be so unreliable that it brings adverse affection to the final policy learning [167]. Besides, in the traditional value function based PGS algorithms, the improvement on the performance of policies is not theoretically guaranteed where severe degradations may occur due to large updating steps [47, 117, 189]. As a matter of fact, these issues can be mitigated by optimizing policies with respect to a performance lower bound as suggested in [117, 189], but it is tricky and challenging to handle

the constraints involved in the optimization. In this context, PPO is proposed to optimize a simplified performance lower bound, which achieves state-of-the-art effectiveness across a series of difficult RL tasks. Given this, we expect that this simplified performance lower bound can be an excellent surrogate model for our policy learning.

However, a key technical challenge faced by PES-S2 is how to obtain accurate value function (specifically the advantage function $A^{\pi}(\vec{s}_t, a_t)$) on which the performance lower bound relies. In the original design of PPO, this is achieved by optimizing the objective function in (2.55) from past environment samples. However, the objective function in (2.55) contains two objectives, i.e., the squared loss of value function in (2.57) and the performance lower bound in (2.56). If optimizing both objectives simultaneously as what PPO does, PES-S1 may take extra computational efforts, because both policy and value function networks [4] with apparently more layers have to be evolved. Consequently, in PES-S2, we propose to first learn the value functions by optimizing the squared loss of value function in (2.57) via PPO to obtain accurate value function estimations. Afterward, we use PES-S1 to evolve solely the policy neural network that maximizes the lower bound in (2.56). In doing so, the sample efficiency of value function learning can be guaranteed equivalent to that of PPO, and PES-S1 can also avoid the risk of learning the entire network.

Surrogate Model Training

For PES-S2, we decide to train the value function network through PPO [191]. Original, PPO firstly samples η new interaction steps at the beginning of every learning iteration, and then, based on which it computes the estimations of advantage functions via Generalized Advantage Estimation (GAE) [190]. With the obtained advantage function estimations, PPO learns the value function (i.e., V_{π} function) as well as the policy (i.e., π) at the same time, and then it discards the η samples to proceed to next learning iteration. However, this may not be the most effective way. Some samples may be very informative for guiding the algorithm converge to the optima, but they can be rarely being revisited in the

[4] The network architecture can be seen in Figure 5.6.

later learning due to the stochasticity of the environment and the learning process. In consequence, it is not sufficient to learn them only once because their impacts will decay quickly along with the learning proceeds.

To address the issue of PPO so as to maximumly reuse the samples, we adopt the technique of Experience Replay with Importance Sampling [229, 144, 131]. By doing so, we can firstly maintain a vast repository for past interactions to without losing important information. Secondly, with the Importance Sampling technique, we can further reuse those samples stored in the experience repository which are obtained by previous different policies. With the help of the design, we expect to achieve a higher sample efficiency in PES-S2 in comparison to those advanced EAs such as OpenAI-ES and Uber-GA.

In general, PES-S2 divides the entire neural network evolution process into multiple learning iterations. Each learning iteration contains two learning phases. The first phase is to use PPO to learn value functions for obtaining accurate estimations of advantage functions. The second phase is further divided into multiple evolution generations for PES-S1 to optimize the lower bound for finding good policies (see next subsection). One learning iteration of PES-S2 can be divided into the following four sequential steps.

Step 1 We use $\pi_{\vec{\theta}_{old}}(a_t|\vec{s}_t)$ determined by the best individual obtained from the previous learning iteration to perform simulations for T time steps by using $\pi_{\vec{\theta}_{old}}$ to obtain T experience data points. Each experience data point contains a state transition at one time point t, i.e., $\{\vec{s}_t, a_t, \vec{s}_{t+1}, r_t\}$ and value function predictions of the current time step t as well as next time step $t+1$ computed by $v_t^{evaluate} = V_{\pi_{\vec{\theta}_{old}}}(\vec{s}_t)$ and $v_t^{target} = V_{\pi_{\vec{\theta}_{old}}}(\vec{s}_{t+1})$ respectively. In addition, each instance also contains the probability of choosing a_t at $\vec{s}_t$ following the current policy $\pi_{\vec{\theta}_{old}}$, i.e., $q(\vec{s}, a) = \pi_{\vec{\theta}_{old}}(\vec{s}_t, a_t)$. Afterward, the experience data are augmented to the experience repository Ξ which allows maximumly 50,000 instances.

Step 2 We adopt importance sampling to update the value function prediction of all historical data instances in Ξ. To do so, we first re-compute all probabilities across all state transitions of the historical data instances, i.e., $p(\vec{s}, a) = \pi_{\vec{\theta}_{old}}(\vec{s}, a)$ where $\vec{s}$ and a represent any past state transitions in Ξ.

Following that, we update all target value function predictions (i.e., v_t^{target}) for all past experiences by the following equation,

$$v_t^{target} = \frac{1}{\sum_i^N \frac{p(\vec{s},a)}{q(\vec{s},a)}} \Big[\frac{p(\vec{s},a)}{q(\vec{s},a)} (r_t + V_{\pi_{\vec{\theta}_{old}}}(\vec{s}_{t+1})) \Big] \tag{5.4}$$

Step 3 We learn the value functions (i.e., $V_{\pi_{\vec{\theta}_{old}}}(\vec{s}_t)$) via experience replay with importance sampling. In this step, we conduct n_v training cycles. For each cycle, we firstly uniformly select T data instances from Ξ at random, and then we use the same procedure of PPO [191] to optimize the squared loss of value function in (2.57) for T/b times where M is the given batch size.

Step 4 We adopt the GAE to estimate the advantage functions for constructing the performance lower bound. Based on the T newly obtained samples in the current learning interaction as stated in Step 1 above, we can obtain a sample based advantage function estimation as,

$$\begin{aligned} \hat{A}_t^{\text{GAE}(\gamma,\lambda)} = & \sum_{t=0}^{T} \Big[(\gamma\lambda)^t (r_t + \gamma V^{\pi_{\vec{\theta}_{old}}}(\vec{s}_{t+1}) \\ & - V^{\pi_{\vec{\theta}_{old}}}(\vec{s}_t)) \Big] \end{aligned} \tag{5.5}$$

where $\hat{A}_t^{\text{GAE}(\gamma,\lambda)}$ is the generalized advantage estimator defined in [190] to achieve reliable learning, λ is a hyper-parameter to adjust the balance between bias and variance.

With the help of the advantage estimator $\hat{A}_t^{\text{GAE}(\gamma,\lambda)}$, we obtain the trained surrogate model that can be used for fitness evaluation in next step.

Surrogate Model based Fitness Evaluation

The CMA-ES fitness evaluation is directly conducted based on computing the fitnesses for each individual by the trained surrogate model. This surrogate model based fitness function can be described as,

$$\mathrm{L}_t^{\text{CLIP}}(\vec{\theta}) = \mathbf{E}_t \Big[\min \Big(\nu_t(\vec{\theta}) \hat{A}_t^{\text{GAE}(\gamma,\lambda)}(\vec{s}_t, a_t), \text{clip}\big(\nu_t(\vec{\theta}), 1-\varepsilon, 1+\varepsilon\big) \hat{A}_t^{\text{GAE}(\gamma,\lambda)}(\vec{s}_t, a_t) \Big) \Big], \tag{5.6}$$

For optimizing the policy network, we use PES-S1 developed in Section 5.2.1. In the second phase of one learning iteration, $\pi_{\vec{\theta}_{old}}(a_t|\vec{s}_t)$ remain unchanged within the learning iteration. Each individual generated by CMA-ES is treated as a $\pi_{\vec{\theta}}(a_t|\vec{s}_t)$. Consequently, the fitness value of each individual can be directly computed based on (5.6) with regard to the $\pi_{\vec{\theta}_{old}}(a_t|\vec{s}_t)$ and the $\pi_{\vec{\theta}}(a_t|\vec{s}_t)$. Note that, we follow the same principle of PPO where all T samples generated by $\pi_{\vec{\theta}_{old}}(a_t|\vec{s}_t)$ within the learning iteration are used to compute the fitness. The algorithm of PES-S2 can be converted by changing Line 14 to $P[p].fitness = ROLLOUT(\vec{\Theta}, I, n_e, T, l)$ and removing Line 19 to Line 26. The entire surrogate model based learning process repeats over many iterations until a sufficiently fitted policy can be found. The algorithmic details can be found in Algorithm 5.2.2.

5.2.3 PES-S3: Local Search Enhanced Learning

CMA-ES and PGS are complementary learning techniques for solving difficult RL problems. Specifically, we found that by using PGS as a local search technique to further improve the best neural network evolved by CMA-ES can achieve a desirable balance between exploration and exploitation. Motivated by this, we develop PES-S3 to realize this local search enhanced learning process. In the process, we use PPO as an local search strategy to fine-tune the policy network obtained from the global evolutionary search (i.e., CMA-ES).

However, it is doomed a failure of bluntly augmenting PPO as an additional learning process on the top of PES-S2, as value functions cannot be learned accurately after policy updating by PES-S2. As can be seen in (2.55), PPO optimizes a joint objective function including a term for optimizing value function by starting from the same policy $\pi_{\vec{\theta}_{old}}$. But as designed in PES-S2, after training the value function under $\pi_{\vec{\theta}_{old}}$, CMA-ES is used to learn only policy network resulting in $\pi_{\vec{\theta}_{new}}$ whereas the value function remains the same as under $\pi_{\vec{\theta}_{old}}$. Therefore, such inconsistency can be exaggerated along the learning proceeds, and it can lead to a more and more inaccurate value function if PPO is directly applied on the top of PES-S2.

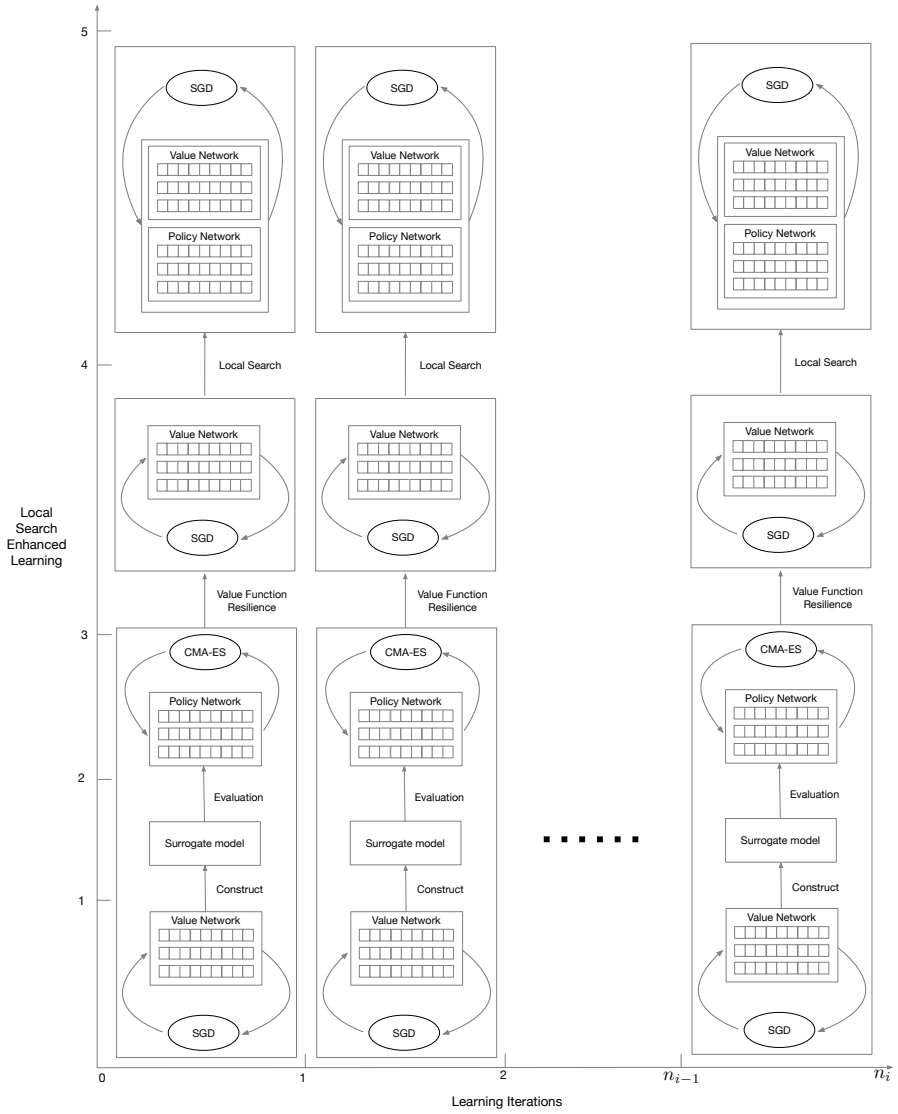


Figure 5.4: The local search enhanced learning process for a three-layer value function network and a three-layer policy network, where a value function resilience re-training process and a PPO driven local search process are added on the top of the surrogate model based learning developed in Section 5.2.2.

To address the issue, as shown in Figure 5.4 we added an extra value function learning phase to achieve smooth the integration between PPO-based local search and PES-S2. The key to the phase is to ensure the value function accurately estimate the value of any state while following the new policy $\pi_{\vec{\theta}_{new}}$. Thus

Algorithm 5.2.2 Proximal Evolutionary Strategy

Require: $\vec{\Omega}$: DNN parameters of value function V_π, $\vec{\Theta}$: DNN parameters of policy π, $\vec{\Theta}'$: DNN parameters of policy π', n_l: number of layers, n_p: population size, σ: step size, n_g: number of generations, n_e: number of episodes, T: the maximumly allowed steps for one iteration, n_i: number of iterations, I^*: the best individual evolved by CMA-ES, Ξ: the experience repository, ξ: a batch of experience data points with size M

Ensure: $\vec{\Theta}$: the best evolved DNN parameters

function PES($\vec{\Theta}$, n_l, n_p, n_g, n_e, n_i, σ)
  for $i = 1, 2, ..., n_i$ **do**
    $\Xi[], \xi \leftarrow$ EXPERIENCE_SAMPLING($\vec{\Omega}$,$\vec{\Theta}$, n_e, T, l) ▷ See Algorithm 5.2.3.
    Update all the past target value function predictions in Ξ according to (5.4)
    Optimize the $\vec{\Omega}$ for K epochs with a batch size $M \leq n_i * T$ according to (2.57)
    Compute advantage function estimations $\hat{A}_1, \ldots, \hat{A}_T$ by (5.5) under $\pi_{\vec{\theta}}$ based on ξ
    for $l = 1, 2, ..., n_l$ **do**
      $P \leftarrow$ INIT_POPULATION($\vec{\Theta}[l]$, n_p) ▷ See Algorithm 5.2.1.
      for $g = 1, 2, ..., n_g$ **do**
        for $p = 1, 2, ..., n_p$ **do**
          $\vec{\Theta}[l] \leftarrow P[p]$
          $P[p].fitness \leftarrow \mathrm{L}_t^{\mathrm{CLIP}}(\vec{\Theta})$ ▷ See (2.56)
        end for
        $P', I^*, \sigma' \leftarrow \mathtt{CMA-ES\_EVOLE}(P, \sigma)$ ▷ See [86].
        $\sigma \leftarrow \sigma'$; $P \leftarrow P'$; $\vec{\Theta}[l] \leftarrow I^*$
      end for
      $\Xi[], \xi \leftarrow$ EXPERIENCE_SAMPLING($\vec{\Omega}$,$\vec{\Theta}$, n_e, T, l) ▷ See Algorithm 5.2.3.
      Compute advantage function estimations $\hat{A}_1, \ldots, \hat{A}_T$ by (5.5) under $\pi_{\vec{\theta}}$ based on ξ
      $\vec{\Theta}_{old} \leftarrow \vec{\Theta}'$
      Optimize $\vec{\Theta}$ for K epochs with a batch size $M \leq n_i * T$ according to (2.55)
    end for
  end for
  return $\vec{\Theta}$ ▷ Return the fittest DNN parameters
end function

after obtaining $\pi_{\vec{\theta}_{new}}$, we re-generate new samples in the environment under $\pi_{\vec{\theta}_{new}}$ and use the gradient ascent to re-train the value function network again to approximate $V^{\pi_{\vec{\theta}_{new}}}$. By doing so, the PPO can be smoothly integrated as a

local search mechanism onto PES-S2 resulting in PES-S3 without experiencing the inconsistent issue mentioned above. The pseudo code of PES-S3 is given in Algorithm 5.2.2.

Algorithm 5.2.3 Experience Sampling

Require: an MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, $\vec{\Omega}$: DNN parameters of value function V_π, $\pi_{\vec{\Theta}}$: DNN parameters of policy π, l: the selected layer No., T: the maximumly allowed steps for one episode, $\vec{\Xi}$: a repository of trajectories

Ensure: $\vec{\Xi}$: the experience repository

1: **function** EXPERIENCE_SAMPLING($\vec{\Omega}$,$\vec{\Theta}$, I, n_e, T, l)
2: $\xi \leftarrow$ ROLLOUT($\vec{\Omega}$,$\vec{\Theta}$, n_e, T, l) ▷ See Algorithm 5.2.4.
3: **if** SIZE(Ξ) $> 50,000$ **then**
4: Delete the oldest M data points from Ξ
5: **end if**
6: $\Xi[] \leftarrow \xi$
7: **return** Ξ, ξ ▷ Return the experience repository and newly sampled experience data of current iteration
8: **end function**

5.2.4 Key Characteristics of Proximal Evolutionary Strategy

By combining the above described three key technical advancements (i.e., PES-S1, PES-S2, and PES-S3), we completed the design of PES for DRL. Here, we summarize distinguishing characteristics of PES in comparison to prominent research works for DRL in recent years.

Use layer-wise learning to enable efficient learning for CMA-ES

Through layer-wise training of DNN (PES-S1), PES can largely reduce the dimensionality of the solution search space, and effectively addresses the computational cost problem of CMA-ES. For example, for training a given DNN with two hidden layers (64×64) containing 4610 parameters (including both weights

Algorithm 5.2.4 Rollout Function

Require: an MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, $\vec{\Theta}$: DNN weights, $\pi_{\vec{\Theta}}$: the policy parameterized by $\vec{\Theta}$, I: an individual of CMA-ES, l: the selected layer No., n_e: number of evaluations, T: the maximumly allowed steps for one episode, $\vec{\Xi}$: a repository of trajectories

Ensure: $\bar{R}$: the averaged total reward of the given individual

function ROLLOUT($\vec{\Theta}$, I, n_e, T, l)
    $\vec{\Theta}[l] \leftarrow I$
    Initial total reward $R \leftarrow 0$
    for $e = 1, 2, ..., n_e$ **do**
        Initial state $\vec{s}_0$
        for $t = 0, 1, ..., T-1$ **do**
            $a_t \sim \pi_{\vec{\Theta}}(\vec{s}_t)$
            Take action a_t, observe reward r_t and new state $\vec{s}_{t+1}$
            $R \leftarrow R + r_t$
        end for
    end for
    $\bar{R} \leftarrow \frac{R}{n_e}$
    return $\bar{R}$, $\vec{\Xi}$ ▷ Return the average reward and a repository of trajectories
end function

and biases), the direct use of CMA-ES yields a covariance matrix with size 4610 $\times$ 4610. To reduce the dimensionality, PES firstly splits the network into three layers where the largest layer has 4096 parameters (i.e., the layer between 64 $\times$ 64 hidden neurons). Next, by setting $u = 0.5$, PES only trains 50% proportion of parameters of each layer, thus for the large layer mentioned above, the covariance matrix only has a size of 2048 $\times$ 2048. Naturally, PES can largely reduce the dimensionality of the solution search space to improve computational efficiency.

Use surrogate model based learning to reduce sample complexity for CMA-ES

The surrogate model-based learning (PES-S2) helps significantly reduce the samples used for evaluations and highly improves the re-usability of samples in CMA-ES. As we only have to train the surrogate model by using T samples [5] in one iteration, the fitness of each individual is computed via the surrogate model without generating new samples. In this way, our PES only requires the same number of samples as PPO, which is much less than any current cutting-edge EAs reported, for example, OpenAi-ES reports achieving 10x as much to TRPO [186].

Use local search enhanced learning to improve learning effectiveness for CMA-ES

PES-S3 uses a PPO-based local search technique to achieve high learning precision, producing a state-of-the-art performance on many challenging RL problems (see Section 5.4.3 for empirical results). This is because of two reasons. Firstly, PES takes advantages of another state-of-the-art PGS, i.e., PPO, by integrating it as a local search mechanism. The integration gives PES an ability to reuse samples to strengthen the exploitation of DRL in contrast to other EAs such as CMA-ES, OpenAI-ES and Uber-GA and guarantees at least an equivalent performance to PPO. Secondly, PES is beneficial for its extra strength of exploration brought by CMA-ES where multiple policies are kept and learned. In comparison to the state-of-the-art PGS algorithm where normally a single policy is learned, PES is less easy to get trapped to local optima.

5.3 Design of Experiments

To empirically evaluate the proposed algorithm, in this section we present the design of experiments. We initiate our discussion from describing ten benchmarked continuous control tasks provided by Bullet Physical Engine [220] and

[5] See Section 5.2.2 for details

GYM benchmark environments [39]. Next, we explain how the experiments are set up with details on competing algorithms, DNN architecture, hyper-parameter configurations, and evaluation criteria. Subsequently, we discuss in details the experiment design in line with our research goals stated in Section 5.1.1.

5.3.1 Experiment Setup

In this subsection, we discuss competing algorithms chosen for our experiments, network architecture, hyper-parameter configurations and evaluation criteria.

Competing Algorithms

In our experiments, the following algorithms are included as competing algorithms: CMA-ES, PES-S1, PES-S2, PES (i.e., PES-S3), OpenAI-ES, Uber-GA, TPRO, PPO, and ACKTR. We select these algorithms according to two criteria: 1) they are either advanced EAs or PGS algorithms for DRL, which are highly suitable for continuous control tasks, and algorithms including Deep Q-Learning Network [155] which are designed for problems with discrete actions such as Atari games will not be considered; 2) they have achieved outstanding and reproducible performance on many difficult continuous control tasks. To ensure good performance for all competing algorithms, we rely on high-quality algorithm implementations provided by OpenAI baselines [6] [56].

5.3.2 Network Architecture

For fair comparisons, we consistently use the same network topologies for all algorithms. DNNs trained by EAs contain only policy networks, we hence only make them topologically identical to the policy networks of the DNNs trained by PGS. The topology used by PES-S1, OpenAI-ES, and Uber-GA is presented in Figure 5.5, whereas the other used for CMA-ES-SM, PES-S2, PPO, TRPO, and

[6] Our implementation of all algorithms can be found at https://github.com/yimingpeng/cmaes_baselines

ACKTR is given in Figure 5.6. As can be seen from Figure 5.6, the value function network and the policy network shares the same input layer, which follows the exact configurations in [191]. Note that, in PES-S2 and PES, only the policy network will be evolved by CMA-ES, and the value function network is solely trained by PPO.

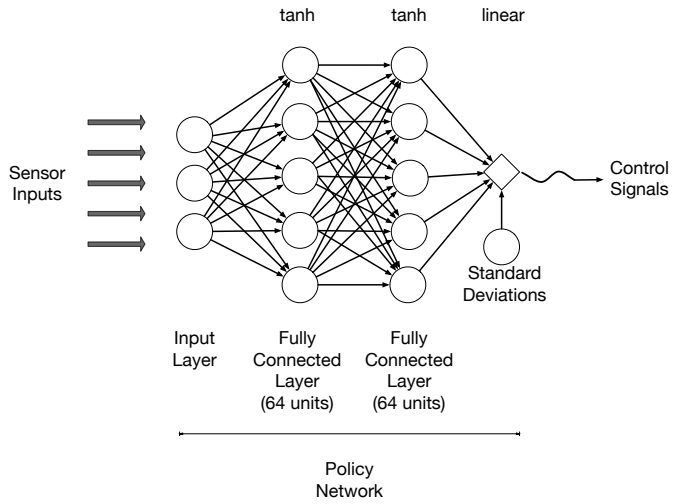


Figure 5.5: The Architecture of DNN for CMA-ES, PES-S1, OpenAI-ES, and Uber-GA.

5.3.3 Hyper-Parameter Configurations

All hyper-parameter configurations are adopted by following the best settings for PPO, TRPO, and ACKTR given in baseline implementation [56]. For CMA-ES training DNNs, including PES-S1, PES-S2, and PES, we use the setting in consistency with [59]. Additionally, we make the local search component follow the same setting to PPO to ensure a fair competition.

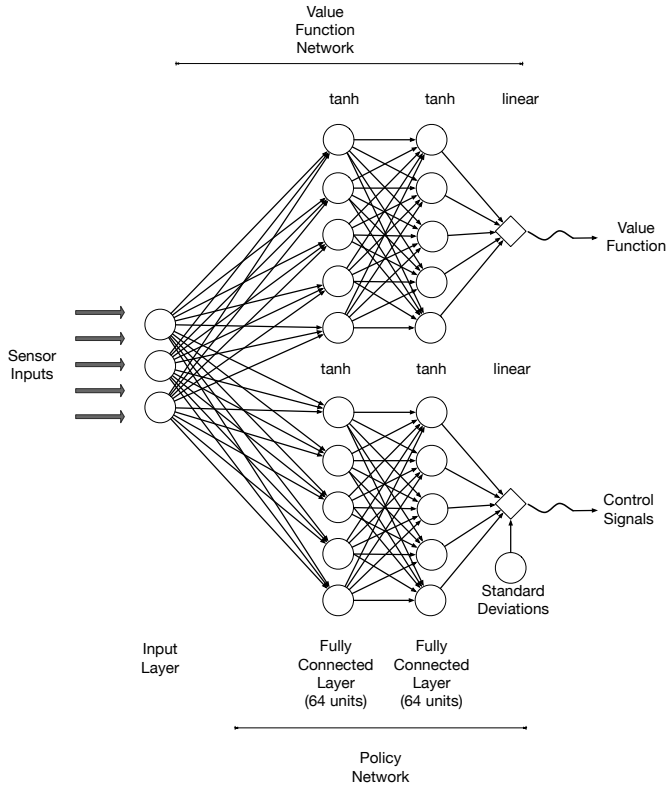


Figure 5.6: The Architecture of DNN for PES-S2, PES, PPO, TRPO, and ACKTR.

Table 5.1: Hyper-parameter configurations of all candidate algorithms: CMA-ES, PES-S1, PES-S2, PES, OpenAI-ES, Uber-GA, TPRO, PPO, and ACKTR.

	EA						**PGS**		
Hyper-parameters	**CMA-ES**	**PES-S1**	**PES-S2**	**PES**	**OpenAI-ES**	**Uber-GA**	**PPO**	**TRPO**	**ACKTR**
Maximum number of samples	5×10^6	5×10^6	5×10^6	5×10^6	5×10^6	5×10^6	5×10^6	5×10^6	5×10^6
Iterations (n_i)	2000	2000	2000	2000	2000	2000	2000	2000	2000
Internal Generations (n_g)	-	10	10	10	-	-	-	-	-
Population Size (n_p)	32	32	32	32	32	32+1	1	1	1
Number of Evaluations (n_e)	3	3	1	1	3	3	1	1	1
Horizon (T)	-	-	2048	2048	-	-	2048	2048	2500
Adam Stepsize	-	-	3×10^{-4}	3×10^{-4}	-	-	3×10^{-4}	3×10^{-4}	2.5×10^{-4}
Num. of Epoch (gradient learning)	-	-	10	10	-	-	10	10	10
Mni-batch Size	-	-	64	64	-	-	64	64	64
Discount Factor (γ)	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
GAE Factor (λ)	-	-	0.95	0.95	-	-	0.95	0.95	-
Variance (σ)	0.001	0.001	0.001	0.001	0.001	0.001	-	-	-
Initial Exploration Rate (ϵ)	-	0.5	0.5	0.5	-	-	-	-	-
Stagnation (sd)	3	3	3	3	3	3	-	-	-
Clipping Factor (ε)	-	-	0.2	0.2	-	-	0.2	-	-

5.3.4 Evaluation Criteria

In the experiments, we aim to assess our algorithms from three aspects, i.e., effectiveness, sample efficiency, and time efficiency. Following the standard setting in the literature [191, 56, 189], effectiveness is defined as the average total rewards of the last 100 episodes [7]. Sample efficiency is interpreted as the number of samples used for achieving similar effectiveness. Time efficiency is measured by the training time (i.e., the average running time for training on 10,000 samples) of each algorithm for the whole training/learning process.

5.3.5 Experiment Design

Following our research objectives, we expect to answer three specific questions through the empirical study. These questions are: (C1) can CMA-ES with layer-wise training can also be a viable algorithm for evolving DNN in terms of effectiveness and computational efficiency? (C2) can the surrogate model help CMA-ES to achieve competitive performance as CMA-ES that evaluates the fitness of each evolved deep neural network through simulations and in the meantime achieve significantly higher sample efficiency? (C3) can the final design of PES supported by PPO-based local search technique perform effectively in comparison to four cutting-edge DRL algorithms (i.e., PPO, TRPO, ACKTR) and two modern EAs based RL algorithms (i.e., OpenAI-ES, Uber-GA)?

To answer these questions, we design three independent experiments following identical settings are given in Section 5.3.1. Each experiment helps answer one of the three questions above. In the first experiment (A1), we compare the effectiveness and time efficiency of six different algorithms, CMA-ES, PES-S1, OpenAI-ES, Uber-GA on the given ten problems. In the next experiment (A2), we focus on the learning performance as well as the sample efficiency obtained by PES-S2 against the algorithms using actual simulations, such as CMA-ES, PES-S2, OpenAI-ES, and Uber-GA. In the last experiment (A3), we assess the

[7] One episode indicates a sequence of interactions (i.e., state transitions) between an agent and an environment, which ends with some terminal conditions. For example, in the Cart Pole problem, one episode starts when the agent balances the pole and terminates when the poles falls.

effectiveness, sample efficiency, time efficiency of PES which combines all improvements (i.e., layer-wise learning, surrogate model-based learning, and local search enhanced learning) in comparison to PPO, TRPO, ACKTR, OpenAI-ES, and Uber-GA.

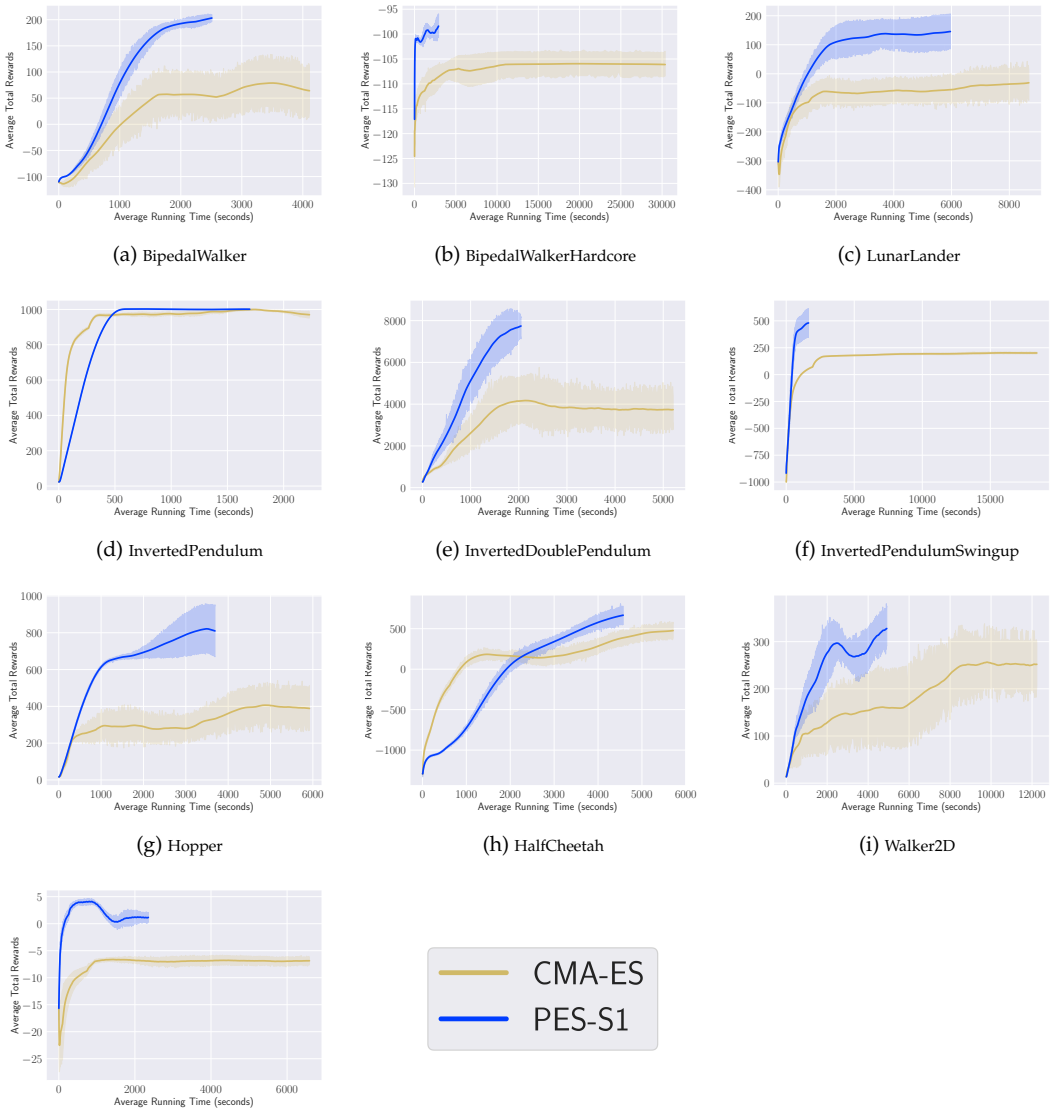
5.4 Results and Discussion

In this section, we discuss the experimental results and provide some insightful analysis. Firstly, the results of PES-S1 obtained from Experiment (A1) are compared to CMA-ES with focus on time efficiency as well as learning performance. Next, the average total rewards obtained from Experiment (A2) by different algorithms (including PES-S2, CMA-ES, OpenAI-ES, and Uber-GA) are depicted as performance curves with respect to the number of samples. Lastly, the final learning performances of our PES algorithm in comparison to PPO, TRPO, ACKTR, OpenAI-ES and Uber-GA are presented and analyzed.

5.4.1 Results of Experiment (A1)

In this experiment (A1), we are interested in comparing the computational efficiency between PES-S1 and CMA-ES. Figure 5.7 presents a learning curve measured in the average total rewards along the y-axis and the total running time along the x-axis. The running time is tracked based on training time required for processing every 10,000 samples by each algorithm.

As can be seen from Figure 5.7 and Table 5.2, PES-S1 clearly uses much less running time to reach higher total rewards in comparison to CMA-ES across all problems. In particular, PES-S1 performs significantly better than CMA-ES on seven out of ten problems, including Bipedal Walker, Bipedal Walker Hardcore, Lunar Lander, Inverted Double Pendulum, Hopper, Walker2D and Reacher. On other three problems, i.e., Inverted Pendulum, Inverted Pendulum Swingup, and HalfCheetah, PES-S1 rises slowly at the begging, but eventually surpasses CMA-ES after a short while. For example, PES-S1 starts to outperform CMA-ES



(a) BipedalWalker (b) BipedalWalkerHardcore (c) LunarLander

(d) InvertedPendulum (e) InvertedDoublePendulum (f) InvertedPendulumSwingup

(g) Hopper (h) HalfCheetah (i) Walker2D

(j) Reacher

Figure 5.7: A comparison of average total rewards over running time (in seconds) obtained by PES-S1 and CMA-ES [88] on the ten benchmark control problems.

after the first 2201 seconds of training time.

These observations are consistent with the results of our complexity analysis presented in Section 5.2.1. In line with the findings, we can conclude that our

Table 5.2: The final episode performance comparison of two algorithms (i.e., CMA-ES and PES-S1) on ten benchmark problems (i.e., Bipedal Walker, BipedalWalkerHardcore, HalfCheetah, Hopper, Inverted Double Pendulum, Inverted Pendulum, Inverted Pendulum Swingup, Lunar Lander Continuous, Reacher, and Walker2D).

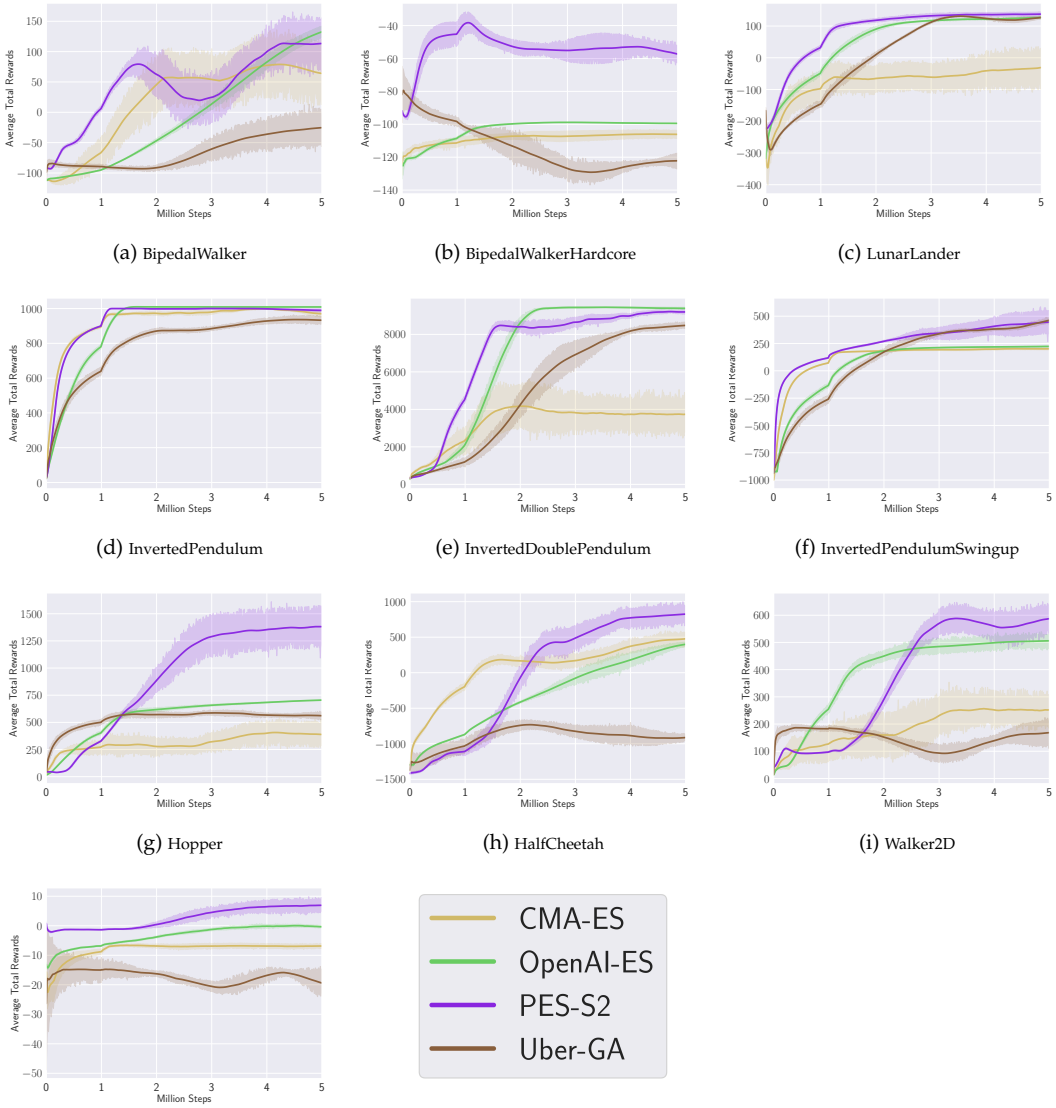
Problems/Algorithms	CMA-ES	PES-S1
BipedalWalker	62.26±98.48	**209.88±11.24**
BipedalWalkerHardcore	-106.53±5.32	**-99.87±2.67**
HalfCheetah	539.62±229.70	**714.88±289.76**
Hopper	377.19±259.54	**752.02±353.75**
InvertedDoublePendulum	3816.13±2120.07	**8118.64±532.40**
InvertedPendulum	960.76±71.37	**998.31±3.77**
InvertedPendulumSwingup	198.88±28.45	**482.85±226.83**
LunarLanderContinuous	-13.85±123.27	**153.86±108.56**
Reacher	-7.35±2.10	**2.41±1.49**
Walker2D	264.58±101.24	**318.56±139.05**

proposed layer-wise learning mechanism (i.e., PES-S1) can significantly reduce the computational cost of the original CMA-ES algorithm when being applied to training DNNs without sacrificing learning effectiveness.

5.4.2 Results of Experiment (A2)

The experiment (A2) aims to examine the sample efficiency of PES-S2 against CMA-ES, and two advanced EAs, i.e., OpenAI-ES and Uber-GA. The learning curves of averaging total rewards are presented in Figure 5.8.

As evidenced in Figure 5.8, PES-S2 is generally more effective and more sample efficient than those competing algorithms on all ten problems. More importantly, PES-S2 performed significantly better than all other algorithms on five out of ten problems, including Bipedal Walker Hardcore, Hopper, HalfCheetah, and Reacher. Regarding the other five problems, i.e., Bipedal Walker, Inverted



(a) BipedalWalker (b) BipedalWalkerHardcore (c) LunarLander

(d) InvertedPendulum (e) InvertedDoublePendulum (f) InvertedPendulumSwingup

(g) Hopper (h) HalfCheetah (i) Walker2D

(j) Reacher

Figure 5.8: A comparison of average total rewards per 10,000 samples (5,000,000 samples in total) obtained by PES-S2, CMA-ES [88], OpenAI-ES [186], and Uber-GA [210] on the ten benchmark control problems.

Pendulum, Inverted Double Pendulum, Inverted Pendulum Swingup, PES-S2 also clearly outperform its base algorithm CMA-ES. In addition, PES-S2 is more sample efficient than OpenAI-ES and Uber-GA, as it can reach higher rewards

Table 5.3: The final episode performance comparison of four algorithms (i.e., CMA-ES, OpenAI-ES, Uber-GA and PES-S2) on ten benchmark problems (i.e., Bipedal Walker, BipedalWalkerHardcore, HalfCheetah, Hopper, Inverted Double Pendulum, Inverted Pendulum, Inverted Pendulum Swingup, Lunar Lander Continuous, Reacher, and Walker2D).

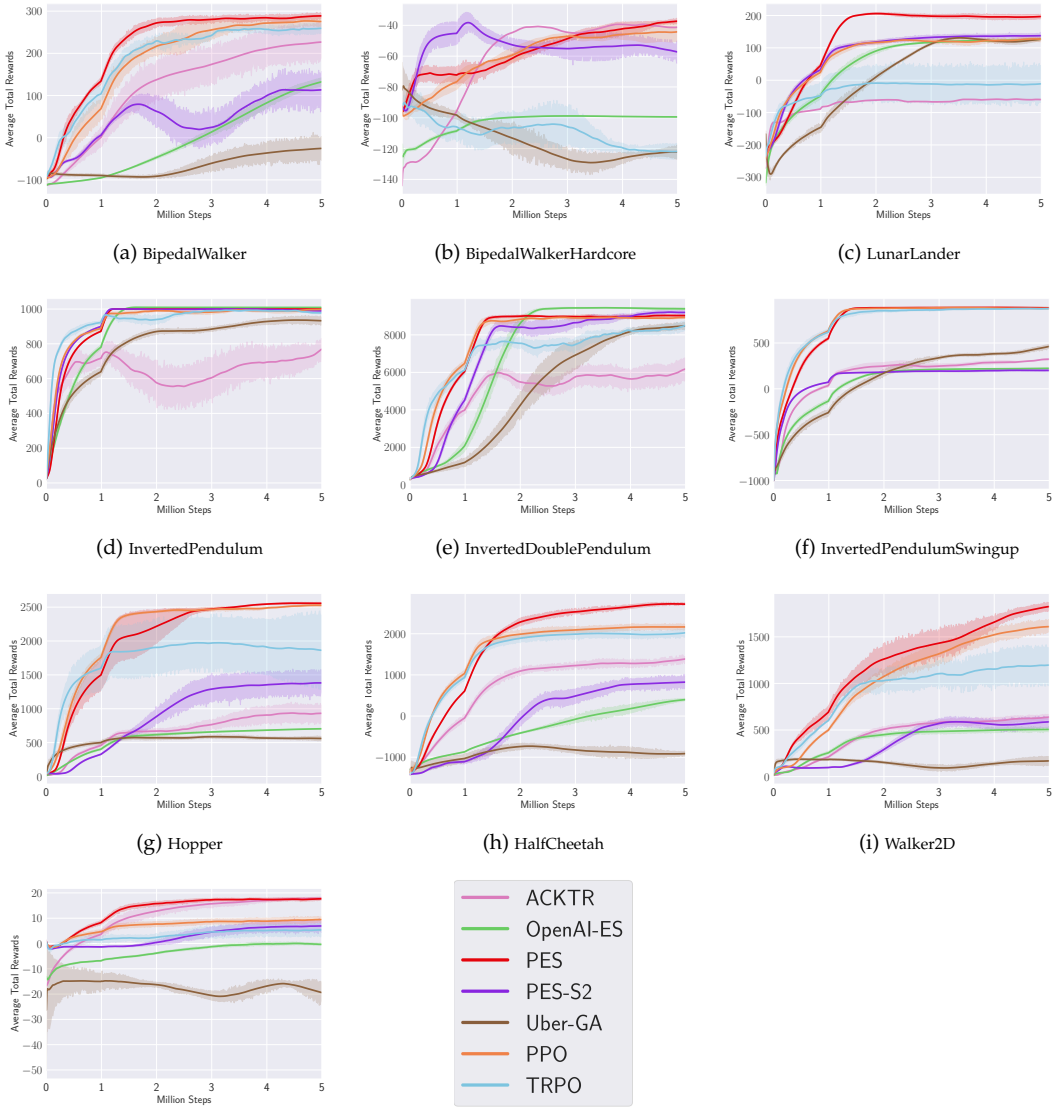
Problems/Algorithms	CMA-ES	OpenAI-ES	PES-S2	Uber-GA
BipedalWalker	62.26±98.48	**146.71±23.52**	137.01±92.01	-21.46±69.31
BipedalWalkerHardcore	-106.53±5.32	-99.82±0.63	**-60.35±15.18**	-121.11±17.47
HalfCheetah	539.62±229.70	434.18±29.93	**856.38±298.15**	-878.93±179.98
InvertedDoublePendulum	3816.13±2120.07	**9399.54±111.06**	9163.02±282.70	8631.40±314.94
InvertedPendulum	960.76±71.37	**1009.12±0.00**	985.39±46.20	924.23±49.29
InvertedPendulumSwingup	198.88±28.45	226.22±8.32	456.65±241.24	**542.75±81.24**
LunarLanderContinuous	-13.85±123.27	128.15±5.97	**140.22±16.49**	137.83±18.25
Reacher	-7.35±2.10	-0.51±1.13	**7.58±5.44**	-21.85±13.85
Walker2D	264.58±101.24	511.33±56.64	**595.94±139.11**	177.34±108.50
Hopper	377.19±259.54	712.25±6.38	**1389.84±369.33**	543.82±131.80

earlier with fewer samples, as can be seen on the results of Lunar Lander, Inverted Pendulum, Inverted Pendulum Swingup.

Interestingly, on Bipedal Walker Hardcore, we also find that PES-S2 starts quickly to reach a high reward of -20, but afterwards, it experiences a sudden drop and performance decrease onwards. It is possibly because of the hyper-parameter configurations, specifically the step size for the algorithm which requires further tunning. Similar decreasing behaviors can also be found with Uber-GA on Bipedal Walker Hardcore, HalfCheetah, Walker2D, and Reacher. This is mainly because of the population size 32 for Uber-GA may be not enough. As reported in the original paper of Uber-GA [210], they have chosen 12,501 individuals to form a population, which extremely is very computational costly and cannot be reproduced in our experiments with the computation facilities available to us.

With the above findings, we can confidently confirm that CMA-ES enhanced by surrogate model base learning, i.e., PES-S2, is more sample efficient than CMA-ES, OpenAI-ES, and Uber-GA.

5.4.3 Results of Experiment (A3)



(a) BipedalWalker (b) BipedalWalkerHardcore (c) LunarLander

(d) InvertedPendulum (e) InvertedDoublePendulum (f) InvertedPendulumSwingup

(g) Hopper (h) HalfCheetah (i) Walker2D

(j) Reacher

Figure 5.9: A comparison of average total rewards per 10,000 samples obtained by PES (i.e., PES-S3), PES-S2, OpenAI-ES [186], Uber-GA [210], TRPO [189], ACKTR [240] and PPO [191] on ten control problems over total 5,000,000 samples.

Table 5.4: The final episode performance comparison of seven algorithms (i.e., PES, PES-S2, OpenAI-ES, Uber-GA, PPO, TRPO, and ACKTR) on ten benchmark problems (i.e., Bipedal Walker, BipedalWalkerHardcore, HalfCheetah, Hopper, Inverted Double Pendulum, Inverted Pendulum, Inverted Pendulum Swingup, Lunar Lander Continuous, Reacher, and Walker2D).

Problems/Algorithms	ACKTR	OpenAI-ES	PES	PES-S2	Uber-GA	PPO	TRPO
BipedalWalker	229.26±99.74	146.71±23.52	**283.87±19.19**	137.01±92.01	-21.46±69.31	273.29±25.27	277.75±7.91
BipedalWalkerHardcore	-40.07±0.00	-99.82±0.63	**-37.05±3.35**	-60.35±15.18	-121.11±17.47	-43.28±4.67	-119.89±8.90
HalfCheetah	1453.33±151.07	434.18±29.93	**2675.90±46.58**	856.38±298.15	-878.93±179.98	2154.35±140.12	2067.42±194.73
InvertedDoublePendulum	6690.00±1956.89	**9399.54±111.06**	8983.04±332.35	9163.02±282.70	8631.40±314.94	8850.75±210.50	8892.38±328.20
InvertedPendulum	926.88±86.35	**1009.12±0.00**	1000.00±0.00	985.39±46.20	924.23±49.29	1000.00±0.00	993.35±8.50
InvertedPendulumSwingup	332.00±51.00	226.22±8.32	**881.85±13.92**	198.88±28.45	542.75±81.24	871.53±19.49	878.27±7.35
LunarLanderContinuous	-57.45±0.00	128.15±5.97	**199.42±11.53**	140.22±16.49	137.83±18.25	129.05±16.62	-12.60±118.56
Reacher	18.23±1.31	-0.51±1.13	**18.53±0.53**	7.58±5.44	-21.85±13.85	10.04±2.83	6.03±4.84
Walker2D	652.20±150.58	511.33±56.64	**1827.97±107.22**	595.94±139.11	177.34±108.50	1603.44±215.07	1101.82±407.30
Hopper	903.50±395.06	712.25±6.38	**2559.39±19.92**	1389.84±369.33	543.82±131.80	2503.88±68.37	1713.87±917.22

The experiment (A3) is conducted with the aim of evaluating the learning effectiveness of PES (i.e., PES-S3, the overall algorithm) in comparison to state-of-the-art algorithms including ACKTR, TRPO, PPO, OpenAI-ES and Uber-GA on the ten benchmarks. Note that, we also include PES-S2 in the comparison to highlight the performance improvement achieved by PES through adopting an extra PPO based local search technique. The learning curves of average total rewards obtained by the six algorithms are reported in Figure 5.9.

Overall, we can clearly see from Figure 5.9 that PES achieved significantly higher performance compared to all the other competing algorithms in terms of both effectiveness and sample efficiency on three problems, i.e., Lunar Lander, HalfCheetah, and Walker2D. Additionally, on Bipedal Walker, PES also outperforms other algorithms, except TRPO and PPO that produced competitive performance. Similar observations can also be found on InvertedPendulum, Inverted Double Pendulum, Inverted Pendulum Swingup, Hopper and Reacher where PES achieves better or the same performance in comparison to the cutting-edge algorithms such as ACKTR, TRPO, and PPO. Furthermore, it is clear to see that the performance of PES-S2 falls far below that of PES on eight out of ten problems. Even on the other two problems, Inverted Pendulum, and Inverted Double Pendulum, PES-S2 shows slower convergence speed than PES.

Besides, we can also see how PES can better balance exploration and exploitation to improve learning effectiveness. For example, compared to the be-

havior of PPO on Hopper, PES exhibits a higher variance (i.e., a large shade area) for learning the first 1,800,000 samples, but its performance surpasses the PPO after learning 2,800,000 samples meanwhile the variance (i.e., the shaded area) becomes smaller and smaller. This is because that the CMA-ES in PES encourages more exploration at the beginning of learning which may fall into some poor-performing regions. In such regions, the learning may vary considerably. However, as the learning proceeds, it gradually converges to high-quality policies, and the learning becomes stable. Meanwhile, during the learning, the local search through PPO helps PES fine-tune (i.e., exploit) the best policy evolved by CMA-ES to find better policies in the vicinity to achieve better learning effectiveness. The same observations can also be found on the problems of Half Cheetah and Walker2D. These all indicate that PES has a better exploration-exploitation trade-off.

Interestingly, we also find that the Bipedal Walker Hardcore is difficult for all algorithms. It seems that PES performs slightly better than other algorithms. Also, it seems that, with more training samples made available, PES, PPO, and ACKTR can possibly achieve better performance as the corresponding performance curves continue to rise till 5,000,000 samples. This implies that Bipedal Walker Hardcore is more difficult than other benchmark control problems.

Based on these observations and understandings, we can draw three conclusions. First, PES achieves competitive and frequently better performances in terms of learning effectiveness as well as sample efficiency in comparison to the state-of-the-art RL algorithms, i.e., ACKTR, TRPO, PPO, OpenAI-ES and Uber-GA. Second, PES can effectively balance the exploration-exploitation trade-off than all the competing algorithms, especially EAs for DRL. Third, PES performs better than PES-S2, which shows that PPO-based local search can significantly enhance the CMA-ES based global search with regard to the learning effectiveness and sample efficiency.

5.5 Chapter Summary

The goal of the chapter was to develop a new evolutionary deep reinforcement learning algorithm, i.e., PES, based on CMA-ES for solving difficult continuous control tasks, which is expected to achieve the state-of-the-art performance in terms of time efficiency, sample complexity, and learning effectiveness. To achieve this goal, we propose to make improvements from three aspects including, 1) improving time efficiency by training a DNN via layer-wise learning mechanism, 2) reducing sample complexity by using a performance lower bound based surrogate model for fitness evaluation, 3) enhancing learning effectiveness via integrating a gradient-based local search with previous two advancements. The new algorithm has been developed successfully to achieve state-of-the-art performance in comparison to three cutting-edge policy gradient search algorithms (i.e., ACKTR, TRPO, and PPO) and two advanced evolutionary algorithms (i.e., OpenAI-ES and Uber-GA).

Firstly, we show that layer-wise learning (PES-S1) can significantly reduce computational cost without sacrificing learning effectiveness for CMA-ES. The layer-wise learning aims to train a DNN layer by layer rather than to train it as a whole. In this way, the solution dimensionality can be largely reduced, which paves the way for CMA-ES to be applied to training a large-scale DNN effectively and improves the computational efficiency of CMA-ES.

Secondly, we show that a surrogate model based learning (PES-S2) can largely improve the re-usability of interaction samples for CMA-ES meanwhile maintain competitive effectiveness to cutting-edge algorithms. PES-S2 is developed with two stages, one is to learn a surrogate model with gradient-descent technique, and the other is to use the model to replace the actual simulation for individual fitness evaluation of CMA-ES. By avoiding evaluating the individuals by collecting new samples from the environment every time, PES-S2 is significantly more sample efficient than other EAs, such as CMA-ES, OpenAI-ES, and Uber-GA.

Thirdly, we show that a local search enhanced learning (PES-S3) can further boost the learning effectiveness by locally fine-tuning the best solution evolved by CMA-ES. In this way, PES seamlessly integrates gradient-based local search

with evolutionary global search, which takes metrics from both EAs and PGS to better balance the exploration and exploitation. Empirically, PES shows better learning performance with equivalent sample efficiency that state-of-the-art DRL algorithms such as ACKTR, TRPO, and PPO.

In summary, PES can solve DRL problems efficiently and effectively with a reasonable level of sample complexity. The PES algorithm developed in this chapter focuses on improving PGS with evolutionary algorithms to reach a state-of-the-art performance level. However, the algorithm can still experience unreliable learning on value functions, which can cause the degradation in learning performance. Clearly, the reliability of value function learning is the key to the success of PGS algorithms. Especially those algorithms rely greatly on value functions. To achieve reliable learning on value functions, it is expected to use some mechanisms to stabilize value function learning. In addition, for most PGS algorithms that follow Policy Gradient Theorem (PGT) [8], a flexible family of compatible functions can also help to improve the learning effectiveness of policy learning as such flexibility can provide opportunities to obtain more accurate estimations of policy gradients. In view of this, the next chapter will develop two approaches, one aims at stabilizing the value function learning, and the other is to generalize compatible function to provide a flexible value function learning.

[8] Please refer to Section 2.2.5 for more technical details about PGT.

Chapter 6

Reliable and Flexible Value Function Learning for Policy Direct Search

This chapter of the thesis aims at achieving the research objective O(3) in Section 1.3 meanwhile answering the research question Q(3) in Section 1.2. Actor-Critic (AC) algorithms, as typical Policy Gradient Search (PGS) algorithms, are usually composed of two distinct learning processes, namely actor (a.k.a, policy) learning and critic (a.k.a, value function) learning. Actor learning is heavily dependent on critic learning. Particularly when critic learning becomes unstable, the learned value function diverges. This will significantly affect the effectiveness of AC algorithms. To address this issue, many successful algorithms have been developed recently with the aim of achieving reliable and flexible learning of value functions. To achieve reliable learning, Gradient Temporal Difference (GTD) algorithm [214] and GTD2 [213] have been proposed to use off-policy training techniques, but they both empirically exhibit divergence on learning. Least Square Temporal Difference (LSTD) [37, 36] as second-order methods can guarantee the value function learning reliability, but they possess high computational complexity $O(n^2)$ (n is the number of state features). These issues can actually lead to less effective policy learning either because of the divergence of value function learning or because of the high sample cost. On the other hand, to achieve flexible value function learning for PGS, the Policy Gradi-

ent Theorem (PGT) [1] proposed by Sutton et.al. [215] provides a theoretical foundation to learn a variety of different value functions as long as the compatible condition [2] is fulfilled. Almost all the PGS algorithms [215, 190, 170, 121, 29, 31] developed so far are strictly stick to the PGT, but they have overlooked that the compatible condition itself can be generalized by q-logarithm [223, 42] to potentially provide more flexible compatible functions. Because the q-logarithm generalization can introduce an extra parameter q to control the degree of freedom of the function, which enables a family of function forms. Thus, in this chapter, we develop two approaches resulting in two new PGS algorithms. One approach is to improve the critic learning reliability by integrating with the Sandpile Model (SM) [18] with a self-organized property, and the other is to generalize the logarithm function based on the q-logarithm principle for compatible function condition to achieve more flexible value function learning. With the development of the two algorithms, we have achieved two contributions as follows:

1. We propose to adopt the Sandpile Model into value function learning process to achieve self-adaptive and reliable learning of value functions. This new technique for critic learning is further integrated with a commonly used PGS algorithm, i.e., Regular Actor-Critic (RAC) [31]. The algorithm is called Sandpile Model based RAC (SM-RAC).

2. We propose to use a generalized logarithm function to create a new and flexible family of compatible functions. The widely used compatible function based on natural logarithm can then be treated as a special case of our compatible functions. This enables us to achieve flexible and adaptive critic learning and more accurate estimation of policy gradients. We have subsequently developed a new algorithm called Generalized Compatible Function Approximation based Regular Actor-Critic (GCFA-RAC).

Our experiments on four benchmark control problems, including Puddle World [212], Cart Pole [212], Mountain Car [212], and Heating Coil [84], have shown that,

[1] Please refer to Section 2.2.5 for more technical details about PGT.

[2] Please refer to Section 2.2.5for more technical details about the compatible function.

- SM-RAC can perform significantly better than its base algorithm, i.e., RAC.

- Under suitable generalization of compatible function approximation, any algorithms that use traditional compatible functions for critic learning can benefit from using our generalized new family of compatible functions.

6.1 Introduction

As introduced previously in Section 2.2.5, AC algorithms are important PGS methods for solving RL problems [121, 122, 79]. In practice, it is difficult to obtain analytical policy gradients, thus unbiased estimation of the policy gradients is often used for actor learning [212, 215, 112, 30, 52]. In AC framework, the policy gradient estimation can be determined from PGT [190],

$$\nabla_{\vec{\theta}} J(\vec{\theta}) = \mathbf{E}\Big[\sum_{t=0}^{\infty} \Psi_t \cdot \Phi(\vec{s}_t, a_t)\Big] \tag{6.1}$$

where $J(\vec{\theta})$ denotes the expected total rewards. $\Phi(\vec{s}_t, a_t) = \nabla_{\vec{\theta}} \ln \pi_{\vec{\theta}}(a_t|\vec{s}_t)$ in PGT serves as a function compatible with policy parameterization. Based on $\Phi(\vec{s}, a)$, Q function can be approximated further as $Q^{\pi}(\vec{s}) \approx \vec{\omega}^{\pi T} \cdot \Phi(\vec{s}, a)$. Moreover, Ψ_t in (6.1) have several different but related expressions as follows:

- $\delta_t = r_t + V^{\pi}(\vec{s}_{t+1}) - V^{\pi}(\vec{s}_t)$: Temporal Difference (TD) error [212]

- $Q^{\pi}(\vec{s}_t, a_t)$: state action value function [212]

- $A^{\pi}(\vec{s}_t, a_t) = Q^{\pi}(\vec{s}_t, a_t) - V^{\pi}(\vec{s}_t)$: advantage value function [215]

According to (6.1), accurate estimation of policy gradients and effective reinforcement learning depends on both Ψ_t and $\Phi(\vec{s}_t, a_t)$. In general, the former is learned via critic learning process and the latter is directly computed from policy π.

6.1.1 Chapter Goals

The overall goal of the chapter is to develop new techniques for reliably adaptive and flexible approximation of valuation functions. As a result, we expect to improve the effectiveness of PGS algorithms further. To achieve this goal, we develop two new mechanisms, i.e., an adapted Sandpile Model based technique to stabilize the critic learning and a flexible new family of compatible functions based on a generalized logarithm function to estimate policy gradients adaptively and accurately. Although we focus primarily on evaluating both techniques on the RAC algorithm [31], they are general enough to be straightforwardly applied to many AC algorithms. RAC is chosen in this chapter due to its theoretical convergence guarantee and proven effectiveness on many benchmark RL problems.

Our research in this chapter also leads to the development of two new algorithms, and the first algorithm is called *Sandpile Model based Regular Actor-Critic (SM-RAC)*, which is expected to improve the learning effectiveness of PGS by enhancing the reliability of value function learning. With SM-RAC, we intend to answer two important research questions:

Q1 Can the learning effectiveness of AC algorithms, such as RAC, be significantly improved upon using our adapted SM to stabilize the critic learning?

Q2 Are effective RL and stable critic learning strongly correlated in AC algorithms, including RAC and SM-RAC?

The second proposed algorithm is called *Generalized Compatible Function Approximation based Regular Actor-Critic (GCFA-RAC)*, which is expected to improve the learning effectiveness of PGS by adopting a generalized family of compatible functions. With the development of GCFA-RAC, we aim to study the research question specifically:

Q3 When the generalized logarithm function $\mathcal{G}(\cdot)$ is used to produce flexible compatible features Φ, will RAC algorithm become more effective and reliable for RL?

6.1.2 Chapter Organization

This chapter is structured as follows. A preliminary knowledge about RAC and SM is given in Section 6.2. The two proposed algorithms, SM-RAC and GCFA-RAC, are presented in Section 6.3. The design of experiments, including feature design, general setup, and experiment design, is detailed in Section 6.4, followed by results and discussions in Section 6.5. A chapter summary is presented in Section 6.6.

6.2 Preliminaries

This section introduces the preliminary knowledge for the research of this chapter and paves the way for the development of two new PGS algorithms. Firstly, the RAC algorithm is depicted. Secondly, the concept of SM is explained.

6.2.1 Regular Actor-Critic Algorithm

Critic learning in RAC is guided by the well-known Temporal Different error (TD error) defined as

$$\delta_t^\pi = r_{t+1} + \gamma V^\pi(\vec{s}_{t+1}) - V^\pi(\vec{s}_t). \tag{6.2}$$

Meanwhile the critic in RAC also approximates the true value function $V^\pi(\vec{s})$ in (2.38) by

$$V^\pi(\vec{s}) \approx \tilde{V}^\pi(\vec{s}) = \vec{v}^{\pi T} \cdot \phi(\vec{s}), \tag{6.3}$$

where $\vec{v}^{\pi T}$ consists of the value function parameters that are linearly associated with the basis function $\phi(\vec{s}) = [\phi_1(\vec{s}), \ldots, \phi_m(\vec{s})] \in \mathbb{R}^m$. Accordingly, the goal of critic learning is to adjust the value function parameters in the direction of reducing the TD error, i.e.,

$$\vec{v}_{t+1}^\pi \leftarrow \vec{v}_t^\pi + \alpha_t \delta_t^\pi \phi(\vec{s}_t), \tag{6.4}$$

where α is the critic learning rate at time t.

Actor learning in RAC aims to find the optimal policy parameter $\vec{\theta}^*$ so as to find the optimal policy. The learning follows the direction given by the policy gradient where Q^π is approximated as,

$$Q^\pi(\vec{s}) \approx \tilde{Q}^\pi(\vec{s}) = \vec{\omega}^{\pi T} \cdot \Phi(\vec{s}, a), \tag{6.5}$$

where $\vec{\omega}^\pi$ is made up of parameters that estimate $Q^\pi(\vec{s})$, and the compatible feature is defined as $\Phi(\vec{s}, a) = \nabla_{\vec{\theta}} \ln \pi(\vec{s}, a)$. Subsequently, through an unbiased estimation of $\nabla_{\vec{\theta}} J(\vec{\theta})$, the incremental rule for actor learning is further proposed in [31] as

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \beta_t \delta_t^\pi \Phi(\vec{s}, a), \tag{6.6}$$

where β_t is the actor learning rate.

Through iterative application of (6.4) and (6.6), RAC is widely shown to successfully solve many benchmark RL problems. It is clear to see that in RAC the estimation of the policy gradients in (6.6) is strongly dependent on the quality of critic learning as well as the effect of the compatible function approximation. Also note that, Algorithm 6.3.1 can be regarded as the exact algorithmic description of RAC after excluding lines from 12 to 26. Also, Algorithm 6.3.2 can be directly adapted to RAC by changing the line 11 back to the original policy updating rule in (6.6).

6.2.2 Sandpile Model

The SM (a.k.a., Bak-Tang-Wiesenfeld model) is a model proposed in [18] to explain an important property of dynamical systems called *Self Organized Criticality (SOC)*. SOC refers to a phenomenon that a system evolves towards a critical point through self-adjustments over its lifetime. In SM, whenever a system (e.g., a sand pile) reaches its critical point, any small perturbation (e.g., dropping a grain of sands) may trigger a *noise* propagation (e.g., an avalanche of the sand pile) of varying sizes. After a short while, the system is guaranteed to return to its the critical point through self-organization.

In [18], the SM is simulated in a 2D $N \times N$ grid with a boundary (i.e., $[0, N]^2$), where each cell (i.e., site (x, y)) of the grid contains an integer value $z(x, y)$ representing the slope of the sand pile on-site. It is also assumed that, if a grain of

sand is added one at a time on a random site and it leads to an avalanche, then only after the avalanche stops can the next grain of sand be added. Following this setting, SM as a mathematical model contains three key components:

- *Neighborhoods and Boundaries*

  In the 2D SM, the neighborhoods of any site (x, y) are defined as its four adjoining sites, namely $(x \pm 1, y)$ and $(x, y \pm 1)$. Besides, the sites on boundaries are defined as $(0, y)$, (N, y), $(x, 0)$, and (x, N).

- *The Reliability Criterion*

  A key factor of the SM is to determine the condition under which an avalanche will be triggered upon adding a new grain of sand. This is equivalent to defining a reliability criterion as below,

$$\varepsilon(z) = \begin{cases} 0, & K - z(x, y) \geq 0 \\ 1, & K - z(x, y) < 0 \end{cases} \tag{6.7}$$

  where K is the predefined threshold (the critical value). As seen in (6.7), the site is reliable when its z value is smaller than K, i.e., $\varepsilon(z) = 0$. On the other hand, an avalanche will be triggered at any site (x, y) when the value $z(x, y)$ exceeds K.

- *The Propagation/Updating Rule*

  An avalanche causes a noise propagation to its neighborhoods, such a propagation can be defined as

$$\begin{aligned} z(x, y) &\leftarrow z(x, y) - \Delta \\ z(x \pm 1, y) &\leftarrow z(x \pm 1, y) + \tfrac{\Delta}{4} \\ z(x, y \pm 1) &\leftarrow z(x, y \pm 1) + \tfrac{\Delta}{4} \end{aligned} , \tag{6.8}$$

  where Δ represents the noise to be propagated from the site (x, y) to its neighborhoods. Note that, the noise being propagated to the boundaries will be disregarded, i.e., $z(x, y) \equiv 0$.

Clearly, although the model description above considers only two dimensions, the SM can be easily expanded to multiple dimensional cases [18]. In Section 6.3.1, we show how to construct an adapted SM for reliable critic learning in RL scenario.

6.3 The Proposed Algorithms — SM-RAC and GCFA-RAC

To achieve the research objectives of this chapter, we develop two new PGS algorithms, i.e., Sandpile Model based Regular Actor-Critic (SM-RAC) and Generalized Compatible Function Approximation based Regular Actor-Critic (GCFA-RAC). In the following section, we first propose SM-RAC to stabilize critic learning. Next, we propose GCFA-RAC to generalize the compatible function so as to further improve the quality of policy gradients.

6.3.1 Sandpile Model based Regular Actor-Critic (SM-RAC)

In this subsection, we propose the SM-RAC algorithm with the aim of improving the critic learning reliability. For this purpose, we firstly propose a criterion for measuring the learning reliability. Following that, we show how to adapt the SM to combine it with the critic learning process in RAC. Lastly, we present an algorithmic description of SM-RAC.

Critic Learning Reliability

We define the concept of the critic learning reliability below.

Definition 6.3.1. The *Critic Learning Reliability* refers to the total probability for the absolute value of the value function's output to be higher than a predefined threshold across all visited states across all possible episodes.

Since $\vec{v}$ and $\vec{\theta}$ jointly determine the behavior of RAC, the reliability criterion for critic learning during any learning episode τ can be presented mathematically as

$$\varepsilon_\tau(\vec{v}, \vec{\theta}) = \int_{\vec{s} \sim d^\pi(\vec{s})} \mathbf{I}(\vec{s}) d^\pi(\vec{s}) d\vec{s}, \tag{6.9}$$

where

$$\mathbf{I}(\vec{s}) = \begin{cases} 0, & \bar{R} - |\vec{v}^T \cdot \phi(\vec{s})| \geq 0 \\ 1, & \bar{R} - |\vec{v}^T \cdot \phi(\vec{s})| < 0 \end{cases}, \tag{6.10}$$

is an indicator function based on a predefined threshold $\bar{R}$. The choices of $\bar{R}$ for practical learning tasks will be explained in Section 6.4.1.

ε in (6.9) is a continuous measure of critic learning reliability. In particular, $\varepsilon_\tau(\vec{v}, \vec{\theta}) = 0$ indicates that the critic learning is *completely reliable*. Otherwise, $\varepsilon_\tau(\vec{v}, \vec{\theta}) > 0$ shows the degree of divergence in the learned value function.

An Adapted Sandpile Model for Critic Learning

To integrate the SM into RAC, based on the three components of the SM described in Section 6.2.2, we propose the adapted SM for critic learning as follows.

- Neighborhoods and Boundaries

  Our adapted SM is defined over the history $\mathcal{H}$ of all previously visited states in an episode. In particular, each state (and its corresponding value) is treated as a separate site in the SM. Hence, any state $V^\pi(\vec{s}_j)$ has its neighborhoods made up of $V^\pi(\vec{s}_{j-1})$ and $V^\pi(\vec{s}_{j+1})$. Moreover, the boundary of the SM is determined by the most recent and least recently visited states in the history, whereas the length of history is bounded from above by $l_\mathcal{H}$.

- *The Reliability Criterion*

  Referring to the reliability criterion in Section 6.3.1, the critical point value K in (6.7) for our criterion is defined as the upper bound of the expected cumulative reward, i.e., $\bar{R}$. Intuitively whenever the critic in RAC predicts non-achievable cumulative rewards, falling outside the range defined by $\bar{R}$, it is highly skeptical that critic learning starts to become unreliable. Based on this idea, we can define the absolute bound $\bar{R}$ as,

  $$\bar{R} = \sum_{i=0}^{T} \gamma^i |r_i| + \epsilon, \tag{6.11}$$

  where ϵ is a small *error margin* which is necessary since the value function is estimated by linear function approximation in RAC.

- *The Propagation/Updating Rule*

Suppose that at least one site, i.e., $\vec{s}_j$, over the full history has been identified as unreliable, similar to the original SM in [18], the propagation/updating rule described below can be applied:

$$\begin{array}{rl} \tilde{V}^{\pi}_{t+1}(\vec{s}_j) & \leftarrow \tilde{V}^{\pi}_{t+1}(\vec{s}_j) - \Delta \\ \tilde{V}^{\pi}_{t+1}(\vec{s}_{j-1}) & \leftarrow \tilde{V}^{\pi}_{t+1}(\vec{s}_{j-1}) + \rho\frac{\Delta}{2} \\ \tilde{V}^{\pi}_{t+1}(\vec{s}_{j+1}) & \leftarrow \tilde{V}^{\pi}_{t+1}(\vec{s}_{j+1}) + \rho\frac{\Delta}{2} \end{array} . \tag{6.12}$$

Δ in (6.12) is defined below

$$\Delta \ = \text{sign}(\tilde{V}^{\pi}(\vec{s}_t))\psi_{\bar{R}}\bar{R} \ , \tag{6.13}$$

where $\text{sign}(\tilde{V}^{\pi}(\vec{s}_t)) = \begin{cases} 1, & \tilde{V}^{\pi}(\vec{s}_t) \geq 0 \\ -1, & \tilde{V}^{\pi}(\vec{s}_t) < 0 \end{cases}$.

Note that, $\psi_{\bar{R}} \in [0, 1]$ is the noisy level factor. For example, if $\psi_{\bar{R}} = 0.1$, it means that 90% of the maximum allowed cumulative rewards will be maintained.

In (6.12), we define a new hyper-parameter $\rho \in [0, 1]$ as the damping factor to avoid big changes to the critic that can potentially affect the learning effectiveness of RAC. Moreover, based on (6.3), we can have (6.12) re-written as

$$\begin{array}{rl} \vec{v}_{t+1} & \leftarrow \vec{v}_{t+1} - \frac{\Delta}{\phi(\vec{s}_j)} \\ \vec{v}_{t+1} & \leftarrow \vec{v}_{t+1} + \frac{0.5\rho\Delta}{\phi(\vec{s}_{j-1})} \\ \vec{v}_{t+1} & \leftarrow \vec{v}_{t+1} + \frac{0.5\rho\Delta}{\phi(\vec{s}_{j+1})} \end{array} . \tag{6.14}$$

The adapted SM proposed above will be applied iteratively at every learning step so as to guarantee critic learning reliability.

The SM-RAC Algorithm

Our adapted SM stated above can be easily incorporated into the RAC algorithm. First, we maintain at most $l_{\mathcal{H}}$ recently visited states in history. Next, the reliability of the SM over all visited states is examined according to (6.9). Subsequently, if there exists one or multiple unreliable sites, one of them will be

Algorithm 6.3.1 Sandpile Model based Regular Actor-Critic (SM-RAC)

Require: an MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, the expected reward upper bound $\bar{R}$, the noisy level factor $\psi_{\bar{R}}$, the damping factor ρ, the maximum length for the state history $l_{\mathcal{H}}$

Ensure: $\vec{\theta}$, $\vec{v}^{\pi}$

Initialization:
$\vec{\theta} \leftarrow \vec{\theta}_0$, $\vec{v}^{\pi} \leftarrow \vec{v}_0^{\pi}$,
$\vec{s}_t \leftarrow \vec{s}_0$, where $\vec{s}_0$ is an arbitrary initial state
$\mathcal{H} \leftarrow \{\}, l \leftarrow 0, j \leftarrow 0, \Delta \leftarrow 0$
Learning Process:
for $\tau = 0, 1, 2, \ldots, \tau_{max}$ **do**
  for $t = 0, 1, 2, \ldots, T$ **do**
    $a_t \sim \pi_{\vec{\theta}}(a|\vec{s}_t)$
    Take action a_t, observe reward r_{t+1} and new state $\vec{s}_{t+1}$
    $\delta_t^{\pi} \leftarrow r_{t+1} + \gamma \vec{v}_t^{\pi T} \cdot \phi(\vec{s}_{t+1}) - \vec{v}_t^{\pi T} \cdot \phi(\vec{s}_t)$
    $\vec{v}_{t+1}^{\pi} \leftarrow \vec{v}_t^{\pi} + \alpha_t \delta_t^{\pi} \phi(\vec{s}_t)$

    $\mathcal{H} \leftarrow \mathcal{H} \cup \{\vec{s}_t\}$
    if $l < l_{\mathcal{H}}$ **then**
      $l \leftarrow l + 1$
    else
      $\mathcal{H} \leftarrow \mathcal{H} \setminus \vec{s}_{t-l}$
    end if
    while $\varepsilon_{\tau}(\vec{v}_{t+1}, \vec{\theta}_t) > 0$ **do**
      while $\vec{s}_j \in \mathcal{H}$ **do**
        if $\bar{R} < |\vec{v}_{t+1}^T \cdot \phi(\vec{s}_j)|$ **then**
          $\Delta \leftarrow \text{sign}(\vec{v}_{t+1}^{\pi T} \cdot \phi(\vec{s}_j)) \bar{R} \psi_{\bar{R}}$
          $\vec{v}_{t+1} \leftarrow (1 - \frac{\Delta}{\vec{v}_{t+1} \cdot \phi(\vec{s}_j)}) \vec{v}_{t+1}$
          if $j > 1$ **then**
            $\vec{v}_{t+1} \leftarrow (1 + \frac{0.5\rho\Delta}{\vec{v}_{t+1} \cdot \phi(\vec{s}_{j-1})}) \vec{v}_{t+1}$
          end if
          if $j < l_{\mathcal{H}} - 1$ **then**
            $\vec{v}_{t+1} \leftarrow (1 + \frac{0.5\rho\Delta}{\vec{v}_{t+1} \cdot \phi(\vec{s}_{j+1})}) \vec{v}_{t+1}$
          end if
        end if
        $j \leftarrow j + 1$
      end while
    end while

    $\vec{\theta}_{t+1} \leftarrow \vec{\theta}_t + \beta_t \delta_t^{\pi} \Phi(\vec{s}_t, a_t)$
  end for
  $\mathcal{H} \leftarrow \{\}, l \leftarrow 0, j \leftarrow 0, \Delta \leftarrow 0$
end for
return $\vec{\theta}$, $\vec{v}^{\pi}$

randomly selected and the propagation rule (6.14) will be applied to it. This procedure will be repeated until all sites in the SM are reliable. We present an algorithmic description of SM-RAC in Algorithm 6.3.1.

6.3.2 Generalized Compatible Function Approximation base Regular Actor-Critic (GCFA-RAC)

In this subsection, we propose the GCFA-RAC algorithm to seek a flexible family of compatible functions to improve the effectiveness of RAC. Firstly, we derive the generalized compatible function approximation to formulate new policy updating rule. Following that, we present the pseudo code of GCFA-RAC.

Generalized Compatible Function Approximation

Here, we focus primarily on generalizing compatible function $\Phi^{\pi}(\vec{s}, a)$. To do so, we propose to use a generalized logarithm function $\mathcal{G}(\cdot)$ as shown below to construct the compatible feature Φ^{π},

$$\mathcal{G}(\pi_{\vec{\theta}}(a|\vec{s}), \nu) = \frac{\pi_{\vec{\theta}}(a|\vec{s})^{1-\nu} - 1}{1-\nu}. \tag{6.15}$$

It is clear to see from (6.15) that the level of generalization in $\mathcal{G}(\cdot)$ is controlled by ν. Whenever $\nu = 1$, $\mathcal{G}(\cdot)$ degrades to the standard logarithm function, i.e.

$$\lim_{\nu \to 1} \mathcal{G}(\pi_{\vec{\theta}}(a|\vec{s}), \nu) = \ln \pi_{\vec{\theta}}(a|\vec{s}). \tag{6.16}$$

Hence, we can formulate the generalization below,

$$\mathcal{G}(\pi_{\vec{\theta}}(a|\vec{s}), \nu) = \begin{cases} \frac{\pi_{\vec{\theta}}(a|\vec{s})^{1-\nu}-1}{1-\nu}, & \nu \neq 1 \\ \ln \pi_{\vec{\theta}}(a|\vec{s}), & \nu = 1 \end{cases} \tag{6.17}$$

For simplicity, ν in (6.17) will be called the *compatible generalization factor*. In consequence, the generalized compatible feature can be constructed as,

$$\begin{aligned} \Phi^{\pi}(\vec{s}, a) &= \frac{\partial \mathcal{G}(\pi_{\vec{\theta}}(a|\vec{s}), \nu)}{\partial \vec{\theta}} \\ &= \frac{\partial \pi_{\vec{\theta}}(a|\vec{s})}{\partial \vec{\theta}} \pi_{\vec{\theta}}(a|\vec{s})^{1-\nu}, \end{aligned} \tag{6.18}$$

Based on (6.2), (6.6) and (6.18), the updating rule for the policy parameter $\vec{\theta}$ can be determined as,

$$\vec{\theta}_{t+1} \leftarrow \vec{\theta}_t + \beta \eta \delta_t^{\pi} \frac{\partial \pi_{\vec{\theta}}(a|\vec{s})}{\partial \vec{\theta}} \pi_{\vec{\theta}}(a|\vec{s})^{1-\nu}, \tag{6.19}$$

where a new constant factor η is introduced in (6.19) to ensure that $\vec{\theta}$ will be updated at the same scale as in the original RAC. For this purpose, we need to solve the following equation,

$$\mathbf{E}[|\frac{\partial \ln \pi_{\vec{\theta}}(a|\vec{s})}{\partial \vec{\theta}}|\big|a] \quad = \mathbf{E}[|\eta \frac{\partial \mathcal{G}(\pi(a|\vec{s}),\nu)}{\partial \vec{\theta}}|\big|a]. \tag{6.20}$$

Follow the convention in many existing RL algorithms [76, 212, 31], we dictate action sampling to obey a Gaussian distribution in any state according to policy $\pi_{\vec{\theta}}(a|\vec{s})$, hence,

$$\pi_{\vec{\theta}}(a|\vec{s}) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(a-\mu)^2}{2\sigma^2}}, \tag{6.21}$$

where $\mu = \vec{\theta}^T \cdot \phi(\vec{s})$ is the mean action output from policy π in state $\vec{s}$, which can be adjusted by changing policy parameters $\vec{\theta}$. On the other hand, the standard deviation σ in (6.21) is pre-determined (i.e., $\sigma = 1.0$). Note that π in RHS of (6.21) refers to the circumference ratio. Therefore, from (6.20) and (6.21), we can directly determine the value for η below,

$$\eta = \begin{cases} \frac{(2\pi)^{\frac{1-\nu}{2}}(\nu-2)\sigma^{1-\nu}\left(1-e^{-\frac{(\mu+k)^2}{2\sigma^2}}\right)}{e^{\frac{(\nu-2)(k-\mu)^2}{2\sigma^2}}+e^{\frac{(\nu-2)(\mu+k)^2}{2\sigma^2}}-2}, & \nu \neq 2 \\ \frac{\sqrt{\frac{2}{\pi}}\sigma\left(1-e^{-\frac{(\mu+k)^2}{2\sigma^2}}\right)}{(\mu+k)^2}, & \nu = 2 \end{cases}. \tag{6.22}$$

Again π in (6.22) is the circumference ratio. Meanwhile, k and $-k$ give the upper and lower bounds for the action output from any policy, respectively.

The GCFA-RAC Algorithm

As discussed above, our generalized compatible function can directly utilized in the RAC algorithm by replacing the policy learning rule in (6.6) to (6.19). The detailed algorithmic description of GCFA-RAC is given in Algorithm 6.3.2.

6.4 Design of Experiments

This section of the chapter introduces our designed experiments for evaluating the two proposed algorithms respectively. For SM-RAC, it introduces general

Algorithm 6.3.2 Generalized Compatible Function Approximation based Regular Actor-Critic Algorithm (GCFA-RAC)

Require: an MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$
Ensure: $\vec{\theta}$, $\vec{v}^{\pi}$
Initialization:
$\vec{\theta} \leftarrow \vec{\theta}_0$
$\vec{v}^{\pi} \leftarrow \vec{v}_0^{\pi}$
$\vec{s} \leftarrow \vec{s}_0$
Learning Process:
for $t = 0, 1, 2, ...$ **do**
    $a_t \sim \pi_{\vec{\theta}}(\vec{s}_t, a)$
    Take action a_t, observe reward r_{t+1} and new state $\vec{s}_{t+1}$
    $\delta_t^{\pi} \leftarrow r_{t+1} + \gamma \vec{v}_t^{\pi T} \cdot \phi(\vec{s}_{t+1}) - \vec{v}_t^{\pi T} \cdot \phi(\vec{s}_t)$
    $\vec{v}_{t+1}^{\pi} \leftarrow \vec{v}_t^{\pi} + \alpha \delta \phi(\vec{s}_t)$
    $\vec{\theta}_{t+1} \leftarrow \vec{\theta}_t + \beta \eta \delta_t^{\pi} \frac{\partial \pi_{\vec{\theta}}(a|\vec{s})}{\partial \vec{\theta}} \pi_{\vec{\theta}}(a|\vec{s})^{1-\nu}$
end for
return $\vec{\theta}$, $\vec{v}^{\pi}$

experimental setups including policy implementation, state feature design, and hyper-parameter configurations. Following that, it discusses the SM-RAC experiment design. For GCFA-RAC, it follows the descriptive structure as that of SM-RAC experiments.

6.4.1 Experiments on SM-RAC

Experiment Setup

In this subsection, we describe the detailed setups of our experiments for SM-RAC. We first formulate the stochastic policy (i.e., Gaussian Policy). Subsequently, we discuss the state feature as well as some important hyper-parameters settings.

Policy Implementation

In this experiment, we consider the stochastic policy, i.e., $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$, which is more robust than a deterministic policy when environments are stochastic [212, 198]. Here, we implement our stochastic policy as a Gaussian distribution which is well-studied for continuous problems [168]. Specifically, the probability density for taking each action is given by (6.21). In (6.21), μ is determined by the policy parameters $\vec{\theta}$ and the basis function $\phi(\vec{s})$, i.e., $\mu = \vec{\theta}^T \cdot \phi(\vec{s})$. On the other hand, σ is considered an exploration parameter and is fixed to 1.0 for all problems. Note that, π at the RHS of (6.21) is the circumference ratio.

State Feature Design

In this chapter, we design our state features as triangle basis functions which have been used in [43]. We have also empirically assessed discretization feature function [126, 212], which performed worse in comparison to our choice. This is because, when projecting the low-level state inputs to the high-level feature spaces, discretization feature function creates discontinuities in policies. Since continuous control problems require a smooth feature space, state features based on triangle basis functions, as shown below, are considered more suitable,

$$\phi : \mathbb{R}^d \rightarrow \mathbb{R}^m,$$

where d is the dimension of the state input, and m is the dimension of the feature space.

Figure 6.1 depicts how the triangle basis function is used for one dimension of the state input. As seen in the figure, each dimension of the state input is limited to the range interval $[\iota_{min}, \iota_{max}]$. We then split the interval into n equal segments with $n - 1$ vertexes. In our experiments, $n = 10$. Next, we select each vertex as an apex to connect with the adjacent vertexes to construct a group of triangles. For the cases when the adjacent vertexes are ι_{min} or ι_{max}, we connect the apex to the boundary instead of vertexes. For each triangle, it covers a partial range of the state input. When the value of the dimension s_d for the

incoming state input (i.e., $\vec{s} = [s_0, s_1, \ldots, s_d] \in \mathbb{R}^d$) falls into the range, its corresponding features will be computed as,

$$\phi(s_k)_i = \begin{cases} 1, & \iota_{min} \leq s_k < \iota_{min} + \kappa \\ 1, & \iota_{max} - \kappa < s_k \leq \iota_{max} \\ \frac{s_k - \kappa*(i+1)}{\kappa*(i+1) - \kappa*i} + 1, & s_k \leq \iota_{min} + \kappa * (i+1) \\ \frac{s_k - \kappa*(i+2)}{\kappa*(i+2) - \kappa*(i+1)}, & s_k \geq \iota_{min} + \kappa * (i+1) \\ 0, & \text{otherwise} \end{cases},$$

where $k \leq d$, and $i = 0, 1, \ldots, n-1$, meanwhile $\kappa = \frac{|\iota_{min}| + |\iota_{max}|}{n}$. Note that, the final feature dimension is determined as $m = d * n$.

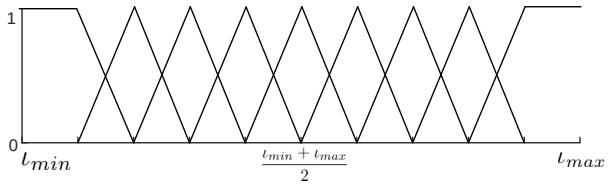


Figure 6.1: The triangle basis function used for defining one single dimension of the state input.

Hyper-Parameter Configurations

Here, we discuss the hyper-parameter configurations for SM-RAC. To investigate the impact of the critic learning reliability on the learning performance, we followed those hyper-parameter settings summarized in Table 6.1. They enable us to study the behavior of RAC when critic learning is clearly unreliable or even diverging. This is important because, when the learned value function satisfies our reliability criterion, SM-RAC and RAC behave the same.

The settings for $\bar{R}$ as seen in Table 6.1 are problem-specific. Regarding the Puddle World problem, following the reward scheme described in Section 3.1 and (6.11), the maximum expected cumulative reward $\bar{R}$ is determined as 120.

Table 6.1: The hyper-parameter configurations for experiments of RAC and SM-RAC on the Puddle World problem and the Mountain Car problem.

Algorithms	**Problems**	**Meta Parameters**						
		α	β	γ	$\bar{\mathcal{R}}$	$\psi_{\bar{\mathcal{R}}}$	ρ	$l_{\mathcal{H}}$
RAC	**Puddle World**	0.01	0.0001	0.99	N/A	N/A	N/A	N/A
	Mountain Car	0.1	0.005	0.99	N/A	N/A	N/A	N/A
SM-RAC	**Puddle World**	0.01	0.0001	0.99	120	0.1	0.1	100
	Mountain Car	0.1	0.005	0.99	100	0.1	0.1	100

We can easily think that the worst case is that the agent is trapped in the environment obtaining -1.0 mostly for 100 steps so that we can have theoretical maximum cumulative rewards as $+100$. Additionally, the error margin (i.e., ϵ) here is set to 20% of the maximum theoretical value. Similarly, in the Mountain Car problem, the threshold $\bar{R}$ can be determined as $+100$.

Experiment Design

Firstly, to obtain reliable experimental results, we give the general running settings for the experiments. For every learning algorithm and every benchmark problem, we will perform 30 independent trials (i.e., one complete training and testing process) over 10000 training episodes. For one trial, after every 50 training episodes, we will run 30 tests to verify the current performance of the actor learned by RAC and SM-RAC respectively. Furthermore, a maximum number of 100 steps applies to every training and testing episode.

Secondly, we design two experiments aiming at addressing the two specific research questions (i.e., **Q1** and **Q2**). In the first experiment, we perform both RAC and SM-RAC based on the running settings given above. During the experiment, we track two important metrics, namely the average total rewards obtained by each algorithm during their testing phases and the average value of value function (i.e., the average expected total reward). Base on the two metrics, we can tell the influences of the reliability variations of critic learning (defined in Section 6.3.1) on the final learning effectiveness (in terms of the average total

rewards). In the second experiment, we use the results of the two metrics obtained from the last experiment to conduct correlation analysis. Here, we only consider the linear correlation between these two metrics.

6.4.2 Experiment on GCFA-RAC

Experiment Setup

Policy Implementation

For experiments of GCFA-RAC, we have adopted the exactly same policy implementation as that of SM-RAC described in Section 6.4.1 above.

State Feature Design

For experiments of GCFA-RAC, we have adopted the exactly same state feature design as that of SM-RAC described in Section 6.4.1 above.

Hyper-Parameter Configurations

For GCFA-RAC, to study the effect of generalization level of compatible function approximation on the final learning effectiveness, we summarized the hyper-parameter setting for RAC on each problem in Table 6.2.

Table 6.2: The hyper-parameter configurations for experiments of RAC on the Puddle World problem, the Mountain Car problem, the Cart Pole problem, and the Heating Coil problem.

Algorithms	**Problems**	**Meta Parameters**		
		α	β	γ
RAC	**Puddle World**	0.1	0.01	0.99
	Mountain Car	0.2	0.07	0.99
	Cart Pole	0.01	0.005	0.99
	Heating Coil	0.1	0.05	0.99

Experiment Design

To understand the efficacy of using generalized updating rule for learning policy parameters in (6.19), we will evaluate the performance of RAC on three benchmark problems. Thus, here we collect only the average total rewards obtained by RAC during its testing phase. In order to obtain reliable results, 30 independent trials will be performed on each benchmark problem and with respect to different settings of the compatible generalization factor ν (note that when $\nu = 1$, the original updating rule in (6.6) is realized). A total of eight different settings for ν have been examined in our experiments (see Table 6.3). To simplify our discussion, in this section, CASE-X will denote the experiments on RAC when $\nu = X$.

Table 6.3: Experiment Common Settings for One Trial.

Problem	**Training**		**Testing**		**Evaluating ν values**
	Episodes	Steps	Episodes	Steps	
Puddle world	10000	1000	50	100	{0.5, 0.7, 0.9, **1.0**, 1.1, 1.5, 2.0}
Cart Pole	20000	50	50	1000	{0.5, 0.7, 0.9, **1.0**, 1.1, 1.5, 2.0}
Heating Coil	5000	500	50	500	{0.5, 0.7, 0.9, **1.0**, 1.1, 1.5, 2.0}

6.5 Results and Discussion

In this section, we present and analyze the experimental results. First, we focus on the experimental results of SM-RAC. In particular, we first discuss results of SM-RAC on Puddle World and Mountain Car in comparison to that of RAC to answer **Q1** in Section 6.1.1. We then analyze the correlation between the learning effectiveness and the critic learning reliability to answer **Q2** in Section 6.1.1. Second, we focus on the experiment results of GCFA-RAC. For GCFA-RAC, we particularly are interested in analyzing the effect of generalization to learning effectiveness on RAC algorithm so as to answer **Q3** in Section 6.1.1.

6.5.1 Discussion on Results of SM-RAC

We will discuss the experimental results on two benchmark problems separately. For the discussion on results collected from each problem, we will first compare the critic learning reliability of RAC and SM-RAC. Based on the reliability differences, we further compare the learning performance of the two algorithms. Lastly, we investigate the relationship in-between the learning effectiveness and the learning reliability by adopting a correlation analysis.

Discussion on Results of SM-RAC in Puddle World

To evaluate the critic learning reliability, we present the average of the absolute values generated from the value function learned by RAC and SM-RAC at every 50 episodes in Figure 6.2. As seen in Figure 6.2, the RAC algorithm exhibits a very unreliable behavior since its corresponding learned critic values fluctuate severely. A sudden change occurs at the 1350th episode indicated as a black dashed line, and then it tends to diverge rapidly. Such a change results in an immediate degradation in the learning performance in terms of the average cumulative rewards as shown in Figure 6.3. In contrast, also from Figure 6.2, the critic learning reliability of SM-RAC has been well maintained at a reasonable level, staying mostly beneath the predefined threshold $\bar{R}$. Correspondingly, the learning performance of SM-RAC also shows a converging behavior evidenced in Figure 6.3.

Next, we will compare the learning performance of SM-RAC and RAC based on average cumulative rewards in Figure 6.3. As discussed above, after the 1350th episode, the entire learning process of RAC diverges and never recovers due to its unreliable critic learning. In contrast, SM-RAC shows a continuous improvement on the learning performance, although it tends to converge after 1350th episode thanks to the convergence of its critic learning. Additionally, we have performed a Student T-test for comparing the performances of the two algorithms which produces a p-value of 9.6864×10^{-10}. This suggests that SM-RAC performs significantly better than RAC on the Puddle World problem.

In summary, Figure 6.2 and Figure 6.3 reflect some facts that 1) SM-RAC

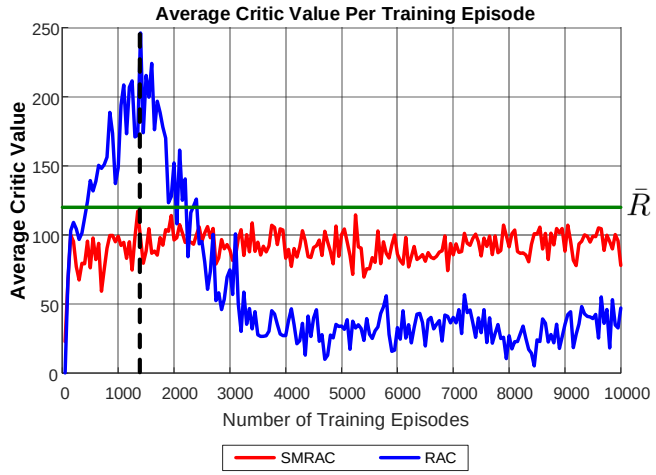


Figure 6.2: Average of the absolute values generated from the value function learned by RAC and SM-RAC at every 50 episodes on the Puddle World problem.

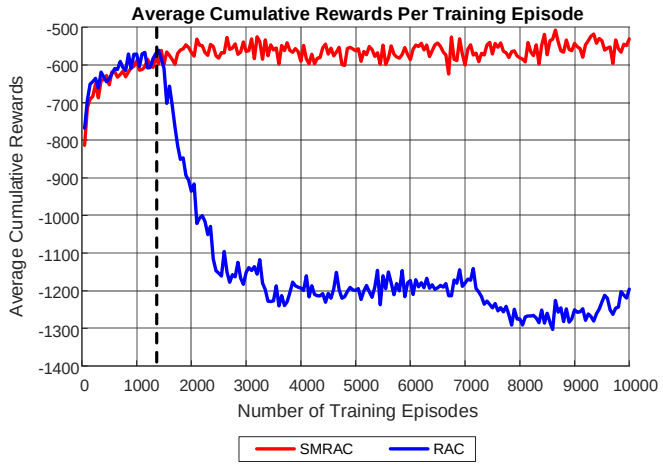


Figure 6.3: Average cumulative rewards obtained by RAC and SM-RAC at every 50 episodes on the Puddle World problem.

performs learning more reliably and more effectively than RAC does, and 2) the learning reliability to some extent correlates to the learning effectiveness.

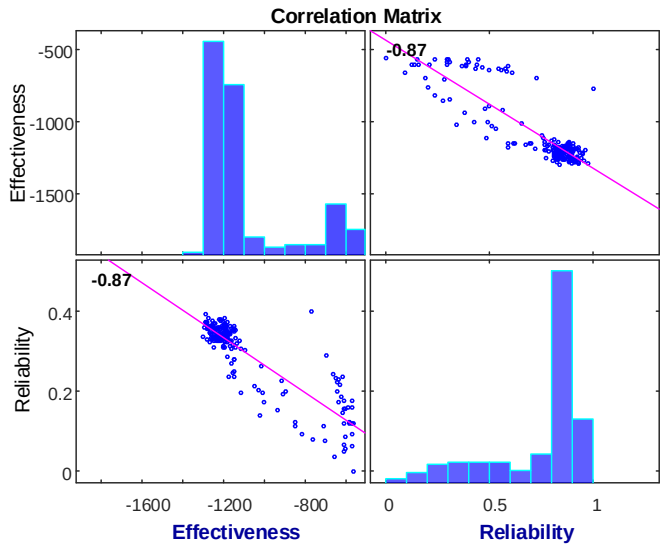


Figure 6.4: Correlation between the learning effectiveness and the learning reliability of RAC on the Puddle World problem.

Aiming at knowing to what extent the critic learning reliability affects the learning effectiveness, we also perform correlation analysis. As seen from Figure 6.2 and Figure 6.3, it is easy to observe that there exists a correlation in-between the learning reliability and the learning effectiveness. To verify this correlation, we adopt the Pearson Correlation analysis here. Firstly, we follow (6.9) to quantify the learning reliability. Besides, we follow the convention to use average cumulative rewards to measure the learning effectiveness. The correlation matrix for RAC on the Puddle World problem is given in Figure 6.4, which shows a correlation coefficient of -0.87 close to -1. This suggests that learning reliability has a strong positive correlation because a higher value in 6.9 indicates poorer reliability for critic learning to the learning performance.

Discussion on Results of SM-RAC in Mountain Car

In comparison to the Puddle World problem, very similar experimental results can be obtained on the Mountain Car problem. Accordingly, most of the claims made in Section 6.5.1 also hold here.

Figure 6.5 shows the average value of the learned critic by RAC and SM-RAC on the Mountain Car problem. Clearly, the critic learning of RAC diverges quickly after 450 episodes, and the value of critic keeps increasing until the 3650th episode. Even the critic value starts to decrease after the 3650th episode, and it cannot prevent policy parameters from diverging further. In contrast, a clear converging trend of SM-RAC can be witnessed in Figure 6.5 towards the critical point $\bar{R}$, suggesting the higher reliability of SM-RAC compared to that of RAC.

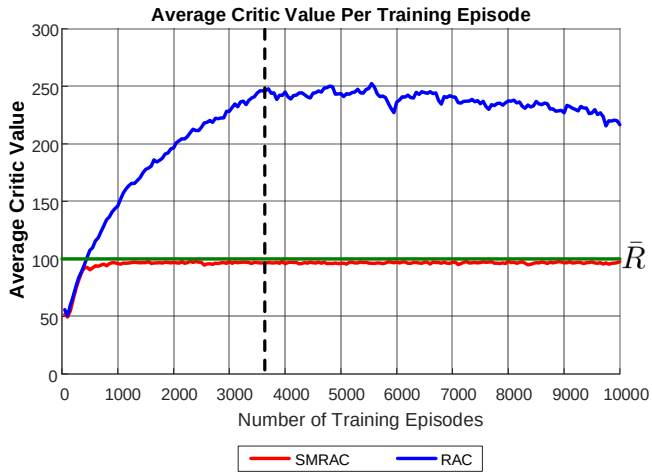


Figure 6.5: Average of the absolute values generated from the value function learned by RAC and SM-RAC at every 50 episodes on the Mountain Car problem.

As in Figure 6.6, the apparent difference can be easily identified the fact that SM-RAC performs significantly better than RAC. This fact is supported by the significance test where the p-value is 0.0145.

The correlation analysis based on the results collected by RAC on the Mountain Car problem is presented in Figure 6.7. This analysis also indicates that the strong positive correlation exists in-between the learning reliability and the learning effectiveness of RAC, as demonstrated by the correlation coefficient

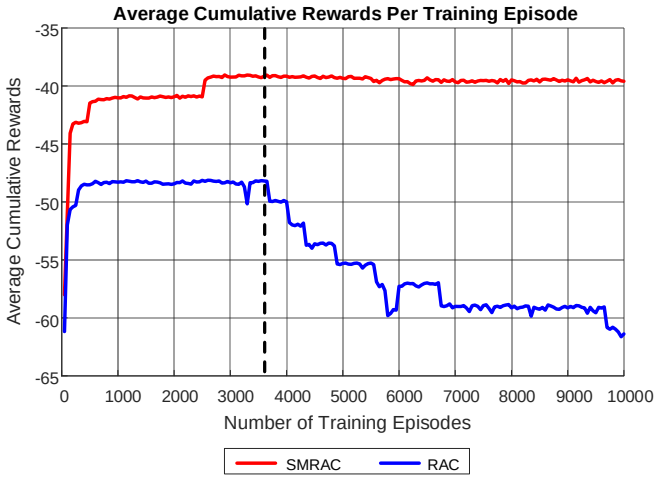


Figure 6.6: Average value function learned by RAC and SM-RAC at every 50 episodes on the Mountain Car problem.

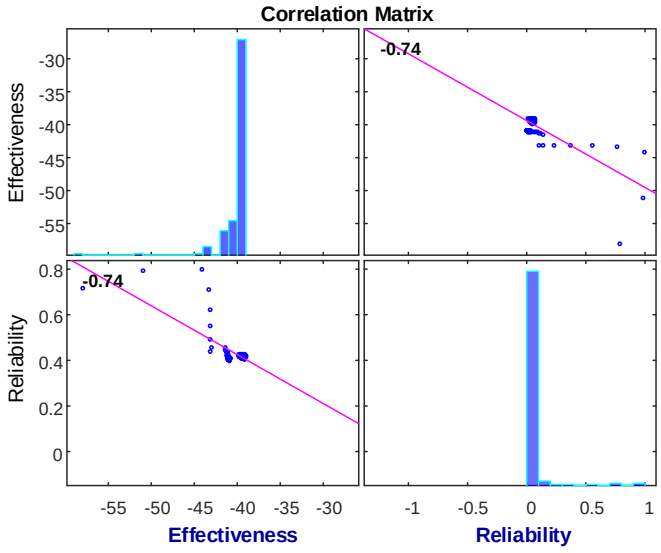


Figure 6.7: Correlation between the learning effectiveness and the learning reliability of RAC on the Mountain Car problem.

equals to -0.74.

6.5.2 Discussion on Results of GCFA-RAC

We will discuss the experimental results on three benchmark problems separately. Here, we only focus on comparisons of learning performances obtained by RAC under different generalization level.

Discussion on Results of GCFA-RAC in Puddle World

To compare the performance differences among various settings of ν, the average steps to reach the goal region upon using the policies learned through the proposed algorithm is presented in Figure 6.8. As seen from Figure 6.8, near-optimal policies can be learned successfully whenever $\nu \in [0.9, 2.0]$. On the other hand, CASE-0.5 and CASE-0.7 failed to solve this problem satisfactorily. Among all the results presented in Figure 6.8, CASE-1.5 appears to achieve the best performance by observation (i.e. on average 19.2 steps to reach the goal region). In comparison, for CASE-1.0 (i.e. original Algorithm 6.3.2), the average steps to reach the goal region is 46.4 after 10,000 learning episodes have been completed. A t-test is performed in between CASE-1.0 and CASE-1.5, and it produces a p-value of 0.10, insufficient to prove that CASE-1.5 is significantly better. However, we found that the problem is solved successfully by CASE-1.5 100% of the time. For CASE-1.0, the problem is solved on only 94% of the trials. This observation suggests that CASE-1.5 can solve the problem more reliably.

Discussion on Results of GCFA-RAC in Cart Pole

To compare the learning performances, the average ξ and the average balancing steps (i.e., the duration for the pole to be balanced continually) are presented in Figure 6.9 and Figure 6.10. It can be observed in Figure 6.9 that, in comparison to other cases, during a long learning period from 3000 training episodes to the end, CASE-1.5 can manage to bring the pole closer to the upright position on average. For example, at 3000 training episodes, the average ξ achieved by CASE-1.5 is -0.01. In comparison, CASE-1.0 can only manage to achieve on average of -0.07 for ξ. However, this observed performance difference is not verifiable through statistical tests (perhaps more repeated tests are to be performed

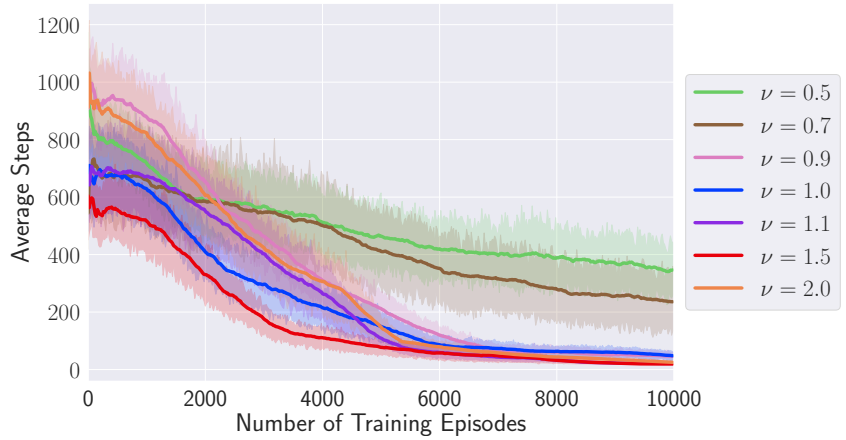


Figure 6.8: Average steps to the goal region of the Puddle World problem ($\nu = 1.0$ is the original RAC).

in order to reveal significant differences between CASE-1.0 and CASE-1.5).

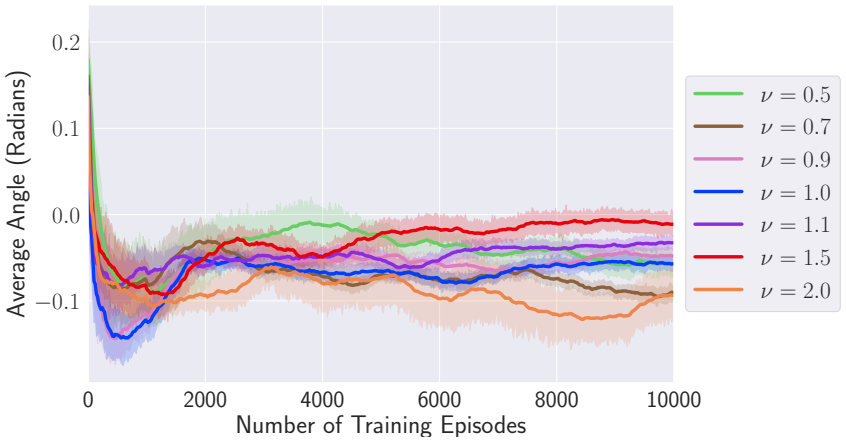


Figure 6.9: Average ξ on the Cart Pole problem ($\nu = 1.0$ is the original RAC).

On the other hand, by checking the average balancing steps in Figure 6.10, we found that most of the cases can solve this problem reasonably well. The only case that falls apart is when $\nu = 2.0$, suggesting that the value for ν cannot significantly differ from 1.0.

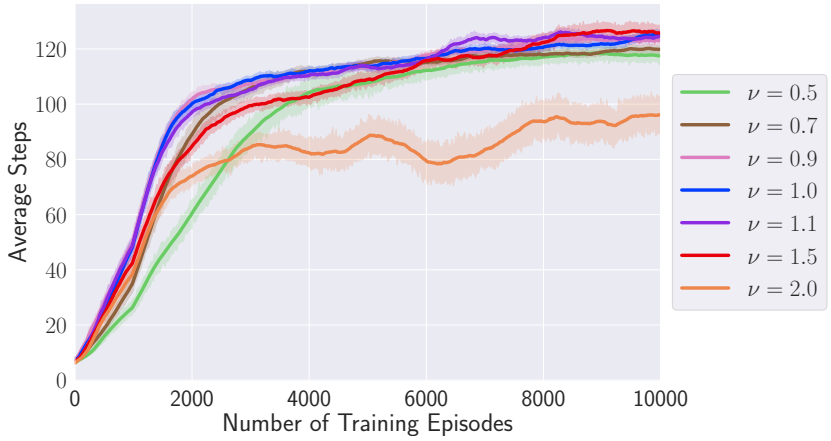


Figure 6.10: Average balancing steps on the Cart Pole problem ($\nu = 1.0$ is the original RAC).

Discussion on Results of GCFA-RAC in Heating Coil

Figure 6.11 displays the learning effectiveness (i.e., average errors) of the algorithm with different ν (from 0.5 to 2.0) as being performed on the Heating-Coil Problem. Similar to evaluations on other benchmark problems, CASE-1.5 maintains the lowest average deviation with a decreasing trend from start to finish in comparison with other cases. In the end, it reaches 1.57 degree whereas CASE-1.0 only learns to reduce the average error to 1.68 degrees. However, the statistical significance is not attained between CASE-1.0 and CASE-1.5 with the p-value 0.11. In fact, regardless of the values of ν, the algorithm does not satisfactorily resolve the problem, as even the lowest degree temperature difference (1.57 degree) obtained so far is still not acceptable for a real-world HVAC system. Perhaps, a fine-tuning process for meta parameters (e.g., learning rate) may be required to ensure the learning effectiveness.

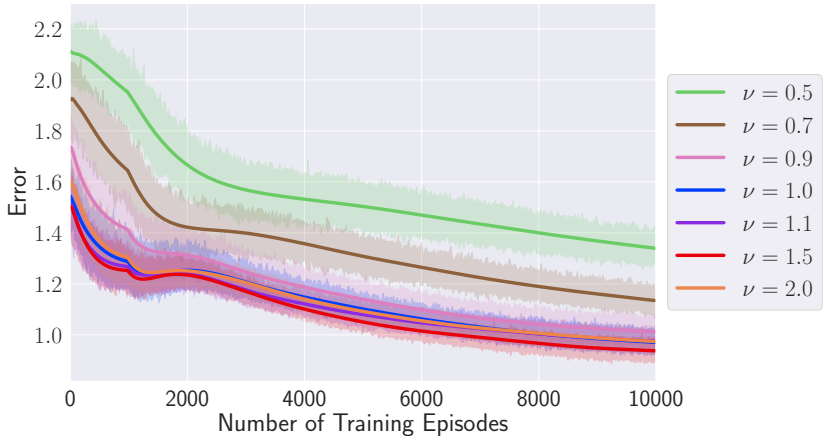


Figure 6.11: The average error (i.e., average deviation) between the output temperature T_{ao} and the target temperature T_d ($\nu = 1.0$ is the original RAC) on the Heating-Coil problem.

6.6 Chapter Summary

In this chapter, we have successfully achieved the goal of developing for stable, flexible and accurate critic learning in an AC algorithm. More specifically, we investigated both the impacts of critic learning reliability and the effects of generalized compatible function on the learning effectiveness of a particular PGS algorithm – RAC.

Experimental results have confirmed the effectiveness of both SM-RAC and GCFA-RAC. By conducting experiments with SM-RAC, we can confidently answer the two research questions (**Q1** and **Q2**) presented in Section 6.1.1. Regarding Q1, we found that SM-RAC outperformed RAC on two benchmark problems significantly, yet neither algorithm has obtained the theoretical best performances shown in [43]. In fact, to achieve the best performance, a fine-tuning process for hyper-parameters must be conducted. To answer Q2, we have adopted the correlation analysis on the RAC algorithm on two benchmark problems. The correlation coefficients are respectively -0.87 and -0.74, clearly indicating a strong positive correlation between learning reliability and learning performance. Regarding GCFA-RAC, our experimental evaluation clearly

Table 6.4: The final episode performance comparison of different ν values (i.e., 0.5, 0.7, 0.9, 1.0, 1.1, 1.5, 2.0) on three benchmark problems (i.e., Cart Pole, Puddle World and Heating Coil).

ν/Problems	Cart Pole	Heating Coil	Puddle World
0.5	10650.12±2472.31	-1471.94±577.65	-20.87±45.07
0.7	11048.34±2120.16	-1454.68±681.98	-1.39±35.14
0.9	10884.25±2406.71	-1672.25±801.01	9.00±23.54
1.0	10287.91±2750.27	-1537.19±576.76	8.32±24.42
1.1	11199.29±3364.35	-1704.35±832.52	14.23±5.46
1.5	**11432.56±3380.95**	**-1608.01±701.52**	**15.01±3.20**
2.0	8474.00±5418.03	-1590.10±596.38	13.52±8.38

shows that there exists a strong correlation between the degree of generalization in (6.19) and the learning effectiveness as well as reliability. Whenever we set ν in (6.19) to a proper value, e.g. 1.5, clear improvements on learning performance and reliably can be witnessed.

With the development of SM-RAC, this chapter shows that stable critic learning and learning effectiveness are strongly correlated. Moreover, we show that the adapted sandpile model can be effectively utilized for improving critic learning reliability in AC algorithms. These findings shed new lights on the future development of AC algorithms.

With the development of GCFA-RAC, this chapter shows the possibility of using generalized compatible features for value function approximation. GCFA-RAC shows the potential of achieving effective and reliable RL through using generalized compatible features and compatible functions. It invites the research community to re-investigate the meaning of compatible value function under the general AC framework.

This chapter does not pay strong attention to state features. Similar to many existing research works, the state features are manually designed in this chapter which is often a tedious task even for domain experts. With the aim of effectively extracting useful state features automatically, the next chapter will develop innovative evolutionary algorithms for automated feature extraction.

Chapter 7

Enhancing Policy Direct Search via Automated Evolutionary Feature Learning

In this chapter, we aim at answering the research question Q(4) presented in Section 1.2 to achieve the research objective O(4) in Section 1.3. As discussed in the previous chapter, Actor-Critic (AC) Algorithms have proven effectiveness on tackling difficult Reinforcement Learning (RL) tasks, where the critic model defines the value function learner whereas the actor model defines the policy [121, 215]. Both models share the same collection of state features. Such features are typically extracted from the raw environmental inputs. However, most of existing AC algorithms focus on improving the critic learning via various gradient-descent algorithms without significantly enhancing the feature extraction process. It is often assumed that good state features can be designed by experienced domain experts, which is usually time-consuming and error-prone [26, 52, 232]. Recently, the issue has seemed to be mitigated due to the progress of deep representation learning where features are expected to be automatically learned through deep network structure [26]. However, the predetermination of the network structure requires the injection of prior knowledge which is human-biased [152].

In order to address the issue and further improve the effectiveness of PGS,

we develop two new PGS algorithms based on automated evolutionary feature learning. First, we propose to use NeuroEvolution, in particular, NeuroEvoluion with Augmenting Topology (NEAT) [206], which can evolve Neural Networks' topologies that are suitable to extract useful features, to automate the feature extraction process. Following the idea, we develop a new algorithm called NEAT with Regular Actor-Critic (NEAT+RAC). Second, NEAT+RAC is further generalized further to construct NEAT with PGS (NEAT+PGS) with improved sample efficiency and effectiveness. With the development of the two algorithms, we successfully achieved three contributions:

1. With NEAT+RAC, we demonstrate the suitability of NEAT for automated feature learning in conjunction with RAC. The resulted algorithm (i.e., NEAT+RAC) is capable of learning not only useful state features but also good policies to tackle complex RL problems.

2. While developing NEAT+PGS, we propose a new three-stage learning scheme that can clearly separate feature learning and policy learning, allowing effective knowledge sharing and learning among multiple concurrently learning agents.

3. Under the framework of NEAT+PGS, various PGS algorithms can be seamlessly integrated with NEAT for training policy networks with deep structures to achieve effective and sample efficient RL.

The experiments have been conducted on two benchmark control problems (i.e., Cart Pole and Mountain Car) and six benchmark Atari game playing tasks (i.e., Asteroids, Breakout, Freeway, Seaquest, SpaceInvaders, and TimePilot). Results of the experiments have shown that,

- NEAT+RAC is significantly more effective than NEAT in terms of learning performance. The learned features through NEAT on one learning problem can be reused to improve the effectiveness of RAC on solving related learning problems.

- NEAT+PGS is more effective and more sample efficient than NEAT and three state-of-the-art Deep Reinforcement Learning (DRL) algorithms

(i.e., Trust Region Policy Optimization (TRPO) [189], Policy learning by weighting exploration with the returns (POWER) [117], Advantage Actor-Critic (A2C) [56]).

7.1 Introduction

Many existing AC algorithms assume that the policy (i.e., actor) can be mathematically described through a linear function of multiple numerical state features. Accordingly, the critic is often modeled through another linear function of compatible state features. In [215], Sutton discovered an important relationship between the critic and the actor such that the compatible state features can be uniquely derived from the state features for the actor. Driven by this mathematical framework, many effective AC algorithms have been proposed, including Regular Actor-Critic (RAC) [31], Natural Actor-Critic (NAC) [170, 112] and so forth [79, 52, 168].

For all these algorithms, it is taken for granted that suitable state features are immediately accessible during reinforcement learning. However, the validity of this assumption is often under challenge in practice. As a matter of fact, researchers found that, for effective RL, these state features must be carefully designed, usually with the support of domain experts. Obviously, engineering useful state features is a time-consuming and error-prone procedure [150, 165, 26, 232]. Even for experienced domain experts, it is difficult to predetermine suitable state features accurately [236, 26]. This difficulty increases the chance of using inappropriate state features, where important state features may be overlooked, resulting in serious deterioration in learning performance.

7.1.1 Chapter Goals

The overall goal of the chapter is to develop new algorithms to improve the learning effectiveness and sample efficiency of existing PGS algorithms with the help of automated evolutionary feature learning. To achieve this goal, we first extend RAC with a new NEAT based automated feature learning component, resulting in the development of the NEAT+RAC algorithm. Next, we further

improve the effectiveness and sample efficiency with the development of the NEAT+RAC algorithm that features the use of an innovative three-stage learning scheme and various cutting-edge DRL algorithms. In this chapter, we aim to achieve four research objectives with the development of NEAT+RAC and NEAT+PGS:

1. (NEAT+RAC) To develop a new method to automatically extract suitable state features through Neural Networks (NN) evolved by NEAT and utilize state features extracted by the evolved NNs in RAC for effective policy search.

2. (NEAT+RAC) To evaluate the effectiveness of NEAT-RAC on two benchmark small-dimensional problems, and to examine the usefulness (i.e., reusability) of the feature learned by NEAT-RAC on one learning problem in improving the effectiveness of RAC on a related learning problem.

3. (NEAT+PGS) To design a three-stage learning scheme that jointly accommodates NEAT based automated feature learning with various PGS algorithms for effective policy learning.

4. (NEAT+PGS) To assess the effectiveness and sample efficiency of NEAT+PGS on large-scale RL problems particularly Atari game playing tasks.

7.1.2 Chapter Organization

The chapter is organized as follows. Session 7.2 presents the two proposed algorithms, NEAT+RAC and NEAT+PGS. Next, Section 7.3 describes the design of the experiments including descriptions on hyper-parameters settings and experiment design. Following that, results and discussions are given in Section 7.4. The chapter is summarized in Section 7.5.

7.2 The Proposed Algorithms - NEAT+RAC and NEAT+PGS

The chapter develops two new PGS algorithms enhanced by NEAT based feature learning, i.e., NEAT based Feature Learning enhanced Regular Actor-Critic (NEAT+RAC) and NEAT based Feature Learning enhanced Policy Gradient Search (NEAT+PGS). We first propose NEAT+RAC to demonstrate the effectiveness of using evolutionary automated feature learning for a conventional PGS algorithm – RAC. Afterward, we propose a generalized algorithm NEAT+PGS by adopting a three-stage learning scheme to enable knowledge sharing among learning agent. NEAT+PGS is expected to achieve better effectiveness and sample efficiency in comparison to both NEAT and NEAT+PGS.

7.2.1 NEAT based Feature Learning enhanced Regular Actor-Critic (NEAT+RAC)

In this section, we propose the NEAT+RAC algorithm to tackle RL problems. Firstly, we present the overall design of NEAT+RAC. Next, we describe NEAT+RAC in detail. Lastly, we discuss the new characteristics of NEAT+RAC in comparison to existing RL algorithms including NEAT and RAC.

Overall Design of NEAT+RAC

Figure 7.1 shows an overall design of our NEAT+RAC algorithm. As shown in the figure, NEAT+RAC evolves a population of RL learning agents, $P = \{A_1, \ldots, A_p\}$. Each agent A_i consists of three components: an NN $\vec{\phi}(\vec{s}) \in \mathbb{R}^z$, a parameter vector for value function $\vec{\omega}_i$, and a parameter vector for policy $\vec{\theta}_i$. Based on the extracted features $\vec{\phi}(\vec{s})$, the value function in each agent is approximated as,

$$V^{\pi}(\vec{s}) \approx \vec{\omega}^{\pi T} \cdot \vec{\phi}(\vec{s}), \tag{7.1}$$

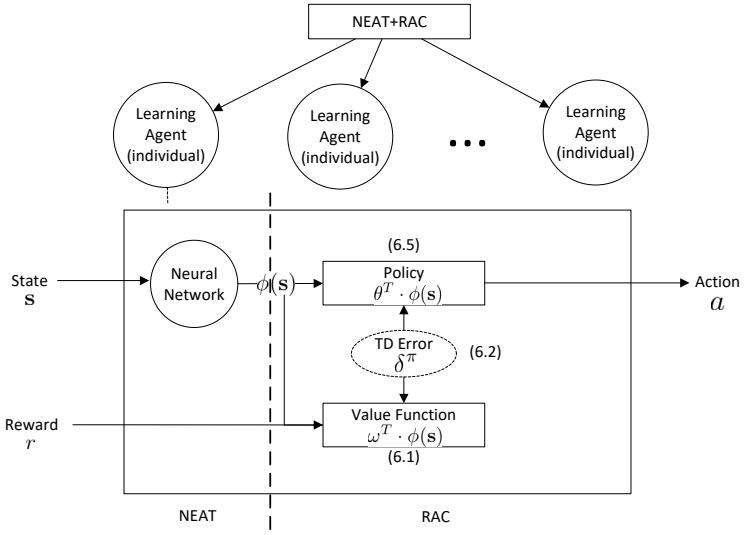


Figure 7.1: The overall design of NEAT+RAC.

and is learned following the reducing direction of Temporal Difference (TD) error at every t time when the agent reaches a new state at $t+1$, i.e.,

$$\delta_t^\pi = r_{t+1} + \gamma V^\pi(\vec{s}_{t+1}) - V^\pi(\vec{s}_t). \tag{7.2}$$

Additionally, the policy for action selection is formulated as,

$$\pi_{\vec{\theta}}(a|\vec{s}) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(a-\mu)^2}{2\sigma^2}}, \tag{7.3}$$

where $\mu = \vec{\theta}^T \cdot \phi(\vec{s})$. $\sigma = 1.0$ is used to control the level to explore new actions. Note that, π at the RHS of (7.3) is the circumference ratio.

In association with (7.1), (7.2) and (7.3), the updating rule for value function parameters is,

$$\vec{\omega}_{t+1}^\pi \leftarrow \vec{\omega}_t^\pi + \alpha_t \delta_t^\pi \vec{\phi}(\vec{s}_t), \tag{7.4}$$

and the rule for policy parameters is

$$\vec{\theta}_{t+1} \leftarrow \vec{\theta}_t + \beta_t \delta_t^\pi \Phi(\vec{s}, a), \tag{7.5}$$

respectively. Note, α_t and β_t are the learning rates, and $\Phi(\vec{s}, a) = \nabla_{\vec{\theta}} \ln \pi(\vec{s}, a) = \frac{a_t - \vec{\theta} \cdot \vec{\phi}(\vec{s}_t)}{\sigma^2} \cdot \vec{\phi}(\vec{s}_t)$ [215].

NEAT based Feature Learning enhanced Regular Actor-Critic

NEAT+RAC is designed with four learning stages, namely *initialization*, *evolution*, *evaluation*, and *termination*.

Initialization

The initialization stage of NEAT+RAC is to initialize a population with p learning agents (i.e., individuals). For the NN in each agent, its inputs and outputs are defined as states $\vec{s}$ and extracted state features $\vec{\phi}(\vec{s})$ respectively. Additionally, its input nodes are directly connected to its output nodes without any hidden nodes, and weights of each NN are randomly initialized. For the critic and actor model in each agent, the value function parameters and policy parameters are initialized to $\vec{\omega}_0$ and $\vec{\theta}_0$ respectively. Note, $\vec{\omega}_0$ and $\vec{\theta}_0$ are vectors with arbitrary values. This stage is described in Algorithm 7.2.1.

Algorithm 7.2.1 NEAT+RAC Initialization

Require: p: population size, d: state dimension, z: feature dimension
Ensure: P: a population of learning agents

```
function INITIALIZATION(p, d, z:)
    P[] ← new array of size p
    for k = 1, 2, ..., p do
        P[k].ω⃗ ← ω⃗_0
        P[k].θ⃗ ← θ⃗_0
        P[k].N ← INIT_NETWORK(d, z)
        P[k].N.fitness ← 0                    ▷ See [206]
    end for
    return P
end function
```

Evolution

NEAT+RAC evolves a new population of agents by performing two sequential tasks according to Section 7.2.1. The first task is to evolve p NNs ($\{\vec{\phi}_1, \ldots, \vec{\phi}_p\}$)

based on the standard evolutionary operators of NEAT defined in [206]. The second task is to initialize $\vec{\omega}_i$ and $\vec{\theta}_i$ to $\vec{0}$ for every agent A_i in the new population. This ensures a fair evaluation across all agents. This algorithmic description of this state is given in Algorithm 7.2.2.

Algorithm 7.2.2 NEAT+RAC Evolution

Require: P: a population of learning agents, p: population size, m_n: add node mutation rate, m_l: add link mutation rate, m_w: weight mutation rate

Ensure: P: a reproduced population of learning agents

function EVOLUTION(P, p, m_n, m_l, m_w)
  P'[] $\leftarrow$ new array of size p
  for $k = 1, 2, ..., p$ **do**
    $P'[k].\vec{\omega} \leftarrow \vec{\omega}_0$
    $P'[k].\vec{\theta} \leftarrow \vec{\theta}_0$
    $P'[k].N \leftarrow$ BREED_NET(P[].N) ▷ See [206]
    if RANDOM() $< m_n$ **then**
      ADD_NOTE_MUTATION($P'[k].N$) ▷ See [206]
    end if
    if RANDOM() $< m_l$ **then**
      ADD_LINK_MUTATION($P'[k].N$) ▷ See [206]
    end if
    if RANDOM() $< m_w$ **then**
      WEIGHTS_MUTATION($P'[k].N$) ▷ See [206]
    end if
    $P'[k].N.fitness \leftarrow 0$
  end for
  return P'
end function

Evaluation

The evaluation stage of the NEAT+RAC has two objectives: one is to compute the fitness value for a single individual, and the other is to find good policies.

The fitness value of each individual is directly computed by averaging total rewards obtained by the learning agent over all episodes. Using an average value can help reduce the variance as rewards are collected in a stochastic environment. To search for the good policies, we use the RAC to learn the critic and actor models for an individual from its continuous interactions with the environment. This stage is presented in Algorithm 7.2.3.

Termination

The termination stage manages the stop criteria for the learning process of NEAT+RAC. To stop the feature extraction process, we define two stop criteria: the first is to stop when the predefined maximum number of generations is reached, and the second is when the fitness values of all individuals do not improve over ten generations. To cease the policy search process, we set up the maximumly allowed learning episodes for an RL agent to interact with an environment. Each episode is composed of multi interaction steps, and it terminates at the situations when, either the predefined maximum number of steps is reached, or the target RL problem is solved (see Section **??**). A full algorithmic description for NEAT+RAC is given in Algorithm 7.2.4.

7.2.2 NEAT based Feature Learning enhanced Policy Gradient Search (NEAT+PGS)

This section presents our new NEAT+PGS learning scheme in detail. We firstly provide an overview of the three-stage learning scheme. Next, we give detailed algorithmic descriptions of all key functions involved.

An Overview of NEAT+PGS

NEAT+PGS is proposed with the aim of achieving three design objectives: (O1) feature learning, and policy learning should be wholly separated and supported by their learning techniques; (O2) the knowledge obtained through learning,

Algorithm 7.2.3 NEAT+RAC Evaluation

Require: P: a population of learning agents, p: population size, e_g: maximum training episodes per generation, T: maximum training steps per episode, α: value function learning rate, β: policy learning rate, $\tilde{R}$: total rewards

Ensure: P: a population of learning agents

function EVALUATION(P, p, e_g, T, α, β)
  for $k = 1, 2, ..., p$ **do**
    $\tilde{R} \leftarrow 0$
    for $j = 1, 2, ..., e_g$ **do**
      $\vec{s}_t \leftarrow \vec{s}_0$
      for $t = 0, 1, \ldots, T-1$ **do**
        $a_t \sim \pi_{\vec{\theta}}(a|\vec{s}_t)$ ▷ See (7.3)
        Take action a_t, observe reward r_{t+1} and new state $\vec{s}_{t+1}$
        $\delta_t \leftarrow r_{t+1} + \gamma\vec{\omega}_t^T \cdot \vec{\phi}(\vec{s}_{t+1}) - \vec{\omega}_t^T \cdot \vec{\phi}(\vec{s}_t)$ ▷ See (7.2)
      ▷ Note: $\vec{\phi} = P[k].N$, $\vec{\omega}_t = P[k].\vec{\omega}$, $\vec{\theta}_t = P[k].\vec{\theta}$
        $\vec{\omega}_{t+1} \leftarrow \vec{\omega}_t + \alpha\delta_t \cdot \vec{\phi}(\vec{s}_t)$ ▷ See (7.4)
        $\vec{\theta}_{t+1} \leftarrow \vec{\theta}_t + \beta\delta_t \cdot \Phi(\vec{s}_t, a_t)$ ▷ See (7.5)
        $\tilde{R} \leftarrow \tilde{R} + r_{t+1}$
        if TERMINAL-STATE($\vec{s}_{t+1}$) **then**
          break
        end if
      end for
    end for
    $P[k].N.fitness \leftarrow \frac{\tilde{R}}{e_g}$
  end for
  return P
end function

in particular, the trained policy network, must be shared across all learning agents to avoid unnecessary wastage of training samples; and (O3) policy network training should be realized with the flexibility of using arbitrary PGS algorithms.

Algorithm 7.2.4 NEAT+RAC Algorithm

Require: an MDP $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, p: population size, g: number of generations, e_g: maximum training episodes per generation, T: maximum training steps per episode, d: state dimension, z: feature dimension, α: value function learning rate, β: policy learning rate, $\vec{\omega}_0$: initial value function parameters, $\vec{\beta}_0$: initial policy parameters

Ensure: P: the population of individuals, N^*: the optimal Neural Network ($\phi(\vec{s})$), $\vec{\omega}^*$: the best value function parameter, $\vec{\theta}^*$: the best policy parameter

Initialization:
$P \leftarrow$ `INITIALIZATION`(p, d, z, $\vec{\omega}_0$, $\vec{\beta}_0$) ▷ See Section 7.2.1
Learning Process:
for $i = 1, 2, ..., g$ **do**
    $P \leftarrow$ `EVALUATION`(P, p, e_g, T, α, β) ▷ See Algorithm 7.2.3
    if $i < g$ **then** ▷ Stop evolution at the final generation
        $P \leftarrow$ `EVOLUTION`(P) ▷ See [206]
    end if
end for
$k^* \leftarrow \text{argmax}_k(P[k].N.fitness)$
$N^* \leftarrow P[k^*].N$
$\vec{\omega}^* \leftarrow P[k^*].\vec{\omega}$
$\vec{\theta}^* \leftarrow P[k^*].\vec{\theta}$
return P, N^*, $\vec{\omega}^*$, $\vec{\theta}^*$

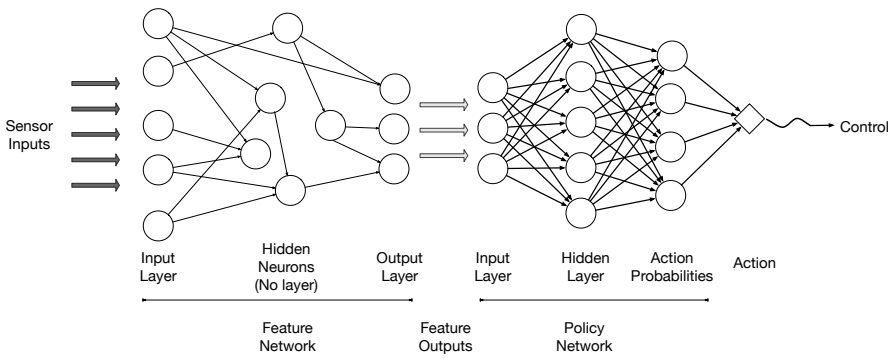


Figure 7.2: An Overview on the NN Architecture of NEAT+PGS.

A Neural Network Architecture

Figure 7.2 depicts the NN architecture to be adopted by NEAT+PGS for RL. As shown, the architecture features a sequential combination of one feature network and one policy network. At any time, the raw observation from the learning environment (i.e., current memory status of an Atari game) is provided first as the state input to the feature network which subsequently produces a group of high-level features to be further fed into the policy network. The policy network then generates stochastic action selection decisions (i.e., the probability of selecting each optional action) that an agent can use to determine one action which will be performed finally in the learning environment, resulting in a sampled transition to a new environment state. An instant reward is also generated as the immediate feedback for continued RL. In NEAT+PGS, the policy network has a fixed topology which will be determined before learning starts. The feature network, on the other hand, has the flexibility for its topology to be evolved by NEAT.

A Three-Stage Design for Feature and Policy Learning

To meet all of our design objectives, NEAT+PGS introduces three consecutive learning stages as briefly explained below:

(1) *Interim Policy Search Stage*: This is the first learning stage with the purpose of quickly pre-training a single interim policy network. This trained policy network establishes preliminary discrimination overall alternative actions that can be performed at any environment state based on a randomly created feature network. It presents a stable starting point and guides the subsequent evolution of feature networks in the second learning stage. Without the policy network pre-training, all high-level features produced by an evolved feature network cannot impose a clear preference of performing any desirable actions. The overall RL performance will be affected as a consequence.

(2) *NEAT based Feature Learning Stage*: This is the second learning stage, at the beginning of which the pre-trained interim policy network is distributed to a group of learning agents, each of which maintains a separate feature network. The population of such feature networks is further evolved through NEAT to

produce eventually one feature network, which can achieve the best learning performance upon using it together with the interim policy network. The high-level features created by this feature network is hence considered suitable for tackling the RL problem.

(3) *Policy Gradient Search Stage*: This is the third learning stage. At this stage, based on the best feature network evolved by NEAT, the policy network is trained again using a state-of-the-art PGS algorithm that is expected to achieve clearly better performance than both NEAT and PGS algorithms when the entire learning process finishes.

The three-stage design of NEAT+PGS enables complete separation of feature learning and policy learning, thereby realizing design objective (O1) mentioned previously. Meanwhile, all feature networks evolved by NEAT in the second learning stage share the same goal of enhancing the effectiveness of the same interim policy network. Therefore, by breaking the independence constraint among learning agents in NEAT+RAC, objective (O2) is fulfilled in NEAT+PGS. Moreover, in the third learning stage, we directly treat high-level features produced by the trained feature network as the state inputs to the policy network. This approach permits arbitrary PGS algorithms to be employed for continued training of the policy network and gives our learning scheme the highest flexibility of using many cutting-edge RL technologies, as required by objective (O3).

Algorithmic Description of NEAT+PGS

In this subsection, we present the high-level algorithmic description of NEAT+GPS in Algorithm 7.2.5, followed by detailed algorithmic descriptions of each learning stage in Algorithm 7.2.6 and Algorithm 7.2.7 respectively. In Algorithm 7.2.6, we present a general description of PGS algorithms which will be employed to train policy networks in the interim policy search stage as well as the policy gradient search stage. In Algorithm 7.2.7, we adopt the standard implementation of NEAT [206] for evolving feature networks.

Algorithm 7.2.5 NEAT+PGS

Require: an MDP $\langle \mathbb{S}, \mathbb{A}, \mathbb{P}, \mathbb{R}, \gamma \rangle$, n_1: the number of interim policy search iterations, n_2: the number of feature learning iterations (generations), n_3: the number of policy gradient search iterations, $\vec{\theta}_0$: the randomly initial policy parameters, p: the population size, $P[]$: a population of NNs

***Initialization*:**
  $\vec{\theta} \leftarrow \vec{\theta}_0$
  $P[] \leftarrow \texttt{INIT\_POPULATION}(\mathbb{S}, \mathbb{A}, p)$
***Interim Policy Search*:**
  $\phi_0 \leftarrow \texttt{RANDOM}(P[])$
  $\vec{\theta} \leftarrow \texttt{PGS}(\vec{\theta}, \phi_0, n_1)$ ▷ See Algorithm 7.2.6
***NEAT based Feature Learning*:**
  $\phi^* \leftarrow \texttt{EVOLVE}(\vec{\theta}, P[], p, n_2, m_n, m_l, m_w)$ ▷ See Algorithm 7.2.7
***Policy Gradient Search*:**
  $\vec{\theta}^* \leftarrow \texttt{PGS}(\vec{\theta}, \phi^*, n_3)$ ▷ See Algorithm 7.2.6
return $\vec{\theta}^*$, ϕ^*

Algorithm 7.2.6 Policy Gradient Search

Require: an MDP $\langle \mathbb{S}, \mathbb{A}, \mathbb{P}, \mathbb{R}, \gamma \rangle$, n: the number of learning iterations, T: the horizon, l: maximum length for one trajectory, M: the batch size, $\mathcal{T}$: sample repository

function $\texttt{PGS}(\vec{\theta}, \phi, n)$
  for $i = 1, 2, ..., n$ **do**
    for $t = 1, 2, ..., T/l$ **do**
      $\mathcal{T} \leftarrow \texttt{ROLLOUT}(\vec{\theta}, \phi)$ ▷ Store T samples into sample repository (See [117]).
    end for
    Update Policy Parameters $\vec{\theta}$ based on the principle of the chosen PGS algorithm with a batch of M samples from $\mathcal{T}$.
  end for
  return $\vec{\theta}^*$ ▷ Return a learned policy
end function

Algorithm 7.2.7 NEAT Feature Learning

Require: P: a population of learning agents, p: population size, m_n: add node mutation rate, m_l: add link mutation rate, m_w: weight mutation rate

Ensure: P: a reproduced population of learning agents

function EVOLVE($\vec{\theta}$, P, p, n_2, m_n, m_l, m_w)
  for $i = 1, 2, ..., n_2$ **do**
    $P'[]$ $\leftarrow$ new array of size p
    for $k = 1, 2, ..., p$ **do**
      $P'[k] \leftarrow$ BREED_NET($P[]$)
      if RANDOM() $< c$ **then**
        $P'[k], P'[k'] \leftarrow$ SELECT($P'[]$)
        $P'[k+1] \leftarrow$ CROSS_OVER($P'[k], P'[k']$)
      if RANDOM() $< m_n$ **then** ADD_NOTE($P'[k]$)
      if RANDOM() $< m_l$ **then** ADD_LINK($P'[k]$)
      if RANDOM() $< m_w$ **then** WEIGHTS_MUTATE($P'[k].\phi$)
      $P'[k].fitness \leftarrow$ ROLLOUT($\vec{\theta}, P'[k]$)
    end for
  end for
  return $P'[k^*]$ ▷ Return the fittest feature network
end function

7.3 Design of Experiments

This section provides general settings of the experiments for NEAT+RAC and NEAT+PGS respectively. We first discuss the experimental settings for NEAT+RAC including experiment setups as well as the design of the experiment in Section 7.3.1. Following the same structure, we discuss the experimental settings for NEAT+PGS in Section 7.3.2.

7.3.1 Experiments on NEAT+RAC

Experiment Setup

Here, we introduce experiment setups for evaluating NEAT+RAC. Firstly, we discuss the stochastic policy implementation for RAC algorithm. Secondly, we explain specifically the feature design for RAC. Lastly, we show the hyper-parameter configurations.

Stochastic Policy Implementation for RAC

We choose a Gaussian distribution to explicitly represent the stochastic policy used for NEAT+RAC, as the distribution has already been well-studied for coping with continuous problems [168]. Based on this policy, the action a to be taken at any state $\vec{s}$ is defined by the probability density given in (7.3). In (7.3), we have $\mu = \vec{\theta}^T \cdot \phi(\vec{s})$, and σ is a hyper-parameter used to control the level to explore new actions, which is fixed to 1.0 for all experiments. Note that, π at the RHS of (7.3) is the circumference ratio.

State Feature Design for RAC

To compare learning performances between NEAT+RAC and RAC so as to show the usefulness of features learned automatically through evolutionary algorithms, we implement the common *discretization technique* [126] for state feature design with respect to all benchmark problems. One important reason to choose discretized features is that of their simplicity, making the technique widely applicable to many continuous RL problems [212]. Meanwhile, it avoids providing too much domain knowledge to the learning algorithms so that we can focus on verifying the usefulness of NEAT to learn state features automatically.

The discretization technique we adopted in this chapter is so-called equal-width discretization [126]. Within the range of one dimension of the state input, we split it into n bins. For a given state input, a discretized feature vector will be a zero vector with a length of n expect that one dimension will be marked as one where the value of the dimension falls in the particular bin. This dis-

cretization will be performed on the rest of dimensions of the state input as well. Afterward, the obtained feature vectors are concatenated together to form the final feature vector representing the raw state input. For example, in Mountain Cart problem, the state input has two dimensions of which the position of the car $x \in [-1.2, 0.6]$ and the velocity of the car $\dot{x} \in [-0.07, 0.07]$. Let us suppose that the number of bins for each dimension are set equally to 5 and a given input $(-1.1, 0)$. Then, we can have a sub feature vector for the dimension x as $[1, 0, 0, 0, 0]$, and the sub feature vector for $\dot{x}$ as $[0, 0, 1, 0, 0]$. Accordingly, we can obtain the final feature vector for the given input as $[1, 0, 0, 0, 0, 0, 0, 1, 0, 0]$.

Hyper-Parameter Configuration

Some important hyper-parameter settings are summarized as follows. Firstly, for hyper-parameters of the NEAT and NEAT component of NEAT+RAC, we adopt identical settings reported in [232] where the effectiveness of NEAT with these settings has been verified on many benchmark problems. Secondly, we choose the commonly used settings ($\alpha = 0.1$, $\beta = 0.01$, $\gamma = 0.99$) reported in [31, 167] for RAC algorithm and RAC component of NEAT+RAC. These hyper-parameter settings are used for experiments on all benchmark problems presented in Table 7.1.

Table 7.1: The meta-parameter settings for NEAT+RAC across all experiments.

Algorithm	**Problems**	**Meta Parameters**			
		α	β	γ	z
NEAT+RAC	**Mountain Car**	0.1	0.01	0.99	20
	Cart Pole	0.1	0.01	0.99	20

Experiment Design for NEAT+RAC

For NEAT+RAC, driven by the second research goal in Section 7.1.1, we will conduct two different types of experiments. In the first type of experiment, we

evaluate the performance of NEAT+RAC on each benchmark problem to understand the effectiveness of the algorithm. The performance is measured by the number of steps for balancing the pole, or the number of steps to drive the car to a mountaintop. To identify any significant performance difference, for each benchmark problem, we conduct 30 independent runs for all algorithms. Among all runs, the population size and the number of generations are set to 100 and 100 respectively. For the Mountain Car problem, we perform 1000 learning episodes with 200 steps in each episode and 25 independent testing episodes where each has 200 testing steps. For the Cart Pole problem, we perform 200 learning steps in each of the 5000 learning episodes. Also, we conduct 1000 testing steps in each of 25 independent testing episodes. All independent tests are performed by the best learning agent of each generation.

In the second type of experiment, we intend to evaluate the usefulness of the state features on a relevant problem. We hypothesize that the learned useful state features can be reused on a related problem to improve the learning performance. For this purpose, in our feature evaluation, we firstly maintain the feature extractor of the current best learning agent by NEAT+RAC (i.e., the NN evolved by NEAT) on Cart Pole. Next, we modify the Cart Pole problem to obtain a relevant problem. The original Cart Pole problem starts at the same position $[0.0, 0.0, 0.0, 0.0]$ for every episode, whereas the modified version is set to start at random initial positions. Lastly, we adopt the RAC learning algorithm with the pre-trained feature extractor on the modified Car Pole problem to determine whether any noticeable improvement in learning performance can be witnessed.

7.3.2 Experiments on NEAT+PGS

Experiment Setup

Here, we present experiment settings for NEAT+PGS, including the network topology and hyper-parameters configuration.

Network Topology

The NN topologies of policy and feature networks for competing algorithms are given in Table 7.2. The table does not specify NEAT's topology which is actually automatically evolved. In addition, the topology of feature network in NEAT+PGS is also expected to be automatically evolved by NEAT. All feature networks are configured to output 32-dimension high-level features. We have examined three opinions: 32, 64 and 128. They do not appear to have any significant impact on final performance. PGS algorithms employed a single NN with 32 by 32 hidden neurons, the first hidden layer and the second layer can be considered as the feature network and policy network respectively. Note that, for all networks, we use the same activation function "RELU" that can help reduce the likelihood of vanishing gradients [73].

Table 7.2: Topology Setups for Policy Networks and Feature Networks.

Network Topology	**Layer**	**NEAT+A2C**	**NEAT+POWER**	**NEAT+TRPO**	**A2C**	**POWER**	**TRPO**
Feature Network Structure	Input	128	128	128	128	128	128
	Hidden	Evolved by NEAT			32	32	32
	Output	32	32	32	32	32	32
Policy Network Structure	Input	32	32	32	32	32	32
	Hidden	32	32	32	32	32	32
	Output	Number of Actions					

Hyper-Parameter Configuration

Hyper-parameter settings for all algorithms have been specified in Table 7.3. For all PGS algorithms, we follow the same settings in [189]. For NEAT, we follow the setting of a work published on GitHub [1] which has applied NEAT to some RAM-based Atari games with reasonably good performance.

Regarding the implementations, we adopt OpenAI-GYM [2] [39] that imple-

[1] https://github.com/HackerHouseYT/OpenAI-NEAT
[2] https://github.com/openai/gym

Table 7.3: Hyper-parameter Configurations for PGS algorithms.

Hyper-Parameters	**A2C**	**POWER**	**TRPO**
policy step size (α)	2.5×10^{-6}	2.5×10^{-6}	2.5×10^{-6}
discount factor (γ)	0.99	0.99	0.99
max size of sample repository	40,000	40,000	40,000
batch size (M)	10,000	10,000	10,000
max trajectory length (l)	10,000	10,000	10,000
interim policy training iterations (n_1)	1	1	1
policy gradient search iterations (n_3)	1,000	1,000	1,000
conjugate gradient iterations	10	-	-
GAE factor (λ)	0.95	-	-
optimization epoch (ϵ)	5	5	5

Table 7.4: Hyper-parameter Configurations for NEAT.

HyperParameters	**NEAT**
population size (p)	100
max generations (n_2)	100
add node rate (m_n)	0.02
add connection rate (m_l)	0.5
weight init mean	0.0
weight init std	1.0
weight mutate rate (m_w)	0.46
weight mutate power	0.825
activation function	relu
crossover rate (c)	0.1

ments ALE, NEAT-python [3] that implements NEAT, and rllab [4] [59] that implements PGS algorithms including A2C, POWER and TRPO.

[3]https://github.com/CodeReclaimers/neat-python
[4]https://github.com/rll/rllab

Experiment Design for NEAT+PGS

For NEAT+PGS, driven by the fourth research goal in Section 7.1.1, we choose seven algorithms for evaluation. Firstly, as NEAT+PGS is an improvement to NEAT, we choose NEAT as the baseline for comparison. Besides, HyperNEAT is not taken into account due to weak performance on RAM-based Atari Games in comparison to NEAT reported in [91]. Next, in order to examine the effectiveness of NEAT+PGS, we select three PGS algorithms, including A2C, POWER, and TRPO, for policy training in NEAT+PGS, resulting in three new algorithms: NEAT+A2C, NEAT+POWER, NEAT+TRPO. Note that, NEAT+Q can also be implemented by adopting our scheme of combining NEAT feature network and Q learning network. We do not include NEAT+Q in experiments because of two reasons: 1) Deep Q learning is reported to perform poorly in comparison to the above cutting-edge PGS methods in the literature [189, 56]. 2) In contrast to Q learning, feature learning is more critical to PGS methods because feature network is shared by both critic and actor. Thus, to investigate the effectiveness of feature learning for PGS algorithms, it is not necessary to further employ the learned features in the Q-learning algorithm. In addition, NEAT+RAC is not necessary to be included in experiments, as it cannot be used to train NNs with deep structures thus not suitable for our experiments. Other than this, we use A2C instead which can be viewed as an enhanced version of RAC.

In the experiments, we track learning performances in terms of average long-term cumulative rewards per episode that an algorithm can obtain after being trained on every 10,000 samples. In total, each algorithm has been trained for 10,000,000 samples. For NEAT as a population-based algorithm, we have put a counter to track the number of samples used for fitness evaluation. As long as the counter reaches a multiple of 10,000, we will select the best known NN in the current population, and its testing performance will be recorded. Regarding NEAT+PGS, we use a total of 10,000 samples in the interim policy search stage. In fact, we have experimentally found that more samples for pre-training the interim policy will not lead to better performance. Moreover, the purpose of this stage is only to let the interim policy distinguish different actions. Thus, we do not require a great number of samples here. Furthermore,

Table 7.5: The final episode performance comparison of two algorithms (i.e., NEAT, and NEAT-RAC-PGS) on two benchmark problems (i.e., Cart Pole and Mountain Car).

Algorithms/Problems	Cart Pole	Mountain Car
NEAT-RAC-PGS	**1000.00±0.00**	**128.82±9.98**
NEAT	85.60±13.41	135.68±11.19

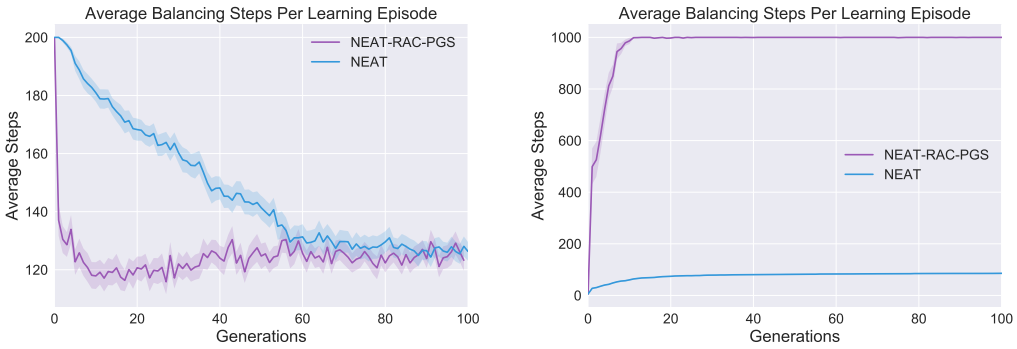
in the NEAT base feature learning stage, we allow exact 2,000,000 samples to be used for evolving good feature networks. Lastly, we will consume another 7,990,000 samples to train policy network in the PGS stage.

7.4 Results and Discussion

This section presents and discusses the experimental results for both NEAT+RAC and NEAT+PGS respectively. To achieve the first two research goals stated in Section 7.1.1, we firstly show the results of NEAT+RAC to analyze the effectiveness of the algorithm as well as the usefulness of feature extracted by NEAT feature extraction process. Next, to achieve the last two research goals in Section 7.1.1, we focus on the results of NEAT+PGS to compare with NEAT, A2C, TRPO and PoWER regarding learning effectiveness and sample efficiency.

7.4.1 Discussion on Results of NEAT+RAC

The experimental results of NEAT+RAC are presented and analyzed in this subsection. We first analyze the learning effectiveness of NEAT+RAC contrary to NEAT on the two benchmark problems. Next, we discuss the usefulness of learned features by NEAT+RAC.



(a) Mountain Car (b) Cart Pole

Figure 7.3: The comparison of learning performance of NEAT+RAC and NEAT on two benchmark problems: (a) displays the averaging steps to reach the goal region on Mountain Car (the smaller the better), (b) displays the averaging steps to balance the pole to the upright position on Cart Pole (the larger the better).

Learning Effectiveness Evaluation

To evaluate the learning effectiveness, we present the learning performances obtained by NEAT+RAC and NEAT on the Mountain Car problem in Figure 7.3(a) and on the Cart Pole problem in Figure 7.3(b) respectively.

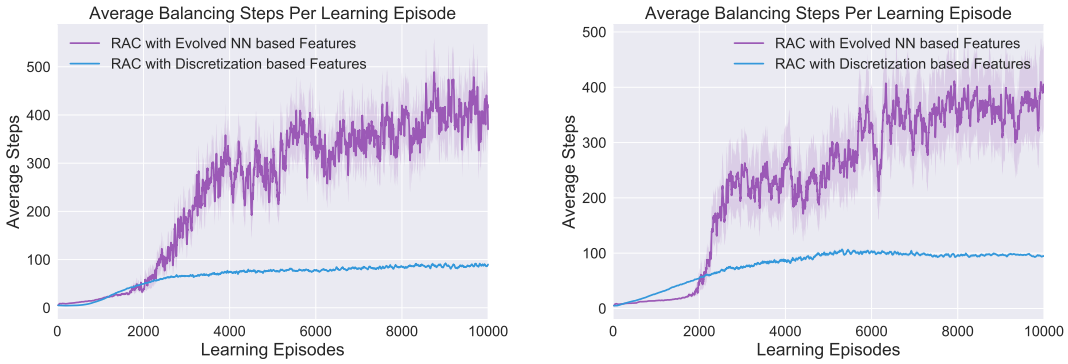
As shown in Figure 7.3(a), after a certain period, both NEAT+RAC and NEAT achieve reasonably good performance (around 120 steps), which is consistent with the results of NEAT reported in [232]. The two high peaks of NEAT+RAC do not imply that it sometimes performs worse than NEAT, as no significance can be found after the 53rd generation according to a statistical test. Meanwhile, owing to the capability of NEAT+RAC to make more effective use of learned features, it eventually achieved better performance than NEAT.

Figure 7.3(b) evidently shows that NEAT+RAC outperforms NEAT with averaging 1000 steps after the 10th generation. As a matter of fact, NEAT achieves the desirable performance (approximately 100 steps) according to the OpenAI GYM benchmark [79]. Without comparing it with NEAT+RAC, NEAT should be treated as an effective algorithm.

Table 7.6: The final episode performance comparison of two algorithms (i.e., NEAT, and NEAT-RAC-PGS) on two benchmark problems (i.e., Cart Pole and Mountain Car).

Algorithms/Problems	Cart Pole	Modified Cart Pole
RAC with Evolved NN based Features	**311.86±407.35**	**399.34±467.98**
RAC without Evolved NN based Features	89.26±22.04	94.46±11.91

Feature Usefulness Evaluation



(a) Cart Pole

(b) Modified Cart Pole

Figure 7.4: The comparison of learning performance of RAC with two different feature extractors (an evolved NN feature extractor and a predefined discretized feature extractor) on the two related problems (see Section 7.3.1): (a) displays learning performances obtained on the standard Cart Pole problem, (b) displays learning performances obtained on the modified Cart Pole problem.

To evaluate the usefulness of features, in Figure 7.4, we compare the learning performances of RAC with two different feature extractors on the standard Cart Pole problem and the modified Cart Pole problem respectively. The first feature extractor is a learned NN by the NEAC component of a NEAT+RAC learning agent on the standard Cart Pole with the highest fitness value. The second feature extractor is the widely-used discretization feature extractor where each dimension of the environment state is discretized into 20 bins as described

in [212].

Figure 7.4(a) reveals that, on the standard Cart Pole problem, RAC with evolved NN features performs significantly and consistently better than RAC with discretized features after the 2370th generation. On the other hand, when the learning environment is changed to the modified Cart Pole problem, we are still able to find a similar observation in Figure 7.4(b) that RAC with evolved NN features outperforms RAC with discretized features after the 2214th generations. Statistical tests reject the null hypothesis after the time points. The RAC with discretized features achieves averaging 100 steps on both problems, which can be regarded as effective as reported in [166]. These results suggest that the features learned by NEAT+RAC on one problem can not only be used to solve the original problem, but also be employed to solve similar but different problems.

Nevertheless, on both problems, the RAC with evolved NN features appears to fluctuate. Actually, the step-based learning process of NNs has already been reported unstable in the literature [81], because step-based learning always picks up the recent state transitions for estimating gradients, which may bring bias into gradient estimation resulting in the unstable updating. A possible solution to address the issue is the use of experience replay, but this is not the focus of this thesis, which will be investigated in the future.

The reason that evolved NN features are superior to discretized features is given below. Firstly, the real-world environment (e.g., Cart Pole) often exhibits high non-linearity, NN is well known as suitable non-linear models in comparison to other models. Secondly, the raw environment input is continuous, some essential information, that is useful for finding good policies, may be lost during the discretization. On the other hand, NN is capable of smoothly producing continuous values as high-level features. More importantly, NEAT can evolve both weights and structures for NNs, which further provides higher chances to find suitable state features.

Table 7.7: The final episode performance comparison of seven algorithms (i.e., NEAT, NEAT+A2C, NEAT+POWER, NEAT+TRPO, A2C, POWER and TRPO) on six Atari games (i.e., Asteroids, Breakout, Freeway, Seaquest, SpaceInvaders, and TimePilot).

Algorithms/Problems	Asteroids	Breakout	Enduro	Freeway	Seaquest	SpaceInvaders	TimePilot
A2C	**3295.74±112.59**	8.48±1.22	0.01±0.01	3.53±3.59	452.29±145.01	740.16±48.70	8502.00±125.96
NEAT	1253.75±498.32	2.89±2.99	27.67±8.69	21.44±3.50	216.11±97.84	397.78±224.94	4041.25±1686.53
NEAT+A2C	3162.75±255.49	**9.25±0.34**	24.72±15.32	**26.45±1.63**	719.80±49.71	**784.72±25.49**	9143.50±430.04
NEAT+POWER	2977.58±342.94	7.71±1.17	27.49±17.42	26.12±1.29	**737.75±28.31**	703.50±48.70	9618.00±776.70
NEAT+TRPO	3139.31±186.63	8.11±1.32	**31.38±21.50**	26.34±1.28	733.88±54.43	744.48±47.00	9225.00±802.58
POWER	2783.25±216.33	9.13±0.99	0.00±0.00	3.80±5.79	387.62±135.22	736.06±35.94	8667.92±145.77
TRPO	2940.38±263.73	8.78±1.09	0.32±1.10	5.71±6.13	360.78±138.78	705.16±60.39	8668.00±273.24

7.4.2 Discussion on Results of NEAT+PGS

In this subsection, we are interested in a discussion on the results of NEAT+PGS. At first, we present the learning effectiveness evaluations of NEAT+PGS contrary to NEAT, A2C, TRPO and PoWER. Afterward, we analyze the sample efficiency of NEAT+PGS in particular comparison to the state-of-the-art PGS algorithms.

Evaluations on Learning Effectiveness

We depict learning curves for all seven algorithms on six different Atari games in Figure 7.5. We can see that NEAT+PGS (including NEAT+A2C, NEAT+POWER, NEAT+TRPO) is generally more effective than other competing algorithms especially NEAT on all six Atari games. In particular, NEAT+PGS apparently outperforms all competing algorithms on Seaquest. On Asteroids and TimePilot, after using the trained feature networks after 2,000,000 samples, NEAT+PGS can clearly learn faster than PGS algorithms that use predefined policy networks, and eventually, surpass these algorithms.

Interestingly, we have also found that, on Freeway, the cutting-edge PGS algorithms have all failed. This result is agreeable with recent findings in [191]. On the other hand, NEAT+PGS manages to perform well and eventually surpass NEAT. This evidence further shows the importance of using NEAT for fea-

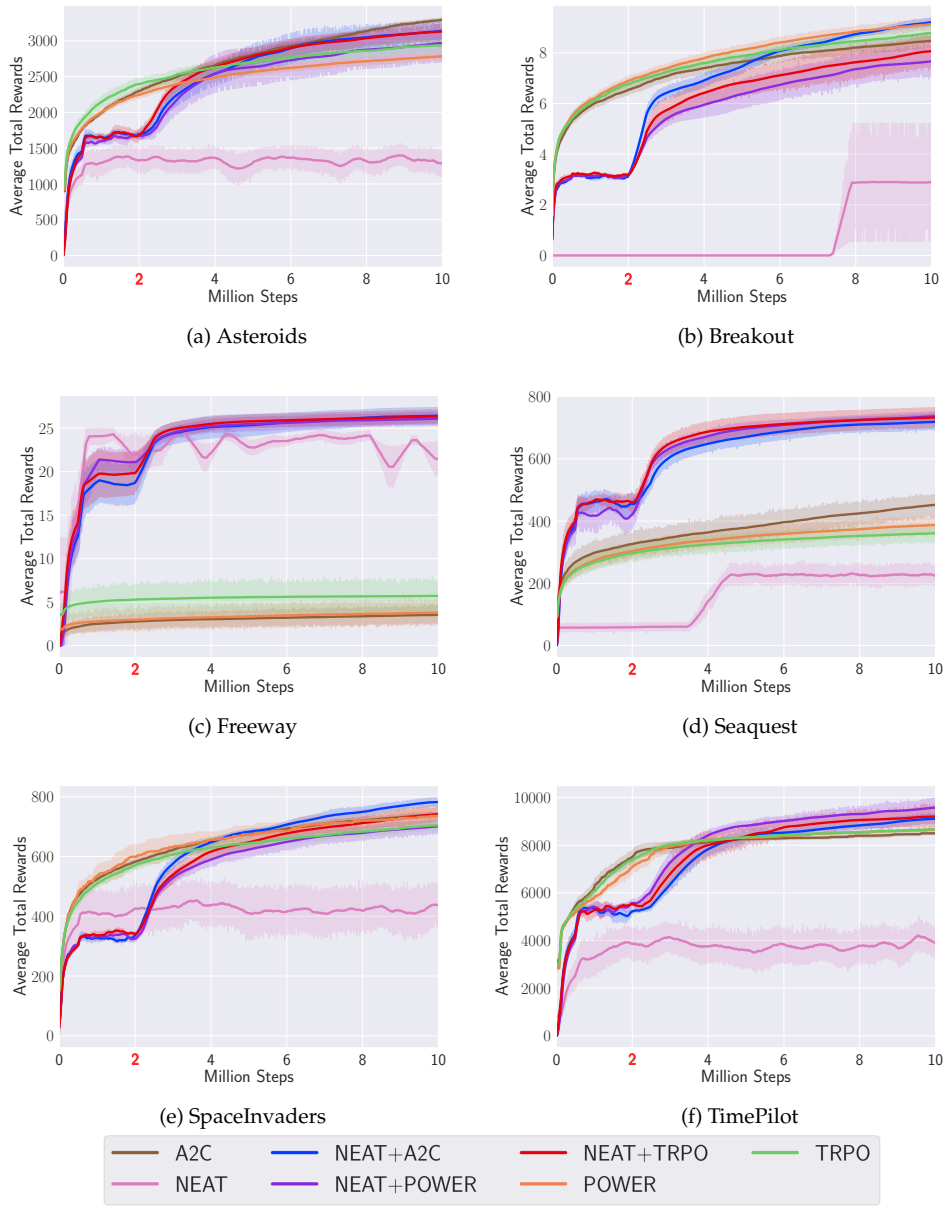


Figure 7.5: Average rewards per 10,000 steps obtained by NEAT, NEAT+A2C, NEAT+POWER, NEAT+TRPO, A2C, POWER and TRPO on six Atari games, including Asteroids, Breakout, Freeway, Seaquest, SpaceInvaders, and TimePilot. As being highlighted with red color, for NEAT+PGS, the NEAT based feature learning stage stops at two million steps (i.e., 2,000,000 samples).

ture learning. In addition, NEAT+PGS algorithms performed competitively as PGS algorithms on Breakout. For NEAT+PGS, we found this is because that NEAT cannot find useful feature networks for the problem for only learning over 2,000,000 samples. In fact, NEAT itself also performed poorly on this game for learning 10,000,000 samples. Even though, NEAT+PGS can still manage to achieve similar or even slightly better performance than PGS, indicating the strong resilience of NEAT+PGS on low-quality feature networks.

Note that, we are aware of that NEAT can actually achieve state-of-the-art performance on some Atari games as reported in [91, 38]. However, the purpose of developing NEAT+PGS is not to make the best scores on different games but to understand the effectiveness of feature learning.

Based on these findings, we can conclude that NEAT+PGS is a generally effective approach. In NEAT+PGS, feature learning and policy learning are mutually supplementary processes. Particularly, even one process fails on a problem, and the other process can cope and ensure final good performance. When both processes are effective, NEAT+PGS can achieve clearly better performance.

Analysis on Sample Efficiency

Algorithm	Asteroids	Breakout	Freeway	Seaquest	SpaceInvaders	TimePilot
NEAT+A2C	246.44 ± 30.05	0.69 ± 0.04	**2.39 ± 0.27**	**62.75 ± 5.96**	**61.99 ± 3.03**	756.26 ± 51.60
NEAT+POWER	**257.79 ± 24.42**	0.59 ± 0.10	**2.42 ± 0.18**	**63.95 ± 4.81**	57.03 ± 4.95	**800.66 ± 62.90**
NEAT+TRPO	**258.38 ± 20.80**	0.62 ± 0.10	**2.43 ± 0.22**	**64.96 ± 6.21**	59.37 ± 4.43	773.14 ± 77.21
NEAT	131.82 ± 47.85	0.02 ± 0.03	2.28 ± 0.25	16.33 ± 5.96	42.04 ± 22.62	370.95 ± 205.36
A2C	246.52 ± 28.37	**0.73 ± 0.11**	0.30 ± 0.32	37.17 ± 15.80	**64.62 ± 4.96**	**779.89 ± 29.44**
POWER	245.75 ± 21.10	**0.78 ± 0.09**	0.33 ± 0.55	33.63 ± 13.37	**64.83 ± 4.58**	778.77 ± 23.58
TRPO	**259.85 ± 20.80**	**0.76 ± 0.10**	0.54 ± 0.61	32.08 ± 13.18	61.96 ± 5.74	**789.64 ± 26.32**

Table 7.8: Sample Efficiency comparison of NEAT+PGS against NEAT, A2C, POWER and TRPO.

Sample efficiency can be understood as the learning speed of an algorithm which is measured in terms of the number of samples used for achieving good performance. Given this, we choose the following learning speed metric to mea-

sure sample efficiency, i.e.,

$$\text{Score} = \sum_{t=0}^{1000} \frac{1}{10000} R_t, \tag{7.6}$$

where t is the tracking point (i.e., 10,000 samples), R_t is the average total reward obtained after learning 10,000 samples across different trials. With the metric, higher learning speed implies higher sample efficiency. Note that, the metric can be intuitively interpreted as the area under the curve.

In Table 7.8, we present the sample efficiency comparisons measured by using (7.6), and top three results for each game are highlighted. Evidently, NEAT+PGS algorithms achieved highlighted results in most cases. For cases on Freeway and Seaquest, the superiority of NEAT+PGS to PGS algorithms is observably significant. For other cases, NEAT+PGS achieves competitive results to those PGS Baselines (even on Breakout). These findings indicate that NEAT+PGS can be more sample efficient than NEAT and PGS algorithms on RAM-based Atari games.

7.5 Chapter Summary

In this chapter, we have the primary goal to develop new techniques with evolutionary feature learning to enhance the effectiveness of existing PGS algorithms. We have successfully achieved the goal by developing two new PGS algorithms enhanced by automated feature learning based on NEAT. The first algorithm is called NEAT+RAC, which is designed to integrate a popular PGS algorithm, i.e., RAC, with NEAT for automated feature extraction. The second algorithm is called NEAT+PGS, where NEAT based feature learning is designed to support a newly proposed three-state scheme that enables effective knowledge sharing among multiple concurrently learning agents. Moreover, NEAT+PGS generalizes NEAT+RAC by replacing the policy search component with modern PGS methods that support policies with deep structures.

The development of NEAT+RAC in this chapter showed that the integration between RAC and NEAT brings two main advantages. First, different

from most existing approaches that only concentrate on automated feature extraction, the proposed NEAT+RAC algorithm can simultaneously find better policies for the RL problems resulting in improved sample efficiency. Second, compared to traditional policy search methods, NEAT+RAC can identify useful state features that can be further reused in related learning problems. Experiment results confirmed both of the two advantages of the proposed algorithm. This work opens new possibilities of exploiting other cutting-edge AC-PGS algorithms based on the same design principle of NEAT+RAC.

However, we also find that NEAT+RAC may fail to perform effectively on large-scale problems due to two reasons. First, the original RAC is designed to train simple policies represented as linear functions of policy parameters. They can easily lose effectiveness while encountering complicated problems that require non-linear and powerful models with deep structures. Second, in the original design of NEAT+RAC, feature learning and policy learning are heavily mingled in a single process, preventing easy sharing of learned knowledge across multiple agents. These issues become major obstacles hindering the learning effectiveness as well as the sample efficiency of NEAT+RAC when solving large-scale problems.

Motivated by the limitations of NEAT+RAC for large-scale RL problems, we have developed NEAT+PGS that seamlessly integrates NEAT based feature learning and advanced PGS algorithms. In comparison to NEAT+RAC, this scheme possesses two major advantages. First, a clear separation between feature learning and policy learning is realized in NEAT+RAC for effective knowledge sharing and learning among agents so as to significantly improve the sample efficiency. Second, a promising way of integrating NEAT with various cutting-edge PGS algorithms is developed to enable effective training of deep policy networks on large-scale RL problems. The experimental results also confirmed these advantages for NEAT+PGS.

In the future, we will further investigate the reliability of NEAT+PGS to hyper-parameter settings. It is also interesting to investigate the effectiveness of using other suitable evolutionary methods for training feature networks. Possible methods can be HyperNEAT, Evolutionary Strategies (Es), Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) and so forth. These methods

may be more suitable for IMAGE-based Atari game playing tasks rather than RAM-based Atari game playing tasks. Additionally, we can explore possibilities of using other advanced PGS algorithms such as Actor-Critic with Experience Replay (ACER) [229], Actor-Critic using Kronecker-Factored Trust Region (ACKTR) [240], and Proximal Policy Optimization (PPO) [191]. By using these techniques as mentioned earlier, we are expecting to tackle IMAGE-based Atari games.

Chapter 8

Conclusions

In the thesis, we have successfully achieved our overall goal, which is to develop new *effective* Policy Direct Search (PDS) algorithms to tackle difficult RL problems through different techniques. Firstly, the goal was to utilize historical gradients for accurate policy gradient estimation through Primal-Dual Approximation technique. Next, it was to enhance Covariance Matrix Adaption Evolutionary Strategy (CMA-ES) based global search, and Proximal Policy Optimization (PPO) based local search for better balancing between exploration and exploitation. Thirdly, it aimed at, for PDS, stabilizing value function learning through Sandpile Model and generalizing the compatible condition to support flexible value function learning through q-logarithm. Lastly, it was to enhance a variety of PDS algorithms through seamless integration with NeuroEvolution based automated feature learning. Based on the understanding of general RL obtained from an intensive review, we have decomposed this central objective into four sub-objectives corresponding to four research questions:

(Q1) How can we improve learning effectiveness of step-wise learning based Policy Gradient Search on solving difficult control problems by better utilizing historical step gradients?

(Q2) How can we achieve state-of-the-art performance by better balancing the exploration-exploitation trade-off with the use of Evolutionary Algorithms and PGS algorithms?

(Q3) How can we enable a more reliable and accurate value function learning to further boost PGS for tackling difficult RL problems?

(Q4) How can we effectively find useful features via automated feature learning to facilitate PDS on particular RL problems?

To answer these questions, we first have investigated the challenges/limitations of existing PGS algorithms. By utilizing different techniques, we have developed eight new PGS algorithms in the thesis that are capable of addressing the challenges/limitations. We have empirically evaluated the proposed new algorithms in comparison to the related PDS algorithms on a range of RL tasks from the control/locomotion benchmarks to the Atari game playing tasks.

The remainder of the chapter first summarizes the major conclusions drawn from the theoretical and empirical evidence presented in preceding contribution chapters. Next, it examines the possible directions for future work.

8.1 Major Conclusions

Through substantial improvement of several existing PDS algorithms, this thesis has achieved (or even surpassed) the state-of-the-art performance on several benchmark problems. More specifically, Chapter 4 demonstrates the proper utilization of historical gradients enables more accurate estimation of policy gradients and hence significantly improves the efficacy of PGS. Following that, Chapter 5 further shows that, with the support of EAs, cutting-edge PGS can better balance the exploration-exploitation to achieve state-of-the-art performances. Next, Chapter 6 shows that the effectiveness of traditional Actor-Critic algorithms can be substantially enhanced through stabilizing the critic (i.e., value function) learning and generalizing compatible features. Finally, Chapter 7 provides pieces of evidence that the learning performance of existing PGS algorithms can be further improved by using useful features automatically extracted from raw state inputs based on an evolutionary feature learning process.

8.1.1 Effective Policy Direct Search through Primal-Dual Approximation

Chapter 4 has successfully answered the research question Q(1) and addressed the research objective O(1) stated in Chapter 1 by proposing three Primal-Dual Approximation (PDA) based PGS algorithms. The proposed algorithms improve the learning effectiveness of traditional step-wise learning based policy gradient search algorithms. The improvement is achieved by adopting the primal-dual approximation technique to obtain more precise policy gradient estimation with the support of cumulative historical gradients.

With the development of the algorithms in the chapter, we have obtained the following three findings:

- We have demonstrated that our PDA based PGS is a generalized framework for step learning based PGS algorithms. Based on this new framework, new learning rules have been successfully developed to transform three existing step-wise PGS algorithms (i.e., Regular Actor-Critic, Natural Actor-Critic with Fisher Information Matrix and Natural Actor-Critic with Advantage Parameters) into their respective new counterparts (i.e., Dual Regular Actor-Critic, Dual Natural Actor-Critic with Fisher Information Matrix and Dual Natural Actor-Critic with Advantage Parameters).

- We have theoretically proved that all three PDA based PGS algorithms can eventually converge under suitable conditions. The proof extends the coverage of convergence guarantee to algorithms that rely on historical gradients, to the best of our knowledge, which has never been explored so far in the literature.

- We have empirically discovered that 1) PGS on linear parametric policies with step learning strategy can be equivalently effective but more sample efficient in comparison to episodic learning strategy based PGS algorithms, such as PPO with linear policies; 2) the proposed PDA based PGS algorithms can perform significantly better than their original counterparts.

8.1.2 Proximal Evolutionary Strategy for Sample Efficient Policy Direct Search

Chapter 5 has successfully achieved the state-of-the-art performance as questioned in Q(2) and the objective O(2) in Chapter 1 by developing an EA (i.e., Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES)) based deep policy search algorithm named Proximal Evolutionary Strategy (PES). In comparison to traditional PGS algorithms, the proposed algorithm has made three technical improvements. It firstly adopts a layer-wise training strategy for CMA-ES to improve time/computation efficiency for training large Deep Neural Networks (DNNs). Secondly, its fitness evaluation is constructed on a proximal performance lower bound based surrogate model that significantly reduces the sample cost. Thirdly, it incorporates a new gradient-based local search technique to improve the effectiveness of the evolutionary policy search.

Through developing the PES algorithm, we have discovered the following three findings:

- We have empirically shown that layer-wise learning can significantly reduce the computational cost for policy search without sacrificing on performance. The finding justifies the first time in literature the possibility of using CMA-ES to train large-scale deeply-layered networks.
- We have also found that PES can be more sample efficient than the advanced Evolutionary Algorithms, such as CMA-ES itself, OpenAI-ES, and Uber-GA.
- We have demonstrated that local search enhanced learning (PES) can further boost the learning effectiveness by fine-tuning the best solution evolved by CMA-ES global search.

8.1.3 Reliable and Flexible Value Function Learning for Policy Gradient Search

In Chapter 6, the research question Q(3) and the research objective O(3) in Chapter 1 have been successfully addressed by developing two new Actor-Critic

(AC) algorithms. In the first algorithm, we adopted a so-called Sandpile Model (SM) with self-organizing behaviors into PGS to stabilize the critic learning. With the SM, the value function can be learned in a self-adaptive manner that prevents the learning from divergence. In the second algorithm, we generalized the logarithm function used for constructing the compatible functions resulting in flexible policy gradient estimation and effective PGS. Particularly, with the generalization, we provide a flexible family of compatible functions which can be applied to any PGS that follows the Policy Gradient Theorem (PGT).

By developing the two new algorithms, we have identified the following three findings:

- We have demonstrated that the Sandpile Model enables self-adaptive and stable learning of value functions. Reliable critic learning can further improve the learning effectiveness of policy learning on a commonly used PGS algorithm such as RAC.

- We have also identified and quantified, the strong relationship between the reliability of critic learning and the effectiveness of policy learning. This finding is essential for the future development of effective powerful AC algorithms.

- We have experimentally discovered that the learning effectiveness of PGS with generalized compatible functions can be noticeable, under suitable conditions, improved in comparison to algorithms without using generalized compatible functions. This finding shows that any PGS algorithms that rely on standard compatible functions for critic learning can benefit from our generalized new family of compatible functions.

8.1.4 Enhancing Policy Direct Search via Automated Evolutionary Feature Learning

Chapter 7 has successfully addressed the research question Q(4) and achieved the research objective O(4) in Chapter 1 by developing two new PGS algorithms seamlessly integrated with a NeuroEvolution based feature learning. In

the chapter, we first developed a PGS algorithm based on the commonly used RAC algorithm by adopting a NEAT based feature extraction process, which is named NEAT+RAC. Following that, NEAT+RAC is further generalized to various PGS algorithm (i.e., NEAT+PGS) with better sample efficiency and learning effectiveness. This is achieved by clearly separating feature learning from policy learning.

With the development of NEAT+RAC and NEAT+PGS, we have obtained the following findings:

- With the development of NEAT+RAC, we have experimentally demonstrated that NeuroEvolution, as exemplified by NEAT, is capable of automating feature learning to extract useful features. Based on the extracted features, the effectiveness of RAC can be further improved on various complex RL problems.

- With the development of NEAT+PGS, we have found that, with the newly designed three-stage learning scheme, NEAT+PGS can separate feature learning from policy learning and achieves effective knowledge sharing and learning across multiple parallel learning agents.

- With the development of NEAT+PGS, we have also empirically demonstrated that the learning effectiveness can be further improved by integrating with various PGS algorithms capable of training deep policy networks.

8.2 Limitations

Although the thesis has taken great efforts to address some issues existing in the current literature, we acknowledged some potential limitations which could be addressed in future research. Here, we identified that three aspects can be further improved, i.e., manual network architecture, trial-and-error based hyperparameter tuning, and pre-defined reward settings.

8.2.1 Manual Network Architecture

In this thesis, we have adopted commonly used architectures in the literature for both policy and value function to maintain fair comparisons against baseline algorithms. This may limit the true effectiveness of the algorithms. It is widely accepted that network architecture plays an important role to achieve effective RL. For instance, Islam et al. [105] showed that policy network architecture can significantly impact results in different PGS algorithms, such as TRPO and DDPG. However, the architectures used in our research is manually designed, which may not be able to show the true effectiveness of the algorithms.

8.2.2 Trial-and-Error based Hyper-parameter Tunning

The experiments in the thesis have followed the same conventional hyper-parameter settings as reported in related literature whose algorithms are used as baseline algorithms. The hyper-parameter settings also play a significant role in eliciting the best performance of an RL algorithm. As reported in [96], the performance of algorithms can be diminished due to poor hyper-parameter settings, and such settings are often inconsistent in related literature. Though we have taken efforts to apply trial-and-error approaches to test a range of settings before applying any algorithms, it still has two limitations with regard to hyper-parameter settings. First, it remains difficult to determine whether the range we selected is appropriate and has good coverage of suitable settings. Second, trial-and-error approaches have been widely accepted as a tedious process.

8.2.3 Pre-defined Reward Settings

Another limitation of the thesis appears to be the pre-defined settings of reward schemes for the benchmark problems used in our experiments. Reward settings have already been shown as a key factor that may significantly affect the experimental results as reported in [59, 96], and a well-learned reward function can help improve the actual performance of the algorithms [212, 96]. However, it can be biased to pre-define a reward function which may not cover the true

landscape of the reward scheme of the environment.

8.3 Future Work

Though we have successfully answered the research questions raised in the thesis, the thesis apparently could not cover all the possible research directions that deserve further studies. In this section, we intend to discuss the possible future work following three directions, namely (1) to improve learning effectiveness by combining improvements developed in the thesis, (2) to improve the effectiveness by wisely combining model-based PDS with model-free PDS, (3) to improve the effectiveness by developing transferable policies, transferable value functions or other transferable key elements in RL, (4) to improve the effectiveness by automating network architecture design and hyper-parameter tuning, and (5) to improve both learning effectiveness and computational efficiency by leveraging the cooperative co-evolutionary techniques.

8.3.1 Combining Improvements for Policy Direct Search

The thesis opens many possible directions for improving the effectiveness of PDS for additional research. Besides exploring deeply into a single direction, the combination across different directions may also be an effective way to achieve the primary goal of our research. Combining different improvements for one particular approach to improving its effectiveness is not new, for example, the Rainbow algorithm proposed by Hessel et al. [98] combines six different extensions of the DQN algorithm and empirically shows the significant improvement on effectiveness. Inspired by this, we can expect the additional improvements in the learning effectiveness of existing PDS algorithms by combining improvements achieved in the thesis.

In the thesis, we can summarize the technical contributions for boosting PDS from four aspects:

- In PDA based PGS, We have adopted the PDA technique to construct effective policy gradient estimations.

- In PES, we have used CMA-ES as global search in conjunction with PPO based local search to better balance exploration-exploitation trade-off.
- In NEAT+PGS, we have employed NEAT based feature learning in combination with PGS algorithms to automatically extract useful features so as to further better promote policy search.
- In SM-RAC, we have stabilized value function learning to enable effective policy learning, and have further generalized compatible functions in GCFA-RAC to obtain accurate policy gradient estimations.

Some of the improvements listed above can be combined effectively. For example, NEAT+PGS can be integrated with PDA based PGS or PES algorithms to extract useful features as well as to find good policies. Besides, GCFA-RAC can be generalized to all PGS to achieve better effectiveness. Furthermore, PES can be combined with PDA based PGS to encourage exploration and retain active exploitation.

Though it seems straightforward to conduct the combinations across different improvements directly in theory, there remain many challenges in practice. For instance, the formulations of PDA based PGS are derived based on linear represented policies while being trained via step-wise learning, and therefore their direct applicability to deep policies is questionable. In consequence, the combination of PDA based PGS and PES may not be accessible as expected. Therefore, the study of how to properly combine these improvements and address different challenges raised during the combination can be a good direction to be further explored.

8.3.2 Model-based vs. Model-free

Sample inefficiency is one of the severe obsessions that hinders the practical usage of model-free reinforcement learning approaches. The key reason is that model-free approaches learn from the sampled data with emphasis on the reward scalar but overlooking other important information such as state transitions. The issue can be solved by model-based approaches, which fully use the sampled data to construct an approximate model of the environment. They

learn the transition function as well as the reward function that is critical to an MDP. Given the MDP model, one can either learn a value function via dynamic programming [52] or improve policy by analytically computing the policy gradients rather than inaccurately estimating the policy gradients based on newly collected environment samples [53].

However, model-based approaches also have limitations. First, they require substantial computational costs on constructing the model, and hence their applicabilities usually are restricted to finite and small MDPs [52]. Second, the goodness of the policy found by model-based approaches may be suboptimal due to the biases in the learned model.

These issues can be addressed effectively by developing hybrid methods that can effectively leverage the benefits of both model-based and model-free approaches. Some recent work has already started some studies in this direction, such as [19, 175]. However, these works either have only shown preliminary results on simple RL problems or have only focused on VIS as the model-free part. Hence, there remains a large room for investigating how to bridge the gap between model-based and model-free approaches, particularly model-based PDS and model-free PDS.

8.3.3 Transfer Learning for Policy Direct Search

In Machine Learning, there is another promising sub-field called Transfer Learning (TL). TL aims to develop methods capable of transferring/reusing the knowledge learned from a set of source tasks onto a target task [46]. With the transfer methods, it is expected to improve the learning performance on the target task. The success of TL has already been widely witnessed in many Supervised Learning tasks, such as recommendation systems, classification tasks, and so forth [162, 203].

Many research work [181, 100, 203, 46, 124] have already studied the adoption of TL into RL in order to solve difficult RL tasks. These works have shown that many components in RL, such as policy parameters, state features, or reward functions, can be learned via transfer learning rather than from scratch [181, 100, 203, 46, 124, 236]. In this way, RL algorithms can either have

a good starting point on a new task or reduce sample cost which can be very expensive in some practical circumstances [236].

In the future, it can be foreseen that some of the algorithmic components of RL discussed in this thesis can be further improved by using TL. For example, the transferability of the reusable state features developed in Chapter 7 can be further investigated. In addition, other algorithmic components improved in our PGS algorithms, such as policy parameters trained by PES (see Chapter 5) or value functions learned from SM-RAC (see Chapter 6), can also be learned via TL. This understanding creates many possibilities for future research.

8.3.4 Automated Network Architecture Design

It is clear that network architecture has important impacts on the effectiveness of NN model based learning as explained in [92, 230, 97]. It is agreed that significant architecture engineering is required for developing NN models [174, 137]. Recent research interests in ML domain mainly focus on designing new Network Architecture Search (NAS) algorithms for automatically designing CNN architecture to solve computer vision tasks such as as [174, 137]. Only a few research work such as [230] briefly touched the impact of architecture design on the learning effectiveness on solving RL problems. This implies that the topic can be an interesting and potential research direction worth further investigation.

In this thesis, to maintain a fair comparison to our competing algorithms, we have used a 2-layer architecture where each layer contains 32 neurons as conventionally defined in [191, 189, 135]. The NN model with this architecture can be sufficient enough to solve the testbeds used in this thesis. However, it is not clear whether or not this architecture provides the most effective model. Apparently, the trial-and-error process for the tunning architecture of an RL model can be very time-consuming and sample inefficient, as each RL model requires massive interactions to learn to be effective. A possible solution can be explored for this problem is the combination of TL and NAS. More specifically, we can search the architecture for an RL model on a simple task, and apply it to a complex task with the help of TL. In this way, learning effectiveness is

expected to be improved.

8.3.5 Automated Hyper-parameter Tunning

The right choices of hyper-parameters can result in the improvement of learning performance [96]. Traditionally, such hyper-parameters are either tuned manually or via a trial-and-error process, which has been proven a tedious and error-prone process[188]. Recent research has shown that such tunning process can be automated by using meta-learning concepts [145, 188]. These work has successfully demonstrated that automated hyper-parameter tunning can further improve the effectiveness of solving RL problems.

Given this, to address the limitations of this thesis, we can first apply meta-learning to learn hyper-parameters for our proposed algorithms. More specifically, we can define the meta-learning process independent to the RL process as another optimization process as evidenced by in [60]. Different from [60] where the optimization process is defined as another RL process, we can use EC approaches instead. Besides, we can also adopt Bayesian Optimization to avoid trial-and-error approaches to tune the hyper-parameters. Bayesian Optimization has recently applied to machine learning hyper-parameters tunning, which achieves better performance while requiring fewer iterations than other methods such as random search.

8.3.6 Cooperative Co-evolution for Policy Direct Search

Cooperative co-evolution is one type of evolutionary algorithms where different species represent solution components mutually benefits each other. It aims at reducing the problem complexity by modularizing each component. By recombining other components that are beneficial for solving the task, each component is evolved toward a complete solution via measuring its contribution. Many works have shown that cooperative co-evolutionary approach can outperform single population algorithms in many situations, as it is clear to think that many problems can be decomposed into low-dimensional subspaces which can be searched by separate species [141].

The cooperative co-evolutionary approaches have already been adopted to solve difficult RL tasks as evidenced in [245, 75, 235, 107, 108, 163]. These work particularly have shown that the applicability of cooperative co-evolution on training NN either at the level of a single neuron or the level of a group of neurons [75]. In line with our layer-wised PES approach in Chapter 5, it is natural to consider the potential use of cooperative co-evolutionary approach at a layer level. In this way, weights of different layers can be decomposed as different populations and can be co-evolved to further improve learning performance.

Bibliography

[1] ABE, N., VERMA, N., APTE, C., AND SCHROKO, R. Cross channel optimized marketing by reinforcement learning. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining* (2004), ACM, pp. 767–772.

[2] ABERDEEN, D. *Policy-gradient algorithms for partially observable Markov decision processes*. PhD thesis, The Australian National University, 2003.

[3] ALPAYDIN, E. *Introduction to machine learning*. MIT press, 2009.

[4] ALTMAN, D. G., AND BLAND, J. M. Parametric v non-parametric methods for data analysis. *Bmj 338* (2009), a3167.

[5] AMARI, S.-I. Natural gradient works efficiently in learning. *Neural Computation 10*, 2 (1998), 251–276.

[6] AMATRIAIN, X. Big & personal: data and models behind netflix recommendations. In *Proceedings of the 2nd international workshop on big data, streams and heterogeneous source Mining: Algorithms, systems, programming models and applications* (2013), ACM, pp. 1–6.

[7] AMERICAN, J. H. S., AND 1992. Genetic algorithms. *JSTOR*.

[8] ANDERSON, C. W. Approximating a policy can be easier than approximating a value function. *Computer Science Technical Report* (2000).

[9] ANDERSON, C. W., HITTLE, D. C., KATZ, A. D., AND KRETCHMAR, R. M. Synthesis of reinforcement learning, neural networks and PI con-

trol applied to a simulated heating coil. *Artificial Intelligence in Engineering 11*, 4 (1997), 421–429.

[10] ANSCHEL, O., BARAM, N., AND SHIMKIN, N. Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning. *arXiv preprint arXiv:1611.01929* (2016).

[11] ARAB, A., SCIENCES, A. A. I., AND 2015. An adaptive gradient descent-based local search in memetic algorithm applied to optimal controller design. *Elsevier 299* (Apr. 2015), 117–142.

[12] ARULKUMARAN, K., DEISENROTH, M. P., BRUNDAGE, M., AND BHARATH, A. A. A Brief Survey of Deep Reinforcement Learning. *IEEE Signal Process. Mag.*, 6 (Aug. 2017), 26–38.

[13] BÄCK, T., FOGEL, D. B., AND MICHALEWICZ, Z. Evolution strategies. *Evolutionary Computation* (2000), 119–126.

[14] BAGNELL, J. A., AND SCHNEIDER, J. Covariant policy search. In *IJCAI* (2003), Citeseer, pp. 1019–1024.

[15] BAGNELL, J. A., AND SCHNEIDER, J. G. Autonomous helicopter control using reinforcement learning policy search methods. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)* (2001), IEEE, pp. 1615–1620.

[16] BAIRD, L. Residual algorithms: Reinforcement learning with function approximation. In *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 30–37.

[17] BAIRD III, L. C., AND MOORE, A. W. Gradient descent for general reinforcement learning. In *Advances in neural information processing systems* (1999), pp. 968–974.

[18] BAK, P., TANG, C., AND WIESENFELD, K. Self-organized criticality: An explanation of the 1/f noise. *Physical review letters 59*, 4 (1987), 381.

[19] BANSAL, S., CALANDRA, R., CHUA, K., LEVINE, S., AND TOMLIN, C. MBMF: Model-Based Priors for Model-Free Reinforcement Learning.

[20] BAXTER, J., AND BARTLETT, P. L. Direct gradient-based reinforcement learning. In *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on* (2000), IEEE, pp. 271–274.

[21] BAXTER, J., AND BARTLETT, P. L. Infinite-horizon policy-gradient estimation. *1 15* (2001), 319–350.

[22] BELL, J. *Machine Learning*. Hands-On for Developers and Technical Professionals. John Wiley & Sons, Nov. 2014.

[23] BELLEMARE, M. G., NADDAF, Y., VENESS, J., AND BOWLING, M. The arcade learning environment: An evaluation platform for general agents. *1 47* (2013), 253–279.

[24] BELLMAN, R. *Dynamic programming*, vol. 342. Princeton University Press, 1957.

[25] BENBRAHIM, H., AND FRANKLIN, J. A. *Biped dynamic walking using reinforcement learning*. PhD thesis, Elsevier, 1997.

[26] BENGIO, Y., COURVILLE, A., AND VINCENT, P. Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence 35*, 8 (2013), 1798–1828.

[27] BENGIO, Y., LAMBLIN, P., POPOVICI, D., AND LAROCHELLE, H. Greedy Layer-Wise Training of Deep Networks. *NIPS* (2006).

[28] BERTSEKAS, D. P., AND TSITSIKLIS, J. N. *Neuro-dynamic Programming*. Athena Scientific, 1996.

[29] BHATNAGAR, S. An actor–critic algorithm with function approximation for discounted cost constrained Markov decision processes. *Systems & Control Letters 59*, 12 (2010), 760–766.

[30] BHATNAGAR, S., SUTTON, R. S., GHAVAMZADEH, M., AND LEE, M. Incremental Natural Actor-Critic Algorithms. *NIPS* (2007).

[31] BHATNAGAR, S., SUTTON, R. S., GHAVAMZADEH, M., AND LEE, M. Natural actor-critic algorithms. *Automatica 45*, 11 (2009), 2471–2482.

[32] BLATT, D., HERO, A. O., AND GAUCHMAN, H. A convergent incremental gradient method with a constant step size. *SIAM J. Optim. 18*, 1 (2007), 29–51.

[33] BORKAR, V. S. Stochastic approximation with two time scales. *Systems & Control Letters 29*, 5 (Feb. 1997), 291–294.

[34] BORKAR, V. S., AND KONDA, V. R. The actor-critic algorithm as multi-time-scale stochastic approximation. *Sadhana 22*, 4 (1997), 525–543.

[35] BOUTILIER, C., DEAN, T., AND HANKS, S. Decision-theoretic planning: Structural assumptions and computational leverage. *1 11* (1999), 1–94.

[36] BOYAN, J. A. Technical update: Least-squares temporal difference learning. *Mach Learn 49*, 2-3 (2002), 233–246.

[37] BRADTKE, S. J., AND BARTO, A. G. Linear least-squares algorithms for temporal difference learning. *Mach Learn 22*, 1-3 (1996), 33–57.

[38] BRAYLAN, A., HOLLENBECK, M., MEYERSON, E., AND MIIKKULAINEN, R. Frame skip is a powerful parameter for learning to play atari. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence* (2015).

[39] BROCKMAN, G., CHEUNG, V., PETTERSSON, L., SCHNEIDER, J., SCHULMAN, J., TANG, J., AND ZAREMBA, W. OpenAI Gym. *arXiv* (June 2016).

[40] BUSONIU, L., ERNST, D., DE SCHUTTER, B., AND BABUSKA, R. Policy search with cross-entropy optimization of basis functions. In *Adaptive Dynamic Programming and Reinforcement Learning, 2009. ADPRL'09. IEEE Symposium on* (2009), IEEE, pp. 153–160.

[41] CAMPBELL, M., HOANE JR., A. J., AND HSU, F.-H. Deep Blue. *Artificial Intelligence 134*, 1-2 (2002), 57–83.

[42] CARTWRIGHT, J. Roll over, boltzmann. *Physics World 27*, 05 (2014), 31.

[43] CHEN, G., DOUCH, C. I., AND ZHANG, M. Accuracy-based learning classifier systems for multistep reinforcement learning: a fuzzy logic approach to handling continuous inputs and learning continuous actions. *IEEE Transactions on Evolutionary Computation 20*, 6 (2016), 953–971.

[44] CHEN, R. Y., SIDOR, S., ABBEEL, P., AND SCHULMAN, J. UCB Exploration via Q-Ensembles. *arXiv preprint arXiv:1706.01502* (2017).

[45] COLAS, C., SIGAUD, O., AND OUDEYER, P.-Y. GEP-PG: Decoupling Exploration and Exploitation in Deep Reinforcement Learning Algorithms. *arXiv preprint arXiv:1802.05054* (2018).

[46] DA SILVA, F. L., AND COSTA, A. H. R. Transfer Learning for Multiagent Reinforcement Learning Systems. *IJCAI* (2016).

[47] DAYAN, P., AND HINTON, G. E. Using Expectation-Maximization for Reinforcement Learning. *Neural Computation* (1997).

[48] DE BOER, P.-T., KROESE, D. P., MANNOR, S., AND RUBINSTEIN, R. Y. A Tutorial on the Cross-Entropy Method. *Annals of Operations Research 134*, 1 (2005), 19–67.

[49] DEGRIS, T., WHITE, M., AND SUTTON, R. S. Off-Policy Actor-Critic. *arXiv* (2012).

[50] DEISENROTH, M. P. *Efficient Reinforcement Learning Using Gaussian Processes*. PhD thesis, KIT Scientific Publishing, 2010.

[51] DEISENROTH, M. P., FOX, D., AND RASMUSSEN, C. E. Gaussian Processes for Data-Efficient Learning in Robotics and Control. *IEEE Trans. Pattern Anal. Mach. Intell.* (2015).

[52] DEISENROTH, M. P., NEUMANN, G., AND PETERS, J. A survey on policy search for robotics. *Foundations and Trends® in Robotics 2*, 1–2 (2013), 1–142.

[53] DEISENROTH, M. P., AND RASMUSSEN, C. E. PILCO - A Model-Based and Data-Efficient Approach to Policy Search. *arXiv* (2011).

[54] DEVORE, R., AND KUNOTH, A. *Multiscale, Nonlinear and Adaptive Approximation*. Dedicated to Wolfgang Dahmen on the Occasion of his 60th Birthday. Springer Science & Business Media, Sept. 2009.

[55] DHAR, D. The abelian sandpile and related models. *Physica A: Statistical Mechanics and its Applications 263*, 1-4 (1999), 4–25.

[56] DHARIWAL, P., HESSE, C., KLIMOV, O., NICHOL, A., PLAPPERT, M., RADFORD, A., SCHULMAN, J., SIDOR, S., WU, Y., AND ZHOKHOV, P. OpenAI Baselines. Tech. rep., 2017.

[57] DI CASTRO, D., AND MANNOR, S. Adaptive bases for reinforcement learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases* (2010), Springer, pp. 312–327.

[58] DORIGO, M., AND BIRATTARI, M. Ant colony optimization. In *Encyclopedia of machine learning*. Springer, 2011, pp. 36–39.

[59] DUAN, Y., CHEN, X., HOUTHOOFT, R., SCHULMAN, J., AND ABBEEL, P. Benchmarking deep reinforcement learning for continuous control. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)* (2016).

[60] DUAN, Y., SCHULMAN, J., CHEN, X., ARXIV, P. B. A. P., AND 2016. RL: Fast Reinforcement Learning via Slow Reinforcement Learning. *arxiv.org*.

[61] DUCHI, J. C., AGARWAL, A., AND WAINWRIGHT, M. J. Dual averaging for distributed optimization: Convergence analysis and network scaling. *IEEE transactions on automatic control 57*, 3 (2012), 592–606.

[62] DUCHI, J. C., HAZAN, E., AND SINGER, Y. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research* (2011).

[63] EL-FAKDI, A., CARRERAS, M., AND PALOMERAS, N. Direct Policy Search Reinforcement Learning for Robot Control. *CCIA* (2005).

[64] ENGELBRECHT, A. P. Computational Intelligence : An Introduction. 1–630.

[65] EREZ, T., TASSA, Y., AND TODOROV, E. Infinite-Horizon Model Predictive Control for Periodic Tasks with Contacts. *Robotics - Science and Systems* (2011).

[66] GEIST, M., AND PIETQUIN, O. Algorithmic survey of parametric value function approximation. *IEEE Transactions on Neural Networks and Learning Systems 24*, 6 (2013), 845–867.

[67] GÉRON, A. Hands-On Machine Learning with Scikit-Learn and TensorFlow. 1–564.

[68] GHAVAMZADEH, M. Variance-Constrained Actor-Critic Algorithms for Discounted and Average Reward MDPs. *arXiv preprint arXiv:1403.6530* (2014).

[69] GHAVAMZADEH, M., AND ENGEL, Y. Bayesian actor-critic algorithms. In *Proceedings of the 24th international conference on Machine learning* (2007), ACM, pp. 297–304.

[70] GIRGIN, S., AND PREUX, P. Basis function construction in reinforcement learning using cascade-correlation learning architecture. In *Machine Learning and Applications, 2008. ICMLA'08. Seventh International Conference on* (2008), IEEE, pp. 75–82.

[71] GIRGIN, S., AND PREUX, P. Feature Discovery in Reinforcement Learning Using Genetic Programming. *EuroGP 4971*, Chapter 19 (2008), 218–229.

[72] GLAUBIUS, R., TIDWELL, T., GILL, C., AND SMART, W. D. Real-time scheduling via reinforcement learning. *arXiv preprint arXiv:1203.3481* (2012).

[73] GLOROT, X., BORDES, A., AND BENGIO, Y. Deep Sparse Rectifier Neural Networks. *AISTATS* (2011).

[74] GOLES, E., LATAPY, M., MAGNIEN, C., MORVAN, M., AND PHAN, H. D. Sandpile models and lattices: a comprehensive survey. *Theoretical Computer Science 322*, 2 (2004), 383–407.

[75] GOMEZ, F., SCHMIDHUBER, J., AND MIIKKULAINEN, R. Accelerated neural evolution through cooperatively coevolved synapses. *Journal of Machine Learning Research 9*, May (2008), 937–965.

[76] GOSAVI, A. Reinforcement learning: A tutorial survey and recent advances. *INFORMS Journal on Computing 21*, 2 (2009), 178–192.

[77] GOUMAGIAS, N. D., HRISTU-VARSAKELIS, D., AND ASSAEL, Y. M. Using deep Q-learning to understand the tax evasion behavior of risk-averse firms. *Expert Systems with Applications 101* (2018), 258–270.

[78] GREENSMITH, E., BARTLETT, P. L., AND BAXTER, J. Variance Reduction Techniques for Gradient Estimates in Reinforcement Learning. *Journal of Machine Learning Research* (2004).

[79] GRONDMAN, I., BUSONIU, L., LOPES, G. A. D., AND BABUSKA, R. A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients. *IEEE Trans. Syst., Man, Cybern. C 42*, 6 (2012), 1291–1307.

[80] GU, S., HOLLY, E., LILLICRAP, T., AND LEVINE, S. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on* (2017), IEEE, pp. 3389–3396.

[81] GU, S., LILLICRAP, T. P., SUTSKEVER, I., AND LEVINE, S. Continuous Deep Q-Learning with Model-based Acceleration. *arXiv* (2016).

[82] HA, S., AND LIU, C. K. Iterative training of dynamic skills inspired by human coaching techniques. *ACM Transactions on Graphics (TOG) 34*, 1 (2014), 1.

[83] HA, S., AND LIU, C. K. Evolutionary optimization for parameterized whole-body dynamic motor skills. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on* (2016), IEEE, pp. 1390–1397.

[84] HAFNER, R., AND RIEDMILLER, M. Reinforcement learning in feedback control. *Mach Learn 84*, 1-2 (Feb. 2011), 137–169.

[85] HANSEN, N. The CMA evolution strategy: a comparing review. In *Towards a new evolutionary computation*. Springer, 2006, pp. 75–102.

[86] HANSEN, N. The CMA Evolution Strategy: A Tutorial. *arXiv* (Apr. 2016).

[87] HANSEN, N., MÜLLER, S. D., AND KOUMOUTSAKOS, P. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation 11*, 1 (2003), 1–18.

[88] HANSEN, N., AND OSTERMEIER, A. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation 9*, 2 (2001), 159–195.

[89] HASSELT, H. V., AND WIERING, M. A. Reinforcement learning in continuous action spaces. In *Proceedings of the IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning, ADPRL* (2007), pp. 272–279.

[90] HAUSKNECHT, M., LEHMAN, J., MIIKKULAINEN, R., AND STONE, P. A neuroevolution approach to general atari game playing. *IEEE Trans. Comput. Intell. AI Games 6*, 4 (2014), 355–366.

[91] HAUSKNECHT, M. J., LEHMAN, J., MIIKKULAINEN, R., AND STONE, P. A Neuroevolution Approach to General Atari Game Playing. *IEEE Trans. Comput. Intellig. and AI in Games 6*, 4 (2014), 355–366.

[92] HAYKIN, S. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[93] HE, Q., AND SHAYMAN, M. A. Using reinforcement learning for proactive network fault management. In *Communication Technology Proceedings, 2000. WCC-ICCT 2000. International Conference on* (2000), IEEE, pp. 515–521.

[94] HEIDRICH-MEISNER, V., AND IGEL, C. Similarities and differences between policy gradient methods and evolution strategies. *ESANN* (2008).

[95] HENDERSON, P., CHANG, W.-D., SHKURTI, F., HANSEN, J., MEGER, D., AND DUDEK, G. Benchmark Environments for Multitask Learning in Continuous Domains. *arXiv cs.AI* (2017).

[96] HENDERSON, P., ISLAM, R., BACHMAN, P., PINEAU, J., PRECUP, D., AND MEGER, D. Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560* (2017).

[97] HERMUNDSTAD, A. M., BROWN, K. S., BASSETT, D. S., AND CARLSON, J. M. Learning, Memory, and the Role of Neural Network Architecture. *PLoS Computational Biology 7*, 6 (2011), e1002063.

[98] HESSEL, M., MODAYIL, J., VAN HASSELT, H., SCHAUL, T., OSTROVSKI, G., DABNEY, W., HORGAN, D., PIOT, B., AZAR, M. G., AND SILVER, D. Rainbow - Combining Improvements in Deep Reinforcement Learning. *arXiv* (2017).

[99] HESTER, T., VECERIK, M., PIETQUIN, O., LANCTOT, M., SCHAUL, T., PIOT, B., HORGAN, D., QUAN, J., SENDONARIS, A., AND DULAC-ARNOLD, G. Deep Q-learning from Demonstrations. *arXiv preprint arXiv:1704.03732* (2017).

[100] HINRICHS, T. R., AND FORBUS, K. D. Transfer Learning through Analogy in Games. *AI Magazine* (2011).

[101] HOUTHOOFT, R., CHEN, R. Y., ISOLA, P., STADIE, B. C., WOLSKI, F., HO, J., AND ABBEEL, P. Evolved Policy Gradients. *arXiv* (Feb. 2018).

[102] HSU, F.-H. *Behind Deep Blue: Building the Computer that Defeated the World Chess Champion*. Princeton University Press, 2004.

[103] IGEL, C. Neuroevolution for reinforcement learning using evolution strategies. In *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on* (2003), IEEE, pp. 2588–2595.

[104] ISBELL, C. L., KEARNS, M., SINGH, S., SHELTON, C. R., STONE, P., AND KORMANN, D. Cobot in LambdaMOO: An adaptive social statistics agent. *Autonomous Agents and Multi-Agent Systems 13*, 3 (2006), 327–354.

[105] ISLAM, R., HENDERSON, P., GOMROKCHI, M., AND PRECUP, D. Reproducibility of Benchmarked Deep Reinforcement Learning Tasks for Continuous Control.

[106] JADERBERG, M., MNIH, V., CZARNECKI, W. M., SCHAUL, T., LEIBO, J. Z., SILVER, D., AND KAVUKCUOGLU, K. Reinforcement Learning with Unsupervised Auxiliary Tasks. *arxiv* (Nov. 2016).

[107] JANSEN, T., AND WIEGAND, R. P. Exploring the explorative advantage of the cooperative coevolutionary (1+ 1) ea. In *Genetic and Evolutionary Computation Conference* (2003), Springer, pp. 310–321.

[108] JANSEN, T., AND WIEGAND, R. P. The cooperative coevolutionary (1+ 1) ea. *Evolutionary Computation 12*, 4 (2004), 405–434.

[109] JOHNS, J., AND MAHADEVAN, S. Constructing basis functions from directed graphs for value function approximation. *arXiv* (2007).

[110] JORDAN, M. I., AND MITCHELL, T. M. Machine learning: Trends, perspectives, and prospects. *Science 349*, 6245 (2015), 255–260.

[111] KAKADE, S., AND LANGFORD, J. Approximately optimal approximate reinforcement learning. In *ICML* (2002), pp. 267–274.

[112] KAKADE, S. M. A natural policy gradient. In *Advances in neural information processing systems* (2002), pp. 1531–1538.

[113] KARMAKAR, P., AND BHATNAGAR, S. Two time-scale stochastic approximation with controlled Markov noise and off-policy temporal-difference learning. *Mathematics of Operations Research 43*, 1 (2017), 130–151.

[114] KENNEDY, J. Swarm intelligence. In *Handbook of nature-inspired and innovative computing*. Springer, 2006, pp. 187–219.

[115] KENNEDY, J. Particle swarm optimization. In *Encyclopedia of machine learning*. Springer, 2011, pp. 760–766.

[116] KOBER, J., BAGNELL, J. A., AND PETERS, J. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research 32*, 11 (Aug. 2013), 1238–1274.

[117] KOBER, J., AND PETERS, J. R. Policy search for motor primitives in robotics. In *Advances in neural information processing systems* (2009), pp. 849–856.

[118] KOHL, N., NETWORKS, R. M. N., AND 2009. Evolving neural networks for strategic decision-making problems. *Elsevier 22*, 3 (Apr. 2009), 326–337.

[119] KOHL, N., AND STONE, P. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on* (2004), IEEE, pp. 2619–2624.

[120] KOLTER, J. Z., AND NG, A. Y. Regularization and Feature Selection in Least-Squares Temporal Difference Learning . In *ICML '09* (New York, New York, USA, 2009), ACM Press, pp. 521–528.

[121] KONDA, V. R., AND TSITSIKLIS, J. N. Actor-critic algorithms. In *Advances in neural information processing systems* (2000), pp. 1008–1014.

[122] KONDA, V. R., AND TSITSIKLIS, J. N. Convergence rate of linear two-time-scale stochastic approximation. *The Annals of Applied Probability 14*, 2 (2004), 796–819.

[123] KONIDARIS, G., OSENTOSKI, S., AND THOMAS, P. S. Value Function Approximation in Reinforcement Learning Using the Fourier Basis. *AAAI* (2011).

[124] KONIDARIS, G., SCHEIDWASSER, I., AND BARTO, A. Transfer in Reinforcement Learning via Shared Features. *Journal of Machine Learning Research 13*, May (2012), 1333–1371.

[125] KORMUSHEV, P., AND CALDWELL, D. G. Reinforcement learning with heterogeneous policy representations.

[126] KOTSIANTIS, S., AND KANELLOPOULOS, D. Discretization techniques: A recent survey. *GESTS International Transactions on Computer Science and Engineering 32*, 1 (2006), 47–58.

[127] KOUTNÍK, J., SCHMIDHUBER, J., AND GOMEZ, F. Evolving deep unsupervised convolutional networks for vision-based reinforcement learning. In *GECCO '14* (New York, New York, USA, 2014), ACM Press, pp. 541–548.

[128] KOZA, J. R. *Genetic programming II, automatic discovery of reusable subprograms*. MIT Press, Cambridge, MA, 1992.

[129] KUSHNER, H., AND YIN, G. *Stochastic Approximation and Recursive Algorithms and Applications*. Springer Science & Business Media, Nov. 2013.

[130] LE ROUX, N., SCHMIDT, M., AND BACH, F. A Stochastic Gradient Method with an Exponential Convergence Rate for Finite Training Sets. *arXiv* (Feb. 2012).

[131] LEVINE, S., AND KOLTUN, V. Guided Policy Search. *arXiv* (2013).

[132] LEVINE, S., POPOVIC, Z., AND KOLTUN, V. Feature Construction for Inverse Reinforcement Learning. *NIPS* (2010).

[133] LI, B., ONG, Y. S., LE, M. N., AND GOH, C. K. Memetic gradient search. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on* (2008), IEEE, pp. 2894–2901.

[134] LI, Z., AND ZHANG, Q. What does the evolution path learn in CMA-ES? In *International Conference on Parallel Problem Solving from Nature* (2016), Springer, pp. 751–760.

[135] LILLICRAP, T. P., HUNT, J. J., PRITZEL, A., HEESS, N., EREZ, T., TASSA, Y., SILVER, D., AND WIERSTRA, D. Continuous control with deep reinforcement learning. *arXiv* (Sept. 2015).

[136] LITTMAN, M. L. *Algorithms for sequential decision making*. PhD thesis, 1996.

[137] LIU, C., ZOPH, B., NEUMANN, M., SHLENS, J., HUA, W., LI, L.-J., FEI-FEI, L., YUILLE, A., HUANG, J., AND MURPHY, K. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2018), pp. 19–34.

[138] LIU, D.-R., LI, H.-L., AND WANG, D. Feature selection and feature learning for high-dimensional batch reinforcement learning: A survey. *Int. J. Autom. Comput. 12*, 3 (May 2015), 229–242.

[139] LOSCALZO, S., WRIGHT, R., AND YU, L. Predictive feature selection for genetic policy search. *Autonomous Agents and Multi-Agent Systems 29*, 5 (2015), 754–786.

[140] LOWELL, J., GRABKOVSKY, S., AND BIRGER, K. Comparison of NEAT and HyperNEAT performance on a strategic decision-making problem. In *2011 Fifth International Conference on Genetic and Evolutionary Computing* (2011), IEEE, pp. 102–105.

[141] MA, X., LI, X., ZHANG, Q., TANG, K., LIANG, Z., XIE, W., AND ZHU, Z. A survey on cooperative co-evolutionary algorithms. *IEEE Transactions on Evolutionary Computation 23*, 3 (2018), 421–441.

[142] MAEI, H. R., SZEPESVÁRI, C., BHATNAGAR, S., AND SUTTON, R. S. Toward off-policy learning control with function approximation. In *ICML* (2010), pp. 719–726.

[143] MAHADEVAN, S., 0006, B. L., THOMAS, P. S., DABNEY, W., GIGUERE, S., JACEK, N., GEMP, I., AND 0002, J. L. Proximal Reinforcement Learning - A New Theory of Sequential Decision Making in Primal-Dual Spaces. *arXiv* (2014).

[144] MAHMOOD, A. R., VAN HASSELT, H., AND SUTTON, R. S. Weighted importance sampling for off-policy learning with linear function approximation. *NIPS* (2014).

[145] MAKMAL, A., MELNIKOV, A. A., DUNJKO, V., AND BRIEGEL, H.-J. Meta-learning within Projective Simulation. *IEEE Access 4* (2016), 2110–2122.

[146] MANIA, H., GUY, A., AND RECHT, B. Simple random search provides a competitive approach to reinforcement learning. *arXiv cs.LG* (2018).

[147] MARSLAND, S. *Machine learning: an algorithmic perspective*. Chapman and Hall/CRC, 2011.

[148] MAUSAM, A. K. Planning with Markov decision processes: an AI perspective. *Morgan & Claypool Publishers* (2012).

[149] MELO, F. S., MEYN, S. P., AND RIBEIRO, M. I. An analysis of reinforcement learning with function approximation. *arXiv* (2008), 664–671.

[150] MENACHE, I., MANNOR, S., AND SHIMKIN, N. Basis Function Adaptation in Temporal Difference Reinforcement Learning. *Annals OR 134*, 1 (2005), 215–238.

[151] MICHALEWICZ, Z. Evolution strategies and other methods. In *Genetic Algorithms+ Data Structures= Evolution Programs*. Springer, 1996, pp. 159–177.

[152] MIIKKULAINEN, R., LIANG, J., MEYERSON, E., RAWAL, A., FINK, D., FRANCON, O., RAJU, B., SHAHRZAD, H., NAVRUZYAN, A., AND DUFFY, N. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Elsevier, 2019, pp. 293–312.

[153] MITCHELL, T. M. *Machine learning*. McGraw-Hill Science/Engineering/Math, 1997.

[154] MNIH, V., BADIA, A. P., MIRZA, M., GRAVES, A., LILLICRAP, T. P., HARLEY, T., SILVER, D., AND KAVUKCUOGLU, K. Asynchronous Methods for Deep Reinforcement Learning. *arXiv* (2016).

[155] MNIH, V., KAVUKCUOGLU, K., SILVER, D., RUSU, A. A., VENESS, J., BELLEMARE, M. G., GRAVES, A., RIEDMILLER, M., FIDJELAND, A. K., OSTROVSKI, G., PETERSEN, S., BEATTIE, C., SADIK, A., ANTONOGLOU, I., KING, H., KUMARAN, D., WIERSTRA, D., LEGG, S., AND HASSABIS, D. Human-level control through deep reinforcement learning. *Nature 518*, 7540 (Feb. 2015), 529–533.

[156] MOODY, J., AND SAFFELL, M. Learning to trade via direct reinforcement. *IEEE transactions on neural Networks 12*, 4 (2001), 875–889.

[157] MORIARTY, D. E., SCHULTZ, A. C., AND GREFENSTETTE, J. J. Evolutionary Algorithms for Reinforcement Learning. *J. Artif. Intell. Res.* (1999).

[158] NESTEROV, Y. Primal-dual subgradient methods for convex problems. *Math. Program. 120*, 1 (June 2007), 221–259.

[159] NESTEROV, Y. Efficiency of Coordinate Descent Methods on Huge-Scale Optimization Problems. *SIAM J. Optim. 22*, 2 (2012), 341–362.

[160] NG, A. Y. *Shaping and policy search in reinforcement learning*. PhD thesis, University of California, Berkeley, 2003.

[161] OSBAND, I., VAN ROY, B., AND WEN, Z. Generalization and Exploration via Randomized Value Functions. *arXiv* (2016).

[162] PAN, S. J., AND YANG, Q. A survey on transfer learning. *IEEE Transactions on knowledge and data ...* (2010).

[163] PANAIT, L., LUKE, S., AND HARRISON, J. F. Archive-based cooperative coevolutionary algorithms. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation* (2006), ACM, pp. 345–352.

[164] PARR, R., LI, L., TAYLOR, G., PAINTER-WAKEFIELD, C., AND LITTMAN, M. L. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. *arXiv* (2008).

[165] PARR, R., PAINTER-WAKEFIELD, C., LI, L., AND LITTMAN, M. L. Analyzing feature generation for value-function approximation. *arXiv* (2007), 737–744.

[166] PENG, Y., CHEN, G., ZHANG, M., AND PANG, S. Generalized compatible function approximation for policy gradient search. In *International Conference on Neural Information Processing* (2016), Springer, pp. 615–622.

[167] PENG, Y., CHEN, G., ZHANG, M., AND PANG, S. A sandpile model for reliable actor-critic reinforcement learning. In *Neural Networks (IJCNN), 2017 International Joint Conference on* (2017), IEEE, pp. 4014–4021.

[168] PETERS, J., AND SCHAAL, S. Policy gradient methods for robotics. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on* (2006), IEEE, pp. 2219–2225.

[169] PETERS, J., AND SCHAAL, S. *Reinforcement learning by reward-weighted regression for operational space control*. ACM, New York, New York, USA, June 2007.

[170] PETERS, J., AND SCHAAL, S. Natural Actor-Critic. *Neurocomputing 71*, 7 (Mar. 2008), 1180–1190.

[171] PETERS, J., AND SCHAAL, S. Reinforcement learning of motor skills with policy gradients. *Neural Networks 21*, 4 (May 2008), 682–697.

[172] PETERS, J., VIJAYAKUMAR, S., AND SCHAAL, S. Reinforcement learning for humanoid robotics. In *Proceedings of the third IEEE-RAS international conference on humanoid robots* (2003), pp. 1–20.

[173] PETERS, J. R. *Machine learning of motor skills for robotics*. PhD thesis, University of Southern California, 2007.

[174] PHAM, H., GUAN, M. Y., ZOPH, B., LE, Q. V., AND DEAN, J. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268* (2018).

[175] PONG, V., GU, S., DALAL, M., AND LEVINE, S. Temporal Difference Models: Model-Free Deep RL for Model-Based Control.

[176] PRASHANTH, L. A., AND GHAVAMZADEH, M. Variance-constrained actor-critic algorithms for discounted and average reward MDPs. *Mach Learn 105*, 3 (Aug. 2016), 367–417.

[177] PROPER, S., AND TADEPALLI, P. Scaling model-based average-reward reinforcement learning for product delivery. In *European Conference on Machine Learning* (2006), Springer, pp. 735–742.

[178] RAJESWARAN, A., LOWREY, K., TODOROV, E. V., AND KAKADE, S. M. Towards Generalization and Simplicity in Continuous Control. 6550–6561.

[179] RECHENBERG, I. Evolutionsstrategie–Optimierung technisher Systeme nach Prinzipien der biologischen Evolution.

[180] ROBERT SEDGEWICK, K. W. Algorithms, Fourth Edition. 1–969.

[181] ROMAIN LAROCHE, M. B. Transfer Reinforcement Learning with Shared Dynamics. 1–7.

[182] ROS, R., AND HANSEN, N. A simple modification in CMA-ES achieving linear time and space complexity. In *International Conference on Parallel Problem Solving from Nature* (2008), Springer, pp. 296–305.

[183] ROSENSTEIN, M. T., AND BARTO, A. G. Robot weightlifting by direct policy search. In *International Joint Conference on Artificial Intelligence* (2001), Citeseer, pp. 839–846.

[184] RUMMERY, G. A., AND NIRANJAN, M. On-line Q-learning using connectionist systems. Tech. rep., 1994.

[185] RUSSELL, S. J., AND NORVIG, P. *Artificial intelligence: a modern approach*, 3rd ed. Pearson Education Limited, 2016.

[186] SALIMANS, T., HO, J., CHEN, X., SIDOR, S., AND SUTSKEVER, I. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864* (2017).

[187] SALLAB, A. E., ABDOU, M., PEROT, E., AND YOGAMANI, S. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging 2017*, 19 (2017), 70–76.

[188] SANTORO, A., BARTUNOV, S., BOTVINICK, M., WIERSTRA, D., AND LILLICRAP, T. P. Meta-Learning with Memory-Augmented Neural Networks. *arXiv* (2016).

[189] SCHULMAN, J., LEVINE, S., ABBEEL, P., JORDAN, M. I., AND MORITZ, P. Trust Region Policy Optimization. *arXiv* (2015).

[190] SCHULMAN, J., MORITZ, P., LEVINE, S., JORDAN, M. I., AND ABBEEL, P. High-Dimensional Continuous Control Using Generalized Advantage Estimation. *arXiv cs.LG* (2015).

[191] SCHULMAN, J., WOLSKI, F., DHARIWAL, P., ARXIV, A. R. A. P., AND 2017. Proximal policy optimization algorithms. *arxiv.org*.

[192] SEHNKE, F., OSENDORFER, C., RÜCKSTIESS, T., GRAVES, A., PETERS, J., AND SCHMIDHUBER, J. Policy gradients with parameter-based exploration for control. In *International Conference on Artificial Neural Networks* (2008), Springer, pp. 387–396.

[193] SENCIANES, A. E.-F. *GRADIENT-BASED REINFORCEMENT LEARNING TECHNIQUES FOR UNDERWATER ROBOTICS BEHAVIOR LEARNING*. PhD thesis, Dec. 2010.

[194] SHIE MANNOR, R. R., AND GAT, Y. The Cross Entropy Method for Fast Policy Search. 1–8.

[195] SI, J., BARTO, A. G., POWELL, W. B., AND WUNSCH, D. *Handbook of learning and approximate dynamic programming*, vol. 2. John Wiley & Sons, 2004.

[196] SIDHU, G., AND CAFFO, B. MONEYBaRL: exploiting pitcher decision-making using reinforcement learning. *The Annals of Applied Statistics* (2014), 926–955.

[197] SILVER, D., HUANG, A., MADDISON, C. J., GUEZ, A., SIFRE, L., VAN DEN DRIESSCHE, G., SCHRITTWIESER, J., ANTONOGLOU, I., PANNEERSHELVAM, V., LANCTOT, M., DIELEMAN, S., GREWE, D., NHAM, J., KALCHBRENNER, N., SUTSKEVER, I., LILLICRAP, T., LEACH, M., KAVUKCUOGLU, K., GRAEPEL, T., AND HASSABIS, D. Mastering the game of Go with deep neural networks and tree search. *Nature 529*, 7585 (Jan. 2016), 484–489.

[198] SILVER, D., LEVER, G., HEESS, N., DEGRIS, T., WIERSTRA, D., AND RIEDMILLER, M. A. Deterministic Policy Gradient Algorithms. *arXiv* (2014).

[199] SILVER, D., SCHRITTWIESER, J., SIMONYAN, K., ANTONOGLOU, I., HUANG, A., GUEZ, A., HUBERT, T., BAKER, L., LAI, M., BOLTON, A., CHEN, Y., LILLICRAP, T., HUI, F., SIFRE, L., VAN DEN DRIESSCHE, G., GRAEPEL, T., AND HASSABIS, D. Mastering the game of Go without human knowledge. *Nature 550*, 7676 (Oct. 2017), 354–359.

[200] SIMSEK, O., ALGORTA, S., AND KOTHIYAL, A. Why Most Decisions Are Easy in Tetris—And Perhaps in Other Sequential Decision Problems, As Well. 1757–1765.

[201] SINGH, S., LITMAN, D., KEARNS, M., AND WALKER, M. Optimizing dialogue management with reinforcement learning: Experiments with the NJFun system. *1 16* (2002), 105–133.

[202] SPALL, J. C. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE transactions on automatic control 37*, 3 (1992), 332–341.

[203] STAMATE, C., MAGOULAS, G. D., AND THOMAS, M. S. C. Transfer learning approach for financial applications. *arXiv* (Sept. 2015).

[204] STANLEY, K. O. Neuroevolution: A different kind of deep learning.

[205] STANLEY, K. O., D'AMBROSIO, D. B., AND GAUCI, J. A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life 15*, 2 (2009), 185–212.

[206] STANLEY, K. O., AND MIIKKULAINEN, R. Efficient Reinforcement Learning Through Evolving Neural Network Topologies. *Gecco*, Chapter 6 (2002).

[207] STANLEY, K. O., AND MIIKKULAINEN, R. Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation 10*, 2 (2002), 99–127.

[208] STANLEY, K. O., AND MIIKKULAINEN, R. Evolving a roving eye for go. In *Genetic and Evolutionary Computation Conference* (2004), Springer, pp. 1226–1238.

[209] STONE, P., AND SUTTON, R. S. Scaling reinforcement learning toward RoboCup soccer. In *ICML* (2001), Citeseer, pp. 537–544.

[210] SUCH, F. P., MADHAVAN, V., CONTI, E., LEHMAN, J., STANLEY, K. O., AND CLUNE, J. Deep Neuroevolution - Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning. *arXiv cs.NE* (2017).

[211] SUTTON, R. S. Learning to predict by the methods of temporal differences. *Mach Learn 3*, 1 (1988), 9–44.

[212] SUTTON, R. S., AND BARTO, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

[213] SUTTON, R. S., MAEI, H. R., PRECUP, D., BHATNAGAR, S., SILVER, D., SZEPESVÁRI, C., AND WIEWIORA, E. Fast gradient-descent methods for

temporal-difference learning with linear function approximation. *arXiv* (2009).

[214] SUTTON, R. S., MAEI, H. R., AND SZEPESVÁRI, C. A Convergent $O(n)$ Temporal-difference Algorithm for Off-policy Learning with Linear Function Approximation. 1609–1616.

[215] SUTTON, R. S., MCALLESTER, D. A., SINGH, S. P., AND MANSOUR, Y. Policy Gradient Methods for Reinforcement Learning with Function Approximation. *NIPS* (1999).

[216] SUTTON, R. S., MODAYIL, J., DELP, M., DEGRIS, T., PILARSKI, P. M., WHITE, A., AND PRECUP, D. *Horde: a scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction*. International Foundation for Autonomous Agents and Multiagent Systems, May 2011.

[217] SUTTON, R. S., PRECUP, D., AND SINGH, S. P. Between MDPs and Semi-MDPs - A Framework for Temporal Abstraction in Reinforcement Learning. *Artif. Intell. 112*, 1-2 (1999), 181–211.

[218] SZEPESVÁRI, C. *Algorithms for reinforcement learning*, vol. 9. Synthesis Lectures on Artificial Intelligence and Machine Learning, Aug. 2010.

[219] TAN, J., GU, Y., TURK, G., AND LIU, C. K. Articulated swimming creatures. *ACM Transactions on Graphics (TOG) 30*, 4 (2011), 58.

[220] TAN, J., ZHANG, T., COUMANS, E., ISCEN, A., BAI, Y., HAFNER, D., BOHEZ, S., AND VANHOUCKE, V. Sim-to-Real: Learning Agile Locomotion For Quadruped Robots. *arXiv preprint arXiv:1804.10332* (2018).

[221] TASSA, Y., EREZ, T., AND TODOROV, E. Synthesis and stabilization of complex behaviors through online trajectory optimization. *IROS* (2012), 4906–4913.

[222] TOKIC, M. Adaptive ε-Greedy Exploration in Reinforcement Learning Based on Value Differences. In *KI 2010: Advances in Artificial Intelligence*. Springer, Berlin, Heidelberg, Berlin, Heidelberg, Sept. 2010, pp. 203–210.

[223] UMAROV, S., TSALLIS, C., AND STEINBERG, S. On a q-central limit theorem consistent with nonextensive statistical mechanics. *Milan journal of mathematics 76*, 1 (2008), 307–328.

[224] VAN DEN BERG, T. G., AND WHITESON, S. Critical factors in the performance of HyperNEAT. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation* (2013), ACM, pp. 759–766.

[225] VAN HASSELT, H. Double Q-learning. *NIPS* (2010).

[226] VIEN, N. A., DANG, V.-H., AND CHUNG, T. A covariance matrix adaptation evolution strategy for direct policy search in reproducing kernel hilbert space. In *Asian Conference on Machine Learning* (2017), pp. 606–621.

[227] WAMPLER, K., AND POPOVIC, Z. Optimal gait and form for animal locomotion. *ACM Trans. Graph. 28*, 3 (2009), 1.

[228] WANG, J. M., HAMNER, S. R., DELP, S. L., AND KOLTUN, V. Optimizing Locomotion Controllers Using Biologically-Based Actuators and Objectives Supplemental Material.

[229] WANG, Z., BAPST, V., HEESS, N., MNIH, V., MUNOS, R., KAVUKCUOGLU, K., AND DE FREITAS, N. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224* (2016).

[230] WANG, Z., SCHAUL, T., HESSEL, M., VAN HASSELT, H., LANCTOT, M., AND DE FREITAS, N. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581* (2015).

[231] WATKINS, C. J., AND DAYAN, P. Q-learning. *Mach Learn 8*, 3-4 (1992), 279–292.

[232] WHITESON, S., AND STONE, P. Evolutionary Function Approximation for Reinforcement Learning. *Journal of Machine Learning Research* (2006).

[233] WHITESON, S., STONE, P., STANLEY, K. O., MIIKKULAINEN, R., AND KOHL, N. Automatic feature selection in neuroevolution. *Gecco* (2005), 1225.

[234] WHITESON, S. A. *Adaptive Representations for Reinforcement Learning*. PhD thesis, May 2007.

[235] WIEGAND, R. P., LILES, W. C., AND JONG, K. A. D. An empirical analysis of collaboration methods in cooperative coevolutionary algorithms. In *Proceedings of the 3rd annual conference on genetic and evolutionary computation* (2001), Morgan Kaufmann Publishers Inc., pp. 1235–1242.

[236] WIERING, M., AND VAN OTTERLO, M. *Reinforcement learning*. Springer, 2012.

[237] WIERSTRA, D., SCHAUL, T., GLASMACHERS, T., SUN, Y., PETERS, J., AND SCHMIDHUBER, J. Natural evolution strategies. *Journal of Machine Learning Research* (2014).

[238] WILLIAMS, R. J. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Mach Learn 8*, 3-4 (1992), 229–256.

[239] WOOKEY, D. S., AND KONIDARIS, G. D. Regularized feature selection in reinforcement learning. *Mach Learn 100*, 2-3 (July 2015), 655–676.

[240] WU, Y., MANSIMOV, E., GROSSE, R. B., LIAO, S., AND BA, J. Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. 5279–5288.

[241] XIAO, L. Dual averaging methods for regularized stochastic learning and online optimization. *Journal of Machine Learning Research 11*, Oct (2010), 2543–2596.

[242] YAN, X., DIACONIS, P., RUSMEVICHIENTONG, P., AND ROY, B. V. Solitaire: Man versus machine. In *Advances in neural information processing systems* (2005), pp. 1553–1560.

[243] YAO, X., AND LIU, Y. A new evolutionary system for evolving artificial neural networks. *IEEE Trans. Neural Networks* (1997).

[244] YAO, X., AND XU, Y. Recent advances in evolutionary computation. *Journal of Computer Science and Technology 21*, 1 (2006), 1–18.

[245] YONG, C. H., AND MIIKKULAINEN, R. Cooperative coevolution of multi-agent systems. *University of Texas at Austin, Austin, TX* (2001).

[246] YU, H., AND BERTSEKAS, D. P. Basis function adaptation methods for cost approximation in MDP. In *Adaptive Dynamic Programming and Reinforcement Learning, 2009. ADPRL'09. IEEE Symposium on* (2009), IEEE, pp. 74–81.

[247] YU, J., ABERDEEN, D., AND SCHRAUDOLPH, N. N. Fast online policy gradient learning with smd gain vector adaptation. In *Advances in neural information processing systems* (2006), pp. 1185–1192.

[248] ZABINSKY, Z. B. Random search algorithms. *Wiley Encyclopedia of Operations Research and Management Science* (2010).

[249] ZHOU, Z., ONG, Y. S., NAIR, P. B., AND ON, A. K. Combining global and local surrogate models to accelerate evolutionary optimization. *ieeexplore.ieee.org* (2007).