On Representations and Exploration in Policy Search Methods for Multi-Agent Robotics

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik des
Karlsruher Instituts für Technologie (KIT)

genehmigte
Dissertation

von
Maximilian Hüttenrauch
geb. in Wiesbaden

Tag der mündlichen Prüfung: 21. Oktober 2024
1. Referent: Prof. Dr. Gerhard Neumann
2. Referent: Prof. Dr. Heinz Köppl

Abstract

Artificial intelligent agents such as robots have made remarkable progress in recent years, slowly escaping from confined industrial manufacturing settings. A key enabler to this transition is reinforcement learning, a machine learning sub-domain that allows agents to learn from trial and error and excel in domains such as manipulation of objects, sorting, or navigation. While many of the recent improvements stem from more and more powerful computation, there are still many aspects of the learning process that can be improved, two of which we investigate in this thesis. In the domain of multi-agent learning where multiple autonomous agents interact and learn from their experiences, data representation presents several significant challenges, while in policy search for robotic manipulation, a central question is how to make more efficient use of data as interaction with the environment can be costly.

In Chapter 3, we investigate the problem of multiple agents interacting to achieve a collaborative goal. A key challenge is to find a good representation of the agents' state space, which needs to be compact and independent of the number of observed agents. We present an agent-centric representation of the local neighborhood, based on histogram embeddings of geometric features such as relative distances and velocities of neighboring agents. We combine these features with an adaptation of Trust-Region Policy Search, a deep reinforcement learning algorithm for single-agent learning, to solve collaborative navigation and coordination tasks. With the ability to embed communicated features of adjacent agents, the tasks can be solved even in the case of limited observability.

One of the shortcomings of the method presented in Chapter 3 is the exponentially growing representation space with each added feature. Thus, Chapter 4 extends the agent-centric representations using learned embeddings of features. These embeddings are based on the idea of mean embeddings where the underlying data distribution is embedded into a higher dimensional latent space. This choice allows for a greater number of features to be taken into account. The method is evaluated on new collaborative tasks with a higher number of agents compated to Chapter 3.

In Chapter 5, we attend to the problem of safe and efficient exploration. We investigate exploration in low dimensional robotic manipulation problems where the robot's trajectories are encoded using movement primitives. The optimization of movement primitive parameters requires careful updates, as the individual parameters are highly correlated. Starting with an existing version of the Model-based Relative Entropy Stochastic Search algorithm, we present a new algorithm with several improvements. We stabilize the optimization process by running separate updates for the mean and covariance of the policy, improve the surrogate model estimation, and introduce an adaptive entropy scaling method. We show the effectiveness of the new algorithm on complex robotic manipulation tasks. Additionally, we sketch how this approach can be extended to multi-agent settings where multiple robots need to solve a task in collaboration.

In summary, this thesis focuses on two core problems in reinforcement learning: (1) state representation in the presence of multiple agents, and (2) efficient exploration of the parameter space in episode-based policy learning using movement primitives.

Zusammenfassung

Künstliche intelligente Agenten wie Roboter haben in den letzten Jahren bemerkenswerte Fortschritte gemacht und sich dabei langsam aus den begrenzten industriellen Produktionsumgebungen befreit. Eine treibende Kraft für diesen Fortschritt ist das Verstärkungslernen, ein Teilbereich des maschinellen Lernens, der es Agenten ermöglicht aus Versuch und Irrtum zu lernen. Dies ermöglicht hervorragende Lösungen in Bereichen wie der Manipulation von Objekten, dem Sortieren oder der Navigation. Obwohl viele der jüngsten Verbesserungen auf immer leistungsfähigere Berechnungen zurückzuführen sind, gibt es noch viele Aspekte des Lernprozesses, die verbessert werden können. Zwei davon werden wir in dieser Arbeit untersuchen. Im Bereich des Multi-Agenten-Lernens, bei dem mehrere Agenten interagieren und aus ihren Erfahrungen lernen, stellt die Datenrepräsentation eine Reihe bedeutender Herausforderungen. Desweiteren ist eine zentrale Frage, wie Daten bei der Suche nach Strategien für die robotische Manipulation effizienter genutzt werden können, da die Interaktion mit der Umwelt kostspielig sein kann.

In Kapitel 3 untersuchen wir das Problem der Interaktion mehrerer Agenten zur Erreichung eines kollaboratives Ziels. Eine zentrale Herausforderung besteht darin, eine gute Darstellung des Zustandsraums der Agenten zu finden, welche sowohl kompakt als auch unabhängig von der Anzahl der beobachteten Agenten sein muss. Wir präsentieren eine agenten-zentrierte Darstellung der lokalen Nachbarschaft, die auf Histogrammeinbettungen von geometrischen Merkmalen wie relativen Distanzen und Geschwindigkeiten benachbarten Agenten basieren. Wir kombinieren diese Merkmale mit einer angepassten Version von Trust-Region Policy Search, einem Algorithmus für das Lernen von einzelnen Agenten, um kollaborative Navigations- und Koordinationsaufgaben zu lösen. Durch die Möglichkeit, kommunizierte Merkmale benachbarter Agenten einzubetten, können die Aufgaben auch bei begrenzter Beobachtbarkeit gelöst werden.

Einer der Schwachpunkte der in Kapitel 3 vorgestellten Methode ist der exponentiell wachsende Repräsentationsraum mit jedem hinzugefügten Merkmal. Daher werden in Kapitel 4 die agenten-zentrierte Repräsentationen durch gelernte Einbettungen von Merkmalen erweitert. Diese Einbettungen basieren auf der Idee der Mittelwert-Einbettung, bei der die zugrunde liegende Datenverteilung in einen einen höherdimensionalen latenten Raum eingebettet wird. Diese Wahl ermöglicht die Berücksichtigung einer größeren Anzahl von Merkmalen. Die Methode wird an neuen kollaborativen Aufgaben mit einer höheren Anzahl von Agenten im Vergleich zu Kapitel 3 getestet.

In Kapitel 5 befassen wir uns mit dem Problem der sicheren und effizienten Exploration. Wir untersuchen die Exploration bei niedrigdimensionalen Robotermanipulationsproblemen, bei denen die Trajektorien des Roboters mit Hilfe von Bewegungsprimitiven kodiert werden. Die Optimierung der Parameter der Bewegungsprimitive erfordert sorgfältige Aktualisierungen, da die einzelnen Parameter stark korreliert sind. Ausgehend von einer bestehenden Version des Model-Based Relative Entropy Stochastic Search Algorithmus präsentieren wir einen neuen Algorithmus mit mehreren Verbesserungen. Wir stabilisieren den Optimierungsprozess, indem wir separate Updates für den Mittelwert und die Kovarianz der Policy einführen, verbessern die Schätzung des Surrogatmodells und führen eins

adaptives Entropieskalierungsverfahren ein. Wir zeigen die Effektivität des neuen Algorithmus an komplexen Robotermanipulations Aufgaben. Außerdem skizzieren wir, wie dieser Ansatz auf Multi-Agenten Probleme erweitert werden kann wo mehrere Roboter eine Aufgabe gemeinsam lösen müssen.

Zusammenfassend konzentriert sich diese Dissertation auf zwei Kernprobleme des Reinforcement Learning: (1) Zustandsrepräsentation in Anwesenheit mehrerer Agenten und (2) effiziente Exploration des Parameterraums beim episodenbasierten Policy-Lernen unter Verwendung von Bewegungsprimitiven.

Acknowledgments

First of all, I would like to thank Geri without whose support I would never have finished this thesis. Your seemingly endless, at times almost annoying, optimism is what kept me going. Thank you for believing in me, especially in times when I didn't. Also, I would like to thank Adrian for the collaboration, giving me great advice, and always being available during the first two years of my PhD.

Next, I would like to thank all the wonderful colleagues at ALR that create an inspiring environment, especially Onur and Philipp who have accompanied me since my restart in Germany. It was good to not move alone to a new city for a third time. Also, thank you Philipp for explaining me more than once what I was actually doing. Thanks to my students Philipp, Philipp, Robin, Felix, Yuhe, Denis, and Lyuba for letting me be your thesis supervisors. It was a pleasure guiding you.

To Sergi, Paolo, and the Lincoln Handball club who made my stay in Lincoln more pleasant. To Philipp, Philipp, Svenja, and Balázs for the Doppelkopf rounds. To Nic for the countless bike discussions. To Michael for being almost as old as I am.

I would like to thank my brother, and my parents who give me all the freedom in the world to pursue whatever goal I had in mind. Thank you for your support and always being there for me.

Finally, I'm grateful to the awesome people at Kohi Karlsruhe, especially Hannah, Cora, and Cat. You've made me feel at home for the first time in a long time.

Contents

Abstract . . . . . **i**

Zusammenfassung . . . . . **iii**

Acknowledgments . . . . . **v**

List of Figures . . . . . **xi**

List of Tables . . . . . **xvii**

List of Algorithms . . . . . **xix**

1. Introduction **1**
1.1. Contributions 1
1.1.1. Local Communication Protocols for Learning Complex Swarm Behaviors with Deep Reinforcement Learning 2
1.1.2. Deep Reinforcement Learning for Swarm Systems 2
1.1.3. Robust Black-Box Optimization for Stochastic Search and Episodic Reinforcement Learning 2
1.1.4. Black-Box Optimization for Episode-Based Multi-Agent Reinforcement Learning 3
1.2. Structure of Thesis 3

2. Fundamentals and State of the Art **5**
2.1. Sequential Decision Making 5
2.1.1. Markov Decision Processes 5
2.1.2. Value Functions 5
2.1.3. Reinforcement Learning 6
2.1.4. Multi-Agent Reinforcement Learning 7
2.2. Episode-Based Reinforcement Learning 8
2.2.1. Probabilistic Movement Primitives as Episode-Based Policy Parameterizations . 9
2.2.2. Advantages and Disadvantages of Episode-Based Reinforcement Learning . . . 9
2.3. Information Representation in Swarms 10
2.3.1. Set Representation of Local Observations 10
2.3.2. Swarm as a Graph 13
2.3.3. Graph Multi-Agent RL 14
2.4. Exploration Strategies in Reinforcement Learning 14
2.4.1. Trust-Region Reinforcement Learning 15
2.4.2. Evolution Strategies 18

3. Local Communication Protocols for Learning Complex Swarm Behaviors with Deep Reinforcement Learning . . . **21**
3.1. Background . . . 22
3.1.1. Trust Region Policy Optimization . . . 22
3.1.2. Problem Domain . . . 23
3.1.3. Related Work . . . 23
3.2. Multi-Agent Learning with Local Communication Protocols . . . 24
3.2.1. Communication Protocols . . . 24
3.2.2. Weight Sharing for Policy Networks . . . 25
3.2.3. Adaptations to TRPO . . . 25
3.3. Experimental Setup . . . 26
3.3.1. Agent Model . . . 26
3.3.2. Tasks . . . 26
3.3.3. Policy Architecture . . . 27
3.4. Results . . . 28
3.4.1. Edge Task . . . 28
3.4.2. Link Task . . . 29
3.5. Conclusions and Future Work . . . 29

4. Deep Reinforcement Learning for Swarm Systems . . . **31**
4.1. Related Work . . . 33
4.1.1. Deep RL . . . 33
4.1.2. Optimization-Based Approaches for Swarm Systems . . . 34
4.1.3. Analytic Approaches . . . 34
4.2. Background . . . 35
4.2.1. Trust Region Policy Optimization . . . 35
4.2.2. Mean Embeddings . . . 35
4.3. Deep Reinforcement Learning for Swarms . . . 36
4.3.1. Problem Domain . . . 36
4.3.2. Local Observation Models . . . 36
4.3.3. Local Communication Models . . . 37
4.3.4. Mean Embeddings as State Representations for Swarms . . . 37
4.3.5. Other Representation Techniques . . . 39
4.3.6. Adaption of TRPO to the Homogeneous Swarm Setup . . . 39
4.4. Experimental Results . . . 40
4.4.1. Swarm Models . . . 40
4.4.2. Rendezvous . . . 41
4.4.3. Pursuit Evasion with a Single Evader . . . 45
4.4.4. Pursuit Evasion with Multiple Evaders . . . 48
4.4.5. Evaluation of Pooling Functions . . . 48
4.4.6. Comparison to Moment-Based Representations . . . 50
4.4.7. Computational Complexity . . . 50
4.5. Conclusion . . . 50

5. Robust Black-Box Optimization for Stochastic Search and Episodic Reinforcement Learning . . . **53**
5.1. Related Work . . . 54
5.1.1. Evolutionary Strategies and Black-Box Optimization . . . 54
5.1.2. Reinforcement Learning . . . 55
5.1.3. Broader Scope . . . 56

5.2. Preliminaries . . . 57
5.2.1. Problem Setting . . . 57
5.2.2. Model-Based Relative Entropy Stochastic Search . . . 57
5.2.3. Relation to Natural Gradient . . . 59
5.3. Improving the MORE Algorithm . . . 61
5.3.1. Disentangled Trust Regions . . . 61
5.3.2. Entropy Control . . . 62
5.3.3. Illustrative Example . . . 64
5.4. Model Learning . . . 65
5.4.1. Least Squares Model Fitting . . . 65
5.4.2. Adaptive Model Complexity . . . 66
5.4.3. Data Pre-Processing . . . 66
5.5. Experiments . . . 68
5.5.1. Black-Box Optimization Benchmarks . . . 69
5.5.2. Episodic Reinforcement Learning Results . . . 71
5.6. Conclusion . . . 77
5.7. Acknowledgements . . . 78

6. Black-Box Optimization for Episode-Based Multi-Agent Reinforcement Learning . . . 79
6.1. Multi-Agent Stochastic Search . . . 79
6.1.1. Factorized MORE . . . 79
6.1.2. Policy Updates . . . 80
6.1.3. Illustrative Experiment . . . 80
6.2. Permutation Invariant Learning of Versatile Multi-Agent Behavior . . . 82
6.2.1. Maximum Entropy Episodic Policy Search . . . 82
6.2.2. Permutation Invariant VIPS . . . 83
6.2.3. Component Updates . . . 85
6.2.4. Weight Updates . . . 86
6.3. Conclusion . . . 86

7. Conclusion . . . 87
7.1. Summary of Contributions . . . 87
7.1.1. Fundamentals and State of the Art . . . 87
7.1.2. Local Communication Protocols for Learning Complex Swarm Behaviors with Deep Reinforcement Learning . . . 87
7.1.3. Deep Reinforcement Learning for Swarm Systems . . . 88
7.1.4. Robust Black-Box Optimization for Stochastic Search and Episodic Reinforcement Learning . . . 88
7.1.5. Black-Box Optimization for Episode-Based Multi-Agent Reinforcement Learning 88
7.2. Discussion and Outlook . . . 88
7.2.1. State Representation for Learning in Swarms . . . 89
7.2.2. Black-Box Optimization for Episode-Based Multi-Agent Reinforcement Learning 89

Bibliography . . . 91

A. Appendix for Chapter 4 . . . 101
A.1. Agent Kinematics . . . 101
A.2. Observation Model . . . 101
A.3. Task Specific Communication Protocols . . . 102

A.4. Controller for Double Integrator Dynamics . . . 102
A.5. Reward Functions . . . 103
A.5.1. Rendezvous . . . 103
A.5.2. Pursuit Evasion . . . 103
A.5.3. Pursuit Evasion with Multiple Evaders . . . 103
A.6. Policy Architectures . . . 103
A.6.1. Neural Network Embedding Policy . . . 104
A.6.2. Histogram Embedding Policy . . . 104
A.6.3. RBF Embedding Policy . . . 104
A.6.4. Concatenation Policy . . . 105

B. Appendix for Chapter 5 . . . 107
B.1. Derivation of CA-MORE Dual . . . 107
B.1.1. Mean Update . . . 107
B.1.2. Covariance Update . . . 108
B.2. Robust Target Normalization . . . 108
B.3. Hyper-Parameters . . . 110
B.4. Black-box Optimization Benchmarks . . . 111
B.5. Episodic RL . . . 111
B.5.1. Holereaching . . . 111
B.5.2. Table Tennis . . . 112
B.5.3. Beerpong . . . 112
B.5.4. Hopper Jump . . . 113

C. Appendix for Chapter 6 . . . 119
C.1. PI VIPS Derivations . . . 119

List of Figures

2.1. Diagram of the DeepSets architecture. The elements $\phi(\boldsymbol{x}_i)$ of the input set $\mathcal{X}$ are transformed into a latent representation and subsequently summed up, followed by further processing steps. The result is a permutation invariant function f. 11
2.2. This figure shows a diagram of an attention based architecture. The processing function ϕ represents a scalar dot-product between individual elements of the input set. 12
2.3. A graph representation of a swarm of eight agents where agents i and j are represented by their node features $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$. The edge feature $\boldsymbol{e}_k$ contains observations of agent i about agent j, as well as additional information sent from j to i. 13
2.4. Illustrations of information flow with message passing over three hops. 14

3.1. This Figure shows an illustration of the histogram-based observation model. Figure 3.1a shows an agent in the center of a circle whose neighborhood relations are to be captured by the histogram representation. The shaded green area is highlighted as a reference for Figures 3.1c and 3.1d. Figure 3.1b hereby shows the one dimensional histogram of agents over the neighborhood range d into four bins, whereas Figure 3.1c shows the histogram over the bearing angles ϕ into eight bins. Figure 3.1d finally shows the two dimensional joint histogram over range and bearing. 24
3.2. This diagram shows a model of our proposed policy with three hidden layers. The numbers inside the boxes denote the dimensionalities of the hidden layers. The plus sign denotes concatenation of vectors. 26
3.3. Illustration of the two cooperative tasks used in this paper. The green dots represent the agents, where the green ring segments located next to the agents indicate the short range IR front sensors. The outer green circles illustrate the maximum range in which distances / bearings to other agents can be observed, depending on the used observation model. **(a) Edge task:** The red rings show the penalty zones where the agents are punished, the outer green rings indicate the zones where legal edges are formed. **(b) Link task:** The red dots correspond to the two points that need to be connected by the agents. 28
3.4. Learning curves for (a), (b) the edge task and (c) the link task. The curves show the mean values of the average undiscounted return of an episode (i.e. the sum of rewards of one episode, averaged over the number of episodes for one learning iteration) over the learning process plus / minus one standard deviation, computed from eight learning trials. Intuitively, the return in the edge task corresponds to the number of edges formed during an episode of length 500 steps. In the link task, it is a measure for the quality of the link. **Legend:** 2DSP: two dimensional histogram over shortest paths, 2D: two-dimensional histogram over distances and bearings, 1D: two independent histograms over distances and bearing, d: distance only histogram, b: bearing only histogram, sensor: no histogram. 29

4.1. Illustration of (a) the neural network mean embedding policy, (b) the network architecture used for the RBF and histogram representation, and (c) for the simple concatenation of observations. The numbers inside the boxes denote the dimensionalities of the hidden layers. The color coding in (a) highlights which layers share the same weights. The plus sign denotes the mean of the feature activations. 38
4.2. Illustration of two neighboring agents facing the direction of their velocity vectors v^i and v^j, along with the observed quantities, shown with respect to agent i. The observed quantities are the bearing $\phi^{i,j}$ to agent j, agent j's relative orientation $\theta^{i,j}$ to agent i, their distance $d^{i,j}$ and a relative velocity vector $\Delta v^{i,j} = v^i - v^j$. In this trivial example, agent i's observed neighborhood size as well as the neighborhood size communicated by agent j are $|\mathcal{N}(i)| = |\mathcal{N}(j)| = 1$. 41
4.3. Learning curves for the rendezvous task with different observation models. The curves show the median of the average return $\bar{G}$ based on the top five trials on a log scale. **Legend:** NN++: neural network mean embedding of *comm* set, NN+: neural network mean embedding of *extended* set, NN: neural network embedding of *basic* set, RBF: radial basis function embedding of *basic* set, HIST: histogram embedding of *basic* set, CONCAT+: simple concatenation of *extended* set. 42
4.4. Visualization of a learned policy for the pursuit evasion task. The policy is learned and executed by 10 agents using a neural network mean embedding of the *extended* set. Pursuers are illustrated in blue, the evader is highlighted in red. Visualization of a learned policy for the rendezvous task. The policy is learned and executed by 20 agents using a neural network mean embedding of the *extended* set. 43
4.5. Comparison of the mean distance between agents in the rendezvous experiment achieved by the best learned policies and the consensus protocol. In (a) and (b), the policy is learned with 20 agents and executed by 20 and 100 agents, respectively. In (c) and (d), the policy is learned with 20 agents and executed by 20 and 10 agents. Results are averaged over 1000 episodes with identical starting conditions. 44
4.6. Learning curves for the pursuit evasion task with different observation models. The curves show the median of the average return $\bar{G}$ based on the top five trials on a log scale. **Legend:** NN++: neural network mean embedding of *comm* set, NN+: neural network mean embedding of *extended* set, RBF: radial basis function embedding of *basic* set, HIST: histogram embedding of *basic* set, CONCAT+: concatenation of *extended* set. 45
4.7. Visualization of a learned policy for the pursuit evasion task. The policy is learned and executed by 10 agents using a neural network mean embedding of the *extended* set. Pursuers are illustrated in blue, the evader is highlighted in red. 46
4.8. Performance comparison of the best learned policies and the optimization approach minimizing Voronoi regions in the pursuit evasion task with global observability. The curves show the probability that the evader is caught after t time steps. All policies are learned with 10 agents but executed with different agent numbers, as indicated below each subfigure. Results are averaged over 1000 episodes with identical starting conditions. 47
4.9. Performance comparison of the best policies in the pursuit evasion task with local observability. The curves show the probability that the evader is caught after t time steps. All policies are learned and executed by 20 agents. Results are averaged over 1000 episodes with identical starting conditions. 48

4.10. Learning curves for 50 agent pursuit evasion with 5 evaders. The curves show the median of the average return $\bar{G}$ based on the top five trials. **Legend:** NN+ 2x: two neural network mean embeddings of the *extended* set, RBF 2x: two radial basis function mean embeddings of the *basic* set, concat: simple concatenation of *extended* set. MSSK+, MSS+, MS+ and M+: Combinations of mean, standard deviation, skew and kurtosis of the features in the *extended* set. 49
4.11. Learning curves of different embedding and pooling architectures based on the *extended* set. The curves show the median of the average return $\bar{G}$ based on the top five trials on a log scale. **Legend:** MEAN: neural network mean embedding, SM: softmax feature pooling, MAX: max feature pooling. 49

5.1. This figure illustrates the concept of the evolution path in a 2D example and is adapted from Hansen [68]. In each figure, arrows with black heads correspond to an update of the mean. The evolution path is a smoothed sum over subsequent mean updates and depicted with an open head. In the left plot, the mean updates show no clear search direction and, as a result, the evolution path is short. The right plot shows the opposite where heavily correlated mean updates lead to a long evolution path. The center plot shows the desired case, where subsequent mean updates are uncorrelated. 63
5.2. This figure shows the function value (left) and the entropy of the search distribution (right) over the course of optimization of a 15-dimensional Rosenbrock function using different variants of MORE. The original MORE is shown in green, while coordinate ascent versions of more are orange and blue where CA-MORE indicates a fixed entropy reduction schedule and CAS-MORE indicates the adaptive entropy schedule using the step-size adaptation. For comparison, CMA-ES is plotted in red. 65
5.3. This figure shows the effects of data pre-processing on the optimization. We plot the function value over the course of optimization for a Rosenbrock (a) and Attractive Sector (b) function in 15 dimensions and turn off whitening and robust target normalization with clipping, respectively. We see that whitening becomes important for low entropy regimes near the optimum, while target normalization is especially important in case of outliers in the function values as can be seen for the Attractive Sector function. 67
5.4. This figure illustrates the robust target normalization scheme on 100 synthetic data points, generated by a reward function $y = -0.5\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}$. We sample $\boldsymbol{x}$ from a two dimensional multivariate normal distribution with zero mean and unit variance and pick 20 random y values and add Gaussian noise with a standard deviation of 10. Plots from top left to bottom right show histograms of the data after standardization of the input in each level of recursion of the procedure. The excess kurtosis is measured on the black data points in the interval (-3, 3) and only these data points are recursively treated again until the excess kurtosis of the clipped data is below the threshold of 0.55. Finally, the bottom right histogram shows the output with the red data points clipped to the minimum and maximum values of the remaining data points. Note, that in each plot the standardized data has zero mean and unit variance but the model quality suffers from the present outliers. 68

5.5. This figure shows the bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for 51 targets with target precision in $10^{[-8..2]}$ in 20-D representing the percentage of targets achieved over the number of function evaluations. The results are averaged over 15 instances of each function. As reference algorithm, the best algorithm from BBOB 2009 is shown as light thick line with diamond markers. The first two rows show the aggregated results for all functions and subgroups of functions. Additionally, we show the results of individual functions in the third and fourth row. Big thin crosses indicate the used budget median for the respective algorithm which is 10 000 n function evaluations for each trial of CAS-MORE. Runtimes to the right of the cross are based on simulated restarts and are used to determine a runtime for unsuccessful runs [72]. 70
5.6. This figure shows an illustration of the hole reacher task. The robot arms starts upright and smoothly reaches down the narrow hole. The policy is learned using CAS-MORE and shows a typical behavior that keeps a safe distance to the ground. 72
5.7. This figure shows the results of the hole-reacher experiment. Plotted is the expected performance of the mean in the left column, the mean of episode returns of the samples drawn in each iteration in the center left column, the percentage of trajectories that led to collisions with the ground in the center right column, and the distance of the end-effector at the end of the trajectory in the right column. The first row contains the results for the noise-less experiment with dense step-based reward, while the second row shows the results of the noisy experiment with dense step-based reward. The third and fourth row show the results for the noiseless and noisy experiments using a sparse episodic reward, respectively. 74
5.8. A successful episode of the table tennis task learned by CAS-MORE. 75
5.9. This figure shows the results of the table tennis experiment. Plotted is the expected performance of the mean in the left column, the mean of episode returns of the samples drawn in each iteration in the center column, and the percentage of trajectories that led to ball landing points within the last 10cm of the table in the right column. 75
5.10. A successful episode of the beerpong task learned by CAS-MORE. 76
5.11. This figure shows the results of the beerpong experiment. Plotted is the expected performance of the mean in the left column, the mean of episode returns of the samples drawn in each iteration in the center column, and the percentage of successful throws where the ball landed in the cup in the right column. 76
5.12. A successful episode of the hopper jump task learned by CAS-MORE. At first, the agent fully bends to then release into a fully extended jump. This way, it can reach a safe height not risking contact with the box's edge. 77
5.13. This figure shows the results of the hopper jump experiment. Plotted is the expected performance of the mean in the left column, the mean of episode returns of the samples drawn in each iteration in the center column, and the percentage of successful throws where the ball landed in the cup in the right column. 77

6.1. An episode of four arms reaching towards a single hole. 81
6.2. An episode of eight arms reaching towards a single hole. 82
6.3. This figure shows learning curves (left) and achieved average end-effector distances to the ground (right) over the optimization iterations for the four arm (upper row) and eight arm hole reacher. The first number in the legend indicates the number of samples drawn per iteration and the second number indicates the size of the reward buffer. Results are averaged over 5 seeds. 83

A.1. Learning curves for 20 agent rendezvous with (a) different activation functions for the mean embedding and (b) different layer numbers and sizes using a RELU activation function. The curves show the median of the average return $\bar{G}$ based on the top five trials. 104

B.1. Comparison of mean/std normalization and robust normalization on a penalty based reward function for the hole-reaching task. 109
B.2. Expected running time (ERT) divided by dimension versus dimension in log-log presentation 115
B.3. Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for 51 targets with target precision in $10^{[-8..2]}$ for all functions and subgroups in 5-D. As reference algorithm, the best algorithm from BBOB 2009 is shown as light thick line with diamond markers. 116
B.4. Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for 51 targets with target precision in $10^{[-8..2]}$ for all functions and subgroups in 20-D. As reference algorithm, the best algorithm from BBOB 2009 is shown as light thick line with diamond markers. 116
B.5. Empirical cumulative distribution of simulated (bootstrapped) runtimes, measured in number of objective function evaluations, divided by dimension (FEvals/DIM) for the 51 targets $10^{[-8..2]}$ in dimension 5. 117

List of Tables

4.1. Number of times the algorithm discovered policies that led to a successful catch. 50

5.1. Runtime comparison for various episodic reinforcement learning algorithms. We measure the time the algorithm takes for one iteration without the sampling process. 73

6.1. Number of model parameters depending on the number of agents n for a 15-dimensional movement primitive per agent. 81

B.1. Empirically found default hyper-parameters for CAS-MORE based on the problem dimensionality n. 109
B.2. Hyper-parameters for the deep RL and BBRL experiments. 110

List of Algorithms

1. Robust target normalization 109

1. Introduction

Artificial agents, may it be robots in the physical world or virtual agents such as chatbots, influence our daily lives. They assemble cars, explore surfaces of far away planets, or, most recently, are capable of answering questions regarding general knowledge in human like language or writing code based on natural language instructions. While, for a long time, these agents had to be hand-programmed to achieve their tasks, there has been a shift in paradigm towards data-driven approaches such as Machine Learning (ML) in the past years. This approach should enable agents to not only work in confined environments such as factories, but, ultimately, should lead to a coexistence between humans and artificial agents. This new approach has been made possible by the ever increasing computational resources, as well as the ability to store and distribute large databases. Sometimes, as in machine translation, labelled data is available, e.g., the input sentence in one language and the desired output in the target language, to train a system. Often however, especially in robotics, we need to generate data during the optimization process. Through means of trial and error, the behavior of an agent is optimized to maximize a performance measure describing the task. This case can be formalized using the framework of Reinforcement Learning (RL). The agent observes its environment, for example through a camera or proprioceptive feedback, and chooses an action using its policy, a function mapping from observations to actions. After the action is applied, the environment emits a reward signal indicating the quality of the taken action in the current state, and a new observation based on the new state of environment.

Two key challenges for achieving efficient RL, which we will investigate in this thesis, are the agents' capabilities of perceiving the world and their ability to efficiently explore their environment. We will study these problems in two different setups. First, we look at information representation in scenarios where multiple agents are working together two solve a collaborative task. This setup is especially suited for exploring information representation as the amount of information to be processed increases with the number of agents within the neighborhood of an agent. In order to perform well informed actions, each agent has to process this information and find a representation that, ideally, is independent of the number and order of its neighbors.

Second, we look at the problem of efficiently exploring large action-spaces. To this end, we study the case of episodic reinforcement, where the behavior is encoded using movement primitives in the form of a low dimensional parameter vector. Compared to step-based RL, where exploration usually happens in the agent's action space, exploration here happens directly in the parameter space. Here, correlations between the parameters become increasingly important. Additionally, the learning process has to be robust in situations where execution of the agent's policy is subject to noise. After studying in-depth the optimization of the movement primitive parameters for a single agent, we provide a formulation for the multi-agent setup.

1.1. Contributions

This thesis addresses two fundamental problems in reinforcement learning. On the one hand, the problem of learning concise representations of an environment in the presence of a large number of

observed entities. On the other hand, the problem of efficient exploration in difficult manipulation tasks. This section serves as a brief summary of the contributions.

1.1.1. Local Communication Protocols for Learning Complex Swarm Behaviors with Deep Reinforcement Learning

Swarm systems constitute a challenging problem for reinforcement learning (RL) as the algorithm needs to learn decentralized control policies that can cope with limited local sensing and communication abilities of the agents. While it is often difficult to directly define the behavior of the agents, simple communication protocols can be defined more easily using prior knowledge about the given task. In this paper, we propose a number of simple communication protocols that can be exploited by deep reinforcement learning to find decentralized control policies in a multi-robot swarm environment. The protocols are based on histograms that encode the local neighborhood relations of the agents and can also transmit task-specific information, such as the shortest distance and direction to a desired target. In our framework, we use an adaptation of Trust Region Policy Optimization to learn complex collaborative tasks, such as formation building and building a communication link. We evaluate our findings in a simulated 2D-physics environment, and compare the implications of different communication protocols.

1.1.2. Deep Reinforcement Learning for Swarm Systems

Recently, deep reinforcement learning (RL) methods have been applied successfully to multi-agent scenarios. Typically, the observation vector for decentralized decision making is represented by a concatenation of the (local) information an agent gathers about other agents. However, concatenation scales poorly to swarm systems with a large number of homogeneous agents as it does not exploit the fundamental properties inherent to these systems: (i) the agents in the swarm are interchangeable and (ii) the exact number of agents in the swarm is irrelevant. Therefore, we propose a new state representation for deep multi-agent RL based on mean embeddings of distributions, where we treat the agents as samples and use the empirical mean embedding as input for a decentralized policy. We define different feature spaces of the mean embedding using histograms, radial basis functions and neural networks trained end-to-end. We evaluate the representation on two well-known problems from the swarm literature — rendezvous and pursuit evasion — in a globally and locally observable setup. For the local setup we furthermore introduce simple communication protocols. Of all approaches, the mean embedding representation using neural network features enables the richest information exchange between neighboring agents, facilitating the development of complex collective strategies.

1.1.3. Robust Black-Box Optimization for Stochastic Search and Episodic Reinforcement Learning

Black-box optimization is a versatile approach to solve complex problems where the objective function is not explicitly known and no higher order information is available. Due to its general nature, it finds widespread applications in function optimization as well as machine learning, especially episodic reinforcement learning tasks. While traditional black-box optimizers like CMA-ES may falter in noisy scenarios due to their reliance on ranking-based transformations, a promising alternative emerges in the form of the Model-based Relative Entropy Stochastic Search (MORE) algorithm. MORE can be derived from natural policy gradients and compatible function approximation and directly optimizes the

expected fitness without resorting to rankings. However, in its original formulation, MORE often cannot achieve state of the art performance. In this paper, we improve MORE by decoupling the update of the search distribution's mean and covariance and an improved entropy scheduling technique based on an evolution path resulting in faster convergence, and a simplified model learning approach in comparison to the original paper. We show that our algorithm performs comparable to state-of-the-art black-box optimizers on standard benchmark functions. Further, it clearly outperforms ranking-based methods and other policy-gradient based black-box algorithms as well as state of the art deep reinforcement learning algorithms when used for episodic reinforcement learning tasks.

1.1.4. Black-Box Optimization for Episode-Based Multi-Agent Reinforcement Learning

We propose a movement primitive based reinforcement learning framework for non-contextual multi-agent path-planning problems in continuous state and action spaces. To this end, we leverage recent advances in episodic policy search and extend them to the multi-agent setting. Instead of learning a joint policy for all agents, individual agents' policies are updated independently based on a factorized model of the cooperative reward. Building on this approach, we show how to learn versatile behavior in scenarios with homogeneous agents. Learning versatile solutions for multi-agent tasks is a hard problem due to the inherent redundancy of solutions introduced by permutations of homogeneous agents. While contemporary deep reinforcement learning based methods are in principle very powerful, they usually only allow for single solutions as they use uni-modal Gaussian policies. Additionally, they lack the simplicity and interpretability of simpler policy parameterizations. We mitigate the problem of redundant solutions by introducing an auxiliary distribution factoring in permutations of the solution space. Preliminary results are presented in simulated 2D robotics environments.

1.2. Structure of Thesis

We start this thesis by laying the foundations of sequential decision making and information representation in Chapter 2. We first give the fundamentals for reinforcement learning in Section 2.1.3. This includes background on deep reinforcement learning in Section 2.1.3.3, multi-agent reinforcement learning in Section 2.1.4, and trust-region reinforcement learning in Section 2.4.1. Additionally, we dedicate a section to information representation using attention and graph neural networks (Section 2.3).

Chapter 4 introduces a framework that applies concepts from Section 2.3 in a swarm reinforcement learning context. It uses mean embedding pooling to represent an agent's local perception of its environment and a TRPO-based deep reinforcement learning algorithm to solve cooperative tasks.

Chapter 5 proposes an episodic reinforcement learning algorithm which is especially suited for hard exploration problems. It improves upon the MORE algorithm, by introducing several algorithmic extensions. An improved model learning scheme, more robust update equations, and a sophisticated exploration approach based on an evolution path lead to excellent results on non-deterministic episodic reinforcement learning problems.

Chapter 6 contains an outline for the application of the algorithm presented in Chapter 5 in multi-agent settings and possible future work directions.

Finally, Chapter 7 concludes this thesis.

2. Fundamentals and State of the Art

This thesis is concerned with two important topics in reinforcement learning: efficient information representation and effective exploration of the parameter space. In this chapter, we lay out the foundations necessary to understand the methods in this thesis and provide an overview over existing work.

We start with a general introduction to sequential decision making problems and present the foundations for reinforcement learning as well as multi-agent reinforcement learning. In the following, we present an episode-based view on reinforcement learning along with its advantages and disadvantages over the step-based view and introduce probabilistic movement primitives, an episode-based policy representation used in this thesis. We then provide the background over state representations for swarms and how they are used in multi-agent RL. Finally, we give an overview over action-space and parameter-space based exploration strategies.

2.1. Sequential Decision Making

Sequential decision making refers to the process of making a series of interconnected choices, or actions, over time, where each decision affects the available options and outcomes for subsequent steps. Critically, the decisions made at each step are influenced by prior choices and expected future consequences. The decision maker, typically called the *agent*, can be all kinds of entities, for example humans, robots, or software programs. A prominent example for a sequential decision making problem is the game of chess. Players make successive moves, considering their opponent's potential responses and adjusting their strategy accordingly.

2.1.1. Markov Decision Processes

Sequential decision making problems can be formalized using the concept of Markov decision processes (MDPs). An MDP is defined as a tuple $\{\mathcal{S}, \mathcal{A}, P, R\}$ where $\mathcal{S}$ is a set of states within the environment, $\mathcal{A}$ is a set of actions available to the agent, P is a transition probability function such that given a state s and action a at time-step t, the probability of the next state s' is given by $P(s_{t+1} = s' \mid s_t = s, a_t = a)$, and a reward function R, an immediate performance measure $r_t = R(s_{t-1} = s, a_{t-1} = a)$ of the agent's action a taken in state s. The agent takes actions according to its policy π, a function that maps from states to actions. This function can either be deterministic, such that $a = \pi(s)$, or stochastic. In the stochastic case, actions are sampled as $a \sim \pi(\cdot \mid s)$.

2.1.2. Value Functions

A useful quantity to describe the quality of a policy on a certain problem are value functions. Two important value functions exist. A state value function $V^\pi(s) = \mathbb{E}_\pi[\sum_{t=0}^{T} \gamma^t R(s_t, a_t) \mid s_t = s]$ that yields the expected return when starting in state $s_t = s$ and subsequently following the policy. Furthermore, the state-action value function, also known as Q-function, $Q^\pi(s, a) = \mathbb{E}_\pi[\sum_{t=0}^{T} \gamma^t R(s_t, a_t) \mid s_t = s, a_t = a]$

that yields the expected return starting in state $s_t = s$ and additionally taking action $a_t = a$, subsequently following π. Here, $\mathbb{E}_\pi = \mathbb{E}_{s\sim d^\pi,\, a\sim\pi}$ denotes that actions are sampled from the policy π and states are visited according to the state distribution d^π induced by π. $0 < \gamma \leq 1$ is a discount factor trading off the importance of short term and long term rewards while ensuring the sum converges to a finite value in case $T \to \infty$.

Given a fixed policy π, the value function corresponding to a task can be computed using a procedure called policy evaluation. This procedure leverages a recursive formulation of the value function,

$$V^\pi(s) = \mathbb{E}_\pi \left[r_{t+1} + \gamma V^\pi(s') \mid s_t = s \right],$$

which is also known as the Bellman equation for V^π. If the environment dynamics are known and the state and action spaces are finite, one can iteratively compute the policy's value function using the update rule

$$\begin{aligned} v_{k+1}(s) &= \mathbb{E}_\pi \left[r_{t+1} + \gamma v_k(s') \mid s_t = s \right] \\ &= \sum_a \pi(a \mid s) R(s,a) + \gamma \sum_{s'} P(s' \mid s, a) v_k(s') \end{aligned}$$

for all states $s \in \mathcal{S}$ and any initialization of v_0. The Q-function can be obtained from the state value function as

$$Q^\pi(s,a) = R(s,a) + \gamma \sum_{s'} P(s' \mid s, a) V^\pi(s')$$

Finally, the advantage of an action a over other actions is given by $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$.

2.1.3. Reinforcement Learning

We are interested in an optimal policy π^* for a task that maximizes the expected cumulative reward

$$J = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t) \right].$$

If we had access to the transition probability function and reward function, we could simply use planning algorithms to arrive at it. However, this is usually not the case. Instead, we can use reinforcement learning (RL), a sub-domain of machine learning, where the agent learns to fulfil a task based on continued interactions with its environment. We refer to Sutton and Barto [148] for an extensive introduction to the topic and only provide a concise overview over the foundations of model-free methods which are the basis for the algorithms presented in this thesis.

2.1.3.1. Value-Based Methods

Value-based RL algorithms provide methods for estimating an optimal policy from a learned value function. One of the most prominent examples is Q-learning [153]. Based on the Bellman equation, the update rule is given by

$$Q_{k+1}(s,a) = (1 - \alpha_k) Q_k(s,a) + \alpha_k (R(s,a) + \gamma \max_{a'} Q_k(s', a'))$$

where α_k is a learning rate. Provided that all states and actions are continually visited, this learning rule is guaranteed to converge to the optimal Q-function Q^*. The optimal policy is given as

$$\pi^*(a \mid s) = \begin{cases} 1 & \text{if } a = \arg\max_a Q^*(s,a) \\ 0 & \text{else.} \end{cases}$$

2.1.3.2. Policy Gradient Methods

A very different approach from value-based methods to learning a policy are policy gradient methods [149, 155]. Whereas the policy in value-based methods is implicitly defined through its value function, policy gradient methods learn an explicit policy function $\pi(a \mid s; \boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$. The only requirement regarding the parameterization is that π is differentiable with respect to $\boldsymbol{\theta}$. The parameters are updated through gradient ascent

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla_{\boldsymbol{\theta}} J$$

where $\nabla_{\boldsymbol{\theta}} J$ is the gradient of the performance measure with respect to the policy parameters $\boldsymbol{\theta}$. The question remains how to obtain this gradient. It is provided by the policy gradient theorem as

$$\nabla_{\boldsymbol{\theta}} J = \int_{\mathcal{S}} d^{\pi}(s) \int_{\mathcal{A}} \nabla_{\boldsymbol{\theta}} \pi(a \mid s; \boldsymbol{\theta}) Q^{\pi}(s, a) \mathrm{da}\, \mathrm{ds}.$$

Using an approximation $\hat{Q}$ for Q^{π} and the identity $\nabla f(\boldsymbol{x}) = \boldsymbol{x} \nabla \log f(\boldsymbol{x})$, we can rewrite the policy gradient as

$$\nabla_{\boldsymbol{\theta}} J \approx \mathbb{E}_{\pi} \left[\hat{Q}(s, a) \nabla_{\boldsymbol{\theta}} \log \pi(a \mid s; \boldsymbol{\theta}) \right]$$

which can be approximated from samples. Policy gradient methods are especially useful for continuous action spaces and are therefore the algorithms of choice in this thesis.

2.1.3.3. Deep Reinforcement Learning

The problems we aim to solve in this thesis are comprised of infinite continuous state and action spaces, and, therefore, cannot be solved using classic reinforcement learning algorithms like Q-learning. In order to solve these problems, we need to rely on approximations of value functions and tractable parameterizations of the policy function. While linear feature approximators showed some success in small continuous state spaces, it was only possible to solve large problems after breakthroughs from deep learning found there way into reinforcement learning, opening the field of deep reinforcement learning. Here, neural networks are usually used to represent the policy and approximated value functions. Perhaps the most prominent example is the Deep Q-Networks algorithm [105] where a Q-function, approximated by a deep neural network, directly mapped from pixels of Atari games to actions, achieving human-level play which was unprecedented at that time. For the deep RL algorithm used in this thesis, we refer to Section 2.4.1.2.

2.1.4. Multi-Agent Reinforcement Learning

In the real world, many tasks involve not just one but several agents, either cooperating to solve a single task, or even competing, pursuing conflicting interests [140]. This can include tasks like traffic management [35], financial markets [138], or manufacturing [102]. While, in theory, any multi-agent system with full observability can be treated as a single agent problem where a centralized controller chooses a joint action for all agents, this approach quickly becomes infeasible with an increasing number of agents or if sensing of the system can only be realized locally on an agent level. To this end, we introduce in the following sections the formal extensions necessary to frame a multi-agent reinforcement learning (MARL) problem, and give an overview over current state of the art algorithms to solve them.

2.1.4.1. Decentralized POMDPs

Decentralized Partially Observable Markov Decision Processes (Dec-POMDPs) have been introduced to formalize multi-agent learning problems where a single task has to be accomplished by a group agents. They are an extension of MDPs and are given by a tuple $\{\mathcal{S}, \{\mathcal{A}_i\}, P, R, \{\Omega_i\}, O\}$ where $\mathcal{S}$ is a set of global states, $\mathcal{A}_i$ is the set of actions for agent i, P is a transition probability function between states, and R is a reward function. The partial observability of the system is described by the set of observation Ω_i for agent i and a set of conditional observation probabilities O.

2.1.4.2. Deep MARL

In the same fashion as presented in Section 2.1.3.3, deep MARL algorithms try to solve sequential decision making problems using neural networks as function approximators. Wong et al. [158] identified four main challenges that arise especially in MARL that algorithms need to overcome:

1. The problem of computational complexity,
2. non-stationarity introduced by all agents learning concurrently,
3. partial observability,
4. and credit assignment.

Until now, many dedicated algorithms to solve multi-agent problems have been proposed, each tackling one or more of these problems. From a learning perspective, the centralized training, decentralized control (CTDE) paradigm offers a remedy for the high computational complexity. In this paradigm, the policy acts locally on each agent's own perception of the world. Tuples of observations, actions and rewards from all agents are then collected and stored centrally for the learning algorithm to access. After the policy update, the new policy is rolled out to all agents again and the process repeats. This scheme is applied in many deep MARL approaches with different focuses. Value-function methods, as studied in Sunehag et al. [147] and Rashid et al. [127] and many others, aim to decompose the global value function into local agent-based value functions to use them for control. Furthermore, Multi-Agent DDPG [101] and its successors can learn continuous policies for competitive environments. In order to overcome the non-stationarity, as well as partial observability, algorithms that try to model the behavior of other agents have been developed. Notably, AlphaZero [141] achieves superior results on classic board games such as Go and chess using neural network policies and Monte-Carlo tree search. Learning to explicitly communicate as in Foerster et al. [47] additionally helps. Finally, reward shaping strategies have been introduced to deal with the credit assignment problem. Foerster et al. [46], for example, introduce COMA to reason about the influence of an agent's action on the outcome of a task. The method uses a counterfactual baseline to marginalise out a single agent's action. The method we propose in Chapter 4 makes use of CTDE and mainly tackles the first and, to some extent, the third problem.

2.2. Episode-Based Reinforcement Learning

So far, we described a step-based view of the policy where the agent observes the current state of the environment and chooses an action for the next time step. Alternatively, a plan for the whole motion can be created and then executed by a low level controller such as a PD controller. In this episode-based view of the policy, the parameters $\boldsymbol{w}$ of an upper level motion planning module are improved based

on the quality of the execution of the whole trajectory. Instead of using individual transition tuples $\{s_t, a_t, r_{t+1}, s_{t+1}\}$, whole trajectories $\tau = \{s_0, a_0, r_1, s_1, a_1, r_1, \ldots, s_T, r_T\}$ are used to update the policy. The trajectory can, for example, be represented by movement primitives that use a low dimensional parameter $\boldsymbol{w}$ to generate a full trajectory. The policy $\pi(\boldsymbol{w})$ is defined as a function over these movement primitive parameters and can be parameterized depending on the nature of the problem. In contextual problems, either a linear mapping or a neural network [114] can be used to map the context to the movement primitive parameters. If no context exists, we can directly learn a mean and covariance of a Gaussian search distribution, as presented in Chapter 5.

We now present the episode-based policy representation used in this thesis and discuss advantages and disadvantages of episode-based reinforcement learning compared to step-based reinforcement learning.

2.2.1. Probabilistic Movement Primitives as Episode-Based Policy Parameterizations

Probabilistic movement primitives (ProMPs) [120] are a method for representing distributions over trajectories. Based on basis function representations, they allow a low dimensional yet expressive encoding of movements. We denote a trajectory $\boldsymbol{\tau} = \{q_t\}_{t=0,\ldots,T}$ for a single degree of freedom over time. The position q_t and velocity $\dot{q}_t$ is represented by a weight vector $\boldsymbol{w}$ and is computed as a linear basis function model

$$\boldsymbol{y}_t = \begin{bmatrix} q_t \\ \dot{q}_t \end{bmatrix} = \boldsymbol{\Phi}_t \boldsymbol{w} + \boldsymbol{\epsilon}_y$$

where $\boldsymbol{\Phi}_t = \begin{bmatrix} \phi_t & \dot{\phi}_t \end{bmatrix}^\mathsf{T} \in \mathbb{R}^{2\times n}$ is the matrix of n time-dependent basis functions and $\boldsymbol{\epsilon}_y \sim \mathcal{N}(\mathbf{0}, \Sigma_y)$ is zero-mean i.i.d Gaussian noise. For stroke-based movements, as we will investigate them in this thesis, we use radial basis functions

$$b_i(z) = \exp\left(-\frac{(z_t - c_i)^2}{2h}\right)$$

where h defines the width and c_i the center of the ith basis function. The phase variable z decouples the movement from the time signal. After normalization, the basis functions are given as

$$\phi_i(z_t) = \frac{b_i(z_t)}{\sum_{j=1}^{n} b_j(z_t)}.$$

The probability of observing a trajectory $\boldsymbol{\tau}$ given a weight vector $\boldsymbol{w}$ is given as

$$p(\boldsymbol{\tau} \mid \boldsymbol{w}) = \prod_t \mathcal{N}(\boldsymbol{y}_t \mid \boldsymbol{\Phi}_t \boldsymbol{w}, \Sigma_y).$$

In the reinforcement learning algorithm presented in Chapter 5, we learn a distribution $\pi(\boldsymbol{w})$ over the movement primitive parameters $\boldsymbol{w}$.

2.2.2. Advantages and Disadvantages of Episode-Based Reinforcement Learning

Episode-based reinforcement, as the name suggests, is especially helpful if a meaningful reward signal is only returned after a full episode. Furthermore, it can deal with non-Markovian or sparse rewards and the policy can be realized with very few parameters. When used with movement primitives, it provides smooth control trajectories, often leading to more energy efficient behavior [114]. On the

downside, an episode-based policy lacks the ability to react to changes in the environment as the whole motion plan is created prior to execution while the policy in the step-based view is able to directly react to feedback from the environment. It can also be less data-efficient than step-based reinforcement as it does not exploit the time-series structure of the problem, especially in the dense reward setting.

2.3. Information Representation in Swarms

Parts of this thesis are concerned with sequential decision making in swarms, i.e., a population of simple homogeneous agents [41], which are a special case of multi-agent systems. By means of collaboration, they are able to solve tasks that are not solvable by individual agents. Additionally, the performance is not affected by a small number of agents failing. This kind of phenomenon can often be found in nature where flocking birds travel large distances or colonies of ants forage for food and supplies [25]. While potentially allowing for more complex tasks to be solved, controlling multiple agents also adds complexity to the learning process. In swarms, not only the learning algorithm itself needs attention. Additionally, an important question is how to efficiently process swarm information, an orthogonal problem which many multi-agent RL works do not explicitly try to solve. A characteristic of swarms that allows for the complexity of the problem to be reduced is looking at groups of homogeneous entities, i.e., agents or objects that are identical in their physical properties. Within these groups, exchanging agents A and B has no influence on the outcome of a planned motion or the way other agents react on the overall agent configuration. Yet, these permutations all constitute individual solutions to a problem and, thus, complicate the learning process, when not explicitly taking these invariances into account. A good representation therefore needs to be independent of the number and order of agents, while avoiding the *curse of dimensionality*, i.e., the problem that with increasing problem dimensionality we need exponentially more data to sufficiently explore the problem space.

In this section, we provide two views of swarm data and how they can be processed. First, we look at local information processing at an agent level and present the current state in processing set data. Second, we take a more abstract view of a swarms in the form of a graph and give a brief outline of graph processing with neural networks.

2.3.1. Set Representation of Local Observations

At an abstract level, locally observed information about a group of homogeneous in an agents neighborhood takes the form of an unordered set. An element $\boldsymbol{x}_i$ of the set $\mathcal{X} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\}$ contains important quantities for decision making such as geometric and kinematic features of neighboring entities. These include, for example, distances, bearing angles, or relative velocities to neighboring agents, points of interest, or objects to interact with. We want to make use of the inherent structure of the data that factors in the knowledge about the existing invariance and avoid learning redundant solutions. In the reinforcement learning context, this translates to the output of the policy π being invariant to permutations $\mathcal{P}_\mathcal{X}$ of the set of input elements, i.e.

$$\pi(\mathcal{X}) = \pi(\mathcal{P}_\mathcal{X}) \quad \forall \mathcal{P}_\mathcal{X} \in S(\mathcal{X})$$

where $S(\mathcal{X})$ is the set of all permutations of $\mathcal{X}$. An efficient representation therefore needs to be invariant to the order of the elements in the set. In Chapter 3, we provide a method based on handpicked features over sets, while in Chapter 4, we extend this approach to learned feature representations. In the following sections, we provide an in-depth overview over the underlying processing techniques.

2.3.1.1. DeepSets

Standard neural network based architectures, such as feed forward MLPs, usually impose some sort of order on the input and are therefore unsuitable for processing sets directly. This shortcoming led to the development of DeepSets [161], a permutation invariant and equivariant deep learning architecture. The general idea is to first transform each element $\boldsymbol{x}_i$ into a representation $\phi(\boldsymbol{x}_i)$ using either hand-designed or learned feature transformations, such as histograms or neural network layers. Afterwards, the representations $\phi(\boldsymbol{x}_i)$ are summed up and may further be processed by subsequent neural network layers. The permutation invariant function is given by

$$f(\mathcal{X}) = \rho\left(\sum_i \phi(\boldsymbol{x}_i)\right)$$

Since all elements are transformed independently, the mapping $\phi(\boldsymbol{x}_i)$ can efficiently be parallelized. An illustration if the architecture can be found in Figure 2.1. In Chapter 4, we will show how to incorporate this type of representation into a swarm reinforcement learning framework.

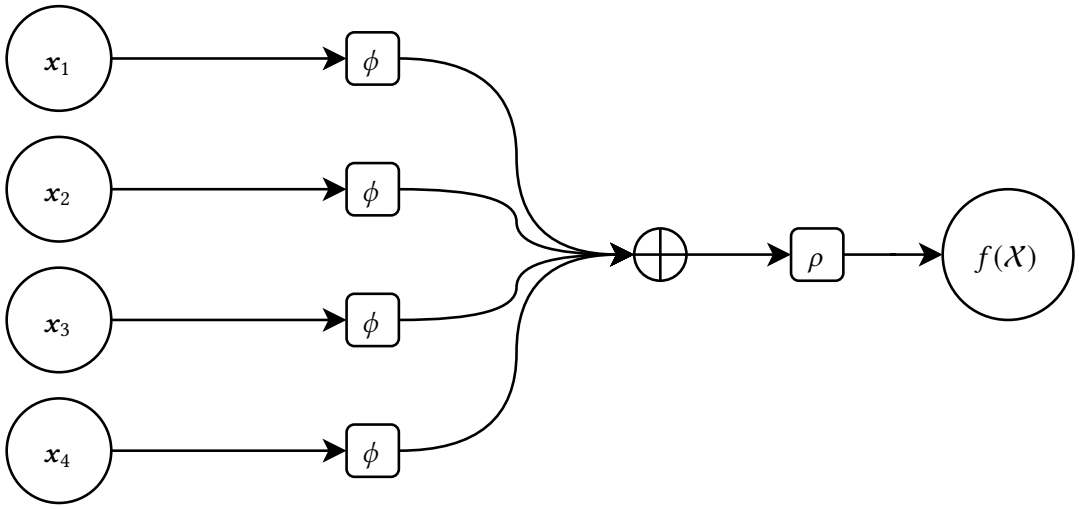


Figure 2.1.: Diagram of the DeepSets architecture. The elements $\phi(\boldsymbol{x}_i)$ of the input set $\mathcal{X}$ are transformed into a latent representation and subsequently summed up, followed by further processing steps. The result is a permutation invariant function f.

2.3.1.2. Relational Approaches

The DeepSets architecture provides a permutation invariant function mapping by first transforming each element of the set independently and pooling these embeddings over the whole set. While transforming individual elements, in principle, is sufficient for universal function approximation, it is often beneficial to incorporate explicit relational reasoning into the model [20]. To this end, several architectures have been introduced. Relation Networks [135] provide a simple neural network module of the form

$$\mathrm{RN}(\mathcal{X}) = \rho\left(\sum_{i,j} \phi(\boldsymbol{x}_i, \boldsymbol{x}_j)\right),$$

where ρ and ϕ are MLPs. By summing over embeddings of pairs of elements, the output becomes permutation invariant. This module is not primarily developed for approximating functions on sets but

rather serves as a drop-in module for convolutional or recurrent neural networks to allow for explicit relational reasoning.

Self-attention extends this concept by an additional weighting step on input pairs of the set. In the simplest case, an attention module for elements of a set is given by

$$\mathrm{Att}(\mathcal{X}) = \omega(\mathcal{X}\mathcal{X}^{\mathsf{T}})\mathcal{X}$$

where $\omega = \mathrm{softmax}(\cdot/\sqrt{d})$ is a scaled softmax to ensure the weights sum up to one. A permutation invariant architecture utilizing attention modules is the Set Transfomer [93]. An illustration of relational architectures using attentioin can be found in Figure 2.2.

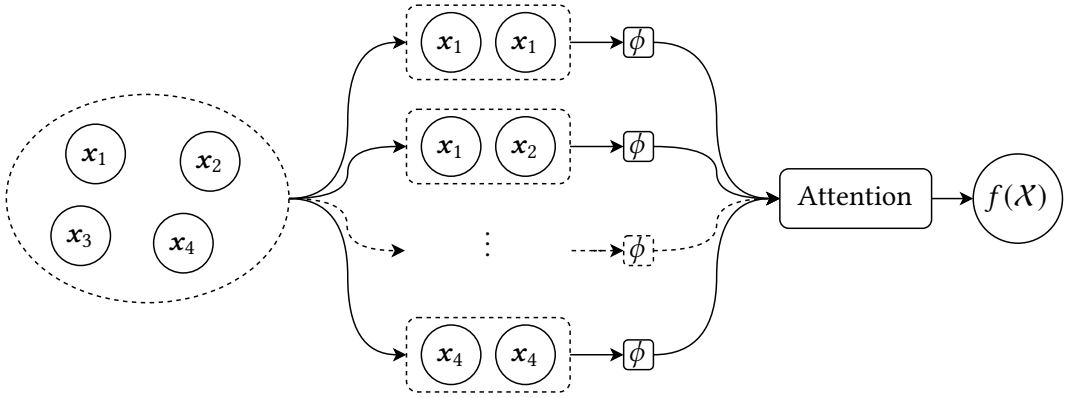


Figure 2.2.: This figure shows a diagram of an attention based architecture. The processing function ϕ represents a scalar dot-product between individual elements of the input set.

2.3.1.3. Generalization

Recently, Wagstaff et al. [152] realized that both, Deep Sets and Attention-based architectures are special cases of the Janossy pooling paradigm [108]. In its most expressive form, a permutation invariant function is constructed from processing all possible permutations of the input set with a permutation sensitive neural network, followed by sum or mean pooling. The permutation invariant output of the pooling operation can then further be processed without any constraints. Mathematically, the function is given by

$$f(\mathcal{X}) = \rho\left(\frac{1}{|S(\mathcal{X})|}\sum_{\sigma\in S(\mathcal{X})}\phi(\sigma)\right).$$

The expressiveness of this formulation comes at a very high computational cost as the number of permutations grows linearly with the with the cardinality of the set, which grows factorially with the number of elements. Murphy et al. [108] propose three ways of limiting the computational complexity: sorting, sampling, and restricting the permutations to k-tuples. Sorting only considers a single permutation, based on an inherent or learned order of data-points. Sampling, on the other hand, considers a small number of randomly chosen permutations from $S(\mathcal{X})$. Finally, models such as DeepSets as well as attention-based approaches fall into the category of k-tuples. Specifically, DeepSets is the result of setting $k = 1$, while attention-based approaches are derived from $k = 2$.

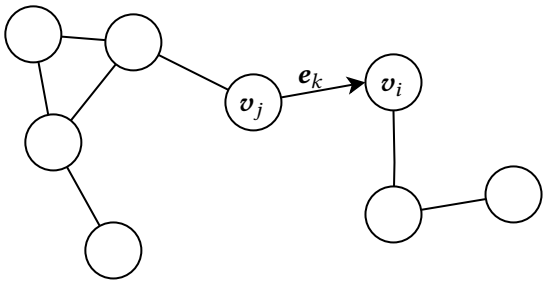


Figure 2.3.: A graph representation of a swarm of eight agents where agents i and j are represented by their node features v_i and v_j. The edge feature e_k contains observations of agent i about agent j, as well as additional information sent from j to i.

2.3.2. Swarm as a Graph

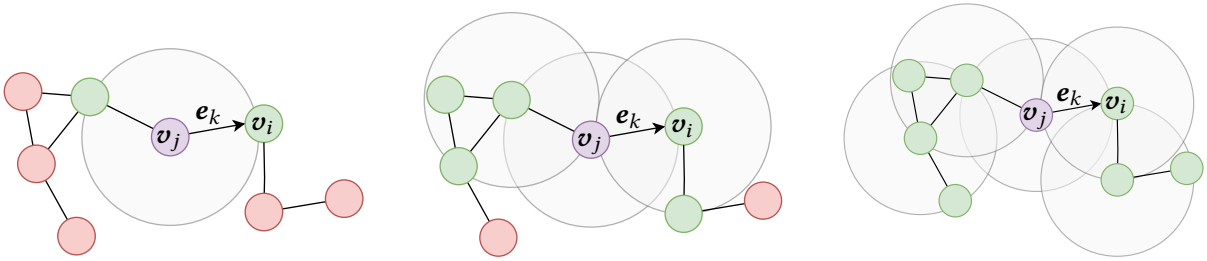
The set representation is a natural choice for information representation at an agent level. Another abstraction of the whole swarm can be made by representing it as a graph $\mathcal{G} = (V, E)$. Agents or objects are treated as nodes, also called vertices, $V = \{v_1, v_2, \ldots, v_N\}$ where v_i contains node i's attributes. Interactions and relations between nodes are modelled as undirected edges $E \subset V \times V$. Additionally, there can be directed edges e_k with an explicit sender s_k and receiver r_k between two nodes v_i and v_j. These edges can represent features such as observations from agent i about agent j or even explicit messages from agent j to agent i. In general, there can exist undirected or (multiple) directed edges or both. In the swarm context, a node feature could, for example, be an agent's location and velocity, as well as an indicator of its current health state, or whether it is carrying an object in a transportation task. Edge features on the other hand may include observations about neighboring agents, such as relative velocities or bearing angles. Finally, global features are collected in a feature vector g. An example graph can be seen in Figure 2.3.

2.3.2.1. Graph Neural Networks

Graph neural networks (GNNs) provide a neural network based architecture to compactly represent the information present in a graph. By now, there exists a plethora of literature that uses different approaches to graph-based problems, such as simulating physical interactions [134] or learning multi-agent dynamics [145]. Although the method presented in this thesis makes use of the graph notation, we do not make use of explicit GNN architectures and therefore only introduce the general architecture and provide a brief overview of a specific GNN flavor. An in-depth overview covering geometric deep learning can be found in Bronstein et al. [28].

Abstractly speaking, a GNN is comprised of three building blocks: Permutation equivariant layers computing transformations on a node level, local pooling operations providing graph coarsening, and a global permutation invariant pooling layer [28]. These building blocks can be implemented in different ways. Particularly suited for processing swarm information are Message Passing Networks [134] which allow an implicit form of message passing that goes beyond "1-hop" information exchange within an agent's neighborhood $\mathcal{N}_i = \{j \mid \{v_i, v_j\} \in E\}$[1]. Thus, the graph representation lends itself naturally to scenarios with limited observability. In graph terms, this relates to a connected, but not complete graph or digraph, where agents cannot observe all other agents in each time-step.

[1] In comparison, DeepSets approaches are only able to process "1-hop" information.



(a) First hop. **(b)** Second hop. **(c)** Third hop.

Figure 2.4.: Illustrations of information flow with message passing over three hops.

The message passing step consists of updates of the latent edge, node, and global features. Initially, node, edge, and global features are linearly transformed into representations $\boldsymbol{H}_v^0$ where a row $\boldsymbol{h}_{v,i}^0$ corresponds to the ith node's embedding, $\boldsymbol{H}_e^0$ where a row $\boldsymbol{h}_{e,k}^0$ corresponds to the embedding of edge feature $\boldsymbol{e}_k$, and $\boldsymbol{h}_g^0$. The edge features are updated according to

$$\boldsymbol{h}_{e,k}^{l+1} = \phi_e^l(\boldsymbol{h}_{e,k}^l, \boldsymbol{h}_{v,i}^l, \boldsymbol{h}_{v,j}^l, \boldsymbol{h}_g^l)$$

where an MLP transforms latent edge features of edge k and the features of the corresponding sending and receiving nodes $(i, j) = (s_k, r_k)$. Next, the node features are updated according to

$$\boldsymbol{h}_{v,i}^{l+1} = \phi_v^l\left(\boldsymbol{h}_{v,i}^l, \bigoplus_{s_k \in \mathcal{N}_i} \boldsymbol{h}_{e,k}^l, \boldsymbol{h}_g^l\right)$$

taking into account the edge information sent from node s_k. Finally, the global feature vector is update according to

$$\boldsymbol{h}_g^{l+1} = \phi_g^l\left(\bigoplus \boldsymbol{H}_v^{l+1}, \bigoplus \boldsymbol{H}_e^{l+1}, \boldsymbol{h}_g^l\right).$$

By repeating this step L times, information flowing along edges can reach nodes that are L hops away. An illustration of this process can be seen in Figure 2.4.

2.3.3. Graph Multi-Agent RL

The aforementioned concepts have been studied in several multi-agent learning frameworks. Jiang et al. [85] model the multi-agent task as a graph and use convolutions with a multi-head attention kernel to extract relational representations between neighbors. Chen et al. [30] use graph neural networks and attention for communication to overcome environment non-stationarity and improve over broadcast-based multi-agent algorithms such as MADDPG [101]. Freymuth et al. [48] use the concept of message-passing networks for adaptive mesh refinement where each element of the mesh can be seen as an agent that decides whether to refine the mesh or not.

2.4. Exploration Strategies in Reinforcement Learning

In the previous sections, we have seen how collected data can be represented and used to update an agent's behavior, but not how to obtain new data for learning, i.e., how to *explore*. Ideally, we want an agent to find the best policy as quickly as possible while avoiding to become trapped in local optima,

or, especially in the case of robotics, entering dangerous regions of the state space or choosing harmful actions. Exploration is still an open topic of research and several very different strategies exist. In the following, we give a general overview over random exploration in action-space and parameter-space. Afterwards, we provide an overview over trust-region methods and evolution strategies, which are the basis of the reinforcement learning algorithms used in this thesis.

Action-Space Exploration Action-space exploration, as the name suggests, performs exploration by perturbing the action taken by the agent. This perturbation can either be inherent if the action is sampled from a stochastic policy or added from an external noise process. For discrete action-space problems, classical strategies are ϵ-greedy or Boltzmann policies derived from the Q-values for a given state [148]. In the case of continuous actions, a stochastic policy is modelled most commonly as a multi-variate Gaussian $\pi(a \mid s) = \mathcal{N}(a \mid \mu(s), \Sigma)$ with state dependent mean μ and covariance Σ. Although exploration is directly driven by the covariance matrix, for problems with well shaped rewards, a diagonal non-contextual covariance is often sufficient [11]. Different techniques exist to steer the covariance during the optimization process. An algorithmic approach that directly encourages the agent to explore without altering the problem is maximum-entropy reinforcement learning [166, 59]. Here, the objective function is augmented with the policy's entropy trading off optimizing the task while acting as randomly as possible. Alternatively, the agent can also be rewarded for entering novel states through intrinsic motivation objectives [15].

Parameter-Space Exploration While action-space exploration is the de-facto standard in deep RL today, there are also certain downsides to it. For example, many real world tasks only provide sparse rewards or work on long time horizons making it hard for unstructured exploration to find optimal solutions. Furthermore, it introduces a high level of noise as even for a fixed state, hardly ever the same action will be taken. Lastly, on real robots it leads to shaky behavior due to jittery trajectories which can damage or break robots [125]. Instead, another way to enforce exploration of the state space is to perturb the parameters of the policy themselves. Evolution strategies, which we will describe in more detail in Section 2.4.2, are a classic example for algorithms that perform exploration in parameter space. The basic idea consists of sampling a generation of candidate parameters, evaluating them, and improve them towards more promising regions. Based on this idea, Plappert et al. [122], for example, show how to combine parameter noise with deep RL algorithms such as DQN [105], DDPG [95], and TRPO [137]. Their algorithms improve exploration especially in sparse reward scenarios. While the ES concept has also been implemented for large scale optimization of neural network policies [133], we take a movement primitive based approach that is especially useful for robotics.

2.4.1. Trust-Region Reinforcement Learning

Trust-region optimization methods have a long history in machine learning and optimization as these methods are especially useful for optimization of non-linear and potentially noisy objective functions. They have been around for as early as the 1960s [53]. Trust-region reinforcement learning algorithms are based on these methods and are used to ensure thorough and stable exploration in both, action-space and parameter-space exploration setups. Within this approach, an objective function is iteratively maximized within a well modelled region such that the optimization parameters do not deviate greatly between subsequent iterations. The model can be obtained, for example, by local first or second order information such as the gradient or Hessian.

In this thesis, we will use Trust Region Policy Optimization [137] in the framework provided in Chapters 3 and 4. In Chapter 5 we will analyse and improve Model-Based Relative Entropy Stochastic Search [3]. In this section, we will provide a brief introduction to constrained optimization and outline the optimization problems that are solved in these algorithms.

2.4.1.1. Constrained Optimization

In its general form, a constrained optimization problem is given by

$$\begin{aligned} \underset{\boldsymbol{\theta}}{\text{maximize}} \quad & f(\boldsymbol{\theta}) \\ \text{subject to} \quad & g_i(\boldsymbol{\theta}) \leq 0, \\ & h_j(\boldsymbol{\theta}) = 0 \end{aligned}$$

where we want to find parameters $\boldsymbol{\theta}$ that maximize an objective function f, inequality constraints are denoted as g_i, $i = (1, \ldots, m)$, equality constraints are denoted as h_j, $j = (1, \ldots, l)$. Under differentiability and convexity assumptions, the constrained optimization problem with inequality constraints can be solved using the Karush–Kuhn–Tucker (KKT) approach, a generalization of the method of Lagrange multipliers. First, we construct the Lagrangian function

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\eta}, \boldsymbol{\omega}) = f(\boldsymbol{\theta}) - \sum_{i=1}^{m} \eta_i g_i(\boldsymbol{\theta}) - \sum_{j=1}^{l} \omega_j h_j(\boldsymbol{\theta})$$

with Lagrange multipliers $\boldsymbol{\eta}$ and $\boldsymbol{\omega}$. The solution to the optimization problem is a saddle point of the Lagrangian function. If we can obtain $\boldsymbol{\theta}^*$ analytically, we first need to find

$$\boldsymbol{\theta}^* = l(\boldsymbol{\eta}, \boldsymbol{\omega}) = \arg\max_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\eta}, \boldsymbol{\omega}).$$

as a function of the Lagrangian multipliers $\boldsymbol{\eta}$ and $\boldsymbol{\omega}$. Subsequently, we plug the solution back into the Lagrangian to obtain the dual function

$$h(\boldsymbol{\eta}, \boldsymbol{\omega}) = \mathcal{L}(l(\boldsymbol{\eta}, \boldsymbol{\omega}), \boldsymbol{\eta}, \boldsymbol{\omega})$$

and solve the dual optimization problem

$$\begin{aligned} \underset{\boldsymbol{\eta}, \boldsymbol{\omega}}{\text{minimize}} \quad & h(\boldsymbol{\theta}) \\ \text{subject to} \quad & \eta_i \geq 0 \end{aligned}$$

to find the solution $\boldsymbol{\eta}^*, \boldsymbol{\omega}^*$. Since the dual function is convex even when the original problem is not concave, this can be done using a standard non-linear optimizer such as L-BFGS-B [165]. The optimal solution of the original problem obeying the constraints is finally given by

$$\boldsymbol{\theta}^* = l(\boldsymbol{\eta}^*, \boldsymbol{\omega}^*).$$

2.4.1.2. KL-Constrained Trust-Region Reinforcement Learning

The optimization problem can be formulated using a constrained optimization problem where f usually relates to the expected cumulative reward J and the constraints ensure that the updated policy does not

deviate too much from the current policy. A common choice is the Kullback-Leibler (KL) divergence $\mathrm{KL}(p \parallel q)$ [90] (also known as relative entropy), a statistical distance of how a probability distribution p is different from a probability distribution q. For continuous distributions, it is defined as

$$\mathrm{KL}(p \parallel q) = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} \mathrm{dx}.$$

In case the policy is modelled as a multivariate Gaussian, the KL divergence can be computed in closed form. In this thesis we will use two trust-region RL algorithms, Trust-Region Policy Optimization (TRPO) [137] and Model-based Relative Entropy Stochastic Search (MORE) [2]. We will modify TRPO to be used in a multi-agent setting in Chapters 3 and 4 and closely analyze and improve MORE in Chapter 5. In the following sections, we will briefly introduce the optimization problems that the methods introduce and show how they are solved.

Trust Region Policy Optimization TRPO is a step-based deep RL algorithm for problems with continuous state and actions spaces where the policy is parameterized by a neural network with parameters $\boldsymbol{\theta}$ that outputs a state dependent mean and learned diagonal covariance matrix of a multivariate Gaussian. The algorithm is based on a monotonic improvement guarantee for general stochastic policies. The optimization problem is given by

$$\begin{aligned} \underset{\pi}{\text{maximize}} \quad & L_t(\pi) \\ \text{subject to} \quad & \overline{\mathrm{KL}}(\pi_t \parallel \pi) \leq \delta \end{aligned}$$

where the objective L_t is given as

$$L_t(\pi) = \mathbb{E}_{s\sim\rho_t,\, a\sim\pi_t} \left[\frac{\pi(a \mid s)}{\pi_t(a \mid s)} A^{\pi_t}(s,a) \right].$$

$\overline{\mathrm{KL}}(\pi_t \parallel \pi) = \mathbb{E}_{s\sim\rho_t}[\mathrm{KL}(\pi_t \parallel \pi)]$ denotes the average KL-divergence between the current policy π_t and new policy π under the state distribution ρ_t induced by π_t. It is bounded by δ to ensure the algorithm is stable. However, the above optimization problem cannot be solved directly. Instead, a linear approximation of the objective and a quadratic approximation of the constraint are made. On this simplified problem, the method of Lagrangian multipliers can be applied. The optimization problem is solved using the conjugate gradient method followed by a line search in order to obey the constraint of the original problem.

Model-based Relative Entropy Stochastic Search MORE falls into the category of derivative-free optimization methods and can be applied to non-contextual episode-based RL problems. The policy is parameterized as a multi-variate Gaussian $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{w} \mid \boldsymbol{\mu}, \Sigma)$. It can be used to choose a parameter vector that describes the trajectory for a whole episode, for example using movement primitives. The optimization problem is formally given by

$$\begin{aligned} \underset{\pi}{\text{maximize}} \quad & \int_{\boldsymbol{w}} \pi(\boldsymbol{w}) f(\boldsymbol{w}) \mathrm{dx} \\ \text{subject to} \quad & \mathrm{KL}(\pi \parallel \pi_t) \leq \epsilon, \\ & H(\pi) \geq \beta, \\ & \int_{\boldsymbol{w}} \pi(\boldsymbol{w}) \mathrm{dw} = 1 \end{aligned}$$

where ϵ and β are hyper-parameters controlling the exploration-exploitation trade-off,

$$H(p) = \int_{\boldsymbol{x}} p(\boldsymbol{x}) \log \pi(\boldsymbol{x}) \mathrm{dx}$$

denotes the Shannon entropy of a distribution p. After substituting the objective function with a quadratic approximation, the optimization problem is solved exactly with analytic updates for the mean and covariance. Full details can be found in Chapter 5.

2.4.2. Evolution Strategies

The MORE algorithm presented in the previous section is closely related to algorithms from the class of evolution strategies (ES) [69, 24]. They provide a framework for black-box optimization of continuous valued objective functions in cases where an objective function is too complex to be modelled and no higher order information is available. In reinforcement learning, they can be applied in episode-based problems performing parameter-space exploration.

The basic paradigm consists of two steps: Sampling new candidate solutions from a search distribution and updating the search distribution parameters based on the quality of the function values evaluated at the candidates. This procedure is executed iteratively until a satisfactory solution is found or a pre-defined budget of function evaluations is exceeded. Prominent algorithms of this category are the cross-entropy method [128], Natural Evolution Strategies [154], and the covariance matrix adaptation evolution strategies [4] which we will discuss in more detail in the next section.

2.4.2.1. Covariance Matrix Adaptation Evolution Strategies

The covariance matrix adaptation evolution strategies (CMA-ES) is an algorithm for finding the minimum of an objective function through heuristically defined updates. A multi-variate normal distribution $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{m}, \sigma^2 \boldsymbol{C})$ is maintained and updated based on a ranking of function values. The parameter $\sigma > 0$ is referred to as the step-size. A key part of the algorithm are so-called evolution paths which accumulate previous update steps. They are functions of the mean displacement $\boldsymbol{m}_{k+1} - \boldsymbol{m}_k$ of subsequent iterations k and $k+1$. In case updates are correlated, the corresponding evolution path is long while it is short if previous updates were uncorrelated.

Mean Update The mean is updated according to

$$\boldsymbol{m}_{k+1} = \sum_{i=1}^{\mu} w_i \boldsymbol{x}_{i:\lambda}$$

where $\{\boldsymbol{x}_{i:\lambda} \mid i = 1, \ldots, \lambda\} = \{\boldsymbol{x}_i \mid i = 1, \ldots, \lambda\}$ are the function value sorted candidate solutions such that $f(\boldsymbol{x}_{1:\lambda}) \leq \cdots \leq f(\boldsymbol{x}_{\mu:\lambda}) \leq f(\boldsymbol{x}_{\lambda:\lambda})$. Typically, $\mu \leq \lambda/2$. The weights are chosen such that $\sum_i w_i = 1$

Step-Size Update The step-size is updated using a technique called cumulative step size adaptation. Based on an evolution path $\boldsymbol{p}_\sigma$, the update is given as

$$\sigma_{k+1} = \sigma_k \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|\boldsymbol{p}_\sigma\|}{\mathbb{E}[\|\mathcal{N}(0, \boldsymbol{I})\|]} - 1\right)\right)$$

where c_σ, d_σ are hyper-parameters and $\mathbb{E}[\|\mathcal{N}(0, \boldsymbol{I})\|]$ is the expectation of the Euclidean norm of a $\mathcal{N}(0, \boldsymbol{I})$ distributed random vector.

Covariance Matrix Update Finally, the covariance matrix update makes use of an evolution path $\boldsymbol{p}_C$. The update is given by

$$\boldsymbol{C}_{k+1} = (1 - c_1 - c_\mu + c_s)\boldsymbol{C}_k + c_1 \boldsymbol{p}_C \boldsymbol{p}_C^\mathsf{T} + c_\mu \sum_{i=1}^{\mu} w_i \frac{\boldsymbol{x}_{i:\lambda} - \boldsymbol{m}_k}{\sigma_k} \left(\frac{\boldsymbol{x}_{i:\lambda} - \boldsymbol{m}_k}{\sigma_k} \right)^\mathsf{T}$$

with hyper-parameters c_1, c_μ, c_s.

For full details, we refer to, e.g., Hansen [68].

3. Local Communication Protocols for Learning Complex Swarm Behaviors with Deep Reinforcement Learning

This chapter has been published in the 11th International Conference on Swarm Intelligence, ANTS 2018 [80].

Nature provides many examples where the performance of a collective of limited beings exceeds the capabilities of one individual. Ants transport prey of the size no single ant could carry, termites build nests of up to nine meters in height, and bees are able to regulate the temperature of a hive. Common to all these phenomena is the fact that each individual has only basic and local sensing of its environment and limited communication capabilities to its neighbors.

Inspired by these biological processes, swarm robotics [21, 8, 31] tries to emulate such complex behavior with a collective of rather simple entities. Typically, these robots have limited movement and communication capabilities and can sense only a local neighborhood of their environment, such as distances and bearings to neighbored agents. Moreover, these agents have limited memory systems, such that the agents can only access a short horizon of their perception. As a consequence, the design of control policies that are capable of solving complex cooperative tasks becomes a non-trivial problem.

In this paper, we want to learn swarm behavior using deep reinforcement learning [137, 105, 136, 150, 57] based on the locally sensed information of the agents such that the desired behavior can be defined by a reward function instead of hand-tuning controllers of the agents. Swarm systems constitute a challenging problem for reinforcement learning as the algorithm needs to learn decentralized control policies that can cope with limited local sensing and communication abilities of the agents.

Most collective tasks require some form of active cooperation between the agents. For efficient cooperation, the agents need to implement basic communication protocols such that they can transmit their local sensory information to neighbored agents. Using prior knowledge about the given task, simple communication protocols can be defined much more easily than directly defining the behavior. In this paper, we propose and evaluate several communication protocols that can be exploited by deep reinforcement learning to find decentralized control policies in a multi robot swarm environment.

Our communication protocols are based on local histograms that encode the neighborhood relation of an agent to other agents and can also transmit task-specific information such as the shortest distance and direction to a desired target. The histograms can deal with the varying number of neighbors that can be sensed by a single agent depending on its current neighborhood configuration. These protocols are used to generate high dimensional observations for the individual agents that is in turn exploited by deep reinforcement learning to efficiently learn complex swarm behavior. In particular, we choose an adaptation of Trust Region Policy Optimization [137] to learn decentralized policies.

In summary, our method addresses the emerging challenges of decentralized swarm control in the following way:

1. **Homogeneity:** explicit sharing of policy parameters between the agents

2. **Partial observability:** efficient processing of action-observation histories through windowing and parameter sharing
3. **Communication:** usage of histogram-based communication protocols over simple features

To demonstrate our approach, we formulate two cooperative learning tasks in a simulated swarm environment. The environment is inspired by the Colias robot [14], a modular platform with two wheel motor-driven movement and various sensing systems.

Paper Outline In Section 3.1, we review the concepts of Trust Region Policy Optimization and describe our problem domain. In Section 3.2, we show in detail how we tackle the challenges of modeling observations and the policy in the partially observable swarm context, and how to adapt Trust Region Policy Optimization to our setup. In Section 3.3, we present the model and parameters of our agents and introduce two tasks on which we evaluate our proposed observation models and policies.

3.1. Background

In this section, we provide a short summary of Trust Region Policy Optimization and formalize our learning problem domain.

3.1.1. Trust Region Policy Optimization

Trust Region Policy Optimization (TRPO) is an algorithm to optimize control policies in single-agent reinforcement learning problems [137]. These problems are formulated as Markov decision processes (MDP) which are compactly written as a tuple $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$. In an MDP, an agent chooses an action $a \in \mathcal{A}$ via some policy $\pi(a \mid s)$, based on its current state $s \in \mathcal{S}$, and progresses to state $s' \in \mathcal{S}$ according to a transition function $P(s' \mid s, a)$. After each step, the agent is assigned a reward $r = R(s, a)$, provided by a reward function R which judges the quality of its decision. The goal of the agent is to find a policy which maximizes the expected cumulative reward $\mathbb{E}[\sum_{k=t}^{\infty} \gamma^{k-t} R(s_k, a_k)]$, discounted by factor γ, achieved over a certain period of time.

In TRPO, the policy is parametrized by a parameter vector θ containing weights and biases of a neural network. In the following, we denote this parameterized policy as π_θ. The reinforcement learning objective is expressed as finding a new policy that maximizes the expected advantage function of the current policy, i.e., $J^{\text{TRPO}} = \mathbb{E}\left[\frac{\pi_\theta}{\pi_{\theta_{\text{old}}}} \hat{A}(s, a)\right]$, where $\hat{A}$ is an estimate of the advantage function of the current policy π_{old} which is defined as $\hat{A}(s, a) = Q^{\pi_{\text{old}}}(s, a) - V^{\pi}_{\text{old}}(s)$. Herein, state-action value function $Q^{\pi_{\text{old}}}(s, a)$ is typically estimated by a single trajectory rollout while for the value function $V^{\pi_{\text{old}}}(s)$ rather simple baselines are used that are fitted to the monte-carlo returns. The objective is to be maximized subject to a fixed constraint on the Kullback-Leibler (KL) divergence of the policy before and after the parameter update, which ensures the updates to the new policy's parameters θ are bounded, in order to avoid divergence of the learning process. The overall optimization problem is summarized as

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & \mathbb{E}\left[\frac{\pi_\theta}{\pi_{\theta_{\text{old}}}} \hat{A}(s, a)\right] \\ \text{subject to} \quad & \mathbb{E}[D_{\text{KL}}(\pi_{\theta_{\text{old}}} || \pi_\theta)] \leq \delta. \end{aligned}$$

The problem is approximately solved using the conjugate gradient optimizer after linearizing the objective and quadratizing the constraint.

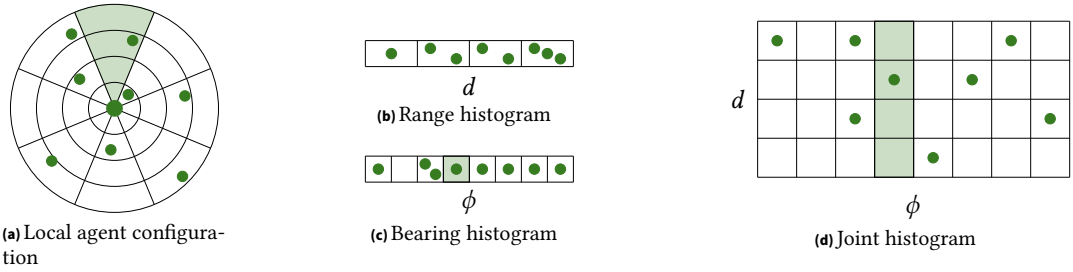
3.1.2. Problem Domain

Building upon the theory of single-agent reinforcement learning, we can now formulate the problem domain for our swarm environments. Because of their limited sensory input, each agent can only obtain a local observation o from the vicinity of its environment. We formulate the swarm system as a swarm MDP (see Šošić et al. [143] for a similar definition) which can be seen as a special case of a decentralized partially observed Markov decision process (Dec-POMDP) [111]. An agent in the swarm MDP is defined as a tuple $\mathbb{A} = \langle \mathcal{S}, \mathcal{O}, \mathcal{A}, O \rangle$, where, $\mathcal{S}$ is a set of local states, $\mathcal{O}$ is the space of local observations, and $\mathcal{A}$ is a set of local actions for each agent. The observation model $O(o|s, i)$ defines the observation probabilities for agent i given the global state s. Note that the system is invariant to the order of the agents, i.e., given the same local state of two agents, the observation probabilities will be the same. The swarm MDP is then defined as $\langle N, \mathcal{E}, \mathbb{A}, P, R \rangle$, where N is the number of agents, $\mathcal{E}$ is the global environment state consisting of all local states $\mathcal{S}^N$ of the agents and possibly of additional states of the environment, and $P : \mathcal{S}^N \times \mathcal{S}^N \times \mathcal{A}^N \to [0, \infty)$ is the transition density function. Each agent maintains a truncated history $h_t^i = (a_{t-\eta}^i, o_{t-\eta+1}^i, \ldots, a_{t-1}^i, o_t^i)$ of the current and past observations $o^i \in \mathcal{O}$ and actions $a^i \in \mathcal{A}$ of length η. All swarm agents are assumed to be identical and therefore use the same distributed policy π (now defined as $\pi(a \mid h)$) which yields a sample for the action of each agent given its current history of actions and observations. The reward function R of the swarm MDP depends on the *global state* and, optionally, all actions of the swarm agents, i.e., $R : \mathcal{S}^N \times \mathcal{A}^N \to \mathbb{R}$. Instead of considering only one single agent, we consider multiple agents of the same type, which interact in the same environment. The global system state is in this case comprised of the local states of all agents and additional attributes of the environment. The global task of the agents is encoded in the reward function $R(\boldsymbol{s}, \boldsymbol{a})$, where we from now on write $\boldsymbol{a}$ to denote the joint action vector of the whole swarm.

3.1.3. Related Work

A common approach to program swarm robotic systems is by extracting rules from the observed behavior of their natural counterparts. Kube and Bonabeau [88], for example, investigate the cooperative prey retrieval of ants to infer rules on how a swarm of robots can fulfill the task of cooperative box-pushing. Similar work can be found e.g. in Martinoli, Easton, and Agassounon [103], Hoff et al. [76], Nouyan et al. [110]. However, extracting these rules can be tedious and the complexity of the tasks that we can solve via explicit programming is limited. More examples of rule based behavior are found in Chen, Gauci, and Groß [31] where a group of swarming robots transports an object to a goal. Further comparable work can be found in Correll and Martinoli [37] for aggregation, Moeslinger, Schmickl, and Crailsheim [106] for flocking, or Goldberg and Mataric [52] for foraging.

In deep RL, currently, there are only few approaches tackling the multi-agent problem. One of these approaches can be found in Lowe et al. [101], where the authors use a variation of the deep deterministic policy gradient algorithm [95] to learn a centralized Q-function for each policy, which, as a downside, leads to a linear increase in dimensionality of the joint observation and action spaces therefore scales poorly. Another algorithm, tackling the credit assignment problem, can be found in Foerster et al. [46]. Here, a baseline of other agents' behavior is subtracted from a centralized critic to reason about the quality of a single agent's behavior. However, this approach is only possible in scenarios with discrete action spaces since it requires marginalization over the agents' action space. Finally, a different line of work concerning the learning of communication models between agents can be found in Foerster et al. [47].



(a) Local agent configuration (b) Range histogram (c) Bearing histogram (d) Joint histogram

Figure 3.1.: This Figure shows an illustration of the histogram-based observation model. Figure 3.1a shows an agent in the center of a circle whose neighborhood relations are to be captured by the histogram representation. The shaded green area is highlighted as a reference for Figures 3.1c and 3.1d. Figure 3.1b hereby shows the one dimensional histogram of agents over the neighborhood range d into four bins, whereas Figure 3.1c shows the histogram over the bearing angles ϕ into eight bins. Figure 3.1d finally shows the two dimensional joint histogram over range and bearing.

3.2. Multi-Agent Learning with Local Communication Protocols

In this section, we introduce different communication protocols based on neighborhood histograms that can be used in combination to solve complex swarm behaviors. Our algorithm relies on deep neural network policies of special architecture that can exploit the structure of the high-dimensional observation histories. We present this network model and subsequently discuss small adaptations we had to make to the TRPO algorithm in order to apply it to this cooperative multi-agent setting.

3.2.1. Communication Protocols

Our communication protocols are based on histograms that can either encode neighborhood relations or distance relations to different points of interest.

Neighborhood Histograms

The individual agents can observe distance and bearing to neighbored agents if they communicate with this agent. We assume that the agents are constantly sending a signal, such that neighbored agents can localize the sources. The arising neighborhood configuration is an important source of information and can be used as observations of the individual agents. One of the arising difficulties in this case is to handle changing number of neighbors which would result in a variable length of the observation vector. Most policy representations, such as neural networks, expect a fixed input dimension.

One possible solution to this problem is to allocate a fixed number of neighbor relations for each agent. If an agent experiences fewer neighborhood relations, standard values could be used such as a very high distance and 0 bearing. However, such an approach comes with several drawbacks. First of all, the size of the resulting representation scales linearly with the number of agents in the system and so does the number of parameters to be learned. Second, the execution of the learned policy will be limited to scenarios with the exact same number of agents as present during training. Third, a fixed allocation of the neighbor relation inevitably destroys the homogeneity of the swarm, since the agents are no longer treated interchangeably. In particular, using a fixed allocation rule requires that the agents must be able to discriminate between their neighbors, which might not even be possible in the first place.

To solve these problems, we propose to use *histograms over observed neighborhood relations*, e.g., distances and bearing angles. Such a representation inherently respects the agent homogeneity and

naturally comes with a fixed dimensionality. Hence, it is the canonical choice for the swarm setting. For our experiments, we consider two different types of representations: 1) concatenated one-dimensional histograms of distance and bearing and 2) multidimensional histograms. Both types are illustrated in Figure 3.1. The one-dimensional representation has the advantage of scalability, as it grows linearly with the number of features. The downside is that potential dependencies between the features are completely ignored.

Shortest Path Partitions

In many applications, it is important to transmit the location of a point of interest to neighbored agents that can currently not observe this point due to their limited sensing ability.

We assume that an agent can observe bearing and distance to a point of interest if it is within its communication radius. The agent then transmits the observed distance to other agents. Agents that can not see the point of interest might in this case observe a message from another agent containing the distance to the point of interest. The distance of the sending agent is added to the received distance to obtain the distance to the point of interest if we would use the sending agent as a via point. Each agent might now compute several of such distances and transmits the minimum distance it has computed to indicate the length of the shortest path it has seen.

The location of neighbored agents including their distance of the shortest path information is important knowledge for the policy, e.g. for navigating to the point of interest. Hence, we adapt the histogram representation. Each partition now contains the minimum received shortest path distance of an agent that is located in this position.

3.2.2. Weight Sharing for Policy Networks

The policy maps sequences of past actions and observations to a new action. We use histories of a fixed length as input to our policy and a feed-forward deep neural network as architecture. To cope with such high input dimensionality, we propose a weight sharing approach. Each action-observation pair in an agent's history is first processed independently with a network using the same weights. After this initial reduction in dimensionality, the hidden states are concatenated in a subsequent layer and finally mapped to an output. The homogeneity of agents is achieved by using the same set of parameters for all policies. A diagram of the architecture is shown in Figure 3.2.

3.2.3. Adaptations to TRPO

In order to apply TRPO to our multi-agent setup, some small changes to the original algorithm have to be made, similar to the formulation of Gupta, Egorov, and Kochenderfer [58]. First, since we assume homogeneous agents, we can have one set of parameters of the policy shared by all agents. Since the agents cannot rely on the global state, the advantage function is redefined as $A(h, a)$. In order to estimate this function, each agent is assigned the same global reward r in each time step and all transitions are treated as if they were executed by a single agent.

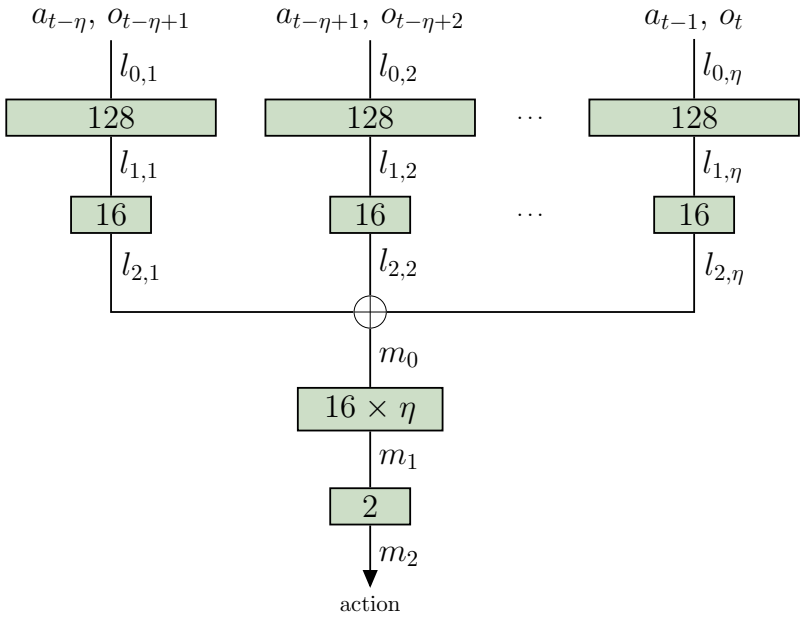


Figure 3.2.: This diagram shows a model of our proposed policy with three hidden layers. The numbers inside the boxes denote the dimensionalities of the hidden layers. The plus sign denotes concatenation of vectors.

3.3. Experimental Setup

In this section, we briefly discuss the used model and state representation of a single agent. Subsequently, we describe our two experimental setups and the policy architecture used for the experiments.

3.3.1. Agent Model

The local state of a single agent is modeled by its 2D position and orientation, i.e., $s^i = [x^i, y^i, \phi^i] \in \mathcal{S} = \{[x, y, \phi] \in \mathbb{R}^3 : 0 \leq x \leq x_{\text{max}},\ 0 \leq y \leq y_{\text{max}},\ 0 \leq \phi \leq 2\pi\}$. The robot can only control the speed of its wheels. Therefore, we apply a force to the left and right side of the agent, similarly to the wheels of the real robot. Our model of a single agent is inspired by the Colias robot (a detailed description of the robot specifications can be found in Arvin et al. [14]), but the underlying principles can be straightforwardly applied to other swarm settings with limited observations. Generally, our observation model is comprised of the sensor readings of the short and long range IR sensors (later denoted as 'sensor' in the evaluations). Furthermore, we augment this observation representation with the communication protocols developed in the following section. Our simulation is using a 2D physics engine (Box2D), allowing for correct physical interaction of the bodies of the agents.

3.3.2. Tasks

The focus of our experiments is on tasks where agents need to collaborate to achieve a common goal. For this purpose, we designed the following two scenarios:

Task 1: Building a Graph

In the first task, the goal of the agents is to find and maintain a certain distance to each other. This kind of behavior is required, for example, in surveillance tasks, where a group of autonomous agents needs to maximize the coverage of a target area while maintaining their communication links. We formulate the task as a graph problem, where the agents (i.e. the nodes) try to maximize the number of active edges in the graph. Herein, an edge is considered active whenever the distance between the corresponding agent lies in certain range. The setting is visualized in Figure 3.3a. In our experiment, we provide a positive reward for each edge in a range between 10 cm and 16 cm, and further give negative feedback for distances smaller than 7 cm. Accordingly, the reward function is

$$R(\boldsymbol{s}, \boldsymbol{a}) = \sum_{i=1}^{M} \sum_{m>i}^{M} \mathbf{1}_{[0.1\,\mathrm{m},\ 0.16\,\mathrm{m}]}(d_m^i) - 5 \sum_{i=1}^{M} \sum_{m>i}^{M} \mathbf{1}_{[0\,\mathrm{m},\ 0.07\,\mathrm{m}]}(d_m^i), \tag{3.1}$$

where $d_m^i = \sqrt{(x_i - x_m)^2 + (y_i - y_m)^2}$ denotes the Euclidean distance between the centers of agent i and agent m and

$$\mathbf{1}_{[a,b]}(x) = \begin{cases} 1 & \text{if } x \in [a, b], \\ 0 & \text{else} \end{cases}$$

is an indicator function. Note that we omit the dependence of d_m^i on the system state s to keep the notion simple.

Task 2: Establishing a Communication Link

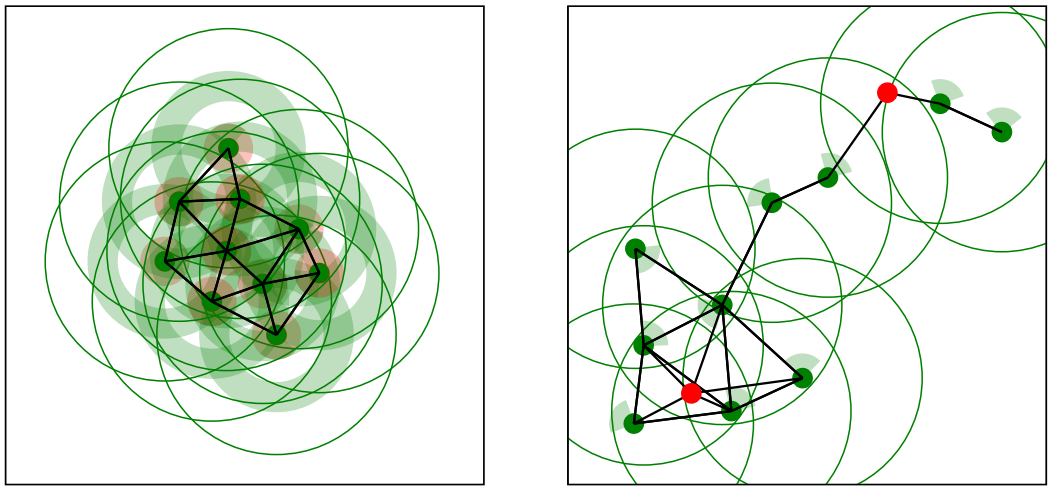
The second task adds another layer of difficulty. While maintaining a network, the agents have to locate and connect two randomly placed points in the state space. A link is only established successfully if there are communicating agents connecting the two points. Figure 3.3b shows an example with an active link spanned by three agents between the two points. The task resembles the problem of establishing a connection between two nodes in a wireless ad-hoc network [19, 157]. In our experiments, the distance of the two points is chosen to be larger than 75 cm, requiring at least three agents to bridge the gap in between. The reward is determined by the length of the shortest distance between the two points d_{opt} (i.e. a straight line) and the length of the shortest active link d_{sp} spanned by the agents,

$$R(\boldsymbol{s}, \boldsymbol{a}) = \begin{cases} \frac{d_{\mathrm{opt}}}{d_{\mathrm{sp}}} & \text{if link is established} \\ 0 & \text{otherwise.} \end{cases}$$

In this task, we use the shortest path partitions as communication protocol. Each agent communicates the shortest path it knows to both points of interests, resulting in two 2-D partitions that are used as observation input for a single time step.

3.3.3. Policy Architecture

We decided for a policy model with three hidden layers. The first two layers process the observation-action pairs (a_{k-1}, o_k) of each timestep in a history individually and map it into hidden layers of size 128 and 16. The output of the second layer is then concatenated to form the input of the third hidden layer which eventually maps to the two actions for the left and right motor.



(a) Edge task **(b)** Link task

Figure 3.3.: Illustration of the two cooperative tasks used in this paper. The green dots represent the agents, where the green ring segments located next to the agents indicate the short range IR front sensors. The outer green circles illustrate the maximum range in which distances / bearings to other agents can be observed, depending on the used observation model. **(a) Edge task:** The red rings show the penalty zones where the agents are punished, the outer green rings indicate the zones where legal edges are formed. **(b) Link task:** The red dots correspond to the two points that need to be connected by the agents.

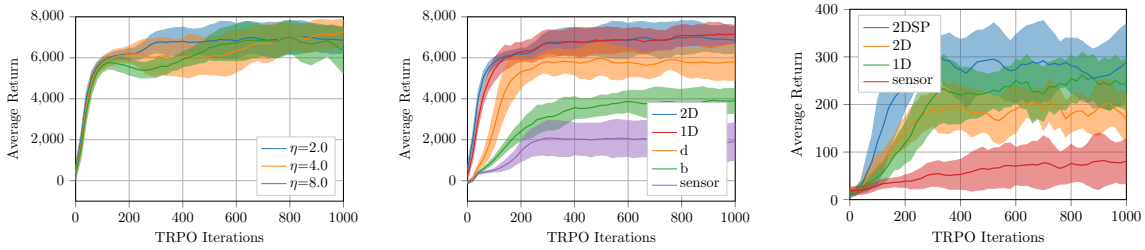
3.4. Results

We evaluate each task in a standardized environment of size $1\,\mathrm{m} \times 1\,\mathrm{m}$ where we initialize ten agents randomly in the scene. Of special interest is how the amount of information provided to the agents affects the overall system performance. Herein, we have to keep in mind the general information-complexity trade-off, i.e., high-dimensional local observations generally provide more information about the global system state but, at the same time, result in a more complex learning task. Recall that the information content is mostly influenced by two factors: 1) the length of the history, and 2) the composition of the observation.

3.4.1. Edge Task

First, we evaluate how the history length η affects the system performance. Figure 3.4a shows an evaluation for $\eta = \{2, 4, 8\}$ and a weight sharing policy using a two-dimensional histogram over distances and bearings. Interestingly, we observe that longer observation histories do not show an increase in the performance. Either the increase in information could not counter the effect of increased learning complexity, or a history length of $\eta = 2$ is already sufficient to solve the task. We use these findings and set the history length to $\eta = 2$ for the remainder of the experiments.

Next, we analyze the impact of the observation model. Figure 3.4b shows the results of the learning process for different observation modalities. The first observation is that, irrespective of the used mode, the agents are able to establish a certain number of edges. Naturally, a complete information of distances and bearing yields the best performance. However, the independent histogram representation yields comparable results to the two dimensional histogram. Again, this is due to the aforementioned complexity trade-off where a higher amount of information makes the learning process more difficult.



(a) Comparison of different history lengths η. (2D histogram)

(b) Edge task: Comparison of different observation models. ($\eta = 2$)

(c) Link task: Comparison of different observation models. ($\eta = 2$)

Figure 3.4.: Learning curves for (a), (b) the edge task and (c) the link task. The curves show the mean values of the average undiscounted return of an episode (i.e. the sum of rewards of one episode, averaged over the number of episodes for one learning iteration) over the learning process plus / minus one standard deviation, computed from eight learning trials. Intuitively, the return in the edge task corresponds to the number of edges formed during an episode of length 500 steps. In the link task, it is a measure for the quality of the link. **Legend:** 2DSP: two dimensional histogram over shortest paths, 2D: two-dimensional histogram over distances and bearings, 1D: two independent histograms over distances and bearing, d: distance only histogram, b: bearing only histogram, sensor: no histogram.

3.4.2. Link Task

We evaluate the link task with raw sensor measurements, count based histograms over distance and bearing, and the more advanced shortest path histograms over distance and bearing. Based on the findings of the edge task we keep the history length at $\eta = 2$. Figure 3.4c shows the results of the learning process where each observation model was again tested and averaged over 8 trials. Since at least three agents are necessary to establish a link between the two points, the models without shortest path information struggle to reliably establish the connection. Their only chance is to spread as wide as possible and, thus, cover the area between both points. Again, it is interesting to see that independent histograms over counts seem to be favorable over the 2D histogram. However, both versions are surpassed by the 2D histogram over shortest paths which yields information about the current state of the whole network of agents, currently connected to each of the points.

3.5. Conclusions and Future Work

In this paper, we demonstrated that histograms over simple local features can be an effective way for processing information in robot swarms. The central aspect of this new model is its ability to handle arbitrary system sizes without discriminating between agents, which makes it perfectly suitable to the swarm setting where all agents are identical and the number of agents in the neighborhood varies with time. We use these protocols and an adaptation of TRPO for the swarm setup to learn cooperative decentralized control policies for a number of challenging cooperative task. The evaluation of our approach showed that this histogram-based model leads the agents to reliably fulfill the tasks.

Interesting future directions include, for example, the learning of an explicit communication protocol. Furthermore, we expect that assigning credit to agents taking useful actions should speedup our learning algorithm.

Acknowledgments.

The research leading to these results has received funding from EPSRC under grant agreement EP/R02572X/1 (National Center for Nuclear Robotics). Calculations for this research were conducted on the Lichtenberg high performance computer of the TU Darmstadt

4. Deep Reinforcement Learning for Swarm Systems

This chapter has been published in the Journal of Machine Learning Research [79].

In swarm systems, many identical agents interact with each other to achieve a common goal. Typically, each agent in a swarm has limited capabilities in terms of sensing and manipulation so that the considered tasks need to be solved collectively by multiple agents.

A promising application where intelligent swarm systems take a prominent role is swarm robotics [21]. Robot swarms are formed by a large number of cheap and easy to manufacture robots that can be useful in a variety of situations and tasks, such as search and rescue missions or exploration scenarios. A swarm of robots is inherently redundant towards loss of individual robots since usually none of the robots plays a specific role in the execution of the task. Because of this property, swarm-based missions are often favorable over single-robot missions (or, let alone, human missions) in hazardous environments. Behavior of natural swarms, such as foraging, formation control, collective manipulation, or the localization of a common 'food' source can be adapted to aid in these missions [21]. Another field of application is routing in wireless sensor networks [132] since each sensor in the network can be treated as an agent in a swarm.

A common method to obtain control strategies for swarm systems is to apply optimization-based approaches using a model of the agents or a graph abstraction of the swarm [97, 83]. Optimization-based approaches allow to compute optimal control policies for tasks that can be well modeled, such as rendezvous or consensus problems [96] and formation control [126], or to learn pursuit strategies to capture an evader [164]. Yet, these approaches typically use simplified models of the agents / the task and often rely on unrealistic assumptions, such as operating in a connected graph [39] or having full observability of the system state [164]. Rule-based approaches use heuristics inspired by natural swarm systems, such as ants or bees [60]. Yet, while the resulting heuristics are often simple and can lead to complex swarm behavior, the obtained rules are difficult to adapt, even if the underlying task changes only slightly.

Recently, deep reinforcement learning (RL) strategies have become popular to solve multi-agent coordination problems. In RL, tasks are specified indirectly through a cost function, which is typically easier than defining a model of the task directly or a finding a heuristic for the controller. Having defined a cost function, the RL algorithm aims to find a policy that minimizes the expected cost. Applying deep reinforcement learning within the swarm setting, however, is challenging due to the large number of agents that need to be considered. Compared to single-agent learning, where the agent is confronted only with observations about its own state, each agent in a swarm can make observations of several other agents populating the environment and thus needs to process an entire set of information that is potentially varying in size. Accordingly, two main challenges can be identified in the swarm setting:

1. High state and observation dimensionality, caused by large system sizes.
2. Changing size of the available information set, either due to addition or removal of agents, or because the number of observed neighbors changes over time.

Most current multi-agent deep reinforcement learning methods either concatenate the information received from different agents [101] or encode it in a multi-channel image, where the image channels contain different features based on a local view of an agent [147, 162]. However, both types of methods bare major drawbacks. Since neural network policies assume a fixed input dimensionality, a concatenation of observations is unsuitable in the case changing agent numbers. Furthermore, a concatenation disregards the inherent permutation invariance of identical agents in a swarm system and scales poorly to large system sizes. Top-down image based representations alleviate the issue of permutation invariance, however, the information obtained from neighboring agents is of mostly spatial nature. While additional information can be captured by adding more image channels, the dimensionality of the representation increases linearly with each feature. Furthermore, the discretization into pixels has limited accuracy due to quantization errors.

In this paper, we exploit the homogeneity of swarm systems and treat the state information perceived from neighboring agents as samples of a random variable. Based on this model, we then use mean feature embeddings (MFE) [142] to encode the current distribution of the agents. Each agent gets a local view of this distribution, where the information obtained from the neighbors is encoded in the mean embedding. Due to the sample-based view of the collected state information, we achieve a permutation invariant representation that is furthermore invariant to the number of agents in the swarm / the number of perceived neighbors.

Mean feature embeddings have so far been used mainly for kernel-based feature representations [55], but they can be also applied to histograms or radial basis function (RBF) networks. The resulting models are closely related to the "invariant model" formulated by Zaheer et al. [161]. However, compared to the summation approach described in their paper, the averaging of feature activations proposed in our approach yields the desired invariance with respect to the observed agent number mentioned above. To the best of our knowledge, we are the first to use mean embeddings inside a deep reinforcement learning framework for swarm systems where both the feature space of the mean embedding as well as the policy are learned end-to-end.

We test our state representation on various rendezvous and pursuit evasion problems using Trust Region Policy Optimization (TRPO) [137] as the underlying deep RL algorithm. In the rendezvous problem, the agents need to find a collective strategy that allows them to meet at some arbitrary location. In the pursuit evasion domain, a group of agents collectively tries to capture one or multiple evaders.

Policies are learned in a *centralized-learning / decentralized-execution fashion* fashion, meaning that during learning data from all agents is collected centrally and used to optimize the parameters as if there was only one agent. Nonetheless, each agent only has access to its own perception of the global system state to generate actions from the policy function. We compare our representation to several deep RL baselines as well as to optimization-based solutions, if available. Herein, we perform our experiments both in settings with global observability (i.e., all agents are neighbors) and in settings with local observability (i.e., agents are only locally connected). In the latter setting, we also evaluate different communication protocols [80] that allow the agents to transmit additional information about their local graph structure. For example, an agent might transmit the number of neighbors within its current neighborhood. Previously, such additional information could not be encoded efficiently due to the poor scalability of the histogram-based approaches.

Our results show that agents using our representation can learn faster and obtain policies of higher quality, suggesting that the representation as mean embedding is an efficient encoding of the global state configuration for swarm-based systems. Moreover, mean embeddings are simple to implement

inside existing neural network architectures and can be applied to any deep RL algorithm, which makes the approach applicable in a wide variety of scenarios.

4.1. Related Work

The main contribution of this work lies in the development of a compact representation of state information in swarm systems, which can easily be used within deep multi-agent reinforcement learning (MARL) settings that contain homogeneous agent groups. In fact, our work is mostly orthogonal to other research in the field of MARL and the presented ideas can be incorporated into most existing approaches. To provide an overview, we begin with a brief survey of algorithms used in (deep) MARL, we revisit the basics of mean embedding theory, and we summarize some classic approaches to swarm control for the rendezvous and pursuit evasion task.

4.1.1. Deep RL

Recently, there has been increasing interest in deep reinforcement learning for swarms and multi-agent systems in general. For example, Zheng et al. [162] provide a many-agent reinforcement learning platform based on a multi-channel image state representation, which uses Deep Q-Networks (DQN) [105] to learn decentralized control strategies in large grid worlds with discrete actions. Gupta, Egorov, and Kochenderfer [58] show a comparison of centralized, concurrent and parameter sharing approaches to cooperative deep MARL, using TRPO [137], DDPG [95] and DQN. They evaluate each method on three tasks, one of which is a pursuit task in a grid world using bitmap-like images as state representation. A variant of DDPG for multiple agents in Markov games using a centralized action-value function is provided by [101]. The authors evaluate the method on tasks like cooperative communication, navigation and others. The downside of a centralized action-value function is that the input space grows linearly with the number of agents, and hence, their approach scales poorly to large system sizes. A more scalable approach is presented by [159]. Employing mean field theory, the interactions within the population of agents are approximated by the interaction of a single agent with the average effect from the overall population, which has the effect that the action-value function input space stays constant. Experiments are conducted on a Gaussian squeeze problem, an Ising model, and a mixed cooperative-competitive battle game. Yet, the paper does not address the state representation problem for swarm systems.

Omidshafiei et al. [112] investigate hysteretic Q-learning [104] and distillation [131]. They use deep recurrent Q-networks [74] to solve single and multi-task Dec-POMDP problems. Following this work, Palmer et al. [117] add leniency [118] to the hysteretic approach to prevent "relative overgeneralization" of agents. The approach is evaluated on a coordinated multi-agent object transportation problem in a grid world with stochastic rewards.

Sunehag et al. [147] tackle the "lazy agent" problem in cooperative MARL with a single team reward by training each agent with a learned additive decomposition of a value function based on the team reward. Experiments show an increase in performance on cooperative two-player games in a grid world. Rashid et al. [127] further develop the idea with the insight that a full factorization of the value function is not necessary. Instead, they introduce a monotonicity constraint on the relationship between the global value function and each local value function. Results are presented on the StarCraft micro management domain.

Finally, Grover et al. [56] show a framework to model agent behavior as a representation learning problem. They learn an encoder-decoder embedding of agent policies via imitation learning based on interactions and evaluate it on a cooperative particle world [107] and a competitive two-agent robo sumo environment [139]. The design of the policy function in the approach of Mordatch and Abbeel [107] is similar to ours but the model uses a softmax pooling layer. However, instead of applying (model-free) reinforcement learning to optimize the parameters of the policy function, they build an end-to-end differentiable model of all agent and environment state dynamics and calculate the gradient of the return with respect to the parameters via backpropagation.

An application related to our approach can be found in the work by [49], where the authors use mean embeddings to learn a centralized controller for object manipulation with robot swarms. Here, the key idea is to directly embed the swarm configuration into a reproducing kernel Hilbert space, whereas our approach is based on embedding the agent's local view. Furthermore, using kernel-based feature spaces for the mean embedding scales poorly in the number of samples and in the dimensionality of the embedded information.

4.1.2. Optimization-Based Approaches for Swarm Systems

To provide a concise summary of the most relevant related work, we concentrate on optimization-based approaches that derive decentralized control strategies for the rendezvous and pursuit evasion problem considered in this paper. Ji and Egerstedt [84] derive a control mechanism preserving the connectedness of a group of agents with limited communication abilities for the rendezvous and formation control problem. The method focuses on high-level control with single integrator linear state manipulation and provides no rules for agents that are not part of the agent graph. Similarly, Gennaro and Jadbabaie [50] present a decentralized algorithm to maximize the connectivity (characterized by an exponential model) of a multi-agent system. The algorithm is based on the minimization of the second smallest eigenvalue of the Laplacian of the proximity graph. An approach providing a decentralized control strategy for the rendezvous problem for nonholonomic agents can be found in the work by [39]. Using tools from nonsmooth Lyapunov theory and graph theory, the stability of the overall system is examined. A control strategy for the pursuit evasion problem with multiple pursuers and single evader that we investigate in more detail later in this paper was proposed [164]. The authors derive decentralized control policies for the pursuers and the evader based on the minimization of Voronoi partitions. Again, the control mechanism is for high-level linear state manipulation. Furthermore, the method assumes visibility of the evader at all times. A survey on pursuit evasion in mobile robotics in general is provided by [36].

4.1.3. Analytic Approaches

Another line of work concerned with the curse of dimensionality can be found in the area of multi-player reach-avoid games. Chen, Zhou, and Tomlin [34], for example, look at pairwise interactions between agents. This way, they are able to use the Hamilton-Jacobian-Isaacs approach to solve a partial differential equation in the joint state space of the players. Similar work can be found in [33, 32, 163].

4.2. Background

In this section, we give a short overview of Trust Region Policy Optimization and mean embeddings of distributions.

4.2.1. Trust Region Policy Optimization

Trust Region Policy Optimization is an algorithm to optimize control policies in single-agent reinforcement learning problems [137]. These problems are formulated as Markov decision processes (MDPs), which can be compactly written as a tuple $\langle \mathcal{S}, \mathcal{A}, P, R \rangle$. In an MDP, an agent chooses an action $a \in \mathcal{A}$ according to some policy $\pi(a \mid s)$ based on its current state $s \in \mathcal{S}$ and progresses to state $s' \in \mathcal{S}$ according to the transition dynamics P, i.e., $s' \sim P(s' \mid s, a)$. After each step, the agent receives a reward $r = R(s, a)$, provided by the reward function R, which judges the quality of its decision. The goal of the agent is to find a policy that maximizes the cumulative reward achieved over a certain period of time.

In TRPO, the policy is parametrized by a parameter vector θ containing the weights and biases of a neural network. In the following, we denote this parametrized policy as π_θ. The reinforcement learning objective is expressed as finding a new policy that maximizes the expected advantage function of the current policy π_{old}, i.e., $J^{\text{TRPO}} = \mathbb{E}\left[\frac{\pi_\theta}{\pi_{\theta_{\text{old}}}} A^{\pi_{\text{old}}}(s, a)\right]$, where $A^{\pi_{\text{old}}}(s, a) = Q^{\pi_{\text{old}}}(s, a) - V^{\pi_{\text{old}}}(s)$. Herein, the state-action value function $Q^{\pi_{\text{old}}}(s, a)$ is typically estimated via trajectory rollouts, while for the value function $V^{\pi_{\text{old}}}(s)$ linear or neural network baselines are used that are fitted to the Monte-Carlo returns, resulting in an estimate $\hat{A}(s, a)$ for the advantage function. The objective is to be maximized subject to a fixed constraint on the Kullback-Leibler (KL) divergence of the policy before and after the parameter update, which ensures that the updates to the policy parameters θ are bounded, in order to avoid divergence of the learning process. The overall optimization problem is summarized as

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & \mathbb{E}\left[\frac{\pi_\theta}{\pi_{\theta_{\text{old}}}} \hat{A}(s, a)\right] \\ \text{subject to} \quad & \mathbb{E}[D_{\text{KL}}(\pi_{\theta_{\text{old}}} || \pi_\theta)] \leq \delta. \end{aligned}$$

The problem is approximately solved using conjugate gradient optimization, after linearizing the objective and quadratizing the constraint.

4.2.2. Mean Embeddings

Our work is inspired by the idea of embedding distributions into reproducing kernel Hilbert spaces [142] from where we borrow the concept of mean embeddings. A probability distribution $P(X)$ can be represented as an element in a reproducing kernel Hilbert space by its expected feature map (i.e., the mean embedding),

$$\mu_X = \mathbb{E}_X[\phi(X)],$$

where $\phi(x)$ is a (possibly infinite dimensional) feature mapping. Given a set of observations $\{x_1, \ldots, x_m\}$, drawn i.i.d. from $P(X)$, the empirical estimate of the expected feature map is given by

$$\hat{\mu}_X = \frac{1}{m} \sum_{i=1}^{m} \phi(x_i).$$

Using characteristic kernel functions $k(x, x') = \langle \phi(x),\ \phi(x') \rangle$, such as Gaussian RBF or Laplace kernels, mean embeddings can be used, for example, in two-sample tests [55] and independence tests [54]. A characteristic kernel is required to uniquely identify a distribution based on its mean embedding. However, this assumption can be relaxed to using finite feature spaces if the objective is merely to extract relevant information from a distribution such as, in our case, the information needed for the policy of the agents.

4.3. Deep Reinforcement Learning for Swarms

The reinforcement learning algorithm presented in the last section has been originally designed for single-agent learning. In order to apply this algorithm to the swarm setup, we switch to a different problem domain and show the implications on the learning algorithm. Policies in this context are then optimized in a centralized–learning / decentralized–execution fashion.

4.3.1. Problem Domain

The problem domain for our swarm system is best described as a swarm MDP environment [143]. The swarm MDP can be regarded as a special case of a decentralized partially observable Markov decision process (Dec-POMDP) [23] and is constructed in two steps. First, an agent prototype is defined as a tuple $\mathbb{A} = \langle \mathcal{S}, \mathcal{O}, \mathcal{A}, \pi \rangle$, determining the local properties of an agent in the system. Herein, $\mathcal{S}$ denotes the set of the agent's local states, $\mathcal{O}$ is the set of possible local observations, $\mathcal{A}$ is the set of actions available to the agent, and $\pi : \mathcal{O} \times \mathcal{A} \to [0, 1]$ is the agent's stochastic control policy. Based on this definition, the swarm MDP is constructed as $\langle N, \mathbb{A}, P, O, R \rangle$, where N is the number of agents in the system and $\mathbb{A}$ is the aforementioned agent prototype. The coupling of the agents is specified through a global state transition model $P : \mathcal{S}^N \times \mathcal{S}^N \times \mathcal{A}^N \to [0, \infty)$ and an observation model $O : \mathcal{S}^N \times \{1, \dots, N\} \to \mathcal{O}$, which determines the local observation $\boldsymbol{o}^i \in \mathcal{O}$ for agent i at a given swarm state $\boldsymbol{s} \in \mathcal{S}^N$, i.e., $\boldsymbol{o}^i = O(\boldsymbol{s}, i)$. Finally, $R : \mathcal{S}^N \times \mathcal{A}^N \to \mathbb{R}$ is the global reward function, which encodes the cooperative task for the swarm by providing an instantaneous reward feedback $R(\boldsymbol{s}, \boldsymbol{a})$ according to the current swarm state $\boldsymbol{s}$ and the corresponding joint action assignment $\boldsymbol{a} \in \mathcal{A}^N$ of the agents. The specific state dynamics and observation models considered in this paper are described in Section 4.4.

The model encodes two important properties of swarm networks: First, all agents in the system are assumed to be identical, and accordingly, they are all assigned the same decentralized policy π. This is an immediate consequence of the two-step construction of the model, which implies that all agents share the same internal architecture. Second, the agents are only partially informed about the global system state, as prescribed by the observation model O. Note that both the transition model and the observation model are assumed to be invariant to permutations of the agents in order to ensure the homogeneity of the system. For details, see [143].

4.3.2. Local Observation Models

The local observation $\boldsymbol{o}^i$ introduced in the last section is a combination of observations o^i_{loc} an agent makes about local properties (like the agent's current velocity or its distance to a wall) and observations O^i of other agents. In order to describe the observation model used for the agents, we use an interaction graph representation of the swarm. This graph is given by nodes $V = \{v_1, v_2, \dots, v_N\}$ corresponding to the agents in the swarm and an edge set $E \subset V \times V$, which we assume contains unordered pairs

of the form $\{v_i, v_j\}$ indicating that agents i and j are neighbors. The interaction graph is denoted as $\mathcal{G} = (V, E)$. If both the set of nodes and the set of edges are not changing, we call $\mathcal{G}$ a static interaction graph; if either of the set undergoes changes, we instead refer to $\mathcal{G}$ as a dynamic interaction graph.

The set of neighbors of agent i in the graph $\mathcal{G}$ is given by

$$\mathcal{N}_{\mathcal{G}}(i) = \{j \mid \{v_i, v_j\} \in E\}.$$

Within this neighborhood, agent i can sense local information about other agents, for example distance or bearing to each neighbor. We denote the information agent i receives from agent j as $o^{i,j} = f(s^i, s^j)$, which is a function of the local states of agent i and agent j. The observation $o^{i,j}$ is available for agent i only if $j \in \mathcal{N}_{\mathcal{G}}(i)$. Hence, the complete state information agent i receives from all neighbors is given by the set $O^i = \left\{o^{i,j} \mid j \in \mathcal{N}_{\mathcal{G}}(i)\right\}$.

As the observations of other agents are summarized in form of sets $\{O^i\}$, we require an efficient encoding that can be used as input to a neural network policy. In particular, it must meet the following two properties:

- The encoding needs to be invariant to the indexing of the agents, respecting the unorderedness of the elements in the observation set. Only by exploiting the system's inherent homogeneity we can escape the curse of dimensionality.
- The encoding must be applicable to varying set sizes because the local graph structure might change dynamically. Even if each agent can observe the entire system at all times, the encoding should be applicable for different swarm sizes.

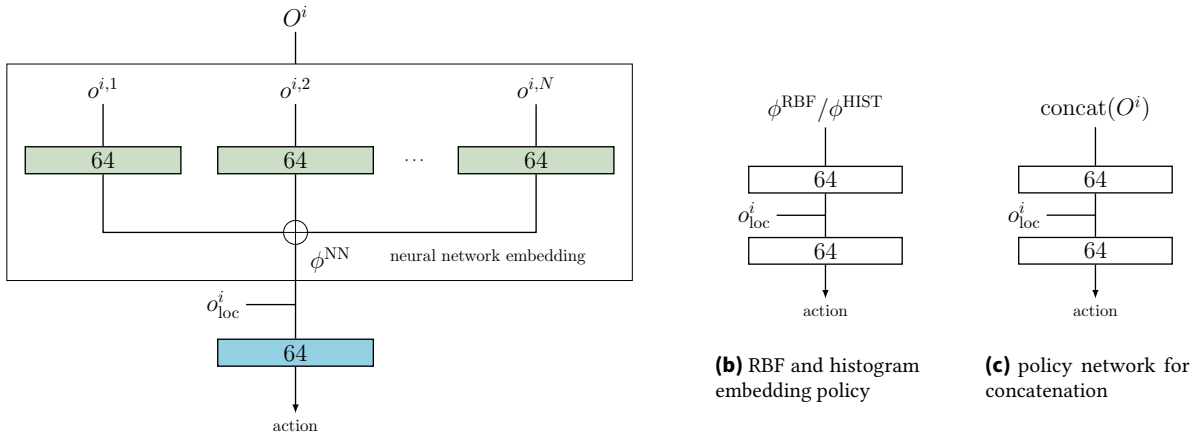
4.3.3. Local Communication Models

In addition to perceiving local state information of neighboring agents, the agents can also communicate information about the interaction graph $\mathcal{G}$ [80]. For example, agent j can transmit the number of perceived neighbors to agent i. Furthermore, the agents can also perform more complex operations on their local neighborhood graph. For example, they could compute the shortest distance to a target point (such as an evader) that is perceived by at least one agent within their local sub-graph. Hence, by using local communication protocols, observation $o^{i,j}$ can contain information about both, the local states s^i and s^j as well as the graph $\mathcal{G}$, i.e., $o^{i,j} = f(s^i, s^j, \mathcal{G})$.

4.3.4. Mean Embeddings as State Representations for Swarms

In the simplest case, the local observation $o^{i,j}$ that agent i receives of agent j is composed of the distance and the bearing angle of agent i to agent j. However, $o^{i,j}$ can also contain more complex information, such as relative velocities or orientations. A straightforward way to represent the information set O^i is to concatenate the local quantities $\{o^{i,j}\}_j$ into a single observation vector. However, as mentioned before, this representation has various drawbacks as it ignores the permutation invariance inherent to a homogeneous agent network. Furthermore, it grows linearly with the number of agents in the swarm and is, therefore, limited to a fixed number of neighbors when used in combination with neural network policies.

To resolve these issues, we treat the elements in the information set O^i as samples from a distribution that characterizes the current swarm configuration, i.e., $o^{i,j} \sim p_i(\cdot \mid \mathbf{s})$. We can now use an empirical encoding of this distribution in order to achieve permutation invariance of the elements of O^i as well



(a) neural network embedding policy network

(b) RBF and histogram embedding policy

(c) policy network for concatenation

Figure 4.1.: Illustration of (a) the neural network mean embedding policy, (b) the network architecture used for the RBF and histogram representation, and (c) for the simple concatenation of observations. The numbers inside the boxes denote the dimensionalities of the hidden layers. The color coding in (a) highlights which layers share the same weights. The plus sign denotes the mean of the feature activations.

as flexibility to the size of O^i. As highlighted in Section 4.2.2, a simple way is to use a mean feature embedding, i.e.,

$$\hat{\mu}_{O^i} = \frac{1}{|O^i|} \sum_{o^{i,j} \in O^i} \phi(o^{i,j}),$$

where ϕ defines the feature space of the mean embedding. The input dimensionality to the policy is given by the dimensionality of the feature space of the mean embedding and, hence, it does not depend on the size of the information set O^i any more. This allows us to use the embedding $\hat{\mu}_{O^i}$ as input to a neural network used in deep RL. In the following sections, we describe different feature spaces that can be used for the mean embedding. Figure 4.1 illustrates the resulting policy architectures with further details given in Appendix A.6.

4.3.4.1. Neural Network Feature Embeddings

In line with the deep RL paradigm, we propose to use a neural network as feature mapping ϕ^{NN} whose parameters are determined by the reinforcement learning algorithm. Using a neural network to define the feature space allows us to handle high dimensional observations, which is not feasible with traditional approaches such as histograms [80]. In our experiments, a rather shallow architecture with one layer of RELU units already performed very well, but deeper architectures could be used for more complex applications. To the best of our knowledge, we present the first approach for using neural networks to define the feature space of a mean embedding.

4.3.4.2. Histograms

An alternative feature space are provided by histograms, which can be related to image-like representations. In this approach, we discretize the space of certain features, such as the distance and bearing to other agents, into a fixed number of bins. This way, we can collect information about neighboring

agents in the form of a fixed-size multi-dimensional histogram. Herein, the histogram bins define a feature mapping ϕ^{HIST} using a one-hot-coding for each observed agent. A detailed description of this approach can be found in our previous work [80]. While the approach works well in discrete environments where each cell is only occupied by a single agent, the representation can lead to blurring effects between agents in the continuous case. Moreover, the histogram approach does not scale well with the dimensionality of the feature space.

4.3.4.3. Radial Basis Functions

A specific problem of the histogram approach is the hard assignment of agents into bins, which results in abrupt changes in the observation space when a neighboring agent moves from one bin to another. A more fine-grained representation can be achieved by using RBF networks with a fixed number of basis functions evenly distributed over the observation space. The resulting feature mapping ϕ^{RBF} is then defined by the activations of each basis function and can be seen as a "soft-assigned" histogram. However, both representations (histogram and RBF) suffer from the curse of dimensionality, as the number of required basis functions typically increases exponentially with the number of dimensions of the observation vector.

4.3.5. Other Representation Techniques

Inspired by the work of Mordatch and Abbeel [107], we also investigate a policy function that uses a softmax pooling layer instead of the mean embedding. The elements of the pooling layer $\psi = [\psi_1, \ldots, \psi_K]$ are given by

$$\psi_k = \frac{\sum_{o^{i,j} \in O^i} \exp\left(\alpha \phi_k(o^{i,j})\right) \phi_k(o^{i,j})}{\sum_{o^{i,j} \in O^i} \exp\left(\alpha \phi_k(o^{i,j})\right)}$$

for each feature dimension of $\phi = [\phi_1, \ldots, \phi_K]$ with a temperature parameter α. Note that the representation becomes identical to our mean embedding for $\alpha = 0$, while setting $\alpha \gg 1$ results in max-pooling and $\alpha \ll -1$ corresponds to min-pooling. In our experiments, we choose $\alpha = 1$ as a trade-off between a mean embedding and max-pooling and additionally study the performance of max-pooling over each individual feature dimension.

4.3.6. Adaption of TRPO to the Homogeneous Swarm Setup

Gupta, Egorov, and Kochenderfer [58] present a parameter-sharing variant of TRPO that can be used in a multi-agent setup. During the learning phase, the algorithm collects experiences made by all agents and uses these experiences to optimize one policy with a single set of parameters θ. Since, in the swarm setup, we assume homogeneous agents that are potentially indistinguishable to each other, we omit the agent index introduced by [58]. The optimization problem is expressed using advantage values based on all agents' observations. During execution, however, each agent has only access to its own perception. Hence, the terminology of centralized–learning / decentralized–execution is chosen.

During the trajectory roll-outs, we use a sub-sampling strategy to achieve a trade-off between the number of samples and the variability in advantage values seen by the learning algorithm. Our implementation is based on the OpenAI baselines version of TRPO with 10 MPI workers, where each worker samples 2048 time steps, resulting in $2048N$ samples. Subsequently, we randomly choose the data of 8 agents, yielding $2048 \times 10 \times 8 = 163840$ samples per TRPO iteration. The chosen number of samples worked well throughout our experiments and was not extensively tuned.

4.4. Experimental Results

Our experiments are designed to study the use of mean embeddings in a cooperative swarm setting. The three main aspects are:

1. How do the different mean embeddings (neural networks, histograms and RBF representation) compare when provided with the same state information content?
2. How does the mean embedding using neural networks perform when provided with additional state information while keeping the dimensionality of the feature space constant?
3. How does the mean embedding of neural network features compare against other pooling techniques?

In this section, we first introduce the swarm model used for our experiments and present the results of different evaluations afterwards. During a policy update, a fixed number of K trajectories are sampled, each yielding a return of $G_k = \sum_{t=1}^{T} r(t)$. The results are presented in terms of the average return, denoted as $\bar{G} = \frac{1}{K} \sum_{k=1}^{K} G_k$.

4.4.1. Swarm Models

Our agents are modeled as unicycles [a commonly used agent model in mobile robotics; see, for example, 42], where the control parameters either manipulate the linear and angular velocities v and ω (single integrator dynamics) or the corresponding accelerations $\dot{v}$ and $\dot{\omega}$ (double integrator dynamics). In the single integrator case, the state of an agent is defined by its location $\boldsymbol{x} = (x, y)$ and orientation ϕ. In case of double integrator dynamics, the agent is additionally characterized by its current velocities. The exact state definition and kinematic models can be found in Appendix A.1. Note that these agent models are more complex than what is typically considered in optimization-based approaches, which mostly assume single integrator dynamics directly on $\boldsymbol{x}$. Depending on the task, we either opt for a closed state space where the limits act as walls, or a periodic toroidal state space where agents exceeding the boundaries reappear on the opposite side of the space. Either way, the state is bounded by $x_{\max} = y_{\max} = 100$.

We study two different observation scenarios for the agents, i.e., global observability and local observability. In the case of global observability, all agents are neighbors, i.e.

$$\mathcal{N}_{\mathcal{G}}(i) = \{j \in \{1, \ldots, N\} \mid i \neq j\},$$

which corresponds to a fully connected static interaction graph. For the local observability case, we use Δ-disk proximity graphs, where edges are formed if the distance $d^{i,j} = \sqrt{(x^i - x^j)^2 + (y^i - y^j)^2}$ between agents i and j is less than a pre-defined cut-off distance d_c for communication, resulting in a dynamic interaction graph. The neighborhood set of the graph is then defined as

$$\mathcal{N}_{\mathcal{G}}(i) = \{j \in \{1, \ldots, N\} \mid i \neq j,\ d^{i,j} \leq d_c\}.$$

For a detailed description of all observational features available to the agents in the tasks, see Appendices A.2 and A.3.

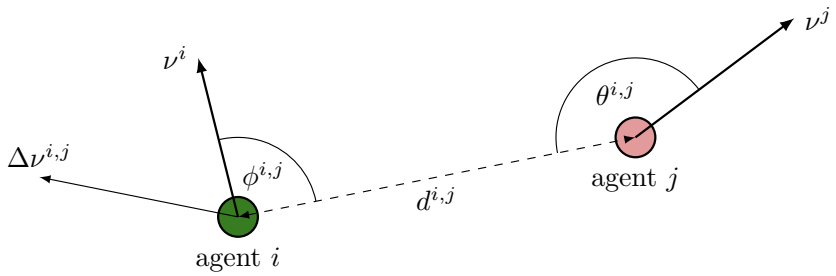


Figure 4.2.: Illustration of two neighboring agents facing the direction of their velocity vectors v^i and v^j, along with the observed quantities, shown with respect to agent i. The observed quantities are the bearing $\phi^{i,j}$ to agent j, agent j's relative orientation $\theta^{i,j}$ to agent i, their distance $d^{i,j}$ and a relative velocity vector $\Delta v^{i,j} = v^i - v^j$. In this trivial example, agent i's observed neighborhood size as well as the neighborhood size communicated by agent j are $|\mathcal{N}(i)| = |\mathcal{N}(j)| = 1$.

4.4.2. Rendezvous

In the rendezvous problem, the goal is to minimize the distances between all agents. The reason why we choose this experiment is because a simple optimization-based baseline controller can be defined by the consensus protocol,

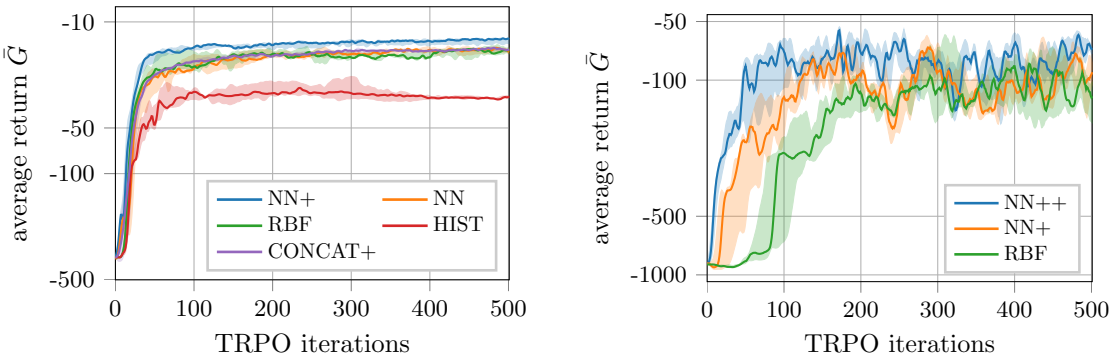
$$\dot{\boldsymbol{x}}^i = -\sum_{j \in \mathcal{N}(i)} (\boldsymbol{x}^i - \boldsymbol{x}^j),$$

where $\boldsymbol{x}^i = (x^i, y^i)$ denotes the location of agent i. To make the solution compatible to the double integrator agent model, we make use of a PD-controller (see Appendix A.1 for details). The reward function for the problem can be found in Appendix A.5.1.

We evaluate different observation vectors $o^{i,j}$ which are fed into the policy. To compare the histogram and RBF embedding with the proposed neural network approach, we restrict the *basic* observation model (see below) to a set of two features: the distance $d^{i,j}$ between two agents and the corresponding bearing $\phi^{i,j}$. This restriction allows for a comparison to the optimization-based consensus protocol, which is based on displacements (an equivalent formulation of distance and bearing). To show that the neural network embeddings can be used with more informative observations, we further introduce an *extended* set and a communication (*comm*) set. These sets may include relative orientations $\theta^{i,j}$ or relative velocities $\Delta v^{i,j}$ (depending on the agent dynamics), as well as the own neighborhood size and those of the neighbors. An illustration of these quantities can be found in Figure 4.2.

4.4.2.1. Global Observability

First, we study the rendezvous problem with 20 agents in the global observability setting with double integrator dynamics to illustrate the algorithm's ability to handle complex dynamics. To this end, we compare the performances of policies using histogram, RBF and neural network embeddings on the *basic* set, as well as neural network embeddings on the *extended* set. The observations $o^{i,j}$ in the *basic* set comprise the distance $d^{i,j}$ and bearing $\phi^{i,j}$. In the *extended* set, which is processed only via neural network embeddings, we additionally add neighboring agents' relative orientations $\theta^{i,j}$ and velocities



(a) 20 agents with global observability

(b) 20 agents with local observability

Figure 4.3.: Learning curves for the rendezvous task with different observation models. The curves show the median of the average return $\bar{G}$ based on the top five trials on a log scale. **Legend:** NN++: neural network mean embedding of *comm* set, NN+: neural network mean embedding of *extended* set, NN: neural network embedding of *basic* set, RBF: radial basis function embedding of *basic* set, HIST: histogram embedding of *basic* set, CONCAT+: simple concatenation of *extended* set.

$\Delta v^{i,j}$. The local properties o^i_{loc} consist of a shortest distance and orientation to the closest boundary, i.e., d^i_{wall} and ϕ^i_{wall}. The sets are summarized as follows:

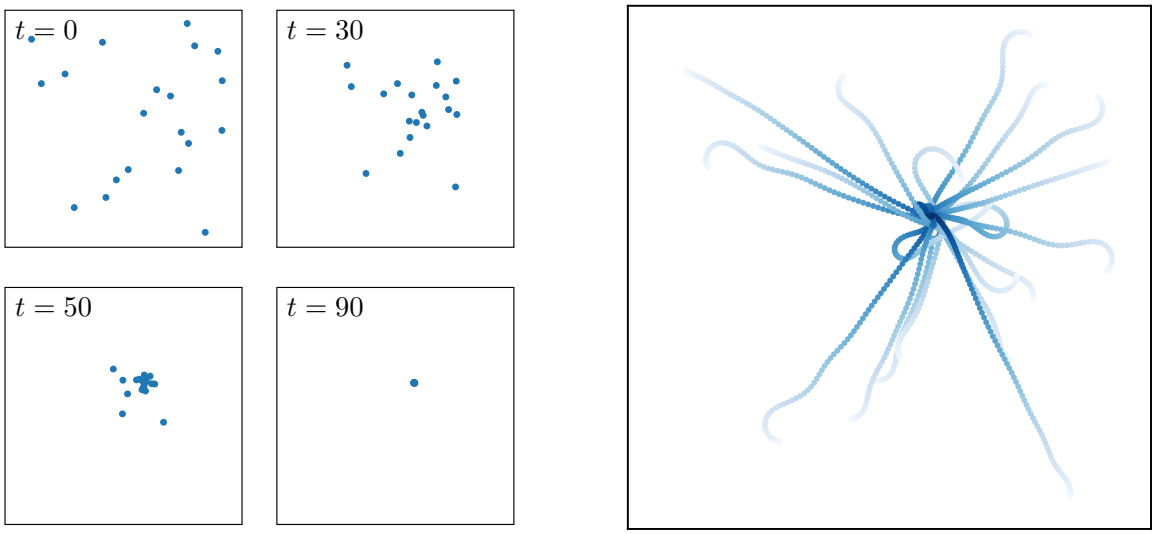
$$\begin{aligned} \textit{Basic}: \quad & o^{i,j} = \{d^{i,j},\ \phi^{i,j}\} && o^i_{\text{loc}} = \{d^i_{\text{wall}},\ \phi^i_{\text{wall}}\} \\ \textit{Extended}: \quad & o^{i,j} = \{d^{i,j},\ \phi^{i,j},\ \theta^{i,j},\ \Delta v^{i,j}\} && o^i_{\text{loc}} = \{d^i_{\text{wall}},\ \phi^i_{\text{wall}}\}. \end{aligned}$$

The results are shown in Figure 4.3a. On first sight, they reveal that all shown methods eventually find a successful strategy, with the histogram approach showing worst performance. Upon a closer look, it can be seen that the best solutions are found with the neural network embedding, in which case the learning algorithm also converges faster, demonstrating that this form of embedding serves as a suitable representation for deep RL. However, there are two important things to note:

- The differences between the approaches seem to be small due to the wide range of obtained reward values, but the NN+ method brings in fact a significant performance gain. Compared to the NN and RBF embedding, the performance of the learned NN+ policy is ~10% better in terms of the average return of an episode (Figure 4.3a) and almost twice as good ($\sim 4 \times 10^{-2}$ versus $\sim 8 \times 10^{-2}$) in terms of the mean distance between agents at the steady state solution after around 200 time steps (Figure 4.5a). Furthermore, the NN+ embedding reaches the mean distance achieved by the NN and RBF embeddings roughly 20 to 30 time steps earlier, which corresponds to an improvement of ~25%.
- Although the performance gain of NN+ can be partly explained by the use of the extended feature set, experiments with the same feature set using the histogram / RBF approach did *not* succeed to find solutions to the rendezvous problem; hence, the corresponding results are omitted. The reason is that the dimensionality of the input space scales exponentially for the histogram / RBF approach while only linearly for the neural network embedding, which results in a more compact feature representation that keeps the learning problem tractable.

Together, these two observations suggest that the neural network embedding provides a suitable learning architecture for deep RL, whereas the histogram / RBF approach is only suited for low-dimensional spaces.

Figure 4.4 shows a visualization of a policy using the neural network mean embedding of the *extended* set. After random initialization, the agents' locations quickly converge to a single point. Figure 4.5



(a) Snapshots

(b) Full episode

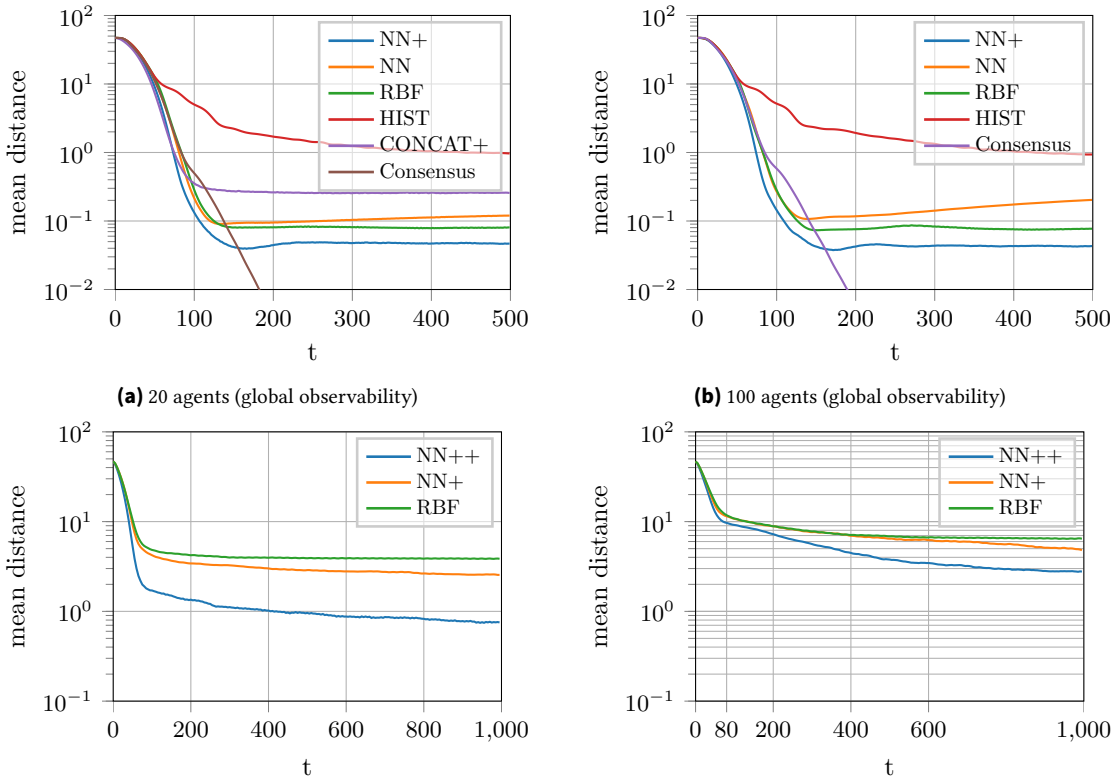
Figure 4.4.: Visualization of a learned policy for the pursuit evasion task. The policy is learned and executed by 10 agents using a neural network mean embedding of the *extended* set. Pursuers are illustrated in blue, the evader is highlighted in red. Visualization of a learned policy for the rendezvous task. The policy is learned and executed by 20 agents using a neural network mean embedding of the *extended* set.

shows performance evaluations of the best policies found with each of the mean embedding approaches. We plot the evolution of the mean distance between all agents over 1000 episodes with equal starting conditions. We also include the performance of the PD-controller defined in Appendix A.1. It can be seen in Figures 4.5a and 4.5c that the policies using the neural network embeddings decrease the mean distance most quickly and also find the best steady-state solutions among all learning approaches. While the optimization-based solution (PD) eventually drives the mean distance to zero, a small error remains for the learning-based approaches. However, the learned policies are faster in reducing the distance and therefore show a better average reward. Although the optimization-based policy is guaranteed to find an optimal stationary solution, the approach is build for simpler dynamics and hence performs suboptimally in the considered scenario. Note, that the controller gains for this approach have been tuned manually to maximize performance.

In order to show the generalization abilities of the embeddings, we finally evaluate the obtained policies (except for the concatenation) with 100 agents. The results are displayed in Figure 4.5b. Again, the neural network embedding of the *extended* set is quickest in reducing the inter-agent distances, resulting in the best overall performance.

4.4.2.2. Local Observability

The local observability case is studied with 20 agents and a communication cut-off distance of $d_c = 40$. Due to the increased difficulty of the task, we resort to single integrator dynamics for this experiment. Again, we evaluate the *basic* and the *extended* set, which in this case contains the single integrator state information. Accordingly, we remove the relative velocities from the information sets. Moreover, we employ a local communication strategy that transmits the number of observed neighbors as additional information. Note that this information can be used by the agents to estimate in which direction the center of mass of the swarm is located.



(a) 20 agents (global observability)

(b) 100 agents (global observability)

(c) 20 agents (local observability)

(d) 10 agents (local observability)

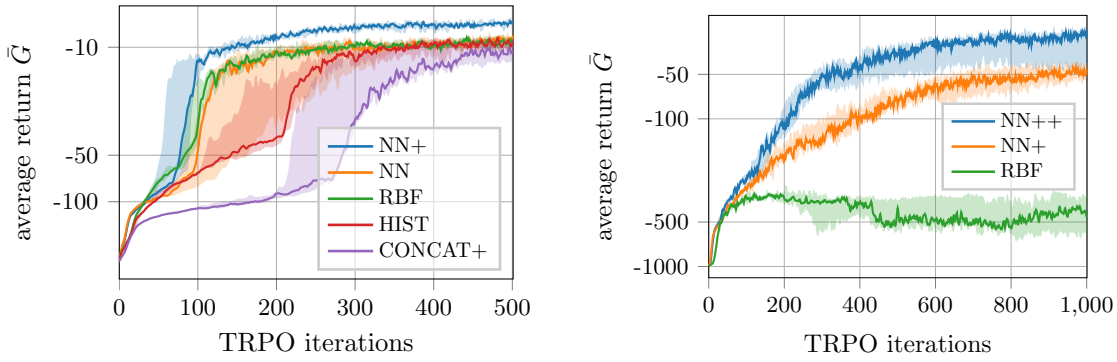
Figure 4.5.: Comparison of the mean distance between agents in the rendezvous experiment achieved by the best learned policies and the consensus protocol. In (a) and (b), the policy is learned with 20 agents and executed by 20 and 100 agents, respectively. In (c) and (d), the policy is learned with 20 agents and executed by 20 and 10 agents. Results are averaged over 1000 episodes with identical starting conditions.

While the received neighborhood sizes $\{|\mathcal{N}(j)|\}_{j\in\mathcal{N}(i)}$ are treated as part of agent i's local observation of the swarm, the own perceived neighborhood size $|\mathcal{N}(i)|$ is considered as part of the local features o^i_{loc}. The observation models for the local observability case are thus summarized as:

$$
\begin{aligned}
\textit{Basic}: \quad & o^{i,j} = \{d^{i,j},\ \phi^{i,j}\} && o^i_{\text{loc}} = \{d^i_{\text{wall}},\ \phi^i_{\text{wall}}\} \\
\textit{Extended}: \quad & o^{i,j} = \{d^{i,j},\ \phi^{i,j},\ \theta^{i,j}\} && o^i_{\text{loc}} = \{d^i_{\text{wall}},\ \phi^i_{\text{wall}}\} \\
\textit{Comm}: \quad & o^{i,j} = \{d^{i,j},\ \phi^{i,j},\ \theta^{i,j},\ |\mathcal{N}(j)|\} && o^i_{\text{loc}} = \{d^i_{\text{wall}},\ \phi^i_{\text{wall}},\ |\mathcal{N}(i)|\}.
\end{aligned}
$$

For the experiment, we limit our comparison to RBF embeddings (which showed best performance among all non-neural-network solutions) of the *basic* set and neural network embeddings of the *extended* set and the *comm* set. The results are illustrated in Figure 4.3b, which shows that the neural network embeddings lead to a quicker learning progress. Furthermore, by introducing the *comm* model, a higher return is achieved. Compared to the global observability case, however, the learning process exhibits an increased variance caused by the information loss in the reward signal (see Appendix A.5).

Figure 4.5c illustrates the performances of the learned policies. Again, the neural network embedding is quicker in reducing the inter-agent distances and converges to better steady-state solutions. In order to test the efficacy of the communication protocol, we further evaluate the learned policies with 10 agents. The results are displayed in Figure 4.5d. As expected, the performance decreases due to the lower chance of agents seeing each other but we still notice a benefit caused by the communication.



(a) 10 agents with global observability

(b) 20 agents with local observability

Figure 4.6.: Learning curves for the pursuit evasion task with different observation models. The curves show the median of the average return $\bar{G}$ based on the top five trials on a log scale. **Legend:** NN++: neural network mean embedding of *comm* set, NN+: neural network mean embedding of *extended* set, RBF: radial basis function embedding of *basic* set, HIST: histogram embedding of *basic* set, CONCAT+: concatenation of *extended* set.

4.4.3. Pursuit Evasion with a Single Evader

Our implementation of the pursuit evasion scenario is based on the work by [164], from which we adopt the evader strategy. The strategy is based on Voronoi regions, which the pursuers try to minimize and the evader tries to maximize. While the original paper considers a closed world, we change the world type from closed to periodic, thereby making it impossible to trap the evader in a corner. In order to encourage a higher level of coordination between the agents, we set the evader's maximum velocity to twice the pursuers' maximum velocity. An episode ends once the evader is caught, i.e., if the distance of the closest pursuer is below a certain threshold. In all our experiments, the evader policy is fixed and not part of the learning process. The reward function for the problem is based on the shortest distance of the closest pursuer and can be found in Appendix A.5.2.

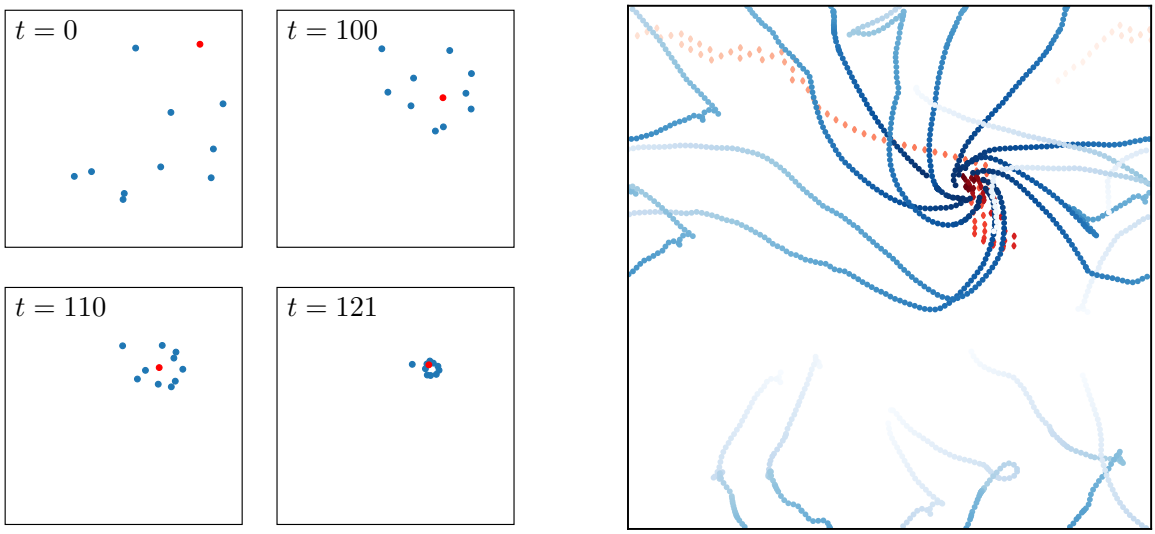
4.4.3.1. Global Observability

Again, we study the global observability case with ten agents. Since the pursuit of an evader is a more challenging task already, we reduce the movement complexity to single integrator dynamics. The *basic* and *extended* set are equal to those in the rendezvous experiment with single integrator dynamics, with additional information about the evader in the local properties o^i_{loc}. In here, we add the distance $d^{i,e}$ and bearing $\phi^{i,e}$ of agent i to the evader e. Accordingly, the observation sets are given as:

$$\begin{aligned} \textit{Basic}: \quad & o^{i,j} = \{d^{i,j},\ \phi^{i,j}\} && o^i_{\text{loc}} = \{d^i_{\text{wall}},\ \phi^i_{\text{wall}},\ d^{i,e},\ \phi^{i,e}\} \\ \textit{Extended}: \quad & o^{i,j} = \{d^{i,j},\ \phi^{i,j},\ \theta^{i,j}\} && o^i_{\text{loc}} = \{d^i_{\text{wall}},\ \phi^i_{\text{wall}},\ d^{i,e},\ \phi^{i,e}\}. \end{aligned}$$

The results in Figure 4.6a reveal that successful strategies can be obtained with all methods. However, this time, a clear advantage can be seen for the policies using neural network mean embeddings of the *extended* set, both in terms of behavior quality and in the number of samples necessary to find the solution.

Figure 4.7 illustrates the strategy that such a policy exerts. After random initialization, the agents first spread in a way that leaves no possibility for the evader to increase its Voronoi region, thereby keeping the evader almost on the same spot. Once this configuration is reached, they surround the evader in a



(a) snapshots

(b) full episode

Figure 4.7.: Visualization of a learned policy for the pursuit evasion task. The policy is learned and executed by 10 agents using a neural network mean embedding of the *extended* set. Pursuers are illustrated in blue, the evader is highlighted in red.

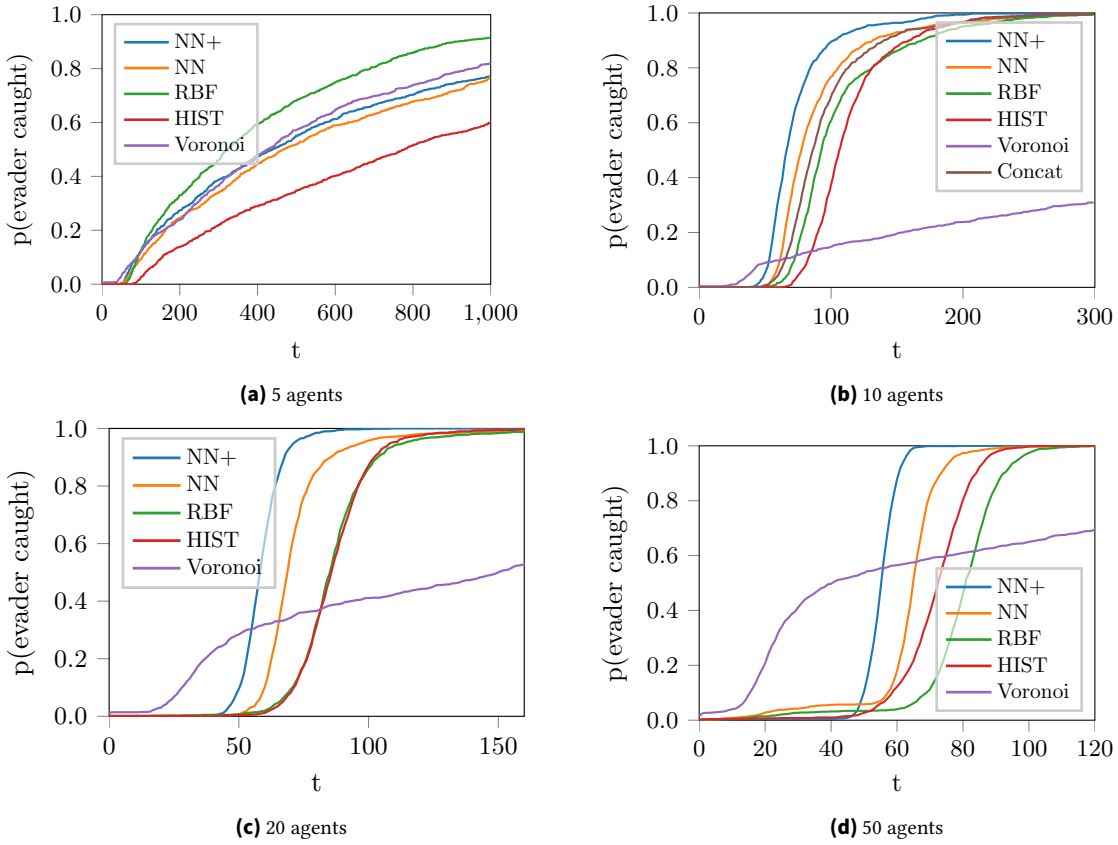
circular pattern and start to reduce the distance until one pursuer successfully reaches the distance threshold.

To investigate the performance of the best mean embedding policies (learned with 10 agents), we estimate the corresponding probabilities that the evader is caught within a certain time frame. For the sake of completeness, we also include the method proposed by [164], which was originally not designed for a setup with a faster evader, though. The results are plotted in Figure 4.8 as the fraction of episodes ending at the respective time instant, averaged over 1000 episodes. The plot in Figure 4.8b reveals that the evader may be caught using all presented methods if the policies are executed for long time periods. As already indicated by the learning curves, using a neural network mean embedding representation yields the quickest capture among all methods. The additional information in the *extended* set further increases performance.

Next, we examine the generalization abilities of the learned policies, this time on scenarios with 5, 20 and 50 agents (Figures 4.8a, 4.8c and 4.8d). Increasing the amount of agents leads to a quicker capture for all methods; however, the best performance is still shown by the agents executing a neural network policy based on embeddings of the *extended* set. Interestingly, when using fewer agents than in the original setup (Figure 4.8a), all methods struggle to capture the evader. After inspection of the behavior, we found that the strategy of establishing a circle around the evader causes too large gaps between the agents through which the evader can escape.

4.4.3.2. Local Observability

The local observability case is studied with 20 agents and a communication cut-off distance of $d_c = 40$. Additionally, we introduce an observation radius $d_o = 20$ within which the pursuers can observe the distance and bearing to the evader. We reuse the *basic* and *extended* set from last section and modify the *comm* set to include the shortest path information of other agents in the neighborhood of agent i to the evader. This way, each agent i can compute a shortest path to the evader over a graph of



(a) 5 agents

(b) 10 agents

(c) 20 agents

(d) 50 agents

Figure 4.8.: Performance comparison of the best learned policies and the optimization approach minimizing Voronoi regions in the pursuit evasion task with global observability. The curves show the probability that the evader is caught after t time steps. All policies are learned with 10 agents but executed with different agent numbers, as indicated below each subfigure. Results are averaged over 1000 episodes with identical starting conditions.

connected agents, such that the path $P = (v^1, v^2, \ldots, v^M)$ minimizes the sum $d^{i,e}_{\text{min}} = \sum_{m=1}^{M-1} d^{m,m+1}$ where v^1 represents agent i and v^M is the evader. The observation sets are given as:

$$
\begin{aligned}
\textit{Basic} : o^{i,j} &= \{d^{i,j},\ \phi^{i,j}\} & o^i_{\text{loc}} &= \{d^i_{\text{wall}},\ \phi^i_{\text{wall}},\ d^{i,e},\ \phi^{i,e}\} \\
\textit{Extended} : o^{i,j} &= \{d^{i,j},\ \phi^{i,j},\ \theta^{i,j}\} & o^i_{\text{loc}} &= \{d^i_{\text{wall}},\ \phi^i_{\text{wall}},\ d^{i,e},\ \phi^{i,e}\} \\
\textit{Comm} : o^{i,j} &= \{d^{i,j},\ \phi^{i,j},\ \theta^{i,j},\ d^{j,e}_{\text{min}}\} & o^i_{\text{loc}} &= \{d^i_{\text{wall}},\ \phi^i_{\text{wall}},\ d^{i,e},\ \phi^{i,e},\ d^{i,e}_{\text{min}}\}.
\end{aligned}
$$

Note that in this case the distance and bearing to an evader are only available if $d^{i,e} \leq d_o$. Furthermore, the correct shortest path is only available if an agent and the evader are in the same sub-graph, otherwise, a pre-defined value is fed into the policy.

Again, we limit the comparison for the local observability case to the more promising methods of neural network and RBF mean embeddings. The results in Figure 4.6b show that the performance gain of the neural network mean embeddings is even more noticeable than in the global observability case, with a clear advantage in the presence of the local communication protocols. The inspection of the termination probabilities in Figure 4.9 confirms that the neural network mean embedding results in a significantly improved policy.

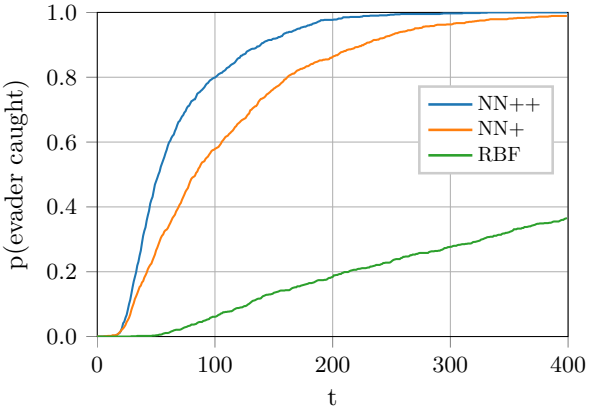


Figure 4.9.: Performance comparison of the best policies in the pursuit evasion task with local observability. The curves show the probability that the evader is caught after t time steps. All policies are learned and executed by 20 agents. Results are averaged over 1000 episodes with identical starting conditions.

4.4.4. Pursuit Evasion with Multiple Evaders

Lastly, we study a pursuit evasion scenario with multiple evaders, i.e., we assume that agent i receives observation samples $\{o^{i,e}\}$ from several evaders, which are processed using a second mean embedding to account for the variable set size. Where in the previous experiment the agents had precise information about the evader in terms of distance and bearing, they now have to extract this information from the respective embedding. An additional level of difficulty results from the fact that the reward function no longer provides any guidance in terms of the distances to the evaders since it only counts the number of evaders caught in each time step (see Appendix A.5.3 for details).

We study a scenario with 50 pursuers and 5 evaders using the global observability setup in Section 4.4.3.1, except that we respawn caught evaders to a new random location instead of terminating the episode. The observation sets, containing the same type of information but arranged according to the inputs of the neural networks, are designed as follows:

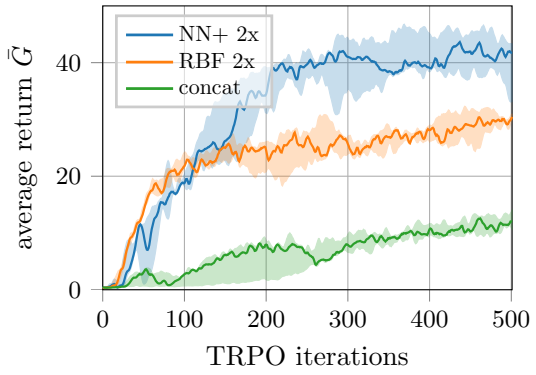
$$\begin{aligned} \textit{Basic}: \quad & o^{i,j} = \{d^{i,j},\ \phi^{i,j}\} && o^{i,e} = \{d^{i,e},\ \phi^{i,e}\} && o^{i}_{\text{loc}} = \{d^{i}_{\text{wall}},\ \phi^{i}_{\text{wall}}\} \\ \textit{Extended}: \quad & o^{i,j} = \{d^{i,j},\ \phi^{i,j},\ \theta^{i,j}\} && o^{i,e} = \{d^{i,e},\ \phi^{i,e}\} && o^{i}_{\text{loc}} = \{d^{i}_{\text{wall}},\ \phi^{i}_{\text{wall}}\}. \end{aligned}$$

Figure 4.10a shows the learning curves for policies with neural network and RBF mean embeddings and for the concatenation approach. The return directly relates to the number of evaders caught during an episode. Again, the neural network mean embedding performs significantly better than the RBF embedding. The curves clearly indicate the positive influence of the additional information present in the *extended* set. With this amount of agents, the dimensionality of the concatenation has increased to a point where learning is no longer feasible.

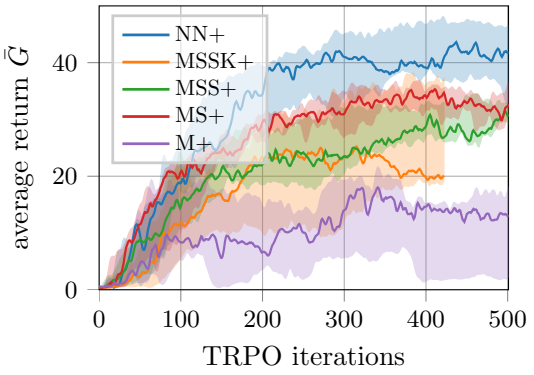
4.4.5. Evaluation of Pooling Functions

Figure 4.11 shows learning curves of policies based on mean embeddings, softmax pooling, and max-pooling (as described in Section 4.3.5) of features of the *extended* set for the rendezvous and pursuit evasion task with global observability.

In the rendezvous task (Figure 4.11a), all pooling techniques eventually manage to find a good solution. Policies using neural network mean embedding, however, on average converge more quickly while

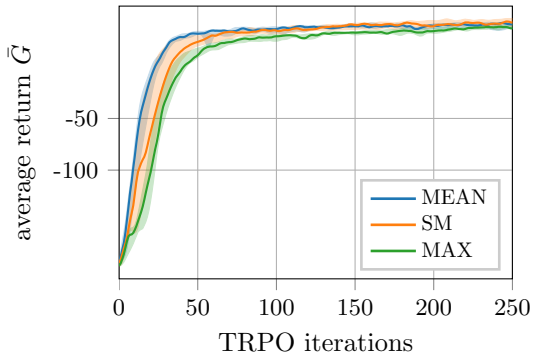


(a) Evaluation of different observation models and embedding techniques.

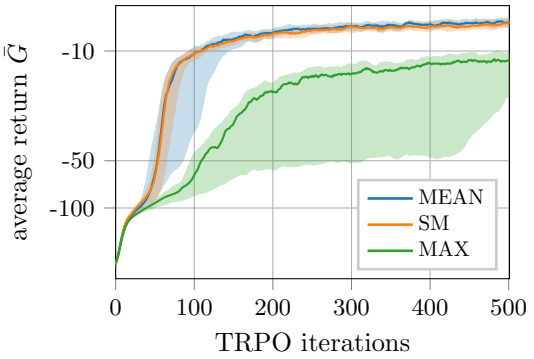


(b) Comparison of the proposed mean embedding policy to policies with statistics of the observations as input.

Figure 4.10.: Learning curves for 50 agent pursuit evasion with 5 evaders. The curves show the median of the average return $\bar{G}$ based on the top five trials. **Legend:** NN+ 2x: two neural network mean embeddings of the *extended* set, RBF 2x: two radial basis function mean embeddings of the *basic* set, concat: simple concatenation of *extended* set. MSSK+, MSS+, MS+ and M+: Combinations of mean, standard deviation, skew and kurtosis of the features in the *extended* set.



(a) Rendezvous with 20 agents and global observability.



(b) Pursuit evasion with 10 pursuers and global observability.

Figure 4.11.: Learning curves of different embedding and pooling architectures based on the *extended* set. The curves show the median of the average return $\bar{G}$ based on the top five trials on a log scale. **Legend:** MEAN: neural network mean embedding, SM: softmax feature pooling, MAX: max feature pooling.

policies using max-pooling show slightly worse performance. Given its reduced computational complexity compared to the softmax-pooling, the mean embedding provides the most effective approach among all proposed architectures.

When examining the results of the pursuit evasion task (Figure 4.11b), we find that the algorithm produces two distinct solutions. A sub-optimal one, which is only able to circle the evader but is unable to catch it (a catch is realized if the distance of the closest pursuer to the evader is below a certain threshold), and a solution which additionally catches the evader after a short period of time. Therefore, we not only report the performance of the top 5 trials out of 16, but also provide the number of times the algorithm was able to discover the better of the two solution (Table 4.1). Once that the algorithm finds a good solution, the mean embedding and softmax solutions perform comparably well but the max-pooling approach shows a significantly worse performance. More importantly, however, the algorithm was able to find the good solution more often using the mean embedding than using the other pooling approaches.

mean	sm	max
10/16	6/16	4/16

Table 4.1.: Number of times the algorithm discovered policies that led to a successful catch.

4.4.6. Comparison to Moment-Based Representations

Finally, we compare the mean embedding of neural network features to a representation using statistics of the input. Figure 4.10b shows an evaluation on the pursuit evasion task with 50 agents and 5 evaders. Here, we use combinations of the empirical mean, standard deviation, skew and kurtosis of each feature of the *extended* set as the input to a policy function. The plot reveals that neural network mean embeddings can capture more relevant information about the characteristics of the distribution of agents than simple statistics of the elements in the *extended* set. Similar results were obtained for the other tasks although the differences in performance were less pronounced.

4.4.7. Computational Complexity

Unlike in classical optimization-based control, where the controller is derived from an assumed dynamics model, model-free reinforcement learning methods like TRPO find their control policies through interaction with the environment, without requiring explicit knowledge of the underlying system dynamics. While this comes at the cost of an additional exploration phase, learning-based approaches typically offer an increased flexibility in that the same control architecture can adapt to different tasks and environments, without being affected by potential model mismatches. More importantly, considering the final learned policy from a computational perspective, the synthesis of the control signal involves no additional conceptual steps compared to an optimization-based approach.

While a typical experiment with 20 agents in our setup takes between four and six hours of training on a machine with ten cores (sampling trajectories in parallel), a forward pass through the trained neural network to compute the instantaneous control signal takes only about 1 ms, which enables an execution in real time. Furthermore, all control strategies learned through our framework are decentralized, which allows an arbitrary system size scaling in a real swarm network, where the required computations are naturally distributed over all agents. When learning new policies, the memory requirements scale $O(N(N-1))$ with the number of agents (assuming global observability) since we need to store the local views of all agents. However, decentralized execution after the policy is learned scales linearly in N per agent. An incremental online computation of the mean can be chosen if memory restrictions exist [44].

For comparison, the complexity of calculating Voronoi regions for the pursuit evasion policy scales $O(N \log N)$ with the number of agents [18]. Concerning the system sizes considered in our experiments, the resulting computation time of both policy types is in the same order of magnitude during task execution.

4.5. Conclusion

In this paper, we proposed the use of mean feature embeddings as state representations to overcome two major problems in deep reinforcement learning for swarms: the high and possibly changing dimensionality of information perceived by each agent. We introduced three different approaches to realize

such embeddings — two manually designed approaches based on histograms / radial basis functions and an end-to-end learned neural network feature representation. We evaluated the approaches on different variations of the rendezvous and pursuit evasion problem and compared their performances to that of a naive feature concatenation method and classical approaches found in the literature. Our evaluation revealed that learning embeddings end-to-end using neural network features scales well with increasing agent numbers, leads to better performing policies, and often results in faster convergence compared to all other approaches. As expected, the naive concatenation approach fails for larger system sizes.

5. Robust Black-Box Optimization for Stochastic Search and Episodic Reinforcement Learning

This chapter has been published in the Journal of Machine Learning Research [78].

Stochastic-Search algorithms [144] are problem independent algorithms well-suited for black-box optimization (BBO) [92] of an objective function. They only require function evaluations and are used when the objective function cannot be modeled analytically and no gradient information is available. This is often the case for real world problems such as robotics [29] where the objective function describes the outcome of a task, medical applications [156], or forensic identification [82].

Typically, these algorithms maintain a search distribution over the optimization variables of the objective function. Solution candidates are sampled, evaluated, and the parameters of the search distribution (e.g. mean and covariance for Gaussian search distributions) are then updated towards a more promising direction. This process is repeated until a satisfactory solution quality is found or a pre-defined budget of objective function evaluations is reached.

In this paper, we re-introduce Model-based Relative Entropy Stochastic Search (MORE) [2], a versatile, general purpose stochastic-search optimization algorithm. Using insights from reinforcement learning (RL) and information-theoretic trust-regions, it aims to update the parameters of a Gaussian search distribution in the direction of the natural gradient [87]. To this end, MORE uses compatible function approximation [149, 116] and learn a quadratic surrogate model of the objective function and bound the Kullback-Leibler (KL) divergence between subsequent search distributions. In its original formulation, a bound on the Kullback-Leibler divergence and a bound on the loss of entropy between the old and new search distribution additionally acts as an exploration-exploitation trade-off to prevent pre-mature convergence of the algorithm.

Most other successful stochastic search algorithms make use of rankings of objective function evaluations for updating the search distribution's parameters [68, 154, 128]. While the use of rankings is well studied for deterministic objective function evaluations, its effects in the case of stochastic objective function evaluations, which are, for example, common in episodic reinforcement learning, are less well understood. We analyze these ranking based algorithms for this stochastic evaluation case and uncover their limitations to solve such tasks. In contrast, MORE directly optimizes the expected objective function values without ranking transformations of the objective function evaluations and can therefore also be applied for such domains.

However, the original MORE approach suffers from (a) the need for conservative KL bounds to keep the covariance updates stable, (b), a fixed entropy decreasing schedule resulting in suboptimal exploration and slow convergence, and (c), it employs an unnecessary complicated and inaccurate model learning method. We aim to fix these issues by (a) splitting the KL divergence into separate updates for the mean and covariance with separate trust region bounds, (b) introducing an adaptive entropy schedule based on an evolution path, and (c) using ordinary least squares with appropriate data pre-processing techniques to simplify the model learning process. We call our new algorithm **C**oordinate-**A**scent MORE with **S**tep Size Adaptation or CAS-MORE for short.

We empirically evaluate our algorithm on simulated robotics tasks, as well as a set of benchmark optimization functions. While we are competitive with state-of-the-art algorithms such as CMA-ES on the benchmark optimization functions, the RL experiments clearly show the strength of MORE.

5.1. Related Work

In the following sections, we describe in more depth current state of the art algorithms from the field of black-box optimization and review common design choices and algorithmic procedures. In addition, we highlight current step- and episode-based reinforcement learning approaches along with their benefits and drawbacks and discuss trust-region based reinforcement learning algorithms.

5.1.1. Evolutionary Strategies and Black-Box Optimization

The MORE algorithm can be seen as an instance of Evolution Strategies [24] from the broader class of Evolutionary Algorithms. The usual procedure involves sampling from a search distribution $\pi(\boldsymbol{x};\theta)$ with parameters θ, evaluating the candidates $\boldsymbol{x}_k$ on an objective function $f(\boldsymbol{x})$, $f:\mathbb{R}^n \to \mathbb{R}$, $\boldsymbol{x} \mapsto f(\boldsymbol{x})$, and the goal is to either find an individual $\boldsymbol{x}^*$ that maximizes f or a distribution π^* that maximizes the optimization objective $J = \mathbb{E}_{\boldsymbol{x}\sim\pi}[f(\boldsymbol{x})]$.

5.1.1.1. Ranking-Based Algorithms

Instead of directly incorporating function values into the optimization process, many algorithms apply a ranking based transformation , i.e., a monotonous mapping of function values sorted by fitness to some pre-defined fixed values. Specifically, for a population $\{\boldsymbol{x}_{i:\lambda} \mid i = 1,\dots,\lambda\} = \{\boldsymbol{x}_i \mid i = 1,\dots,\lambda\}$ sorted by function values such that $f(\boldsymbol{x}_{1:\lambda}) \leq f(\boldsymbol{x}_{2:\lambda}) \leq \cdots \leq f(\boldsymbol{x}_{\lambda:\lambda})$, the function values are substituted by weight values $w_1 <, \dots, < w_\lambda$, before updating parameters. While this makes the optimization process invariant to certain transformations and adds to the robustness of algorithms, it also changes the optimization objective into $\theta^* = \arg\max_\theta \sum_i w_i \log \pi(\boldsymbol{x}_i;\theta)$. As we will show in our experiments, this change can have severe downsides in reinforcement learning problems. In non-deterministic problems, it leads to an over- or underestimation of a sample's performance and requires averaging over several sample evaluations in order to obtain a reliable estimate for the ranking [70, 75].

The cross-entropy method [128, 26, 10] is one of the simplest representatives of ranking based algorithms. It evolves the search distribution by only incorporating an elite set of samples into the next generation which can be seen as a rank-based update.

The Covariance Matrix Adaptation - Evolution Strategy (CMA-ES) [68] performs well established heuristics to update the mean and covariance matrix of a Gaussian search distribution as an interpolation of the weighted sample mean and covariance and the old mean and covariance. The weights of the samples are chosen according to the ranking of the samples. CMA-ES also introduces the evolution path, an exponentially smoothed cumulative sum of previous update steps, which acts as a momentum term, similar to gradient based optimizers [94, 129]. Building on the basic CMA-ES approach, many derivative algorithms exist, mainly aiming at improving sample efficiency. Notable extensions are those which include restarts with increasing population sizes [16], restarts that alternate between large and small population sizes [67], a version with decreasing step-sizes [99], or incorporating second order information into the update of the covariance matrix [17].

Another class of algorithms using rank-based fitness shaping are those from the family of Natural Evolution Strategies (NES) [154]. Instead of updating the search distribution parameters with heuristics, NES follow a sample-based search gradient based on the same objective as MORE. Yet, due to the use of rankings, the connection to the natural gradient of the original expected reward objective is lost. We describe the relation to MORE in more detail in the next section.

Contrary to these algorithms, MORE does not discard poorly performing candidates in its optimization process and only applies common data pre-processing techniques such as standardization before feeding them to the learning algorithm. There also exist variations of the CMA-ES using surrogate models such as [100] or [66]. Whereas these algorithms use the surrogate to generate approximate function values, which are then again used to generate a ranking, MORE directly uses the model parameters to update the search distribution parameters.

5.1.1.2. Natural Gradients for Black-Box Optimization

The natural gradient is often used in optimization as it has shown to be more effective when a parameter space has a certain underlying structure [9]. Important algorithms belong to the family of Natural Evolution Strategies (NES) [154]. They use a Taylor approximation of the KL-divergence between subsequent updates to estimate a search gradient in the direction of the natural gradient. Instead of information theoretic trust-regions on the update as used in MORE, they use either fixed [146, 51] or heuristically updated learning rates [154]. NES also uses ranking-based fitness shaping. The ranking is required to improve the robustness of the algorithm, yet, it also changes the objective and the search direction does not correspond to the natural gradient anymore. In contrast, MORE is inherently robust due to the used trust-regions and therefore, does not require a ranking-based transformation of the rewards. The ROCK$*$ algorithm presented in [81] uses the natural gradient on a global approximation of the objective generated with kernel regression.

5.1.1.3. Other Black-Box Optimization Approaches

There exists a wide variety of stochastic search algorithm classes for black-box optimization, each with their own benefits and drawbacks. Classic algorithms such as Nelder-Mead [109] use a simplex to find the minimum of a function. Bayesian optimization techniques such as Gaussian processes, for example, aim at finding global optima in low dimensional problem domains [113]. However, they suffer from high computation time and scale poorly with problem dimensionality and data points. Other directions are genetic algorithms [77] which are easy to implement but suffer from the need for good heuristics and random search [160, 124]. While both approaches can yield good results, they are not computationally efficient.

5.1.2. Reinforcement Learning

As MORE is based on the policy search objective, we also review some of the recent work from the domain of reinforcement learning. Here, the objective is based on the reward function defining the task to be solved. A detailed introduction to the problem domain can be found in Section 5.2.

5.1.2.1. Step-based Trust-Region Reinforcement Learning Algorithms

Updates bounded by a trust-region are a common approach in the reinforcement learning literature. One of the first deep reinforcement learning algorithms to successfully make use of a trust-region update is TRPO [137] which enforces a bound on an average KL divergence trust-region. Similar sample-based approximations of the trust-region are also employed by recent deep reinforcement learning techniques such as Maximum A-Posteriori Policy Optimization (MPO) [1]. Compatible function approximation for deep reinforcement learning has been explored in [116]. An improved way of enforcing trust-regions directly in the policy function by using differentiable trust-region layers (TRPLs) has been introduced in [115].

5.1.2.2. Episode-based Reinforcement Learning

Episode-based reinforcement learning algorithms make use of black-box optimization techniques to optimize the expected return of an entire trajectory. These algorithms excel when tasks are defined through sparse and non-Markovian rewards, whereas step-based approaches typically fail in these cases. Due to the use of sparse rewards, learned behaviors are more energy efficient and less sensitive to high action costs compared to policies learned by step-based reinforcement learning algorithms. On the downside, the improved performance often comes at a higher sample complexity [114].

A closely related method to MORE is the Relative Entropy Policy Search (REPS) for step-based reinforcement learning [121] and its episodic formulation (albeit derived in a contextual setting) in [91]. Both use samples to approximate the objective and the trust-regions which does not guarantee that the KL trust-region is enforced for the new parametric policy in practice. Layered direct policy search (LaDIPS) [43] extends MORE to the contextual setting using a linear mapping from the context to the mean. Additionally, the usage of natural gradients can be derived by using a second order approximation of the KL-divergence, resulting in the Fisher information matrix used in NES.

Recently, Otto et al. [114] extended TRPLs to the episodic case and use a policy gradient method for learning movement primitives that allow a non-linear mapping from a context vector to the motion primitive parameters. They show that other algorithms such as PPO [136] are unsuitable as they suffer from the higher dimensional action space induced by the motion primitives. In contrast, we focus on the non-contextual case where the policy is a Gaussian distribution with full covariance matrix. We show that in this case, the natural gradient updates from MORE clearly outperform the trust-region policy gradient updates from TRPL, indicating that extending the exact natural gradient updates from MORE to deep neural networks is an interesting direction for future work.

5.1.3. Broader Scope

The MORE algorithm finds application in a variety of fields, such as variational inference [12] or density estimation [22]. In the context of trajectory optimization, ideas from the MORE algorithm are explored in the MOTO algorithm [7].

5.2. Preliminaries

Before we introduce our new algorithm, we present in detail the problem setting and the original MORE algorithm. Additionally, we provide another view on the algorithm derived from compatible feature approximation and its relation to natural gradient optimization.

5.2.1. Problem Setting

The goal is to find a distribution $\pi(\boldsymbol{x};\theta)$, parameterized by θ, over variables $\boldsymbol{x} \in \mathbb{R}^n$ that maximizes [1] the expected value $\mathcal{F}(\boldsymbol{x}) = \mathbb{E}[F(\boldsymbol{x}, \xi(\omega))]$ of a noisy objective function F where $\xi(\omega)$ is a typically unknown noise process. In the episodic reinforcement learning case [38, 114], F describes the episodic return. There can be multiple sources of noise $\xi(\omega)$, such as noise in the dynamics or the state observations of the given system. We frame the stochastic search objective as maximizing the expected fitness function, i.e.,

$$\max_{\pi} J = \max_{\pi} \mathbb{E}_{\boldsymbol{x}\sim\pi}[\mathcal{F}(\boldsymbol{x})] \tag{5.1}$$

which is a well known objective in policy search [38]. We consider the black-box scenario where we only have access to the noisy function evaluations $f(\boldsymbol{x}) = F(\boldsymbol{x}, \zeta)$, $\zeta \sim \xi(\omega)$, of the objective function, i.e., the expected function values $\mathcal{F}(\boldsymbol{x})$ are unknown and no gradients of the objective function are available.

5.2.2. Model-Based Relative Entropy Stochastic Search

MORE [2] is an iterative stochastic search algorithm where the search distribution $\pi_t(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_t, \Sigma_t)$ in each iteration t is modelled as a multivariate Gaussian distribution. The parameters to be optimized are $\theta = \{\boldsymbol{\mu}, \Sigma\}$ and the optimization problem is given by

$$\begin{aligned} \underset{\pi}{\text{maximize}} \quad & \int_{\boldsymbol{x}} \pi(\boldsymbol{x})\mathcal{F}(\boldsymbol{x})\mathrm{d}\mathrm{x} \\ \text{subject to} \quad & \mathrm{KL}(\pi \parallel \pi_t) \leq \epsilon, \\ & H(\pi) \geq \beta, \\ & \int_{\boldsymbol{x}} \pi(\boldsymbol{x})\mathrm{d}\mathrm{x} = 1 \end{aligned} \tag{5.2}$$

where ϵ and β are hyper-parameters controlling the exploration-exploitation trade-off,

$$H(\pi) = \int_{\boldsymbol{x}} \pi(\boldsymbol{x}) \log \pi(\boldsymbol{x})\mathrm{d}\mathrm{x}$$

denotes the Shannon entropy of the search distribution, and

$$\mathrm{KL}(\pi \parallel \pi_t) = \int_{\boldsymbol{x}} \pi(\boldsymbol{x}) \log \frac{\pi(\boldsymbol{x})}{\pi_t(\boldsymbol{x})}\mathrm{d}\mathrm{x}$$

is the KL divergence between the current and new search distribution. MORE optimizes the objective from Equation (5.2) by substituting the expected objective value with a learned quadratic approximation

$$\mathcal{F}(\boldsymbol{x}) \approx \hat{\mathcal{F}}(\boldsymbol{x}) = -1/2\, \boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} + \boldsymbol{x}^{\mathrm{T}}\boldsymbol{a} + a_0. \tag{5.3}$$

[1] Equivalently, a cost can be minimized by maximizing the negative objective function.

Using a quadratic surrogate and a Gaussian search distribution, the optimal solution in each iteration would be to set the mean of the search distribution to the optimum of the surrogate and collapse the search distribution to a point estimate which would prevent it from further exploration. To control the exploration-exploitation trade-off, additional constraints on the updated search distribution need to be introduced. First, the KL-divergence between the current search distribution and the solution of the optimization problem is upper bounded which ensures that the mean and covariance only slowly change and the approximate model is not over-exploited. Furthermore, an additional lower bound on the entropy is introduced to prevent premature convergence.

Closed Form Updates The optimization problem can be solved in closed form using the method of Lagrangian multipliers. The dual function is given by

$$g(\eta, \omega) = \eta\epsilon - \omega\beta + (\eta + \omega) \log \left(\int \pi_t(\boldsymbol{x})^{\frac{\eta}{\eta+\omega}} \exp\left(\frac{\mathcal{F}(\boldsymbol{x})}{\eta + \omega}\right) \mathrm{dx} \right)$$

with Lagrangian multipliers η and ω. The new search distribution is then given in terms of the Lagrangian multipliers as

$$\pi(\boldsymbol{x}) \propto \pi_t(\boldsymbol{x})^{\frac{\eta}{\eta+\omega}} \exp\left(\frac{\mathcal{F}(\boldsymbol{x})}{\eta + \omega}\right).$$

Analytic Solution The use of a quadratic model is beneficial for two reasons. First, it is expressive enough to solve a wide range of problems while being computationally easy to obtain. Second, quadratic features are the compatible features of the Gaussian distribution which make the integrals tractable. As shown in Section 5.2.3, the use of quadratic features also allows us to obtain exact natural gradient updates. The optimization problem in terms of the natural parameters $\boldsymbol{m} = \Sigma^{-1}\boldsymbol{\mu}$ and $\Lambda = \Sigma^{-1}$ is solved by minimizing the Lagrangian dual function, which is given as

$$\begin{aligned} g(\eta, \omega) =& \eta\epsilon - \omega\beta + \frac{1}{2}\big(\omega k \log(2\pi) - \eta(\log|\Lambda_t^{-1}| + \boldsymbol{m}_t^{\mathrm{T}}\Lambda_t^{-1}\boldsymbol{m}_t) \\ &+ (\eta + \omega)(\log|\Lambda^{-1}| + \boldsymbol{m}^{\mathrm{T}}\Lambda^{-1}\boldsymbol{m})\big), \end{aligned}$$

and the optimization problem becomes

$$\begin{aligned} \underset{\eta,\,\omega}{\text{minimize}} \quad & g(\eta, \omega) \\ \text{subject to} \quad & \eta \geq 0, \\ & \omega \geq 0 \end{aligned}$$

with Lagrangian multipliers η and ω. The convex dual function can be optimized using a constrained non-linear optimization method such as L-BFGS [98] to obtain the optimal solutions η^* and ω^*. The update rules for the new search distribution based on the Lagrangian multipliers are given by

$$\Lambda = \frac{\eta^*\Lambda_t + \boldsymbol{A}}{\eta^* + \omega^*}, \qquad \boldsymbol{m} = \frac{\eta^*\boldsymbol{m}_t + \boldsymbol{a}}{\eta^* + \omega^*}.$$

Mean and covariance can be recovered by the inverse transformations and are given by $\boldsymbol{\mu} = \Lambda^{-1}\boldsymbol{m}$ and $\Sigma = \Lambda^{-1}$. We can now see that the new search distribution's parameters are an interpolation between the natural parameters of the old distribution and the parameters of the quadratic model from Equation (5.3).

5.2.2.1. Model Learning

The parameters of the quadratic model can be learned from the noisy evaluations $f(\boldsymbol{x})$ using linear regression with quadratic features. The original MORE algorithm first uses a probabilistic dimensionality reduction technique to project the problem parameters into a lower dimensional space. Afterwards, weighted Bayesian linear regression is used to solve for the model parameters in the reduced space. Lastly, the model parameters are projected back into the original space. This approach has several drawbacks, as it introduces additional hyper-parameters to the algorithm and involves a sample-based approach to integrate out the projection matrix which is very costly in terms of computation time. We will later on show that by using appropriate data pre-processing techniques, standard linear least squares is better suited to efficiently fit the quadratic models and also allows a direct connection of MORE to natural gradients.

5.2.2.2. Entropy Control

The entropy of the search distribution can be controlled with parameter β. The bound β is chosen such that entropy of the search distribution $H(\pi)$ decreases by a certain percentage until a minimum entropy H^0 is reached, i.e. $\beta = \gamma(H(\pi_t) - H^0) + H^0$. Alternatively, a simple alternative is to linearly decrease β in each iteration until the lower bound H^0 is reached, i.e. $\beta = \max(H(\pi_t) - \delta, H^0)$.

5.2.3. Relation to Natural Gradient

The natural gradient is an optimization technique that considers the geometry of the parameter space to improve the convergence speed and efficiency of training machine learning models, especially in situations where parameters are highly correlated [9]. The natural gradient is given by scaling the "vanilla" gradient

$$\nabla_\theta J = \mathbb{E}_{\boldsymbol{x}\sim\pi}[\nabla_\theta \log \pi(\boldsymbol{x}) \mathcal{F}(\boldsymbol{x})] \approx \sum_i \nabla_\theta \log \pi(\boldsymbol{x}_i) f(\boldsymbol{x}_i)$$

with the inverse of the Fisher information matrix (FIM) $\boldsymbol{F} = \mathbb{E}_{\boldsymbol{x}\sim\pi}[\nabla_\theta \log \pi(\boldsymbol{x}) \nabla_\theta \log \pi(\boldsymbol{x})^{\mathrm{T}}]$, i.e.,

$$\boldsymbol{g}_{\mathrm{NG}} = \boldsymbol{F}^{-1} \nabla_\theta J.$$

5.2.3.1. Approximating the Natural Gradient with Samples

A sample based approximation of the natural gradient is given as

$$\begin{aligned} \boldsymbol{g}_{\mathrm{NG}} &\approx \left(\sum_i \nabla_\theta \log \pi(\boldsymbol{x}_i) \nabla_\theta \log \pi(\boldsymbol{x}_i)^{\mathrm{T}} \right)^{-1} \sum_i \nabla_\theta \log \pi(\boldsymbol{x}_i) f(\boldsymbol{x}_i) \\ &= \left(\boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi} \right)^{-1} \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{y} \end{aligned} \tag{5.4}$$

with

$$\boldsymbol{\Phi} = \begin{bmatrix} \nabla_\theta \log \pi(\boldsymbol{x}_1) \\ \vdots \\ \nabla_\theta \log \pi(\boldsymbol{x}_N) \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}$$

where $y_i = f(\boldsymbol{x}_i),\ i = 1, \ldots, N$. Equation (5.4) resembles the least squares solution to a linear regression problem which is given by

$$\boldsymbol{w}^* = \arg\min_{\boldsymbol{w}} \sum_i \left(\nabla_\theta \log \pi(\boldsymbol{x}_i)^{\mathrm{T}} \boldsymbol{w} - y_i\right)^2 \tag{5.5}$$

using the function approximator $\nabla_\theta \log \pi(\boldsymbol{x})^{\mathrm{T}} \boldsymbol{w}$ where the features are given by the gradient of the log search distribution with respect to the parameters of π. In this case, the natural gradient is given by the least squares solution, i.e., $\boldsymbol{g}_{\mathrm{NG}} = \boldsymbol{w}^*$.

5.2.3.2. Compatible Function Approximation for Stochastic Search

Equation (5.5) is a special case of compatible function approximation. To see this, we will quickly review this concept. Originally defined in the context of policy gradients with function approximation for step-based reinforcement learning [149], the compatible function theorem states that the policy gradient with function approximation is exact under two conditions:

1. The gradient of the advantage function approximator is equal to the log gradients of the policy, i.e. $\nabla_w \hat{A}(s, a; w) = \nabla_\theta \log \pi(a \mid s; \theta)$
2. the parameters w of the advantage function minimize a mean-squared error.

Stochastic search can be seen as a state-less one-step RL problem where $A(s, a) = \mathcal{F}(\boldsymbol{x})$ and the policy $\pi(a \mid s; \theta)$ is the search distribution $\pi(\boldsymbol{x})$. For the gradient to be unbiased in the case of MORE where we optimize an approximated surrogate objective, we need to replace $\mathcal{F}(\boldsymbol{x})$ with a learned approximation $\hat{\mathcal{F}}$ that uses compatible features. For a Gaussian distribution with natural parameters $\boldsymbol{m} = \Sigma^{-1}\boldsymbol{\mu}$ and $\Lambda = \Sigma^{-1}$, these features are given by

$$\begin{aligned}
\nabla_{\boldsymbol{m}} \log \pi(\boldsymbol{x}) &= \boldsymbol{x}^{\mathrm{T}} + c_1, \\
\nabla_{\Lambda} \log \pi(\boldsymbol{x}) &= -\frac{1}{2}\boldsymbol{x}\boldsymbol{x}^{\mathrm{T}} + c_2.
\end{aligned}$$

Thus, the compatible features are given by

$$\begin{aligned}
\phi(\boldsymbol{x}) &= \left[\nabla_{\boldsymbol{m}} \log \pi(\boldsymbol{x}), \mathrm{vec}(\nabla_{\Lambda} \log \pi(\boldsymbol{x}))\right] \\
&= \left[1, \boldsymbol{x}, -\tfrac{1}{2}\,\mathrm{vec}(\boldsymbol{x}\boldsymbol{x}^{\mathrm{T}})\right].
\end{aligned}$$

Consequently, if the model is obtained by a linear least squares fit using quadratic features, the MORE algorithm is performing an unbiased update in the natural gradient direction where the learning rate is specified implicitly by the trust region ϵ. This is reflected in the update equations of MORE which, assuming an entropy bound $\beta \to -\infty$, are simply given by

$$\boldsymbol{m} = \boldsymbol{m}_t + \eta^{-1}\boldsymbol{a}, \quad \Lambda = \Lambda_t + \eta^{-1}\boldsymbol{A},$$

where $\boldsymbol{a}$ and $\boldsymbol{A}$ are the estimated parameters of the compatible function approximation. They directly correspond to the fitted model parameter $\boldsymbol{w}^*$ in Equation (5.5). In difference to other methods such as CMA-ES [6] or NES, where a relation to natural gradients has also been established, the natural gradient update of MORE is exact since the estimation of the quadratic model is unbiased in expectation and no ranking based reward transformation is used.

5.3. Improving the MORE Algorithm

In its original formulation, MORE has several drawbacks as already pointed out earlier. In this section, we will introduce a new version of MORE that aims at improving convergence speed and simplifying the model learning approach. We achieve this by (a) disentangling the trust regions for mean and covariance, (b) an adaptive entropy control mechanism, (c) a simplified but improved model learning approach based on standard linear least squares. We also propose a robust fitting of the surrogate to stabilize the model fitting process. We will refer to the new algorithm as **C**oordinate-**A**scent MORE with **S**tep Size Adaptation or CAS-MORE for short.

5.3.1. Disentangled Trust Regions

The trust-region radius ϵ controls the change of the distributions between subsequent updates and thus has an influence on the speed of convergence. While a large trust region should encourage larger steps of the mean, we observed in our experiments that the result is mainly a large change in the covariance matrix leading to instabilities and divergence of the algorithm, especially on problems where it is difficult to estimate the quadratic model.

Therefore, limiting the KL-divergence between the policies in subsequent iterations is a sub-optimal choice as we have no direct control whether the mean or covariance of the search distribution is updated more aggressively. To alleviate this problem, we propose to decouple the mean and covariance update by employing a block coordinate ascent strategy for the mean and the covariance matrix which allows for setting different bounds on each of the components. This approach results in independent updates for the mean and covariance as in CMA-ES, albeit more principled, and has already been explored successfully in other episodic [114] and step-based [3] reinforcement algorithms.

The optimization problems we will be solving are

$$\begin{aligned} \underset{\boldsymbol{\mu}}{\text{maximize}} \quad & \int_{\boldsymbol{x}} \pi(\boldsymbol{x}) \hat{\mathcal{F}}(\boldsymbol{x}) \mathrm{dx} \Big|_{\Sigma=\Sigma_t} \\ \text{subject to} \quad & \mathrm{KL}(\pi(\boldsymbol{x}) \parallel \pi_t(\boldsymbol{x}))|_{\Sigma=\Sigma_t} \leq \epsilon_\mu \end{aligned} \tag{5.6}$$

for the mean and

$$\begin{aligned} \underset{\Sigma}{\text{maximize}} \quad & \int_{\boldsymbol{x}} \pi(\boldsymbol{x}) \hat{\mathcal{F}}(\boldsymbol{x}) \mathrm{dx} \Big|_{\boldsymbol{\mu}=\boldsymbol{\mu}_t} \\ \text{subject to} \quad & \mathrm{KL}(\pi \parallel \pi_t)|_{\boldsymbol{\mu}=\boldsymbol{\mu}_t} \leq \epsilon_\Sigma \end{aligned} \tag{5.7}$$

for the covariance in each iteration. By choosing a small bound ϵ_Σ, we can drop the entropy constraint from the optimization as the entropy is not decreasing as quickly.

5.3.1.1. Updating the Mean

We start updating the mean by setting $\Sigma = \Sigma_t$ and introducing a bound ϵ_μ to limit the change of the mean displacement. The dual function to the problem in Equation (5.6) is given by

$$g_\mu(\lambda) = \lambda \epsilon_\mu + \frac{1}{2}\left(\boldsymbol{m}_\mu(\lambda)^\mathrm{T} \boldsymbol{M}_\mu(\lambda)^{-1} \boldsymbol{m}_\mu(\lambda) - \lambda \boldsymbol{m}_t^\mathrm{T} \boldsymbol{M}_t^{-1} \boldsymbol{m}_t \right)$$

where λ is a Lagrangian multiplier and

$$\boldsymbol{M}_\mu(\lambda) = \lambda\Sigma_t^{-1} + \boldsymbol{A}, \qquad \boldsymbol{m}_\mu(\lambda) = \lambda\Sigma_t^{-1}\boldsymbol{\mu}_t + \boldsymbol{a}.$$

A detailed derivation of the dual function can be found in Appendix B.1.1. We now solve the dual problem

$$\begin{aligned} \underset{\lambda}{\text{minimize}} \quad & g_\mu(\lambda) \\ \text{subject to} \quad & \lambda > 0 \end{aligned}$$

and obtain the new mean as

$$\boldsymbol{\mu}^* = \boldsymbol{M}_\mu(\lambda^*)^{-1}\boldsymbol{m}_\mu(\lambda^*)$$

where λ^* is the solution of the optimization problem which we find using a non-linear optimization algorithm [2].

5.3.1.2. Updating the Covariance Matrix

Next, we set $\boldsymbol{\mu} = \boldsymbol{\mu}_t$ and introduce a bound ϵ_Σ to constrain the change of the covariance. The dual function (see Appendix B.1.2 for a detailed derivation) for the problem in Equation (5.7) is given by

$$g_\Sigma(\nu) = \nu\epsilon_\Sigma + \frac{1}{2}\nu\Big(\log|\boldsymbol{\Lambda}(\nu)^{-1}| - \log|\boldsymbol{\Lambda}_t^{-1}|\Big)$$

with

$$\boldsymbol{\Lambda}(\nu) = \frac{\nu\Sigma_t^{-1} + \boldsymbol{A}}{\nu}$$

and we need to solve the following optimization problem

$$\begin{aligned} \underset{\nu,\,\omega}{\text{minimize}} \quad & g_\Sigma(\nu) \\ \text{subject to} \quad & \nu > 0 \end{aligned}$$

where ν is again a Lagrangian multiplier. The optimal solution Σ^* in terms of the solution ν^* can be found analogously and is given by

$$\Sigma^* = \boldsymbol{\Lambda}(\nu^*)^{-1}.$$

5.3.2. Entropy Control

The standard MORE algorithm is only able to decrease the entropy of the search distribution in each iteration. This might result in slow convergence in particularly if the search distribution is initialized with too small variances[3]. On the other hand, a too large covariance can also lead to slow convergence

[2] We used the implementation of L-BFGS-B [98] provided by the python package NLOpt [86].

[3] This is often necessary in case a large initial exploration is generating samples that are very costly or dangerous to evaluate, e.g., in robotics.

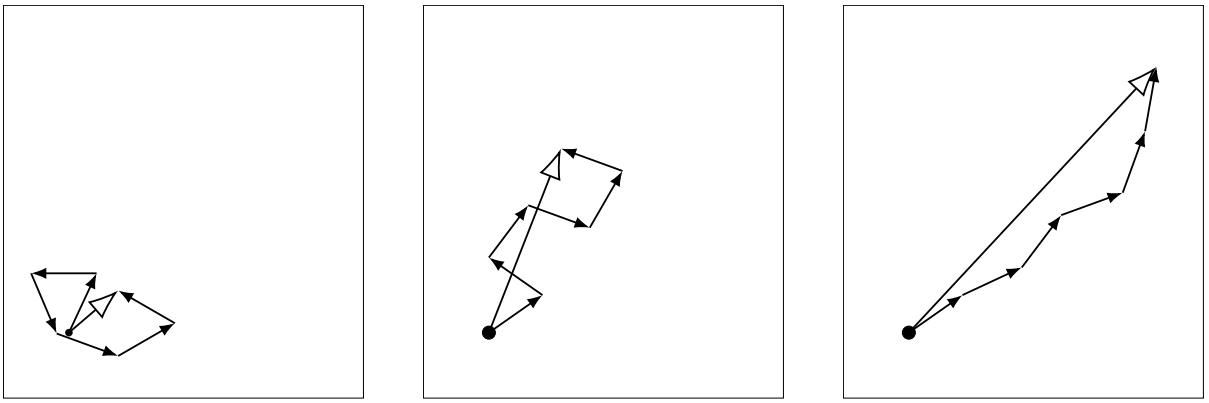


Figure 5.1.: This figure illustrates the concept of the evolution path in a 2D example and is adapted from Hansen [68]. In each figure, arrows with black heads correspond to an update of the mean. The evolution path is a smoothed sum over subsequent mean updates and depicted with an open head. In the left plot, the mean updates show no clear search direction and, as a result, the evolution path is short. The right plot shows the opposite where heavily correlated mean updates lead to a long evolution path. The center plot shows the desired case, where subsequent mean updates are uncorrelated.

as it takes a long time to settle in on a solution and simply increasing the bound on the covariance can lead to unstable updates.

To solve these issues, we take inspiration from CMA-ES which already makes use of an entropy control mechanism in form of the step-size update [68]. The crucial property underlying this update is the so-called evolution path which is a smoothed sum over previous mean updates. The idea of step-size adaptation is the following. Whenever we take steps in the same direction, we could essentially take a single, larger step, while dithering around a constant location indicates no clear search direction. In the first case, a larger entropy would allow for bigger steps, in the second case, we need to decrease entropy. We illustrate the concept in Figure 5.1. We can achieve this by scaling the covariance after the optimization with a scalar factor

$$\sigma_{t+1} = \exp\left(-\frac{\delta_{t+1}}{n}\right)$$

after the optimization steps where n is the problem dimensionality and δ_{t+1} is the desired change between the entropy in iteration t and the next iteration $t+1$ of the algorithm. The question is now how to determine a suitable value for δ_{t+1}.

The step size update we propose is inspired by the step size control mechanism used in CMA-ES called cumulative step size adaptation (CSA). We also use the evolution path $\boldsymbol{p}$ which is a vector tracking previous mean updates and compare its length to a hypothetical length of the evolution path resulting from uncorrelated mean updates. Therefore, the evolution path update needs to be constructed in a way that we can derive a computable desired length from it. Inspired by CMA-ES, we initialize the evolution path $\boldsymbol{p}_0$ to zero and update it as follows

$$\boldsymbol{p}_{t+1} = (1 - c_\sigma)\boldsymbol{p}_t + \sqrt{\frac{c_\sigma(2 - c_\sigma)}{2\epsilon_\mu}}\Sigma_t^{-\frac{1}{2}}(\boldsymbol{\mu}_{t+1} - \boldsymbol{\mu}_t)$$

where $c_\sigma < 1$ is a hyper parameter. The factors are chosen analogously to CMA-ES such that $(1 - c_\sigma)^2 + \sqrt{c_\sigma(2 - c_\sigma)}^2 = 1$. While multiplying the shift in means with the inverse square root of the covariance matrix in CMA-ES makes the evolution path roughly zero mean and unit variance distributed, we know

its exact length due to the constraint on the mean update in Equation (5.6) as long as the mean update is at its bound. The vector $\Sigma_t^{-\frac{1}{2}}(\boldsymbol{\mu}_{t+1} - \boldsymbol{\mu}_t)$ has in that case length $\sqrt{2\epsilon_\mu}$ and by scaling it with $(\sqrt{2\epsilon_\mu})^{-1}$ the resulting length is 1. We set

$$\delta_{t+1} = \alpha \left(1 - \frac{\|\boldsymbol{p}_{t+1}\|}{\|\boldsymbol{p}_{t+1}^{\text{des}}\|}\right),$$

where $\|\boldsymbol{p}_{t+1}^{\text{des}}\|$ is a desired length of the evolution path given by the length of a evolution path resulting from fully uncorrelated updates and α influences the magnitude of entropy change which was set to 1 in the black-box optimization experiments, and 0.1 in the episodic RL experiments. This rule for determining the change in entropy is almost identical to the one in CMA-ES. If the norm of the current evolution path $\boldsymbol{p}_{t+1}$ is smaller than $\boldsymbol{p}_{t+1}^{\text{des}}$, i.e., the mean updates were dithering around a constant location, we reduce entropy (i.e. $\delta_{t+1} > 0$) while entropy is increased if the length of the evolution path is longer than desired.

In order to determine the length of an uncorrelated path $\|\boldsymbol{p}_{t+1}^{\text{des}}\|$, we first look at the length of two vectors

$$\|\boldsymbol{a} + \boldsymbol{b}\| = \sqrt{\|\boldsymbol{a}\|^2 + \|\boldsymbol{b}\|^2 + 2\|\boldsymbol{a}\|\|\boldsymbol{b}\| \cos\theta},$$

where θ is the angle between vectors $\boldsymbol{a}$ and $\boldsymbol{b}$. If these two vectors are uncorrelated (i.e. $\theta = \pi/2$), the length becomes $\sqrt{\|\boldsymbol{a}\|^2 + \|\boldsymbol{b}\|^2}$. Under the assumption that the mean update is always at the bound, i.e. $(2\epsilon_\mu \Sigma_t)^{-\frac{1}{2}}(\boldsymbol{\mu}_{t+1} - \boldsymbol{\mu}_t) = 1$, the length of the evolution path p_{t+1} given the previous length p_t is then

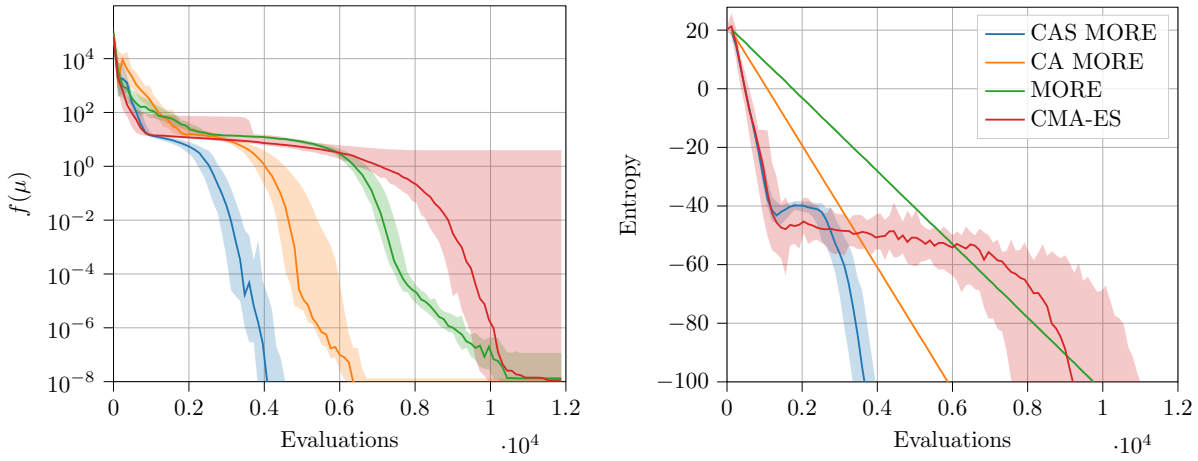
$$p_{t+1} = \sqrt{(1 - c_\sigma)^2 p_t^2 + c_\sigma(2 - c_\sigma)}.$$

In practice, we set the desired angle $\theta = \frac{3\pi}{8}$ and compute $\|\boldsymbol{p}_{t+1}^{\text{des}}\|$ accordingly to account for unavoidable correlations between iterations due to sample reuse and constrained updates. Finally, the adapted policy is given by

$$\pi_{t+1}^{\sigma} = \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_{t+1}, \sigma_{t+1}^2 \Sigma_{t+1}).$$

5.3.3. Illustrative Example

We demonstrate the characteristics of CAS-MORE by comparing it to the original formulation of MORE and Coordinate Ascent MORE without adaptive entropy control (CA-MORE). We therefore use a 15 dimensional Rosenbrock function as defined in Hansen et al. [73] and run each variant of MORE 20 times with different seeds. The optimization is stopped once a target function value of 1×10^{-8} is reached, or 12 000 function evaluations are exceeded. Figure 5.2 shows median and 5% / 95% quantiles of the function value at the mean (Figure 5.2a) and the entropy of the search distribution (Figure 5.2b) over the optimization. The hyper-parameters for the KL bounds (and entropy schedule for MORE) are chosen individually for each algorithm based on a grid search. We observe that decoupling the mean and covariance update already significantly improves convergence speed as it allows for a higher learning rate for the mean and therefore a quicker entropy reduction. Note, that reducing entropy even quicker leads to premature convergence while increasing ϵ for MORE leads to divergence of the algorithm. This problem is alleviated by introducing the adaptive entropy schedule that first quickly reduces entropy, then plateaus, and, finally, quickly reduces entropy again, resulting in an accelerated optimization process.



(a) Function value evaluated at μ_t

(b) Entropy of the search distribution

Figure 5.2.: This figure shows the function value (left) and the entropy of the search distribution (right) over the course of optimization of a 15-dimensional Rosenbrock function using different variants of MORE. The original MORE is shown in green, while coordinate ascent versions of more are orange and blue where CA-MORE indicates a fixed entropy reduction schedule and CAS-MORE indicates the adaptive entropy schedule using the step-size adaptation. For comparison, CMA-ES is plotted in red.

5.4. Model Learning

In this section, we introduce a simple but effective approach of learning a quadratic model based on polynomial ridge regression using the method of least squares as replacement for the more complex Bayesian dimensionality reduction technique introduced in the original paper [2]. We design the model learning with two objectives in mind. First, it should be data- and time-efficient to allow for quick execution of the algorithm. Additionally, it should be robust towards outliers and samples from very low entropy regimes. To this end, we apply a series of data pre-processing techniques, re-use old samples, and start learning a model with a lower complexity and increase it once sufficient data is available.

5.4.1. Least Squares Model Fitting

In each iteration, we generate a fixed number K of samples $\boldsymbol{x}_i \sim \pi$, $i = 1, \ldots, K$, evaluate them on the objective function to obtain $y_i = f(\boldsymbol{x}_i)$ and add the tuple $(\boldsymbol{x}_i, y_i)$ to a data-set $\mathcal{D} = \{(\boldsymbol{x}_q, y_q) \mid q = 1 \ldots Q\}$ of length Q[4]. The goal of the model fitting process is to find the parameters $\boldsymbol{A}$, $\boldsymbol{a}$, and a of a quadratic model of the form

$$\mathcal{F}(\boldsymbol{x}) \approx \hat{\mathcal{F}}(\boldsymbol{x}) = -1/2\, \boldsymbol{x}^{\mathrm{T}} \boldsymbol{A} \boldsymbol{x} + \boldsymbol{x}^{\mathrm{T}} \boldsymbol{a} + a.$$

These parameters can be found by solving a regularized least squares problem

$$\min_{\boldsymbol{w}} \|(\boldsymbol{y} - \boldsymbol{\Phi} \boldsymbol{w})\|_2^2 + \lambda \|\boldsymbol{w}\|_2^2,$$

[4] Once the number of data-points Q in $\mathcal{D}$ exceeds a maximum queue-size $Q_{\max}$, we discard old samples in a first-in, first-out manner.

where $\mathbf{\Phi}$ is the design matrix whose rows are given by a feature transformation $\phi(\boldsymbol{x})$, i.e $\mathbf{\Phi} = [\phi(\boldsymbol{x}_1), \ldots, \phi(\boldsymbol{x}_Q)]^{\mathrm{T}}$, $\boldsymbol{y}$ is a vector containing the fitness evaluations $y_i = f(\boldsymbol{x}_i)$ and λ is the regularization factor. The solution is given by the well known ridge regression estimator

$$\hat{\boldsymbol{w}} = (\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi} + \lambda \boldsymbol{I})^{-1}\mathbf{\Phi}^{\mathrm{T}}\boldsymbol{y}.$$

5.4.2. Adaptive Model Complexity

A full quadratic model has in the order of $O(n^2)$ parameters that need to be estimated. Compared to model-free algorithms, this can be a disadvantage if we initially need to sample enough parameters for the first model to be built. Instead, the complexity of the feature function $\phi(\boldsymbol{x})$ will be gradually increased from a linear to a diagonal and, finally, a full quadratic model depending on the number of obtained samples. The simplest model is a linear model where

$$a = w_1 \qquad \boldsymbol{a} = [w_2, \ldots, w_{n+1}]^{\mathrm{T}} \qquad \boldsymbol{A} = \mathbf{0}.$$

Next, we estimate a diagonal model where

$$\boldsymbol{A} = -\mathrm{diag}([w_{n+2}, \ldots, w_{2n+1}]).$$

Once sufficiently many data-points are available, we estimate a full quadratic model where

$$\boldsymbol{A} = -(\boldsymbol{L} + \boldsymbol{L}^{\mathrm{T}})$$

with

$$\boldsymbol{L} = \begin{bmatrix} w_{n+2} & 0 & \ldots & \ldots & 0 \\ w_{n+3} & w_{n+4} & 0 & \ldots & 0 \\ \vdots & & \ddots & & \\ w_{n(n+3)/2-1} & \ldots & \ldots & w_{n(n+3)/2-n-1} & 0 \\ w_{n(n+3)/2-n+2} & \ldots & \ldots & \ldots & w_{n(n+3)/2+1} \end{bmatrix}.$$

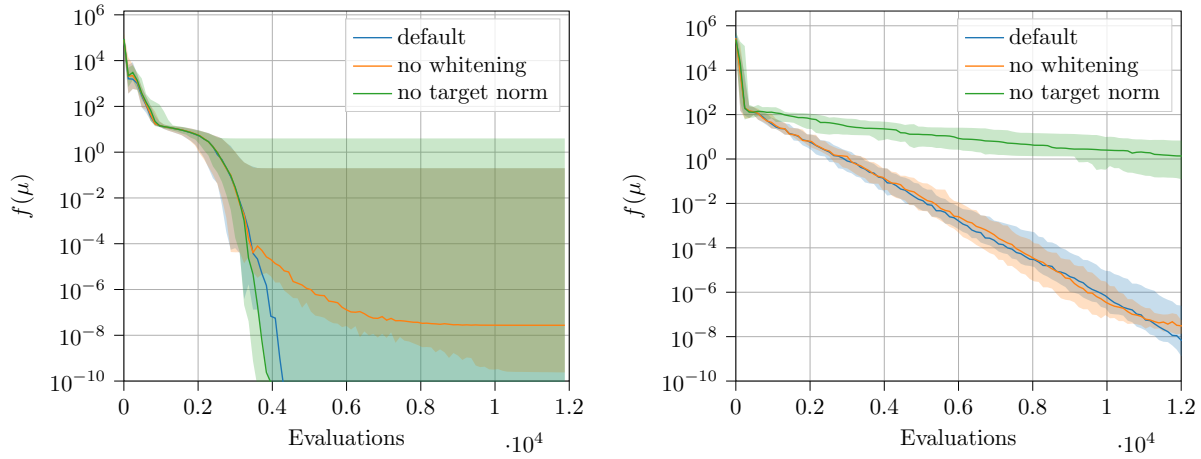
The feature functions for the models are given by

$$\begin{aligned} \phi_{\mathrm{lin}}(\boldsymbol{x}) &= [1, x_1, x_2, \ldots, x_n]^{\mathrm{T}} \\ \phi_{\mathrm{diag}}(\boldsymbol{x}) &= [\phi_{\mathrm{lin}}(\boldsymbol{x})^{\mathrm{T}}, x_1^2, x_2^2, \ldots, x_n^2]^{\mathrm{T}} \\ \phi_{\mathrm{full}}(\boldsymbol{x}) &= [\phi_{\mathrm{lin}}(\boldsymbol{x})^{\mathrm{T}}, x_1^2, x_1x_2, x_1x_3, \ldots, x_1x_n, x_2^2, x_2x_3, \ldots x_2x_n, x_3x_4, \ldots, x_n^2]^{\mathrm{T}}. \end{aligned}$$

We empirically found that we require at least 10% more samples than the model has parameters, i.e. start with a linear model once $|\mathcal{D}| \geq 1.1(1+n)$ and continue with a diagonal model once $|\mathcal{D}| \geq 1.1(1+2n)$. Finally, we switch to estimating a full quadratic model once $|\mathcal{D}| \geq 1.1(1+n(n+3)/2)$.

5.4.3. Data Pre-Processing

Estimating the model parameters can be numerically difficult with function values spanning several orders of magnitude, outliers, and very low variances of input and output values near the optimum. Therefore, we perform data pre-processing on the input values $\boldsymbol{x}$, as well as the design matrix $\mathbf{\Phi}$ and the target values y before solving the least squares problem. In particular, we perform whitening of the inputs $\boldsymbol{x}$, standardization of the design matrix $\mathbf{\Phi}$ and a special form of standardization of the target values y and construct the normalized data set $\mathcal{D}_w$. We demonstrate the effect of data pre-processing in Figure 5.3.



(a) Rosenbrock **(b)** Attractive Sector

Figure 5.3.: This figure shows the effects of data pre-processing on the optimization. We plot the function value over the course of optimization for a Rosenbrock (a) and Attractive Sector (b) function in 15 dimensions and turn off whitening and robust target normalization with clipping, respectively. We see that whitening becomes important for low entropy regimes near the optimum, while target normalization is especially important in case of outliers in the function values as can be seen for the Attractive Sector function.

5.4.3.1. Data Whitening

Before estimating $\hat{\boldsymbol{\beta}}$, we whiten the input data and obtain

$$\boldsymbol{x}_w = \bar{C}_{\mathcal{D}}^{-1}(\boldsymbol{x} - \bar{\boldsymbol{x}}_{\mathcal{D}}) \; \forall \; \boldsymbol{x} \in \mathcal{D}$$

where $\bar{\boldsymbol{x}}_{\mathcal{D}}$ is the empirical mean and $\bar{C}_{\mathcal{D}}$ is the Cholesky-factorization of the empirical covariance of the data-set. The parameters $\hat{\boldsymbol{\beta}}_w$ learned with the whitened data-set to be transformed back into unwhitened parameters. This transformation is given by

$$\begin{aligned} \boldsymbol{A} &= \bar{C}_{\mathcal{D}}^{-\mathrm{T}} \boldsymbol{A}_w \bar{C}_{\mathcal{D}}^{-1}, \\ \boldsymbol{a} &= \boldsymbol{A}\bar{\boldsymbol{x}}_{\mathcal{D}} + \bar{C}_{\mathcal{D}}^{-\mathrm{T}} \boldsymbol{a}_w, \\ a &= a_w + \bar{\boldsymbol{x}}_{\mathcal{D}}^{\mathrm{T}}(\boldsymbol{A}\bar{\boldsymbol{x}}_{\mathcal{D}} - \bar{C}_{\mathcal{D}}^{-\mathrm{T}} \boldsymbol{a}_w), \end{aligned}$$

where $\boldsymbol{A}_w$, $\boldsymbol{a}_w$ and a_w are the model parameters in the whitened space.

5.4.3.2. Target Normalization

Normalization of function values is beneficial as it stabilizes the model learning process. Depending on the number of samples we draw in each iteration as well as how well behaved the objective function is, we propose two target normalization methods. The first one allows for exact natural gradient updates, while the second one is more robust towards outliers.

Standard Target Normalization The first target normalization method is based on a simple standardization

$$y_w = (y - \bar{y}_{\mathcal{D}})/s_{\mathcal{D}} \; \forall \; y \in \mathcal{D}$$

of the target values where $\bar{y}_{\mathcal{D}}$ is the empirical mean and $s_{\mathcal{D}}$ is the empirical standard deviation of the target values in $\mathcal{D}$.

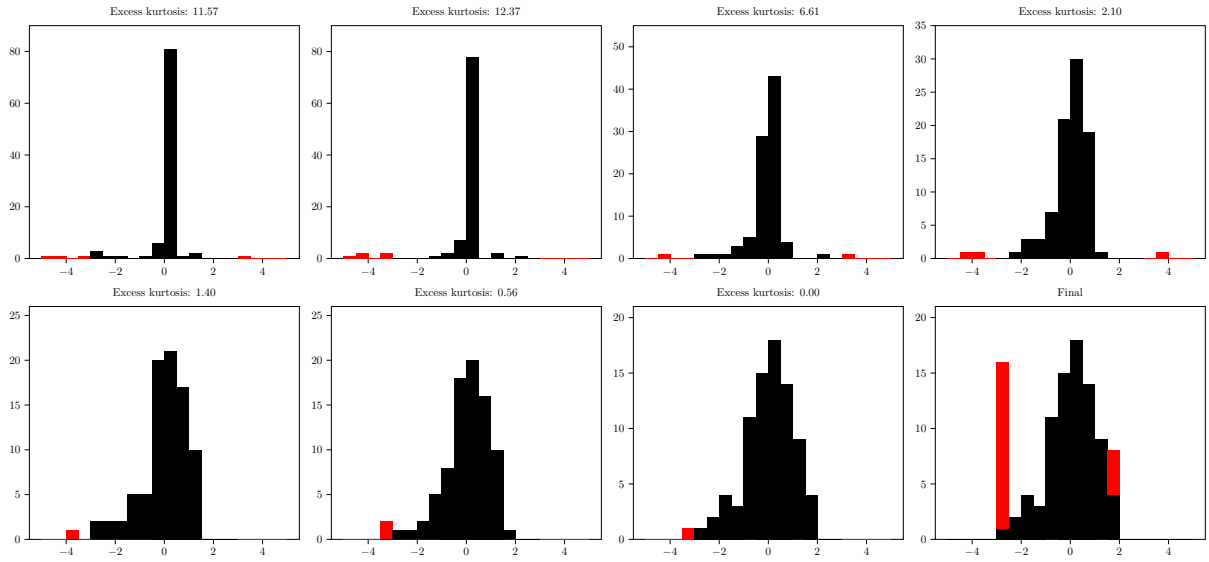


Figure 5.4.: This figure illustrates the robust target normalization scheme on 100 synthetic data points, generated by a reward function $y = -0.5\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}$. We sample $\boldsymbol{x}$ from a two dimensional multivariate normal distribution with zero mean and unit variance and pick 20 random y values and add Gaussian noise with a standard deviation of 10. Plots from top left to bottom right show histograms of the data after standardization of the input in each level of recursion of the procedure. The excess kurtosis is measured on the black data points in the interval (-3, 3) and only these data points are recursively treated again until the excess kurtosis of the clipped data is below the threshold of 0.55. Finally, the bottom right histogram shows the output with the red data points clipped to the minimum and maximum values of the remaining data points. Note, that in each plot the standardized data has zero mean and unit variance but the model quality suffers from the present outliers.

Robust Target Normalization Additionally, we propose an adaptive iterative normalization and clipping scheme based on the excess kurtosis of the target values resulting in a normalization process more robust towards outliers. Given the data-set in a particular iteration, we first perform standardization like above, then treat all values outside the interval $[-v_{\text{clip}}, v_{\text{clip}}]$ as outliers. Afterwards, we look at the excess kurtosis of the standardized targets in the interval $(-v_{\text{clip}}, v_{\text{clip}})$. If it is high, the data is still compacted to a small interval while outliers at the borders negatively influence the least squares optimization. In order to evenly spread the data, we repeat the procedure (normalization and clipping), until the excess kurtosis falls below a threshold or the standard deviation of the remaining values is close to 1. After the procedure, we replace the negative and positive outliers with the minimum and maximum value of the remaining targets, respectively. We illustrate the procedure in Figure 5.4, pseudo-code of the approach can be found in Appendix B.2. While this effectively changes the optimization objective, it is not as severe as replacing all function values with a ranking and, thus, the update direction still corresponds to an approximated natural gradient. An exact quantification of the error is subject to future work.

5.5. Experiments

In this section, we evaluate the performance of CAS-MORE in the black-box stochastic search and episodic reinforcement learning setting. While we use the former to highlight the applicability over a wide range of objective functions using as few samples as possible, it is the episodic reinforcement learning domain where the core strengths of the algorithm come into play. We start by looking at the performance in terms of sample efficiency on a set of black-box optimization benchmark functions

provided by Hansen et al. [71] and compare CAS-MORE to the original formulation of MORE, as well as competitors such as CMA-ES [68] and XNES [154]. Afterwards, we run our algorithm on a suite of different episodic reinforcement learning tasks from the domain of robotics where the fitness function evaluation is inherently noisy and we also care about the quality of the generated samples (e.g., the sample evaluations should not damage the robot) instead of plainly looking at the fitness evaluation at the mean of the search distribution. We compare against state of the art episodic and step-based reinforcement learning algorithms.

5.5.1. Black-Box Optimization Benchmarks

The COCO framework [71] provides continuous benchmark functions to compare the performance of optimizers on problems with different problem dimensions. In these deterministic problems, it is only important to find a good point estimate $\boldsymbol{x}^*$. We choose this benchmark to show that our algorithm is competitive with other black-box optimizers in using as few samples as possible. In order to avoid the center-bias problem [89], the optimum of the objective function is randomly shifted away from zero for each instance.

5.5.1.1. Experimental Setup

We evaluate MORE on the 24 functions of the Black-box Optimization Benchmark (BBOB) [64] suite in dimensions 2, 3, 5, 10, 20 and 40. We run the experiments without explicit optimization of the algorithm's hyper parameters for individual functions. Table B.1 shows a set of default parameters which we found to robustly perform well on a wide variety of benchmark functions in terms of the problem dimensionality n. Additionally, we allow restarts of the algorithm with a larger population size whenever the optimization process fails (see [16]) until a budget of $10\,000n$ function evaluations is exceeded or the final target of 1×10^{-8} is reached. Since MORE maximizes, we multiply the function by -1 in order to minimize the objective.

5.5.1.2. Black-box Optimization Benchmarks

Figure 5.5 shows runtime results aggregated over function groups and on single functions in dimension 20. Plotted is the percentage of targets reached within the interval 1×10^2 to 1×10^{-8} versus the log of objective function evaluations divided by the problem dimensionality. The further left a curve is, the quicker it reached a certain target. The first two rows correspond to combined results on separable, moderate, ill-conditioned, multi-modal and weakly structured multi-modal functions, as well as combined results from all 24 functions. The third and fourth row shows results on selected individual functions.

We first notice that CAS-MORE has a significant advantage over the original formulation of MORE on almost all functions. Furthermore, we can see that, especially on functions from the group of moderate and ill-conditioned functions such as Ellipsoid, Rosenbrock or Bent Cigar, CAS-MORE is able to improve state of the art results of competitors like CMA-ES and XNES. On other functions on the other hand, for example the Attractive Sector function, MORE has slower convergence. We found this shortcoming is mainly due to difficulties estimating a good model for this objective (see also Figure 5.3b). Other hard functions include multi-modal functions such as the Rastrigin function, where MORE often focuses on a local optimum. Here, the solution is to draw more samples in each iteration to improve the estimated model. For more results on the BBOB suite we refer to Appendix B.4.

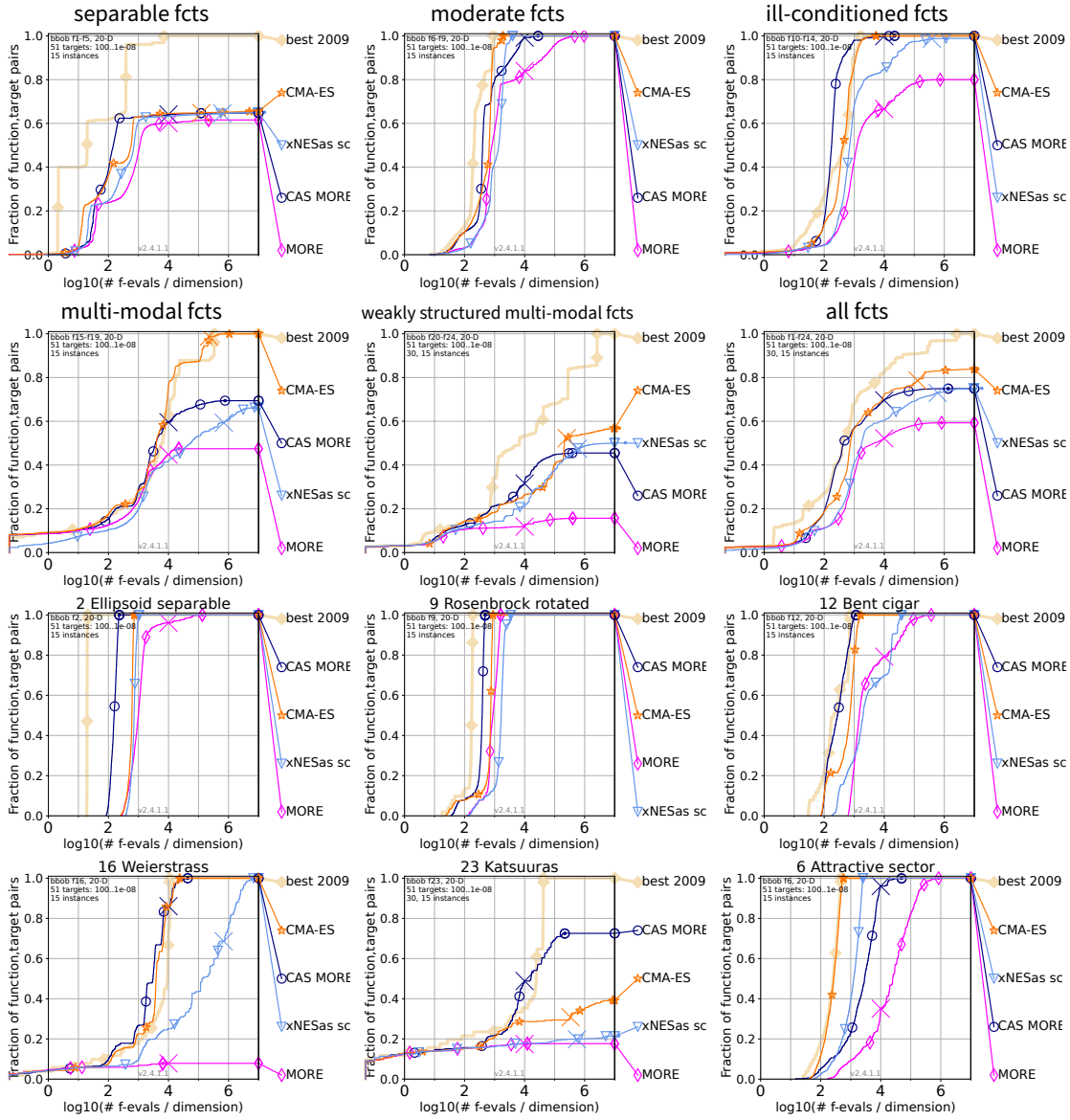


Figure 5.5.: This figure shows the bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for 51 targets with target precision in $10^{[-8..2]}$ in 20-D representing the percentage of targets achieved over the number of function evaluations. The results are averaged over 15 instances of each function. As reference algorithm, the best algorithm from BBOB 2009 is shown as light thick line with diamond markers. The first two rows show the aggregated results for all functions and subgroups of functions. Additionally, we show the results of individual functions in the third and fourth row. Big thin crosses indicate the used budget median for the respective algorithm which is 10 000 n function evaluations for each trial of CAS-MORE. Runtimes to the right of the cross are based on simulated restarts and are used to determine a runtime for unsuccessful runs [72].

Typically, these black-box optimization benchmarks also focus purely on the fitness evaluation of the mean of the distribution and disregard the quality of the samples. While CAS-MORE is comparable to state-of-the-art black-box optimizers such as CMA-ES in these domains, the real benefits of MORE appear if we consider problems with noisy fitness evaluations and whenever we care also about the quality of the generated samples. This is for example the case in robot reinforcement learning problems which are discussed in the next sub-section.

5.5.2. Episodic Reinforcement Learning Results

Our experiments aim to showcase the benefits of CAS-MORE over other black-box optimization techniques in the episodic RL setting, as well as deep RL algorithms for the step-based RL setting. To this end, we compare CAS-MORE to the original MORE algorithm that uses the dimensionality reduction model, an intermediate algorithm labeled as MORE-LS comprising of the same joint KL and entropy constraint as in MORE and the simplified model learning approach as in CAS-MORE, and CMA-ES, representing black-box stochastic search optimizers. Furthermore, we compare to policy gradient based episodic reinforcement learning using movement primitives with and without trust regions [114] labeled as BBRL-TRPL and BBRL-PPO, respectively. Note that these algorithms have been developed for contextual episodic RL, which allows to learn a deep neural network mapping from a context vector, which describes the task, to a motion primitive parameter vector, which describes the corresponding robot motion. In this work, we concentrate on the non-contextual case where we only have to learn a single Gaussian distribution. While this case is arguably simpler, we show that CAS-MORE significantly outperforms trust-region policy gradient based methods, showing the benefits of a closed form natural gradient update. Finally, we compare with contemporary step-based deep RL algorithms such as PPO, SAC, and TD-3. We run step-based PPO using approximately the same number of transitions as the episode-based algorithms. For SAC and TD-3, we are limited by a 24 hour compute budget and therefore report the performance after this budget is exceeded.

In our experiments, we analyze the performance of the mean the search distribution, as well as other performance indicators such as success rates. The experiments are designed to be non-contextual with unchanged starting configurations of the agent in a noisy environment. For the episodic RL experiments, we use Probabilistic Movement Primitives [119] to plan trajectories. A PD-controller provides the torques necessary to follow them. For the step-based deep RL algorithms, we provide proprioceptive feedback in form of the joint angles and joint velocities, as well as a normalized time-step to the agent. The policy directly outputs the torques applied to the joints. For the BBRL and deep RL experiments, we orient ourselves towards the hyper-parameters used in Otto et al. [114]. They are provided in Appendix B.3. For more careful exploration, we set $\epsilon_\mu = 0.05$ and $\epsilon_\Sigma = 0.005$ and draw 64 samples per iteration for the episodic RL algorithms. Since we do not operate using a very low number of new samples per iteration and the reward functions do not produce large outliers, we can use the standard target normalization allowing for more exact natural gradient updates. We report the results in terms of the interquartile mean (IQM) with a 95% stratified bootstrap confidence interval [5].

5.5.2.1. Hole Reacher

The first task is an advanced version of the holereaching task from Abdolmaleki et al. [2] that aims to show the difference between step-based and episodic reward formulations and the benefits of maximizing the expected fitness. The end-effector of a 5-link planar robot arm has to reach into a narrow hole without colliding with the ground or the walls of the hole. The arm is controlled in joint-space by applying torques to each of the 5 joints. Each link has a length of 1 m, the hole has a width of 20 cm, a depth of 1 m and is located 2 m away from the robot's base. For an illustration of a successful episode, see Figure 5.6. While the simulation of the arm is rather crude, this experiment has other properties that emphasize certain shortcomings of step-based RL, as well as rank-based optimization techniques. These properties include two distinct regions of exploration where above ground level, the agent can explore freely, while below the ground level even slight errors lead to a collision with the walls of the hole. Additionally, the depth of the hole, which is a main contributor to the reward function, is subject to noise in the noisy environment.

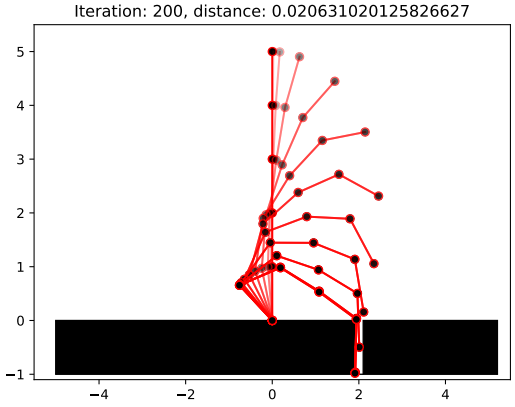


Figure 5.6.: This figure shows an illustration of the hole reacher task. The robot arms starts upright and smoothly reaches down the narrow hole. The policy is learned using CAS-MORE and shows a typical behavior that keeps a safe distance to the ground.

For the episodic RL setting, we learn the parameters of a ProMP with 3 basis functions, resulting in 15 parameters to be optimized. An episode consists of 200 time steps and in each iteration of the learning algorithm, we draw 64 new samples for the episodic RL algorithms. The cost function of the task is composed of two stages. First, the distance of the end-effector to the entrance of the hole is minimized, subsequently, the reward increases the further down the end-effector reaches. The reward is scaled down with a constant factor if the robot touches the ground. Additional penalty costs for acceleration ensure a smooth and energy-efficient trajectory. This reward function is different from the original MORE paper and is designed to highlight the risk awareness of the tested algorithms as the reward increases towards the bottom of the hole and then suddenly drops. The exact reward function can be found in Appendix B.5.1.

We examine 4 different settings of this experiment where we highlight the consequences of each choice on the learning process, namely

1. noise-free environment with dense in time reward,
2. noisy environment with dense in time reward,
3. noise-free environment with sparse in time reward,
4. noisy environment with sparse in time reward,

where in the noisy environment the perception of the depth of the hole is subject to noise[5]. The difficulty of this task lies in the two distinct sectors above and below the ground level. While above ground level, the agent can explore without negative consequences, it has to be much more careful once navigating into the narrow hole. The results of these experiments can be found in Figure 5.7. We plot the performance at the mean of the search distribution in left column, the expected performance under the search distribution in the center left column, the percentage of samples that collided with the ground in the center right column, and the distance of the end-effector to the bottom of the hole at the end of the episode in the right column. The result is averaged over 20 seeds for the episodic RL algorithms and 10 seeds for the step-based RL algorithms.

[5] In the beginning of the episode, we sample the depth of the hole uniformly from (1.00 ± 0.02) m.

Algorithm	Median (s)	Mean (s)	Standard Deviation (s)
BBRL PPO	0.23196	0.25007	0.03200
BBRL TRPL	0.42725	0.43615	0.11009
CAS MORE	0.01095	0.01117	0.00191
CMA-ES	0.00601	0.00613	0.00192
MORE	2.24552	2.20081	0.36460
MORE LS	0.01070	0.01462	0.08261

Table 5.1.: Runtime comparison for various episodic reinforcement learning algorithms. We measure the time the algorithm takes for one iteration without the sampling process.

Dense in time reward We first examine the results of the experiment in the noise-free environment with dense in time reward, the usual setting of current deep RL literature, in the first row of Figure 5.7. When looking at the second column, we can see that CAS-MORE optimizes towards a stable solution reaching into the ground that avoids collision with the ground. While CMA-ES finds a solution that comes closer to the bottom of the hole (right column) leading to a higher performance at the mean, it often draws samples that collide with the walls during the optimization as can be seen in the center left column. Compared to the original MORE algorithm, MORE-LS using a model based on our newly introduced model learning approach leads to a similar performance as CAS-MORE. BBRL-TRPL also performs similarly to CAS-MORE, yet, at the cost of much higher computation time. For a comparison of run times, see also Table 5.1. PPO optimizes towards fast movement of the end-effector in the direction of the entrance of the hole but fails to consistently reach into it. When looking at individual learning curves, we found that it often finds policies that successfully reach into the hole but soon after forgets this solution again. SAC, TD-3, BBRL-PPO, and also the original MORE with the dimensionality reduction model fail to find policies that reach into the ground in our experiments.

Next, we look at the results of the experiments in the noisy environment in the second row of Figure 5.7. Compared to the noise-free experiment, we can see the biggest impact on CMA-ES. CMA-ES again optimizes very fast towards the ground of the hole but due to the noise it quickly ends up in a solution that collides with the ground. CAS-MORE, MORE-LS and BBRL-TRPL on the other hand are hardly influenced by the noise in the environment and still provide solutions that reach into hole, albeit maintaining a slightly larger distance to the ground compared to the noise-less case. The results of the rest of the algorithms stay more or less unchanged as they never get close to the ground.

Sparse in time reward As noted in Otto et al. [114], a dense reward leads to a fast movement and overall poor energy efficiency. Thus, a better task description for this task is to only provide a distance-based reward in the last time-step and penalize high action values in each time-step. Again, we first examine the noise-free environment results which can be found in the third row of Figure 5.7. Even without noise, the results of CMA-ES become inconsistent over individual seeds, while CAS-MORE consistently optimizes towards well performing distributions. We assume this is the case as the distance between the initialization of the movement primitive parameters and a well performing solution is quite large. MORE-LS and BBRL-TRPL still find good solutions but perform slightly worse as can be seen from the distance of the end-effector to the ground in the last column. All step-based algorithms now fail to find good policies.

Last, we look at the results of the experiments in the noisy environment in the last row of Figure 5.7. We can now see the full potential of optimizing the expected reward and individual updates of the mean and covariance of the search distribution, as CAS-MORE finds the policies with the best performance

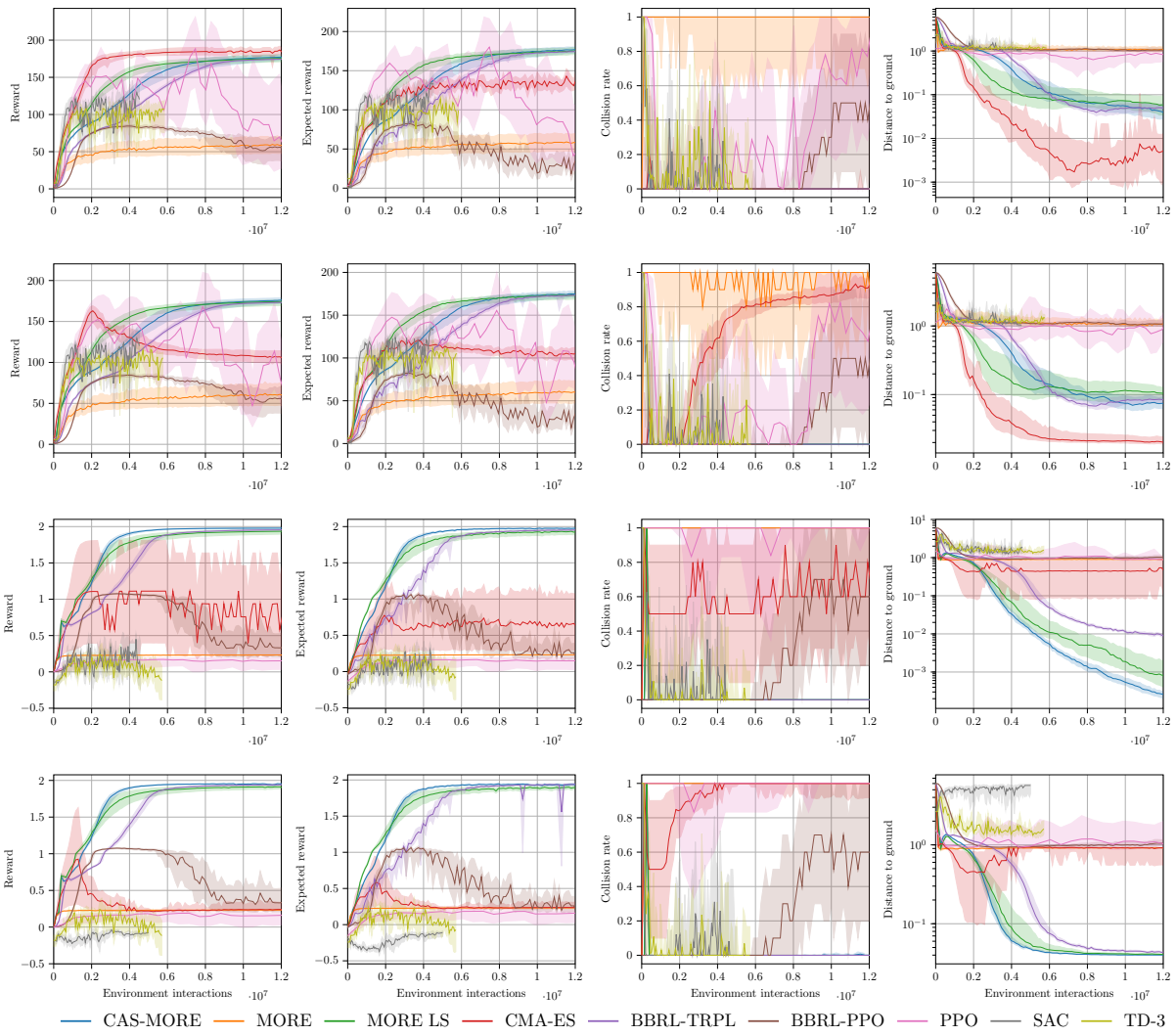


Figure 5.7.: This figure shows the results of the hole-reacher experiment. Plotted is the expected performance of the mean in the left column, the mean of episode returns of the samples drawn in each iteration in the center left column, the percentage of trajectories that led to collisions with the ground in the center right column, and the distance of the end-effector at the end of the trajectory in the right column. The first row contains the results for the noise-less experiment with dense step-based reward, while the second row shows the results of the noisy experiment with dense step-based reward. The third and fourth row show the results for the noiseless and noisy experiments using a sparse episodic reward, respectively.

at the mean as well as the best performing samples during the optimization process, with MORE-LS and BBRL-TPRL slightly below them. CMA-ES now consistently produces policies that collide with the ground. Results for all other algorithms remain poor.

5.5.2.2. Table Tennis

In the second task, we want to teach a 7 DoF Barrett WAM robotic arm a fore-hand smash. The goal is to return a table-tennis ball and place it as close as possible to the far edge of table on the opponent's side. For this and all following experiments, we concentrate on the noisy environment with sparse in time reward setting. Thus, in every episode, the ball is initialized at the same position but the velocity in x-direction (approaching the robot) is noisy. A robust strategy is to account for the uncertain velocity

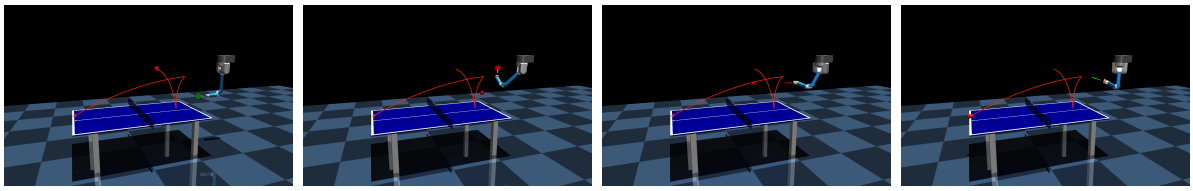


Figure 5.8.: A successful episode of the table tennis task learned by CAS-MORE.

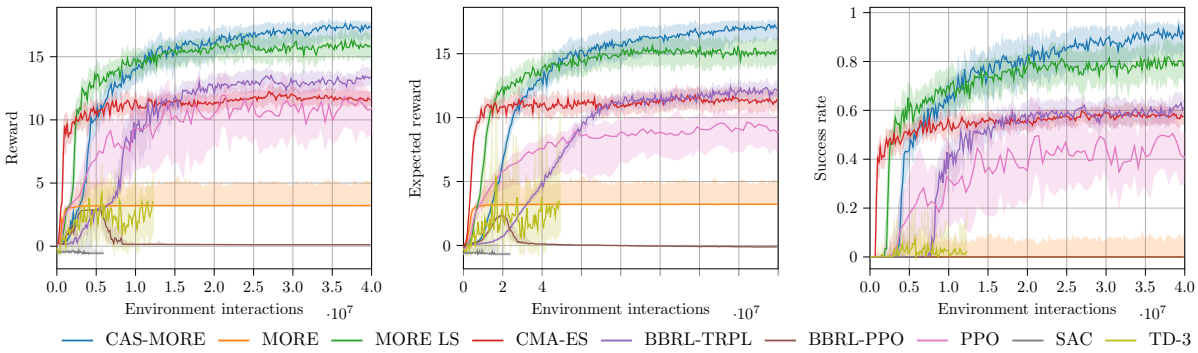


Figure 5.9.: This figure shows the results of the table tennis experiment. Plotted is the expected performance of the mean in the left column, the mean of episode returns of the samples drawn in each iteration in the center column, and the percentage of trajectories that led to ball landing points within the last 10cm of the table in the right column.

and aim for a spot that is not too close to the edge. An illustration of a successful execution of the task can be found in Figure 5.8.

We learn the parameters of a ProMP where we use 2 basis functions for each joint, resulting in 14 parameters to be optimized. The weights of the movement primitive are initialized as zero, so that the robot needs to learn the movement from scratch. The reward function is composed of three stages. First, the distance between the ball and the racket is minimized to enforce hitting of the ball. Second, a term is added to minimize the distance between the landing point of the ball and the far edge of the table. Once the ball lands on the table, the reward increases with the distance of the ball. Note, that different from the hole-reacher task, the reward for this task is non-Markovian as it depends on the whole trajectory [114]. In order to provide a step-based based reward for the step-based RL algorithms, we run the simulation until the racket touches the ball while only return the action cost. The rest of the episode is then simulated without agent interaction and presented to the learning algorithm as one last time-step with the task reward as in Otto et al. [114]. For more details on the environment, we refer the reader to Appendix B.5.2. The reward function differs from Otto et al. [114] to showcase the risk awareness of the optimization algorithms, as the reward drops as soon as the ball flies past the table.

The results of the experiment are presented in Figure 5.9. We can again see that CAS-MORE and MORE using the novel model learning approach produce the best results, this time with a distinct advantage over both, step-based RL algorithms PPO, TD3 and SAC, as well as gradient based episodic RL algorithms BBRL TRPL and BBRL PPO. Only CAS-MORE is able to consistently return around 90% of the balls into the target zone towards the edge of the table. CMA-ES and BBRL TRPL only produce distributions that are able to return around 60% of the balls.

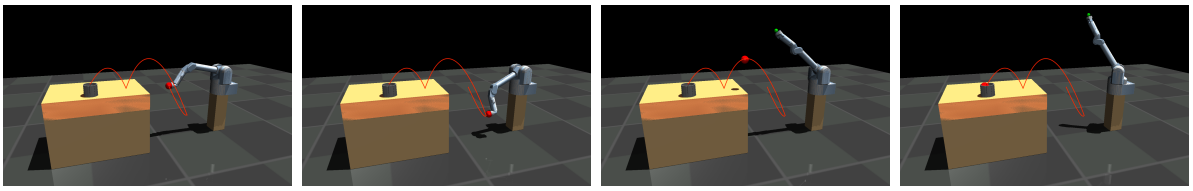


Figure 5.10.: A successful episode of the beerpong task learned by CAS-MORE.

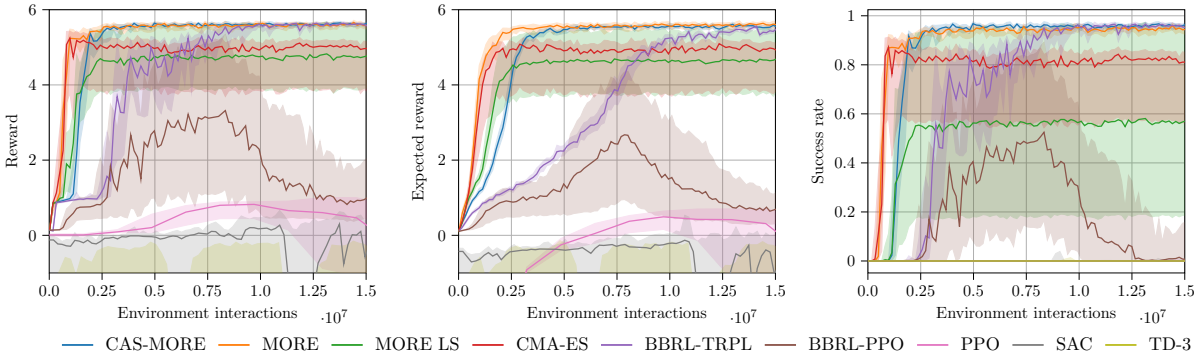


Figure 5.11.: This figure shows the results of the beerpong experiment. Plotted is the expected performance of the mean in the left column, the mean of episode returns of the samples drawn in each iteration in the center column, and the percentage of successful throws where the ball landed in the cup in the right column.

5.5.2.3. Beerpong

We use the same simulated robot as before to throw a ball towards a table where it first needs to bounce at least once and then fly into a cup. The task is depicted in in Figure 5.10. As the robot has no actuated end-effector and the ball is directly attached to the robot's last joint, we only need to actuate the first 6 joints, resulting in a 12 dimensional search space for the episodic RL algorithms. In this experiment, the ball release is noisy and designed in a way that it is not possible to find a conservative strategy. The noise will always cause some trials to miss the cup. The results can be found in Figure 5.11.

Overall, CAS-MORE again provides the best trade-off between convergence speed and task performance. Interestingly, the original MORE performs better than MORE using the least squares model while the step-based reinforcement learning tasks fail to find any good strategy. This experiment also shows the benefits of trust-region optimization, as BBRL-PPO starts to approach a good strategy but then fails due to, presumably, too aggressive policy updates.

5.5.2.4. Hopper Jump

The last task we consider is a modified version of the Hopper task from OpenAI gym [27] where the agent is rewarded for jumping onto a box. To this end, we place a box in front of the Hopper and use an updated reward function Appendix B.5.4. In the simulation, the initial distance to the agent and height of the box are subject to noise. For the episodic RL experiments, we choose 3 basis functions for each of the 3 degrees of freedom, resulting in 9 parameters to be optimized. Compared to the original version of the Hopper task, it is very difficult for step-based reinforcement learning algorithms, since a successful strategy requires broad exploration and involves tucking by bending the knee joint and an explosive move upwards to gain enough momentum to reach the top of the box. An illustration of this behavior can be found in Figure 5.12

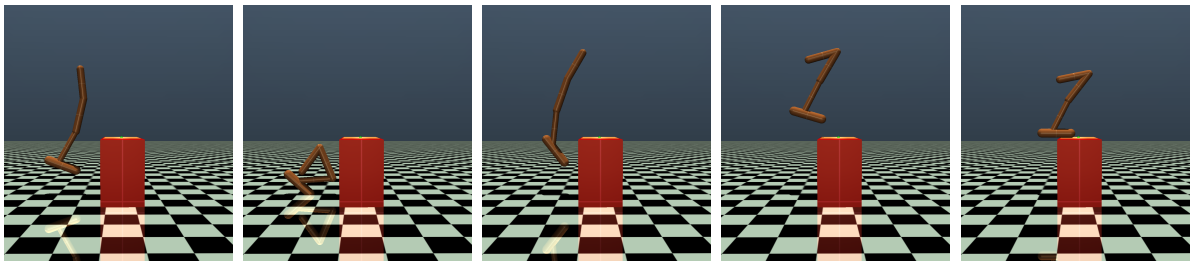


Figure 5.12.: A successful episode of the hopper jump task learned by CAS-MORE. At first, the agent fully bends to then release into a fully extended jump. This way, it can reach a safe height not risking contact with the box's edge.

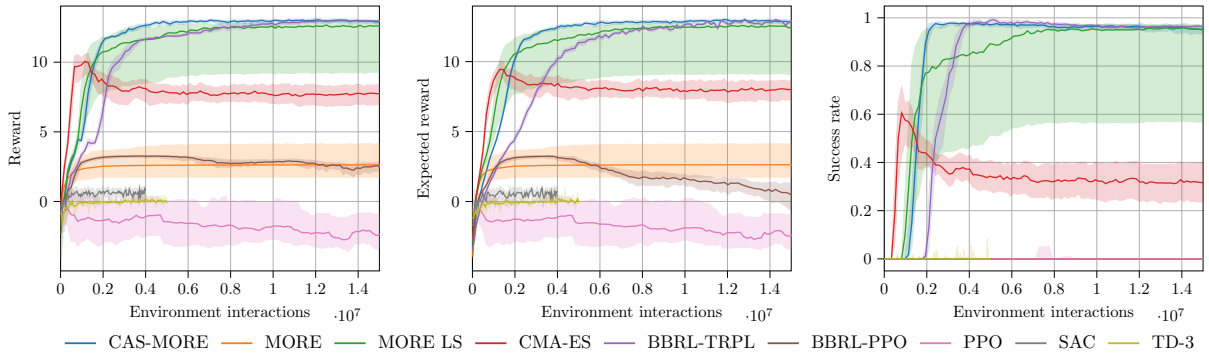


Figure 5.13.: This figure shows the results of the hopper jump experiment. Plotted is the expected performance of the mean in the left column, the mean of episode returns of the samples drawn in each iteration in the center column, and the percentage of successful throws where the ball landed in the cup in the right column.

The results are presented in Figure 5.13. We first observe, that only CAS-MORE, MORE LS and BBRL TRPL are able to solve the task. Again, CAS-MORE finds good solutions more robustly than MORE LS, as indicated by the confidence interval, and more efficient in terms of samples and in terms of computation time compared to BBRL TRPL. CMA-ES again quickly optimizes towards a solution but soon fails once the noise has a direct impact on the trajectory of the agent. All other algorithms hardly find policies that produce jumping strategies.

5.6. Conclusion

In this paper, we presented CAS-MORE, a new version of Model-based Relative Entropy Stochastic Search, based on a coordinate ascent strategy on the mean and covariance of the search distribution and an adaptive entropy schedule. Additionally, we provide an improved model fitting process that is computationally more efficient and show how to deal with objective functions that produce large outliers. We show how the parameter updates follow the direction of the natural gradient and, as the model estimation is not based on rankings of objective function values, we truly optimize the expected fitness under the search distribution. The result is a robust risk-aware optimization which we show to outperform state of the art algorithms such as CMA-ES on stochastic search task, as well as episodic and step-based reinforcement learning tasks, especially on noisy episodic reinforcement learning tasks.

Limitations and Future Work Arguably the biggest limitation of CAS-MORE is the restriction to the non-contextual setting. However, we believe that CAS-MORE still is a valuable tool in the prototyping stage of a more complex contextual problem, or if the problem at hand is non-contextual. Here, the

robustness and speed of the algorithm provide a large benefit over more complex algorithms. In addition, the algorithm can easily be extended for small context spaces with low-dimensional contexts using a linear mapping from the context to the mean.

5.7. Acknowledgements

The authors acknowledge support by the state of Baden-Württemberg through bwHPC. Research that lead to this work was funded by the Federal Ministry of Education and Research (BMBF) and the state of Hesse as part of the NHR Program.

6. Black-Box Optimization for Episode-Based Multi-Agent Reinforcement Learning

The concepts developed in Chapter 5 are the foundation for several interesting continuations. To some extent, these continuations have already been started. This chapter contains these fundamentals and serves as a starting point for future work.

6.1. Multi-Agent Stochastic Search

We consider an episodic multi-agent reinforcement learning setup where we want to optimize trajectories either in task- or in joint-space for n agents. Each individual agent i's trajectory is parameterized by a ProMP whose parameters $\boldsymbol{\theta}_i = \{\boldsymbol{\mu}_i, \Sigma_i\}$ we want to optimize. The agents are rewarded by a multi-variable reward function $f : \boldsymbol{X} \to \mathbb{R}$, $\boldsymbol{X} = \{(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \in \mathbb{R}^{n \dim(\mu)}\}$. This implies that agents are not rewarded individually. Instead, the collaborative effort of all agents is rewarded. The goal is to find a policy $\pi : \boldsymbol{X} \to \mathbb{R}$ that maximizes the expected reward $\mathbb{E}_{\boldsymbol{X} \sim \pi}[f(\boldsymbol{X})]$.

6.1.1. Factorized MORE

In single-agent RL, episodic RL using Probabilistic Movement Primitives (c.f. Chapter 5) provides a well working framework to learn parametric distributions over movement trajectories in task or joint space. Naïvely applying episodic RL to a multi-agent problem (i.e., treating all agents as one single agent), however, quickly becomes infeasible due to the increasing parameter space. In MORE, the main limitation for scaling is the estimation of the quadratic model as the number of parameters to be learned for a full model scales quadratically with the problem dimensionality and is in the order of $O((n\dim(\boldsymbol{\mu}))^2)$. To alleviate this problem, we propose to factorize the joint policy, effectively not modelling inter-agent correlations explicitly, and estimate a factorized quadratic surrogate based on the cooperative reward. It is equivalent to learning a block-diagonal Gaussian distribution over movement primitive parameters where each block corresponds to an individual agent. Still, we learn a joint quadratic model on the collaborative reward, which now is block-diagonal as well, and thus, the inter-agent correlations are modeled implicitly by the reward function surrogate. This choice reduces the overall parameter complexity to $O(n\dim(\boldsymbol{\mu})^2)$. Mathematically, this means that the objective of maximizing the expected reward can be rewritten as

$$\max_{\pi} \mathbb{E}_{X \sim \pi}[f(X)] = \max_{\pi_1, \ldots, \pi_n} \mathbb{E}_{\boldsymbol{x}_1 \sim \pi_1, \ldots, \boldsymbol{x}_n \sim \pi_n}[f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)]$$

and we optimize each function π_i individually.

6.1.1.1. Trust-Region Objective

We propose the following multi-variable optimization problem in n variables $\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_n$

$$\begin{aligned} \underset{\pi_1, \ldots, \pi_n}{\text{maximize}} \quad & \mathbb{E}_{\boldsymbol{x}_1 \sim \pi_1, \ldots, \boldsymbol{x}_n \sim \pi_n}[f(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)] \\ \text{subject to} \quad & \sum_{i=1}^{n} \mathrm{KL}(\pi_i \parallel \pi_{i,t}) \leq \epsilon \end{aligned}$$

where we employ the same coordinate-ascent strategy as proposed in Chapter 5, now iteratively optimizing each agent's policy individually in each step of the algorithm.

6.1.1.2. Model Learning

We learn a quadratic surrogate of the form

$$f(\boldsymbol{X}) \approx \hat{f}(\boldsymbol{X}) = -0.5 \sum_{i=1}^{n} \boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{A}_i \boldsymbol{x}_i + \boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{a}_i + a_0$$

using the method of least squares and employing the same pre-processing techniques as described in Chapter 5.

6.1.2. Policy Updates

The optimization problem is solved individually in each iteration for $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_n$ and $\Sigma_1, \ldots, \Sigma_n$ using the method of Lagrangian multipliers and an individual trust-region ϵ_μ for the means and ϵ_Σ for the covariances. The update for a mean $\boldsymbol{\mu}_i$ is given by

$$\boldsymbol{\mu}_i = (\lambda \Sigma_{i,t}^{-1} + \boldsymbol{A}_i)^{-1} (\lambda \Sigma_{i,t}^{-1} + \boldsymbol{a}_i)$$

and for a covariance Σ_i by

$$\Sigma_i = \nu (\nu \Sigma_{i,t}^{-1} + \boldsymbol{A}_i)^{-1}$$

where λ and ν are the solutions of the Lagrangian dual problem.

6.1.3. Illustrative Experiment

We demonstrate the applicability of the approach on a variant of the Hole Reacher problem introduced in Section 5.5.2.1. The new tasks involve several 5-link planar robots that need to reach into a hole, for example to pick an object that is too heavy for an individual robot to carry.

We again use 3 basis functions for each of the 5 joints, resulting in 15 parameters to be optimized for each arm. Table 6.1 shows the number of model parameters that need to be estimated depending on the number of agents. We can see that the dimensionality of the full quadratic model scales quadratically with the number of agents, while the block-diagonal model only scales linearly. The task reward function

$$R_{\text{task}} = \begin{cases} 0.25 \exp\left(-\frac{n_g \bar{d}_g)}{n}\right. & \text{if a collision happened,} \\ 0.25 + \exp\left(-\frac{n_g \bar{d}_g)}{n}\right. & \text{if } t = T \text{ and no collision happened} \end{cases}$$

is based on the average distance $\bar{d}_g$ of all the end-effectors to their corresponding goal points. The sparse-in-time reward only returns the task reward in the terminal time-step T which can either be the last time-step of the episode or the time-step where a collision happens and is given by

$$r_t = \begin{cases} -10^{-5}\tau_t & \text{if } t < T, \\ R_{\text{task}} - 10^{-5}\tau_t & \text{if } t = T. \end{cases}$$

Figure 6.1 shows an example of a successful episode of a policy learned with the block-diagonal quadratic model. Even though inter-agent correlations are not explicitly modelled, the arms coordinate to not collide on their way towards the bottom of the hole. Figure 6.2 shows an even higher dimensional example with 8 arms, resulting in 120 parameters to be optimized.

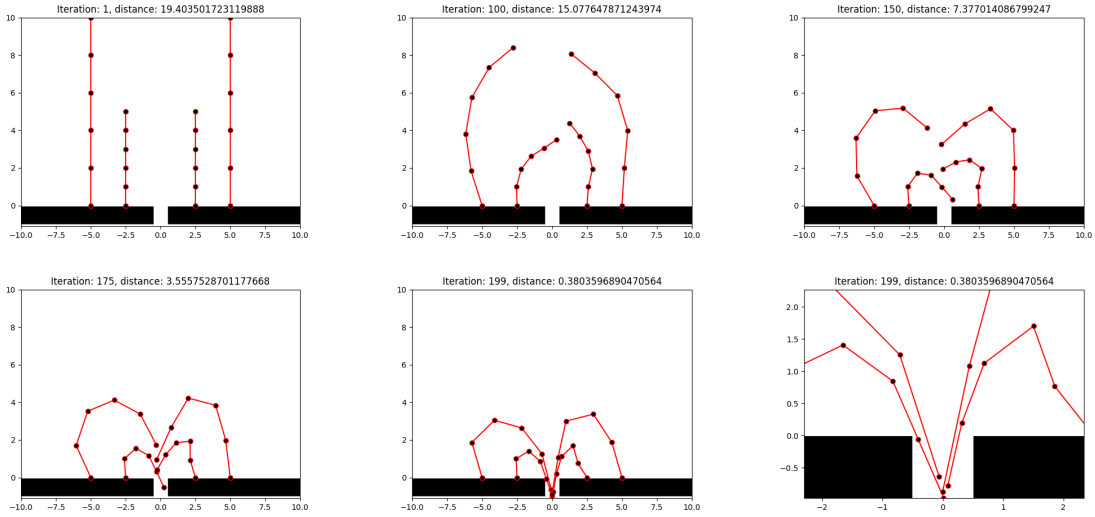


Figure 6.1.: An episode of four arms reaching towards a single hole.

We run an experiment to test the effectiveness of the block diagonal model and compare it to CAS-MORE with a full quadratic model, both using more samples than parameters to estimate the model. Additionally, we run CAS-MORE with the full quadratic model using the same number samples as the block-diagonal version. The learning curves in Figure 6.3 indicate that the factorized variant of CAS-MORE is able to achieve similar performance to the full version with a much lower sample count. Additionally, it performs favorable in comparison to CAS-MORE with a full quadratic model that uses a low number of samples.

n	$\dim(\boldsymbol{\mu})$	lin	diag	block	full
4	60	61	121	541	1891
8	120	121	241	1081	7381
16	240	251	481	2161	29161

Table 6.1.: Number of model parameters depending on the number of agents n for a 15-dimensional movement primitive per agent.

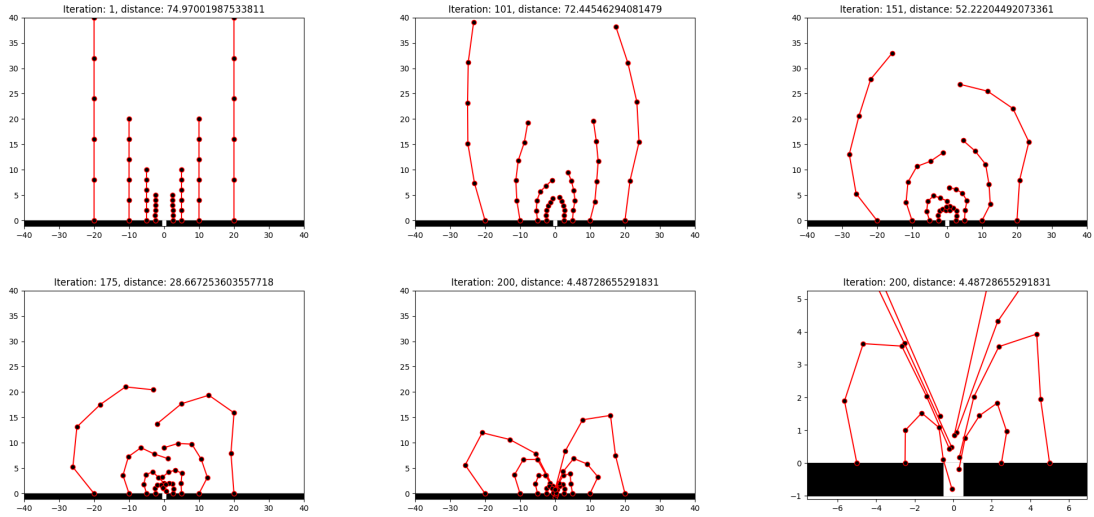


Figure 6.2.: An episode of eight arms reaching towards a single hole.

6.2. Permutation Invariant Learning of Versatile Multi-Agent Behavior

The episodic formulation additionally lends itself naturally for learning versatile behavior in the form of Gaussian mixture models. In principle, versatile behavior can be achieved by running multiple instances of MORE from various starting values. A drawback of this approach is that one may end up with many of them learning the same solution. Instead, we want to learn a Gaussian mixture model (GMM) $q(\boldsymbol{x}) = \sum_o q(\boldsymbol{x} \mid o)q(o)$ where each component o ideally learns a different solution. To this end, we can resort to Variational Inference by Policy Search (VIPS) [13], an algorithm for learning GMMs, typically applied in variational inference. It is based on a maximum entropy objective and for each component we optimize

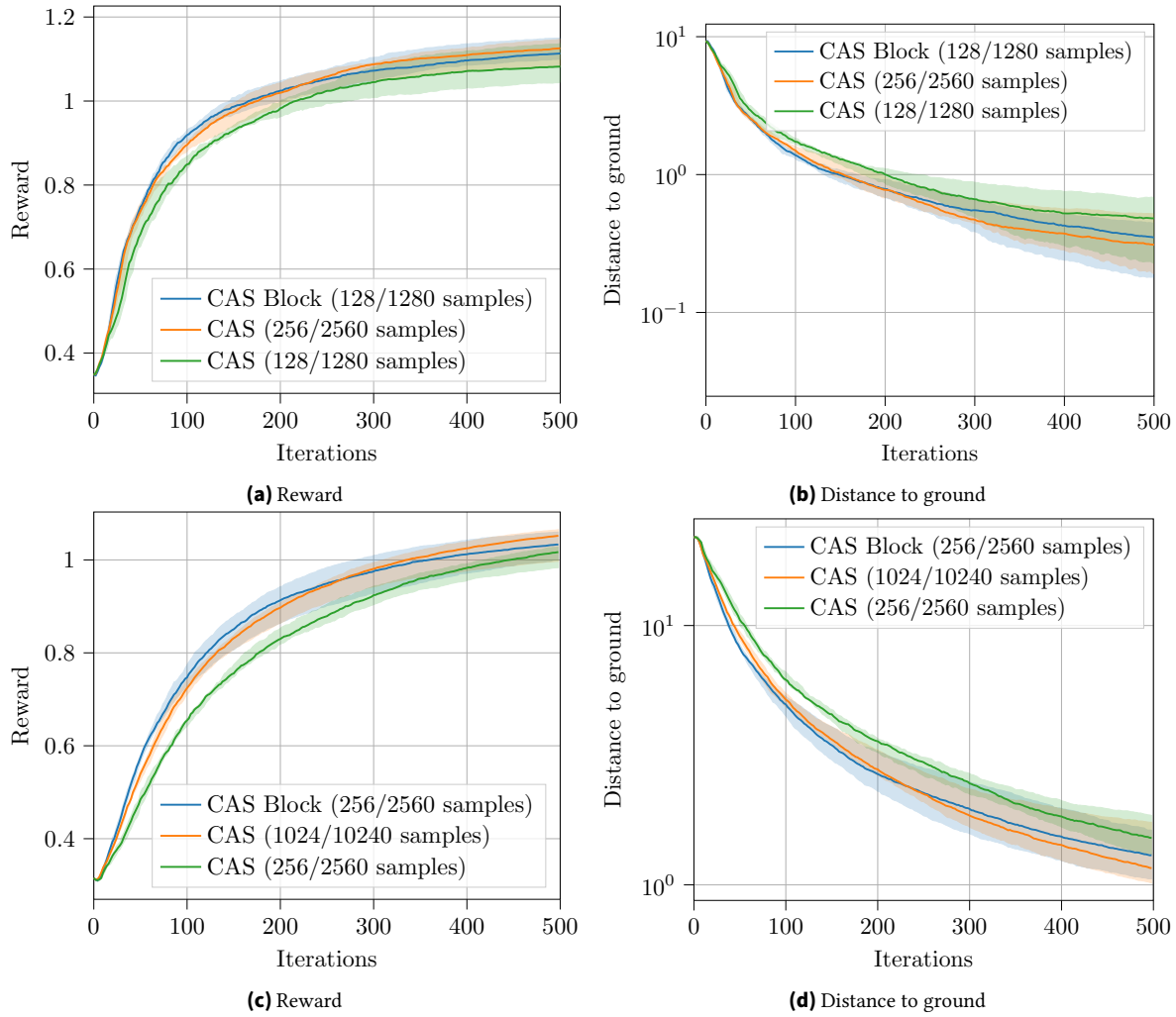
$$\begin{aligned} \underset{\pi_o}{\text{maximize}} \quad & \int_{\boldsymbol{x}} \pi_o f_o(\boldsymbol{x}) \mathrm{d}\mathrm{x} + \rho H(\pi_o) \\ \text{subject to} \quad & \mathrm{KL}(\pi_o \parallel \pi_{o,t}) \leq \epsilon \end{aligned} \tag{6.1}$$

where $\pi_o = q(\boldsymbol{x} \mid o)$ and $\pi_{o,t} = q_t(\boldsymbol{x} \mid o)$ is the component from the previous iteration. The component specific reward function $f_o(\boldsymbol{x}) = f(\boldsymbol{x}) + \rho(\log q_t(\boldsymbol{x} \mid o) - \log q_t(\boldsymbol{x}))$ contains the original reward value but, additionally, penalizes regions where other components already have high probability mass. We leave out the the update for the component weights as we are only interested in the policies themselves.

6.2.1. Maximum Entropy Episodic Policy Search

While originally not in a maximum entropy formulation, the algorithm from Chapter 5 can easily be augmented by reformulating the optimization problem

$$\begin{aligned} \underset{\pi}{\text{maximize}} \quad & \int_{\boldsymbol{x}} \pi(\boldsymbol{x}) f(\boldsymbol{x}) \mathrm{d}\mathrm{x} + \rho H(\pi) \\ \text{subject to} \quad & \mathrm{KL}(\pi(\boldsymbol{x}) \parallel \pi_t(\boldsymbol{x})) \leq \epsilon \end{aligned} \tag{6.2}$$



(a) Reward **(b)** Distance to ground

(c) Reward **(d)** Distance to ground

Figure 6.3.: This figure shows learning curves (left) and achieved average end-effector distances to the ground (right) over the optimization iterations for the four arm (upper row) and eight arm hole reacher. The first number in the legend indicates the number of samples drawn per iteration and the second number indicates the size of the reward buffer. Results are averaged over 5 seeds.

to contain an entropy bonus in the objective. The update equations are given by

$$\boldsymbol{\mu} = (\lambda\Sigma_t^{-1} + \boldsymbol{A})^{-1}(\lambda\Sigma_t^{-1}\boldsymbol{\mu}_t + \boldsymbol{a}) \qquad \Sigma = \left(\frac{\nu\Sigma_t^{-1} + \boldsymbol{A}}{\nu + \rho}\right)^{-1}$$

for the mean and covariance where λ and ν are Lagrangian multipliers found by solving the dual of the optimization problem. In contrast to Chapter 5 we omit the step-size adaptation in favor of the maximum entropy formulation to be able to use it in the framework of VIPS presented in the next section.

6.2.2. Permutation Invariant VIPS

Learning versatile solutions with multiple agents is of high complexity as, at least from the perspective of the learning algorithm, permutations of agents look like different solutions, albeit showing the same overall behavior. More precisely, each additional agent leads to factorially many permutations

of unique solutions to a problem. In this section, we sketch how to avoid learning permutations of solutions by extending the optimization problem introduced in Equation (6.1). To this end, we propose a permutation invariant form of a Gaussian mixture model where each component is mapped to its permutations with equal weight, simulating a larger GMM with factorially more components. We show that the same decomposition as in VIPS can be applied on this extended permutation invariant mixture model.

6.2.2.1. Permutation Invariant GMMs

In order to separate the space of unique solutions from the space of solutions that include permutations, we introduce a latent variable model

$$q(\boldsymbol{X} \mid o) = \int_Z q(\boldsymbol{X} \mid \boldsymbol{Z}) q(\boldsymbol{Z} \mid o) d\boldsymbol{Z}$$

over latent variable $\boldsymbol{Z} \in \mathbb{R}^{\dim(\boldsymbol{X})}$. The auxiliary distribution $q(\boldsymbol{X} \mid \boldsymbol{Z})$ acts as a mapping from the space of a unique solution to all permutations of it and can be defined as

$$q(\boldsymbol{X} \mid \boldsymbol{Z}) = \begin{cases} 1/|S(\boldsymbol{X})| & \text{if } \boldsymbol{Z} \in S(\boldsymbol{X}) \\ 0 & \text{else} \end{cases} \tag{6.3}$$

where $S(\boldsymbol{X})$ is the set of all legal permutations of $\boldsymbol{X}$. The mixture model is then defined as

$$\begin{aligned} q_\pi(\boldsymbol{X}) &= \int_Z q(\boldsymbol{X} \mid \boldsymbol{Z}) q_z(\boldsymbol{Z}) \mathrm{d}\boldsymbol{z} \\ &= \sum_o q(o) \int_Z q(\boldsymbol{X} \mid \boldsymbol{Z}) q_z(\boldsymbol{Z} \mid o) \mathrm{d}\boldsymbol{z} \end{aligned}$$

where $q_z(\boldsymbol{Z}) = \sum_o q(o) q(\boldsymbol{Z} \mid o)$ now is a mixture of block-diagonal GMMs ($q(\boldsymbol{Z} \mid o)$ corresponding to $\pi(\boldsymbol{X})$) in the space of unique solutions to the multi-agent problem. [1] Using the definition of the mapping in Equation (6.3), the distribution

$$q_\pi(\boldsymbol{X}) = |S(\boldsymbol{X})|^{-1} \sum_o q(o) \sum_{\tilde{\boldsymbol{X}} \in S(\boldsymbol{X})} q(\boldsymbol{Z} = \tilde{\boldsymbol{X}} \mid o)$$

assigns equal probability to all samples $\tilde{\boldsymbol{X}} \in S(\boldsymbol{X})$ while only requiring a single component in $\boldsymbol{Z}$ to represent all the permuted modes.

6.2.2.2. Objective

We are interested in solving

$$\underset{q_\pi}{\text{maximize}} \quad \int_{\boldsymbol{x}} q_\pi(\boldsymbol{X}) f(\boldsymbol{X}) \mathrm{dx} + \rho H(q_\pi)$$

[1] We use subscripts π and z to denote whether a distribution is defined in the space of $\boldsymbol{X}$ or $\boldsymbol{Z}$ to avoid confusion.

with the latent variable model introduced in the previous section by applying the decomposition proposed in Arenz, Zhong, and Neumann [13]. Using the identities

$$\log q(\boldsymbol{X}) = \log q(\boldsymbol{X} \mid \boldsymbol{Z}) + \log q(\boldsymbol{Z}) - \log q(\boldsymbol{Z} \mid \boldsymbol{X}),$$
$$\log q(\boldsymbol{Z}) = \log q(\boldsymbol{Z} \mid o) + \log q(o) - \log q(o \mid \boldsymbol{Z})$$

we can reformulate the objective as

$$\begin{aligned} L_\pi &= \int_{\boldsymbol{X}} q_\pi(\boldsymbol{X})(f(\boldsymbol{X}) - \rho \log(q_\pi(\boldsymbol{X})))\mathrm{d}\boldsymbol{X} \\ &= \sum_o q(o) \int_{\boldsymbol{X}} \int_{\boldsymbol{Z}} q(\boldsymbol{X} \mid \boldsymbol{Z}) q(\boldsymbol{Z} \mid o) \Big(\\ &\quad f(\boldsymbol{X}) - \rho(\log q(\boldsymbol{X} \mid \boldsymbol{Z}) - q(o \mid \boldsymbol{Z}) - \log q(\boldsymbol{Z} \mid \boldsymbol{X})) \Big) \mathrm{d}\boldsymbol{Z}\mathrm{d}\boldsymbol{X} \\ &\quad + \rho(H(q(\boldsymbol{Z} \mid o)) + H(q(o))) \end{aligned}$$

The occurrences of the log responsibilites $q(\boldsymbol{Z} \mid \boldsymbol{X})$ and $q(o \mid \boldsymbol{Z})$ prevent us from optimizing each component independently. However, by adding and subtracting the auxiliary distributions $\tilde{q}(\boldsymbol{Z} \mid \boldsymbol{X})$ and $\tilde{q}(o \mid \boldsymbol{Z})$, we obtain

$$\begin{aligned} L_\pi = \tilde{L}(&\tilde{q}(o \mid \boldsymbol{Z}) \\ &+ \rho \int_{\boldsymbol{X}} q(\boldsymbol{X}) \mathrm{KL}(q(\boldsymbol{Z} \mid \boldsymbol{X}) \parallel \tilde{q}(\boldsymbol{Z} \mid \boldsymbol{X})) \mathrm{d}\boldsymbol{X} \\ &+ \rho \int_{\boldsymbol{X}} q(\boldsymbol{X}) \mathrm{KL}(q(\boldsymbol{o} \mid \boldsymbol{Z}) \parallel \tilde{q}(\boldsymbol{o} \mid \boldsymbol{Z})) \mathrm{d}\boldsymbol{X} \end{aligned}$$

where

$$\begin{aligned} \tilde{L}(\tilde{q}(o \mid \boldsymbol{Z})) = \sum_o q(o) \int_{\boldsymbol{X}} \int_{\boldsymbol{Z}} q(\boldsymbol{X} \mid \boldsymbol{Z}) q(\boldsymbol{Z} \mid o) (\\ f(\boldsymbol{X}) + \rho(\log \tilde{q}(\boldsymbol{Z} \mid o) + \log \tilde{q}(o) - \log \tilde{q}(\boldsymbol{X})) \mathrm{d}\boldsymbol{Z}\mathrm{d}\boldsymbol{X} \\ + \rho(H(q(\boldsymbol{Z} \mid o)) + H(q(o))) \end{aligned}$$

is a lower bound to L_π as the expected KL divergences are always non negative. Full derivations can be found in Appendix C.1.

6.2.3. Component Updates

Ignoring the terms that don't affect the maximization of the lower bound L_π with respect to a single component $q(\boldsymbol{Z} \mid o)$, we can optimize

$$\begin{aligned} \underset{q(\boldsymbol{Z} \mid o)}{\text{maximize}} \quad & \int_{\boldsymbol{Z}} q(\boldsymbol{Z} \mid o) R_o(\boldsymbol{Z}) \mathrm{d}\boldsymbol{Z} + \rho H(q(\boldsymbol{Z} \mid o)) \\ \text{subject to} \quad & \mathrm{KL}(q(\boldsymbol{Z} \mid o) \parallel q_t(\boldsymbol{Z} \mid o)) \leq \epsilon \end{aligned}$$

individually for each component using the method presented in Section 6.1. Using the fact that $f(\boldsymbol{X})$ and $q(\boldsymbol{X})$ are permutation invariant by definition, the component-specific reward $R_o(\boldsymbol{Z})$ is given by

$$\begin{aligned} R_o(\boldsymbol{Z}) &= \int_{\boldsymbol{X}} q(\boldsymbol{X} \mid \boldsymbol{Z})(f(\boldsymbol{X}) + \rho(\log \tilde{q}(\boldsymbol{Z} \mid o) - \log \tilde{q}(\boldsymbol{X}))\mathrm{d}\boldsymbol{X} \\ &= f(\boldsymbol{X} = \boldsymbol{Z}) + \rho(\log \tilde{q}(\boldsymbol{Z} \mid o) - \log \tilde{q}(\boldsymbol{X} = \boldsymbol{Z})) \end{aligned}$$

6.2.4. Weight Updates

From a reinforcement learning perspective, the weight update is not strictly necessary but presented here for the sake of completeness. The mixture coefficients $q(o)$ can be updated by optimizing

$$\underset{q(o)}{\text{maximize}} \quad \sum_o q(o)R(o) + \rho H(q(o))$$

where the reward function $R(o)$ is given by

$$\begin{aligned} R(o) = &\int_Z q(\boldsymbol{Z} \mid o) \int_X q(\boldsymbol{X} \mid \boldsymbol{Z})(f(\boldsymbol{X}) + \rho(\log \tilde{q}(\boldsymbol{Z} \mid o) \\ &+ \log \tilde{q}(o) - \log \tilde{q}(\boldsymbol{X}))\mathrm{d}\boldsymbol{X}\mathrm{d}\boldsymbol{Z} + \rho H(q(\boldsymbol{Z} \mid o)) \\ = &\int_Z q(\boldsymbol{Z} \mid o)(f(\boldsymbol{X} = \boldsymbol{Z}) + \rho(\log \tilde{q}(\boldsymbol{Z} \mid o) + \log \tilde{q}(o) \\ &- \log \tilde{q}(\boldsymbol{X} = \boldsymbol{Z})))\mathrm{d}\boldsymbol{Z} + \rho H(q(\boldsymbol{Z} \mid o)) \end{aligned}$$

6.3. Conclusion

In this section, we laid out the challenges that arise when scaling episode-based reinforcement learning methods using movement primitives to problems involving multiple agents. We showed that the inevitable increase in problem dimensionality can be mitigated by factorizing the joint policy and support this idea with initial experiments supporting the reduced computational and sample complexity while maintaining performance compared to a joint model. Next, we attended to the challenges trying to learn versatile solutions involving multiple agents. Here, each agent introduces factorially many redundant solutions due to permutations of agents resulting in identical behavior. Based on a permutation invariant formulation of a Gaussian mixture model, we proposed an extension to the VIPS algorithm to learn versatile behavior while avoiding to learn redundant solutions. An experimental evaluation of this algorithm is left for future work.

7. Conclusion

Sequential decision making problems often challenge reinforcement learning algorithms with very large state and action spaces. Especially so in multi-agent scenarios, where spaces increase with each additional agent. Yet, if they have a special structure, this can be taken into account by the learning algorithm. The question underlying this thesis therefore was to exploit these structures to improve on state representation for deep reinforcement learning in swarms, as well as exploration in high dimensional action spaces which are often found in episode-based reinforcement learning using movement primitives.

7.1. Summary of Contributions

This thesis is concerned with state representation in homogeneous swarms, as well as exploration of high dimensional action spaces in episode-based reinforcement learning. In this section, we briefly revisit the main contributions of this thesis.

7.1.1. Fundamentals and State of the Art

This chapter serves as an introduction to the topic of sequential decision making and reviews current relational state representation methods and their application in multi-agent learning. First, the foundations for reinforcement learning in the step-based and episode-based learning setting are laid out. The step-based view is taken in Chapters 3 and 4 while the episode-based view is taken in Chapters 5 and 6. Then, the underlying data structure the agents in Chapters 3 and 4 is explained in detail with additional references in recent literature. Last, we provide an in-depth look on exploration in reinforcement learning with a special focus on trust-region methods which are used throughout this thesis.

7.1.2. Local Communication Protocols for Learning Complex Swarm Behaviors with Deep Reinforcement Learning

Recent deep multi-agent reinforcement learning literature has focused mainly on improving algorithmic details such as new methods for value function estimation with multiple agents, or credit assignment. Data representation techniques, however, hardly received attention with many works using simple techniques such as concatenation of observations. With an increasing number of agents, this approach quickly becomes infeasible. An area to study these techniques is swarm learning, where many identical agents cooperate to achieve a common goal and the data observed by the agents has the special structure of a set. In Chapter 3, we propose embeddings of geometric features using histograms that exploit this structure. We study the effect of different history length, as well as different numbers of observed features per agent. This work led to the publication of Hüttenrauch, Šošić, and Neumann [80].

7.1.3. Deep Reinforcement Learning for Swarm Systems

In Chapter 4, we extend the idea of feature embeddings to learned embeddings. To this end, each item of an agent's observation is first projected non-linearly into a higher dimensional features space. Afterwards, these latent features are aggregated using a permutation invariant function such as the mean. Last, global task features can be concatenated to the embedding and serve as the input to the policy. We provide extensive comparisons between hand-designed and learned embeddings on different feature sets. In these experiments, the learned mean embeddings show superior performance to previous methods. The state representations using mean embeddings have been published in Hüttenrauch, Šošić, and Neumann [79].

7.1.4. Robust Black-Box Optimization for Stochastic Search and Episodic Reinforcement Learning

In Chapter 5, we turn to the problem of exploration in high dimensional action spaces for episode-based reinforcement learning. We present Coordinate Ascent MORE with Step-Size Adaptation, a zero-order black-box optimization algorithm based on a policy search objective. While the original MORE algorithm improved the exploration process through a lower bound on the entropy of the policy, we propose a new version of the algorithm that is comprised of separate updates or the mean and covariance of policy. Additionally, we use an evolution path, introduced to evolutionary algorithms by the CMA-ES algorithm, to scale the covariance depending on previous update directions. Last, we exchange the previous surrogate modelling approach with a much simpler ordinary least squares approach. We show that this new algorithm is more sample efficient than policy gradient methods and provides better final performance on stochastic objectives. This work has been published in the Journal of Machine Learning Research [78].

7.1.5. Black-Box Optimization for Episode-Based Multi-Agent Reinforcement Learning

After studying in-depth the functioning of black-box optimization in single agent episode-based reinforcement learning, we show how to scale the approach to scenarios with multiple agents. The limiting factor here is the number of samples necessary for the estimation of a high quality local model of the objective, which scales quadratically with the number of agents. We show that inter-agent correlation do not need to be modelled explicitly reducing the complexity to linear in the number of agents. We provide initial experiments that show promising results. We additionally sketch a method to learn versatile behavior in the presence of multiple agents. Learning multi-modal policies is especially challenging since exchanging agents introduces factorially more, yet redundant, additional solutions. By introducing a permutation invariant version of a Gaussian mixture model, learning these redundant solutions can be prevented. Experimental validation, however, remains future work.

7.2. Discussion and Outlook

Several core aspects of state representation and parameter exploration for reinforcement learning have been discussed in this thesis. In the following, we discuss the proposed methods and provide an outlook on future work directions.

7.2.1. State Representation for Learning in Swarms

The first part of this thesis concentrated on state representation techniques for data that is presented in form of a set. Starting with heuristically chosen histograms of features, we arrived at the formulation of mean embeddings as a scalable and expressive method for deep reinforcement learning in swarms. This line of work opens several interesting future work directions. First of all, mean aggregation puts equal weight on all latent features, which may be necessary to accurately describe the underlying data distribution. In swarm learning, however, information from far away agents may not constitute to solving the task at all. Instead, a way to focus on relevant information could be to use learned attention mechanisms, effectively filtering out irrelevant information. Further, more sophisticated aggregation methods could be helpful in scenarios with noisy sensing. A promising technique here is Bayesian aggregation [151] where an estimate of the uncertainty is calculated in addition to a mean aggregation. These approaches have been initially tried by my student Robin Ruede during his master's thesis [130] where we compared Bayesian and attentive aggregation to the mean embedding approach. Despite the more powerful capabilities, we could only establish similar performance to the simpler mean embedding approach. Thus, further studies are necessary to understand where these approaches have a bigger impact.

A different approach, at least at first sight, is solving the multi-agent learning problem at a graph level using graph neural networks. Using graph neural networks allows for more sophisticated forms of communication along the edges of connected agents. First experiments have been conducted by my student Lyubomira Dimitrova in her master's theses [40]. Here, we tested the influence of message passing networks in limited visibility scenarios of the problems established in Chapter 4. Initial results suggest an advantage over mean embeddings, albeit consistency and stability of the results needs further work. Additionally, we established a connection between mean embeddings and graph neural networks. Mean embeddings can be seen as a form of proto-GNN. They only collect information from immediate neighbors, similar to a one-hop message passing network.

7.2.2. Black-Box Optimization for Episode-Based Multi-Agent Reinforcement Learning

The second part of the thesis is concerned with black-box optimization of movement primitive parameters. We first identified two main attributes that contribute to a successful optimization using the MORE algorithm in single-agent learning. The first attribute is the estimation of the model parameters. The method introduced in Section 5.4 leads to an efficient and more exact estimation of the natural gradient direction compared to other evolutionary approaches, as well as policy gradient based methods. This sparks the question of how to combine the more powerful policy gradient based methods, which are able to use non-linear mappings of high dimensional context vectors, with the more exact natural gradient provided by the surrogate model estimation process of CAS-MORE. The second attribute is the more powerful entropy regularization. The original MORE algorithm only featured a lower bound on the entropy of the search distribution using a fixed entropy reduction that usually needs problem specific adjustments. The approach introduced in Section 5.3.2 replaces the lower bound on the entropy with an adaptive scaling mechanism for the covariance, based on previous update steps. Again, combining this approach with more powerful contextual algorithms remains future work.

Next, we started extending the approach to the multi-agent setup. We provided a formulation for learning movement primitive parameters using a joint model learning, yet individual policy learning approach. Initial results are promising but need further investigation on more complex tasks. Again, it would be interesting to extend the findings to policy gradient based methods.

Bibliography

[1] Abbas Abdolmaleki et al. “Maximum a Posteriori Policy Optimisation”. In: *International Conference on Learning Representations*. 2018.

[2] Abbas Abdolmaleki et al. “Model-Based Relative Entropy Stochastic Search”. In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes et al. Vol. 28. Curran Associates, Inc., 2015.

[3] Abbas Abdolmaleki et al. “Relative entropy regularized policy iteration”. In: *arXiv preprint arXiv:1812.02256* (2018).

[4] “Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation”. In: *Proceedings of IEEE international conference on evolutionary computation*. IEEE. 1996, pp. 312–317.

[5] Rishabh Agarwal et al. “Deep reinforcement learning at the edge of the statistical precipice”. In: *Advances in neural information processing systems* 34 (2021), pp. 29304–29320.

[6] Youhei Akimoto et al. “Bidirectional relation between CMA evolution strategies and natural evolution strategies”. In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2010, pp. 154–163.

[7] Riad Akrour et al. “Model-free trajectory-based policy optimization with monotonic improvement”. In: *The Journal of Machine Learning Research* 19.1 (2018), pp. 565–589.

[8] Javier Alonso-Mora et al. “Distributed multi-robot formation control among obstacles: A geometric and optimization approach with consensus”. In: *Proceedings of the IEEE International Conference on Robotics and Automation*. 2016, pp. 5356–5363.

[9] Shun-Ichi Amari. “Natural gradient works efficiently in learning”. In: *Neural computation* 10.2 (1998), pp. 251–276.

[10] Brandon Amos and Denis Yarats. “The differentiable cross-entropy method”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 291–302.

[11] Marcin Andrychowicz et al. “What Matters for On-Policy Deep Actor-Critic Methods? A Large-Scale Study”. In: *International Conference on Learning Representations*. 2021.

[12] Oleg Arenz, Gerhard Neumann, and Mingjun Zhong. “Efficient gradient-free variational inference using policy search”. In: *International conference on machine learning*. PMLR. 2018, pp. 234–243.

[13] Oleg Arenz, Mingjun Zhong, and Gerhard Neumann. “Trust-Region Variational Inference with Gaussian Mixture Models.” In: *Journal of Machine Learning Research* 21 (2020), pp. 163–1.

[14] Farshad Arvin et al. “Colias: An autonomous micro robot for swarm robotic applications”. In: *International Journal of Advanced Robotic Systems* 11.7 (2014), p. 113.

[15] Arthur Aubret, Laetitia Matignon, and Salima Hassas. “A survey on intrinsic motivation in reinforcement learning”. In: *arXiv preprint arXiv:1908.06976* (2019).

[16] Anne Auger and Nikolaus Hansen. “A restart CMA evolution strategy with increasing population size”. In: *2005 IEEE congress on evolutionary computation*. Vol. 2. IEEE. 2005, pp. 1769–1776.

[17] Anne Auger, Marc Schoenauer, and Nicolas Vanhaecke. “LS-CMA-ES: A second-order algorithm for covariance matrix adaptation”. In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2004, pp. 182–191.

[18] Franz Aurenhammer. “Voronoi diagrams—a survey of a fundamental geometric data structure”. In: *ACM Computing Surveys (CSUR)* 23.3 (1991), pp. 345–405.

[19] P. Basu and J. Redi. “Movement control algorithms for realization of fault-tolerant ad hoc robot networks”. In: *IEEE Network* 18.4 (2004), pp. 36–44.

[20] Peter W Battaglia et al. “Relational inductive biases, deep learning, and graph networks”. In: *arXiv preprint arXiv:1806.01261* (2018).

[21] Levent Bayındır. “A review of swarm robotics tasks”. In: *Neurocomputing* 172 (2016), pp. 292–321.

[22] Philipp Becker, Oleg Arenz, and Gerhard Neumann. “Expected Information Maximization: Using the I-Projection for Mixture Density Estimation”. In: *International Conference on Learning Representations*. 2019.

[23] Daniel S. Bernstein et al. “The Complexity of Decentralized Control of Markov Decision Processes”. In: *Mathematics of Operations Research* 27.4 (2002), pp. 819–840.

[24] Hans-Georg Beyer and Hans-Paul Schwefel. “Evolution strategies–a comprehensive introduction”. In: *Natural computing* 1.1 (2002), pp. 3–52.

[25] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, 1999.

[26] Zdravko I Botev et al. “The cross-entropy method for optimization”. In: *Handbook of statistics*. Vol. 31. Elsevier, 2013, pp. 35–59.

[27] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: `arXiv:1606.01540`.

[28] Michael M Bronstein et al. “Geometric deep learning: Grids, groups, graphs, geodesics, and gauges”. In: *arXiv preprint arXiv:2104.13478* (2021).

[29] Konstantinos Chatzilygeroudis et al. “Black-box data-efficient policy search for robotics”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 51–58.

[30] Haoqiang Chen et al. “Gama: Graph attention multi-agent reinforcement learning algorithm for cooperation”. In: *Applied Intelligence* 50 (2020), pp. 4195–4205.

[31] Jianing Chen, Melvin Gauci, and Roderich Groß. “A strategy for transporting tall objects with a swarm of miniature mobile robots”. In: *Proceedings of the IEEE International Conference on Robotics and Automation*. 2013, pp. 863–869.

[32] Mo Chen, Zhengyuan Zhou, and Claire J Tomlin. “A path defense approach to the multiplayer reach-avoid game”. In: *IEEE 53rd Annual Conference on Decision and Control (CDC)*. 2014, pp. 2420–2426.

[33] Mo Chen, Zhengyuan Zhou, and Claire J Tomlin. “Multiplayer reach-avoid games via low dimensional solutions and maximum matching”. In: *American Control Conference (ACC)*. 2014, pp. 1444–1449.

[34] Mo Chen, Zhengyuan Zhou, and Claire J Tomlin. “Multiplayer reach-avoid games via pairwise outcomes”. In: *IEEE Transactions on Automatic Control* 62.3 (2017), pp. 1451–1457.

[35] Tianshu Chu et al. “Multi-agent deep reinforcement learning for large-scale traffic signal control”. In: *IEEE Transactions on Intelligent Transportation Systems* 21.3 (2019), pp. 1086–1095.

[36] Timothy H. Chung, Geoffrey A. Hollinger, and Volkan Isler. "Search and Pursuit-Evasion in Mobile Robotics". In: *Autonomous Robots* 31.4 (2011), p. 299.

[37] Nikolaus Correll and Alcherio Martinoli. "Modeling and designing self-organized aggregation in a swarm of miniature robots". In: *The International Journal of Robotics Research* 30.5 (2011), pp. 615–626.

[38] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. "A survey on policy search for robotics". In: *Foundations and trends in Robotics* 2.1-2 (2013), pp. 388–403.

[39] D. V. Dimarogonas and K. J. Kyriakopoulos. "On the Rendezvous Problem for Multiple Nonholonomic Agents". In: *IEEE Transactions on Automatic Control* 52.5 (2007), pp. 916–922.

[40] Lyubomira Dimitrova. "Swarm Reinforcement Learning in Limited Visibility with Graph Neural Networks". MA thesis. KIT, 2022.

[41] Russell C Eberhart, Yuhui Shi, and James Kennedy. *Swarm intelligence*. Elsevier, 2001.

[42] M. Egerstedt and Xiaoming Hu. "Formation Constrained Multi-Agent Control". In: *IEEE Transactions on Robotics and Automation* 17.6 (2001), pp. 947–951.

[43] Felix End et al. "Layered direct policy search for learning hierarchical skills". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 6442–6448.

[44] Tony Finch. "Incremental calculation of weighted mean and variance". In: *University of Cambridge* 4 (2009), pp. 11–5.

[45] S. Finck et al. *Real-Parameter Black-Box Optimization Benchmarking 2009: Presentation of the Noiseless Functions*. Tech. rep. 2009/20. Updated February 2010. Research Center PPE, 2009.

[46] Jakob Foerster et al. "Counterfactual multi-agent policy gradients". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.

[47] Jakob Foerster et al. "Learning to communicate with deep multi-agent reinforcement learning". In: *Advances in neural information processing systems* 29 (2016).

[48] Niklas Freymuth et al. "Swarm Reinforcement Learning for Adaptive Mesh Refinement". In: *ICLR 2023 Workshop on Physics for Machine Learning*. 2023.

[49] G. H. W. Gebhardt et al. "Learning Robust Policies for Object Manipulation with Robot Swarms". In: *IEEE International Conference on Robotics and Automation*. 2018.

[50] M. C. De Gennaro and A. Jadbabaie. "Decentralized Control of Connectivity for Multi-Agent Systems". In: *IEEE Conference on Decision and Control*. 2006, pp. 3628–3633.

[51] Tobias Glasmachers et al. "Exponential natural evolution strategies". In: *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. 2010, pp. 393–400.

[52] Dani Goldberg and Maja J Mataric. "Robust behavior-based control for distributed multi-robot collection tasks". In: (2000).

[53] Stephen M Goldfeld, Richard E Quandt, and Hale F Trotter. "Maximization by quadratic hill-climbing". In: *Econometrica: Journal of the Econometric Society* (1966), pp. 541–551.

[54] Arthur Gretton et al. "A Kernel Statistical Test of Independence". In: *Advances in Neural Information Processing Systems*. 2008, pp. 585–592.

[55] Arthur Gretton et al. "A Kernel Two-Sample Test". In: *Journal of Machine Learning Research* 13.Mar (2012), pp. 723–773.

[56] Aditya Grover et al. "Learning Policy Representations in Multiagent Systems". In: *arXiv:1806.06464* (2018).

[57] Shixiang Gu et al. “Q-Prop: Sample-efficient policy gradient with an off-policy critic”. In: *Proceedings of the 5th International Conference on Learning Representations*. 2017.

[58] Jayesh K. Gupta, Maxim Egorov, and Mykel Kochenderfer. “Cooperative Multi-Agent Control using Deep Reinforcement Learning”. In: *International Conference on Autonomous Agents and Multiagent Systems*. 2017, pp. 66–83.

[59] Tuomas Haarnoja et al. “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor”. In: *International conference on machine learning*. PMLR. 2018, pp. 1861–1870.

[60] Julia Handl and Bernd Meyer. “Ant-based and Swarm-based Clustering”. In: *Swarm Intelligence* 1.2 (2007), pp. 95–113.

[61] N. Hansen et al. “COCO: A Platform for Comparing Continuous Optimizers in a Black-Box Setting”. In: *Optimization Methods and Software* (2020).

[62] N. Hansen et al. “COCO: Performance Assessment”. In: *ArXiv e-prints* arXiv preprint arXiv:1605.03560 (2016).

[63] N. Hansen et al. “COCO: The Experimental Procedure”. In: *ArXiv e-prints* arXiv preprint arXiv:1603.08776 (2016).

[64] N. Hansen et al. *Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions*. Tech. rep. RR-6829. Updated February 2010. INRIA, 2009.

[65] N. Hansen et al. *Real-Parameter Black-Box Optimization Benchmarking 2012: Experimental Setup*. Tech. rep. INRIA, 2012.

[66] Nikolaus Hansen. “A global surrogate assisted CMA-ES”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2019, pp. 664–672.

[67] Nikolaus Hansen. “Benchmarking a BI-population CMA-ES on the BBOB-2009 function testbed”. In: *Proceedings of the 11th annual conference companion on genetic and evolutionary computation conference: late breaking papers*. 2009, pp. 2389–2396.

[68] Nikolaus Hansen. “The CMA evolution strategy: A tutorial”. In: *arXiv preprint arXiv:1604.00772* (2016).

[69] Nikolaus Hansen, Dirk V Arnold, and Anne Auger. “Evolution strategies”. In: *Springer handbook of computational intelligence* (2015), pp. 871–898.

[70] Nikolaus Hansen et al. “A method for handling uncertainty in evolutionary optimization with an application to feedback control of combustion”. In: *IEEE Transactions on Evolutionary Computation* 13.1 (2008), pp. 180–197.

[71] Nikolaus Hansen et al. “COCO: A platform for comparing continuous optimizers in a black-box setting”. In: *Optimization Methods and Software* 36.1 (2021), pp. 114–144.

[72] Nikolaus Hansen et al. “COCO: The experimental procedure”. In: *arXiv preprint arXiv:1603.08776* (2016).

[73] Nikolaus Hansen et al. *Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions*. Research Report RR-6829. INRIA, 2009.

[74] Matthew Hausknecht and Peter Stone. “Deep Recurrent Q-Learning for Partially Observable MDPs”. In: *AAAI Fall Symposium Series*. 2015.

[75] Verena Heidrich-Meisner and Christian Igel. “Hoeffding and Bernstein races for selecting policies in evolutionary direct policy search”. In: *Proceedings of the 26th Annual International Conference on Machine Learning*. 2009, pp. 401–408.

[76] N. R. Hoff et al. “Two foraging algorithms for robot swarms using only local communication”. In: *Proceedings of the IEEE International Conference on Robotics and Biomimetics*. 2010, pp. 123–130.

[77] John H Holland. “Genetic algorithms”. In: *Scientific american* 267.1 (1992), pp. 66–73.

[78] Maximilian Hüttenrauch and Gerhard Neumann. “Robust Black-Box Optimization for Stochastic Search and Episodic Reinforcement Learning”. In: *Journal of Machine Learning Research* 25.153 (2024), pp. 1–44.

[79] Maximilian Hüttenrauch, Adrian Šošić, and Gerhard Neumann. “Deep reinforcement learning for swarm systems”. In: *Journal of Machine Learning Research* 20.54 (2019), pp. 1–31.

[80] Maximilian Hüttenrauch, Adrian Šošić, and Gerhard Neumann. “Local Communication Protocols for Learning Complex Swarm Behaviors with Deep Reinforcement Learning”. In: *International Conference on Swarm Intelligence*. 2018.

[81] Jemin Hwangbo et al. “ROCK∗—Efficient black-box optimization for policy learning”. In: *2014 IEEE-RAS International Conference on Humanoid Robots*. IEEE. 2014, pp. 535–540.

[82] Oscar Ibáñez et al. “An experimental study on the applicability of evolutionary algorithms to craniofacial superimposition in forensic identification”. In: *Information Sciences* 179.23 (2009), pp. 3998–4028.

[83] A. Jadbabaie, Jie Lin, and A. S. Morse. “Coordination of Groups of Mobile Autonomous Agents using Nearest Neighbor Rules”. In: *IEEE Transactions on Automatic Control* 48.6 (2003), pp. 988–1001.

[84] M. Ji and M. Egerstedt. “Distributed Coordination Control of Multiagent Systems While Preserving Connectedness”. In: *IEEE Transactions on Robotics* 23.4 (2007), pp. 693–703.

[85] Jiechuan Jiang et al. “Graph Convolutional Reinforcement Learning”. In: *International Conference on Learning Representations*. 2019.

[86] Steven G Johnson. *The NLopt nonlinear-optimization package*. 2014.

[87] Sham M Kakade. “A natural policy gradient”. In: *Advances in neural information processing systems* 14 (2001).

[88] C.Ronald Kube and Eric Bonabeau. “Cooperative transport by ants and robots”. In: *Robotics and Autonomous Systems* 30.1 (2000), pp. 85–101.

[89] Jakub Kudela. “A critical problem in benchmarking and analysis of evolutionary computation methods”. In: *Nature Machine Intelligence* (2022), pp. 1–8.

[90] Solomon Kullback and Richard A Leibler. “On information and sufficiency”. In: *The annals of mathematical statistics* 22.1 (1951), pp. 79–86.

[91] A Kupcsik et al. “Data-efficient contextual policy search for robot movement skills”. In: *Proceedings of the National Conference on Artificial Intelligence (AAAI)*. Bellevue. 2013.

[92] Jeffrey Larson, Matt Menickelly, and Stefan M Wild. “Derivative-free optimization methods”. In: *Acta Numerica* 28 (2019), pp. 287–404.

[93] Juho Lee et al. “Set transformer: A framework for attention-based permutation-invariant neural networks”. In: *International conference on machine learning*. PMLR. 2019, pp. 3744–3753.

[94] Zhenhua Li and Qingfu Zhang. “What does the evolution path learn in CMA-ES?” In: *Parallel Problem Solving from Nature–PPSN XIV: 14th International Conference, Edinburgh, UK, September 17-21, 2016, Proceedings 14*. Springer. 2016, pp. 751–760.

[95] Timothy P. Lillicrap et al. “Continuous control with deep reinforcement learning”. In: (2016). Ed. by Yoshua Bengio and Yann LeCun.

[96] J. Lin, A. Morse, and B. Anderson. "The Multi-Agent Rendezvous Problem. Part 2: The Asynchronous Case". In: *SIAM Journal on Control and Optimization* 46.6 (2007), pp. 2120–2147.

[97] Zhiyun Lin, M. Broucke, and B. Francis. "Local control strategies for groups of mobile autonomous agents". In: *IEEE Transactions on Automatic Control* 49.4 (2004), pp. 622–629.

[98] Dong C Liu and Jorge Nocedal. "On the limited memory BFGS method for large scale optimization". In: *Mathematical programming* 45.1 (1989), pp. 503–528.

[99] Ilya Loshchilov, Marc Schoenauer, and Michele Sebag. "Alternative restart strategies for CMA-ES". In: *International Conference on Parallel Problem Solving from Nature*. Springer. 2012, pp. 296–305.

[100] Ilya Loshchilov, Marc Schoenauer, and Michele Sebag. "Self-adaptive surrogate-assisted covariance matrix adaptation evolution strategy". In: *Proceedings of the 14th annual conference on Genetic and evolutionary computation*. 2012, pp. 321–328.

[101] Ryan Lowe et al. "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments". In: *Advances in Neural Information Processing Systems*. 2017, pp. 6379–6390.

[102] Renzhi Lu et al. "Multi-agent deep reinforcement learning based demand response for discrete manufacturing systems energy management". In: *Applied Energy* 276 (2020), p. 115473.

[103] Alcherio Martinoli, Kjerstin Easton, and William Agassounon. "Modeling swarm robotic systems: a case study in collaborative distributed manipulation". In: *The International Journal of Robotics Research* 23.4-5 (2004), pp. 415–436.

[104] L. Matignon, G. J. Laurent, and N. L. Fort-Piat. "Hysteretic Q-Learning: An Algorithm for Decentralized Reinforcement Learning in Cooperative Multi-Agent Teams". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2007, pp. 64–69.

[105] Volodymyr Mnih et al. "Human-Level Control through Deep Reinforcement Learning". In: *Nature* 518 (2015), pp. 529–533.

[106] Christoph Moeslinger, Thomas Schmickl, and Karl Crailsheim. "Emergent flocking with low-end swarm robots". In: *Proceedings of the 7th International Conference on Swarm Intelligence Swarm Intelligence*. Ed. by Marco Dorigo et al. 2010, pp. 424–431.

[107] Igor Mordatch and Pieter Abbeel. "Emergence of Grounded Compositional Language in Multi-Agent Populations". In: *AAAI Conference on Artificial Intelligence*. 2018.

[108] R Murphy et al. "Janossy Pooling: Learning Deep Permutation-Invariant Functions for Variable-Size Inputs". In: *International Conference on Learning Representations (ICLR 2019)*. 2019.

[109] John A Nelder and Roger Mead. "A simplex method for function minimization". In: *The computer journal* 7.4 (1965), pp. 308–313.

[110] S. Nouyan et al. "Teamwork in self-organized robot colonies". In: *IEEE Transactions on Evolutionary Computation* 13.4 (2009), pp. 695–711.

[111] Frans Oliehoek. "Decentralized POMDPs". In: *Reinforcement Learning: State of the Art* 12 (2013), pp. 471–503. DOI: 10.1007/978-3-642-27645-3_15.

[112] Shayegan Omidshafiei et al. "Deep Decentralized Multi-task Multi-Agent Reinforcement Learning under Partial Observability". In: *International Conference on Machine Learning*. 2017, pp. 2681–2690.

[113] Michael A Osborne, Roman Garnett, and Stephen J Roberts. "Gaussian processes for global optimization". In: *3rd international conference on learning and intelligent optimization (LION3)*. 2009, pp. 1–15.

[114] Fabian Otto et al. “Deep black-box reinforcement learning with movement primitives”. In: *Conference on Robot Learning*. PMLR. 2023, pp. 1244–1265.

[115] Fabian Otto et al. “Differentiable Trust Region Layers for Deep Reinforcement Learning”. In: *International Conference on Learning Representations*. 2020.

[116] Joni Pajarinen et al. “Compatible natural gradient policy search”. In: *Machine Learning* 108.8 (2019), pp. 1443–1466.

[117] Gregory Palmer et al. “Lenient Multi-Agent Deep Reinforcement Learning”. In: *arXiv:1707.04402* (2017).

[118] Liviu Panait, Keith Sullivan, and Sean Luke. “Lenient Learners in Cooperative Multiagent Systems”. In: *International Joint Conference on Autonomous Agents and Multiagent Systems*. 2006, pp. 801–803.

[119] Alexandros Paraschos et al. “Probabilistic Movement Primitives”. In: *Advances in Neural Information Processing Systems*. Ed. by C. J. C. Burges et al. Vol. 26. Curran Associates, Inc., 2013.

[120] Alexandros Paraschos et al. “Probabilistic movement primitives”. In: *Advances in neural information processing systems* 26 (2013).

[121] Jan Peters, Katharina Mulling, and Yasemin Altun. “Relative entropy policy search”. In: *Twenty-Fourth AAAI Conference on Artificial Intelligence*. 2010.

[122] Matthias Plappert et al. “Parameter Space Noise for Exploration”. In: *International Conference on Learning Representations*. 2018.

[123] Kenneth Price. “Differential evolution vs. the functions of the second ICEO”. In: *Proceedings of the IEEE International Congress on Evolutionary Computation*. Piscataway, NJ, USA: IEEE, 1997, pp. 153–157. DOI: 10.1109/ICEC.1997.592287.

[124] WL1551847 Price. “Global optimization by controlled random search”. In: *Journal of optimization theory and applications* 40.3 (1983), pp. 333–348.

[125] Antonin Raffin, Jens Kober, and Freek Stulp. “Smooth exploration for robotic reinforcement learning”. In: *Conference on Robot Learning*. PMLR. 2022, pp. 1634–1644.

[126] B. Ranjbar-Sahraei et al. “A Novel Robust Decentralized Adaptive Fuzzy Control for Swarm Formation of Multiagent Systems”. In: *IEEE Transactions on Industrial Electronics* 59.8 (2012), pp. 3124–3134.

[127] Tabish Rashid et al. “QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning”. In: *arXiv:1803.11485* (2018).

[128] Reuven Y Rubinstein and Dirk P Kroese. *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation, and machine learning*. Vol. 133. Springer, 2004.

[129] Sebastian Ruder. “An overview of gradient descent optimization algorithms”. In: *arXiv preprint arXiv:1609.04747* (2016).

[130] Robin Ruede. “Bayesian and Attentive Aggregation for Multi-Agent Deep Reinforcement Learning”. MA thesis. KIT, 2021.

[131] Andrei A Rusu et al. “Policy Distillation”. In: *arXiv:1511.06295* (2015).

[132] Muhammad Saleem, Gianni A Di Caro, and Muddassar Farooq. “Swarm intelligence based routing protocol for wireless sensor networks: Survey and future directions”. In: *Information Sciences* 181.20 (2011), pp. 4597–4624.

[133] Tim Salimans et al. “Evolution strategies as a scalable alternative to reinforcement learning”. In: *arXiv preprint arXiv:1703.03864* (2017).

[134] Alvaro Sanchez-Gonzalez et al. “Learning to simulate complex physics with graph networks”. In: *International conference on machine learning*. PMLR. 2020, pp. 8459–8468.

[135] Adam Santoro et al. “A simple neural network module for relational reasoning”. In: *Advances in neural information processing systems* 30 (2017).

[136] John Schulman et al. “Proximal policy optimization algorithms”. In: *arXiv:1707.06347* (2017).

[137] John Schulman et al. “Trust Region Policy Optimization”. In: *International Conference on Machine Learning*. 2015, pp. 1889–1897.

[138] Ali Shavandi and Majid Khedmati. “A multi-agent deep reinforcement learning framework for algorithmic trading in financial markets”. In: *Expert Systems with Applications* 208 (2022), p. 118124.

[139] Maruan Al-Shedivat et al. “Continuous Adaptation via Meta-Learning in Nonstationary and Competitive Environments”. In: *International Conference on Learning Representations*. 2018.

[140] Yoav Shoham and Kevin Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2008.

[141] David Silver et al. “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play”. In: *Science* 362.6419 (2018), pp. 1140–1144.

[142] Alex Smola et al. “A Hilbert Space Embedding for Distributions”. In: *International Conference on Algorithmic Learning Theory*. 2007, pp. 13–31.

[143] Adrian Šošić et al. “Inverse Reinforcement Learning in Swarm Systems”. In: *International Conference on Autonomous Agents and Multiagent Systems*. 2017, pp. 1413–1421.

[144] James C Spall. *Introduction to stochastic search and optimization: estimation, simulation, and control*. Vol. 65. John Wiley & Sons, 2005.

[145] Chen Sun et al. “Stochastic prediction of multi-agent interactions from partial observations”. In: *arXiv preprint arXiv:1902.09641* (2019).

[146] Yi Sun et al. “Efficient natural evolution strategies”. In: *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. 2009, pp. 539–546.

[147] Peter Sunehag et al. “Value-Decomposition Networks For Cooperative Multi-Agent Learning”. In: *arXiv:1706.05296* (2017).

[148] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[149] Richard S Sutton et al. “Policy gradient methods for reinforcement learning with function approximation”. In: *Advances in neural information processing systems* 12 (1999).

[150] Yee Whye Teh et al. “Distral: robust multitask reinforcement learning”. In: *arXiv:1707.04175* (2017).

[151] Michael Volpp et al. “Bayesian Context Aggregation for Neural Processes”. In: *International Conference on Learning Representations*. 2020.

[152] Edward Wagstaff et al. “Universal approximation of functions on sets”. In: *The Journal of Machine Learning Research* 23.1 (2022), pp. 6762–6817.

[153] Christopher JCH Watkins and Peter Dayan. “Q-learning”. In: *Machine learning* 8 (1992), pp. 279–292.

[154] Daan Wierstra et al. “Natural evolution strategies”. In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 949–980.

[155] Ronald J Williams. “Simple statistical gradient-following algorithms for connectionist reinforcement learning”. In: *Machine learning* 8.3 (1992), pp. 229–256.

[156] Susanne Winter et al. “Registration of CT and intraoperative 3-D ultrasound images of the spine using evolutionary and gradient-based methods”. In: *IEEE Transactions on Evolutionary Computation* 12.3 (2008), pp. 284–296.

[157] Ulf Witkowski et al. “Ad-hoc network communication infrastructure for multi-robot systems in disaster scenarios”. In: *Proceedings of the IARP/EURON Workshop on Robotics for Risky Interventions and Environmental Surveillance*. 2008.

[158] Annie Wong et al. “Deep multiagent reinforcement learning: Challenges and directions”. In: *Artificial Intelligence Review* 56.6 (2023), pp. 5023–5056.

[159] Yaodong Yang et al. “Mean Field Multi-Agent Reinforcement Learning”. In: *arXiv:1802.05438* (2018).

[160] Zelda B Zabinsky. “Random Search Algorithms”. In: *Wiley Encyclopedia of Operations Research and Management Science* (2010).

[161] Manzil Zaheer et al. “Deep sets”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 3391–3401.

[162] Lianmin Zheng et al. “MAgent: A Many-Agent Reinforcement Learning Platform for Artificial Collective Intelligence”. In: *arXiv:1712.00600* (2017).

[163] Zhengyuan Zhou et al. “A general, open-loop formulation for reach-avoid games”. In: *IEEE 51st Annual Conference on Decision and Control (CDC)*. 2012, pp. 6501–6506.

[164] Zhengyuan Zhou et al. “Cooperative Pursuit with Voronoi Partitions”. In: *Automatica* 72 (2016), pp. 64–72.

[165] Ciyou Zhu et al. “Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization”. In: *ACM Transactions on mathematical software (TOMS)* 23.4 (1997), pp. 550–560.

[166] Brian D Ziebart et al. “Maximum entropy inverse reinforcement learning.” In: *Aaai*. Vol. 8. Chicago, IL, USA. 2008, pp. 1433–1438.

A. Appendix for Chapter 4

A.1. Agent Kinematics

In the single integrator case, the state of an agent is given by $s^i = [x^i, y^i, \phi^i] \in \mathcal{S} = \{[x, y, \phi] \in \mathbb{R}^3 : 0 \leq x \leq x_{\max},\ 0 \leq y \leq y_{\max},\ 0 \leq \phi < 2\pi\}$, and the linear velocity v and angular velocity ω can be directly controlled by the agent. The kinematic model is given by

$$\begin{aligned}\dot{x} &= v \cos\phi \\ \dot{y} &= v \sin\phi \\ \dot{\phi} &= \omega.\end{aligned}$$

In the double integrator case, the state is given by $s^i = [x^i, y^i, \phi^i, v^i, \omega^i] \in \mathcal{S}, \mathcal{S} = \{[x, y, \phi, v, \omega] \in \mathbb{R}^5 : 0 \leq x \leq x_{\max},\ 0 \leq y \leq y_{\max},\ 0 \leq \phi < 2\pi, |v| \leq v_{\max}, |\omega| \leq \omega_{\max}\}$ and the agent can only indirectly change its velocity by acceleration. With the control inputs a_v and a_ω, the model is then given by

$$\begin{aligned}\dot{v} &= a_v \\ \dot{\omega} &= a_\omega \\ \dot{x} &= v \cos\phi \\ \dot{y} &= v \sin\phi \\ \dot{\phi} &= \omega.\end{aligned}$$

For the experiments, we use finite differences to model the system in discrete time.

A.2. Observation Model

Irrespective of the task, an agent i can sense the following properties about other agents $j \in \mathcal{N}(i)$ within its neighborhood:

$$\begin{aligned}d^{i,j} &&\text{distance to neighboring agents} \\ \phi^{i,j} &= \arctan\left(\frac{y^j - y^i}{x^j - x^i}\right) - \phi^i &\text{bearing to neighboring agents} \\ \theta^{i,j} &= \arctan\left(\frac{y^i - y^j}{x^i - x^j}\right) - \phi^j &\text{relative orientation} \\ \Delta v^{i,j} &= v^i \begin{bmatrix}\cos\phi^i \\ \sin\phi^i\end{bmatrix} - v^j \begin{bmatrix}\cos\phi^j \\ \sin\phi^j\end{bmatrix} &\text{relative velocity}\end{aligned}$$

Furthermore, each agent has access to the following local properties:

$$
\begin{aligned}
d^i_{\text{wall}} &= \min\left(\begin{bmatrix} x^i - x_{\min} \\ y^i - y_{\min} \\ x_{\max} - x^i \\ y_{\max} - y^i \end{bmatrix}\right) && \text{distance to closest wall} \\
\phi^i_{\text{wall}} &= \varphi^i_{\text{wall}} - \phi^i && \text{orientation to closest wall} \\
v^i, \omega^i & && \text{own velocity}
\end{aligned}
$$

where φ^i_{wall} denotes the absolute bearing of agent i to the closest wall segment.

A.3. Task Specific Communication Protocols

In the rendezvous task, agent i additionally can sense information about neighborhood sizes:

$$
\begin{aligned}
|\mathcal{N}(i)| && \text{own neighborhood size} \\
|\mathcal{N}(j)| : j \in \mathcal{N}(i) && \text{neighborhood size of neighbor } j
\end{aligned}
$$

In pursuit evasion, we additionally have one or multiple evaders with states $s^e = [x^e, y^e] \in \{[x, y] \in \mathbb{R}^2 : 0 \leq x \leq x_{\max},\ 0 \leq y \leq y_{\max}\}$. Agents can sense the distance and bearing to an evader, given that the evader is within an observation distance d_o:

$$
\begin{aligned}
d^{i,e} &= \sqrt{(x^i - x^e)^2 + (y^i - y^e)^2} \quad \text{if } d^{i,e} \leq d_o && \text{distance to evader} \\
\phi^{i,e} &= \arctan\left(\frac{y^e - y^i}{x^e - x^i}\right) - \phi^i \quad \text{if } d^{i,e} \leq d_o && \text{bearing to evader}
\end{aligned}
$$

Furthermore, we assume that each agent i can compute a shortest path to the evader over a graph of connected agents, such that the path $P = (v^1, v^2 \ldots, v^M)$ minimizes the sum $\sum_{m=1}^{M-1} d^{m,m+1}$ where v^1 is agent i and v^M is the evader.

A.4. Controller for Double Integrator Dynamics

We use a simple PD-controller to transform the consensus protocol with high-level direct state manipulation to the unicycle model with double integrator dynamics. It is given by

$$
\begin{aligned}
a_v &= K_1(v_d - v) \\
a_\omega &= K_2(\phi_d - \phi) + D_2(\omega_d - \omega) \\
v_d &= \|\dot{\boldsymbol{x}}\| \\
\phi_d &= \arctan\left(\frac{\dot{y}}{\dot{x}}\right) \\
\omega_d &= 0,
\end{aligned}
$$

where the parameters K_1, K_2 and D_2 are tuned manually to give good performance on the problem.

A.5. Reward Functions

A.5.1. Rendezvous

The reward function is defined in terms of the inter-agent distances $\{d^{i,j}\}$ as

$$R(\boldsymbol{s}, \boldsymbol{a}) = \alpha \sum_{i=1}^{N} \sum_{j=i+1}^{N} \min(d^{i,j}, d_c) + \beta \|\boldsymbol{a}\|,$$

where in the global observability case we set the cut-off distance $d_c = \max(x_{\max}, y_{\max})$ to the maximum possible inter-agent distance in the respective environment. The factor $\alpha = -\left(\frac{N(N-1)}{2} d_c\right)^{-1}$ serves as a reward normalization factor and $\beta = -1 \times 10^{-3}$ controls how strongly high action outputs of the policy are penalized.

A.5.2. Pursuit Evasion

For the case of a single evader, the pursuit evasion objective may be expressed in terms of the distance to the closest pursuer. More specifically, the reward function is given as

$$R(\boldsymbol{s}, \boldsymbol{a}) = -\frac{1}{d_o} \min(d_{\min}, d_o),$$

where $d_{\min} = \min(d^{1,e}, \ldots, d^{N,e})$. For the global observability case, we set d_o to the maximum possible distance of $d^{i,e}$.

A.5.3. Pursuit Evasion with Multiple Evaders

In the case of multiple evaders, we use a sparser reward function that counts how many evaders are caught per time step, with no additional guidance of inter-agent distances. An evader e is assumed to be caught if the closest pursuer's distance $d_{\min,e} = \min(d^{1,e}, \ldots, d^{N,e})$ is closer than a threshold distance $d_t = 3$. The reward function is given by

$$R(\boldsymbol{s}, \boldsymbol{a}) = \sum_{e=1}^{E} \mathbf{1}_{[0,d_t]}(d_{\min,e}),$$

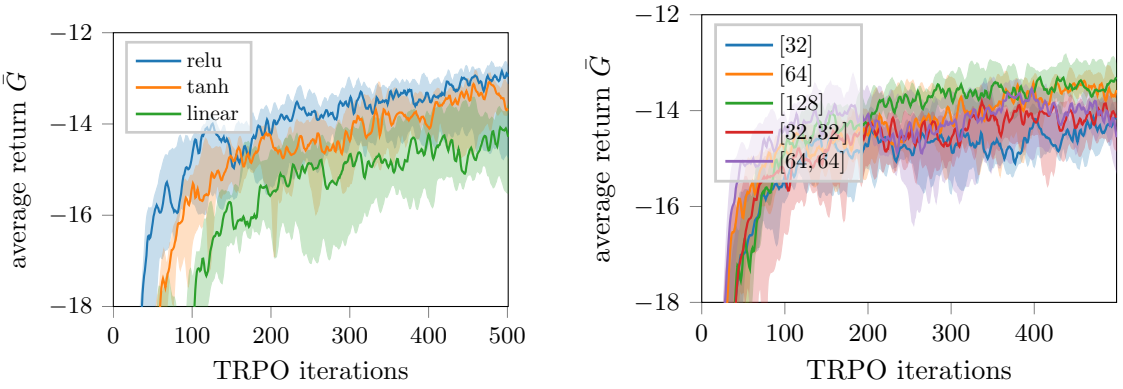
where E is the number of evaders and

$$\mathbf{1}_{[a,b]}(x) = \begin{cases} 1 & \text{if } x \in [a, b] \\ 0 & \text{else} \end{cases}$$

is the indicator function.

A.6. Policy Architectures

This section briefly summarizes the chosen policy architectures. Illustrations can be found in Figure 4.1.



(a) Activation functions evaluation.

(b) Layer size evaluation.

Figure A.1.: Learning curves for 20 agent rendezvous with (a) different activation functions for the mean embedding and (b) different layer numbers and sizes using a RELU activation function. The curves show the median of the average return $\bar{G}$ based on the top five trials.

A.6.1. Neural Network Embedding Policy

We evaluated different layer sizes and activation functions on the rendezvous problem and show the results in Figure A.1. In all other experiments, the neural network mean feature embedding for agent i, given by

$$\phi^{\mathrm{NN}}(O^i) = \frac{1}{|O^i|} \sum_{o^{i,j} \in O^i} \phi(o^{i,j}),$$

is realized as the empirical mean of the outputs of a single layer feed-forward neural network,

$$\phi(o^{i,j}) = h(W o^{i,j} + b),$$

with 64 neurons and a RELU non-linearity h.

A.6.2. Histogram Embedding Policy

The histogram embedding is achieved with a two-dimensional histogram over the distance and bearing space to other agents. We use eight evenly spaced bins for each feature, resulting in a 64 dimensional feature vector.

A.6.3. RBF Embedding Policy

The RBF embedding is given by a vector $\phi^{\mathrm{RBF}}(O^i) = \left[\psi_1(O^i), \ldots, \psi_{M^2}(O^i)\right]$ of M^2 contributions from $M = 8$ radial basis functions whose center points are evenly distributed in the distance and bearing space. With $o^{i,j} = [d^{i,j}, \phi^{i,j}]$, $\mu_m = [\mu_{d,m}, \mu_{\phi,m}]$, and $\sigma = [\sigma_d, \sigma_\phi]$ its components are given by

$$\psi_m(O^i) = \sum_{o^{i,j} \in O^i} \rho_m(o^{i,j}),$$

where we choose

$$\rho_m(o^{i,j}) = \exp\left(-\frac{1}{2}\left[\frac{(d^{i,j} - \mu_{d,m})^2}{\sigma_d^2} + \frac{(\phi^{i,j} - \mu_{\phi,m})^2}{\sigma_\phi^2}\right]\right).$$

The policy network structure used for both, the histogram and the RBF representations, is illustrated in Figure 4.1b.

A.6.4. Concatenation Policy

For the concatenation method, we first concatenate agent i's neighborhood observations contained in the set O^i and process them with one hidden layer of 64 neurons and a RELU non-linearity. The resulting feature vector is then concatenated with the local properties o^i_{loc} and fed into a second layer of same size. Finally, the output of the second layer is mapped to the action. The corresponding policy network structure can be seen in Figure 4.1c.

B. Appendix for Chapter 5

B.1. Derivation of CA-MORE Dual

Before we solve the optimization problems, we will first state the closed-form solution of the objective and KL-divergence using a quadratic model under multi-variate Gaussian distributions. The solution to the objective is given by

$$\int_{\boldsymbol{x}} \pi(\boldsymbol{x}) \hat{f}(\boldsymbol{x}) \mathrm{d}x = -\frac{1}{2}\boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{\mu} - \frac{1}{2}\mathrm{tr}(\boldsymbol{A}\Sigma) + \boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{a} + a_0$$

and the KL-divergence between two Gaussian distributions is given by

$$\begin{aligned}\mathrm{KL}(\pi \parallel \pi_t) = &\frac{1}{2}\left\{(\boldsymbol{\mu}_t - \boldsymbol{\mu})^{\mathrm{T}}\Sigma_t^{-1}(\boldsymbol{\mu}_t - \boldsymbol{\mu})\right. \\ &\left. + \mathrm{tr}(\Sigma_t^{-1}\Sigma) - k + \log|\Sigma_t| - \log|\Sigma|\right\}.\end{aligned}$$

B.1.1. Mean Update

Setting $\Sigma = \Sigma_t$, the optimization problem is given by

$$\begin{aligned}\underset{\boldsymbol{\mu}}{\text{maximize}} \quad & -\frac{1}{2}\boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{\mu} + \boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{a} \\ \text{subject to} \quad & \frac{1}{2}(\boldsymbol{\mu}_t - \boldsymbol{\mu})^{\mathrm{T}}\Sigma_t^{-1}(\boldsymbol{\mu}_t - \boldsymbol{\mu}) \leq \epsilon_\mu\end{aligned}$$

and the Lagrangian is given by

$$\begin{aligned}L(\boldsymbol{\mu}, \lambda) = &-\frac{1}{2}\boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{\mu} + \boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{a} \\ &+ \lambda\left(\epsilon_\mu - \frac{1}{2}(\boldsymbol{\mu}_t - \boldsymbol{\mu})^{\mathrm{T}}\Sigma_t^{-1}(\boldsymbol{\mu}_t - \boldsymbol{\mu})\right)\end{aligned}$$

where λ is a Lagrangian multiplier. The optimal solution $\boldsymbol{\mu}^*$ in terms of the Lagrangian multipliers can be found by differentiating L with respect to $\boldsymbol{\mu}$ and setting it to 0, i.e,

$$\frac{\partial L}{\partial \boldsymbol{\mu}} = -\boldsymbol{A}\boldsymbol{\mu} + \boldsymbol{a} + \lambda\Sigma_t^{-1}(\boldsymbol{\mu}_t - \boldsymbol{\mu}) \stackrel{!}{=} 0.$$

Using the solution λ^*,

$$\boldsymbol{\mu}^* = \underbrace{(\lambda^*\Sigma_t^{-1} + \boldsymbol{A})}_{\boldsymbol{M}_\mu(\lambda^*)}{}^{-1} \underbrace{(\lambda^*\Sigma_t^{-1}\boldsymbol{\mu}_t + \boldsymbol{a})}_{\boldsymbol{m}_\mu(\lambda^*)} = \boldsymbol{M}_\mu(\lambda^*)^{-1}\boldsymbol{m}_\mu(\lambda^*).$$

After rearranging terms, the dual problem for the mean is given by

$$\begin{aligned}g_\mu(\lambda) = \lambda\epsilon_\mu + \frac{1}{2}\Big(&\boldsymbol{m}_\mu(\lambda)^{\mathrm{T}}\boldsymbol{M}_\mu(\lambda)^{-1}\boldsymbol{m}_\mu(\lambda) \\ &- \lambda\boldsymbol{m}_t^{\mathrm{T}}\boldsymbol{M}_t^{-1}\boldsymbol{m}_t\Big).\end{aligned}$$

B.1.2. Covariance Update

Analogously, we set $\boldsymbol{\mu} = \boldsymbol{\mu}_t$. The optimization problem for the covariance is given by

$$\begin{aligned} \underset{\boldsymbol{\Sigma}}{\text{maximize}} \quad & -\frac{1}{2}\text{tr}(\boldsymbol{A}\boldsymbol{\Sigma}) \\ \text{subject to} \quad & \frac{1}{2}\left(\text{tr}(\boldsymbol{\Sigma}_t^{-1}\boldsymbol{\Sigma}) - k + \log|\boldsymbol{\Sigma}_t| - \log|\boldsymbol{\Sigma}|\right) < \epsilon_\Sigma \end{aligned}$$

and the Lagrangian for the covariance matrix optimization is given by

$$\begin{aligned} L(\boldsymbol{\Sigma}, \nu) = & -\frac{1}{2}\text{tr}(\boldsymbol{A}\boldsymbol{\Sigma}) \\ & + \nu\left(\epsilon_\Sigma - \frac{1}{2}\left(\text{tr}(\boldsymbol{\Sigma}_t^{-1}\boldsymbol{\Sigma}) - k + \log|\boldsymbol{\Sigma}_t| - \log|\boldsymbol{\Sigma}|\right)\right) \end{aligned}$$

where ν is again a Lagrangian multiplier. The optimal solution $\boldsymbol{\Sigma}^*$ can be found analogously and is given by

$$\frac{\partial L_\Sigma}{\partial \boldsymbol{\Sigma}} = -\frac{1}{2}\boldsymbol{A} - \frac{1}{2}\nu\boldsymbol{\Sigma}_t^{-1} + \frac{1}{2}\nu\boldsymbol{\Sigma}^{-1} \stackrel{!}{=} 0.$$

With the solution ν^*,

$$\boldsymbol{\Sigma}^* = \underbrace{\left((\nu^*\boldsymbol{\Sigma}_t^{-1} + \boldsymbol{A})/\nu^*\right)}_{\boldsymbol{S}(\nu^*)}{}^{-1} = \boldsymbol{S}(\nu^*)^{-1}.$$

After rearranging terms again, the dual for the covariance is given by

$$\Lambda(\nu) = \frac{\nu\boldsymbol{\Sigma}_t^{-1} + \boldsymbol{A}}{\nu}.$$

B.2. Robust Target Normalization

In reinforcement learning, a reward function that accurately describes the task and is easy to optimize is not always given. Here, we want to demonstrate that even with a reward function that includes large jumps, using a robust target normalization scheme leads to good results. To this end, we use a different reward function formulation for the hole-reacher task from Section 5.5.2.1. The reward in this case is defined as the negative squared distance to a target point at the bottom of the hole minus a large penalty whenever the robot hits the ground. We leave the depth at -1 to focus on the effects of target normalization. Figure B.1 shows the negative reward (the cost) on a log-scale and we compare CAS-MORE using mean/std normalization and robust mean/std normalization. A cost of 1 corresponds to the end-effector of the robot moving close to the entrance of the hole but failing to reach inside. Only robust normalization is able to capture both, the task reward and the penalty, and guides the robot to reach down the hole. We provide pseudo-code for the robust target normalization technique in Algorithm 1.

Algorithm 1 Robust target normalization

procedure NORMALIZE($\mathcal{Y}$) ▷ Robust normalization of targets
  $\bar{y}_\mathcal{Y} \leftarrow \frac{1}{|\mathcal{Y}|}\sum_q^{|\mathcal{Y}|} y_q$
  $\sigma_\mathcal{Y} \leftarrow \sqrt{\frac{1}{|\mathcal{Y}|}\sum_q^{|\mathcal{Y}|}(y_q - \bar{y}_\mathcal{Y})^2}$
  $y \leftarrow \frac{y - \bar{y}_\mathcal{Y}}{\sigma_\mathcal{Y}}$ ▷ Standardize all elements in $\mathcal{Y}$
  $\mathcal{I} \leftarrow -v_{\text{clip}} < y < v_{\text{clip}}$ ▷ Boolean mask of all elements in $\mathcal{Y}$ that are in $(-v_{\text{clip}}, v_{\text{clip}})$
  $\mathcal{S} \leftarrow \{y \in \mathcal{Y} \mid -v_{\text{clip}} < y < v_{\text{clip}}\}$ ▷ All elements in $\mathcal{Y}$ that are in $(-v_{\text{clip}}, v_{\text{clip}})$
  $k \leftarrow \text{Kurt}[\mathcal{S}] - 3$ ▷ Excess kurtosis of the elements in $\mathcal{S}$
  if $k > 0.55$ **and** $\sigma_\mathcal{Y} \neq 1$ **then**
    $\mathcal{Y}(\mathcal{I}) \leftarrow$ NORMALIZE($\mathcal{S}$) ▷ Normalize elements in $\mathcal{S}$
  end if
  $\mathcal{Y} \leftarrow \text{clip}(\mathcal{Y}, \min(\mathcal{Y}(\mathcal{I})), \max(\mathcal{Y}(\mathcal{I}))$
  return $\mathcal{Y}$
end procedure

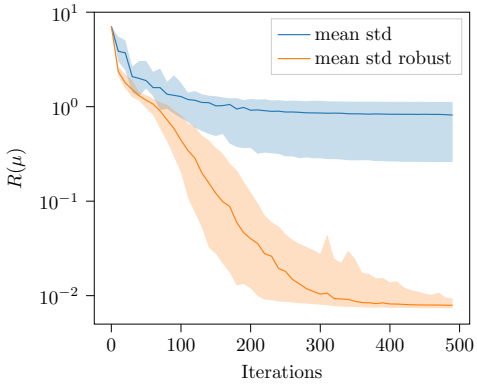


Figure B.1.: Comparison of mean/std normalization and robust normalization on a penalty based reward function for the hole-reaching task.

Table B.1.: Empirically found default hyper-parameters for CAS-MORE based on the problem dimensionality n.

Parameter	Default Value
K: Population size	$4 + \lfloor 3\log(n) \rfloor$
Q_{max}: Maximum queue size	$\max\{\lceil 1.5(1 + \frac{n(n+3)}{2}) \rceil,\ 8(n+1)\}$
ϵ_μ: Trust-region for the mean	0.5
ϵ_Σ: Trust-region for the covariance	$\frac{1.5}{10+n^{1.5}}$
c_σ: Smoothing factor of evolution path	$\frac{1}{2+n^{0.75}}$
v_{clip}: Clip value for robust normalization	3
Excess kurtosis threshold	0.55

B.3. Hyper-Parameters

We provide default hyper-parameters for CAS-MORE in terms of the problem dimensionality n in Table B.1 which we empirically found to work well over all benchmark functions. For some functions, a higher bound on the mean often leads to quicker convergence but may result in divergence for others.

Table B.2 provides the chosen hyper-parameters for the step-based deep RL experiments (PPO, SAC, TD3), as well as the BBRL experiments. Note, that samples per iteration corresponds to tuples (s, a, r, s') in the step-based case, and whole trajectories in the BBRL case.

Table B.2.: Hyper-parameters for the deep RL and BBRL experiments.

	PPO	SAC	TD3	BBRL-PPO	BBRL-TRPL
samples per iteration	16000	1	1	64	64
GAE λ	0.95	n.a.	n.a.	n.a.	n.a.
discount factor	0.99	0.99	0.99	n.a.	n.a.
ϵ_μ	n.a.	n.a.	n.a.	n.a.	0.05
ϵ_Σ	n.a.	n.a.	n.a.	n.a.	0.005
optimizer	adam	adam	adam	adam	adam
epochs	10	n.a.	n.a.	100	100
learning rate	3e-4	3e-4	3e-4	1e-3	3e-4
use critic	True	True	True	False	False
epochs critic	10	n.a.	n.a.	n.a.	n.a.
learning rate critic	3e-4	3e-4	3e-4	n.a.	n.a.
learning rate alpha	n.a.	3e-4	n.a.	n.a.	n.a.
warm up steps	0	10000	25000	0	0
minibatch size	512	n.a.	n.a.	64	64
batch size	n.a.	256	256	n.a.	n.a.
buffer size	n.a.	1e6	1e6	n.a.	n.a.
polyak_weight	n.a.	5e-3	5e-3	n.a.	n.a.
normalized observations	True	False	False	False	False
normalized rewards	True	False	False	False	False
critic clip	0.2	n.a.	n.a.	0.2	n.a.
importance ratio clip	0.2	n.a.	n.a.	0.2	n.a.
hidden layers	[256, 256]	[256, 256]	[256, 256]	n.a	n.a
hidden layers critic	[256, 256]	[256, 256]	[256, 256]	n.a.	n.a.
hidden activation	tanh	ReLU	ReLU	n.a.	n.a.
initial std	0.6	1.0	0.1	1.0	1.0

B.4. Black-box Optimization Benchmarks

Results from experiments according to Hansen et al. [63] and Hansen et al. [62] on the benchmark functions given in Finck et al. [45] and Hansen et al. [64] are presented in Figures B.2 to B.4. The experiments were performed with COCO [61], version 2.4.1.1, the plots were produced with version 2.4.1.1. The **expected runtime (ERT)**, used in the figures and tables, depends on a given target function value, $f_t = f_{opt} + \Delta f$, and is computed over all relevant trials as the number of function evaluations executed during each trial while the best function value did not reach f_t, summed over all trials and divided by the number of trials that actually reached f_t [65, 123]. **Statistical significance** is tested with the rank-sum test for a given target Δf_t using, for each trial, either the number of needed function evaluations to reach Δf_t (inverted and multiplied by -1), or, if the target was not reached, the best Δf-value achieved, measured only up to the smallest number of overall function evaluations for any unsuccessful trial under consideration.

B.5. Episodic RL

In this section, we provide additional information and reward functions for the episodic RL tasks. The action cost for all tasks is given by

$$\tau_t = \sum_i^K (a_t^i)^2.$$

B.5.1. Holereaching

The reward function is composed of two phases. In the first phase, the distance $d_g = \|\boldsymbol{p}_t - \boldsymbol{p}_g\|$ of the position of the end-effector $\boldsymbol{p}_t = (x_{ee,t}, y_{ee,t})$ to a goal point $\boldsymbol{p}_g = (2, -0.1)$ below the entrance of the hole is minimized. Afterwards, the reward scales linearly with the absolute value $y_{ee,t}$ of the end-effector. The task ends if either $t = 200$ or a collision happens. In the deterministic case, the hole has a depth of 1 m, while in the stochastic case, the depth is sampled uniformly from $\mathcal{U}(0.98\,\text{m}, 1.02\,\text{m})$. The task reward is given by

$$R_{task} = \begin{cases} c_{coll} \exp(-d_g) & \text{if a collision happened or } t = T, \\ \exp(-d_g) & \text{if } y_{ee,t} \geq 0 \text{ and no collision happened}, \\ 1 + |y_{ee,t}| & \text{if } y_{ee,t} < 0 \text{ and no collision happened} \end{cases}$$

where $c_{coll} = 0.25$ if there is a collision and $c_{coll} = 1$ otherwise.

Dense reward. The dense reward is given in each time-step by

$$r_t = R_{task} - 0.01\tau_t$$

Sparse reward. The sparse reward only returns the task reward in the terminal time-step T which can either be the last time-step of the episode or the time-step where a collision happens and is given by

$$r_t = \begin{cases} -0.001\tau_t & \text{if } t < T, \\ R_{task} - 0.001\tau_t & \text{if } t = T. \end{cases}$$

B.5.2. Table Tennis

In the table tennis task, the episode starts with a fixed ball position and velocity. In order to add stochasticity to the environment, we add noise to the initial velocity in x-direction (i.e., along the long edge of the table tennis table). The task is to hit the ball so that it lands close to the opponent's side short edge of the table. The reward function is non Markovian and is based on the minimum distance $d_{b,r} = \min_t \|\boldsymbol{p}_{b,t} - \boldsymbol{p}_{r,t}\|$ between the ball position $\boldsymbol{p}_{b,t}$ and the racket position $\boldsymbol{p}_{r,t}$, the minimum distance $d_{b,g} = \min_t \|\boldsymbol{p}_{b,t} - \boldsymbol{p}_g\|$ between the ball position and a goal point $\boldsymbol{p}_g = (-1.3, 0, 0.77)$, and the distance $d_{l,e} = |p_{l,x} - p_{g,x}|$ between the x-coordinate of the ball landing position $\boldsymbol{p}_l$ and x-coordinate of the opponent's edge $p_{g,x} = -1.3$ within an episode. We use the transformation

$$\rho(x) = 1 - \frac{x}{1+x}$$

to convert large distances to a value of 0 and small distances to a value of 1. The task reward is given as

$$R_{\text{task}} = \begin{cases} 0.2\rho(d_{b,r}) & \text{if cond. 1,} \\ 0.5\rho(d_{b,r}) + \rho(d_{b,g}) & \text{if cond. 2,} \\ 2\rho(d_{b,r}) + 3\rho(d_{l,e}) & \text{if cond. 3 ,} \\ 5\rho(d_{b,r}) + 10|p_{l,x}| & \text{if cond. 4 ,} \end{cases}$$

where the conditions are given by

- cond. 1: the ball was not hit,
- cond. 2: the ball was hit but landed on floor,
- cond. 3: the ball was hit, landed on the table, and $p_{l,x} \geq -1.25$,
- cond. 4: the ball was hit, landed on the table and $p_{l,x} < -1.25$.

We consider a trajectory to be successful if the ball lands within 10cm of the opponent's side edge of the table. We use the sparse-in-time reward

$$r_t = \begin{cases} -0.001\tau_t & \text{if } t < T, \\ R_{\text{task}} - 0.001\tau_t & \text{if } t = T. \end{cases}$$

B.5.3. Beerpong

In the table beerpong task, the episode starts with the ball attached to the end-effector of the arm. In order to add stochasticity to the environment, we add noise to the velocity at a pre-defined fixed time-step of the release of the ball. For a successful throw, the ball first needs to bounce once on the table and then land inside the cup. The reward function is again non Markovian and is based on the final distance $d_{b,c,T} = \|\boldsymbol{p}_{b,T} - \boldsymbol{p}_c\|$, and on the minimum distance $d_{b,c} = \min_t \|\boldsymbol{p}_{b,t} - \boldsymbol{p}_c\|$ between the ball position $\boldsymbol{p}_{b,t}$ and the center of the cup $\boldsymbol{p}_c$ within an episode. With the same transform $\rho(x)$ as before, the task reward is given by

$$R_{\text{task}} = \begin{cases} 0.2\rho(d_{b,c}) + 0.1\rho(d_{b,c,T}) & \text{if cond. 1,} \\ \rho(d_{b,c}) + 0.5\rho(d_{b,c,T}) & \text{if cond. 2,} \\ \rho(d_{b,c}) + 2\rho(d_{b,c,T}) + 1 & \text{if cond. 3,} \\ \rho(d_{b,c}) + 2\rho(d_{b,c,T}) + 3 & \text{if cond. 4,} \end{cases}$$

where the conditions are given by

- cond. 1: the ball first has contact with anything but the table,
- cond. 2: the ball first has contact with the table but is not in cup,
- cond. 3: the ball is in cup, without contact with the table,
- cond. 4: the ball is in cup and had contact with the table before.

The sparse-in-time reward is given by

$$r_t = \begin{cases} -0.1\tau_t & \text{if } t < T, \\ R_{\text{task}} - 0.1\tau_t & \text{if } t = T. \end{cases}$$

B.5.4. Hopper Jump

The task reward for the hopper task is composed of a reward term for jumping as high as possible and a distance minimization term to the center of top of the box. Additionally, bonuses are added for successfully landing upright and landing on the box. High velocities, as well as joint angles that lead to falling over on the box result in a penalty. We record the maximum height h_{max} of the agent's torso during an episode, the minimum distance $d_{f,c} = \min_t \|\boldsymbol{p}_{f,t} - \boldsymbol{p}_c\|$ and final distance $d_{f,c,T} = \|\boldsymbol{p}_{f,T} - \boldsymbol{p}_c\|$ between the agent's foot position $\boldsymbol{p}_f$ and the top center of the box $\boldsymbol{p}_c$. The individual reward terms are given as

$$R_{\text{height}} = \begin{cases} 5h_{\text{max}} & \text{if } h_{\text{max}} < 2, \\ 10 + h_T & \text{if } h_{\text{max}} \geq 2 \text{ and cond. 1}, \\ 2h_{\text{max}} & \text{if cond. 2}, \end{cases}$$

$$R_{\text{dist}} = \begin{cases} -5 & \text{if } h_{\text{max}} < 2, \\ -5d_{f,c,T} & \text{if } h_{\text{max}} \geq 2, \end{cases}$$

$$R_{\text{min dist}} = -2d_{f,c}$$

$$R_{\text{healthy}} = \begin{cases} 1 & \text{if } z_T \in [0.7, \infty], \phi \in [-\infty, \infty] \text{ and } \theta, \dot{\theta}, \in [-100, 100], \\ 0 & \text{else}, \end{cases}$$

$$R_{\text{box}} = \begin{cases} 5 & \text{if cond. 3}, \\ 0 & \text{else}, \end{cases}$$

$$R_{\text{box vel}} = \begin{cases} -\min(10v_x^2, 1) & \text{if cond. 4}, \\ 0 & \text{else}. \end{cases}$$

with conditions

- cond. 1: the agent did not fall after landing on the box,
- cond. 2: the agent is on the box but falling over,
- cond. 3: the agent is on the box and $h_T > 1.6$,
- cond. 4: the agent is on the box.

The task reward is given as

$$R_{\text{task}} = R_{\text{height}} + R_{\text{dist}} + R_{\text{min dist}} + R_{\text{healthy}} + R_{\text{box}} + R_{\text{box vel}}.$$

and the total reward for each time step is given as

$$r_t = \begin{cases} -10^{-4}\tau_t & \text{if } t < T, \\ R_{\text{task}} - 10^{-4}\tau_t & \text{if } t = T. \end{cases}$$

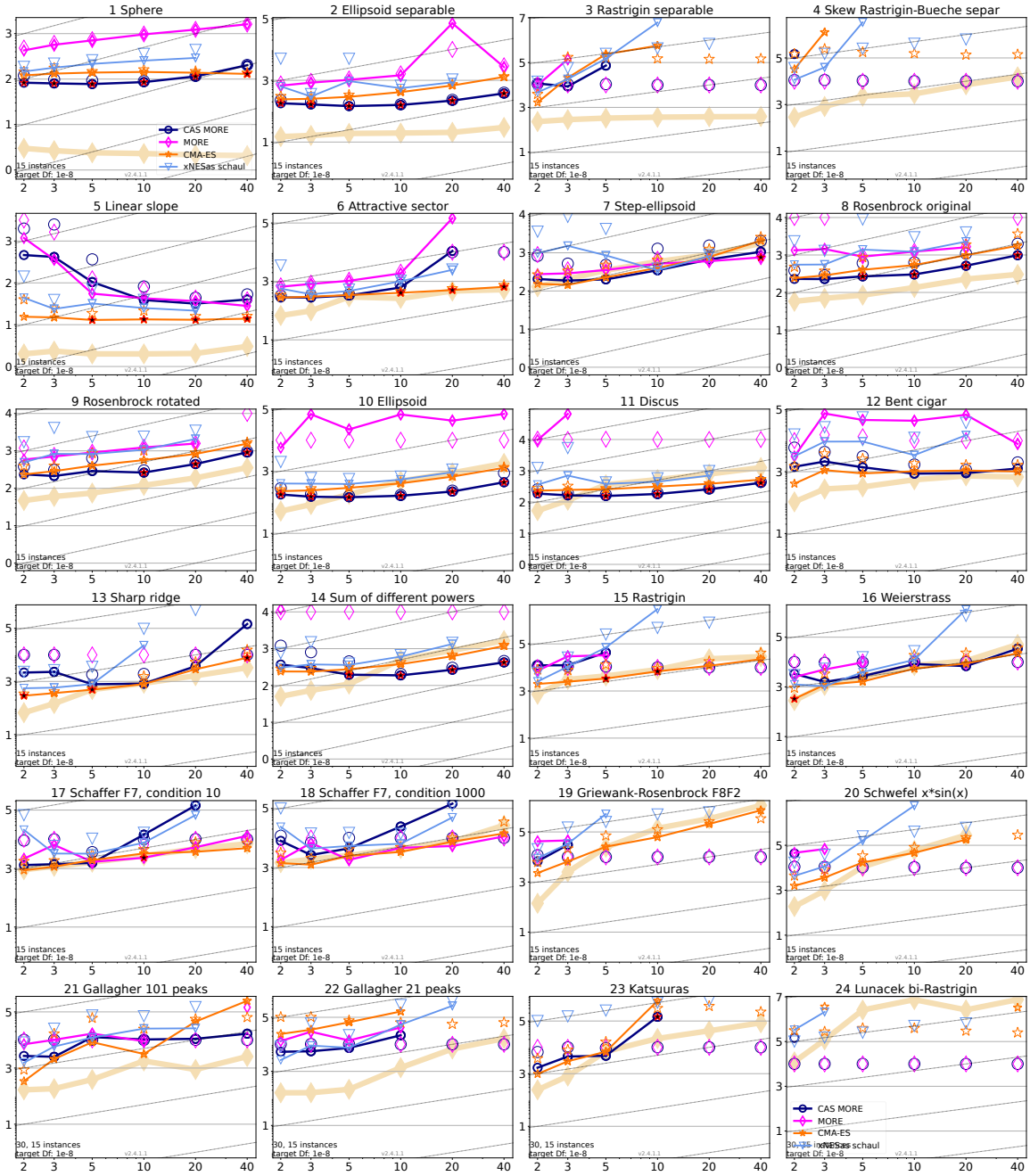


Figure B.2.: Expected running time (ERT in number of f-evaluations as $\log_{10}$ value), divided by dimension for target function value 10^{-8} versus dimension. Slanted grid lines indicate quadratic scaling with the dimension. Different symbols correspond to different algorithms given in the legend of f_1 and f_{24}. Light symbols give the maximum number of function evaluations from the longest trial divided by dimension. Black stars indicate a statistically better result compared to all other algorithms with $p < 0.01$ and Bonferroni correction number of dimensions (six). Legend: ○: CAS-MORE, ◇: MORE, ★: CMA-ES, ▽: xNESas.

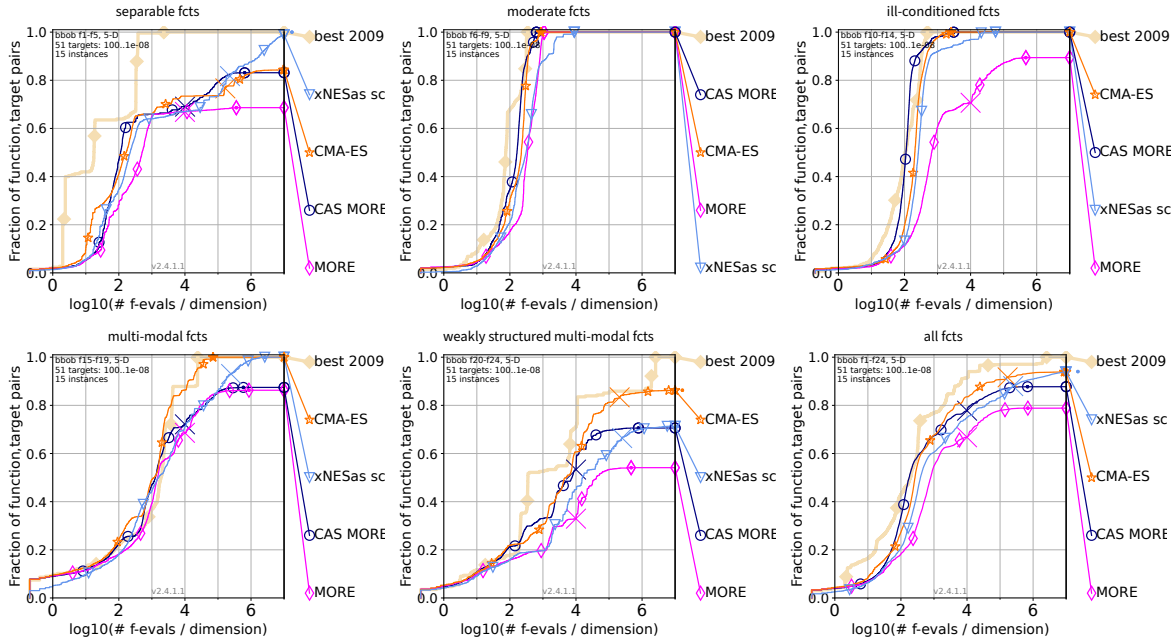


Figure B.3.: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for 51 targets with target precision in $10^{[-8..2]}$ for all functions and subgroups in 5-D. As reference algorithm, the best algorithm from BBOB 2009 is shown as light thick line with diamond markers.

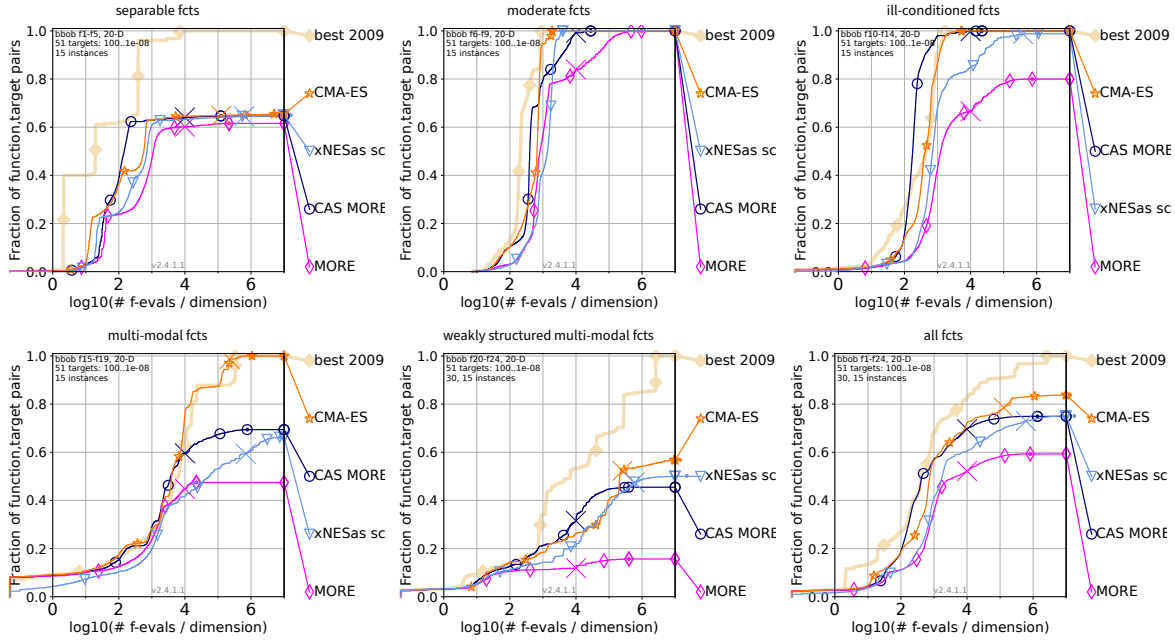


Figure B.4.: Bootstrapped empirical cumulative distribution of the number of objective function evaluations divided by dimension (FEvals/DIM) for 51 targets with target precision in $10^{[-8..2]}$ for all functions and subgroups in 20-D. As reference algorithm, the best algorithm from BBOB 2009 is shown as light thick line with diamond markers.

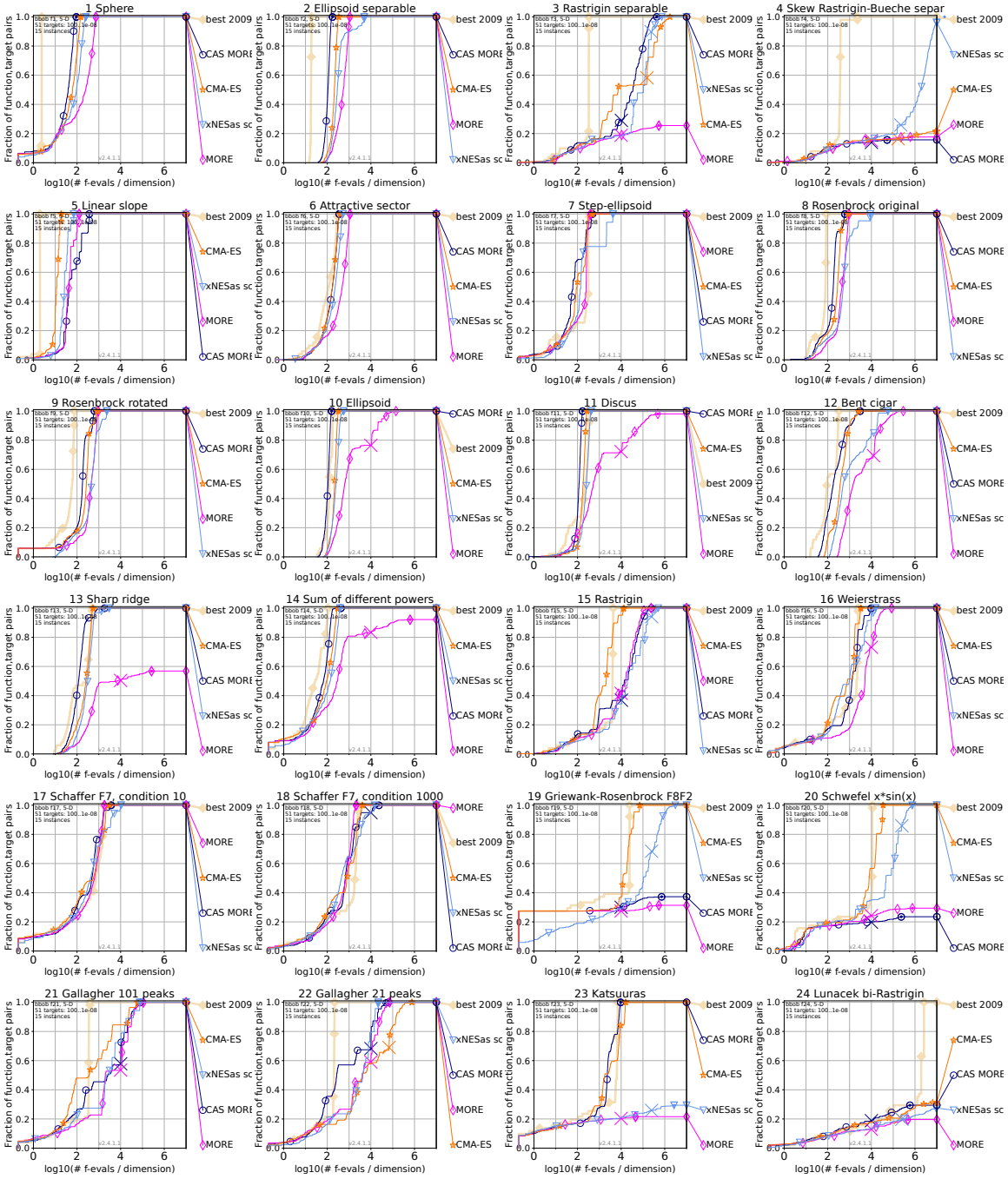


Figure B.5.: Empirical cumulative distribution of simulated (bootstrapped) runtimes, measured in number of objective function evaluations, divided by dimension (FEvals/DIM) for the 51 targets $10^{[-8..2]}$ in dimension 5.

C. Appendix for Chapter 6

C.1. PI VIPS Derivations

For the derivations, we use the following definitions and identities. The distribution we want to optimize is

$$\begin{aligned} q_\pi(\boldsymbol{X}) &= \sum_o q(\boldsymbol{X} \mid o) q(o) \\ &= \sum_o q(o) \int_Z q(\boldsymbol{X} \mid Z, o) q(Z \mid o) \mathrm{d}Z \\ &= |S(\boldsymbol{X})|^{-1} \sum_o q(o) \sum_{\tilde{\boldsymbol{X}} \in S(\boldsymbol{X})} q(Z = \tilde{\boldsymbol{X}} \mid o) \end{aligned}$$

where we use

$$\begin{aligned} q(\boldsymbol{X} \mid Z) &= \begin{cases} |S(\boldsymbol{X})|^{-1} & \text{if } Z \in S(\boldsymbol{X}) \\ 0 & \text{else} \end{cases} \\ q(\boldsymbol{X} \mid o) &= \int_Z q(\boldsymbol{X} \mid Z) q(Z \mid o) \mathrm{d}Z \\ &= |S(\boldsymbol{X})|^{-1} \sum_{\tilde{\boldsymbol{X}} \in S(\boldsymbol{X})} q(Z = \tilde{\boldsymbol{X}} \mid o) \end{aligned}$$

Identities:

$$\begin{aligned} \log q(\boldsymbol{X}) &= \log q(\boldsymbol{X} \mid Z) + \log q(Z) - \log q(Z \mid \boldsymbol{X}), \\ \log q(Z) &= \log q(Z \mid o) + \log q(o) - \log q(o \mid Z) \end{aligned}$$

We begin by writing down and reformulating the objective function

$$
\begin{aligned}
L_\pi &= \int_X q_\pi(\boldsymbol{X})(f(\boldsymbol{X}) - \rho \log q(\boldsymbol{X}))\mathrm{d}\boldsymbol{X} \\
&= \sum_o q(o) \int_X q(\boldsymbol{X} \mid o)(f(\boldsymbol{X}) - \rho \log q(\boldsymbol{X}))\mathrm{d}\boldsymbol{X} \\
&= \sum_o q(o) \int_X \int_Z q(\boldsymbol{X} \mid \boldsymbol{Z}) q(\boldsymbol{Z} \mid o)(f(\boldsymbol{X}) - \rho \log q(\boldsymbol{X}))\mathrm{d}\boldsymbol{Z}\mathrm{d}\boldsymbol{X} \\
&= \sum_o q(o) \int_X \int_Z q(\boldsymbol{X} \mid \boldsymbol{Z}) q(\boldsymbol{Z} \mid o)(f(\boldsymbol{X}) \\
&- \rho(\log q(\boldsymbol{X} \mid \boldsymbol{Z}) + \log q(\boldsymbol{Z} \mid o) + \log q(o) - \log q(o \mid \boldsymbol{Z}) \\
&- \log q(\boldsymbol{Z} \mid \boldsymbol{X})\mathrm{d}\boldsymbol{Z}\mathrm{d}\boldsymbol{X} \\
&= \sum_o q(o) \int_X \int_Z q(\boldsymbol{X} \mid \boldsymbol{Z}) q(\boldsymbol{Z} \mid o)(\\
f(\boldsymbol{X}) &- \rho(\log q(\boldsymbol{X} \mid \boldsymbol{Z}) - \log q(o \mid \boldsymbol{Z}) - \log q(\boldsymbol{Z} \mid \boldsymbol{X})\mathrm{d}\boldsymbol{Z}\mathrm{d}\boldsymbol{X} \\
&+ \rho H(q(\boldsymbol{Z} \mid o)) + \rho H(q(o))
\end{aligned}
$$

Adding and subtracting $\tilde{q}(o \mid \boldsymbol{Z})$ and $\tilde{q}(\boldsymbol{Z} \mid \boldsymbol{X})$, we rewrite the loss as

$$
\begin{aligned}
L_\pi = \tilde{L}(\tilde{q}(o \mid \boldsymbol{Z})) \\
&+ \rho \int_X q(\boldsymbol{X})\mathrm{KL}(q(\boldsymbol{Z} \mid \boldsymbol{X}) \parallel \tilde{q}(\boldsymbol{Z} \mid \boldsymbol{X}))\mathrm{d}\boldsymbol{X} \\
&+ \rho \int_X q(\boldsymbol{X})\mathrm{KL}(q(o \mid \boldsymbol{Z}) \parallel \tilde{q}(o \mid \boldsymbol{Z}))\mathrm{d}\boldsymbol{X}
\end{aligned}
$$

and obtain the variational lower bound

$$
\begin{aligned}
\tilde{L}(\tilde{q}(o \mid \boldsymbol{Z})) = \sum_o q(o) \int_X \int_Z q(\boldsymbol{X} \mid \boldsymbol{Z}) q(\boldsymbol{Z} \mid o)(\\
f(\boldsymbol{X}) + \rho(\log \tilde{q}(\boldsymbol{Z} \mid \boldsymbol{X}) + \log \tilde{q}(o \mid \boldsymbol{Z})) - \log q(\boldsymbol{X} \mid \boldsymbol{Z})))\mathrm{d}\boldsymbol{z}\mathrm{d}\boldsymbol{X} \\
+ \rho H(q(\boldsymbol{X} \mid o)) + \rho H(q(o))
\end{aligned}
$$

Using Bayes identities again, we can write

$$
\begin{aligned}
\tilde{L}(\tilde{q}(o \mid \boldsymbol{Z})) = \sum_o q(o) \int_X \int_Z & q(\boldsymbol{X} \mid \boldsymbol{Z}) q(\boldsymbol{Z} \mid o)(f(\boldsymbol{X}) \\
& + \rho(\underbrace{\log \tilde{q}(\boldsymbol{X} \mid \boldsymbol{Z})}_{= q(\boldsymbol{X} \mid \boldsymbol{Z})} + \underline{\log \tilde{q}(\boldsymbol{Z})} - \log \tilde{q}(\boldsymbol{X}) \\
& + \log \tilde{q}(\boldsymbol{Z} \mid o) + \log \tilde{q}(o) - \underline{\log \tilde{q}(\boldsymbol{Z})} \\
& - \underline{\log q(\boldsymbol{X} \mid \boldsymbol{Z})}))\mathrm{d}\boldsymbol{Z}\mathrm{d}\boldsymbol{X} \\
& + \rho H(q(\boldsymbol{X} \mid o)) + \rho H(q(o)) \\
= \sum_o q(o) \int_X \int_Z & q(\boldsymbol{X} \mid \boldsymbol{Z}) q(\boldsymbol{Z} \mid o)(f(\boldsymbol{X}) \\
& + \rho(\log \tilde{q}(\boldsymbol{Z} \mid o) - \log \tilde{q}(\boldsymbol{X}) + \log \tilde{q}(o)) \\
& + \rho H(q(\boldsymbol{X} \mid o)) + \rho H(q(o))
\end{aligned}
$$

Given the fact that $f(\boldsymbol{X})$ and $q(\boldsymbol{X})$ are permutation invariant, we can write

$$
\tilde{L}(\boldsymbol{\theta}) = \sum_o q(o) \int_Z q(\boldsymbol{Z} \mid o) R(\boldsymbol{Z}) \mathrm{d}\boldsymbol{Z} + \rho H(q(\boldsymbol{Z} \mid o)) + \rho H(q(o))
$$

with

$$
\begin{aligned}
R(\boldsymbol{Z}) &= \int_X q(\boldsymbol{X} \mid \boldsymbol{Z})(f(\boldsymbol{X}) + \rho(\log \tilde{q}(\boldsymbol{Z} \mid o) - \log \tilde{q}(\boldsymbol{X}) + \log \tilde{q}(o))\mathrm{d}\boldsymbol{X} \\
&= f(\boldsymbol{Z}) + \rho \left(\log \tilde{q}(\boldsymbol{Z} \mid o) - \log \tilde{q}(\boldsymbol{X} = \boldsymbol{Z}) + \log \tilde{q}(o)\right)
\end{aligned}
$$