TECHNISCHE UNIVERSITÄT WIEN

AUTOMATION & CONTROL INSTITUTE
INSTITUT FÜR AUTOMATISIERUNGS- & REGELUNGSTECHNIK

Local Cluster Experience Replay

DIPLOMARBEIT

Conducted in partial fulfillment of the requirements for the degree of a

Diplom-Ingenieur (Dipl.-Ing.)

supervised by

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. M. Vincze
Univ.Ass. Dipl.-Ing. Matthias Hirschmanner

submitted at the

TU Wien
Faculty of Electrical Engineering and Information Technology
Automation and Control Institute

by
Stefan Zahlner

Vienna, February 2023

Vision for Robotics Group
A-1040 Wien, Gusshausstr. 27, Internet: http://www.acin.tuwien.ac.at

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Preamble

This diploma thesis is submitted as the final work in the completion of studies in „Energie- und Automatisierungstechnik“. I want to take this opportunity to thank those who made this work possible and contributed to its creation.

First of all, I would like to thank my supervisor Matthias Hirschmanner. His encouragement and input throughout the thesis were invaluable and I am truly grateful for his willingness to share his knowledge and for his constant support. Thank you.

Furthermore, I would like to thank Professor Markus Vincze, through whose lectures I first became aware of the topic of this thesis, and thus he aroused my interest. He also gave me the opportunity to write this thesis in parallel to my regular work. Thank you.

Vienna, February 2023

Stefan Zahlner

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Abstract

Improving the sample efficiency of Reinforcement Learning (RL) algorithms plays a crucial role for their application in situations where data is scarce or expensive to collect. This thesis presents Local Cluster Experience Replay (LCER), an algorithm that aims to mitigate this problem by synthetic sample generation. LCER forms clusters within the replay-buffer of off-policy RL algorithms. It creates new and unseen state transitions by interpolating between samples from the same cluster, ensuring that interpolation only occurs on transitions that are adjacent in the state-action space. Conceptually, LCER creates locally linear models between different samples in the replay-buffer, allowing interpolation between various episodes and enhancing policy generalizability. We combine our approach with state-of-the-art RL algorithms and evaluate on continuous locomotive and continuous robotic control environments. LCER demonstrates significant improvement in sample efficiency over baseline RL algorithms in both environment domains. Additionally, LCER can effectively handle large and complex environments, making it a promising approach for improving the sample efficiency of a wide range of RL applications.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Kurzzusammenfassung

Die Verbesserung der Stichprobeneffizienz von Reinforcement Learning (RL) Algorithmen spielt eine entscheidende Rolle für deren Anwendung in Situationen, in denen Daten knapp oder schwer zu erheben sind. In dieser Arbeit wird mit Local Cluster Experience Replay (LCER) ein Algorithmus vorgestellt, der dieses Problem durch synthetische Stichprobengenerierung schmälert. LCER bildet Cluster innerhalb des Replay-Buffers von Off-Policy RL Algorithmen. Er erzeugt neue und ungesehene Stichproben durch Interpolation zwischen Übergängen aus demselben Cluster, wodurch sichergestellt wird, dass die Interpolation nur zwischen Zustandsübergängen erfolgt, die im Zustands-Aktionsraum benachbart sind. Konzeptionell erstellt LCER lokal lineare Modelle zwischen verschiedenen Übergängen im Replay-Buffer, die eine Interpolation zwischen verschiedenen Episoden ermöglichen und die Verallgemeinerbarkeit von Entscheidungsstrategien verbessern. Wir kombinieren unseren Ansatz mit modernen RL Algorithmen und evaluieren ihn in kontinuierlichen Fortbewegungs- und Robotersteuerungsumgebungen. LCER zeigt signifikante Verbesserungen in der Stichprobeneffizienz gegenüber RL Standardalgorithmen in beiden Umgebungsdomänen. Darüber hinaus ist LCER in der Lage, große und komplexe Umgebungen effektiv zu handhaben. Damit ist er ein vielversprechender Ansatz für die Verbesserung der Stichprobeneffizienz einer Vielzahl von RL Anwendungen.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Contents

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Problem Statement . . . 2
1.3 Contributions of this Thesis . . . 3
1.4 Chapter Organization . . . 4

2 Background **5**
2.1 Reinforcement Learning . . . 5
2.2 Types of Reinforcement Learning . . . 10
2.2.1 Model-Based and Model-Free Reinforcement Learning . . . 10
2.2.2 On-Policy and Off-Policy Reinforcement Learning . . . 11
2.3 Policy Gradient and Actor-Critic Algorithms . . . 12
2.3.1 Deep Deterministic Policy Gradient . . . 13
2.3.2 Soft Actor-Critic . . . 15
2.4 Experience and Replay-Buffer . . . 18
2.4.1 N-step Return . . . 19
2.4.2 Prioritized Experience Replay . . . 20
2.4.3 Neighborhood Mixup Experience Replay . . . 21
2.4.4 Hindsight Experience Replay . . . 22
2.5 MDP Dynamics Model . . . 22
2.5.1 Model-Based Policy Optimization . . . 23
2.6 Unsupervised Learning . . . 25
2.6.1 k-Means . . . 26

3 Related Work **27**

4 Local Cluster Experience Replay **31**
4.1 LCER Cluster Center . . . 31
4.2 LCER Random Member . . . 37
4.3 LCER and HER . . . 40
4.4 LCER and NMER . . . 40
4.5 Mixup Sampling and Interpolation . . . 41

5 Experiments **43**
5.1 Experimental Setup and Evaluation Metrics . . . 43
5.2 Continuous Locomotive Control Environments . . . 44
5.3 Continuous Robotic Control Environments . . . 50

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

5.4 Comparison of Computation-Time 55

6 Conclusion **60**
6.1 Implementation 60
6.2 Results 61
6.3 Future Work 62

A Software Components **69**

B Hyperparameters **70**

C Algorithms **73**
C.1 Deep Deterministic Policy Gradient 73
C.2 Soft Actor-Critic 74
C.3 Model-Based Policy Optimization 75
C.4 Prioritized Experience Replay 76
C.5 Neighborhood Mixup Experience Replay 77
C.6 Hindsight Experience Replay 78
C.7 Mini-batch k-Means 79

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

List of Figures

1.1 LCER from a higher-level perspective 3

2.1 Supervised and unsupervised learning use cases 5
2.2 Agent and environment interaction loop 6
2.3 Non-exhaustive taxonomy of modern RL algorithms 10
2.4 Model-free off-policy actor-critic RL 12
2.5 Backup diagrams of n-step methods 19
2.6 NMER nearest neighbors and mixup sampling 21
2.7 Model-Based Policy Optimization 24
2.8 Hierachical clustering dendrogram 25

4.1 LCER and its two variants LCER-CC and LCER-RM 32
4.2 LCER clusters in the state-action space 33
4.3 LCER-CC clusters and mixup sampling 35
4.4 LCER-RM clusters and mixup sampling 38

5.1 OpenAI Gym MuJoCo continuous locomotive control environments 45
5.2 OpenAI Gym continuous locomotive control environments evaluation results 46
5.3 OpenAI Gym MuJoCo and ShadowHand-Gym PyBullet continuous robotic control environments 51
5.4 OpenAI Gym MuJoCo and ShadowHand-Gym PyBullet robotic control environments evaluation results 52
5.5 Evaluation results of the computation-time comparison 57

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

List of Tables

5.1 Baseline RL algorithms used in the experiments 44
5.2 State and action space dimensions for the continuous locomotive control OpenAI Gym MuJoCo environments 45
5.3 `InvertedPendulum-v2` evaluation results 47
5.4 `Hopper-v2` evaluation results 47
5.5 `AntTruncated-v2` and `Walker2d-v2` evaluation results 48
5.6 `HalfCheetah-v2` evaluation results 48
5.7 State and action space dimensions for the continuous robotic control OpenAI Gym MuJoco and ShadowHand-Gym PyBullet environments 51
5.8 `FetchReach-v1` evalutaion results 53
5.9 `ShadowHandReach-v1` and `ShadowHandReachHard-v1` evaluation results 53
5.10 `ShadowHandBlock-v1` evaluation results 54
5.11 Evaluation results of the computation-time comparison 58

B.1 State and action space dimensions and maximum episode length for the OpenAI Gym MuJoco and ShadowHand-Gym PyBullet environments 70
B.2 OpenAI Gym continuous locomotive control environments evaluation hyperparameters 71
B.3 OpenAI Gym MuJoCo and ShadowHand-Gym PyBullet robotic control environments evaluation hyperparameters 72

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

List of Algorithms

4.1 Local Cluster Experience Replay Cluster Center (LCER-CC) 36
4.2 Local Cluster Experience Replay Random Member (LCER-RM) . . 39

C.1 Deep Deterministic Policy Gradient (DDPG) 73
C.2 Soft Actor-Critic (SAC) 74
C.3 Model-Based Policy Optimization (MBPO) 75
C.4 Prioritized Experience Replay (PER) 76
C.5 Neighborhood Mixup Experience Replay (NMER) 77
C.6 Hindsight Experience Replay (HER) 78
C.7 Mini-batch k-Means 79

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Acronyms

BFGS	Broyden–Fletcher–Goldfarb–Shanno
BNN	Bayesian Neural Network
CEM	Cross Entropy Method
CPU	Central Processing Unit
DDPG	Deep Deterministic Policy Gradient
FAISS	Facebook AI Similarity Search
GP	Gaussian Process
GPU	Graphics Processing Unit
HER	Hindsight Experience Replay
iLQG	iterative Linear Quadratic Gaussian
iLQR	iterative Linear Quadratic Regulator
k**-NN**	k-Nearest-Neighbor
LCER	Local Cluster Experience Replay
LCER-CC	Local Cluster Experience Replay Cluster Center
LCER-RM	Local Cluster Experience Replay Random Member
MBPO	Model-Based Policy Optimization
MBRL	Model-Based Reinforcement Learning
MC	Monte Carlo
MDP	Markov Decision Process
MFRL	Model-Free Reinforcement Learning
ML	Machine Learning
MPC	Model Predictive Control
NAF	Normalized Advantage Functions
NMER	Neighborhood Mixup Experience Replay
NN	Neural Network
RBF	Radial Bias Function
REDQ	Randomized Ensemble Double Q-Learning
PER	Prioritized Experience Replay
RL	Reinforcement Learning
SAC	Soft Actor-Critic
TD	Temporal Difference
TD3	Twin Delayed DDPG

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

1 Introduction

In modern society, data is often considered a valuable commodity. It drives many of the technological advancements we see today, from self-driving cars to personalized recommendations on e-commerce websites. However, collecting data can be time-consuming and costly, especially in situations where large amounts of data are required. This is where sample efficiency becomes important.

Sample efficiency refers to the ability of a learning algorithm to learn from a smaller amount of data. This is especially relevant in situations where data is scarce or hard to obtain, such as in real-time control tasks or in situations where data collection is expensive or impractical. With real hardware, data is extremely tedious to generate and the process is time-consuming, which often is infeasible for small research teams.

In Reinforcement Learning (RL), an agent learns to make decisions by performing actions in an environment and receiving rewards or penalties in return. Sample efficiency is a crucial factor in RL, because of the data-driven nature of the field. The RL agent must learn from a sufficient number of samples in order to accurately learn the optimal strategy for a given task. The larger the dataset required for training, the more computational resources are required, which can also be a limiting factor. Therefore, it is highly desirable to have an RL agent that is sample efficient, i.e., that can learn effectively with fewer samples.

1.1 Motivation

A major part of the existing RL methods are model-free and therefore, do not make any assumption about the underlying dynamics of the system. These methods are referred to as Model-Free Reinforcement Learning (MFRL). They are very flexible, as they can be applied to a wide variety of tasks and environments. However, this flexibility comes at a cost, as MFRL algorithms often require an immense amount of data to learn effectively.

One reason for this behavior is because model-free RL does not have a priori knowledge about the dynamics of the system it is learning from. It has to explore the environment extensively to learn the optimal policy. Another reason why model-free RL is data intensive is that MFRL often uses function approximation to represent the value function or policy. Function approximation can be very powerful, but it also requires a large number of samples to be effective. Despite sample inefficiency, learning through exploration and interaction with the environment

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

in particular often leads to good asymptotic performance, which is one of the advantages of model-free RL algorithms.

Model-Based Reinforcement Learning (MBRL) mitigates the need for training samples by maintaining a model of the system. This can be known or learned and predicts the next state of the environment, given the current state and an action. One application is to utilize the system model to generate synthetic data that can be used to train the policy. However, an additional model of the system also comes with its own set of challenges. For example, there is the additional computational effort required to learn and maintain the system model, which can be time-consuming and resource-intensive. Additionally, the system model is prone to errors, such as model uncertainty and model inaccuracy that can negatively impact the performance of the policy. Careful attention must be paid to these errors to ensure that the agent is able to accurately control the system.

Experience replay is another, powerful technique used in reinforcement learning to improve the sample efficiency and learning speed of the agent. For this purpose, past experiences are stored in a replay-buffer, so that the agent can learn from them. This allows past experiences to be reused and avoids the need to explore the same states multiple times, as they were generated while interacting with the environment. Thus, the agent can learn from a much larger number of experiences than it could by learning from just the most recent interactions alone. It also allows the agent to learn from a diverse set of experiences, rather than being limited to the most recent ones.

However, traditional experience replay, as used in almost all RL methods, has its limitations and can still be sample inefficient. This is because the agent may still need to explore a large number of states to learn effectively. In addition, the experiences stored in the replay-buffer may not always be representative of the current state of the environment. This can happen if the agent has not explored a wide enough range of states, or if the environment itself is changing over time.

1.2 Problem Statement

The goal of this thesis is to investigate ways to improve the sample efficiency of model-free RL, i.e., the agent's ability to learn from a smaller number of interactions with the environment. This is important because the training process in RL can be time-consuming and resource-intensive, and in some cases, it is not practical to wait for the agent to learn over a long period of time or to collect a large amount of data. By improving the sample efficiency of model-free RL, we aim to make the training process more efficient and practical for a wider range of applications while benefiting from good asymptotic performance.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

1.3 Contributions of this Thesis

In this thesis, we propose to extend the model-free RL concept by integrating Local Cluster Experience Replay (LCER) to increase its sample efficiency. LCER introduces k-Means clustering and mixup sampling (Zhang et al. [1]) to the replay-buffer of off-policy RL algorithms. It forms clusters within the replay-buffer and interpolates new, unseen environment interactions that are used in the training process of the RL agent. Figure 1.1 schematically depicts LCER from a higher-level perspective.

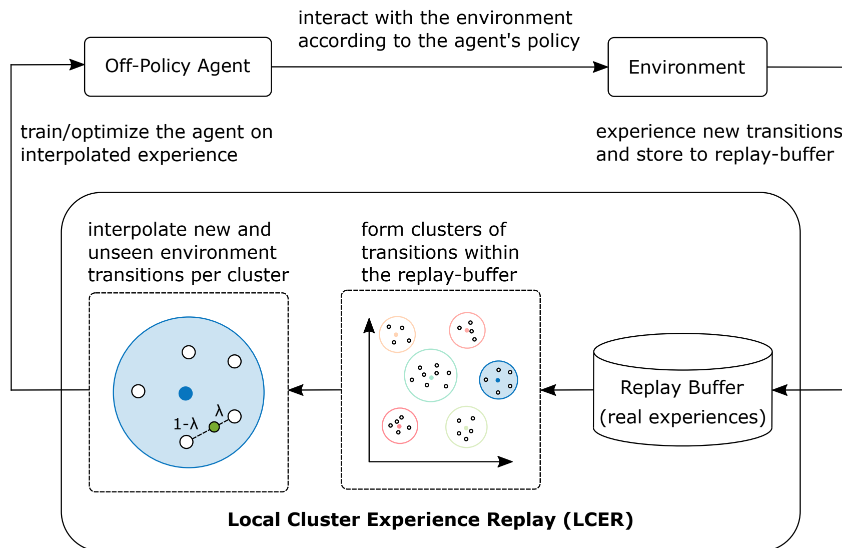

Figure 1.1: LCER from a higher-level perspective: Experience is generated by the off-policy RL agent while interacting with the environment. It is collected and stored in a replay-buffer. LCER forms clusters of transitions within the replay-buffer and interpolates new, unseen environment interactions. These are used to train the RL agent.

This approach combines the advantages of learning on real environment samples while exploiting the superior asymptotic performance of model-free RL. Conceptually, LCER is inspired by and therefore closely related to Neighborhood Mixup Experience Replay (NMER) (Sander et al. [2], [3]), although the concepts used are considerably different. The following points summarize the contribution of this thesis.

Sample Efficiency: The main focus of this thesis is to improve the sample efficiency of model-free RL algorithms, which refers to the ability to solve a given task using fewer interactions with the environment. This is important, as it allows for more efficient and cost-effective learning and decision-making processes. Improving sample efficiency can also have significant practical benefits, such as reducing the amount of data that needs to be collected and processed, or the amount of time and resources needed to train a model.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

RL and k-Means Clustering: LCER introduces clustering to the field of RL, in particular to the replay-buffer. In our implementation we utilize the k-Means clustering algorithm, which allows simple and fast grouping of adjacent environment interactions that are further used as interpolation pairs. To the best of our knowledge, this particular combination is a novel approach.

Intuitive Usage: This is important when it comes to implementing and using new algorithms. The additional overhead in terms of implementation effort and computational power can be a significant burden, especially for complex algorithms that require a considerable amount of resources to run. LCER achieves this requirement by design. It acts as a wrapper around the standard replay-buffer of off-policy RL algorithms, making it easy to integrate into existing RL workflows without having to make significant changes to the codebase.

Open Source: The code for Local Cluster Experience Replay is open source and freely available at `https://github.com/szahlner/lcer`.

1.4 Chapter Organization

Chapter 2 introduces the algorithms, methods, and concepts used in this thesis. Starting from the basic principles of RL, it describes actor-critic algorithms, a commonly used type of RL agents, which is also implemented and utilized in this thesis. Further, concepts on experience replay and replay-buffers are elaborated, followed by a section on dynamics models of the system and a section on k-Means clustering. Other approaches from the field of RL, especially concerning model-based RL, are discussed in Chapter 3. Chapter 4 describes the implementation of LCER and both of its variants. The experiments with LCER are presented in Chapter 5, as well as the obtained results. Chapter 6 concludes this thesis, and possibilities for future work are discussed.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2 Background

This chapter explains the methods, algorithms, and software components used in this thesis. Beginning with reinforcement learning, its classification into model-based and model-free approaches, the experience stored in replay-buffers, right up to world-models are explained and discussed, as well as the utilized RL algorithms. At the end, a section on unsupervised learning with a focus on clustering concludes this chapter. The following sections contain numerous statements that are based on Achiam [4], hence, not every reference is explicitly stated.

2.1 Reinforcement Learning

Reinforcement learning refers to a subset of Machine Learning (ML). Together with supervised and unsupervised learning, RL is one of the three main pillars of ML. Figure 2.1 illustrates the usage of supervised and unsupervised learning with typical use cases. In contrast to learning with a labeled training set (supervised learning) or pattern detection in unlabeled data (unsupervised learning), RL learns to act based on feedback.

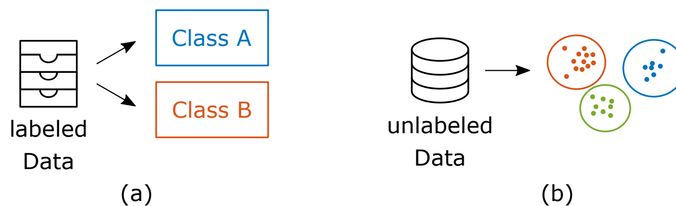

Figure 2.1: Supervised and unsupervised learning use cases: (a) object classification with already labeled data, (b) clustering similar data based on certain conditions.

RL is the study of agents, learning specific tasks by interacting with their environment. The idea is based on a reward system that emphasizes good behavior and punishes bad behavior. The goal is to learn a set of rules, a policy, that is used to decide which actions to take. For each interaction the environment returns an immediate reward, which can be used in the decision-making process of the agent, to maximize the cumulative sum of rewards. This is equivalent to optimizing the agent's sequential series of decisions to increase or decrease the probability of repeating or forgoing certain behavior in the future.

This work considers a standard RL setup consisting of an agent interacting with an environment in discrete timesteps as shown in Figure 2.2. At each timestep t, the agent receives an observation x_t and a scalar reward signal r_t. Based on the observation the agent takes an action a_t according to its learned policy[1].

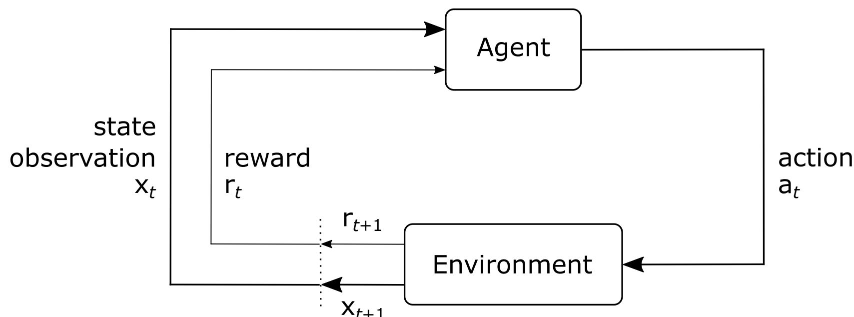

Figure 2.2: Interaction loop between agent and environment in RL: Based on the current observation x_t the agent takes an action a_t in the environment and receives back the next observation x_{t+1}, as well as a scalar reward signal r_{t+1}. Adapted from Sutton and Barto [5].

It is assumed that all environments are described by a Markov Decision Process (MDP) (Bellman [6], Puterman [7]) by the tuple $\{\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, p(s_0)\}$. The transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to p(\mathcal{S})$ is the probability of transitioning into state s_{t+1}, when starting in state s_t and taking action a_t. The reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ returns a scalar reward signal r_t of the current transition and $p(s_0)$ denotes the initial state distribution.

Starting with the first state $s_0 \sim p(s_0)$, the state s_1 is observed conditioned by choosing the action a_0 at timestep $t = 0$. Further, taking the action a_t the environment returns the next state $s_{t+1} \sim \mathcal{T}(\cdot|s_t, a_t)$ and the associated reward $r_t = \mathcal{R}(s_t, a_t, s_{t+1})$. An additional binary signal d_t, symbolizing the last transition in a sequential series of transitions can be added[2]. The transitions T can be denoted as quintuples

$$T = \Big\{s_t, a_t, s_{t+1}, r_t, d_t\Big\}, \tag{2.1}$$

where

[1]This also includes random actions that are used for exploration.
[2]This binary signal d_t does not distinguish between termination or truncation of the current transition series.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

$$
\begin{aligned}
s_t, s_{t+1} \in \mathcal{S} \quad &\text{and} \quad \mathcal{S} \subseteq \mathbb{R}^{d_{\text{obs}}}, \\
a_t \in \mathcal{A} \quad &\text{and} \quad \mathcal{A} \subseteq \mathbb{R}^{d_{\text{action}}}, \\
r_t \in \mathcal{R} \quad &\text{and} \quad \mathcal{R} \subseteq \mathbb{R}, \\
d_t \in \{0; 1\}&.
\end{aligned}
$$

Temporally connected transitions are also referred to as episodes, trajectories, or rollouts and represent the agent's experience, denoted as

$$
\tau = \left\{T_t\right\}_{t=0}^{N} = \left\{s_t, a_t, s_{t+1}, r_t, d_t\right\}_{t=0}^{N}, \tag{2.2}
$$

where $N \in \mathbb{N}$ is the amount of transitions in this particular rollout.

In addition, the states are assumed to be completely observable for all environments. Here $s_t = x_t$ is valid, different from the general case, where the MDP dynamics may be partially observable, so that the state can only be described by the entire history of the observation and action pairs $s_t = (x_1, a_1, ..., a_{t-1}, x_t)$.

The behavior of the agent is defined by its policy π. This is a set of rules that maps states to actions $\pi : \mathcal{S} \to \mathcal{A}$, whereby only real-valued actions are considered in this thesis. In reinforcement learning, the policies are parameterized. Thus, in order to solve a given task, these parameters need to be optimized accordingly.

To emphasize the policy's dependency on its parameters, it is often written with an appropriate subscript θ as

$$
a_t = \mu_\theta(s_t), \quad a_t \sim \pi_\theta(\cdot|s_t), \tag{2.3}
$$

where μ denotes a deterministic and π a stochastic policy. Typical examples of the parameters are the weights and biases of a Neural Network (NN).

The cumulative sum of rewards of a single rollout can be formalized as

$$
R(\tau) = \sum_{t=0}^{N} \gamma^t r_t, \tag{2.4}
$$

where N is the horizon of the environment, which denotes the maximal amount of actions to take. The discount factor $\gamma \in [0, 1]$ allows to apply weighting, and thus assigns greater importance to more recent rewards in terms of time.

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

The expected sum of rewards $J(\pi)$, refers to the expectation $\mathbb{E}\left[R(\tau)\right]$ of getting a specific cumulative sum of rewards by acting according to the policy π. This can be denoted as

$$J(\pi) = \mathop{\mathbb{E}}_{\tau\sim\pi}\Big[R(\tau)\Big] = \int_{\tau} P(\tau|\pi)R(\tau), \tag{2.5}$$

with $P(\tau|\pi)$ being the probability of a rollout with N timesteps. It is decomposed into a chain of probabilities by the MDP assumption, where the next action only depends on the current state and the next state only depends on the current state and action

$$P(\tau|\pi) = p_0(s_0)\prod_{t=0}^{N-1} P\Big(s_{t+1}|s_t, a_t\Big)\pi(s_t|a_t). \tag{2.6}$$

With the preceding equations, the core problem of reinforcement learning, namely maximizing the expected sum of rewards, or the cumulative reward, can be denoted as

$$\pi^* = \arg\max_{\pi} J(\pi), \tag{2.7}$$

where π^* is the optimal policy.

Depending on the utilized RL algorithm, there is also a value function $V : \mathcal{S} \to \mathbb{R}$ or action-value function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ respectively. It approximates the value of states or state-action pairs and is used to guide the policy towards a high cumulative reward. The value of a state or state-action pair is the expected sum of rewards, starting in that particular state or state-action pair, and acting according to the policy forever after.

The focus of this work is on reinforcement learning algorithms that use an action-value function $Q^{\pi}(s, a)$. Similar to the value defined before, the action-value is a measure of the expected cumulative reward, starting in state s, taking an arbitrary action[3] a and acting according to the learned policy π forever after, or at least until hitting the environment horizon N. This can be formalized as

$$\begin{aligned} Q^{\pi}(s,a) &= \mathop{\mathbb{E}}_{\tau\sim\pi}\Big[R(\tau)\;\Big|\; s_t = s, a_t = a\Big], \\ &= \mathop{\mathbb{E}}_{\tau\sim\pi}\Big[\sum_{t}^{N}\gamma^t r_t \;\Big|\; s_t = s, a_t = a\Big]. \end{aligned} \tag{2.8}$$

[3]This first action may not be according to the policy π. It can be completely random.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

The goal is to find an optimal policy π^* that maximizes the action-value function $Q^\pi(s,a)$

$$\pi^* = \arg\max_\pi Q^\pi(s,a), \tag{2.9}$$

which, in turn, is associated with the maximum sum of expected rewards.

Vice versa, the optimal action-value function $Q^*(s,a)$ can be defined as

$$Q^*(s,a) = \max_\pi \mathop{\mathbb{E}}_{\tau\sim\pi}\Big[\sum_t^N \gamma^t r_t \,\Big|\, s_t = s, a_t = a\Big]. \tag{2.10}$$

The reason why this work focuses on RL algorithms utilizing the action-value function $Q^\pi(s,a)$ is because a value function $V^\pi(s)$ cannot be used stand-alone to decide on a policy. It either needs a separate policy function $\pi(\cdot|s)$ for which it is used as a value function, or it is possible to derive a policy from the value function. However, the latter requires full access to the environment's distribution model $P(\tau|\pi)$ (see Equation 2.6), which is not available in many cases.

The relationship between the two Equations 2.8 and 2.10 is

$$Q^*(s,a) \geq Q^\pi(s,a). \tag{2.11}$$

It can be shown, that there is at least one optimal policy π^* which is better or equal to all other policies (Sutton and Barto [5]). Following this idea, it can be concluded, that there is at least one optimal action-value function $Q^*(s,a)$.

The Equations 2.8 and 2.10 obey a special self-consistency that allows the starting point to be moved arbitrarily. Its value is the expected reward for starting at this very point, plus the associated reward of the next state, given by the action of the policy. This fact is denoted in the Bellman equations (Bellman [6]) and can be formalized as

$$Q^\pi(s,a) = \mathop{\mathbb{E}}_{s_{t+1}\sim\mathcal{T}(\cdot|s_t,a_t)}\left[r(s,a) + \gamma \mathop{\mathbb{E}}_{a_{t+1}\sim\pi(\cdot|s)}\left[Q^\pi(s_{t+1},a_{t+1})\right]\right], \tag{2.12}$$

or, considering the optimal action-value function

$$Q^*(s,a) = \mathop{\mathbb{E}}_{s_{t+1}\sim\mathcal{T}(\cdot|s_t,a_t)}\left[r(s,a) + \gamma \max_{a_{t+1}} Q^*(s_{t+1},a_{t+1})\right], \tag{2.13}$$

with the right-hand side, namely the reward plus the next value, being the Bellman backup for a state-action pair.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2.2 Types of Reinforcement Learning

The landscape of modern reinforcement learning algorithms is very broad and it is hard to give an accurate taxonomy that is all-encompassing and contains all types and spin-offs. Figure 2.3 tries to give an overview as a tree-diagram.

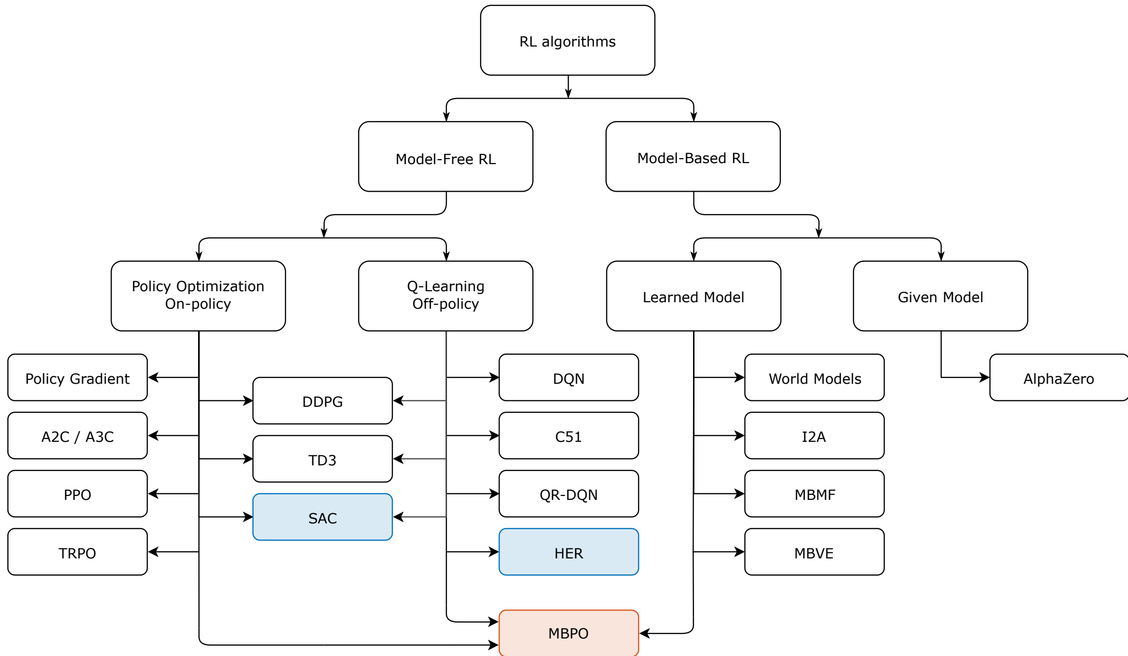

Figure 2.3: Non-exhaustive taxonomy of modern RL algorithms. The algorithms utilized in this thesis are marked blue and red. The red ones are used for comparison purposes only. Adapted from Achiam [4].

2.2.1 Model-Based and Model-Free Reinforcement Learning

One important classification characteristic of reinforcement learning algorithms is the distinction between model-based RL and model-free RL.

The main difference between MBRL and MFRL is the access to a model of the MDP dynamics. This is an additional function that predicts the transitions and their associated rewards. The model can be known (e.g., hard-coded) or it is learned. It enables the agent to query the MDP at any desired state-action pair without requiring a specific order, the access to the MDP dynamics is reversible.

Furthermore, model-based RL agents are able to generate their own sets of rollouts and transitions without any environmental interactions. MBRL algorithms can be distinguished into algorithms with a known or a learned model of the MDP dynamics.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- MBRL with a **known model**: The model can be queried right from the beginning of the training process. As an example, this can be achieved by an analytical and hard-coded solution of the MDP dynamics.

  Prominent representatives of this type of reinforcement learning are Expert Iteration (ExIt) (Anthony et al. [8]) and AlphaZero (Silver et al. [9]).

- MBRL with a **learned model**: The model is learned in the course of the training process, in contrast to the first type. A common choice to represent the model is a neural network that is optimized by minimizing the reconstruction loss. This is a measure that determines how well a model is able to reconstruct a given input.

  Examples of this type are Dyna (Sutton [10]) and World Models (Ha and Schmidhuber [11]).

Known models of the MDP dynamics are precise and accurate, however, they are limited in their usage, due to the scope of the analytical solution of the environment. Approaches with learned dynamics typically are confronted with the problem of model uncertainty. In a lot of cases, it is also not possible to learn a model of the complete dynamics, due to partial observability or non-stationarity. In addition, there are other issues that have to be addressed such as stochasticity, possible multi-step prediction, state-abstraction, and temporal abstraction as described in Moerland et al. [12].

In MFRL algorithms, the agent is tied to the specific order of the state-action pairs in which it visits them. Taking an action a_t in the state s_t leads to the next state s_{t+1}, whereby this trace in the environment can only be queried again in the exact same order. The RL agent has irreversible access to the MDP dynamics.

MFRL algorithms train the policy with the experiences given by its transitions and rollouts and thus need a large number of interactions with the environment.

2.2.2 On-Policy and Off-Policy Reinforcement Learning

Model-free RL algorithms can be further divided into agents that learn on-policy and agents that learn off-policy. The main difference between these two approaches is the experience that is being used to optimize the policy.

- **On**-policy RL: Data collected with the most recent version of the policy is used to train the agent (online).

  This subgroup includes algorithms such as Trust Region Policy Optimization (TRPO) (Schulman et al. [13]) and Proximal Policy Optimization (PPO) (Schulman et al. [14]).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- **Off**-policy RL: The collected experience is stored in a replay-buffer. This allows replaying the data at any point during the optimization. Thus, the available experience does also include trajectories sampled with older versions of the policy.

  Deep Q-Networks (DQN) (Mnih et al. [15]) and Categorical 51-Atom DQN (C51) (Bellemare et al. [16]) are two prominent algorithms from this group. Figure 2.4 depicts the off-policy reinforcement learning approach.

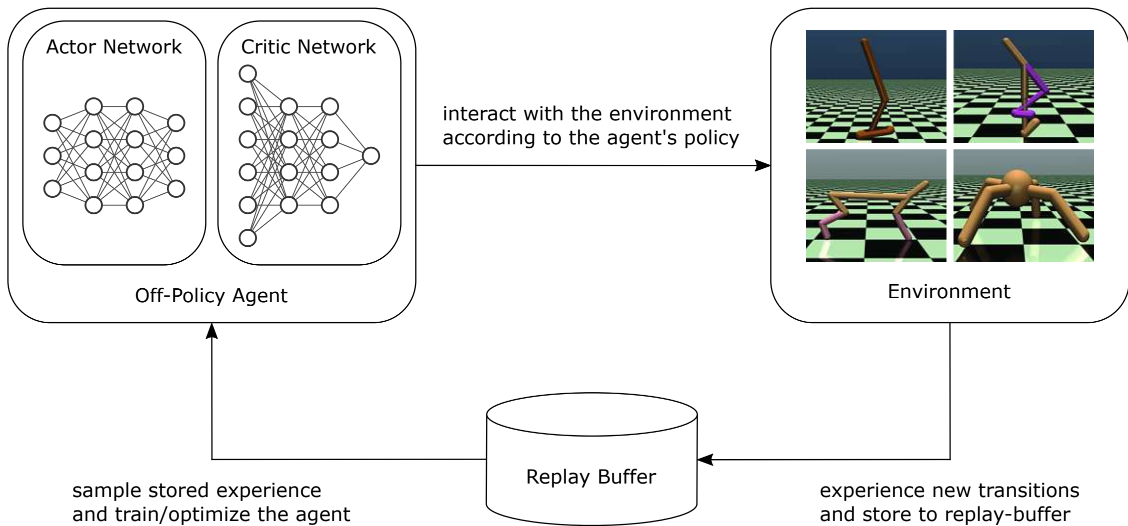

Figure 2.4: Model-free off-policy actor-critic reinforcement learning: The experience is generated by the agent while interacting with the environment. It is collected and stored in a replay-buffer. This replay-buffer allows the agent to replay the data at any point during the policy optimization process. The agent is realized in an actor-critic fashion, consisting of two networks, one for the actor and the other for the critic respectively.

Both on- and off-policy reinforcement learning algorithms have their strengths and weaknesses. Maintaining a replay-buffer is more sample efficient, but it only indirectly optimizes the agent because of the transitions taken from previous policy generations. On-policy RL algorithms directly optimize the agent but are less sample efficient (Achiam [4]).

2.3 Policy Gradient and Actor-Critic Algorithms

RL policies are parameterized, as aforementioned and formalized in Equation 2.3. The goal in reinforcement learning is to tune the parameters of the agent, or adjust the policy accordingly, to maximize the expected sum of environment rewards, the cumulative reward, $J(\pi_\theta)$ (see Equation 2.5).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Neural networks are a popular choice to represent the policy. NNs consist of a set of computational neurons that can approximate arbitrary functions (Goodfellow et al. [17], Schmidhuber [18]). This also includes nonlinear functions. The learning process is done by optimizing the weights and biases that connect the neurons, using training data from the interactions with the environment.

Policy gradient is one possible strategy to optimize the parameters. It executes a direct gradient update on the policy by performing gradient ascent on θ

$$\theta^{t+1} \leftarrow \theta^{t} + \alpha \, \nabla_\theta J\left(\pi_\theta\right)\big|_{\theta^t}\,, \tag{2.14}$$

with $\nabla_\theta J(\pi_\theta)$ being the gradient of the policy's performance.

This work focuses on locomotive and robotic control tasks with continuous and high dimensional state and action spaces, which makes it essential to choose appropriate RL algorithms. A common choice for this type of tasks are Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al. [19]), Twin Delayed DDPG (TD3) (Fujimoto et al. [20]) and Soft Actor-Critic (SAC) (Haarnoja et al. [21]). All of them are model-free, off-policy, and use policy gradient, as well as the particularly salient actor-critic implementation to optimize their agent.

Actor-critic learning is an RL technique that learns a value or action-value function ($V^\pi(s)$, $Q^\pi(s,a)$) in addition to the policy π. The policy represents the actor: It makes decisions and predicts the best actions according to the current environment state. The action-value function takes the role of the critic: It gives feedback, comprising the value of the state-action pair that helps to lead the expected sum of rewards to a maximum. Figure 2.4 shows the combination of actor and critic within the off-policy agent.

The actor-critic RL technique has shown great performance (Achiam [4], Raffin [22], Wang et al. [23]) and can be seen as a fusion between policy gradient and Q-learning. Therefore, a subset, namely DDPG and SAC are explored in further detail in the subsequent subsections below.

2.3.1 Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient is a model-free, off-policy reinforcement learning algorithm, which simultaneously learns a deterministic policy μ_θ and a Q-function value estimator (Achiam [4], Lillicrap et al. [19]).

Based on the Bellman Equation 2.12, DDPG learns to approximate the optimal action-value function $Q^*\left(s,a\right)$ with its Q-function value estimator, which, in turn, is used to learn and guide the policy towards a high cumulative reward[4].

[4] The influence of Q-learning is clearly evident: If the optimal action-value function $Q^*\left(s,a\right)$ is known, then the optimal action $a^*\left(s\right)$ can be found by solving $a^*\left(s\right) = \arg\max_a Q^*\left(s,a\right)$.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

DDPG in its original implementation is particularly adapted for continuous action spaces only. This allows computing the optimal action given a state to estimate the value

$$\max_a Q(s,a) \rightarrow Q(s,\mu(s)) . \tag{2.15}$$

Additionally to the actor-critic implementation, DDPG uses another RL technique, namely target networks. Target networks are an exact copy of the original network, but they are less frequently updated. They lag behind the original network, serving as a stable target. In the further course of this work, the actor, or policy, is parameterized by θ and the critic, or Q-function(s), by ϕ.

DDPG updates its targets utilizing polyak averaging according to

$$\begin{aligned} \theta_{\text{targ}} &\leftarrow \rho_{\text{actor}}\theta_{\text{targ}} + (1-\rho_{\text{actor}})\theta, \\ \phi_{\text{targ}} &\leftarrow \rho_{\text{critic}}\phi_{\text{targ}} + (1-\rho_{\text{critic}})\phi, \end{aligned} \tag{2.16}$$

where $\rho_{\text{actor}} \in (0,1)$ and $\rho_{\text{critic}} \in (0,1)$ are the polyak coefficients for the actor and critic respectively. They determine the amount of new information that should be considered during the target update.

DDPG optimizes its Q-function by minimizing the Mean-Squared Bellman Error

$$L_{\text{MSBE}}(\phi,\mathcal{D}) = \mathop{\mathbb{E}}_{(s,a,r,s',d)\sim\mathcal{D}}\left[\Big(Q_\phi(s,a) - y(r,s',d,a')\Big)^2\right], \tag{2.17}$$

with the target being

$$y(r,s',d,a') = r + \gamma(1-d)\max_{a'} Q_\phi(s',a'). \tag{2.18}$$

Because of the special self-consistency of the Bellman equations, it is possible to learn in an off-policy manner. Regardless of the specific state-action pairs, the Bellman equations should be satisfied.

The policy μ_θ is optimized by performing gradient ascent on

$$\max_\theta \mathop{\mathbb{E}}_{s\sim\mathcal{D}}\Big[Q_\phi\big(s,\mu_\theta(s)\big)\Big], \tag{2.19}$$

where the critic's parameters ϕ are treated as constants.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Since DDPG uses a deterministic policy, noise is added to the predicted actions of the actor during training to ensure a balance between exploration and exploitation of the state and action spaces. During evaluation, the noise is removed, which allows for measuring the actor's plain performance.

In DDPG, both the actor and the critic, as well as their target networks are implemented with neural networks. The weights and biases of the critic are optimized using stochastic gradient descent, according to

$$\phi^{(t+1)} \leftarrow \phi^{(t)} - \alpha_{\text{critic}} \nabla_\phi J_{\text{critic}}(\phi), \tag{2.20}$$

$$\nabla_\phi J_{\text{critic}}(\phi) = \nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d)\in B} \Big(Q_\theta(s,a) - y(r,s',d)\Big)^2, \tag{2.21}$$

where $y(r, s', d)$ is the target value, which is computed with the target networks resulting in

$$y(r,s',d) = r + \gamma(1-d) Q_{\phi_{\text{targ}}}\Big(s', \mu_{\theta_{\text{targ}}}(s')\Big). \tag{2.22}$$

The policy is updated similarly using stochastic gradient ascent

$$\theta^{(t+1)} \leftarrow \theta^{(t)} + \alpha_{\text{actor}} \nabla_\theta J_{\text{actor}}(\theta), \tag{2.23}$$

$$\nabla_\theta J_{\text{actor}}(\theta) = \nabla_\theta \frac{1}{|B|} \sum_{s\in B} Q_\theta\Big(s, \mu_\theta(s)\Big). \tag{2.24}$$

Algorithm C.1 depicts a pseudo-code implementation of DDPG.

2.3.2 Soft Actor-Critic

Soft Actor-Critic is a model-free, off-policy, and stochastic reinforcement learning algorithm, implemented in an actor-critic fashion. It utilizes entropy regularization, a squashing function, and the double-Q trick to ensure a robust learning process (Achaim [4], Haarnoja et al. [21]).

Entropy regularization is one of the core features of SAC. As a measure of randomness, it is closely related to the trade-off between exploration and exploitation, whereby a higher entropy results in more exploration.

Let x be a random variable with probability mass or density function P, the entropy H of x is computed from its distribution P according to

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

$$H(P) = \mathop{\mathbb{E}}_{x \sim P}\Big[-\log P(x) \Big]. \tag{2.25}$$

In SAC, the RL agent receives an extra reward at each timestep that is proportional to the entropy of the policy at the current timestep[5]. Thus, the reinforcement learning problem (see Equation 2.7) changes to

$$\pi^* = \arg\max_{\pi} \mathop{\mathbb{E}}_{\tau \sim \pi} \left[\sum_{t=0}^{N} \gamma^t \Big(R(s_t, a_t, s_{t+1}) + \alpha H\big(\pi(\cdot|s_t)\big) \Big) \right], \tag{2.26}$$

where $\alpha \in \mathbb{R}^+$ denotes the entropy regularization coefficient. It is responsible for the trade-off between encouraging exploration and return optimization.

The action-value function $Q^\pi(s,a)$ also includes the entropy bonus for every timestep, except for the first[6]. This can be formalized as

$$Q^\pi(s,a) = \mathop{\mathbb{E}}_{\tau \sim \pi} \left[\sum_{t=0}^{N} \gamma^t R(s_t, a_t, s_{t+1}) + \alpha \sum_{t=1}^{N} \gamma^t H\big(\pi(\cdot|s_t)\big) \Big| s_0 = s, a_0 = a \right]. \tag{2.27}$$

Unlike DDPG, SAC learns two Q-functions instead of one and uses the smaller of the two Q-values to form the target in the Mean-Squared Bellman Error

$$L_{\text{MSBE}}(\phi_i, \mathcal{D}) = \mathop{\mathbb{E}}_{(s,a,r,s',d) \sim \mathcal{D}} \left[\Big(Q_{\phi_i}(s,a) - y(r,s',d) \Big)^2 \right], \tag{2.28}$$

with the target being

$$y(r, s', d) = r + \gamma(1-d) \left(\min_{j=1,2} Q_{\phi_{\text{targ},j}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s') \right), \quad \text{with} \quad \tilde{a}' \sim \pi_\theta(\cdot|s'). \tag{2.29}$$

Using the smaller Q-value for the target and progressing towards that value, helps to avoid overestimating the Q-function.

Additionally, SAC utilizes a squashing function, namely $\tanh(\cdot)$, to compress the output of the policy to the finite range $(-1, 1)$. However, this also changes the

[5] Generally, this applies to all entropy-regularized RL algorithms.

[6] The entropy bonus is not added to the first timestep, since the first action can be arbitrary and does not need to be according to the policy.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

distribution of the policy output from being a factored Gaussian before the squashing function, to a clipped, distorted Gaussian after the $\tanh(\cdot)$.

The fourth trick being used in Soft Actor-Critic is called the reparameterization trick. It allows bypassing the dependence of the expectation over actions on the policy parameters as given by

$$\mathop{\mathbb{E}}_{a\sim\pi_\theta}\Big[Q^{\pi_\theta}(s,a) - \alpha \log \pi_\theta(a|s)\Big]. \tag{2.30}$$

SAC draws actions from the policy $\pi_\theta(\cdot|s)$ according to

$$\mu_\theta(s) + \sigma_\theta(s) \odot \xi, \quad \text{with} \quad \xi \sim \mathcal{N}(0, I), \tag{2.31}$$

where $\mu_\theta(s)$ and $\sigma_\theta(s)$ denote measures for the mean and standard deviation of the action, depending on the state and with respect to the policy parameters. ξ is sampled from a Gaussian distribution.

Based on the preceding facts, an action $\tilde{a}_\theta(s,\xi)$, drawn from $\pi_\theta(\cdot|s)$, can thus be notated as

$$\tilde{a}_\theta(s,\xi) = \tanh\Big(\mu_\theta(s) + \sigma_\theta(s) \odot \xi\Big). \tag{2.32}$$

This allows to rewrite the expectation over actions into an expectation over noise, finally bypassing the dependence on the policy parameters

$$\mathop{\mathbb{E}}_{a\sim\pi_\theta}\Big[Q^{\pi_\theta}(s,a) - \alpha \log \pi_\theta(a|s)\Big] = \mathop{\mathbb{E}}_{\xi\sim\mathcal{N}}\Big[Q^{\pi_\theta}\Big(s,\tilde{a}_\theta(s,\xi)\Big) - \alpha \log \pi_\theta\Big(\tilde{a}_\theta(s,\xi)\Big|s\Big)\Big], \tag{2.33}$$

leading to the target according to which the policy is optimized

$$\max_\theta \mathop{\mathbb{E}}_{s\sim\mathcal{D},\xi\sim\mathcal{N}}\left[\min_{j=1,2} Q_{\phi_j}\Big(s,\tilde{a}_\theta(s,\xi)\Big) - \alpha \log \pi_\theta\Big(\tilde{a}_\theta(s,\xi)\Big|s\Big)\right]. \tag{2.34}$$

During the training process, both critic networks are optimized using stochastic gradient descent

$$\phi^{(t+1)} \leftarrow \phi^{(t)} - \alpha_{\text{critic}} \nabla_\phi J_{\text{critic}}(\phi), \tag{2.35}$$

$$\nabla_\phi J_{\text{critic}}(\phi) = \nabla_\theta \frac{1}{|B|} \sum_{(s,a,r,s',d)\in B} \Big(Q_\theta(s,a) - y(r,s',d)\Big)^2, \quad \text{for} \quad i = 1{,}2, \tag{2.36}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

with the Q-function target, $y(r, s', d)$, according to Equation 2.29.

The policy is updated similarly using stochastic gradient ascent

$$\theta^{(t+1)} \leftarrow \theta^{(t)} + \alpha_{\text{actor}} \nabla_\theta J_{\text{actor}}(\theta), \tag{2.37}$$

$$\nabla_\theta J_{\text{actor}}(\theta) = \nabla_\theta \frac{1}{|B|} \sum_{(s)\in B} \left(\min_{i=1,2} Q_{\phi_i}\Big(s, \tilde{a}_\theta(s)\Big) - \alpha \log \pi_\theta\Big(\tilde{a}_\theta(s)\Big|s\Big)\right). \tag{2.38}$$

Algorithm C.2 depicts a pseudo-code implementation of SAC.

2.4 Experience and Replay-Buffer

Experience replay is one of the core components of off-policy reinforcement learning algorithms. The ability to store and reuse previous transitions proves to be sample efficient and also enhances the stability of the training process (Sander et al. [2], Fedus et al. [24]).

The experience from the interactions with the environment is stored in a replay-buffer. This is usually implemented in the form of a ring buffer with a fixed maximum number of stored transitions. When the buffer is full, the oldest transition is overwritten, thus, a cycle is generated to guarantee a balance between old and new data. This concept also takes into account that the agent perpetually explores the environment with its actions, generating new data to learn with every single interaction.

An important property of replay-buffers in combination with off-policy RL algorithms is the possibility to use high update-to-data ratios (Chen et al. [25]). This allows doing multiple policy updates per environment transition. Typically the number of policy gradient steps per environment sample is 1 or 2. In most cases, however, the replay-buffer contains a large number of transitions, even though they were generated with older policy versions. The usage of higher update-to-data ratios without any additional mechanisms generally results in unstable training behavior. However, with appropriate mechanisms and algorithms, this works well and can improve the sample efficiency of the learning process. Prominent examples, that make use of this fact are Neighborhood Mixup Experience Replay (Sander et al. [2]), Randomized Ensemble Double Q-Learning (REDQ) (Chen et al. [25]) and Model-Based Policy Optimization (MBPO) (Janner et al. [26]).

In the context of the replay-buffer, there are several approaches that try to improve the training process of reinforcement learning. A handful of promising examples include Prioritized Experience Replay (PER) (Schaul et al. [27], Horgan et al. [28]), n-step return (Sutton and Barto [5], Fedus et al. [24]), Neighborhood Mixup

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Experience Replay (Sander et al. [2]) and Hindsight Experience Replay (HER) (Andrychowicz et al. [29]). Due to their strong performance in a wide range of continuous state and action locomotive and robotic environments, these RL algorithms are briefly outlined below. HER is particularly noteworthy because it is one of a few algorithms that can deal with sparse rewards, as described in Plappert et al. [30].

2.4.1 N-step Return

N-step return belongs to Temporal Difference (TD) learning and addresses the question of how many steps to include in the value function estimation during an update in Q-learning.

Considering the Bellman Equation 2.12, n-step methods target the Bellman backup, i.e., the right-hand side of the equation. They determine how many steps per update are to be included and range within the interval $[1, \infty) \in \mathbb{N}$, or at least the environment horizon.

1-step TD methods perform the update for each state using the next reward and the estimated value of the next state as an approximation for the following rewards. This can be generalized and denoted as

$$G_t^n = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... + \gamma^n Q^\pi(s_{t+n}, a_{t+n}). \tag{2.39}$$

Monte Carlo (MC) methods, however, perform the update using the entire sequence of rewards that are observed during a rollout. In this sense 1-step TD methods can be thought of using shallow backups and MC methods using deep backups as described in Sutton and Barto [5]. Figure 2.5 illustrates this difference.

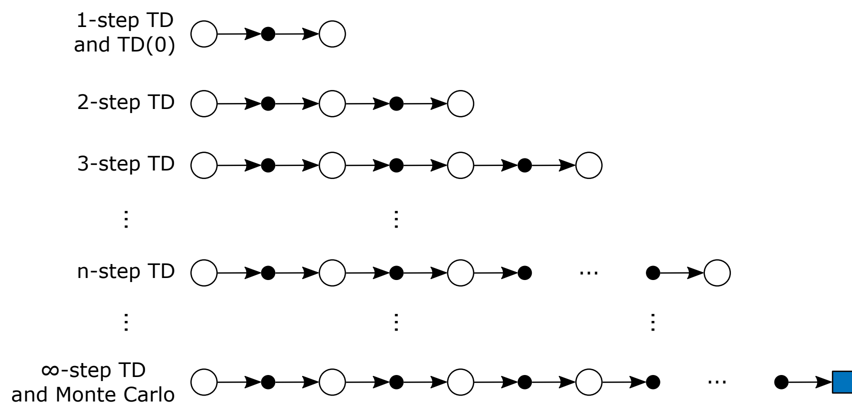

Figure 2.5: Backup diagrams of n-step methods. These methods range from 1-step TD methods to Monte Carlo methods. Adapted from Sutton and Barto [5].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2.4.2 Prioritized Experience Replay

Prioritized Experience Replay is based on the idea of differently important experiences. It addresses the prioritization of important transitions, and thus the more frequent selection of them. This is in contrast to standard sampling, which is uniformly random.

Using PER, the probability $P(n)$ of replaying a particular transition n from the replay-buffer is proportional to its priority $p_n > 0$ according to

$$P(n) = \frac{p_n^\alpha}{\sum_k p_k^\alpha}, \tag{2.40}$$

where α denotes how much prioritization is used. $\alpha = 0$ corresponds to the uniform case, as described by Schaul et al. [27].

The priority p_n is commonly set with respect to the magnitude of the transition's TD error δ_n and a small constant $\epsilon > 0$, to ensure the possibility of resampling this particular transition

$$p_n = |\delta_n| + \epsilon. \tag{2.41}$$

In Q-learning, the correlations of the observations are removed by sampling uniformly random from the replay-buffer. Further, it is assumed that the optimal value estimator is updated with data from the same distribution as the expectation. Using PER changes this distribution in an uncontrolled fashion and may also establish correlations in the state space. It introduces bias because it does not sample transitions uniformly at random due to the sampling proportion corresponding to the TD error (Schaul et al. [27]).

To circumvent this bias, PER utilizes importance sampling weights according to

$$w_n = \left(\frac{1}{N} \cdot \frac{1}{P(n)}\right)^\beta, \tag{2.42}$$

with N being the number of transitions used in an update and β denotes how much to compensate for the non-uniform sampling. Choosing $\beta = 1$ results in full compensation.

These weights are further utilized in the Q-learning update by using $w_n \delta_n$ instead of δ_n. Algorithm C.4 depicts a pseudo-code implementation of PER.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2.4.3 Neighborhood Mixup Experience Replay

Neighborhood Mixup Experience Replay is a strategy for efficiently using experiences stored in the replay-buffer of a model-free and off-policy RL agent. It creates locally linear models around different transitions by interpolating between nearby samples using mixup sampling (Sander et al. [2]).

Mixup sampling introduces new experiences as a linear combination of existing transitions (Zhang et al. [1], Sander et al. [2]). They are generated according to

$$x_{\text{interpolated}} = \lambda x_{\text{sample}} + (1 - \lambda) x_{\text{neighbor}}, \tag{2.43}$$

where λ is sampled from a β-distribution or uniformly chosen from the interval $[0, 1)$. x represents each member of the transition quintuple, except for the terminal flag d_t, as they are all joined by the same factor.

NMER ensures the proximity of the transitions used in Equation 2.43, based on their L2 distance, measured in the dimensions of the standardized state-action space, further called z-space. Figure 2.6 depicts the z-space, as well as the neighborhood around a sample transition.

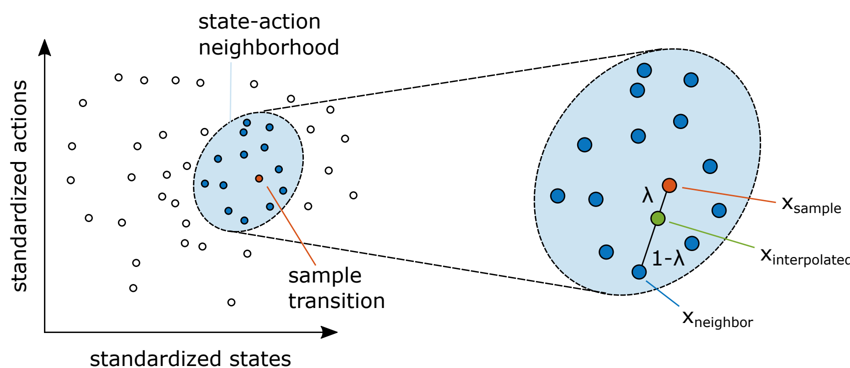

Figure 2.6: NMER nearest neighbors and mixup sampling. The nearest transition in the replay-buffer is chosen according to the nearest neighbors in the z-space (standardized state-action coordinates). A new transition is created by interpolation using mixup sampling. Adapted from Sander et al. [2], [3].

This replay strategy introduces new, unseen experiences to the agent and allows the use of high update-to-data ratios without making the training unstable. In addition, this mechanism can prevent the agent from overfitting to a particular transition outcome. By using different nearest neighbors, near identical inputs produce different outcomes (Sander et al. [3]).

Algorithm C.5 depicts a pseudo-code implementation of NMER.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2.4.4 Hindsight Experience Replay

Hindsight Experience Replay follows the approach of Universal Value Function Approximators (Schaul et al. [31]) and optimizes a model-free, off-policy agent to achieve multiple different goals[7]. This presupposes that the policies and action-value functions additionally take a goal $g \in \mathcal{G}$ as input, which can be formalized as $\pi : \mathcal{S} \times \mathcal{G} \to \mathcal{A}$ and $Q : \mathcal{S} \times \mathcal{G} \times \mathcal{A} \to \mathbb{R}$ respectively.

HER assumes that each goal $g \in \mathcal{G}$ corresponds to a predicate $f_g : \mathcal{S} \to \{0, 1\}$, whereby it is the agent's goal to get to any state that satisfies $f_g(s) = 1$. Further, this can be generalized into a mapping from states to goals

$$m : \mathcal{S} \to \mathcal{G} \text{ s.t. } \forall_{s \in \mathcal{S}} \; f_{m(s)}(s) = 1, \tag{2.44}$$

to find a goal g according to a given state s that yields in $f_g(s) = 1$.

Hindsight Experience Replay can be combined with any off-policy RL algorithm, as long as it maintains a replay-buffer. Additionally to storing the experience generated by the agent while interacting with the environment, HER also stores the same experience with a subset of different goals. This procedure proves to be valid, since the intended goal influences the agent's actions, but not the MDP dynamics of the environment.

A crucial design parameter of HER is the set of additional goals used for the experience replay. Following the simplest version of HER, each member of a rollout gets assigned the goal that is achieved in the final state of this very episode, as explained by Andrychowicz et al. [29].

Algorithm C.6 depicts a pseudo-code implementation of HER.

2.5 MDP Dynamics Model

The following section covers model-based reinforcement learning with learned models. Here, a model of the environment is learned in the course of the training process and further represented as a neural network.

Generally, learning a model of the environment's MDP dynamics is a supervised learning problem. It is based on learning transitions from observed data. Given the transition quintuple $\{s_t, a_t, s_{t+1}, r_t, d_t\}$, the model learns to predict the next state s_{t+1} starting from the current state s_t and taking action a_t. Additionally, it is assumed that the reward function is unknown and, therefore, is also predicted as a function of the current state and action. The resulting forward model can

[7] This also includes the special case, where there is only one goal.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

be formalized as $\mathcal{M} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S} \times \mathbb{R}$. For the scope of this thesis, the terminal function is considered to be known. Hence, d_t does not need to be predicted by the model.

A common choice to implement the model of the MDP dynamics is a neural network. This mainly stems from the fact that NNs learn quickly and impress with outstanding performance (Janner et al. [26], Nagabandi et al. [32], Chua et al. [33]). Another quite popular possibility to represent the environment dynamics are Gaussian Processes (GPs) (Sutton and Barto [5], Wang et al. [34], Gadd et al. [35]), even though they are not considered in the context of this thesis.

In the remainder of this work, the environmental model structure as proposed in Model-Based Policy Optimization (Janner et al. [26]) is utilized as a base framework to compare.

2.5.1 Model-Based Policy Optimization

Model-Based Policy Optimization (Janner et al. [26]) is a model-based, off-policy RL algorithm that uses SAC to optimize its policy. Additionally, it utilizes probabilistic neural networks as an MDP dynamics model, which outputs a parameterized Gaussian distribution with diagonal covariance. This can be formalized as

$$\mathcal{M}(\cdot|s_t, a_t) = \mathcal{N}\left(\mu(s_t, a_t), \sum(s_t, a_t)\right), \tag{2.45}$$

resulting in

$$\widetilde{s}_{t+1}, \widetilde{r}_t \sim \mathcal{M}(\cdot|s_t, a_t), \tag{2.46}$$

where μ denotes the mean value, $\sum$ refers to a diagonal covariance matrix and $\widetilde{s}_{t+1}$, $\widetilde{r}_t$ are the predicted next states and the associated rewards.

MBPO is designed and implemented, to deal with the problem of model uncertainty. On the one hand, it accounts for aleatoric uncertainty, which is the noise in the outputs with respect to the inputs. On the other hand, it addresses the problem of epistemic uncertainty, or the uncertainty in the model parameters, which is particularly dominant in regions where data is scarce.

To cope with the latter problem, the dynamics model of MBPO utilizes a bootstrap ensemble of neural networks $\mathcal{M} = \{\mathcal{M}^1, ..., \mathcal{M}^n\}$, whereby any model of the ensemble members is selected uniformly at random to make a prediction. This allows sampling transitions from different dynamics models during a single rollout.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

The probabilistic settings of each individual member of the bootstrap ensemble account for the aleatoric uncertainty.

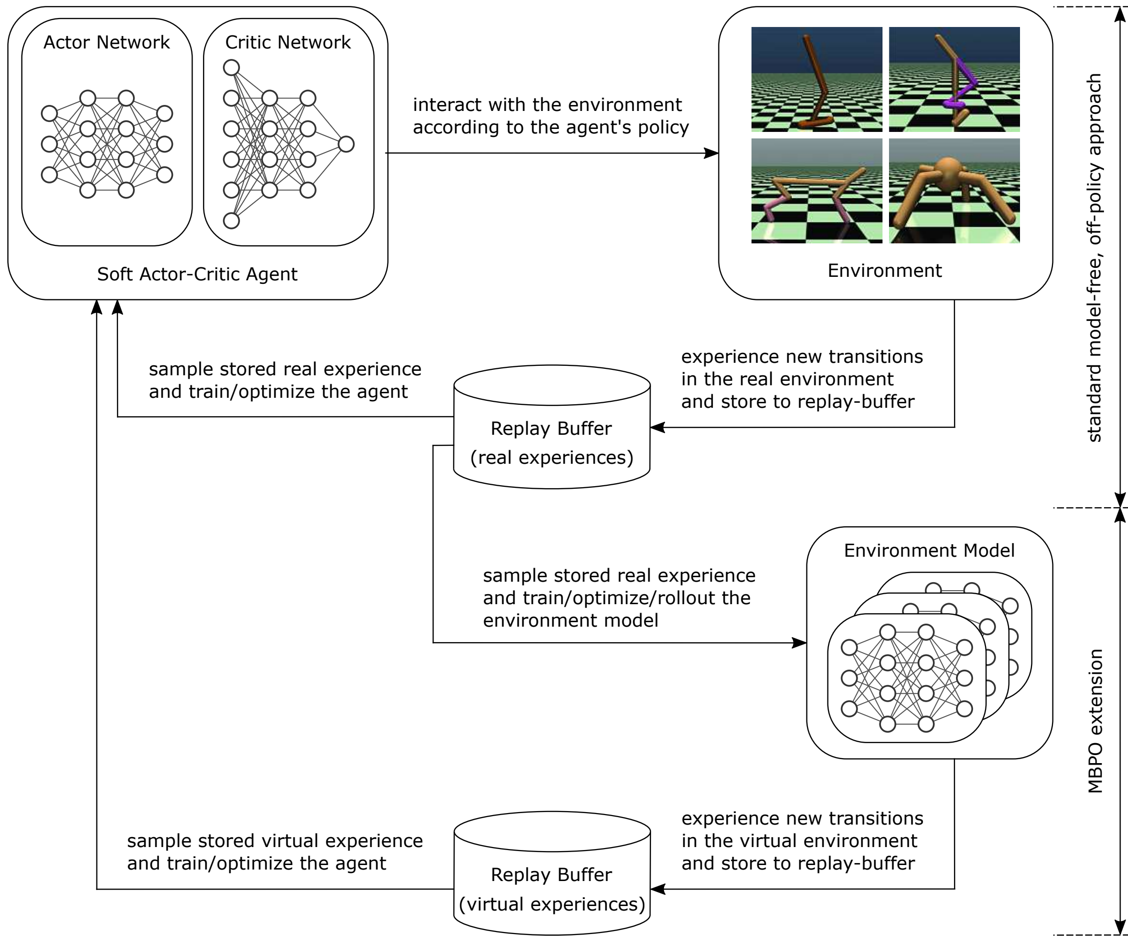

Figure 2.7: Model-Based Policy Optimization: The experience, generated by the agent while interacting with the environment, is collected and stored in a replay-buffer. Based on this data, the model of the environment's MDP dynamics is optimized. This model is used to perform virtual rollouts, starting from a set of arbitrary real states. During the agent's optimization routine, real and virtual experiences are used.

Model-Based Policy Optimization utilizes its MDP dynamics model to generate artificial rollouts. MBPO starts from real states, sampled from the replay-buffer, and performs many short horizon rollouts to circumvent large cumulative errors from extended virtual trajectories. This strategy yields a large amount of additional experience that can be used in the SAC policy optimization routine. In fact, it allows taking many more policy gradient steps per real environment sample than is typically stable in model-free RL algorithms. Typically the number of policy gradient steps

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

per real environment sample is 1 or 2, whereby MBPO uses between 10 and 40 (Sutton and Barto [5], Chen et al. [25], Janner et al. [26]). Figure 2.7 schematically depicts the implementation and inner processes of MBPO and Algorithm C.3 shows a pseudo-code implementation.

2.6 Unsupervised Learning

Unsupervised learning is another important branch of machine learning besides RL. It is used to detect patterns and structures in unlabeled data. The most popular unsupervised learning algorithm is clustering, wherein data points are grouped into different sets or clusters based on their degree of similarity.

On a higher-level description, the various types of clustering are hierarchical clustering and partitioning clustering. The first one uses tree-like structures and can be further divided into agglomerative and divisive approaches. Figure 2.8 shows a visual representation of both methods.

- **Agglomerative** clustering is a bottom-up approach. It starts by splitting the dataset into singleton nodes, sequential merging them into bigger nodes based on their mutual distance, until there is only one node left, representing the entire dataset (Müllner [36]).
- **Divisive** clustering is a top-down approach. Initially, the entire dataset belongs to one single node. Moving down the hierarchy, splits are performed until the dataset is partitioned into singleton nodes (Roux [37]).

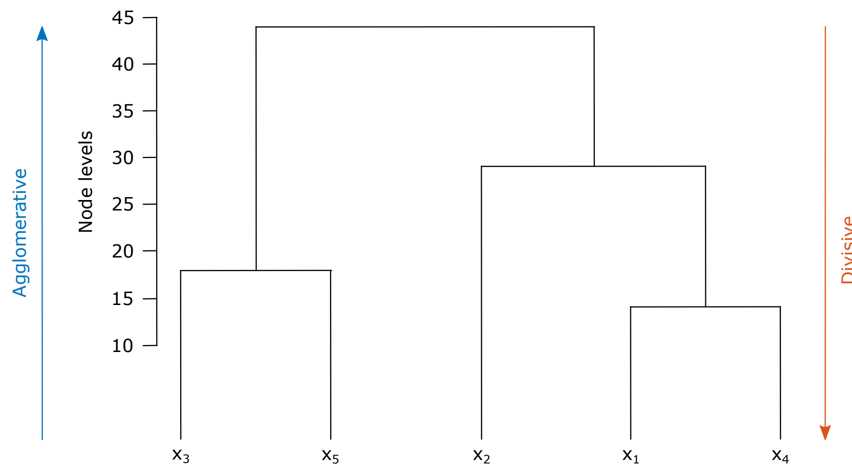

Figure 2.8: Hierachical clustering dendrogram: Agglomerative clustering (blue) as a bottom-up approach and divisive clustering (orange) as a top-down approach. Adapted from Roux [37].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Partitioning clustering classifies the information within the data into multiple groups right from the beginning. These groups are based on the characteristics and similarity of the data, where each partition represents a cluster and its particular region. Depending on the clustering algorithm, the number of partitions can either be computed or chosen by the user. Prominent examples of these unsupervised learning algorithms are k-Means (Agarwal et al. [38], Arthur and Vassilvitskii [39]) and k-Medoids (Newling and Fleuret [40]). This work sets its focus on the k-Means algorithm, which will be outlined further.

2.6.1 k-Means

The k-Means problem is to find a set C with $|C| = k$ cluster centers $cc \in \mathbb{R}^d$ and their associated partitions of the finite dataset $X \subseteq \mathbb{R}^d$, with d denoting the numbers of features. The cluster centers follow the objective function

$$\min \sum_{x \in X} \left|\left| f(C, x) - x \right|\right|^2, \tag{2.47}$$

where $f(C, x)$ returns the nearest cluster center $cc \in C$ to x, measured with the Euclidean distance (Sculley [41]).

The most popular implementation of k-Means is Llolyd's algorithm (Lloyd [42]). It consists of an assignment step and an update step, which are executed alternately. In the update step, each cluster center cc is set to the mean of the partition assigned to it. During the assignment step, the centers are frozen and each new data point is assigned to its nearest cluster center. Llolyd's algorithm is considered to be a local algorithm, where distant centers do not affect each other. Therefore, it has a tendency to terminate in poor minima if not well initialized (Newling and Fleuret [40]).

This work is primarily concerned with the field of reinforcement learning. Therefore, already existing implementations of the k-Means algorithm, provided by and based on the scikit-learn library from Pedregosa et al. [43], are utilized. For a detailed examination of the initialization method, reference is made to more in-depth literature (Arthur and Vassilvitskii [39], Newling and Fleuret [40]).

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

3 Related Work

The previous chapter covers all methods, algorithms, and software components that are utilized in this thesis. Further, their basic components and derivations are described and a possible classification of reinforcement learning is included. In contrast, other approaches from the field of RL, especially concerning model-based RL, are explained in this chapter. Since they are not implemented in this work, they are not discussed in the same detail.

Starting with model-based reinforcement learning and its various applications, the joint usage of MBRL and model-free RL, also known as hybrid approach, is addressed. In the subsequent course, different combinations of MBRL with planning aspects are described. At the end, possible problems regarding model-based reinforcement learning are discussed, followed by a final section on Randomized Ensemble Double Q-Learning (Chen et al. [25]), a promising, fast learning and model-free RL algorithm.

Although model-based RL has shown great potential in recent years, this strategy dates back quite a while, e.g. Miller et al. [44] and Schmidhuber [45]. In general, MBRL has many possible advantages compared to model-free reinforcement learning approaches. The main focus of this context is on sample efficiency, planning aspects, and the ability to generalize to new tasks in the environment, without having to retrain the agent (Whitney and Fergus [46]).

These benefits arise from the accessibility of a model of the MDP dynamics, which, in turn, is associated with additional implementation effort and computational power. Maintaining such a model allows the generation of artificial rollouts and transitions without interacting with the environment. Regardless of the MBRL type (learned or known model), this proves to be sample efficient since the majority of the training data can be generated synthetically (Sutton and Barto [5], Kaelbling et al. [47], Buckman et al. [48]).

Janner et al. [26] and Deisenroth and Rasmussen [49] are both examples that utilize an MDP dynamics model to train an agent and significantly reduce the interactions with the environment. They only use a handful of real experiences to train a dynamics model, which generates up to 95% of the training data for the agent. This allows increasing the policy gradient steps per environmental step beyond a point that is typically unstable in model-free RL. Hence, they solve the task faster than traditional MFRL algorithms in terms of steps taken in the environment. Even so, the wall time (the actual time taken from the start of the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

training process to the end) of the entire process is poor, due to the frequent need to refit the MDP dynamics model. In contrast, our LCER approach also generates synthetic experiences from which the agent learns. However, this is done without the additional need for a dynamics model. LCER and its two variants, LCER-CC and LCER-RM, interpolate between adjacent transitions in the replay-buffer. This allows the creation of new and unseen transitions, and further, increasing the policy update ratio with a reduced risk of unstable training behavior.

Nagabandi et al. [32] is another model-based RL approach with promising results. They utilize a neural network as their dynamics model for robotic tasks and use it in tandem with the real environment to take actions. In addition to a standard model-free policy, a second model-based policy is trained, based on the MDP model, and further utilized to accelerate the training of the model-free policy with imitation learning concepts.

Besides neural networks, other options to represent MDP dynamics models are Bayesian Neural Networks (BNNs) and Gaussian Processes. Depeweg et al. [50] is a representative of the former. They utilize a BNN that includes additional stochastic input variables to capture statistical patterns within the model dynamics. This model is combined with stochastic optimization for policy learning and obtains promising results in real-world scenarios and an industrial benchmark.

One disadvantage of BNNs is their computational complexity. This can be intensive, especially when compared to traditional feed-forward neural networks. Another drawback is the model selection. It involves choosing appropriate values for various hyperparameters such as priors and posterior functions (Jospin et al. [51]). LCER is an extension for replay-buffers of off-policy RL algorithms. Therefore, it is not limited to any particular policy structure. However, in this work, we refrain from using BNNs to avoid the problems mentioned above and focus on the essential features.

Gaussian Processes are non-parametric models, that are a popular approach with the benefit of low sample complexity and the ability to explicitly represent epistemic uncertainty. Consequently, they are becoming more and more widely used. Successful representatives of model-based RL with GPs are Wang et al. [34], Deisenroth and Rasmussen [49] and Kocijan et al. [52]. Gadd et al. [35] use GPs with increased depth to rise the model capacity and, in turn, the model complexity. They also incorporate prior knowledge of the dynamics to further improve performance concerning smoothness and structure. In addition, this approach addresses the problem of accumulating model errors and inaccuracies during virtual rollouts, which are exclusively executed on the MDP dynamics model.

A drawback of the MBRL approach with GPs, is the inevitable choice of a kernel, which might not scale to large and high-dimensional data settings. This also leads to potentially severe smoothing constraints and limits the asymptotic performance of

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

the model. In comparison, LCER is designed to work with both small and large data. For small sets, the benefit of local interpolation is exploited, whereas, for large sets, clustering provides additional advantages. Further, the local interpolation mechanism utilizes mixup sampling, which facilitates the processing of high-dimensional data.

In general, a combination of model-based and model-free methods turns out to be very auspicious. Model-based RL algorithms are sample efficient and model-free RL methods are superior regarding the asymptotic performance (Nagabandi et al. [32], Chua et al. [33]). Several approaches have proposed to learn the MDP dynamics model using a handful of real rollouts and further utilize the model to train a model-free policy (Janner et al. [26], Nagabandi et al. [32]).

The joint combination of MBRL and MFRL is also commonly used in the field of planning, where trajectories are calculated (planned) and optimized before interacting with the environment. These trajectories are utilized to solve the given task. Generally, this domain strongly benefits from model-based RL. The MDP dynamics model can be used to calculate the optimal trajectories, without the need to interact with the real environment. Chua et al. [33] and Deisenroth and Rasmussen [49] are prominent approaches that incorporate MBRL in the planning domain. The latter maintains a GP model to train a Radial Bias Function (RBF) network policy using the conjugate gradient method or the limited-memory Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm. Chua et al. [33] utilize an ensemble of neural network models to represent the MDP dynamics and generate optimal trajectories with the help of Model Predictive Control (MPC). Maintaining an ensemble of dynamics models accounts for epistemic uncertainty, or the uncertainty in the model parameters, which is particularly dominant in regions where data is scarce.

These planned trajectories or rollouts can further be used to train a model-free RL policy. Levine et al. [53] fit locally linear models around rollouts and train a neural network policy to follow trajectories, which are found by an iterative Linear Quadratic Regulator (iLQR) (Todorov and Li [54]). Silver et al. [55] learn an implicit model of the dynamics for implicit planning via value estimation and Tamar et al. [56] utilize value iteration networks that can learn to plan and are suitable for predicting outcomes that involve planning-based reasoning. The concept of training a fast reactive (deep neural network) and a slow, non-reactive policy, that can plan multiple steps ahead, in compound, is also used by Sun et al. [57] and analyzed with a focus set on convergence.

As the previous examples show, planning methods using model-based RL are very popular. In this context, algorithms with simple planners have emerged and shown great performance. Especially model predictive or receding horizon control (Mayne and Michalska [58]) is widely used. Ranging from replacing the standard PI controller in chemical processes (Draeger et al. [59]), through control

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

strategies of existing thermoelectric power plants (Grancharova et al. [60]), up to a differentiable policy class for reinforcement learning in continuous state and action spaces (Amos et al. [61]).

Another, simple planning algorithm that is commonly used in combination with model-based RL is the Cross Entropy Method (CEM) (Botev et al. [62]). CEM approximates the optimal importance sampling estimator. It works with two probability distributions, where one acts as the working distribution and the other one as the target. Samples are drawn from the working distribution in order to minimize the cross entropy with respect to the target distribution to produce better samples. Pourchot and Siguad [63] utilize this method in combination with Twin Delayed DDPG to achieve great performance on robotic locomotive control tasks. They use the experiences, generated from planned rollouts, to train the TD3 algorithm. Chua et al. [33] is another prominent example. They utilize CEM in their implementation, which serves as the basis for their MPC. It exclusively acts in the virtual world, provided by an ensemble of MDP dynamics models.

All of the above-mentioned MDP dynamics models have in common that they need to deal with a variety of model errors. Typical problems are model uncertainty, model bias, and the lack of real data to learn a complete model of the dynamics (Moerland et al. [12]). This is also true for the planning domain, especially, where the task is learned based on trajectories generated by a planning algorithm. Here, it is essential for the agent to also experience bad actions, to understand which actions are better and which actions are worse. This issue is demonstrated and addressed by Gu et al. [64]. They utilize their own off-policy algorithm, called Normalized Advantage Functions (NAF), which is a variant of Q-learning, and combine it with an iterative Linear Quadratic Gaussian (iLQG) planner.

As a result of these additional problems, inevitably associated with maintaining an MDP dynamics model of the environment, other strategies to cope with the sample inefficiency of model-free algorithms have emerged. Chen et al. [25] achieve great performance on continuous control tasks with their implementation of Randomized Ensemble Double Q-Learning. They utilize a Soft Actor-Critic algorithm with an ensemble of critics, allowing them to raise the number of policy updates per environment step to values that normally yield unstable training. Their results are comparable to state-of-the-art model-based approaches, due to the high update-to-data ratio.

One disadvantage of REDQ, however, arises from its core feature, the ensemble of critics. It requires individual updates of each member and leads to increased computation and poor wall time. The use of LCER circumvents the necessity of additional critics to ensure a stable training behavior. The utilized interpolation strategy allows both variants, LCER-CC and LCER-RM, to generate new and unseen transitions, reducing the risk of overfitting and unstable training.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

4 Local Cluster Experience Replay

We combine the components described in Chapter 2 to create Local Cluster Experience Replay (LCER). LCER is an extension for replay-buffers to increase the sample efficiency of off-policy reinforcement learning algorithms. It is inspired by and therefore has a close connection to the implementation of Neighborhood Mixup Experience Replay (NMER).

The goal is to reduce the number of required interactions with the environment to maximize the per-step reward and, in turn, the cumulative reward. At the same time, the additional overhead in terms of implementation effort and computational power should be kept at a minimum.

In order to cope with these requirements, LCER is designed as a wrapper around the standard replay-buffer. Figure 4.1 depicts a schematic representation of LCER and its two variants Local Cluster Experience Replay Cluster Center (LCER-CC) and Local Cluster Experience Replay Random Member (LCER-RM).

The same figure also shows the introduction of k-Means clustering and mixup sampling to the replay-buffer, which is the main idea of LCER. Further, it can be seen that we use Soft Actor-Critic RL agents. This reinforcement learning algorithm is chosen, due to its strong performance across a wide range of applications and environments (Achiam [4], Haarnoja et al. [21], Wang et al. [23]).

In this chapter, starting with the idea, the implementation of LCER is described. Section 4.1 covers LCER-CC and Section 4.2 describes LCER-RM, which are both variants of LCER. The necessary extension to use Local Cluster Experience Replay with HER is part of Section 4.3 and at the end, the relationship to NMER, as well as mixup sampling and the utilized local, linear interpolation are discussed.

4.1 Local Cluster Experience Replay Cluster Center

Local Cluster Experience Replay introduces k-Means clustering and mixup sampling to the replay-buffer of off-policy RL algorithms. LCER-CC forms clusters within the replay-buffer and interpolates new, unseen transitions between sampled transitions and their assigned cluster centers. Conceptually, LCER-CC creates locally linear models between different transitions in the replay-buffer and their associated cluster centers.

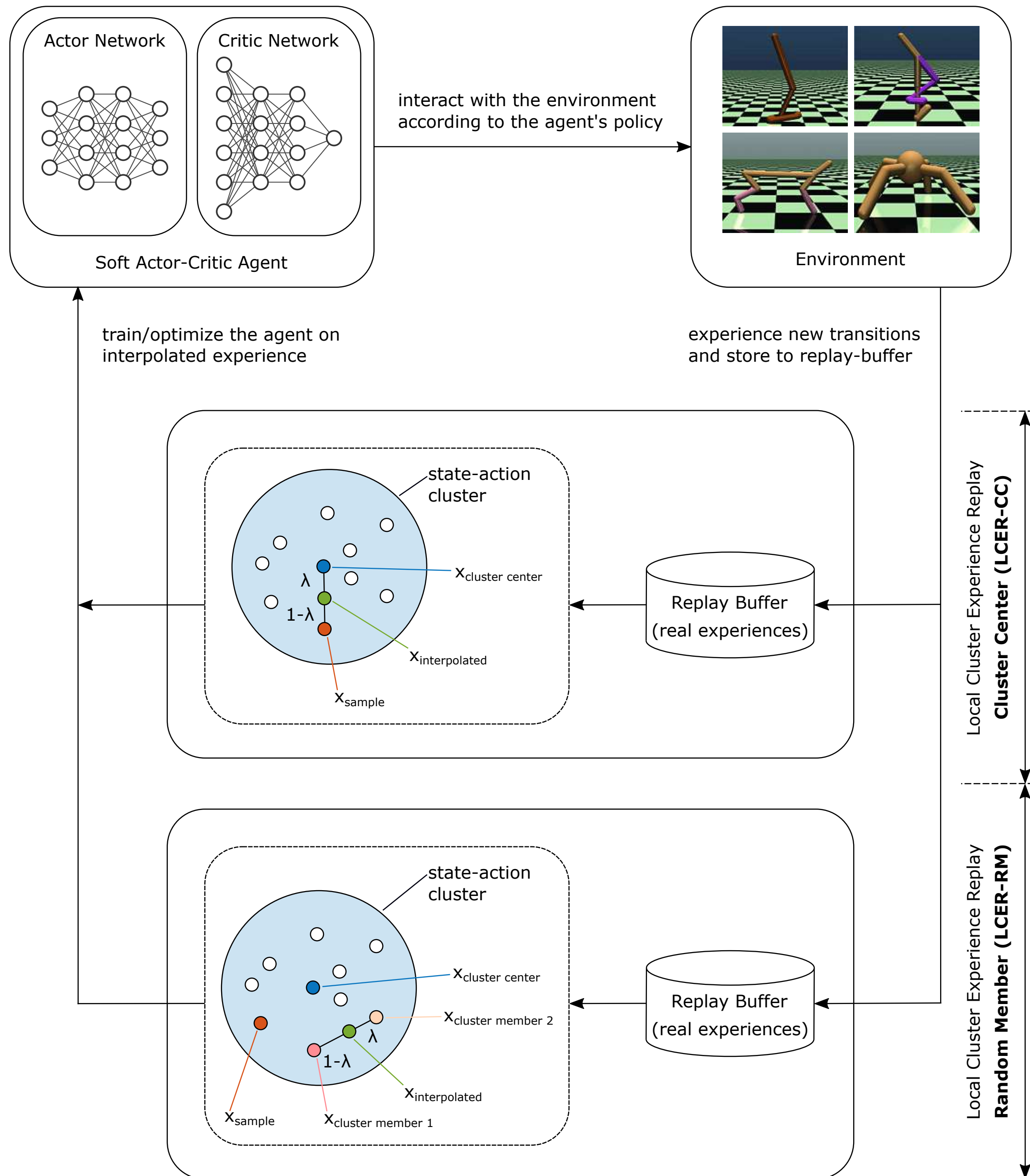

Figure 4.1: LCER and its two variants LCER-CC and LCER-RM. The experience, generated by the agent while interacting with the environment, is collected and stored in a replay-buffer. To train the agent, a sample transition is randomly chosen from the replay-buffer and assigned to a cluster in the z-space (standardized state-action coordinates). LCER-CC: A new transition is created by interpolation between the sample transition and its assigned cluster center, using mixup sampling. LCER-RM: A new transition is created by interpolation between two transitions from the same cluster as the sample transition belongs to, using mixup sampling.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

The usage of clustering algorithms to identify and group adjacent transitions within the replay-buffer allows the handling of even large buffers. Clustering algorithms can divide the data into smaller, more manageable groups, allowing them to process the samples faster and more efficiently. In addition, LCER does not rely on a ranking of nearest transitions, which favors grouping by clustering.

LCER-CC utilizes mini-batch k-Means as the underlying cluster algorithm. It is a variation of the standard k-Means method that enables online clustering. As such, mini-batch k-Means is designed to process data in a stream-like fashion. Small batches of input data are used to partially update the clusters and their centers. Similar to vanilla k-Means, mini-batch k-Means alternates between two steps, which are repeated until convergence, or a maximum of iterations is reached.

1. In the **first step**, random samples are drawn from the dataset and mini-batches are formed.
2. In the **second step**, the cluster centers are updated.

In contrast to standard k-Means, this is done on a per-sample basis. For each sample in the mini-batch, the associated cluster center is updated by taking the running average of all previously assigned samples and the new sample. This decreases the rate of change for a cluster center over time (Pedregosa et al. [43]). Algorithm C.7 depicts a pseudo-code implementation of the utilized mini-batch k-Means method.

The clustering for LCER is done in the dimensions of the standardized (zero mean, unit variance) state-action space, further called z-space. This guarantees to group transitions with similar inputs and addresses the problem of changing environment states and corresponding policy actions over time. It is also designed to keep the additional computational overhead as low as possible. Figure 4.2 schematically depicts the clusters within the z-space.

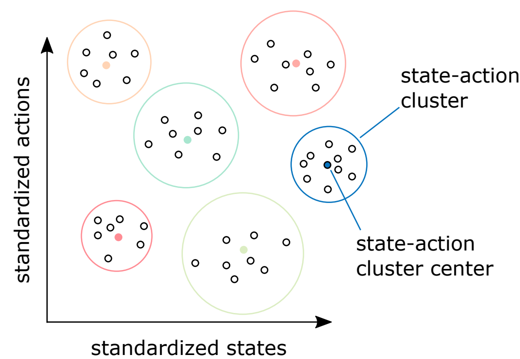

Figure 4.2: LCER clusters in the state-action space.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

From a conceptual point of view, the implementation of LCER-CC consists of two major steps.

1. In the **cluster update step**, a batch of new environment transitions is added to the replay-buffer. These are utilized to

   a) update the running average μ_z and standard deviation σ_z that are used to standardize states and actions.

   b) update the mini-batch k-Means algorithm, utilizing the standardized states and actions.

   c) update the data structures to calculate the running averages $\mu(l_c)$ and standard deviations $\sigma(l_c)$ for all clusters containing all members of an environment transition quintuple, except for the terminal flag d_t

$$\begin{aligned} \mu(l_c) &\leftarrow \text{MEAN}(s_b, a_b, r_b, s_b^{'}), \\ \sigma(l_c) &\leftarrow \text{STD}(s_b, a_b, r_b, s_b^{'}), \end{aligned} \tag{4.1}$$

   where the subscript b represents the batch of new environment transitions and l_c denotes the corresponding cluster label.

   The **cluster update step** is typically done at the end of an episode rollout.

2. The **cluster sampling step** is performed right before the update step of the RL policy and is used to provide a batch of training samples $\mathcal{B}_{\text{train}}$ from the replay-buffer. LCER-CC performs the following steps for this purpose

   a) Sample a batch $\mathcal{B}$ uniformly at random from the replay-buffer $\mathcal{D}$,

$$\mathcal{B} \overset{iid}{\sim} \mathcal{U}(\mathcal{D}). \tag{4.2}$$

   b) Standardize states and actions of the sampled transitions (zero mean, unit variance).

   c) Query the associated cluster labels l_c for each sampled transition in the batch $\mathcal{B}$, utilizing the mini-batch k-Means algorithm[1].

   d) Query associated running averages $\mu(l_c)$ and standard deviations $\sigma(l_c)$, based on the cluster labels.

[1]The utilized mini-batch k-Means implementation does not store and update the associated cluster labels for each cluster member. Thus, this step is necessary, but may be omitted with other implementations.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

e) Calculate cluster centers x_{cc} for all components of an environment transition

$$x_{cc} = \mu(l_c) + \alpha \mathcal{N}(0,1)\,\sigma(l_c), \tag{4.3}$$

where $\alpha \in \mathbb{R}^+$ denotes a weighting factor. This step is significant because, in addition to moving the cluster centers during the cluster update step, it superimposes extra noise, which emphasizes exploration.

f) For each transition in the batch $\mathcal{B}$ sample a mixup coefficient $\lambda \sim \mathcal{U}[0,1)$ and interpolate between the sampled transition and the corresponding cluster center x_{cc}

$$x_i = \lambda x_{cc} + (1-\lambda)x_s. \tag{4.4}$$

g) Add the interpolated samples x_i to the training batch $\mathcal{B}_{\text{train}}$ and return $\mathcal{B}_{\text{train}}$.

Algorithm 4.1 depicts these steps as pseudo-code. Figure 4.3 gives a schematic, visual representation of LCER-CC, whereby the focus is set on the clustering and the interpolation.

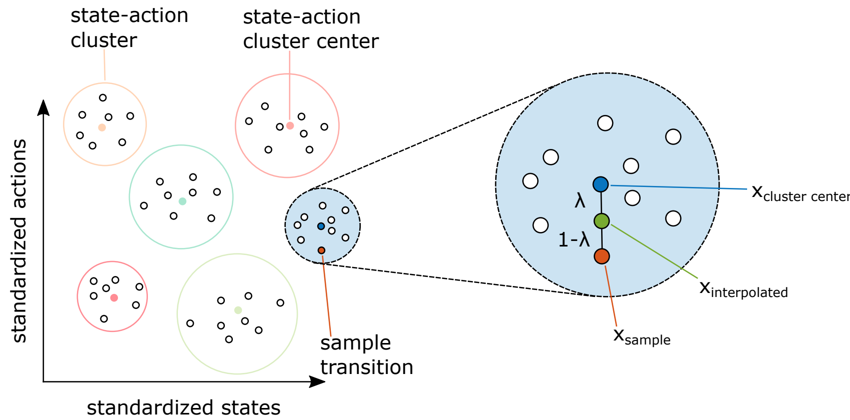

Figure 4.3: LCER-CC clusters and mixup sampling. The sample transition is randomly chosen from the replay-buffer and assigned to a cluster in the z-space (standardized state-action coordinates). A new transition is created by interpolation between the sample transition and its assigned cluster center, using mixup sampling.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Algorithm 4.1 Local Cluster Experience Replay Cluster Center (LCER-CC).

1: **Input**: off-policy RL algorithm $\mathbb{A}$ (e.g. DDPG, SAC), replay-buffer $\mathcal{D}$, maximum timesteps T, horizon N, mini-batch k-Means object KMO, arrays μ and σ for running averages and standard deviations, hyperparameter $\alpha \in \mathbb{R}^+$ and counter n
2: Initialize $\mathbb{A}$, replay-buffer $\mathcal{D}$, k-Means object KMO, μ, σ and counter $n = 0$
3: Observe initial state s_0
4: **for** t = 0, $T - 1$ **do**
5: Choose action $a_t \sim \pi_\theta(s_t)$
6: Observe s_{t+1}, r_t, d_t and store transition $(s_t, a_t, r_t, s_{t+1}, d_t)$ in $\mathcal{D}$
7: Increase counter n
8: **if** $(t \bmod N = 0) \vee (d_t = true)$ **then**
9: Update ZScore with state and actions, $\text{ZScore}(s_{t-n:t}, a_{t-n:t})$
10: Standardize states and actions

$$(\widetilde{s}_{t-n:t}, \widetilde{a}_{t-n:t}) \leftarrow \text{ZScore}(s_{t-n:t}, a_{t-n:t})$$

11: Update clusters with standardized states and actions, $\text{KMO}(\widetilde{s}_{t-n:t}, \widetilde{a}_{t-n:t})$
12: Update running averages $\mu(l_c)$ and standard deviations $\sigma(l_c)$

$$\mu(l_c) \leftarrow \text{MEAN}(s_{t-n:t}, a_{t-n:t}, r_{t-n:t}, s_{t+1-n:t+1})$$
$$\sigma(l_c) \leftarrow \text{STD}(s_{t-n:t}, a_{t-n:t}, r_{t-n:t}, s_{t+1-n:t+1})$$

13: Reset counter $n = 0$
14: **end if**
15: **if** it's time to update **then**
16: **for** however many updates **do**
17: Sample batch $\mathcal{B}$ uniformly from replay-buffer, batch size K

$$\mathcal{B} = \left\{(s_k, a_k, r_k, s_k^{'})\right\}_{k=1}^{K} \overset{iid}{\sim} \mathcal{U}(\mathcal{D})$$

18: Standardize states and actions of sampled transitions

$$(\widetilde{s}_k, \widetilde{a}_k) \leftarrow \text{ZScore}(s_k, a_k), \quad \forall s_k, a_k \in \mathcal{B}$$

19: Get associated cluster labels $l_c \leftarrow \text{KMO}(\widetilde{s}_k, \widetilde{a}_k)$
20: Get associated running averages $(\overline{s}_{l_c}, \overline{a}_{l_c}, \overline{r}_{l_c}, \overline{s}_{l_c}^{'}) \leftarrow \mu(l_c)$ and standard deviations $(\underline{s}_{l_c}, \underline{a}_{l_c}, \underline{r}_{l_c}, \underline{s}_{l_c}^{'}) \leftarrow \sigma(l_c)$
21: Determine cluster centers

$$x_{cc} = (\overline{s}_{l_c}, \overline{a}_{l_c}, \overline{r}_{l_c}, \overline{s}_{l_c}^{'}) + \alpha\,\mathcal{N}(0, 1)\,(\underline{s}_{l_c}, \underline{a}_{l_c}, \underline{r}_{l_c}, \underline{s}_{l_c}^{'})$$

22: Sample mixup coefficient $\lambda \sim \mathcal{U}[0, 1)$
23: Interpolate sampled transitions x_s and cluster centers x_{cc}

$$x_i = \lambda x_{cc} + (1 - \lambda) x_s$$

24: Add interpolated samples x_i to training batch $\mathcal{B}_{\text{train}} \leftarrow x_i$
25: **end for**
26: **end if**
27: **end for**

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

4.2 Local Cluster Experience Replay Random Member

Similar to LCER-CC, LCER-RM also forms clusters within the replay-buffer. However, it utilizes a standard and Graphics Processing Unit (GPU) accelerated k-Means algorithm. LCER-RM randomly samples two transitions from the same cluster as the sampled transition and interpolates between them to generate new and unseen transitions. In the same way, as with LCER-CC, locally linear models between different transitions in the replay-buffer are created.

LCER-RM is designed to prevent possible disadvantages of LCER-CC. Unlike LCER-CC, LCER-RM recalculates the clusters in each cluster update step. This overcomes the problem of a potentially worse fit of the clusters by only being adjusted. LCER-RM also mitigates the problem of the one-directional interpolation towards the cluster center only, as well as the need to keep a reference of data structures used to calculate the running averages $\mu(l_c)$ and standard deviations $\sigma(l_c)$.

The implementation of LCER-RM consists of two major steps.

1. In the **cluster update step**, a batch of new environment transitions is added to the replay-buffer. These are utilized to
   a) update the running average μ_z and standard deviation σ_z that are used to standardize states and actions.
   b) update the k-Means algorithm and recalculate the clusters, utilizing the standardized states and actions of the entire replay-buffer.

   This step is typically done at the end of an episode rollout.
2. The **cluster sampling step** is performed right before the update step of the RL policy and is used to provide a batch of training samples $\mathcal{B}_{\text{train}}$ from the replay-buffer. LCER-RM performs the following steps for this purpose
   a) Sample a batch $\mathcal{B}$ uniformly at random from the replay-buffer $\mathcal{D}$,
   $$\mathcal{B} \overset{iid}{\sim} \mathcal{U}(\mathcal{D}). \tag{4.5}$$
   b) Standardize states and actions of the sampled transitions (zero mean, unit variance).
   c) Query the associated cluster labels for each sampled transition in the batch $\mathcal{B}$, utilizing the k-Means algorithm.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

d) Randomly chose two transitions, x_{cm1} and x_{cm2}, from the same cluster as each sampled transition in the batch $\mathcal{B}$, utilizing the cluster labels[2].

e) For each transition in the batch $\mathcal{B}$ sample a mixup coefficient $\lambda \sim \mathcal{U}[0, 1)$ and interpolate between the two previous chosen transitions

$$x_i = \lambda x_{cm2} + (1 - \lambda) x_{cm1}. \tag{4.6}$$

f) Add the interpolated samples x_i to the training batch $\mathcal{B}_{\text{train}}$ and return $\mathcal{B}_{\text{train}}$.

Algorithm 4.2 depicts these steps as pseudo-code. Figure 4.4 gives a schematic, visual representation of LCER-RM, whereby the focus is set on the clustering and the interpolation.

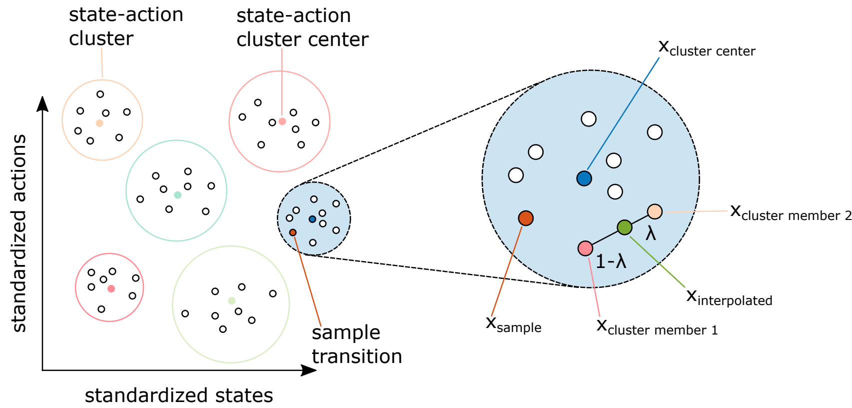

Figure 4.4: LCER-RM clusters and mixup sampling. The sample transition is randomly chosen from the replay-buffer and assigned to a cluster in the z-space (standardized state-action coordinates). A new transition is created by interpolation between two randomly chosen transitions from the same cluster as the sample transition, using mixup sampling.

[2]Randomly choosing two transitions mitigates the problem of choosing the same transition multiple times.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Algorithm 4.2 Local Cluster Experience Replay Random Member (LCER-RM).

1: **Input**: off-policy RL algorithm $\mathbb{A}$ (e.g. DDPG, SAC), replay-buffer $\mathcal{D}$, maximum timesteps T, horizon N, k-Means object KMO and counter n
2: Initialize $\mathbb{A}$, replay-buffer $\mathcal{D}$, k-Means object KMO and counter $n = 0$
3: Observe initial state s_0
4: **for** t = 0, $T - 1$ **do**
5: Choose action $a_t \sim \pi_\theta(s_t)$
6: Observe s_{t+1}, r_t, d_t and store transition $(s_t, a_t, r_t, s_{t+1}, d_t)$ in $\mathcal{D}$
7: Increase counter n
8: **if** $(t \bmod N = 0) \vee (d_t = true)$ **then**
9: Update ZScore with state and actions, ZScore$(s_{t-n:t}, a_{t-n:t})$
10: Standardize states and actions of entire replay-buffer $\mathcal{D}$

$$(\tilde{s}, \tilde{a}) \leftarrow \text{ZScore}(s, a), \quad \forall s, a \in \mathcal{D}$$

11: Update clusters with standardized states and actions, KMO$(\tilde{s}, \tilde{a})$
12: Reset counter $n = 0$
13: **end if**
14: **if** it's time to update **then**
15: **for** however many updates **do**
16: Sample batch $\mathcal{B}$ uniformly from replay-buffer, batch size K

$$\mathcal{B} = \left\{(s_k, a_k, r_k, s_k^{'})\right\}_{k=1}^{K} \overset{iid}{\sim} \mathcal{U}(\mathcal{D})$$

17: Standardize states and actions of sampled transitions

$$(\tilde{s}_k, \tilde{a}_k) \leftarrow \text{ZScore}(s_k, a_k), \quad \forall s_k, a_k \in \mathcal{B}$$

18: Get associated cluster labels $l_c \leftarrow \text{KMO}(\tilde{s}_k, \tilde{a}_k)$
19: Randomly choose two transitions per cluster label l_c

$$x_{cm1} = (s_1, a_1, r_1, s_1^{'}) \leftarrow \mathcal{D}(l_c), \quad x_{cm2} = (s_2, a_2, r_2, s_2^{'}) \leftarrow \mathcal{D}(l_c)$$

20: Sample mixup coefficient $\lambda \sim \mathcal{U}[0, 1)$
21: Interpolate previous chosen transitions x_{cm1} and x_{cm2}

$$x_i = \lambda x_{cm2} + (1 - \lambda) x_{cm1}$$

22: Add interpolated samples x_i to training batch $\mathcal{B}_{\text{train}} \leftarrow x_i$
23: **end for**
24: **end if**
25: **end for**

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

4.3 Local Cluster Experience Replay and Hindsight Experience Replay

Hindsight Experience Replay presupposes the additional input of a goal in the policies and action-value functions, see Section 2.4.4. LCER takes this into account by including the goal in the z-space. This increases the dimensions of the clustering from the standardized state-action coordinates to the standardized state-goal-action coordinates.

Further, concerning LCER-CC, the data structures to calculate the running averages $\mu(l_c)$ and standard deviations $\sigma(l_c)$ for all clusters and all components of environment transitions (see Equation 4.1) also need to be extended

$$\begin{aligned} \mu(l_c) &\leftarrow \text{MEAN}(s_b, g_b, a_b, ag_b, s_b^{'}, ag_b^{'}), \\ \sigma(l_c) &\leftarrow \text{STD}(s_b, g_b, a_b, ag_b, s_b^{'}, ag_b^{'}), \end{aligned} \tag{4.7}$$

where g_b and ag_b denote the goal and the achieved goal, respectively.

This provides all the additional components needed to use LCER in combination with HER.

4.4 Local Cluster Experience Replay and Neighborhood Mixup Experience Replay

LCER is closely related to NMER, although there are clear differences in the utilized methods. NMER uses the L2 distance, measured in the dimensions of the standardized state-action space, to compute the nearest neighbors and to ensure the proximity between transitions in the replay-buffer, see Section 2.4.3. LCER, in contrast, provides the same characteristic by utilizing k-Means clustering. This applies to both LCER-CC and LCER-RM.

Although the k-Means clustering used in LCER is also based on the L2 distance between two transitions, it offers two significant advantages in its application and processing. The k-Nearest-Neighbor (k-NN) algorithm requires a lot of computational power to calculate and sort the distances to all transitions in the buffer. This is true, even if it is done only once per environmental episode. LCER-CC utilizes mini-batch k-Means, a special variant for stream-like input data, to reduce the computational overhead. Here, only the L2 distances to the existing cluster centers, which are fixed in number, have to be calculated. Even with the additional need to refit the cluster centers, mini-batch k-Means is significantly faster than the standard k-NN approach. LCER-RM, in contrast, benefits from a standard but GPU accelerated k-Means variant, which is responsible for the fast computation.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

The second advantage of using k-Means instead of the k-NN algorithm stems from the additional introduced exploration. NMER utilizes a pool of the first few nearest neighbors (typically 5 to 10)[3], from which an interpolation partner is randomly chosen. LCER-RM, on the other hand, provides clusters that contain only proximal transitions. Thus, the number of participants in the pool increases towards the number of members in the cluster. LCER-CC achieves this behavior by always interpolating towards the cluster center. Due to the periodic refit of the centers and the additional application of noise, an increased exploration is ensured.

Figure 4.1 schematically depicts the implementation and inner processes of both LCER variants, LCER-CC and LCER-RM.

4.5 Mixup Sampling and Local, Linear Interpolation

In this final section, the combination of clustering with mixup sampling and local linear interpolation is discussed. The need for adjacent transitions and the continuous state and action spaces used with advantage are addressed. Furthermore, potential situations where the presented strategy may have problems are described, as well as how LCER deals with them.

Mixup sampling is an interpolation mechanism used to improve the generalizability of a reinforcement learning agent. It generates interpolated samples by taking convex combinations of previously experienced transitions, which can be used to support the policy. Further, mixup sampling assumes that linear combinations of adjacent state-action pairs result in the same linear combinations of the corresponding reward and next state pairs. This assumption is known as a prior (Sander et al. [2]).

This assumption, or prior, is exploited in LCER, which only interpolates between samples within the same cluster. The assignment for each transition sample to its corresponding cluster is done during the cluster update step in the standardized dimensions of the state-action space and ensures close proximity in the z-space. Invoking the prior, the interpolation concept of LCER starts from similar initial pairs in the z-space, and thus, in turn, leads to similar pairs of rewards and next states.

In this work, only agents with continuous state and action spaces are considered. This fact benefits the prior, which, although strong in some settings, is also valid even when the linear assumption only approximately holds. Mixup sampling can be particularly useful when the transitions in the replay-buffer form a convex manifold. This is because mixup sampling generates convex combinations of experiences, which ensures that the interpolated transitions lie within this manifold (Sander et al. [2]).

[3]These numbers were determined empirically and may differ. For a more detailed examination, please refer to Sander et al. [2], [3].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

The assumption of a convex transition manifold does not hold in many real-world environments. This is especially true for continuous control tasks with complex dynamics, high-dimensional state and action spaces, and settings with nonlinear rewards (e.g. sparse rewards). LCER addresses this issue during the cluster sampling step, where interpolation is only done between sampled transitions from the same cluster. The proximity within one cluster ensures that linear interpolation is a suitable approximation for interpolation between adjacent transitions.

The goal of LCER is a simple implementation, as well as to keep the additional computational power as low as possible. Therefore, the mixup sampling parameter λ in the Equations 4.4 and 4.6 is sampled uniformly at random $\lambda \sim \mathcal{U}[0, 1)$, rather than from a β-distribution, as it is the case in Zhang et al. [1] and Sander et al. [2].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

5 Experiments

In this chapter, the experimental setup and results are described. It is grouped into experiments with continuous locomotive control tasks, in Section 5.2, and experiments with continuous robotic control tasks, in Section 5.3. The results are evaluated and compared with state-of-the-art reinforcement learning algorithms, in particular, to quantify the sample efficiency of Local Cluster Experience Replay. The two variants LCER-CC and LCER-RM are considered separately. At the end of this chapter, the additional computational effort, added by the use of LCER, is further discussed and compared to existing implementations.

5.1 Experimental Setup and Evaluation Metrics

Local Cluster Experience Replay benefits from continuous control tasks where both the state and action spaces are continuous. The environmental experiments include a handful of commonly used OpenAI Gym (Brockman et al. [65]) MuJoCo (Todorov et al. [66]) environments and environments from ShadowHand-Gym (Zahlner [67]), which utilize PyBullet (Coumans and Bai [68]) as their underlying physics engine. This covers simple tasks, such as swinging a pendulum, moderately difficult tasks, like moving rigid bodies, to very complex tasks, such as rolling a block within a high-dimensional robotic hand. Table B.1 depicts the names of the utilized environments along with their corresponding state and action space dimensions and their maximum episode lengths, measured in timesteps.

The sampling efficiency of Local Cluster Experience Replay and its variants LCER-CC and LCER-RM is compared to a set of state-of-the-art model-free and model-based reinforcement learning algorithms. As with LCER, these algorithms are applied to the same environments, tested and their performance is evaluated. Table 5.1 provides the used baseline algorithms and additional notes on the implementations, as well as the codebases.

The performance comparison of reinforcement learning agents using LCER with standard RL agents is done using the evaluation reward or evaluation success rate at fixed timesteps. Which of the two metrics is used, is determined by the environment. Results are reported as average reward or success rate and interquartile range, where each result is averaged over 3 different seeds, each evaluated for 10 episodes. Exponential moving average, based on exponential smoothing, is utilized to smooth the results, whereby a factor of 0.75 is used.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

RL algorithm	Implementation note
SAC	vanilla implementation, code based on pranz24 [69]
SAC utd x	vanilla SAC with x policy gradient steps per environment step
SAC nub x	vanilla SAC with x policy gradient steps per environment episode
PER	vanilla implementation, code based on MrSyee [70]
PER utd x	vanilla PER with x policy gradient steps per environment step
MBPO	vanilla implementation, code based on Xingyu-Lin [71]
NMER	vanilla implementation, code based on rmsander [72]
HER+SAC	vanilla SAC, adapted to work with HER, code based on TianhongDai [73]
HER+SAC nub x	vanilla HER+SAC with x policy gradient steps per environment episode
HER+MBPO	vanilla MBPO, adapted to work with HER
HER+NMER	vanilla NMER, adapted to work with HER

Table 5.1: Baseline RL algorithms and additional notes on the implementations used in the experiments. Supplementary, the codebases are also listed.

5.2 Continuous Locomotive Control Environments

This section details the experiment results for the continuous locomotive control environments, which are all members of the OpenAI Gym MuJoCo suite. The included environments are `InvertedPendulum-v2`, `Hopper-v2`, `Walker2d-v2`, `AntTruncated-v2` and `HalfCheetah-v2`. They are briefly described below, adapted from OpenAI Gym [65].

`InvertedPendulum-v2`: This setup consists of a cart that is capable of linear movement. It has a pole attached on one end and the objective is to balance the pole by applying forces to the cart through pushing it left or right.

`Hopper-v2`: The hopper is a two-dimensional, one-legged figure. It is comprised of four main components: a torso at the top, a thigh in the middle, a leg at the bottom, and a single foot serving as the base of support. The objective is to move forward (to the right) by making hops, achieved through applying torque on the three hinges linking the four body parts.

`Walker2d-v2`: The walker is a two-dimensional figure with two legs. It consists of seven main body parts: a single torso, two thighs located below the torso, and two legs positioned below the thighs. Two additional feet serve as the base of support. The task is to move forward (to the right) by coordinating the limbs, achieved by applying torque on the six hinges of the walker.

`Ant-v2` and **`AntTruncated-v2`**: The ant is a three-dimensional robot comprised of a single torso, a freely rotating body, and four legs attached to it. Each leg has two links. The goal is to move forward (to the right) by coordinating the motion of the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

four legs. This is accomplished through the application of torque to the eight hinges that connect the two limbs of each leg and the torso. In the conducted experiments, a truncated and simplified version of this environment is used, as described below. `AntTruncated-v2` is a special variant of the standard `Ant-v2` environment, where the contact forces, normally applied to the center of mass of each of the links, are not considered. These forces are specified in the observation in the last 84 elements, which are therefore truncated.

`HalfCheetah-v2`: The cheetah is a two-dimensional robot consisting of nine links and eight joints, including two paws. The objective is to maximize the forward (right) speed of the cheetah by applying torque to the joints. The performance of the cheetah is evaluated based on the distance traveled in the forward direction, with positive rewards given for forward motion and negative rewards for backward motion. The torso and head of the cheetah are fixed. Torque can only be applied to the remaining six joints, connecting the front and back thighs to the torso, the shins to the thighs, and the feet to the shins.

Figure 5.1 shows a visual representation of the described environments and Table 5.2 depicts the corresponding dimensions of the state and action spaces. In Figure 5.2 the mean reward for all five continuous locomotive control environments is shown. The Tables 5.3, 5.4, 5.5 and 5.6 provide additional per-timestep evaluations. Detailed hyperparameters, used in the experiments, can be found in Appendix B.

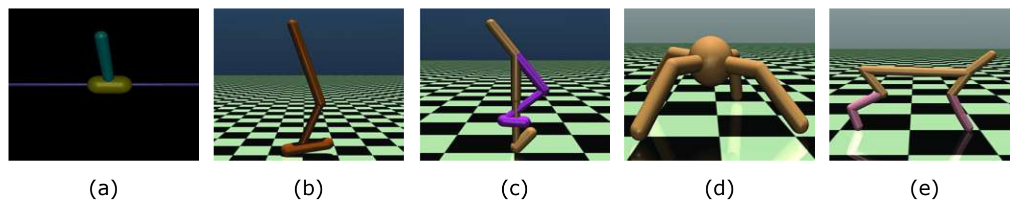

Figure 5.1: OpenAI Gym MuJoCo (Brockman et al. [65], Todorov et al. [66]) continuous locomotive control environments: (a) `InvertedPendulum-v2`, (b) `Hopper-v2`, (c) `Walker2d-v2`, (d) `AntTruncated-v2` and (e) `HalfCheetah-v2`. Adapted from Brockman et al. [65].

Environment	No. of actions	No. of states
`AntTruncated-v2`	8	27
`HalfCheetah-v2`	6	17
`Hopper-v2`	3	11
`InvertedPendulum-v2`	1	4
`Walker2d-v2`	6	17

Table 5.2: State and action space dimensions for the locomotive control OpenAI Gym MuJoCo (Brockman et al. [65], Todorov et al. [66]) environments.

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

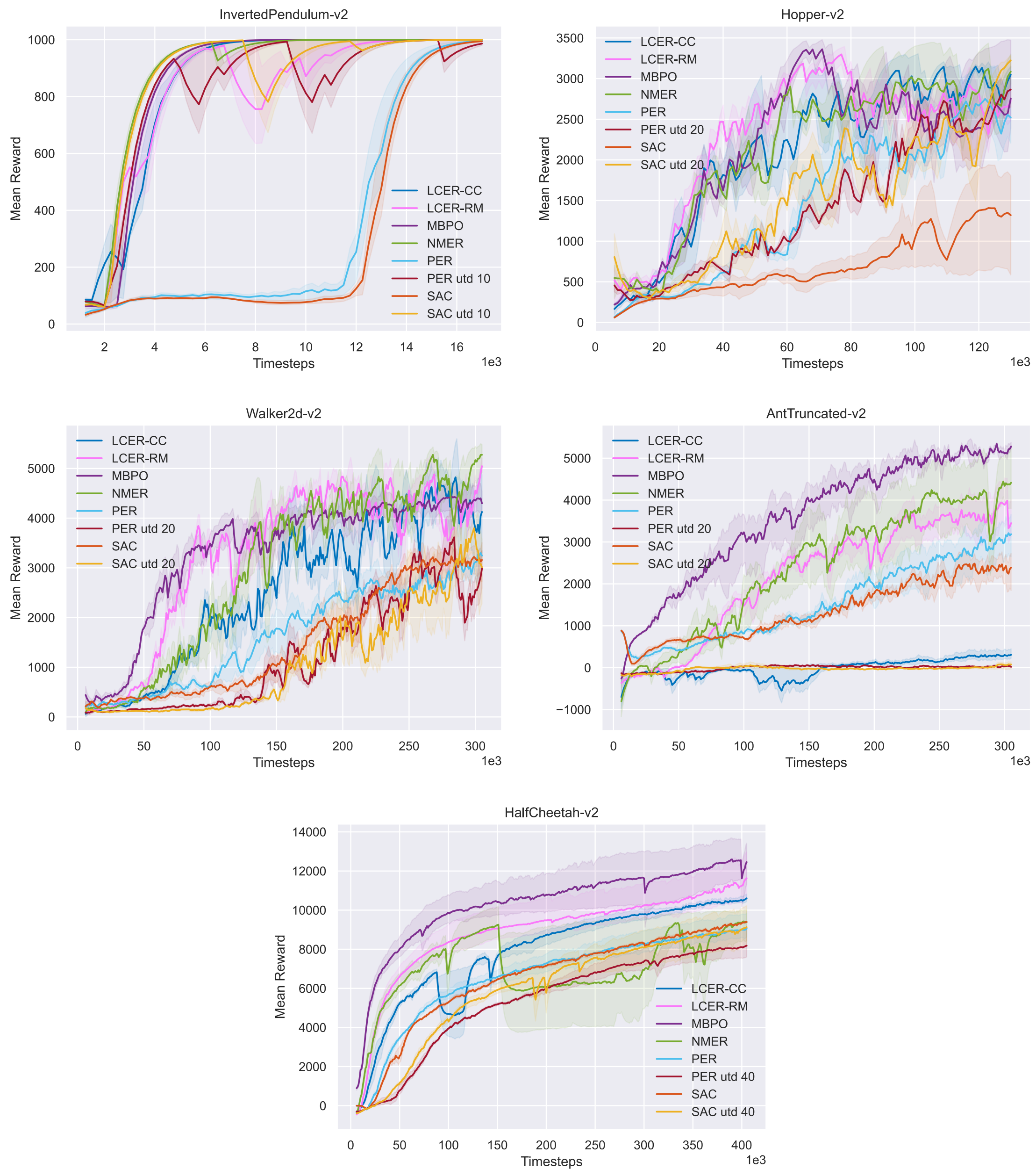

Figure 5.2: Evaluation results for the `InvertedPendulum-v2`, `Hopper-v2`, `Walker2d-v2`, `AntTruncated-v2` and `HalfCheetah-v2` OpenAI Gym MuJoCo (Brockman et al. [65], Todorov et al. [66]) environments. Results are reported as average reward (line) and interquartile range (shaded area).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

	RL agent	5k μ	5k IQR	10k μ	10k IQR	15k μ	15k IQR
InvertedPendulum	LCER-CC	901	36	**1000**	**0**	**1000**	**0**
	LCER-RM	903	71	872	156	999	1
	MBPO	942	8	**1000**	**0**	**1000**	**0**
	NMER	**967**	**5**	999	2	**1000**	**0**
	PER	97	15	115	54	960	29
	PER utd 10	901	70	812	279	998	2
	SAC	92	12	77	14	948	27
	SAC utd 10	964	5	961	58	998	2

Table 5.3: Evaluation results for the `InvertedPendulum-v2` OpenAI Gym MuJoCo (Brockman et al. [65], Todorov et al. [66]) environment for 5k, 10k and 15k timesteps. Results are reported as average reward (μ) and interquartile range (IQR).

	RL agent	30k μ	30k IQR	60k μ	60k IQR	90k μ	90k IQR	120k μ	120k IQR
Hopper	LCER-CC	931	558	2248	926	2789	534	**3074**	**135**
	LCER-RM	**1537**	**462**	2879	304	**2847**	**336**	2814	20
	MBPO	1183	437	**2886**	**653**	2781	873	2400	1386
	NMER	1335	367	2798	164	2843	855	2838	777
	PER	426	93	837	141	2173	612	2667	756
	PER utd 20	576	200	1007	93	1485	72	2469	449
	SAC	389	110	582	189	808	372	1378	1248
	SAC utd 20	493	165	1439	105	1574	187	2195	609

Table 5.4: Evaluation results for the `Hopper-v2` OpenAI Gym MuJoCo (Brockman et al. [65], Todorov et al. [66]) environment for 30k, 60k, 90k and 120k timesteps. Results are reported as average reward (μ) and interquartile range (IQR).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

	RL agent	Timesteps							
		75k		150k		225k		300k	
		μ	IQR	μ	IQR	μ	IQR	μ	IQR
`AntTruncated`	LCER-CC	-92	122	-238	506	142	114	303	234
	LCER-RM	840	239	2544	231	3651	855	3886	1173
	MBPO	**2465**	**741**	**3969**	**1115**	**4867**	**471**	**5113**	**88**
	NMER	1030	1051	2944	1915	3750	1656	4339	1127
	PER	690	91	1225	311	2365	1103	2995	941
	PER utd 20	-84	51	46	39	-6	20	30	39
	SAC	671	89	1142	265	2036	281	2376	641
	SAC utd 20	1	17	-21	53	-19	2	59	144
`Walker2d`	LCER-CC	1009	504	2391	388	4257	1305	3793	1644
	LCER-RM	2448	358	**4037**	**456**	4317	609	4554	572
	MBPO	**2963**	**1100**	3651	927	4204	521	4352	513
	NMER	1100	633	3766	752	**4671**	**1096**	**5168**	**694**
	PER	655	363	1782	882	2731	651	3116	168
	PER utd 20	193	51	936	517	2636	519	2554	728
	SAC	492	141	1024	512	2562	476	3191	545
	SAC utd 20	131	41	394	22	1589	906	3585	1160

Table 5.5: Evaluation results for the `AntTruncated-v2` and `Walker2d-v2` OpenAI Gym MuJoCo (Brockman et al. [65], Todorov et al. [66]) environments for 75k, 150k, 225k and 300k timesteps. Results are reported as average reward (μ) and interquartile range (IQR).

	RL agent	Timesteps							
		100k		200k		300k		400k	
		μ	IQR	μ	IQR	μ	IQR	μ	IQR
`HalfCheetah`	LCER-CC	4689	3310	8735	763	9784	564	10512	303
	LCER-RM	8362	1532	9470	1802	10239	1785	11251	2025
	MBPO	**9859**	**1436**	**10790**	**1978**	**11627**	**2746**	**11618**	**2305**
	NMER	6927	1819	6086	5211	6939	4716	9350	1607
	PER	5682	481	7283	920	8292	1110	9006	1179
	PER utd 40	3945	37	6001	436	7383	1105	8108	1474
	SAC	5291	120	7153	165	8306	95	9338	87
	SAC utd 40	4299	105	5776	2615	8084	1739	9023	2009

Table 5.6: Evaluation results for the `HalfCheetah-v2` OpenAI Gym MuJoCo (Brockman et al. [65], Todorov et al. [66]) environment for 100k, 200k, 300k and 400k timesteps. Results are reported as average reward (μ) and interquartile range (IQR).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

InvertedPendulum-v2 clearly is a very simple environment and can easily be solved by all eight methods. Only vanilla SAC and PER need a bit more timesteps to swing up the pendulum. The remaining algorithms have in common that they all use an increased update-to-data ratio and thus solve the task in very few timesteps. The update ratio for this environment is 10 policy updates per environmental step, which is only moderately increased and leads to a stable training behavior for all utilized algorithms.

On the Hopper-v2 and Walker2d-v2 environments, LCER-RM clearly outperforms the other model-free configurations. This is especially true for the Walker2d-v2 environment. Compared to MBPO, a state-of-the-art model-based reinforcement learning algorithm, LCER-RM is in no way inferior to it and achieves the same sample efficiency. Local Cluster Experience Replay Random Member also surpasses NMER in the first few timesteps but achieves the same asymptotic convergence. LCER-CC, on the other hand, solves both tasks faster than the SAC and PER configurations and is on equal terms with NMER.

AntTruncated-v2 is the environment with the highest number of observation dimensions, regarding the continuous locomotive control tasks. Here, Model-Based Policy Optimization clearly is the best algorithm. However, LCER-RM and NMER are right behind MBPO and LCER-RM achieves about the same performance as NMER. Interestingly, LCER-CC performs worse in this environment and is not able to successfully solve the task. In fact, LCER-CC does not learn how to move the ant. There are many possible reasons why this is the case. This ranges from a bad cluster initialization to a faulty calculation of the cluster centers, as described in Equation 4.3, to the simple cause of three bad seeds. These reasons certainly apply to both variants of LCER. LCER-CC specific reasons for this behavior can be found first in the mini-batch k-Means algorithm and second in the interpolation of the cluster members to the cluster center only. Unlike LCER-RM, LCER-CC only adjusts the cluster centers but does not recalculate them each cluster update step. This can lead to a worse fit of the clusters and thus lower the learning curve. The behavior is also supported by the one-directional interpolation towards the cluster center, which was mentioned before.

The last environment of this section, HalfCheetah-v2, is the environment with the highest update-to-data ratio. For this task, an update rate of 40 policy updates per environmental step is used. The results are similar to that of the AntTruncated-v2 environment. Model-Based Policy Optimization is the best algorithm, closely followed by LCER-RM. This time, however, LCER-CC ranks third. NMER shows a drop in the evaluation reward at about 150k timesteps. This is caused by a single bad seed that has a strong impact. The same is true for LCER-CC at about 100k timesteps.

The overall evaluation of Local Cluster Experience Replay and its two variants LCER-CC and LCER-RM is consistently strong. Vanilla SAC and PER are com-

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

pletely outperformed. LCER-CC and LCER-RM are both on par with the performance of NMER. However, in comparison with Model-Based Policy Optimization, LCER does turn out slightly worse. Nevertheless, comparable results can be seen here as well, especially on the `Hopper-v2` and `Walker2d-v2` environments. An important feature of LCER is the short computation-time, especially compared to MBPO. This is not evident from the results in Figure 5.2 but will be discussed in Section 5.4.

5.3 Continuous Robotic Control Environments

This section details the experiment results for the continuous robotic control environments. Included are `FetchReach-v1`, `ShadowHandReach-v1`, `ShadowHandReachHard-v1` and `ShadowHandBlock-v1`, where the first one is part of the OpenAI Gym MuJoCo suite and the latter ones are PyBullet environments from ShadowHand-Gym. Below follows a brief description of the utilized environments, adapted from Plappert et al. [30] and Zahlner [67].

`FetchReach-v1`: This setup consists of a 7 degree-of-freedom Fetch robotic arm[1], which is equipped with a two-fingered parallel gripper. The objective is to move the gripper to a randomly specified 3-dimensional target position, given in Cartesian coordinates.

`ShadowHandReach-v1` and **`ShadowHandReachHard-v1`**: These environments are based on the Shadow Dexterous Hand[2], which is an anthropomorphic robotic hand with 24 degrees-of-freedom. 20 of these can be independently controlled, while the remaining ones are coupled. The task is a simple reaching task, in which the goal is to move the fingertips of the hand to a specific 15-dimensional target position, denoted by the target Cartesian coordinates of each fingertip.

`ShadowHandBlock-v1`: Like the `ShadowHandReach` environments, this environment is based on the Shadow Dexterous Hand. The block manipulation task is to manipulate and position a block on the palm of the hand to reach a specific target pose. The objective has 7 dimensions and is composed of the desired target position (in Cartesian coordinates) and rotation (in quaternions). The task is considered solved when the block reaches a certain orientation, regardless of its position above the hand palm.

In contrast to the previous section, the environments considered here focus more on typical robotic tasks, such as aligning a robot or a robot interacting with an object. Therefore, the rewards are binary and sparse, with a score of 0 given if the objective is achieved and -1 otherwise. In addition, it is also possible to define

[1] `https://fetchrobotics.com/`
[2] `https://www.shadowrobot.com/dexterous-hand-series/`

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

several different tasks within one environment, such as different robot end-effector positions that have to be reached.

The sparse reward setup benefits the use of Hindsight Experience Replay, as described in the Sections 2.4 and 2.4.4. Hence, all further used algorithms are jointly combined with HER. This also allows comparing to HER+SAC, which is one of the state-of-the-art RL algorithms for environments using sparse rewards. Vanilla SAC is only added as a baseline algorithm.

Figure 5.3 shows a visual representation of the described environments and Table 5.7 depicts the corresponding dimensions of the state and action spaces. In Figure 5.4 the mean success rate for all four robotic control environments is shown. The Tables 5.8, 5.9 and 5.10 provide additional per-timestep evaluations, comparing LCER and the other utilized RL algorithms. Detailed hyperparameters, used in the experiments, can be found in Appendix B.

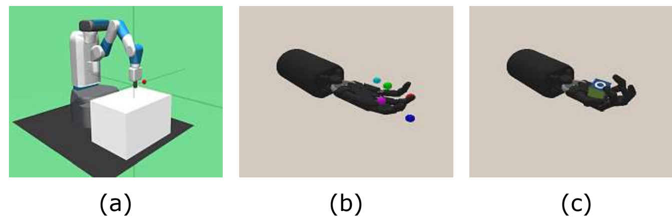

Figure 5.3: OpenAI Gym MuJoCo (Plappert et al. [30], Brockman et al. [65], Todorov et al. [66]) and ShadowHand-Gym PyBullet (Zahlner [67], Coumans and Bai [68]) continuous robotic control environments: (a) `FetchReach-v1`, (b) `ShadowHandReach-v1` and (c) `ShadowHandBlock-v1`. Adapted from Plappert et al. [30] and Zahlner [67].

Environment	No. of actions	No. of states
`FetchReach-v1`	4	16
`ShadowHandBlock-v1`	20	67
`ShadowHandReach-v1`	20	70
`ShadowHandReachHard-v1`	20	70

Table 5.7: State and action space dimensions for the robotic control OpenAI Gym MuJoco (Brockman et al. [65], Todorov et al. [66]) and ShadowHand-Gym PyBullet (Zahlner [67], Coumans and Bai [68]) environments.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

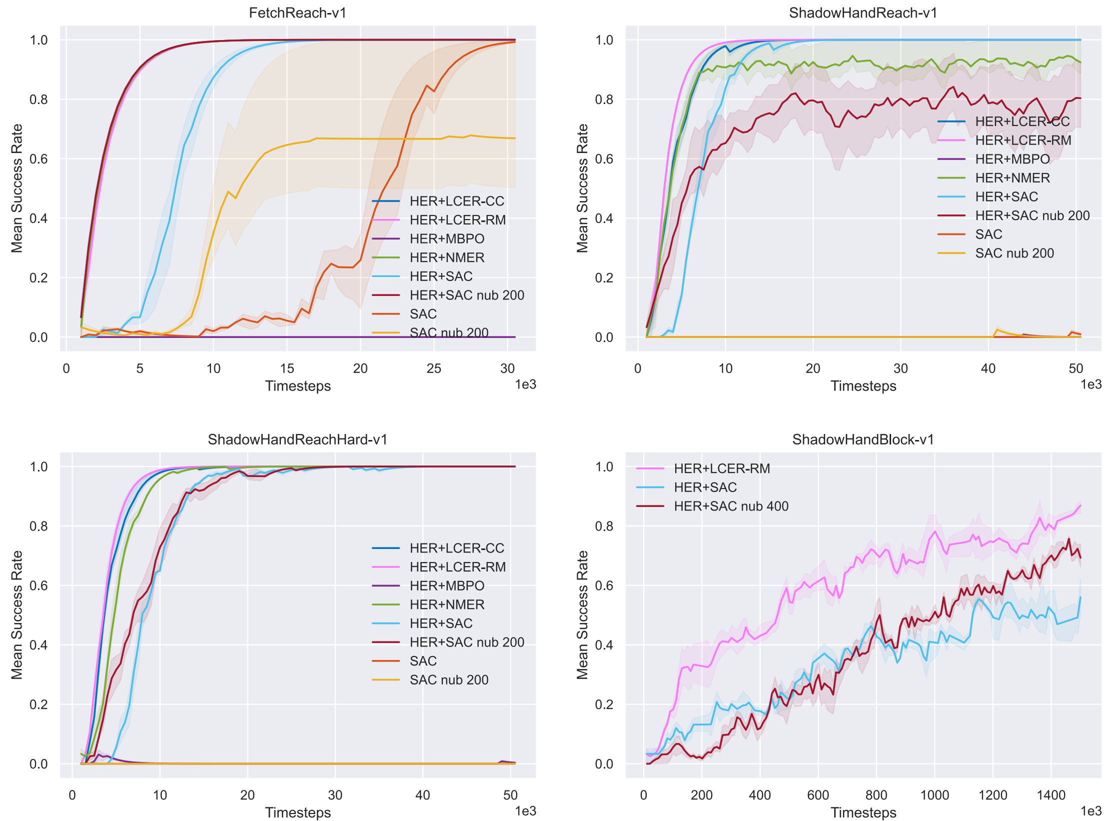

Figure 5.4: Evaluation results for the `FetchReach-v1` OpenAI Gym (Plappert et al. [30], Brockman et al. [65]) environment, which is part of the MuJoCo (Todorov et al. [66]) suite. `ShadowHandReach-v1`, `ShadowHandReachHard-v1` and `ShadowHandBlock-v1` are PyBullet (Coumans and Bai [68]) based environments from ShadowHand-Gym (Zahlner [67]). Results are reported as average success rate (line) and interquartile range (shaded area).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

	RL agent	Timesteps 10k μ	IQR	20k μ	IQR	30k μ	IQR
FetchReach	HER+LCER-CC	**99**	**0**	**100**	**0**	**100**	**0**
	HER+LCER-RM	**99**	**0**	**100**	**0**	**100**	**0**
	HER+MBPO	0	0	0	0	0	0
	HER+NMER	**99**	**0**	**100**	**0**	**100**	**0**
	HER+SAC	87	7	**100**	**0**	**100**	**0**
	HER+SAC nub 200	**99**	**0**	**100**	**0**	**100**	**0**
	SAC	2	2	26	25	99	1
	SAC nub 200	35	24	67	50	67	50

Table 5.8: Evaluation results for the `FetchReach-v1` OpenAI Gym MuJoCo (Plappert et al. [30], Brockman et al. [65], Todorov et al. [66]) environment for 10k, 20k and 30k timesteps. Results are reported as average success rate (μ) and interquartile range (IQR).

	RL agent	Timesteps 10k μ	IQR	20k μ	IQR	30k μ	IQR	40k μ	IQR	50k μ	IQR
SHR	HER+LCER-CC	98	1	**100**	**0**	**100**	**0**	**100**	**0**	**100**	**0**
	HER+LCER-RM	**99**	**0**	**100**	**0**	**100**	**0**	**100**	**0**	**100**	**0**
	HER+MBPO	0	0	0	0	0	0	0	0	0	0
	HER+NMER	91	12	93	9	91	14	93	11	93	10
	HER+SAC	84	9	**100**	**0**	**100**	**0**	**100**	**0**	**100**	**0**
	HER+SAC nub 200	65	12	80	22	80	20	79	23	80	23
	SAC	0	0	0	0	0	0	0	0	1	2
	SAC nub 200	0	0	0	0	0	0	0	0	0	0
SHR Hard	HER+LCER-CC	98	1	**100**	**0**	**100**	**0**	**100**	**0**	**100**	**0**
	HER+LCER-RM	**99**	**0**	**100**	**0**	**100**	**0**	**100**	**0**	**100**	**0**
	HER+MBPO	0	0	0	0	0	0	0	0	0	1
	HER+NMER	96	1	**100**	**0**	**100**	**0**	**100**	**0**	**100**	**0**
	HER+SAC	65	2	97	1	**100**	**0**	**100**	**0**	**100**	**0**
	HER+SAC nub 200	73	12	97	3	**100**	**0**	**100**	**0**	**100**	**0**
	SAC	0	0	0	0	0	0	0	0	0	0
	SAC nub 200	0	0	0	0	0	0	0	0	0	0

Table 5.9: Evaluation results for the `ShadowHandReach-v1 (SHR)` and `ShadowHandReachHard-v1 (SHR Hard)` PyBullet (Coumans and Bai [68]) based ShadowHand-Gym (Zahlner [67]) environments for 10k, 20k, 30k, 40k and 50k timesteps. Results are reported as average success rate (μ) and interquartile range (IQR).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

	RL agent	Timesteps									
		350k		650k		950k		1250k		1500k	
		μ	IQR	μ	IQR	μ	IQR	μ	IQR	μ	IQR
SHB	HER+LCER-RM	**46**	**7**	**57**	**10**	**68**	**10**	**74**	**3**	**87**	**4**
	HER+SAC	20	8	35	7	40	8	54	11	56	12
	HER+SAC nub 400	11	5	23	11	48	7	65	2	69	8

Table 5.10: Evaluation results for the `ShadowHandBlock-v1 (SHB)` PyBullet (Coumans and Bai [68]) based ShadowHand-Gym (Zahlner [67]) environment for 350k, 650k, 950k, 1250k and 1500k timesteps. Results are reported as average success rate (μ) and interquartile range (IQR).

`FetchReach-v1` is the continuous robotic control environment with the lowest state and action space dimensions compared to the `ShadowHandReach-v1`, `ShadowHandReachHard-v1` and `ShadowHandBlock-v1` environments. In the same way, but not conditioned to this, the task to be solved is the simplest one. Therefore, six out of eight methods are able to successfully solve the problem. Vanilla SAC with an increased update-to-data ratio is not able to fully learn the task within the given timesteps. Only HER+MBPO fails to make any progress at all. This is interesting since MBPO shows strong performance in the continuous locomotive control environments, see Section 5.2. The poor performance of HER+MBPO in most of the environments in this section is due to the use of sparse rewards, which are based on whether a target condition is met or not. The synthetic transitions taken from the dynamics model to evaluate the target condition can lead to incorrect rewards because of inaccuracies in the model. This can cause both false positive and false negative rewards, thus hindering proper learning.

The `ShadowHandReach-v1` and `ShadowHandReachHard-v1` environments both address the same task, the difference being in the details. In the former environment, five finger position patterns are specified as goal configurations. Changing the goal changes the entire pattern of all five fingers at once. In the `ShadowHandReachHard-v1` environment, the finger position patterns of each finger are randomly changed, allowing for complex poses. Nevertheless, the results for both are very similar. HER+LCER-CC and HER+LCER-RM perform the best, closely followed by HER+NMER. All three require significantly fewer timesteps to learn the task than vanilla HER+SAC. Again, as with `FetchReach-v1`, HER+MBPO does not solve the task. On top of the already mentioned sparse rewards, the high dimensions of the state and action spaces (see Table B.1) cause additional problems, amplifying the model uncertainties.

The `ShadowHandBlock-v1` environment is the most complex environment considered in this thesis. This is due to the use of the complex robot and the fact that it has to interact with an external object. It requires the longest training period, both in terms of time and number of timesteps, hence only HER+LCER-RM and HER+SAC in

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

two configurations are trained, tested, and compared. HER+LCER-RM clearly outperforms the other two competitors right from the start and performs consistently strong. This is due to the additional exploration that comes from mixup sampling within the different clusters. Increasing the update-to-data ratio of the vanilla HER+SAC policy improves the performance after 800k timesteps. However, it is still considerably worse than with LCER-RM.

As for the locomotion tasks from the previous Section 5.2, Local Cluster Experience Replay and especially LCER-RM show advantages for the considered robotic control environments. HER+LCER clearly outperforms vanilla HER+SAC and demonstrates at least the same, if not better performance as HER+NMER. In comparison, MBPO outperformed all other algorithms in the locomotion tasks, however, it was not able to solve most continuous robotic control environments. The experiments with the `ShadowHandReach-v1` and `ShadowHandReachHard-v1` environments show that LCER has no problems with high state and action space dimensions and the `ShadowHandBlock-v1` environment indicates that LCER can handle complex tasks very well.

5.4 Comparison of Computation-Time

In this final section, we want to compare the additional effort in terms of computation-time of the different proposed approaches. All methods from previous sections use the same underlying RL algorithms. The main difference is, how they process data of the replay-buffer to generate new and unseen transitions. Since this step has to be repeated at regular intervals, we want to compare how much computation-time it requires. The considered algorithms and operations can be summarized as follows.

LCER-CC: The first step involves constructing and standardizing the z-space, based on the latest batch of transitions from the environment. Next, the clusters are updated using the mini-batch k-Means algorithm. Finally, the data structures ($\mu(l_c)$ and $\sigma(l_c)$) to compute cluster centers containing all components of a transition quintuple, except for the terminal flag d_t, are updated[3]. The cluster update step of LCER-CC is described in Section 4.1.

LCER-RM: In a first step, similar to LCER-CC, the z-space is constructed and standardized. In contrast, all states and actions of the replay-buffer are used. Then, utilizing a standard, but GPU accelerated k-Means algorithm, the clusters are completely recomputed, based on these standardized states and actions. In the last step, references to the transitions in the replay-buffer are grouped, according to their cluster labels[4]. The cluster update step of LCER-RM is described in Section 4.2.

[3]These data structures are necessary to allow interpolation and mixup sampling with cluster centers, containing all components of environment transitions in a later step.

[4]This allows for easier access later on.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

NMER: The considered operational steps of NMER include the construction and standardization of the z-space and the computation of the nearest neighbors of every transition within the replay-buffer. The algorithm is described in Section 2.4.3.

MBPO: The computation-time comparison focuses only on the update step of the MDP dynamics model. This includes sampling real experiences from the replay-buffer, that are further utilized to train and optimize the dynamics model. For simplicity, we limit our analysis to this model update step, as the inclusion of virtual rollouts would add a number of parameters that require tuning and may distort the results. For more information about MBPO, see Section 2.5.1.

The computation-time comparison is done using three environments, each with a different amount of state and action space dimensions. This ranges from the low-dimensional `InvertedPendulum-v2` environment, to `Hopper-v2`, up to `ShadowHandReach-v1`, which is a representative of high-dimensional environments.

The baseline implementations of the mini-batch k-Means and the k-NN algorithms are both provided by the scikit-learn framework as described in Pedregosa et al. [43]. The same is true for the mini-batch k-Means accel variant, which uses another, faster approach to maintain the additional data structures $\mu(l_c)$ and $\sigma(l_c)$. In contrast, the GPU accelerated k-Means algorithm is based on the fast-pytorch-kmeans library from Omer [74], and MBPO uses its own approach, provided by the authors Xingyu-Lin [71]. Additionally, we also compare a GPU accelerated and tuned k-NN algorithm, as it is used in the implementation of NMER[5] and provided by the Facebook AI Similarity Search (FAISS) library from Johnson et al. [75].

At this point, it should be explicitly noted that scikit-learn only provides Central Processing Unit (CPU) based implementations. In the further course of this section, these are compared with GPU implementations, which highlights the advantages and disadvantages of both variants. Regarding MBPO, it is worth mentioning that on average the MDP dynamics model is adjusted more often than the recalculation of the clusters or nearest neighbors is performed. This is necessary to obtain a dynamics model that is as accurate as possible. In this experiment, the dynamics model is adjusted at the same ratio as the clusters and nearest neighbors are updated, but in practice, this varies across environments. This also increases the computation-time of MBPO by this factor. Figure 5.5 depicts the results and Table 5.11 provides additional per-timestep evaluations. Table B.1 shows the corresponding environment state and action space dimensions for reference.

It is noticeable that mini-batch k-Means (LCER-CC) outperforms most of its competitors and its accelerated variant (mini-batch k-Means accel) is comparable to FAISS k-NN (NMER). Mini-batch k-Means is faster in all three environments,

[5]The implementation of NMER includes both a variant with k-NN from scikit-learn and a variant with k-NN from FAISS.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

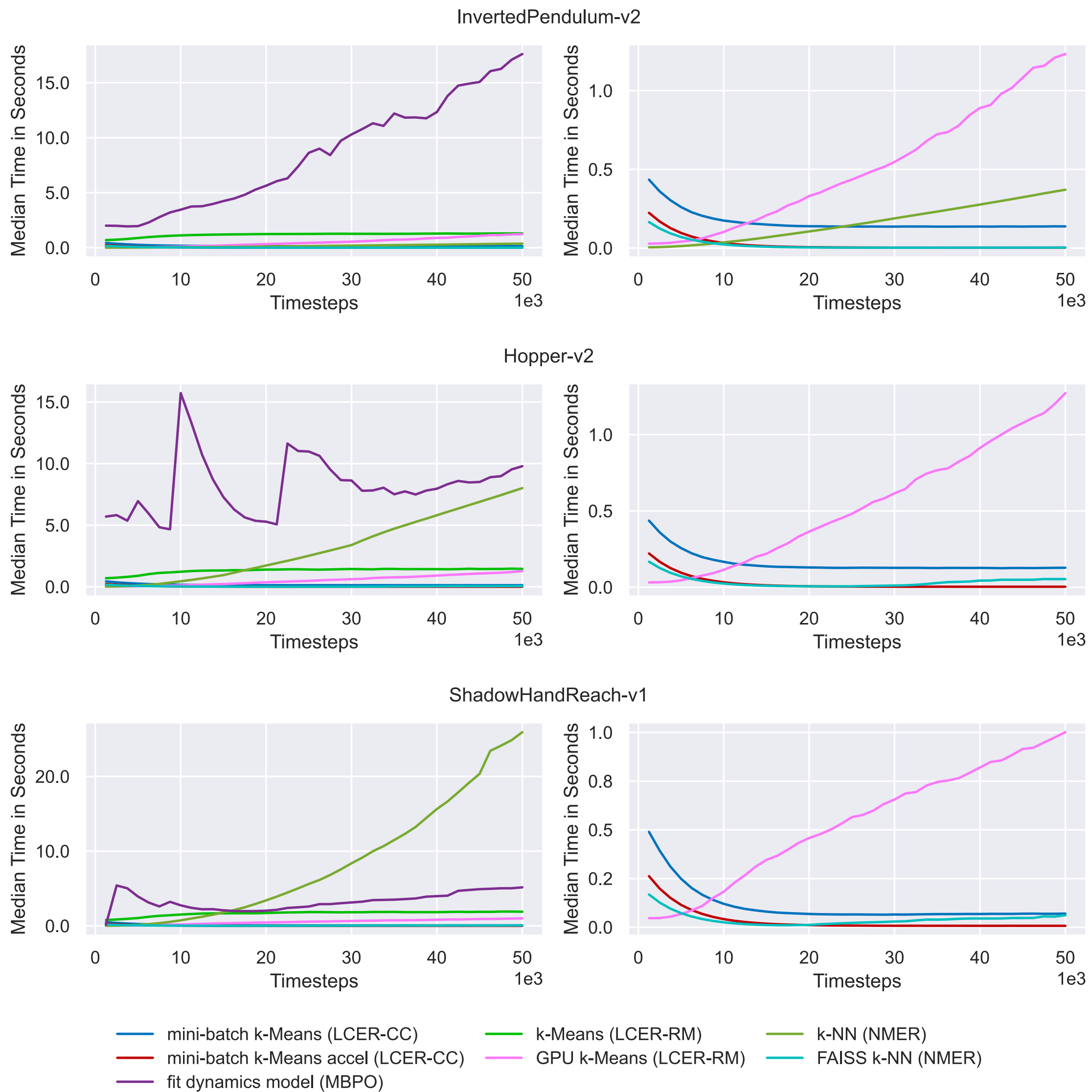

Figure 5.5: Evaluation results of the computation-time comparison. Results are reported as median computation-time over a corresponding amount of timesteps. The results are evaluated based on the OpenAI Gym MuJoCo (Brockman et al. [65], Todorov et al. [66]) environments `InvertedPendulum-v2` and `Hopper-v2` and the PyBullet (Coumans and Bai [68]) environment `ShadowHandReach-v1` from ShadowHand-Gym (Zahlner [67]). These range from low up to high state and action space dimensions, see Table B.1 for reference.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

except for the initial states with only a few timesteps. However, these are of little importance in reinforcement learning. The speed-up in computation-time from its design for stream-like input data is undeniable and the reason for its superior performance.

	Algorithm	Timesteps 10k	20k	30k	40k	50k
IP	mini-batch k-Means (LCER-CC)	2.15	3.35	4.45	5.54	6.64
	mini-batch k-Means accel (LCER-CC)	0.82	0.92	0.96	0.98	1.00
	k-Means (LCER-RM)	7.30	16.89	26.94	37.09	47.40
	GPU k-Means (LCER-RM)	0.43	2.23	5.82	11.68	20.42
	k-NN (NMER)	**0.14**	0.73	1.94	3.84	6.47
	FAISS k-NN (NMER)	0.60	**0.67**	**0.70**	**0.72**	**0.75**
	fit dynamics model (MBPO)	19.62	55.55	121.38	214.53	340.07
Hopper	mini-batch k-Means (LCER-CC)	2.12	3.23	4.25	5.26	6.27
	mini-batch k-Means accel (LCER-CC)	0.81	0.92	0.95	**0.98**	**1.01**
	k-Means (LCER-RM)	7.68	18.37	29.68	41.15	52.75
	GPU k-Means (LCER-RM)	**0.49**	2.45	6.48	12.72	21.52
	k-NN (NMER)	1.39	10.14	31.23	69.81	126.11
	FAISS k-NN (NMER)	0.61	**0.69**	**0.75**	**0.98**	1.38
	fit dynamics model (MBPO)	55.00	117.62	193.77	255.93	327.03
SHR	mini-batch k-Means (LCER-CC)	2.07	2.73	3.25	3.79	4.34
	mini-batch k-Means accel (LCER-CC)	0.98	1.13	1.20	1.26	**1.32**
	k-Means (LCER-RM)	9.15	22.51	37.05	51.81	66.90
	GPU k-Means (LCER-RM)	0.74	3.55	8.09	14.07	21.42
	k-NN (NMER)	2.15	18.60	66.51	163.41	335.75
	FAISS k-NN (NMER)	**0.62**	**0.72**	**0.91**	**1.23**	1.63
	fit dynamics model (MBPO)	26.21	43.14	64.75	93.56	132.21
		cumulative Time in Seconds				

Table 5.11: Evaluation results of the computation-time comparison. Results are reported as cumulative computation-time for 10k, 20k, 30k, 40k, and 50k timesteps. The results are evaluated based on the OpenAI Gym MuJoCo (Brockman et al. [65], Todorov et al. [66]) environments `InvertedPendulum-v2 (IP)` and `Hopper-v2` and the PyBullet (Coumans and Bai [68]) environment `ShadowHandReach-v1 (SHR)` from ShadowHand-Gym (Zahlner [67]). These range from low up to high state and action space dimensions, see Table B.1 for reference.

The computation-time of the baseline k-NN implementation can only keep up with the other approaches in the low-dimensional `InvertedPendulum-v2` environment. However, in `Hopper-v2` and `ShadowHandReach-v1`, the time required to compute and sort the distances to all transitions in the replay-buffer significantly exceeds

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

the time required to rearrange the cluster centers, as in mini-batch k-Means, or compute entirely new clusters with the (GPU) accelerated k-Means variant.

The update of the dynamics model of MBPO in the `InvertedPendulum-v2` environment is clearly outperformed by all other implementations. This outcome is expected as the environment has a low dimensionality, which favors the other approaches. However, as the state and action space dimensions increase, MBPO benefits from its model update strategy. It trains the dynamics model as long as there is improvement. In higher dimensions, the model accuracy decreases and so does the improvement. Despite the faster computation-time, the dynamics model is less accurate, hence, it needs to be updated more frequently, resulting in a higher overall computation-time.

The advantage of mini-batch k-Means is supported by the computational complexity of the considered methods. The standard k-NN algorithm has a complexity of $O(nd + kd)$, where n is the number of samples, d is the number of features, and k is the number of nearest neighbors. On the other hand, baseline k-Means has an average complexity of $O(knid)$, where i additionally denotes the number of iterations. Meanwhile, mini-batch k-Means has $O(k(n/b)id)$, with b being the batch size (Sculley [41], Pedregosa et al. [43]). For a large number of samples n, the complexity of k-NN grows linearly with the number of samples, while the complexity of k-Means grows sublinearly. This indicates that the relative performance of k-NN compared to k-Means decreases as the number of samples increases. Since mini-batch k-Means is less complex than k-Means, this is also true for this algorithm.

The performance and efficiency gain of the (GPU) accelerated and modified versions of the baseline algorithms is also evident in the experiments. This can be verified by the speed-up of FAISS k-NN compared to the standard k-NN algorithm. Further confirmation of this fact can be observed with the k-Means algorithms. Although the GPU k-Means is not the fastest in terms of performance, it still shows improvement compared to the baseline. However, it was not our goal to develop and implement a (mini-batch) k-Means algorithm that outperforms all others, as that would exceed the scope of this work. Nevertheless, we would like to point out that the utilized algorithms have been chosen with care, even if there is room for further improvement.

These experiments show that the computational overhead added by extending a standard replay-buffer with LCER is very low. This is true for both variants of LCER, and especially for environments with moderate to high state and action space dimensions. Compared to MBPO, both LCER-CC and LCER-RM show their advantages in terms of computation-time. Regarding NMER, the implementation of LCER-CC is at least equivalent in terms of required computation-time. This is also true compared to the implementation with FAISS k-NN.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

6 Conclusion

In this thesis, Local Cluster Experience Replay, an extension for replay-buffers of off-policy reinforcement learning algorithms, was introduced. The goal was to increase the sample efficiency of model-free RL algorithms by reducing the number of required interactions with the environment to maximize the cumulative reward. At the same time, the additional overhead in terms of implementation effort and computational power should be kept at a minimum. In this chapter, the strengths and weaknesses of the introduced approach, as well as possible future work are discussed. It is divided into topics concerning the implementation and results achieved with LCER.

6.1 Implementation

To address the need for low implementation and computational overhead, LCER was designed as a wrapper around the standard replay-buffer. This is true for both of its variants LCER-CC and LCER-RM. The introduction of k-Means clustering and mixup sampling to the replay-buffer of off-policy RL algorithms is a novel approach that creates locally linear models between different transitions. These allow new and unseen data to be synthetically generated by interpolation. The additional transitions can be used to support the policy and to increase the policy updates per environmental step to numbers that are typically unstable in model-free RL. In our implementation, we use between 10 and 40 policy update steps per environment step without the risk of becoming unstable.

LCER-CC interpolates between transitions and their corresponding cluster centers. This approach works well but has two disadvantages. First, the utilized mini-batch k-Means algorithm only adjusts the cluster centers but does not recalculate them in each cluster update step. This can lead to a worse fit of the clusters and thus lower the learning curve. Second, the interpolation of new data is always directed towards the according cluster centers. LCER-RM avoids these problems by recalculating the clusters in each cluster update step. Further, LCER-RM arbitrarily takes two transitions from the same cluster and interpolates between them. This also increases the exploration rate, resulting in better overall performance. Despite the need to refit the clusters in regular intervals, LCER-RM is still extremely fast in terms of wall time, due to the utilized GPU accelerated clustering algorithm.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

6.2 Results

Local Cluster Experience Replay and its two variants LCER-CC and LCER-RM show promising results. This is true for both tested environment domains. In the first domain, represented by continuous locomotive control tasks such as `Hopper-v2`, `Walker2d-v2` or `HalfCheetah-v2` from the OpenAI Gym suite, LCER performs consistently strong. It clearly outperforms the vanilla implementations of SAC and PER in almost all tested environments. In general, LCER-RM has better results than LCER-CC. This is due to the way the clusters are formed (complete refit vs. adjustments only) and which interpolation pairs are used (two random members vs. one-directional towards the center), see Sections 4.1, 4.2 and 6.1.

LCER is inspired by the implementation of Neighborhood Mixup Experience Replay. Based on the reward or success rate evaluations of the experiments, LCER is on par with NMER. The advantage of LCER comes into play in the time-based experiments. Here, the benefits of clustering over the k-NN algorithm are clearly evident, at least considering the baseline algorithms. This is supported by the computational complexity of both methods and is especially true for a large number of samples and environments with high-dimensional state and action spaces. Nevertheless, this advantage over NMER is reduced by the use of accelerated and modified versions of the baseline algorithms, such as FAISS k-NN.

Both variants of LCER are also compared to a state-of-the-art model-based RL algorithm, namely Model-Based Policy Optimization. Although LCER-RM achieves results equal to MBPO in the environments `Hopper-v2` and `Walker2d-v2`, it generally performs slightly worse. Nevertheless, LCER offers the advantage of faster computation-time, since it avoids the need to maintain an MDP dynamics model of the environment. Furthermore, this also bypasses the additional challenges, inevitably associated with a dynamics model.

In the second domain of the tested environments, of continuous robotic control tasks such as `FetchReach-v1` from the OpenAI Gym suite or `ShadowHandBlock-v1` from ShadowHand-Gym, LCER also demonstrates a strong performance. For these experiments, LCER was combined with Hindsight Experience Replay. On the one hand, this allows showing the possibility of combining these two methods, demonstrating the versatility and flexibility of LCER. On the other hand, it allows a fair comparison of HER+SAC and HER+LCER.

In all tested environments, including `FetchReach-v1`, `ShadowHandReach-v1`, `ShadowHandReachHard-v1` and `ShadowHandBlock-v1`, HER+LCER clearly outperforms vanilla HER+SAC. The experiments with the environments from ShadowHand-Gym show that LCER has no problems with high state and action space dimensions and especially the `ShadowHandBlock-v1` environment clarifies that LCER can handle complex tasks very well. This is in contrast to the results of HER+MBPO, which was not able to solve any of the robotic control tasks.

TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

6.3 Future Work

In this section, suggestions for further improvement of Local Cluster Experience Replay are presented. LCER is capable of processing data from a wide range of sources, regardless of its origin. Whether the data is generated internally or externally, LCER is able to seamlessly integrate and utilize it. This is particularly useful when working with pre-generated data. The data may originate from an expert (e.g. a human operator) or other sources like an MDP dynamics model of the environment. This allows the incorporation of additional knowledge and insights that may not be available otherwise and can further lead to an improvement in sample efficiency.

Expert data that contains complete trajectories, or step-by-step solutions to a given problem or task, is considered especially valuable and promising in this regard. By providing the algorithm with a clear road map, these types of data can help guide the system to a successful solution with a minimum of trial and error. This not only saves time but also allows the system to avoid unnecessary mistakes or detours that could lead to sub-optimal solutions. In this context, LCER can be combined with behavioral cloning methods. As an example, two prominent representatives Vecerik et al. [76] and Nair et al. [77] are mentioned. The former use demonstration data to pre-train the agent to perform well right from the beginning of the learning process. The latter introduce a separate loss function that takes into account and incorporates the data from demonstrations. In both cases, LCER could potentially provide a significant increase in sample efficiency.

Similarly, LCER could also be used in combination with model-based RL algorithms. Here, LCER could provide data in addition to the MDP dynamics model, or it could provide data for learning the environment model. Which of the two mentioned variants is better could be investigated in future work.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Bibliography

[1] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, „mixup: Beyond Empirical Risk Minimization,“ in *International Conference on Learning Representations*, 2018.

[2] R. Sander, W. Schwarting, T. Seyde, I. Gilitschenski, S. Karaman, and D. Rus, „Neighborhood Mixup Experience Replay: Local Convex Interpolation for Improved Sample Efficiency in Continuous Control Tasks,“ in *Learning for Dynamics and Control Conference*, PMLR, 2022, pp. 954–967.

[3] R. Sander, W. Schwarting, T. Seyde, I. Gilitschenski, S. Karaman, and D. Rus, „Neighborhood Mixup Experience Replay: Local Convex Interpolation for Improved Sample Efficiency in Continuous Control Tasks,“ Tech. Rep., 2022.

[4] J. Achiam, „Spinning Up in Deep Reinforcement Learning,“ 2018.

[5] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT press, 2018.

[6] R. Bellman, „A Markovian Decision Process,“ *Journal of Mathematics and Mechanics*, vol. 6, no. 5, pp. 679–684, 1957.

[7] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, 1st ed. New York, NY, USA: John Wiley & Sons, 1994.

[8] T. Anthony, Z. Tian, and D. Barber, „Thinking Fast and Slow with Deep Learning and Tree Search,“ in *Advances in Neural Information Processing Systems*, vol. 30, Curran Associates, Inc., 2017.

[9] D. Silver, T. Hubert, J. Schrittwieser, *et al.*, „A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play,“ *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[10] R. S. Sutton, „Dyna, an Integrated Architecture for Learning, Planning, and Reacting,“ *ACM Sigart Bulletin*, vol. 2, no. 4, pp. 160–163, 1991.

[11] D. Ha and J. Schmidhuber, „Recurrent World Models Facilitate Policy Evolution,“ in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31, Curran Associates, Inc., 2018.

[12] T. M. Moerland, J. Broekens, and C. M. Jonker, „Model-based Reinforcement Learning: A Survey.,“ *arXiv preprint arXiv:2006.16712*, 2020.

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

[13] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, „Trust Region Policy Optimization," in *International Conference on Machine Learning*, PMLR, 2015, pp. 1889–1897.

[14] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, „Proximal Policy Optimization Algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[15] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, „Playing Atari With Deep Reinforcement Learning," in *Neural Information Processing Systems Deep Learning Workshop*, 2013.

[16] M. G. Bellemare, W. Dabney, and R. Munos, „A Distributional Perspective on Reinforcement Learning," in *International Conference on Machine Learning*, PMLR, 2017, pp. 449–458.

[17] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT press, 2016.

[18] J. Schmidhuber, „Deep Learning in Neural Networks: An Overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[19] T. P. Lillicrap, J. J. Hunt, A. Pritzel, *et al.*, „Continuous control with deep reinforcement learning," in *International Conference on Learning Representations*, 2016.

[20] S. Fujimoto, H. Hoof, and D. Meger, „Addressing Function Approximation Error in Actor-Critic Methods," in *International Conference on Machine Learning*, PMLR, 2018, pp. 1587–1596.

[21] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, „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.

[22] A. Raffin, *RL Baselines3 Zoo*, `https://github.com/DLR-RM/rl-baselines3-zoo`, 2020.

[23] T. Wang, X. Bao, I. Clavera, *et al.*, „Benchmarking Model-Based Reinforcement Learning," *arXiv preprint arXiv:1907.02057*, 2019.

[24] W. Fedus, P. Ramachandran, R. Agarwal, *et al.*, „Revisiting Fundamentals of Experience Replay," in *International Conference on Machine Learning*, PMLR, 2020, pp. 3061–3071.

[25] X. Chen, C. Wang, Z. Zhou, and K. Ross, „Randomized Ensembled Double Q-Learning: Learning Fast Without a Model," *arXiv preprint arXiv:2101.05982*, 2021.

[26] M. Janner, J. Fu, M. Zhang, and S. Levine, „When to Trust Your Model: Model-Based Policy Optimization," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[27] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, „Prioritized Experience Replay," in *International Conference on Learning Representations*, 2016.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[28] D. Horgan, J. Quan, D. Budden, *et al.*, „Distributed Prioritized Experience Replay,“ in *International Conference on Learning Representations*, 2018.

[29] M. Andrychowicz, F. Wolski, A. Ray, *et al.*, „Hindsight Experience Replay,“ *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[30] M. Plappert, M. Andrychowicz, A. Ray, *et al.*, „Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research,“ *arXiv preprint arXiv:1802.09464*, 2018.

[31] T. Schaul, D. Horgan, K. Gregor, and D. Silver, „Universal Value Function Approximators,“ in *International Conference on Machine Learning*, PMLR, 2015, pp. 1312–1320.

[32] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, „Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning,“ in *International Conference on Robotics and Automation*, IEEE, 2018, pp. 7559–7566.

[33] K. Chua, R. Calandra, R. McAllister, and S. Levine, „Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models,“ *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[34] J. Wang, A. Hertzmann, and D. J. Fleet, „Gaussian Process Dynamical Models,“ *Advances in Neural Information Processing Systems*, vol. 18, 2005.

[35] C. Gadd, M. Heinonen, H. Lähdesmäki, and S. Kaski, „Sample-Efficient Reinforcement Learning using Deep Gaussian Processes,“ *arXiv preprint arXiv:2011.01226*, 2020.

[36] D. Müllner, „Modern hierarchical, agglomerative clustering algorithms,“ *arXiv preprint arXiv:1109.2378*, 2011.

[37] M. Roux, „A comparative study of divisive hierarchical clustering algorithms,“ *arXiv preprint arXiv:1506.08977*, 2015.

[38] P. K. Agarwal and N. H. Mustafa, „k-Means Projective Clustering,“ in *Proceedings of the Twenty-Third ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2004, pp. 155–165.

[39] D. Arthur and S. Vassilvitskii, „k-means++: The Advantages of Careful Seeding,“ Stanford, Tech. Rep., 2006.

[40] J. Newling and F. Fleuret, „K-Medoids For K-Means Seeding,“ *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[41] D. Sculley, „Web-Scale K-Means Clustering,“ in *Proceedings of the 19th International Conference on World Wide Web*, 2010, pp. 1177–1178.

[42] S. Lloyd, „Least Squares Quantization in PCM,“ *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

[43] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, „Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[44] W. Miller, R. Hewes, F. Glanz, and L. Kraft, „Real-time dynamic control of an industrial manipulator using a neural network-based learning controller," *IEEE Transactions on Robotics and Automation*, vol. 6, no. 1, pp. 1–9, 1990.

[45] J. Schmidhuber, „An On-Line Algorithm for Dynamic Reinforcement Learning and Planning in Reactive Environments," in *1990 IJCNN International Joint Conference on Neural Networks*, IEEE, 1990, pp. 253–258.

[46] W. Whitney and R. Fergus, „Understanding the Asymptotic Performance of Model-Based RL Methods," 2018.

[47] L. P. Kaelbling, M. L. Littman, and A. W. Moore, „Reinforcement Learning: A Survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[48] J. Buckman, D. Hafner, G. Tucker, E. Brevdo, and H. Lee, „Sample-Efficient Reinforcement Learning with Stochastic Ensemble Value Expansion," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[49] M. Deisenroth and C. E. Rasmussen, „PILCO: A Model-Based and Data-Efficient Approach to Policy Search," in *International Conference on Machine Learning*, 2011, pp. 465–472.

[50] S. Depeweg, J. M. Hernández-Lobato, F. Doshi-Velez, and S. Udluft, „Learning and Policy Search in Stochastic Dynamical Systems with Bayesian Neural Networks," in *International Conference on Learning Representations*, 2017.

[51] L. V. Jospin, H. Laga, F. Boussaid, W. Buntine, and M. Bennamoun, „Hands-On Bayesian Neural Networks—A Tutorial for Deep Learning Users," *IEEE Computational Intelligence Magazine*, vol. 17, no. 2, pp. 29–48, 2022.

[52] J. Kocijan, R. Murray-Smith, C. E. Rasmussen, and A. Girard, „Gaussian Process Model Based Predictive Control," in *Proceedings of the 2004 American Control Conference*, IEEE, vol. 3, 2004, pp. 2214–2219.

[53] S. Levine, C. Finn, T. Darrell, and P. Abbeel, „End-to-End Training of Deep Visuomotor Policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[54] E. Todorov and W. Li, „A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *Proceedings of the 2005, American Control Conference, 2005.*, IEEE, 2005, pp. 300–306.

[55] D. Silver, H. Hasselt, M. Hessel, *et al.*, „The Predictron: End-To-End Learning and Planning," in *International Conference on Machine Learning*, PMLR, 2017, pp. 3191–3199.

[56] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel, „Value Iteration Networks," *Advances in Neural Information Processing Systems*, vol. 29, 2016.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[57] W. Sun, G. J. Gordon, B. Boots, and J. Bagnell, „Dual Policy Iteration," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[58] D. Q. Mayne and H. Michalska, „Receding horizon control of nonlinear systems," in *Proceedings of the 27th IEEE Conference on Decision and Control*, IEEE, 1988, pp. 464–465.

[59] A. Draeger, S. Engell, and H. Ranke, „Model Predictive Control Using Neural Networks," *IEEE Control Systems Magazine*, vol. 15, no. 5, pp. 61–66, 1995.

[60] A. Grancharova, J. Kocijan, and T. A. Johansen, „Explicit stochastic predictive control of combustion plants based on Gaussian process models," *Automatica*, vol. 44, no. 6, pp. 1621–1631, 2008.

[61] B. Amos, I. Jimenez, J. Sacks, B. Boots, and J. Z. Kolter, „Differentiable MPC for End-to-end Planning and Control," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[62] Z. I. Botev, D. P. Kroese, R. Y. Rubinstein, and P. L'Ecuyer, „The Cross-Entropy Method for Optimization," in *Handbook of Statistics*, vol. 31, Elsevier, 2013, pp. 35–59.

[63] A. Pourchot and O. Sigaud, „CEM-RL: Combining evolutionary and gradient-based methods for policy search," in *International Conference on Learning Representations*, 2019.

[64] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, „Continuous Deep Q-Learning with Model-based Acceleration," in *International Conference on Machine Learning*, PMLR, 2016, pp. 2829–2838.

[65] G. Brockman, V. Cheung, L. Pettersson, *et al.*, „OpenAI Gym," *arXiv preprint arXiv:1606.01540*, 2016.

[66] E. Todorov, T. Erez, and Y. Tassa, „MuJoCo: A physics engine for model-based control," in *2012 IEEE/RSJ international conference on intelligent robots and systems*, IEEE, 2012, pp. 5026–5033.

[67] S. Zahlner, *ShadowHand-Gym*, `https://github.com/szahlner/shadowhand-gym`, 2021.

[68] E. Coumans and Y. Bai, *PyBullet, a Python module for physics simulation for games, robotics and machine learning*, `http://pybullet.org`, 2021.

[69] pranz24, *pytorch-soft-actor-critic*, `https://github.com/pranz24/pytorch-soft-actor-critic`, 2019.

[70] MrSyee, *PG is all you need!* `https://github.com/MrSyee/pg-is-all-you-need`, 2019.

[71] Xingyu-Lin, *mbpo_pytorch*, `https://github.com/Xingyu-Lin/mbpo_pytorch`, 2021.

[72] rmsander, *Neighborhood Mixup Experience Replay (NMER)*, `https://github.com/rmsander/interreplay`, 2022.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[73] TianhongDai, *Hindsight Experience Replay (HER)*, https://github.com/TianhongDai/hindsight-experience-replay, 2019.

[74] S. Omer, *Fast Pytorch Kmeans*, https://github.com/DeMoriarty/fast_pytorch_kmeans, 2020.

[75] J. Johnson, M. Douze, and H. Jégou, „Billion-scale similarity search with GPUs,“ *IEEE Transactions on Big Data*, vol. 7, no. 3, pp. 535–547, 2019.

[76] M. Vecerik, T. Hester, J. Scholz, *et al.*, „Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards,“ *arXiv preprint arXiv:1707.08817*, 2017.

[77] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, „Overcoming Exploration in Reinforcement Learning with Demonstrations,“ in *IEEE International Conference on Robotics and Automation*, IEEE, 2018, pp. 6292–6299.

[78] NVIDIA, P. Vingelmann, and F. H. Fitzek, *CUDA*, https://developer.nvidia.com/cuda-toolkit, 2020.

[79] C. R. Harris, K. J. Millman, S. J. van der Walt, *et al.*, „Array programming with NumPy,“ *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020.

[80] A. Paszke, S. Gross, F. Massa, *et al.*, „PyTorch: An Imperative Style, High-Performance Deep Learning Library,“ in *Advances in Neural Information Processing Systems*, vol. 32, Curran Associates, Inc., 2019.

[81] Martín Abadi, Ashish Agarwal, Paul Barham, *et al.*, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, Software available from tensorflow.org, 2015. [Online]. Available: https://www.tensorflow.org/.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

A Software Components

The implementation of this thesis is based on a handful of selected packages, which are briefly listed below. The code of LCER is open source and freely available at `https://github.com/szahlner/lcer`.

- **CUDA** (NVIDIA et al. [78]): Used for GPU based hardware acceleration.
- **MuJoCo** (Todorov et al. [66]): A physics simulator that is used in the OpenAI Gym environments.
- **NumPy** (Harris et al. [79]): A library for numerical calculations. It is mainly used for processing arrays and all non-tensor-related objects.
- **OpenAI Gym** (Brockman et al. [65]): A collection of environments for RL. Among them, we use the continuous locomotive control environments `InvertedPendulum-v2`, `Hopper-v2`, `Walker2d-v2`, `Ant-v2` and `HalfCheetah-v2`, and `FetchReach-v1` from the robotic environments.
- **PyBullet** (Coumans and Bai [68]): A physics simulator that is used in the ShadowHand-Gym environments.
- **PyTorch** (Paszke et al. [80]): A fully featured framework for building deep learning models with Python. It is used for processing all tensor-related objects.
- **Scikit-Learn** (Pedregosa et al. [43]): A library for predictive data analysis. It is mainly used to create the k-Means objects, as well as the nearest neighbor object.
- **ShadowHand-Gym** (Zahlner [67]): A collection of environments for RL. Among them, we use `ShadowHandReach-v1`, `ShadowHandReachHard-v1` and `ShadowHandBlock-v1` in our implementation.
- **TensorFlow** (Abadi et al. [81]): An open source framework for performing machine learning and other statistical and predictive analysis tasks. This implementation uses only the Tensorboard library of this framework.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

B Hyperparameters

This section specifies the parametrization for the RL agents and algorithms used in the experiments. Table B.2 depicts the hyperparameters for the OpenAI Gym continuous locomotive control environments and Table B.3 shows the parametrization for the OpenAI Gym MuJoCo and ShadowHand-Gym PyBullet continuous robotic control environments.

Table B.1 depicts the state and action space dimensions for all OpenAI Gym MuJoCo and ShadowHand-Gym PyBullet environments, as well as the maximum episode length, measured in timesteps.

Environment	No. of actions	No. of states	Max. episode length
`AntTruncated-v2`	8	27	1000
`HalfCheetah-v2`	6	17	1000
`Hopper-v2`	3	11	1000
`InvertedPendulum-v2`	1	4	1000
`Walker2d-v2`	6	17	1000
`FetchReach-v1`	4	$10 + 3 + 3 = 16$	50
`ShadowHandBlock-v1`	20	$40 + 13 + 7 + 7 = 67$	100
`ShadowHandReach-v1`	20	$40 + 15 + 15 = 70$	50
`ShadowHandReachHard-v1`	20	$40 + 15 + 15 = 70$	50

Table B.1: State and action space dimensions and maximum episode length, measured in timesteps, for the OpenAI Gym MuJoco (Brockman et al. [65], Todorov et al. [66]) and ShadowHand-Gym PyBullet (Zahlner [67], Coumans and Bai [68]) environments.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

RL agent	Hyperparameter	Value				
LCER	Gradient steps per environment step	10		20		40
	k-clusters			1000		
	Noise weight (α)			0.01		
MBPO	Epoch length			1000		
	Gradient steps per environment step	10		20		40
	Model activation function			Swish		
	Model hidden units			[200, 200, 200, 200, 200]		
	Model retain epochs			1		
	Rollout max epoch	15	150		100	150
	Rollout min epoch	1		20		
	Rollout max length	1	15	1	25	1
	Rollout min length			1		
	Rollout samples			100000		
	Update model every steps			250		
	V-ratio			0.95		
NMER	Gradient steps per environment step	10		20		40
	k-neighbors			10		
PER	Alpha (α)			0.4		
	Beta (β)			0.6		
	Priority eps			$1 \cdot 10^{-6}$		
SAC	Actor, Critic activation function			ReLU		
	Actor, Critic hidden units			[256, 256]		
	Actor, Critic learning rate			$3 \cdot 10^{-4}$		
	Alpha learning rate			$3 \cdot 10^{-4}$		
	Clip actions			False		
	Entropy target	-0.05	-1	-3	-4	-3
	Gamma (γ)			0.99		
	Initial entropy (α)			0.2		
	N-step			1		
	Normalize actions			False		
	Polyak coefficient (τ)			$5 \cdot 10^{-3}$		
	Replay-buffer size			$1 \cdot 10^{6}$		
	Target network update per gradient step			1		
	Training batch size			256		
	Twin-Q			True		
Environment		`IP`	`Hop`	`W2d`	`Ant`	`HC`

Table B.2: Evaluation hyperparameters for the `InvertedPendulum-v2` (`IP`), `Hopper-v2` (`Hop`), `Walker2d-v2` (`W2d`), `AntTruncated-v2` (`Ant`) and `HalfCheetah-v2` (`HC`) OpenAI Gym MuJoCo (Brockman et al. [65], Todorov et al. [66]) environments.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

<table>
<tr><th>RL agent</th><th>Hyperparameter</th><th colspan="4">Value</th></tr>
<tr><td rowspan="4">HER</td><td>Gradient steps per environment episode</td><td colspan="4">20</td></tr>
<tr><td>Normalizing states and goals</td><td>False</td><td colspan="3">True</td></tr>
<tr><td>Replay k</td><td colspan="4">4</td></tr>
<tr><td>Replay strategy</td><td colspan="4">future</td></tr>
<tr><td rowspan="3">LCER</td><td>Gradient steps per environment episode</td><td colspan="3">200</td><td>400</td></tr>
<tr><td>k-clusters</td><td colspan="3">50</td><td>100</td></tr>
<tr><td>Noise weight (α)</td><td colspan="3">0.01</td><td>-</td></tr>
<tr><td rowspan="12">MBPO</td><td>Epoch length</td><td colspan="3">1000</td><td>-</td></tr>
<tr><td>Gradient steps per environment episode</td><td colspan="3">200</td><td>-</td></tr>
<tr><td>Model activation function</td><td colspan="3">Swish</td><td>-</td></tr>
<tr><td>Model hidden units</td><td colspan="3">[200, 200, 200, 200, 200]</td><td>-</td></tr>
<tr><td>Model retain epochs</td><td colspan="3">1</td><td>-</td></tr>
<tr><td>Rollout max epoch</td><td colspan="3">1</td><td>-</td></tr>
<tr><td>Rollout min epoch</td><td colspan="3">0</td><td>-</td></tr>
<tr><td>Rollout max length</td><td colspan="3">3</td><td>-</td></tr>
<tr><td>Rollout min length</td><td colspan="3">3</td><td>-</td></tr>
<tr><td>Rollout samples</td><td colspan="3">100000</td><td>-</td></tr>
<tr><td>Update model every steps</td><td colspan="3">250</td><td>-</td></tr>
<tr><td>V-ratio</td><td colspan="3">0.95</td><td>-</td></tr>
<tr><td rowspan="2">NMER</td><td>Gradient steps per environment episode</td><td colspan="3">200</td><td>-</td></tr>
<tr><td>k-neighbors</td><td colspan="3">10</td><td>-</td></tr>
<tr><td rowspan="15">SAC</td><td>Actor, Critic activation function</td><td colspan="4">ReLU</td></tr>
<tr><td>Actor, Critic hidden units</td><td colspan="4">[256, 256, 256]</td></tr>
<tr><td>Actor, Critic learning rate</td><td colspan="4">$1 \cdot 10^{-3}$</td></tr>
<tr><td>Alpha learning rate</td><td colspan="4">$1 \cdot 10^{-3}$</td></tr>
<tr><td>Clip actions</td><td colspan="4">False</td></tr>
<tr><td>Entropy target</td><td>0.2</td><td colspan="3">0.01</td></tr>
<tr><td>Gamma (γ)</td><td colspan="4">0.98</td></tr>
<tr><td>Initial entropy (α)</td><td>0.2</td><td colspan="3">0.01</td></tr>
<tr><td>N-step</td><td colspan="4">1</td></tr>
<tr><td>Normalize actions</td><td colspan="4">False</td></tr>
<tr><td>Polyak coefficient (τ)</td><td colspan="4">$5 \cdot 10^{-3}$</td></tr>
<tr><td>Replay-buffer size</td><td colspan="4">$1 \cdot 10^{6}$</td></tr>
<tr><td>Target network update per gradient step</td><td colspan="4">1</td></tr>
<tr><td>Training batch size</td><td colspan="4">256</td></tr>
<tr><td>Twin-Q</td><td colspan="4">True</td></tr>
<tr><td colspan="2">Environment</td><td>FR</td><td>SHR</td><td>SHRH</td><td>SHB</td></tr>
</table>

Table B.3: Evaluation hyperparameters for the `FetchReach-v1 (FR)` OpenAI Gym MuJoCo (Brockman et al. [65], Todorov et al. [66]) and `ShadowHandReach-v1 (SHR)`, `ShadowHandReachHard-v1 (SHRH)` and `ShadowHandBlock-v1 (SHB)` ShadowHand-Gym PyBullet (Zahlner [67], Coumans and Bai [68]) environments.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

C Algorithms

C.1 Deep Deterministic Policy Gradient

Algorithm C.1 Deep Deterministic Policy Gradient (DDPG).
Adapted from Achiam [4].

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

$$y\left(r, s', d\right) = r + \gamma\left(1 - d\right) Q_{\theta_{\text{targ}}}\left(s', \mu_{\theta_{\text{targ}}}\left(s'\right)\right)$$

13: Update Q-function by one step of gradient descent using

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

14: Update policy by one step of the gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\theta\left(s, \mu_\theta\left(s\right)\right)$$

15: Update target networks with

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1 - \rho)\phi$$
$$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1 - \rho)\theta$$

16: **end for**
17: **end if**
18: **until** convergence

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

C.2 Soft Actor-Critic

Algorithm C.2 Soft Actor-Critic (SAC).
Adapted from Achiam [4].

1: **Input**: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay-buffer $\mathcal{D}$
2: Set target parameters equal to main parameters $\theta_{\text{targ},1} \leftarrow \theta_1, \phi_{\text{targ},1} \leftarrow \phi_1$
3: **repeat**
4: Observe state s and select action $a \sim \pi_\theta (\cdot|s)$
5: Execute a in the environment
6: Observe next state s', reward r and done signal d to indicate whether s' is terminal
7: Store (s, a, r, s', d) in replay-buffer $\mathcal{D}$
8: If s' is terminal, reset environment state
9: **if** it's time to update **then**
10: **for** however many updates **do**
11: Randomly sample a batch of transitions $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
12: Compute targets for the Q-functions:

$$y (r, s', d) = r + \gamma (1 - d) \left(\min_{i=1,2} Q_{\phi_{\text{targ},i}} (s', \tilde{a}') - \alpha \log \pi_\theta (\tilde{a}'|s') \right), \quad \tilde{a}' \sim \pi_\theta (\cdot|s')$$

13: Update Q-functions by one step of gradient descent using

$$\nabla_\theta \frac{1}{|B|} \sum_{(s,a,r,s',d)\in B} (Q_\theta (s, a) - y (r, s', d))^2, \quad \text{for } i = 1{,}2$$

14: Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{(s)\in B} \left(\min_{i=1,2} Q_{\phi_i} (s, \tilde{a}_\theta (s)) - \alpha \log \pi_\theta (\tilde{a}_\theta (s) |s) \right)$$

where $\tilde{a}_\theta (s)$ is a sample from $\pi_\theta (\cdot|s)$ which is differentiable w.r.t. θ via the reparameterization trick
15: Update target networks with

$$\phi_{\text{targ},i} \leftarrow \rho \phi_{\text{targ},i} + (1 - \rho) \phi_i, \quad \text{for } i = 1{,}2$$

16: **end for**
17: **end if**
18: **until** convergence

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

C.3 Model-Based Policy Optimization

Algorithm C.3 Model-Based Policy Optimization (MBPO).
Adapted from Janner et al. [26].

1: Initialize policy π_θ, predictive model p_θ, environment dataset $\mathcal{D}_{env}$, model dataset $\mathcal{D}_{model}$
2: **for** N epochs **do**
3: Train model p_θ on $\mathcal{D}_{env}$ via maximum likelihood
4: **for** E steps **do**
5: Take action in environment according to π_θ; add to $\mathcal{D}_{env}$
6: **for** M model rollouts **do**
7: Sample s_t uniformly from $\mathcal{D}_{env}$
8: Perform k-step model rollout starting from s_t using policy π_ϕ; add to $\mathcal{D}_{model}$
9: **for** G gradient updates **do**
10: Update policy parameters on model data

$$\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi \left(\phi, \mathcal{D}_{model}\right)$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

C.4 Prioritized Experience Replay

Algorithm C.4 Prioritized Experience Replay (PER).
Adapted from Schaul et al. [27].

1: **Input**: mini-batch k, step-size η, replay period K and size N, exponents α and β, budget T
2: Initialize replay-buffer $\mathcal{D}$, $\Delta = 0$, $p_1 = 1$
3: Observe s_0 and choose $a_0 \sim \pi_\theta(s_0)$
4: **for** $t = 1$ to T **do**
5: Observe s_t, r_t, γ_t
6: Store transition $(s_{t-1}, a_{t-1}, r_t, \gamma_t, s_t)$ in $\mathcal{D}$ with maximal priority $p_t = \max_{k<t} p_k$
7: **if** $t \equiv 0 \bmod K$ **then**
8: **for** $n = 1$ to m **do**
9: Sample transition

$$n \sim P(n) = \frac{p_n^\alpha}{\sum_k p_k^\alpha}$$

10: Compute importance-sampling weight

$$w_n = \left(\frac{1}{N} \cdot \frac{1}{P(n)}\right)^\beta \cdot \frac{1}{\max_n w_n}$$

11: Compute TD-error

$$\delta_n = R_n + \gamma_n Q_{\text{targ}}\Big(s_n, \arg\max_a Q(s_n, a) - Q(s_{n-1}, a_{n-1})\Big)$$

12: Update transition priority $p_n \leftarrow |\delta_n|$
13: Accumulate weight change

$$\Delta \leftarrow \Delta + w_n \delta_n \nabla_\theta Q(s_{n-1}, a_{n-1})$$

14: **end for**
15: Update weights $\theta \leftarrow \theta + \eta \cdot \Delta$, reset $\Delta = 0$
16: From time to time copy weights into target network $\theta_{\text{targ}} \leftarrow \theta$
17: **end if**
18: Choose action $a_t \sim \pi_\theta(s_t)$
19: **end for**

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

C.5 Neighborhood Mixup Experience Replay

Algorithm C.5 Neighborhood Mixup Experience Replay (NMER). Adapted from Sander et al. [3].

1: **Input**: replay-buffer $\mathcal{D}$, mixup hyperparameter $\alpha > 0$, batch size N
2: **Output**: interpolated training batch $\mathcal{B}_{\text{train}}$
3: Sample batch $\mathcal{B}$ uniformly from replay-buffer

$$\mathcal{B} = \left\{(s_t, a_t, r_t, s_t^{'})\right\}_{t=1}^{N} \overset{iid}{\sim} \mathcal{U}(\mathcal{D})$$

4: Pre-allocate training batch $\mathcal{B}_{\text{train}} \leftarrow \text{array}()$
5: **for** t in N **do**
6: Sample transition for NMER

$$(s_s, a_s, r_s, s_s^{'}) \leftarrow \mathcal{B}[t]$$

7: Standardize states and actions of sampled transition

$$(\tilde{s}_s, \tilde{a}_s) \leftarrow \text{ZScore}(s_s, a_s)$$

8: Standardized local neighborhood of sampled transition

$$K_s \leftarrow \text{NN}\Big((\tilde{s}_s, \tilde{a}_s), \mathcal{B}\Big)$$

9: Sample neighboring transition from local neighborhood

$$(s_n, a_n, r_n, s_n^{'}) \sim \mathcal{U}(K_s)$$

10: Sample mixup coefficient $\lambda \sim \beta(\alpha, \alpha)$
11: Sampled transition features $x_s \leftarrow (s_s, a_s, r_s, s_s^{'})$
12: Neighboring transition features $x_n \leftarrow (s_n, a_n, r_n, s_n^{'})$
13: Interpolate sampled and neighboring transitions using mixup

$$x_i = \lambda x_s + (1 - \lambda) x_n$$

14: Add interpolated sample to training batch $\mathcal{B}_{\text{train}}[t] \leftarrow x_i$
15: **end for**
16: **return** $\mathcal{B}_{\text{train}}$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

C.6 Hindsight Experience Replay

Algorithm C.6 Hindsight Experience Replay (HER).
Adapted from Andrychowicz et al. [29].

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

2: Initialize $\mathbb{A}$, replay-buffer $\mathcal{D}$
3: **for** episode = 1, M **do**
4: Sample a goal g and an initial state s_0
5: **for** t = 0, $T-1$ **do**
6: Sample an action a_t using the behavioral policy from $\mathbb{A}$

$$a_t \leftarrow \pi_\theta(s_t||g)$$

7: Execute the action a_t and observe a new state s_{t+1}
8: **end for**
9: **for** t = 0, $T-1$ **do**
10: $r_t := r(s_t, a_t, g)$
11: Store the transition $(s_t||g, a_t, r_t, s_{t+1}||g)$ in $\mathcal{D}$
12: Sample a set of additional goals for replay $G := \mathbb{S}($**current episode**$)$
13: **for** $g' \in G$ **do**
14: $r' := r(s_t, a_t, g')$
15: Store the transition $(s_t||g', a_t, r', s_{t+1}||g')$ in $\mathcal{D}$
16: **end for**
17: **end for**
18: **for** t = 1, N **do**
19: Sample a mini-batch B from the replay-buffer $\mathcal{D}$
20: Perform one step of optimization using $\mathbb{A}$ and mini-batch B
21: **end for**

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

C.7 Mini-batch k-Means

Algorithm C.7 Mini-batch k-Means.
Adapted from Sculley [41].

1: **Given**: k, mini-batch size b, iterations t, dataset X
2: Initialize each $cc \in C$ with an x picked randomly from X
3: $v \leftarrow 0$
4: **for** $n = 1$ to t **do**
5: $\quad M \leftarrow b$ examples picked randomly from X
6: $\quad$ **for** $x \in M$ **do**
7: $\quad\quad d[x] \leftarrow f(C, x)$ // Cache the center nearest to x
8: $\quad$ **end for**
9: $\quad$ **for** $x \in M$ **do**
10: $\quad\quad cc \leftarrow d[x]$ // Get cached center for this x
11: $\quad\quad v[cc] \leftarrow v[cc] + 1$ // Update per-center counts
12: $\quad\quad \eta \leftarrow \frac{1}{v[cc]}$ // Get per-center learning rate
13: $\quad\quad cc \leftarrow (1 - \eta)cc + \eta x$ // Take gradient step
14: $\quad$ **end for**
15: **end for**

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Erklärung

Hiermit erkläre ich, dass die vorliegende Arbeit gemäß dem Code of Conduct - Regeln zur Sicherung guter wissenschaftlicher Praxis, insbesondere ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt wurde. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Wien, im Februar 2023

Stefan Zahlner

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub