TU WIEN Informatics

Self-supervision, Data Augmentation and Online fine-tuning for Offline RL

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Data Science

eingereicht von

Thomas Schmied, BSc.
Matrikelnummer 01553816

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing.(BA) Dr.rer.nat. Thomas Gärtner, MSc

Wien, 31. Jänner 2022

Thomas Schmied	Thomas Gärtner

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.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 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.

TU WIEN Informatics

Self-supervision, Data Augmentation and Online fine-tuning for Offline RL

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Data Science

by

Thomas Schmied, BSc.
Registration Number 01553816

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing.(BA) Dr.rer.nat. Thomas Gärtner, MSc

Vienna, 31st January, 2022

Thomas Schmied	Thomas Gärtner

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.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 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.

Erklärung zur Verfassung der Arbeit

Thomas Schmied, BSc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 31. Jänner 2022

Thomas Schmied

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 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.

Acknowledgements

I want to thank Prof. Gärtner for his valuable feedback throughout the writing of this thesis and for letting me explore my scientific interests.

Special thanks to Hannah and my family.

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 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.

Kurzfassung

Reinforcement learning (RL) Methoden lernen durch Interaktion mit einer Umwelt. Im RL Paradigma wird also online gelernt. Online zu lernen ist jedoch für viele Anwendungen in der echten Welt sehr unpraktikabel, da zumeist Ressourcen- oder Sicherheitseinschränkungen bestehen. Im Gegensatz zu online RL ermöglicht es offline RL, das primäre Thema dieser Arbeit, Entscheidungsstrategien aus vorher gesammelten Trainingsdaten zu erlernen. Allerdings haben RL Algorithmen einige weitere wesentliche Limitationen, darunter Datenineffizienz. Zwei vielversprechende Forschungsströme, die diese Limitation behandeln, sind selbst-überwachte Methoden und Datenaugmentierung. Diese Methoden wurden jedoch im online RL Kontext entwickelt und es ist daher noch nicht klar, ob sich deren Vorteile auch auf offline RL transferieren lassen. Nichtsdestotrotz ist es nicht immer ideal, Interaktion mit der Umwelt vollkommen zu eliminieren. Sowohl online RL als auch offline RL haben individuelle Vorteile und Nachteile. Algorithmen, die beide Ansätze vereinen, zum Beispiel durch offline pre-training und online fine-tuning, können die Vorteile beider Welten nutzen. Daher braucht es RL Agenten, die fähig sind sowohl online also auf offline in einer daten-effizienten Weise zu lernen.

In dieser Arbeit verbessern wir die Lernleistung von offline RL Algorithmen, indem wir existierende selbst-überwachte RL Methoden, Datenaugmentierung und online fine-tuning in den Lernprozess integrieren. Wir wählen drei selbst-überwachte online RL Architekturen (Curl, SPR, SGI) und fünf beliebte Datenaugmentierungen aus und adaptieren sie für den offline Kontext. Dann erweitern wir einen State-of-the-Art offline RL Algorithmus, Conservative Q-Learning (CQL), mit diesen Methoden und vergleichen sie mit fünf etablierten Baselines. Wir evaluieren alle Algorithmen sowohl auf diskreten als auch kontinuierlichen Kontrollaufgaben unter Verwendung von offline Atari und Gym-MuJoCo Datensätzen. Folglich wählen wir vier Atarispiele (Pong, Breakout, Seaquest, QBert) und drei Gym-MuJoCo Aufgaben (Halfcheetah, Hopper, Walker-2d) für unsere Experimente aus. Unsere Resultate zeigen, dass selbst-überwachte Methoden und Datenaugmentierung Baseline Agenten übertreffen können und zu beachtlichen Verbesserungen der Lernfähigkeit der offline RL Algorithmen auf Gym-MuJoCo führen können, aber nicht hilfreich auf Atari sind. Außerdem untersuchen wir, wie sich offline pre-training gefolgt von online fine-tuning auf die Lernfähigkeit des ausgewählten offline RL Algorithmus auswirkt. Unsere Resultate zeigen weiters, dass Algorithmen, die sowohl offline als auch online lernen, jenen Algorithmen, die nur online oder offline lernen, weitaus überlegen sein können.

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 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.

Abstract

Reinforcement learning (RL) methods learn through interaction with an environment. The RL paradigm is inherently designed to be performed in an online fashion. However, for many applications in the real world, learning online is not always feasible due to resource and/or safety constraints. Unlike online RL, offline RL, the main topic of this thesis, allows the agent to learn policies from previously collected datasets. Current RL algorithms have a number of other major limitations, among them data-inefficiency. Two promising streams of research that address this limitation are self-supervised methods and data augmentation. These methods were, however, developed for online RL, and it is not yet clear if their benefits translate to the offline case. Moreover, it is not always ideal to eliminate online environment interaction altogether. Both online RL and offline RL have their individual advantages and disadvantages. Algorithms that combine both approaches, e.g., via offline pre-training and online fine-tuning, can draw from the best of both worlds. Consequently, there is a need for RL agents that can learn both online and offline in a data-efficient way.

In this thesis, we improve the learning performance of offline RL algorithms by integrating existing self-supervised methods, data augmentations and online fine-tuning into the learning process. We select three established self-supervised online RL architectures (Curl, SPR, SGI) and five prominent data augmentations and adapt them for the offline setting. We then augment a state-of-the-art offline RL algorithm, Conservative Q-Learning (CQL), with the selected methods and compare them against five established baselines. We empirically evaluate all algorithms on both discrete and continuous control tasks using offline Atari and Gym-MuJoCo datasets, respectively. To this end, we select four Atari games (Pong, Breakout, Seaquest, QBert) and three Gym-MuJoCo tasks (Halfcheetah, Hopper, Walker-2d) for our experiments. Our results show that self-supervised methods and data augmentations can outperform the baseline agents and considerably improve the learning performance of offline RL algorithms on Gym-MuJoCo but are not beneficial on Atari. Furthermore, we investigate how offline pre-training followed by online fine-tuning affects the learning performance of the selected offline RL algorithm. Our results further demonstrate that hybrid algorithms that learn both offline and online can be far superior to learning online or offline alone.

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 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.

Contents

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 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.
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.
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.

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.

CHAPTER 1

Introduction

In this section, we first introduce the motivation and the problem statement for this thesis. Then we describe the aim of this work as well as the research questions we investigate. Finally, we present the contributions that we make in this thesis and outline the contents of the subsequent chapters.

1.1 Motivation & Problem statement

Reinforcement learning (RL) methods learn through interaction with an environment [Sutton and Barto, 2018]. The trial-and-error learning paradigm they adhere to is inherently designed to be performed in an online fashion. However, this can be problematic for a number of reasons. For many applications in the real world, such as autonomous driving, robotics, or healthcare, learning online is not always feasible due to resource and/or safety constraints. *Online RL* initially relies on random exploration of the environment at hand and can thus be unsafe or simply slow until it discovers high-reward behaviour. Unlike online RL, *offline RL*, the main topic of this thesis, allows the agent to learn policies from previously collected datasets (i.e., offline) that may contain transitions originating from a mixture of policies [Levine et al., 2020]. This game-changing modification of the learning paradigm has the potential to enable reinforcement learning based applications in the real world for a variety of decision-making problems where online interaction is not an option. The appeal of offline RL is further motivated by a simple observation: the success of Machine learning (ML) in applications such as Computer Vision, Natural language processing (NLP) or Speech Recognition, has been mainly driven by large and diverse datasets [Deng et al., 2009; Rajpurkar et al., 2016; Panayotov et al., 2015], not only by more advanced methods.

However, current reinforcement learning algorithms have a number of other major limitations, among them *data-inefficiency* [Dulac-Arnold et al., 2021]. In fact, some of the most sophisticated RL agents have to acquire hundreds of years' worth of real-time

experience, before they can perform their tasks [Vinyals et al., 2019; Berner et al., 2019]. Consequently, RL works particularly well for games or simulation-based environments, due to the fact that environment interaction under those circumstances is relatively inexpensive. Yet, in the real world, environment interaction may be costly and only a limited number of samples can be obtained. Data efficiency can be particularly problematic for offline RL methods, as the dataset is static, only a finite amount of observations is available, and no further observations can be collected. Therefore, methods have to be developed, that enable RL agents to leverage the available sets of previously collected observations in an effective way. Different solutions were proposed in the online RL context that aim to address data-inefficiency from different angles, among them are *self-supervised methods* and *data augmentation*. Both strategies can be used to learn more effective representations of the environment, allowing for more efficient learning behaviour. These methods were, however, developed for online RL and it is not clear yet if their benefits translate to the offline case. The appeal of self-supervised methods for offline RL is further motivated by another observation: many of the recent significant performance improvements in Computer Vision [Chen et al., 2020a; Grill et al., 2020; Chen et al., 2020b], NLP [Devlin et al., 2019; Brown et al., 2020; He et al., 2021] and Speech Recognition [Baevski et al., 2020] can be attributed to the ability to leverage large datasets via self-supervised learning techniques.

It is, however, not always ideal or even desirable to eliminate online environment interaction altogether. After all, the interaction of an agent with an environment is precisely what makes RL such an interesting research direction for the development of intelligence. In addition, online RL has led to some of the most spectacular achievements in the field of Artificial intelligence (AI) and can arguably lead to emergence of complex behaviour in complex environments [Silver et al., 2016; Berner et al., 2019; Vinyals et al., 2019]. However, both online RL and offline RL have their own individual advantages and disadvantages (e.g., fast/slow initial learning progress, high/low potential for long-term improvement) and thus, algorithms that unite both paradigms may be able to draw from the best of both worlds. In NLP it is, for instance, common practice to pre-train language models on large corpora of text and consequently fine-tune them for individual downstream tasks [Devlin et al., 2019]. Similarly, the "*offline pre-training, online fine-tuning*" paradigm is well suited for RL, especially for real world applications with safety and/or resource constraints. Current online and offline RL algorithms are, however, typically only specialized for online RL or offline RL alone, but not for both. In the long term, if we are to build increasingly intelligent RL agents they need to have the ability to flexibly leverage information from a variety of sources, both offline data from a mixture of policies and data obtained by online environment interaction alike. Therefore, there is a need for hybrid RL algorithms that can learn both *online and offline*.

1.2 Aim of the work & Research questions

The goal of this thesis is to improve the learning performance of offline RL algorithms by integrating existing self-supervised methods, data augmentations and online fine-tuning

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 Your knowledge hub WIEN

into the learning process.

In particular, we investigate the following research questions:

- What state-of-the-art self-supervised methods perform best in the offline RL context on continuous control (Gym-MuJoCo) and discrete control tasks (Atari) in terms of average return?
- What kinds of data augmentations perform best in the offline RL context on continuous control (Gym-MuJoCo) and discrete control tasks (Atari) in terms of average return?
- What are effective ways to combine offline pre-training with online fine-tuning to benefit from both RL paradigms?

Self-supervised methods have been applied successfully in the online RL setting, as demonstrated by systems such as Unreal [Jaderberg et al., 2017], Contrastive Unsupervised Representations for RL (Curl) [Laskin et al., 2020b] and Self-predictive Representations (SPR) [Schwarzer et al., 2021a]. Likewise, online RL agents equipped with data augmentation, such as RL with Augmented Data (RAD) [Laskin et al., 2020a] and Data-regularized Q (DrQ) [Yarats et al., 2021b] lead to more data efficient learning. Data augmentation is an established technique in Computer Vision and Speech recognition, but only recently has its effectiveness for online RL been demonstrated. However, while the effectiveness of self-supervised methods and data augmentation techniques has been explored in online RL, it has not yet been studied in the offline RL context. Due to the lack of prior research, it is not yet clear whether techniques that perform well online also work for offline RL. It might be possible that different kinds of techniques work better online than offline or vice versa, as a result of the differences in the paradigms. In this thesis, we aim to fill the gap. Furthermore, only limited research exists on offline pre-training and online fine-tuning [Nair et al., 2020; Lu et al., 2020]. Therefore, it remains to be understood what design decisions work best for translating behaviour learned during offline pre-training to online fine-tuning.

If offline RL agents are able to leverage the power of self-supervised learning, data augmentation and online fine-tuning, this will result in much more capable RL agents.

1.3 Contributions

First, we provide a comprehensive overview of online RL, offline RL, self-supervised learning in RL, data augmentation in RL and online fine-tuning for RL. Then, we conduct a thorough analysis of existing state-of-the-art offline RL algorithms, self-supervised learning techniques and data augmentations. In particular, we review the current literature on offline RL and identify suitable base agents that can be extended with self-supervised methods and data augmentations. In addition, we examine what self supervised tasks and what data augmentations are suitable in the offline RL context.

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 Your knowledge hub WIEN

To this end, we use Conservative Q-learning (CQL) [Kumar et al., 2020] as our base offline RL algorithm. We select three self-supervised architectures, namely Curl [Laskin et al., 2020b], SPR [Schwarzer et al., 2021a] and a modified version of SGI [Schwarzer et al., 2021b]. All self-supervised architectures we select have not been applied to offline RL prior to this work. Moreover, we select five augmentations that worked best in previous work with data augmentations in RL [Laskin et al., 2020a; Yarats et al., 2021b]: random-cropping, random-shifting and random cutout for discrete control tasks, and random amplitude scaling and Gaussian noise for continuous control tasks. As these architectures were developed for online RL, we extend them for offline RL.

We also investigate how offline pre-training followed by online fine-tuning affects the learning performance of CQL. CQL is a state-of-the-art offline RL algorithm, and was thus specifically developed for the offline context. However, we show that with the right design decisions, CQL is well suited for the "offline pre-training, online fine-tuning" paradigm and can be far superior to online learning and offline learning alone.

We evaluate all selected algorithms on both discrete and continuous control tasks using offline Atari and Gym-MuJoCo datasets provided by the offline RL benchmark suites RL Unplugged [Gülçehre et al., 2020] and Datasets for Reinforcement Learning (D4RL) [Fu et al., 2020], respectively. We select four Atari games (Pong, Breakout, Seaquest, QBert) and three Gym-MuJoCo tasks (Halfcheetah, Hopper, Walker2d) for our experiments. While Atari provides image-based inputs, Gym-MuJoCo emits state-based inputs. All the self-supervised architectures we evaluate in this work were specifically developed for image-based tasks. Therefore, we adjust them to state-based tasks accordingly.

On each Atari and Gym-MuJoCo task we compare our implementations against five established baselines: a random policy, Behavioural Cloning (BC) [Pomerleau, 1988], CQL without any extensions, online/offline Deep Q-Networks (DQN) [Mnih et al., 2015] on Atari, and online/offline Soft Actor Critic (SAC) [Haarnoja et al., 2018] on Gym-MuJoCo.

All agents are evaluated based on average return, normalized scores (with respect to random and expert agents), interval estimates of the normalized performance for the mean, median and interquartile mean (IQM), and probability of improvement. Our evaluation methodology is described in more detail in Section 3.

In summary, we make the following **contributions** in this thesis:

- We provide a comprehensive overview of the state-of-the-art in the research field of offline RL.
- We evaluate the effectiveness of existing self-supervised methods and data augmentation techniques for offline RL and show that they can lead to considerable improvements on Gym-MuJoCo but are not beneficial on Atari.
- We demonstrate that the "offline pre-training, online-fine tuning" RL paradigm can be far superior to online or offline RL individually and advocate for hybrid RL algorithms that are designed to learn both offline and online.

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 Your knowledge hub WIEN

1.4 Structure of this thesis

The rest of this thesis is structured as follows:

- In Chapter 2 we discuss the theoretical background of this thesis. In particular, this includes a brief introduction to the online RL framework and a discussion of offline RL, the main topic of this thesis. Furthermore, the current challenges of both frameworks, as well as the approaches to address these challenges, namely self-supervised learning, and data augmentation, are discussed.
- In Chapter 3 we describe our methodological approach for this thesis. This includes our selection of offline RL algorithms, self-supervised methods, data augmentations, offline datasets & benchmark tasks, as well as our rationale for the selection. In addition, we examine our evaluation methodology as well as our software/hardware setup.
- In Chapter 4 we discuss what experiments we conduct and how we conduct them. This includes a detailed description of the hyperparameters and adjustments we make to the learning algorithms.
- In Chapter 5 we present the results we obtained. This includes a detailed empirical analysis and a discussion of the insights we gained from our experiments.
- Finally, in Chapter 6 we conclude this thesis.

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 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.

CHAPTER 2

Background

In this section, we discuss the theoretical background of the relevant topics for this thesis. This section will serve as a theoretical basis for the practical part of this thesis. First, the general reinforcement learning framework and its inherent challenges are introduced. Then offline RL, the main topic of this thesis, and its advantages as well as disadvantages that come with it, are reviewed. Finally, we discuss two approaches to address the limitations of current offline RL methods: self-supervised learning and data augmentation. Each approach is examined individually, first the general background is explained, then we discuss how self-supervised learning and data augmentation are applied in the RL context.

2.1 Online RL: learning by interaction

2.1.1 Overview

The reinforcement learning paradigm is based on learning by interaction, or learning by trial-and-error, with an environment. Conceptually, RL is concerned with decision-making, learning what to do in a particular situation to achieve a good outcome (i.e., to receive the maximal reward) [Sutton and Barto, 2018, p. 2].
Figure 2.1 illustrates the RL framework and the key concepts graphically: an *agent* (e.g., a robot), interacts with an *environment*. The robot finds itself in a particular *state* s_t within the environment and selects an *action* a_t. Then it transitions to a new state s_{t+1} and receives a *reward* r_t. Therefore, each interaction step can be summarized as an experience tuple (s_t, a_t, s_{t+1}, r_t) (that can optionally be stored to an *experience replay buffer*, $\mathcal{D}$), consisting of current state, current action, next state and reward obtained. The reward can be positive or negative, high, or low depending on how beneficial it was for the agent to perform that particular action a_t in the particular state s_t. The action is selected according to the learned *policy* π that maps states to probabilities of selecting

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 Your knowledge hub WIEN

each possible action, $a_t \sim \pi(a_t \mid s_t)$. The goal the agent pursues is to maximize its total reward obtained. To achieve this, it has to learn how to select good actions in the states it finds itself in, i.e., it has to adjust its policy π. This interaction loop simply continues until the agent learns the desired behaviour and is able to solve the target task or runs out of compute budget. Therefore, we can say that the policy is learned in an *online* fashion, giving rise to the term online RL. In practice, learning to solve a particular task may require a large number of trials, one of the major limitations of online RL algorithms [Dulac-Arnold et al., 2021]. This will be discussed in more detail in Section 2.1.3.

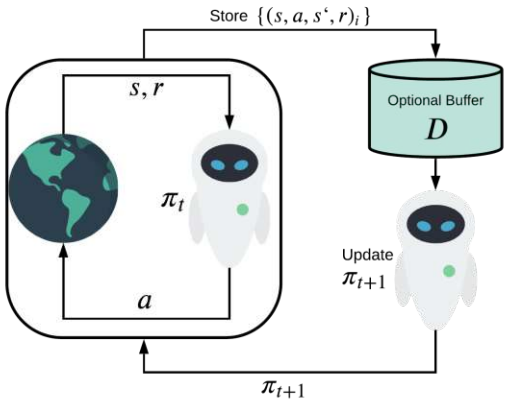


Figure 2.1: The online RL framework. Inspired by Levine et al. [2020].

This simple yet powerful framework has repeatedly proven to deliver remarkable results, in particular for simulation-based environments (e.g., games). Some of the most prominent achievements with RL at the centre are beating the human world champions in board games (such as go, chess & shogi) [Silver et al., 2016, 2017, 2018; Schrittwieser et al., 2020] and real-time combat arena video games like Dota and StarCraft II [Berner et al., 2019; Vinyals et al., 2019]. However, applications of RL are not only limited to games. RL has already been applied successfully to a variety of other applications, from optimizing data centre energy consumption, to safely navigating balloons high up in the stratosphere, to designing more efficient chip layouts as well as neural architectures [Lazic et al., 2018; Bellemare et al., 2020; Mirhoseini et al., 2020; Zoph and Le, 2017; Pham et al., 2018].

Reinforcement learning is a subfield of machine learning. In ML, one historically differentiates between *supervised* and *unsupervised* learning. The former is based on learning by examples that are annotated by a knowledgeable (human) supervisor (e.g., image-label pairs) [Sutton and Barto, 2018, p. 2]. Therefore, this paradigm provides a clear indication of "correctness" of the predictions made by the learning algorithm via labels. The latter paradigm, on the other hand, involves no supervision (i.e., no labels) throughout the learning process. Rather, the learning occurs by means of observation or discovery of hidden structures within the unlabelled data points [Sutton and Barto, 2018, p. 2]. The RL paradigm, however, is different from both supervised and unsupervised learning.

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 Your knowledge hub WIEN

While in unsupervised learning, supervision is not involved at all, RL aims to maximize the feedback reward signal obtained. While in supervised learning the labels give a clear indication of right or wrong, the reward signal in RL only indicates whether a behaviour in a particular situation is good or bad. [Sutton and Barto, 2018, p. 2]

2.1.2 Formal background

Mathematically, RL is formalized as a Markov decision process (MDP), the standard framework for sequential decision-making problems [Sutton and Barto, 2018, p. 47]. The MDP is defined as the tuple $M = (\mathcal{S}, \mathcal{A}, T, r, \gamma)$ with:

- the state space $\mathcal{S}$, the set of all possible states $s \in \mathcal{S}$
- the action space $\mathcal{A}$, the set of all possible actions $a \in \mathcal{A}$
- the conditional transition probability function $T(s_{t+1} \mid s_t, a_t)$
- the reward function $r(s_t, a_t)$ for state-action-pairs s_t and a_t
- the discount factor γ

Both $\mathcal{S}$ and $\mathcal{A}$ can be discrete or continuous. Often, the initial state distribution $d_0(s_0)$ is also included in the MDP definition, $M = (\mathcal{S}, \mathcal{A}, T, r, \gamma, d_0)$. In some cases, the state space is only partially observable, giving rise to the notion of the Partially Observable Markov decision process (POMDP) [Sutton and Barto, 2018, p. 466]. Chess and Poker are examples for an MDP and a POMDP, respectively. In general, the RL agent pursues the goal of maximizing its total reward obtained. The total reward obtained, or sum of all rewards, over a finite or infinite *horizon* H, is referred to as the return G_t:

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... + \gamma^H r_{t+H} = \sum_{k=t}^{H} \gamma^{k-t} r(s_k, a_k) = r_t + \gamma G_{t+1} \tag{2.1}$$

In fact, Equation 2.1 shows the *discounted return*. The discount factor γ ensures that immediate rewards are perceived as better than rewards received in the future [Sutton and Barto, 2018, p. 55]. The rewards are obtained from the reward function, i.e., $r_t = r(s_t, a_t)$. A sequence of states and actions is called a *trajectory*, $\tau = (s_0, a_0, ..., s_H, a_H)$ and the *trajectory distribution* is:

$$p_\pi(\tau) = d_0(s_0) \prod_{t=0}^{H} \pi(a_t \mid s_t) T(s_{t+1} \mid s_t, a_t) \tag{2.2}$$

The overall RL objective is thus given by (with $t = 0$):

$$J(\pi) = \mathbb{E}_{\tau \sim p_\pi(\tau)} \left[\sum_{k=t}^{H} \gamma^{k-t} r(s_k, a_k) \right] \tag{2.3}$$

To achieve its goal, the agent has to learn a good policy π. As mentioned in Section 2.1.1, the policy π maps states to probabilities of selecting each possible action, $a_t \sim \pi(a_t \mid s_t)$ [Sutton and Barto, 2018, p. 58]. It can either be learned directly via policy gradients or indirectly via value functions. This is one of the most important distinctions in RL. Algorithms that directly learn the policy are known as *policy optimization* methods.

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 Your knowledge hub WIEN

Algorithms that learn the desired behaviour based on value functions are referred to as *value-based* methods (with Q-learning [Watkins and Dayan, 1992] being the most prominent one). The combination of both approaches is known as *actor-critic* methods.

Value-based methods. The *value* of a state or state-action pair determines how good that state or performing a specific action in that state is in the long term [Sutton and Barto, 2018, p. 58]. In contrast, the reward indicates what is beneficial in an immediate sense. The concept of value functions is formalized by the Bellman equations for state-values and state-action-values, $V_\pi(s_t)$ and $Q_\pi(s_t, a_t)$, respectively. The *state-value function* determines the value of a state s_t, when starting in s_t and following π:

$$\begin{aligned} V_\pi(s_t) &= \mathbb{E}_\pi[G_t \mid s_t] \\ &= \mathbb{E}_\pi[r_t + \gamma G_{t+1} \mid s_t] \\ &= \sum_{a \in \mathcal{A}} \pi(a \mid s_t) \sum_{s' \in \mathcal{S}} T(s' \mid s_t, a)[r(s_t, a) + \gamma \mathbb{E}_\pi[G_{t+1} \mid s']] \\ &= \sum_{a \in \mathcal{A}} \pi(a \mid s_t) \sum_{s' \in \mathcal{S}} T(s' \mid s_t, a)[r(s_t, a) + \gamma V_\pi(s')] \\ &= \mathbb{E}_{a_t \sim \pi(a_t|s_t), s_{t+1} \sim T(s_{t+1}|s_t, a_t)}[r_t + \gamma V_\pi(s_{t+1})] \end{aligned} \tag{2.4}$$

Likewise, the *state-action-value function* determines how good it is to perform a specific action a_t in a specific state s_t if starting in s_t and following π:

$$\begin{aligned} Q_\pi(s_t, a_t) &= \mathbb{E}_\pi[G_t \mid s_t, a_t] \\ &= \mathbb{E}_\pi[r_t + \gamma G_{t+1} \mid s_t, a_t] \\ &= \sum_{s' \in \mathcal{S}} T(s' \mid s_t, a_t)[r(s_t, a_t) + \gamma V_\pi(s')] \\ &= \mathbb{E}_{s_{t+1} \sim T(s_{t+1}|s_t, a_t)}[r_t + \gamma V_\pi(s_{t+1})] \end{aligned} \tag{2.5}$$

Consequently, the state-value function from Equation 2.4 can be expressed in terms of the state-action value function as:

$$V_\pi(s_t) = \mathbb{E}_{a_t \sim \pi(a_t|s_t)}[Q_\pi(s_t, a_t)] \tag{2.6}$$

Similarly, the state-action-value function from Equation 2.5 can be expressed as:

$$Q_\pi(s_t, a_t) = \mathbb{E}_{s_{t+1} \sim T(s_{t+1}|s_t, a_t), a_{t+1} \sim \pi(a_{t+1}|s_{t+1})}[r_t + \gamma Q_\pi(s_{t+1}, a_{t+1})] \tag{2.7}$$

The state and state-action-value function are learned through iterative updates of a value table or function approximator. The most prominent method is known as *Q-learning* [Watkins and Dayan, 1992] and employs the following update rule:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{2.8}$$

Where the learning rate α is a hyperparameter that regulates the magnitude of the update step. The term in brackets in Equation 2.8 is known as the *Bellman error*, $\mathcal{E} = r_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)$ [Sutton and Barto, 2018, p. 268]. We drop the reference to π as the current Q-estimates are recorded in the value table. The value 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 state-action value function determines how good it is to perform a certain action in a particular state. Therefore, it can be used to select the best action to perform, e.g., by taking the action with the highest Q-value, $\max_a Q(s_t, a)$. Consequently, learning the Q-function is an indirect way of learning the policy. Well-known and established examples of value-based methods are DQN (+ its variants) [Mnih et al., 2015; Van Hasselt et al., 2016; Wang et al., 2016; Hessel et al., 2018], Hindsight Experience Replay (HER) [Andrychowicz et al., 2017] and R2D2 [Kapturowski et al., 2018].

Policy optimization methods. The policy can also be learned directly via the *policy gradient* without employing a value function [Sutton and Barto, 2018, p. 321]. In this scenario, we assume a parametrized policy $\pi_\theta(a_t \mid s_t)$ with θ, the parameters of the selected function approximator (e.g., a neural network). Policy gradient methods then perform Stochastic Gradient Descent (SGD) on a performance measure $J(\theta)$ to update the parameters:

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta) \tag{2.9}$$

According to the policy gradient theorem [Sutton et al., 1999], the policy gradient $\nabla J(\theta)$ is given by:

$$\begin{aligned} \nabla J(\theta) &= \mathbb{E}_\pi \left[\frac{\nabla \pi_\theta(a_t \mid s_t)}{\pi_\theta(a_t \mid s_t)} G_t \right] \\ &= \mathbb{E}_\pi [\nabla \log \pi_\theta(a_t \mid s_t) G_t] \end{aligned} \tag{2.10}$$

Equation 2.10 shows the policy gradient used in the classic REINFORCE algorithm [Williams, 1992]. Prominent examples of policy gradient methods are Trust Region Policy Optimization (TRPO) [Schulman et al., 2015] and Proximal Policy Optimization (PPO) [Schulman et al., 2017].

Actor-critic methods. Another fundamental category of algorithms is known as *actor-critic* methods, the combination of policy optimization and value-based methods [Sutton and Barto, 2018, p. 331]. In Equation 2.10 G_t represents the return. To improve the algorithm, by decreasing the variance of the policy gradient, we can additionally subtract a baseline term $b(s_t)$ [Sutton and Barto, 2018, p. 330]:

$$\nabla J(\theta) = \mathbb{E}_\pi [\nabla \log \pi_\theta(a_t \mid s_t)(G_t - b(s_t))] \tag{2.11}$$

The baseline can be an estimation of the state-value function, $\hat{V}(s_t)$ (e.g., a neural network) and learned via SGD. Actor-critic methods learn both the policy and a value function. The parametrized policy π_θ represents the *actor*, while the additional learned value function $\hat{V}$ acts as the *critic*, hence the name. Prominent examples of actor-critic methods are A3C/A2C [Mnih et al., 2016; Wang et al., 2017], Deep Deterministic Policy Gradient (DDPG) [Lillicrap et al., 2016], SAC [Haarnoja et al., 2018] and Importance Weighted Actor-Learner Architecture (IMPALA) [Espeholt et al., 2018].

On-policy vs. off-policy. A major distinction between different RL algorithms is whether they learn *on-policy* or *off-policy*. In principle, RL algorithms aim to learn the optimal behaviour in a given environment. To learn the optimal behaviour, however, they have to explore the state/action space. But to explore the state/action space, they

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 Your knowledge hub WIEN

have to behave non-optimally. This dilemma is known as *exploration* versus *exploitation* [Sutton and Barto, 2018, p. 103]. Now, on-policy methods, on the one hand, learn from experience generated using the same policy that is being optimized. Therefore, in on-policy algorithms, the policy is near-optimal to allow for exploration. Off-policy methods, on the other hand, use two policies: a *target policy* (the one being learned) and a *behaviour policy* (the one to generate experience) [Sutton and Barto, 2018, p. 103]. The separation of experience generation and learning in off-policy methods has an important implication: experience can be reused. Therefore, Figure 2.1 depicts an (optional) experience replay buffer/memory $\mathcal{D}$ that stores experience tuples (s_t, a_t, s_{t+1}, r_t). The replay buffer was one of the major innovations of DQN [Mnih et al., 2015]. However, there are limitations to how different the data-generating policy and the target policy can be. This will be further discussed in Section 2.2. In practice, the behaviour policy is often a copy of the target policy and periodically synced. In this work, we denote it as π_β. REINFORCE and Q-learning are classic examples for on-policy and off-policy methods, respectively.

Model-free vs. model-based. Another fundamental distinction of RL algorithms is whether they learn a model of the environment or not [Sutton and Barto, 2018, p. 159]. Like the policy or value functions, the dynamics $T(s_{t+1} \mid s_t, a_t)$ and/or the reward function $r(s_t, a_t)$ of the environment can be learned. The learned representations of environment dynamics and reward function are referred to as the model. Algorithms that learn such a model are *model-based*, while the ones that do not learn a model are *model-free*. Value-based methods, policy optimization and actor-critic methods are in general model-free. Learning a model can be useful, as it allows the agent to simulate experience and learn the value function or policy without having to interact with the actual environment. This is important as environment interaction can be costly in terms of time or resources. Prominent examples of model-based methods are Dyna [Sutton, 1991], Monte Carlo Tree Search [Coulom, 2006], AlphaGo (+ its variants) [Silver et al., 2016, 2017, 2018; Schrittwieser et al., 2020] and Dreamer [Hafner et al., 2019, 2021]

Deep reinforcement learning. In the classic RL setup the policy, value functions and the model are represented by tables, but due to memory constraints this becomes infeasible as the state and/or action space grows. Therefore, they have to be approximated. Most modern RL algorithms use deep neural networks as their function approximators, giving rise to the term Deep reinforcement learning (DRL) [Sutton and Barto, 2018, p. 475]. For example, in the parametrized policy π_θ, θ refers to the neural network weights learned by SGD. Similarly, the value functions and model components can be parametrized by neural networks and learned via gradient descent, e.g., $V_\theta(s_t)$, $Q_\theta(s_t, a_t)$, $T_\theta(s_t \mid s_t, a_t)$ and $r_\theta(s_t, a_t)$. For instance, the original formulation of DQN modifies the classic Q-learning update rule in Equation 2.8 to obtain the following optimization objective:

$$J(\theta) = \mathbb{E}_{(s,a,s',r)\sim\mathcal{D}}[\underbrace{(r + \gamma \max_{a'} Q_{\theta^-}(s', a') - Q_\theta(s, a))^2}_{\text{Bellman error } \mathcal{E}(\theta)}] \tag{2.12}$$

where states, next states, actions, and rewards are sampled from the replay buffer $\mathcal{D}$. s, a, s' and a' refer to state, action, next state and next action, respectively (using

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 Your knowledge hub WIEN

primes instead of subscripts often simplifies the notation). Furthermore, θ^- denotes the parameters of the target Q-network, a copy of the parameters of the current Q-network and periodically synced [Mnih et al., 2015]. The parameters θ are optimized via SGD.

Online RL taxonomy. To summarize, Figure 2.2 gives a graphical overview of the different categories of online RL algorithms.

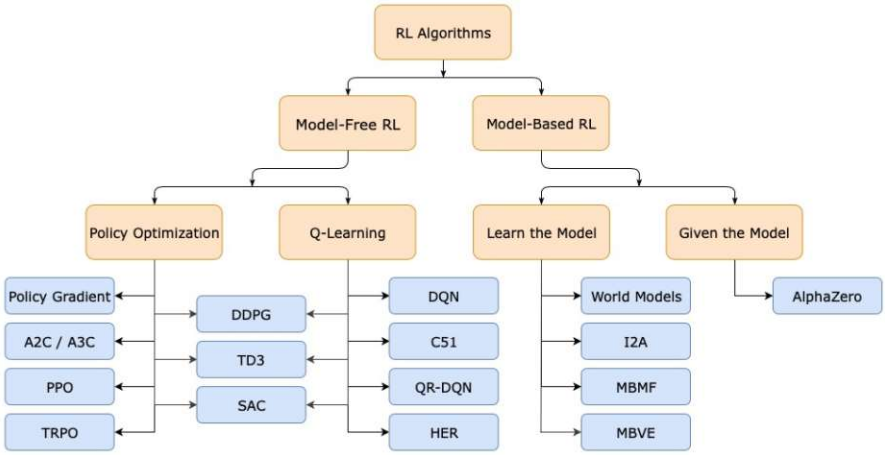


Figure 2.2: The online RL taxonomy [Achiam, 2018].

2.1.3 Challenges & limitations of online RL

Although the RL framework has enjoyed great success, it has a variety of notable limitations that constrain its applicability in real world applications. Dulac-Arnold et al. [2021] identified a set of nine fundamental challenges that need to be overcome to enable RL applications in the real world, among them are **(1)** learning offline from fixed, previously collected experience originating from an external behaviour policy and **(2)** learning from a limited amount of samples (i.e., in a data-efficient way) [Dulac-Arnold et al., 2021; Dulac-Arnold et al., 2020]. We briefly addressed both challenges in Section 1.1 to motivate the topic of this work. In this section, we continue the discussion.

Online learning is arguably one of the core characteristics that make the RL framework so appealing to many RL practitioners. After all, the agent-in-an-environment setup resembles the human perspective, how we all experience our everyday lives. By many, intelligence is considered as "an agent's ability to achieve goals in a wide range of environments" [Legg and Hutter, 2007]. This definition is well suited for RL. Silver et al. [2021] even argue that every conceivable task can be formulated as an RL objective, thus postulating RL as a viable path towards artificial general intelligence (AGI). So, why would learning online be a problem for machines but not for humans? Typically, RL agents have to solve a very specific task. Examples of real-world RL tasks are to steer a car/truck/drone, to recommend products to a user, or to control a robotic arm in a warehouse (many other examples are provided in Dulac-Arnold et al. [2020]). For humans,

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 Your knowledge hub WIEN

each task is embedded in a lifetime of experience (a lifetime of tasks) and as a consequence a rich, implicit understanding of the environment (as studied by Dubey et al. [2018] in the context of Atari). Thus, humans can quickly adapt to solve a variety of difficult and diverse problems. In contrast, RL agents essentially learn to solve each individual tasks from scratch by trial-and-error, without an analogous, implicit understanding of how the world works. Therefore, they initially act randomly and might require a long time to accomplish their task (or run out of compute budget), as we show in Chapter 5. However, many applications in the real-world, including the ones listed above, are subject to resource, safety, time, and/or other constraints. Online learning may even be dangerous. For example, an RL agent steering an autonomous vehicle, cannot just learn to avoid car accidents by actually causing them. Similarly, the monetary loss caused by an RL agent that recommends random products would not be tolerated for long or at all. An RL agent controlling a robotic arm operating in a warehouse must not hurt its human co-workers, and cannot learn this restriction by actually hurting them. Furthermore, it might simply be overly cost or time-intensive to operate the RL agent, making online learning in the real world less effective. This suggests that in the long term, online RL alone might not in every case be the optimal approach to enable decision-making agents in the real world.

One interesting stream of research to address this problem, known as Sim-to-Real, is to learn the decision-policy in simulation and then transfer it to the real-world [Tan et al., 2018; Peng et al., 2018; Chebotar et al., 2019]. The advantage of simulations is that they allow for errors and cheap environment interaction. However, it is typically costly to build a simulator. Also, if the task changes and the underlying simulation is not sufficiently general to account for the differences, a new simulation might have to be built. Furthermore, even though modern simulations are very realistic, a reality gap might still be present in the simulator, which the RL agent could happily exploit [Jakobi et al., 1995; Peng et al., 2018]. Hence, the agent's ability to act in the real world is effectively restricted by the quality of the simulation.

A more scalable alternative to deal with this limitation is to decouple environment interaction and learning procedure to leverage large, diverse, and previously collected datasets [Levine et al., 2020]. This is the fundamental idea of offline RL, the main topic of this thesis. Offline RL and its advantages/disadvantages will be discussed in Section 2.2. Even though online RL has its weaknesses, it is a very powerful framework with many attractive characteristics. Therefore, a combination of offline pre-training and online fine-tuning may draw from the best of both worlds, as we show in Chapter 5.

Another major limitation of RL algorithms is their data-inefficiency. Even though RL has achieved impressive results on board games, real-time video games, and dexterous robotic manipulation of objects, the underlying algorithms sometimes require hundreds (and even thousands) of years' worth of real-time experience [Vinyals et al., 2019; Berner et al., 2019; Akkaya et al., 2019]. This is intractable in the real-world due to the aforementioned resource and time constraints. In fact, for most real world-applications it is only feasible to obtain a tiny amount of experience through online interaction, even though it is often

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 Your knowledge hub WIEN

possible to collect large amounts of samples over time, e.g., from human demonstrations or system logs. Therefore, it remains to be understood how to learn more efficiently from fewer samples and environment interaction steps. Humans (and arguably other animals) are, after all, able to learn complex skills from only a relatively small number of samples [Lu et al., 2020]. A potential reason for this is that they acquire a rich understanding of how the world they live in works over their lifetimes. Therefore, one approach towards more data-efficient RL algorithms is to learn richer and more meaningful representations of the environment, i.e., *world knowledge*. Two promising approaches to achieve this are self-supervised learning and data augmentation [Jaderberg et al., 2017; Laskin et al., 2020b; Schwarzer et al., 2021a; Yarats et al., 2021b; Laskin et al., 2020a]. We discuss the general background and existing architectures in the RL context of self-supervised learning and data augmentation in Sections 2.3 and 2.4, respectively.

2.2 Offline RL: learning from data

2.2.1 Overview

Offline RL, or batch RL as it is also called in the literature, has been studied for many years. Yet, only recently, offline RL has started to receive wide-spread attention within the reinforcement learning community. A detailed discussion of the field is provided by Levine et al. [2020]. The fundamental difference to online RL is that in offline RL environment interaction and learning procedure are decoupled. The policy is learned entirely from previously collected experience, without any interaction with the environment [Levine et al., 2020]. Similar to Figure 2.1 in Section 2.1, Figure 2.3 illustrates the offline RL framework and the difference to online RL graphically. As before, the target policy, behaviour policy, state, action, next state, reward, and experience buffer (or dataset) are represented by $\pi, \pi_\beta, s_t, a_t, s_{t+1}, r_t$ and $\mathcal{D}$, respectively. Essentially, offline RL can be separated into three individual phases: **(1)** generating the offline dataset, **(2)** learning the policy from the offline dataset and **(3)** deploying the trained agent to the environment.

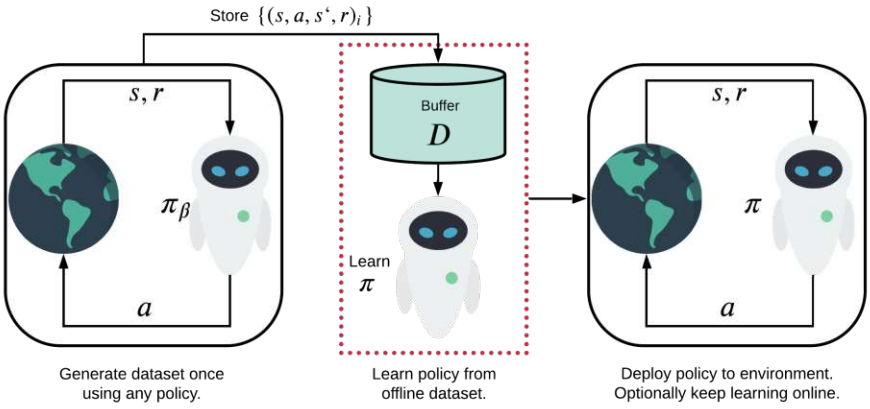


Figure 2.3: The offline RL framework. Inspired by Levine et al. [2020].

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 Your knowledge hub WIEN

The offline dataset can be generated using any kind of *behaviour generating policy* π_β, or even a mixture of multiple different policies, that is/are typically unknown. For example, the observed experience tuples could originate from expert demonstrations (e.g., by humans), other (suboptimal) online RL agents, or simply random interactions with the environment. Notably, offline RL has similarities with off-policy learning. In fact, in off-policy RL, the behaviour policy π_β can also be any policy *other* than the current target policy [Levine et al., 2020]. However, off-policy learning usually also involves online interaction with the environment, while in offline RL the policy is learned entirely offline. In principle, any off-policy algorithm could be used to learn a policy from the static dataset, but this approach has its limitations. Also, the dataset has to be obtained somehow in the first place. We discuss these shortcomings in Section 2.2.5. Once trained, however, the agent can be deployed to the environment. Then the agent already has a certain understanding of what it is supposed to do and does not act entirely at random, as would be the case with online RL. Hence, it does not violate the constraints we discussed in Section 2.1.3. The RL agent can even continue learning online to improve upon the behaviour it acquired offline. This can be a very desirable approach, as we show in Chapter 5.

Offline RL improves over the limitations of online RL we discussed in Section 2.1.3, and thus is attractive for a variety of applications in the real world. The framework has already been applied to learning robotic manipulation policies in a safe and efficient way [Kalashnikov et al., 2018, 2021], goal-oriented dialogue [Jaques et al., 2020], and recommending treatment policies in healthcare [Nie et al., 2020]. A more detailed overview of existing offline RL applications and offline RL in general is given by Levine et al. [2020]. The possibilities for further real-world use-cases of offline RL are enormous, and due to the recency of the field only a few areas have been explored yet.

In addition, the offline RL framework offers a number of practical advantages over online RL from a software engineering point of view, as the static dataset is now simply readily available. Setting up an efficient environment interaction loop can be a major challenge in online RL. In practice, RL algorithms strongly benefit from distributed model training over many machines, as a large number of environments can be simulated this way [Espeholt et al., 2018; Liang et al., 2018; Hoffman et al., 2020]. However, such a demanding setup is largely out of reach for most researchers in academia and industry. In contrast, in offline RL, the dataset is readily available. Therefore, best practices from supervised ML training can be applied more easily. As a result, offline RL poses lower entry barriers in terms of compute or setup complexity.

Overall, offline RL has the potential to transform machines into powerful decision-making agents in the real world [Levine et al., 2020]. Ultimately, the current success of ML in a variety of practical domains can be largely attributed to their ability to leverage large and diverse datasets. In comparison, RL has mainly been successful in simulation-based environments. Of course, beating the best players in the world's hardest strategy games is an amazing achievement. Nevertheless, accomplishments of similar scale for practical problems in the real world have not materialized until now. Therefore, a general

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 Your knowledge hub WIEN

data-driven approach for RL might enjoy similar success in the near future as ML has.

2.2.2 Formal background

In offline RL, we again assume the standard MDP formulation with $M = (\mathcal{S}, \mathcal{A}, T, r, \gamma)$, but now no environment interaction is involved. Instead, the agent is provided with the dataset (replay buffer) $\mathcal{D}$ consisting of experience tuples, $\mathcal{D} = \{(s, a, s', r)_i\}$ (with primes instead of subscripts). The actions were selected according to one or multiple behaviour policies used during experience generation, collectively denoted by π_β. The RL objective does not change: maximize the reward obtained in the long run by learning policy π. However, now the observations are not emitted by the environment, but directly sampled from the static dataset. Notably, this setup is very similar to the conventional supervised learning setup, where $\mathcal{D}$ would be referred to as the training set [Levine et al., 2020]. But in supervised ML, the dataset $\mathcal{D}$ would consist of a set of features $X = \{x_1, x_2, ..., x_k\}$ and the corresponding labels y: $\mathcal{D} = \{(X, y)_i\}$ (e.g., image-label pairs).

So, why is it problematic to use online RL algorithms with offline datasets? The fundamental challenge is known as *distributional shift*: the policy/value function is trained under one distribution, but is evaluated under a different distribution. Distributional shift can occur at test and training time. At test time, *state distribution shift* can occur, as the dataset might have different state visitation frequencies than are observed during environment interaction. In particular, in out-of-distribution states, erroneous actions might be produced. At training time, *action distribution shift* can occur, as the target values in Equation 2.7 require evaluating the next action a_{t+1}. The selected action might lie out-of-distribution of actions that the Q-function was trained on, and thus the estimated Q-values might be erroneous. This is particularly problematic, as π is optimized to maximize the Q-values [Levine et al., 2020].
To address this problem, three broad strategies have been proposed: *policy constraints*, *uncertainty estimation* and *conservative value functions* [Levine et al., 2020]. Policy constraint methods, such as BCQ [Fujimoto et al., 2019] and AWAC [Nair et al., 2020] constrain the deviation of the target policy π_θ from the behaviour policy π_β represented in the dataset. Uncertainty based methods, on the other hand, directly estimate their uncertainty about state-action pairs and utilize their estimation to mitigate distributional shift. Conservative methods regularize the value function directly to avoid overestimation of out-of-distribution actions. Furthermore, model-based RL, as discussed in Section 2.1.2, can be adjusted for the offline scenario and used to address distributional shift [Levine et al., 2020]. In this section, we will briefly introduce the different types of offline RL methods and cover the algorithms used in our experiments. For a more in-depth analysis of the individual categories, we refer to Levine et al. [2020].

Policy constraints. Policy constraint methods limit how much the target policy can deviate from the behaviour policy. The constraints can be formulated as: **(1)** *direct policy constraints* or **(2)** via a *policy penalty* [Levine et al., 2020].

Policy penalty methods incorporate the constraint into actor-critic methods by subtracting

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 Your knowledge hub WIEN

a penalty term $D(\pi(a \mid s) \mid\mid \pi_\beta(a \mid s))$ from the obtained reward or subtracting it directly from the target Q-values as well as actor objective. For example, the resulting reward function is then given by $\bar{r}(s,a) = r(s,a) - \alpha D(\pi(a \mid s) \mid\mid \pi_\beta(a \mid s))$. Where α is a hyperparameter and the Kullback-Leibler divergence (KL-divergence) is typically used for D, with $D_{KL}(\pi, \pi_\beta) = \mathbb{E}_\pi[\log \pi(a \mid s) - \log \pi_\beta(a \mid s)]$[Levine et al., 2020]. However, π_β is typically unknown and has to be approximated from samples in the dataset. Examples of policy penalty algorithms are behaviour regularized actor critic (BRAC) [Wu et al., 2019] and way off-policy learning (WOP) [Jaques et al., 2020].

Direct policy constraints, on the other hand, can be enforced either as **(a)** explicit f-divergence constraints, **(b)** implicit f-divergence constraints, or as **(c)** integral probability metric (IPM) constraints [Levine et al., 2020]. The KL-divergence is an f-divergence and can be enforced explicitly or implicitly. Examples for implicit f-divergence constraints are AWR [Peng et al., 2019] and AWAC [Nair et al., 2020]. IPM constraints instead can be enforced by the maximum mean discrepancy (MMD) or the Wasserstein distance. One example for this class of methods is BEAR [Kumar et al., 2019].

Uncertainty estimation. In contrast, uncertainty estimation methods estimate their uncertainty of a state-action pair. The uncertainty estimation indicates when an action is out-of-distribution and can be considered during updating the value function or policy, i.e., by subtracting the uncertainty [Levine et al., 2020]. These methods require learning an uncertainty set/distribution over possible action-value functions, $\mathcal{P}_\mathcal{D}(Q_\pi)$ and a function to determine the uncertainty, $\text{Unc}(\mathcal{P}_\mathcal{D}(Q_\pi))$. To represent $\mathcal{P}_\mathcal{D}(Q_\pi)$ confidence bounds, bootstrap ensembles, or parametric distributions such as a Gaussian can be used. Depending on the representation, the uncertainty function Unc can be expressed differently, e.g., by taking the variance across the ensemble predictions [Levine et al., 2020]. An example of an uncertainty estimation method is REM [Agarwal et al., 2020].

Conservative value functions. Conservative methods regularize the value function directly to avoid overestimation of out-of-distribution actions [Levine et al., 2020]. A prominent example of this category is CQL [Kumar et al., 2020]. We describe CQL in more detail in Section 2.2.3, as we use the algorithm in our subsequent experiments.

Model-based learning. Finally, model-based methods, previously described in Section 2.1.2, can be an effective technique for offline RL. However, the learned model representation $T_\theta(s' \mid s, a)$ also suffers from distributional shift [Levine et al., 2020]. To address the distributional shift recent methods, MoREL [Kidambi et al., 2020] and MOPO [Yu et al., 2020], employ uncertainty estimation to introduce pessimism into the learned model. The idea is to train an ensemble of models to estimate their collective uncertainty about transitions by taking the disagreement of the individual models in the ensemble into account. When the ensemble is uncertain, the model is likely to be incorrect and the policy should be prevented from learning from actions that cause incorrectness of the model. Another more recent model-based offline RL method, Combo, instead relies on conservative value function regularization [Yu et al., 2021].

To summarize, Table 2.1 gives an overview of the categories of offline RL algorithms.

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 Your knowledge hub WIEN

Authors	Model	Environment	Category
Wu et al., 2019	BRAC	Gym-MuJoCo	Policy contraint (direct)
Jaques et al., 2020	WOP	Open-domain dialogue	Policy contraint (direct)
Peng et al., 2019	AWR	Gym-MuJoCo	Policy contraint (penalty)
Nair et al., 2020	AWAC	Adroit, Gym-MuJoCo	Policy contraint (penalty)
Kumar et al., 2019	BEAR	Gym-MuJoCo	Policy contraint (penalty)
Fujimoto et al., 2019	BCQ	Gym-MuJoCo	Policy contraint (penalty)
Agarwal et al., 2020	REM	Atari, Gym-MuJoCo	Uncertainty estimation
Wang et al., 2020	CRR	DMControl	Policy constraint
Kumar et al., 2020	CQL	Adroit, Atari, Franka Kitchen, Gym-MuJoCo	Conservative value functions
Kidambi et al., 2020	MoREL	Gym-MuJoCo	Model-based
Yu et al., 2020	MOPO	Gym-MuJoCo	Model-based
Yu et al., 2021	Combo	Gym-MuJoCo, DMControl, Sawyer door opening	Model-based
Chen et al., 2021*	DT	Gym-MuJoCo, Atari	-
Janner et al., 2021 *	TT	Gym-MuJoCo	-
Kostrikov et al., 2021*	IQL	Gym-MuJoCo	-

Table 2.1: Offline RL architectures. Methods with * were published during writing.

2.2.3 Conservative Q-learning (CQL)

CQL is a state-of-the-art algorithm for offline RL [Kumar et al., 2020]. In principle, CQL modifies the Bellman error in Equation 2.12 by adding a penalty term:

$$\tilde{\mathcal{E}}(\theta) = \alpha\mathcal{C}(\theta) + \mathcal{E}(\theta) \tag{2.13}$$

where α represents a hyperparameter and $\mathcal{C}$ the penalty term. The assumption is that the Bellman error will be high for out-of-distribution actions and therefore the penalty term should decrease the impact such "false" high errors have on the gradient update, i.e., push down overly high Q-values [Levine et al., 2020]. Kumar et al. [2020] choose the penalty term to be:

$$\mathcal{C}(\theta) = \mathbb{E}_{s\sim\mathcal{D},a\sim\mu(a|s)}[Q_\theta(s,a)] - \mathbb{E}_{(s,a)\sim\mathcal{D}}[Q_\theta(s,a)] \tag{2.14}$$

The first term in Equation 2.14 minimizes the Q-values of all state-action pairs with actions coming from some distribution $\mu(a \mid s)$. The second term maximizes the Q-values of all state-action tuples in the dataset $\mathcal{D}$. The distribution $\mu(a \mid s)$ can be chosen adversarially, e.g., to maximize the regularized penalty term $\mathcal{C}(\theta)$ [Levine et al., 2020]:

$$\mu = \arg\max_{\mu} \mathbb{E}_{s\sim\mathcal{D}}[\mathbb{E}_{a\sim\mu(a|s)}[Q_\theta(s,a) + \mathcal{H}(\mu(\cdot \mid s))]] \tag{2.15}$$

where $\mathcal{H}$ is the entropy and is used as regularization term. This term pushes down overly high q-values. Kumar et al. [2020] show that $\mu(a \mid s) \propto \exp(Q(s,a))$ and thus Equation 2.14 becomes:

$$\mathcal{C}(\theta) = \mathbb{E}_{s\sim\mathcal{D}}[\log \sum_{a} \exp Q_\theta(s,a)] - \mathbb{E}_{(s,a)\sim\mathcal{D}}[Q_\theta(s,a)] \tag{2.16}$$

The conservative penalty term $\mathcal{C}(\theta)$ can easily be integrated into regular online RL algorithms. Kumar et al. [2020] build their implementations of CQL on top of SAC and QR-DQN [Dabney et al., 2018] for continuous and discrete control tasks, respectively.

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 Your knowledge hub WIEN

2.2.4 Online fine-tuning for offline RL

Another interesting research direction for offline RL are algorithms that combine offline pre-training with online fine-tuning. In NLP it is common practice to first pre-train language models on large corpora of text and consequently fine-tune them for individual downstream tasks [Devlin et al., 2019]. However, naively fine-tuning offline RL algorithms such as BRAC [Wu et al., 2019], BEAR [Kumar et al., 2019] or AWR [Peng et al., 2019] does not work satisfactorily [Nair et al., 2020], as the performance drops heavily once the agent transitions to online training. Therefore, Nair et al. [2020] proposed an algorithm called AWAC (implicit policy constraint, mentioned in Section 2.2) that adopts the pre-training, fine-tuning approach and delivers good performance under both circumstances. Recently, Lu et al. [2021] conducted a large-scale study on learning robotic skill with offline pre-training and online fine-tuning.

Both online RL and offline RL have their advantages and disadvantages. As we demonstrate in Chapter 5, offline RL is typically characterized by fast initial learning progress but low potential for long term improvement. This means that the learning curves of offline RL agents flatten out once the dataset has been leveraged to extent. Online RL, on the other hand, is characterized by slow initial learning, as the agents first has to discover high-reward behaviour by randomly interacting with the environment. However, it has, in contrast to offline RL, high potential for long term improvement, due to its freedom to explore the environment and to selectively seek useful information. Therefore, algorithms that unite both paradigms may be able to draw from the best of both worlds.

2.2.5 Challenges & limitations of offline RL

One apparent limitation of offline RL is that the dataset actually has to be obtained somehow in the first place. But how problematic is this in practice? As was discussed in Section 2.1.3 online learning can be impractical due to a variety of constraints, such as time, cost, or safety constraints. This is especially true, as each time an agent is trained, the online data collection has to be started anew. In contrast, in offline RL data collection and learning is decoupled. Thus, the data has to be collected only once and afterwards it can be used many times to learn policies offline. This, again, resembles the supervised learning paradigm, where datasets, such as ImageNet [Deng et al., 2009], LibriSpeech [Panayotov et al., 2015] or Penn Treebank [Marcus et al., 1993], also had to be collected in the first place. But once collected, they can then be used many times by hundreds of research labs around the globe to train algorithms that power AI applications in the real world. This speaks to the necessity of publicly available datasets for RL that can be used by the entire research community. For this purpose, two benchmarks were proposed only recently that provide offline datasets for a variety of tasks: D4RL [Fu et al., 2020] and RL Unplugged [Gülçehre et al., 2020]. We also use these benchmarks in this work, as will be described in Chapter 3. Currently, they only provide datasets for simulation-based environments (such as Atari games), but it can be expected that similar datasets for real-world tasks are going to be released soon (one real-world benchmark has recently been introduced by Lee et al. [2021]). In many domains, it is conceivably easier

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 Your knowledge hub WIEN

to collect data once, than to set up online environment interaction. For applications of offline RL in recommender systems, data can be collected by logging user behaviour and hence is relatively inexpensive. Similarly, for goal-oriented dialogue applications, user interaction with human operators or automated systems can be collected. In robotics and autonomous systems, it is possible to observe and store interactions of hundreds of devices that operate simultaneously. Likewise, for autonomous vehicles data collection is relatively easy compared to learning online. Hence, for many application areas, datasets might already exist, or it should be possible to construct them.

The main advantage of the offline RL framework, namely learning from previously collected datasets, can also be a major limitation. Exploration of the environment is a critical component of online RL and one of the core ingredients that make it work [Sutton and Barto, 2018, p. 3]. *Exploring* the environment means to visit parts of the state or action space that the agent has not seen before. Exploring the environment often stands in contrast with *exploiting* what the agent already knows about it (from prior experience) to obtain high reward. This dilemma is known as the trade-off between exploration and exploitation [Sutton and Barto, 2018, p. 3]. However, in offline RL the dataset is static and thus exploration is not possible. Therefore, it might often be desirable to let the RL agent continue to learn online to improve upon the behaviour it acquired during offline training. This combination of offline pre-training and online fine-tuning effectively unites the best of both worlds. The results of our experiments in Chapter 5 suggest that such hybrid algorithms are a promising path for future RL research.

In offline RL, the dataset is finite and may be relatively small, compared to the amount of experience that online RL agents require. In addition, data-inefficiency is a big concern in RL, as discussed in Section 2.1.3. To address these limitations, the available observations in the offline dataset have to be used as effectively as possible. Two promising approaches to achieve this are self-supervised learning and data augmentation. They are used to learn richer and more meaningful representations of the environment. In addition, they enable the agent to experience a more diverse environment. This is important as the offline dataset is static and might only contain a limited amount of samples. Therefore, more diversity can be a critical resource. In this work, we aim to apply self-supervised methods and data augmentation to offline RL in order to leverage the available observations in the dataset as effectively as possible. We discuss the theoretical background of self-supervised learning and data augmentation in the next two chapters.

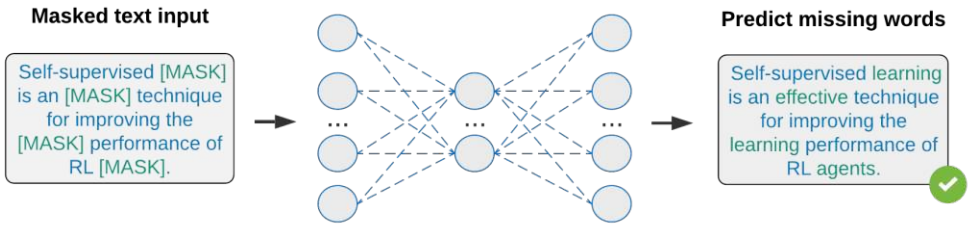
2.3 Self-supervised Learning

2.3.1 Overview

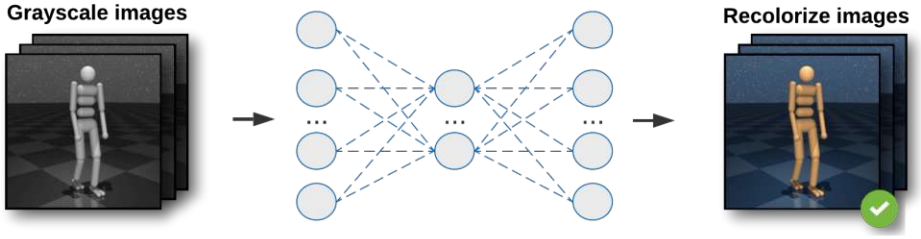
In recent years, self-supervised learning (SSL) has played a key role in some of the most impressive achievements throughout the field of Deep learning (DL) [Chen et al., 2020a; Devlin et al., 2019; Baevski et al., 2020]. In NLP self-supervised learning is successfully used to train large language models such as BERT [Devlin et al., 2019] (+ its variants [Liu et al., 2019; He et al., 2021]) or GPT-3 [Brown et al., 2020]. Similarly, self-supervised

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 Your knowledge hub WIEN

learning is the driving force behind recent advances in computer vision and speech recognition as demonstrated by systems such as SimCLR [Chen et al., 2020a,b], BYOL [Grill et al., 2020] and wav2vec 2.0 [Baevski et al., 2020].



(a) SSL in NLP.



(b) SSL in Computer Vision.

Figure 2.4: An illustration of self-supervised learning tasks.

Self-supervised learning is similar to both supervised and unsupervised learning, but yet decisively different. While supervised learning relies on the availability of labels within the dataset, unsupervised learning does not use any labels, but rather aims to find hidden structures within the unlabelled data points (as discussed in Section 2.1.1). Like in unsupervised learning, self-supervised learning does not take advantage of a labelled dataset. Rather, in self-supervised learning, the "labels" are generated artificially from the available unlabelled data points. Hence, SSL constructs its own labels. Like supervised learning, self-supervised learning then aims to predict the (artificially generated) labels correctly. Intuitively, self-supervised learning is analogous to "learning to fill the blanks".

The self-supervised paradigm can be very convenient compared to supervised learning, as it is often easily possible to obtain vast amounts of unlabelled data points, but in contrast hard to obtain labelled ones. For example, in NLP large quantities of text can simply be downloaded from the web. Language models then aim to predict the next word from a set of previous words or masked out words from the surrounding context words [Devlin et al., 2019]. In computer vision, self-supervised tasks include predicting the angle of rotation of an image, predicting the colours of a black-and-white image, or predicting a future frame from the current frame in a video [Jing and Tian, 2020]. In practice, self-supervised tasks may come in different shapes and sizes, but the overarching idea is simple: predict some part of a given input from another part. Figure 2.4 shows

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 Your knowledge hub WIEN

two illustrative self-supervised tasks.

The self-supervised learning paradigm has also shown to be an effective technique for improving the learning performance of online RL agents, in particular for vision-based environments [Jaderberg et al., 2017; Shelhamer et al., 2017; Laskin et al., 2020b; Schwarzer et al., 2021a]. In RL, an agent interacts with the environment it finds itself in. To interact more effectively, it is important to understand how the environment works, i.e., to acquire some sort of world knowledge. For this purpose, self-supervised methods in RL typically are concerned with predicting a particular part of the environment from another part, e.g., the future state (or states) and/or reward from the current state. This can be accomplished either directly in pixel space, but also in latent space [Jaderberg et al., 2017; Oord et al., 2018]. In fact, this has similarities with learning the dynamics of the environment in model-based RL, as discussed in Section 2.1.2. However, in model-based RL the learned environment model is usually employed to update policy or value-functions by simulating experience (a process known as *planning*) [Sutton and Barto, 2018, p. 160]. This is not necessarily the case for self-supervised methods in RL. Rather, they are used as auxiliary objectives in addition to the primary RL optimization target. The objective of self-supervised methods in the context of RL is to learn useful representations of the environment, that facilitate the interaction with it.

2.3.2 Self-supervised learning in RL

In RL, as in other domains, self-supervised learning can come in many shapes. In fact, it might even serve different purposes. For example, while some methods aim for more data-efficient learning [Laskin et al., 2020b; Schwarzer et al., 2021a], other architectures employ self-supervision to guide exploration [Pathak et al., 2017; Burda et al., 2018; Aytar et al., 2018; Pathak et al., 2019]. Table 2.2 gives a comprehensive, yet incomplete, overview of RL architectures that employ self-supervised learning. Self-supervised RL is an active area of research, and more advanced architectures are published on a continuous basis. Therefore, Table 2.2 was periodically expanded during the writing of this work.

Usually, self-supervised tasks are formalized as *auxiliary losses* or *auxiliary rewards*. Auxiliary losses define an additional loss function on the self-supervised task that is optimized in addition to the RL objective. Similarly, auxiliary reward tasks formulate additional self-supervised rewards that are added to the reward function.

Unreal. One of the first architectures to use self-supervised learning was Unsupervised Reinforcement and Auxiliary Learning (Unreal) [Jaderberg et al., 2017], shown in Figure 2.5. We use this model to illustrate self-supervised learning in RL. Unreal is based on A3C (an actor critic algorithm mentioned in Section 2.1.2) and uses two kinds of self-supervised tasks: control tasks and reward prediction. The authors proposed two kinds of control tasks, pixel control and feature control, but pixel control (maximizing the change in pixel intensity of experienced states) proved to be most effective. Assuming a set of auxiliary control tasks $\mathcal{C}$, the RL objective becomes $\arg\max_\theta \mathbb{E}_\pi[G_t] + \sum_{c\in\mathcal{C}} \lambda_c \mathbb{E}_{\pi_c}[G_t^{(c)}]$, where π, π_c, θ, $G_t^{(c)}$, and λ_c denote the regular RL policy, the policy for control task c, the set of

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.

Authors	Model	Environment	Description
Jaderberg et al., 2017	Unreal	Atari, 3D Mazes	Control & reward tasks
Shelhamer et al., 2017	-	Atari	Representation learning
Mirowski et al., 2017	-	3D Mazes	Depth prediction, loop closure tasks
Pathak et al., 2017	Curiosity	VizDoom, Super Mario	Exploration
Burda et al., 2018	-	Atari, Super Mario	Exploration
Pathak et al., 2019	Disagreement	Atari, VizDoom, Mujoco	Exploration
Laskin et al., 2020b	Curl	Atari, DMControl	Representation learning via contrastive learning
Schwarzer et al., 2021a	SPR	Atari	Representation learning via self-predictive representation
Mazoure et al., 2020	DIM	Ms.Pacman, Procgen	Representation learning via InfoMax
Yarats et al., 2021a	Proto-RL	DMControl	Exploration via prototypical representations
Schwarzer et al., 2021b	SGI	Atari	Representation learning via SPR and pre-training

Table 2.2: RL architectures that employ self-supervised methods.

parameters for π and all π_c's, the return, and the weights for the individual control tasks $c \in \mathcal{C}$, respectively. The return for control task c is made up of the auxiliary rewards $r_t^{(c)}$, based on which a separate Q-function $Q^{(c)}$ is learned. The loss function for a control task, denoted $\mathcal{L}_{Q^{(c)}}$, is the regular DQN loss (Equation 2.12). Similarly, the auxiliary reward prediction loss is denoted $\mathcal{L}_{RP}$. For this objective, the agent is tasked to predict whether the next reward r_{t+k+1} is positive, negative, or zero given a sequence of states $s_t, s_{t+1}, ..., s_{t+k}$ (consecutive image frames). Therefore, multi-class cross-entropy loss over the three classes is used for $\mathcal{L}_{RP}$. To summarize, the Unreal loss function is defined as:

$$\mathcal{L}_{Unreal}(\theta) = \mathcal{L}_{A3C} + \lambda_{VR}\mathcal{L}_{VR} + \sum_c \lambda_c \mathcal{L}_{Q^{(c)}} + \lambda_{RP}\mathcal{L}_{RP} \tag{2.17}$$

where $\mathcal{L}_{A3C}$ and $\mathcal{L}_{VR}$ (value function replay) refer to the A3C agent losses and $\lambda_{VR}, \lambda_c, \lambda_{RP}$ are hyperparameters to weigh the individual loss terms. The Unreal architecture demonstrates that self-supervised auxiliary tasks can simply augment RL agents and are optimized jointly with the primary learning objective.

Curl. Another successful model architecture is Curl [Laskin et al., 2020b]. Curl is based on *contrastive learning* (or contrastive predictive coding (CPC) [Oord et al., 2018]), a self-supervised representation learning strategy. Contrastive learning organizes data points into similar and dissimilar pairs to learn similar and dissimilar representations. In fact, the strategy is widely applied in computer vision (SimCLR is an example). The Curl architecture was applied to Atari games [Bellemare et al., 2013a] (100k environment interaction steps, as proposed by [Kaiser et al., 2020]), as well as DeepMind Control (DMControl) suite [Tassa et al., 2018] tasks and outperformed prior methods on most of them in terms of data-efficiency.

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 Your knowledge hub WIEN

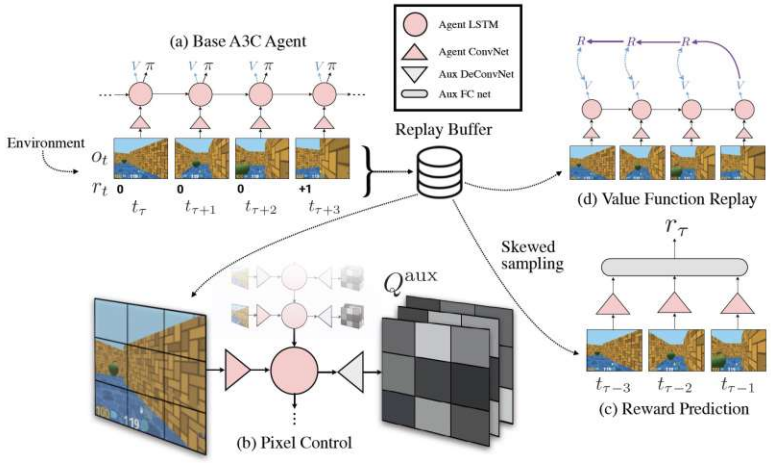


Figure 2.5: The Unreal architecture [Jaderberg et al., 2017].

Figure 2.6 depicts this architecture. The contrastive learning component in Curl is framed as a dictionary look-up task with query q (or anchor) and keys $\mathcal{K} = \{k_0, k_1, ...\}$ (or targets). One of the keys is a positive sample k_+ and all other keys are negative samples $\mathcal{K} \setminus \{k_+\}$. The goal of contrastive learning is that the anchor paired with positive keys obtains a higher matching score than the anchor paired with negative keys. Formally, this target can be expressed by the InfoNCE loss [Oord et al., 2018], corresponding to the "Contrastive loss" box depicted in Figure 2.6:

$$\mathcal{L}_q = \log \frac{\exp(q^T W k_+)}{\exp(q^T W k_+) + \sum_{i=0}^{n-1} \exp(q^T W k_i)} \tag{2.18}$$

where the term $q^T W k$ represents the bi-linear product with learned weight matrix W.

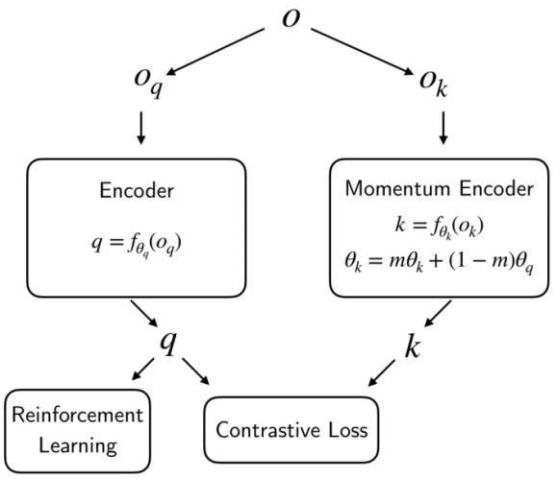


Figure 2.6: The Curl architecture [Laskin et al., 2020b].

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 Your knowledge hub WIEN

So, how do we obtain query and keys? They are constructed during environment interaction from an input observation (or state) in Figure 2.6 denoted by o (e.g., an image frame). While the raw query observation o_q and positive key observation o_{k_+} are different random crops of the *same* input observation o, the negative key observations o_{k_-} are random crops of *different* observations (e.g., other frames in the same batch). Hence, o_q and o_{k_+} are similar but o_q and o_{k_-} dissimilar. Nevertheless, the contrastive loss is not computed in pixel space but from latent representations thereof. Given an observation o_q and a neural network encoder f_{θ_q} with parameters θ_q, the query q is obtained by encoding o_q with f_{θ_q}. Similarly, to obtain k the key observations are encoded by f_{θ_k}, where θ_k is the exponential moving average (EMA) of they query encoder parameters θ_q:

$$\theta_k \leftarrow m\theta_k + (1-m)\theta_q \quad \text{with } m \in [0,1] \tag{2.19}$$

This approach to parameter sharing of query and key encoder is known as Momentum contrast (MOCO) [He et al., 2020]. Then q, the latent representation of the query observation, is fed into the RL agent. The assumption is that the latent representation encodes the environment more effectively (e.g., in a more compressed form that removes abstraction) and thus leads to more data efficient learning.
Curl simply extends two established RL algorithms SAC [Haarnoja et al., 2018] and Rainbow DQN [Hessel et al., 2018], but any RL algorithm could be used. Hence, the "Reinforcement learning" box in Figure 2.6 contains either of these algorithms. Again, the self-supervised method simply augments the RL agent.

SPR. Another powerful self-supervised approach for RL is SPR [Schwarzer et al., 2021a]. Just as Curl, SPR aims to improve data-efficiency through self-supervised representation learning. Instead of formulating a contrastive objective, SPR relies on next state prediction. It achieves this by learning a transition model that predicts the future latent state representations. Unlike in model-based RL, the transition model is not used to update the policy or value functions, but simply to learn more effective environment representations. The SPR architecture is depicted in Figure 2.7. Conceptually, SPR is similar to Curl as it aims to learn an effective latent representation of the current state that can be fed into the RL agent. In Figure 2.7, the current observation and latent representation are denoted by s_t and z_t, respectively (before o and q). For simplicity, we use the same variables as depicted in the Figure 2.7.

In addition, SPR applies data augmentations to input states ($s_t + \text{aug}$), this will be the topic of Section 2.4 and thus ignored for now. Overall, SPR consists of an online encoder f_o that transforms observed states s_t to their latent representations $z_t = f_o(s_t)$ and a target encoder f_m that encodes future states to their latent representations, $\tilde{z}_{t+k} = f_m(s_{t+k})$. The parameters of online and target encoders (neural networks) are denoted θ_o and θ_m, respectively. As before, the encoder neural network parameters are shared: θ_m is an EMA of the online encoder parameters and thus is not updated by gradient descent. Furthermore, to predict the next K future latent states SPR employs an action-conditioned transition model h. For each time step $t+k$, the transition model takes the latent representation of a state and the action performed, z_{t+k} and a_{t+k}, and

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 Your knowledge hub WIEN

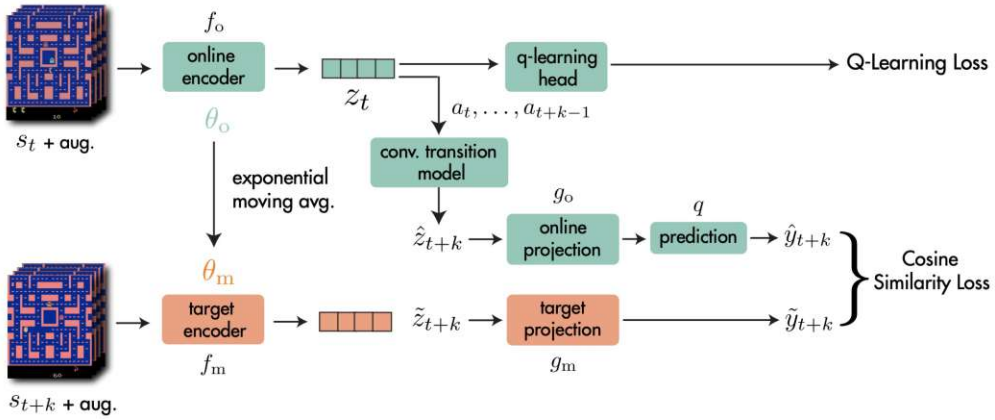


Figure 2.7: The SPR architecture [Schwarzer et al., 2021a].

produces the next latent representation. This process is repeated until the prediction depth K is reached. Thus, the transition model produces predictions of the future latent states $\hat{z}_{t+1:t+K} = h(\hat{z}_{t+k}, a_{t+k})$ (with $\hat{z}_t = z_t = f_o(s_t)$). The actual next K future states $s_{t+1:t+K}$ are fed through the target encoder to obtain the latent representations of the actual future states: $\tilde{z}_{t+k} = f_m(s_{t+k})$. The transition model h is a Convolutional Neural Network (CNN) and the maximum prediction depth K is set to 5. Then, as Figure 2.7 illustrates, the latent representations of predicted future states $\hat{z}_{t+k}$ and latent representations of actual future states $\tilde{z}_{t+k}$ are fed into additional online and target projection heads, g_o and g_m, to obtain the final latent online and target representations $\hat{y}_{t+k}$ and $\tilde{y}_{t+k}$, respectively. Again, the projection heads are neural networks, and the target projection parameters are an EMA of the online projection head. Finally, the cosine similarities between final online and target representations is used as the prediction loss:

$$\mathcal{L}_{SPR}(s_{t:t+K}, a_{t:t+K}) = -\sum_{k=1}^{K} \left(\frac{\tilde{y}_{t+k}}{\|\tilde{y}_{t+k}\|_2} \right)^T \left(\frac{\hat{y}_{t+k}}{\|\hat{y}_{t+k}\|_2} \right) \tag{2.20}$$

Hence, the overall loss is given by $\mathcal{L}_{Total} = \mathcal{L}_{RL} + \lambda_{SPR}\mathcal{L}_{SPR}$, similar to Equation 2.17 with the SPR weight λ_{SPR}. The regular RL loss is represented by $\mathcal{L}_{RL}$ (e.g., Equation 2.12). In fact, the authors augment Data-efficient Rainbow DQN [van Hasselt et al., 2019] with self-predictive representations, and the resulting architecture outperformed prior methods on Atari games in terms of data-efficiency [Schwarzer et al., 2021a]. Most importantly, the SPR loss also just augments the RL objective.

2.3.3 Outlook

Overall, these architectures demonstrate that self-supervised learning can be a powerful extension to the RL framework. Yet, all discussed algorithms were developed for the online setup. The rest of this thesis aims to adjust the proposed methods for offline RL.

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 Your knowledge hub WIEN

2.4 Data Augmentation

2.4.1 Overview

Data augmentation (DA) is an established technique that has shown to be effective for many ML applications, in particular for images-based tasks [Perez and Wang, 2017; Shorten and Khoshgoftaar, 2019]. The idea of data augmentation is to artificially increase the amount of available training data by modifying existing training observations in some meaningful way. The modified samples are then used as additional training samples. This simple yet powerful technique has shown to mitigate overfitting, i.e., it strengthens the algorithm's ability to generalize to unseen data points [Shorten and Khoshgoftaar, 2019]. Thus, data augmentation is especially effective if the amount of training data at one's disposal is limited or the training data exhibits high class-imbalance (e.g., in domains such as medicine & biology).

There are many ways to augment training observations. Examples are geometric transformations (such as rotating, resizing, flipping or cropping an image), colour transformations (such as colour removal/filtering, colour jitter, colour scaling, or adding noise), random erasing methods (such as cutout) and even combinations of the former [Shorten and Khoshgoftaar, 2019]. Nevertheless, not every augmentation is suitable, or "safe" for every task. For instance, in image classification while it is fine to flip an image of a dog horizontally/vertically, the same augmentation might not be advantageous for digits ("9" becomes "6" and vice versa) [Shorten and Khoshgoftaar, 2019]. Figure 2.8 illustrates the concept of data augmentations and shows a few exemplary tasks.

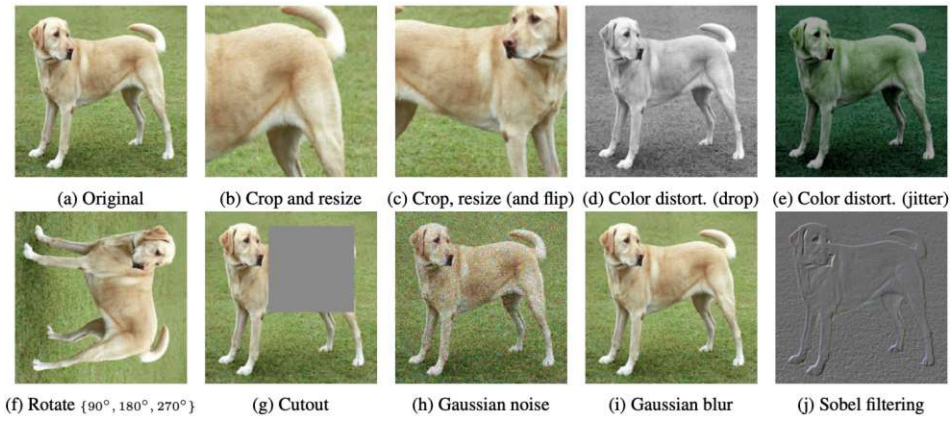


Figure 2.8: An illustration of data augmentations. Figure from [Chen et al., 2020a]

Only recently, the RL community has started to embrace data augmentations techniques. Their effectiveness has been primarily demonstrated in the context of improving that data efficiency of online RL algorithms [Yarats et al., 2021b; Laskin et al., 2020a; Raileanu et al., 2021; Hansen and Wang, 2021]. Individually, data augmentation based online RL agents have even been shown to deliver better performance in terms of data-efficiency than

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 Your knowledge hub WIEN

RL agents augmented with self-supervised auxiliary tasks. However, the combination of both approaches delivered the best performance, as demonstrated by SPR [Schwarzer et al., 2021a].

As self-supervised auxiliary tasks, data augmentation techniques are useful for RL, as they enable the agent to learn more meaningful representations of the environment. By modifying observations, data augmentations allow the agent to experience a more diverse environment. The consequence is a more data-efficient learning behaviour. We assume that data augmentations will be particularly useful for offline RL, as in this scenario the dataset is static and hence overfitting can be an issue.

2.4.2 Data augmentation in RL

Data augmentations are, as self-supervised methods, an extension to the general RL framework. They can in principle be used with any kind of RL algorithm. Assuming the standard MDP formulation, the only modification is that a given state s is fed through a transformation function $f(s, v)$ that applies the data augmentation before it is used to update the policy/value function/model. Where v is some augmentation from the set of all data augmentations, $v \in \mathcal{V}$. In online RL, the data augmentations are typically applied randomly to all samples across a batch (and consistently across a frame stack for vision-based tasks). For offline RL, different strategies (e.g., augmenting the whole dataset beforehand) are conceivable.

In RL, however, augmentation "safety" is even more critical compared to other tasks, as agent-environment interaction is subject to certain semantics. For example, in a 2D Maze environment an agent may be able to move into every direction, i.e., $\mathcal{A} = \{\text{up}, \text{down}, \text{left}, \text{right}\}$. Now, if the observation is flipped horizontally, left becomes right and right becomes left. Thus, the transformation alters the action semantics. Therefore, when choosing data augmentations, these constraints have to be considered. In general, data augmentations that do not change the environment behaviour (too severely) are most desirable.

In Table 2.3, we summarize RL architectures that employ data augmentation to improve the online learning performance. These architectures mostly differ in the kinds of augmentations they use, but also in the way the augmentations are applied.

RAD. One such method is RAD [Laskin et al., 2020a]. Similar to the self-supervised methods discussed in Section 2.3, RAD augments two established RL agents, SAC (off-policy) and PPO (on-policy). During the learning process, the augmentations are applied to either a batch sampled from the replay buffer (off-policy) or a recent trajectory (on-policy). Overall, they evaluate ten different image-based data augmentations: random crop, translate, window, greyscale, cutout, cutout-colour, flip, rotate, random convolution, and colour-jitter. In addition, two state-based data augmentations are applied: random amplitude scaling, which multiplies the state s with a uniform random variable (i.e., $s * \mathcal{U}(\alpha, \beta)$, where α, β are lower and upper bound, respectively), and Gaussian noise, which adds a Gaussian random variable (i.e., $s + \mathcal{N}(0, 1)$). Finally, the authors evaluate

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 Your knowledge hub WIEN

Authors	Model	Environment	Description
Laskin et al., 2020a	RAD	DMControl, Procgen	Crop, cutout, flips, rotate, etc.
Yarats et al., 2021b	DrQ	Atari, DMControl	Cutout, flips, random shift, intensity, etc.
Raileanu et al., 2021	DrAC	DMControl, Procgen	Upper confidence bounds, meta learning
Hansen and Wang, 2021	SODA	DMControl	Soft data augmentation
Schwarzer et al., 2021a	SPR	Atari	Self-predictive representations + data augmentation

Table 2.3: RL architectures that employ data augmentations.

the augmentations on DMControl [Tassa et al., 2018] as well as ProcGen bechmark tasks [Cobbe et al., 2020]. Their results suggest that all augmentations improve performance, but individually random cropping and cutout are the most effective tasks. Even though the technique is very simple, RAD delivered excellent performance and outperformed prior methods, such as Curl in terms of data efficiency [Laskin et al., 2020a]. Their results suggest that data augmentation can be an effective tool for the offline RL scenario.

DrQ. Another recent architecture that uses data augmentations to improve the learning performance of the underlying RL agent is DrQ [Yarats et al., 2021b]. The authors extend the SAC agent and observe similar performance gains as RAD on DMControl [Tassa et al., 2018]. In addition, the method was also applied to Atari games and obtained state-of-the-art results in this domain by extending a DQN agent [Bellemare et al., 2013a]. DrQ also employs a range of data augmentations (there is an overlap with RAD) but primarily focuses on random shifting, as this augmentation proved successful in preliminary experiments. In addition, the authors also proposed to augment the input observations several times and to average the Q-values of the respective transformed inputs. In particular, Yarats et al. [2021b] average the target Q-values over K augmentations and the regular Q-values over M augmentations. In their work $[K = 2, M = 2]$ worked best.

SPR. Furthermore, SPR also employed data augmentations in addition to self-supervised auxiliary tasks [Schwarzer et al., 2021a]. In fact, SPR simply applies the same set of data augmentations that was proposed for the DrQ architecture [Yarats et al., 2021b]. The given observations are augmented before being used in the self-supervised objective and RL objective, as shown in Figure 2.7. Overall, data augmentations significantly improved the SPR agent's performance on Atari benchmark tasks when compared to only using the self-supervised methods. Hence, their results suggest that self-supervised methods and data augmentation are largely orthogonal and complement each other effectively.

2.4.3 Outlook

The exemplary architectures presented in this section demonstrate that data augmentation, just like self-supervised methods, is an effective technique for improving the learning

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 Your knowledge hub WIEN

performance of online RL agents. The online RL architectures also show that data augmentations, as well as self-supervised methods, simply extend the RL framework. In addition, both approaches complement each other. Nevertheless, the effectiveness of data augmentation has not yet been investigated in the offline RL context. In offline RL, the dataset is static and potentially limited in size or diversity. Data augmentation is well suited for this setup, as it enables the agent to experience a more diverse environment and thus may prevent overfitting.

2.5 Related work

Closest to this thesis is recent work known as Surprisingly Simple Self-Supervised offline RL (S4RL) [Sinha et al., 2021]. S4RL was published during the writing of this thesis, and also studies data augmentation in the context of offline RL. They show that data augmentation in the style of DrQ can be an effective technique for offline RL. Similar to our work, their underlying offline RL algorithm is CQL.

There are, however, many differences to our work. S4RL only uses data augmentations in DrQ style, while we also study the effectiveness of a number of self-supervised architectures for offline RL, as well as online fine-tuning for offline RL. Furthermore, Sinha et al. [2021] only test their algorithm on state-based continuous control tasks, while we also evaluate on image-based discrete control tasks. Thus, also the selected data augmentations differ.

2.6 Summary

In this chapter, we discussed the theoretical background for this thesis. First, we introduced the general **online RL** framework in Section 2.1. In recent years, online RL has led to impressive achievements, but it still has limitations. Real world applications are usually subject to certain restrictions, such as safety, time, or resource constraints. In particular, the online-interaction and trial-and-error setup makes it unsuitable for many real-world use cases. In addition, current online RL algorithms are incredibly data inefficient and thus often only work well for simulation-based environments that are not subject to real-world constraints.

In Section 2.2, we reviewed the main subjects of this work, **offline RL**. Offline RL offers an attractive alternative to the online framework, as it decouples environment interaction and learning process. The offline RL framework is comparatively simple to apply in practice and thus has a wide range of real-world application areas, from robotics to healthcare. However, its main advantage is also a major limitation: the static, previously collected dataset and no possibility for environment interaction. Therefore, the available observations have to be leveraged as effectively as possible. Two methods to achieve this are self-supervised learning and data augmentation. Both approaches have already been applied to online RL but have not yet been studied in the offline RL context. Furthermore, in many cases, it might be desirable to continue learning online after an initial offline pre-training phase to combine the benefits of online RL and offline RL.

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 Your knowledge hub WIEN

We described **self-supervised methods for RL** in Section 2.3. The overall idea of self-supervision is to predict some part of the input from another part. First, we illustrated the general idea of self-supervised learning by a few exemplary tasks from diverse domains. Then, self-supervised learning was discussed in the context of RL. Lately, a variety of different self-supervised methods, such as contrastive learning or self-predictive representations, has been proposed that serve different purposes. Our discussion focuses on methods that aim to improve the data-inefficiency problem. Our analysis of a few selected algorithms, namely Unreal [Jaderberg et al., 2017], Curl [Laskin et al., 2020b] and SPR [Schwarzer et al., 2021a], has shown that self-supervised tasks plug into the RL framework relatively seamlessly and require only minor adjustments of the underlying algorithms. Most importantly, they significantly improve the learning performance of RL agents in terms of data-efficiency on a multitude of diverse benchmark tasks.

Similarly, in Section 2.4 we reviewed the literature on **data augmentation for RL**. As with self-supervised tasks, the recent years have brought forth many architectures that integrate data augmentation techniques into RL agents. Again, our analysis of established architectures, RAD [Laskin et al., 2020a], DrQ [Yarats et al., 2021b] and SPR [Schwarzer et al., 2021a], has shown that data augmentation is a powerful extension to the RL framework. In offline RL, the agent is faced with a static dataset and data augmentation techniques offer a way to construct more diverse experience to inhibit overfitting. Consequently, data augmentation led to more data efficient learning behaviour in online RL.

The rest of this thesis aims to apply self-supervised methods, data augmentations and online fine-tuning to offline RL.

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 Your knowledge hub WIEN

CHAPTER 3

Methodology

In this section, we explain the methodological approach of this thesis. First, we outline the general structure of our approach and the steps that we conduct. In addition, we illustrate our implementation workflow. Then each of the steps is examined in more detail individually. In particular, we describe the selection of offline RL agents, self-supervised methods, and data augmentations used in this work. Furthermore, we examine our task setup and evaluation procedure. This includes the selected offline datasets & benchmarks, baseline architectures, as well as the specific target metrics we employ to compare our agents.

3.1 Overview

3.1.1 Structure

The methodological approach of this thesis consists of the following steps:

1. **Algorithm selection**.
   Drawing from the insights we obtained in Chapter 2, we first select the most promising algorithms for our analysis. There are three major types of decisions that we need to make:

   a) What **offline RL agents** do we select?

   b) What **self-supervised methods/architectures** do we select?

   c) What **data augmentations** do we select?

   In Section 3.2, we give detailed answers to these questions. In particular, we discuss our choices and the rationales for them.

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 Your knowledge hub WIEN

2. **Evaluation**.
   Next, the task selection and evaluation procedure have to be defined. Therefore, we choose suitable datasets/environments/tasks to compare the architectures selected in the previous step, baselines to compare them against as well as the respective target metrics.

   a) **Offline datasets & Benchmarks**: The offline datasets/environments/tasks have to be suitable for comparing the performance of different architectures. Therefore, we evaluate all methods on selected tasks from established open-source benchmark suites D4RL [Fu et al., 2020] and RL Unplugged [Gülçehre et al., 2020]. Both benchmarks consist of tasks that are characterized by properties relevant for real-world applications. To provide a comprehensive evaluation of our algorithms, we select both continuous as well as discrete control tasks. In particular, in this work, we focus on Gym-MuJoCo robotics tasks from D4RL and Atari games from RL Unplugged. We discuss our selection in more detail in Section 3.3.1

   b) **Evaluation metrics**: Appropriate evaluation metrics ensure a fair comparison of all architectures. We discuss our selection in Section 3.3.2.

   c) **Baselines**: To compare the proposed architectures against prior methods, established baselines (e.g., behavioural cloning, online/offline RL algorithms without self-supervised methods and augmentations) have to be selected. We discuss our selection of baselines in Section 3.3.3.

3. **Implementation & setup**.
   In Section 3.4 we give an overview of the software stack used for our implementations as well as our hardware setup.

3.1.2 Methodology Workflow

Before we explain our choices for the model components, we give a high-level overview of how the individual pieces fit together. Figure 3.1 illustrates our approach and Algorithm 3.1 shows the (simplified) procedure using pseudocode.

The entry point is the offline dataset denoted by $\mathcal{D}$ that is provided by the respective benchmark suite. The dataset stores previously collected experience tuples for a particular target task (e.g., a specific Atari game or Gym-MuJoCo task). The experience might originate from one or multiple behaviour policies π_β, as was discussed in Section 2.2.

At each time step a batch of experience tuples $\mathcal{B} = \{(s, a, s', r)\}$ is sampled from the offline dataset D, where s, a, s', and r refer to state, action, next state and reward. The selected data augmentations $v \in \mathcal{V}$ are applied to the states, where $\mathcal{V}$ represents the set of all data augmentations. Augmentations are applied randomly across a batch, but by default we apply the same augmentations consistently to s and s'. We discuss our augmentation strategies in more detail in Section 4.4. Depending on the current experiment, data augmentation is optional (e.g., if only self-supervised tasks are examined).

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 Your knowledge hub WIEN

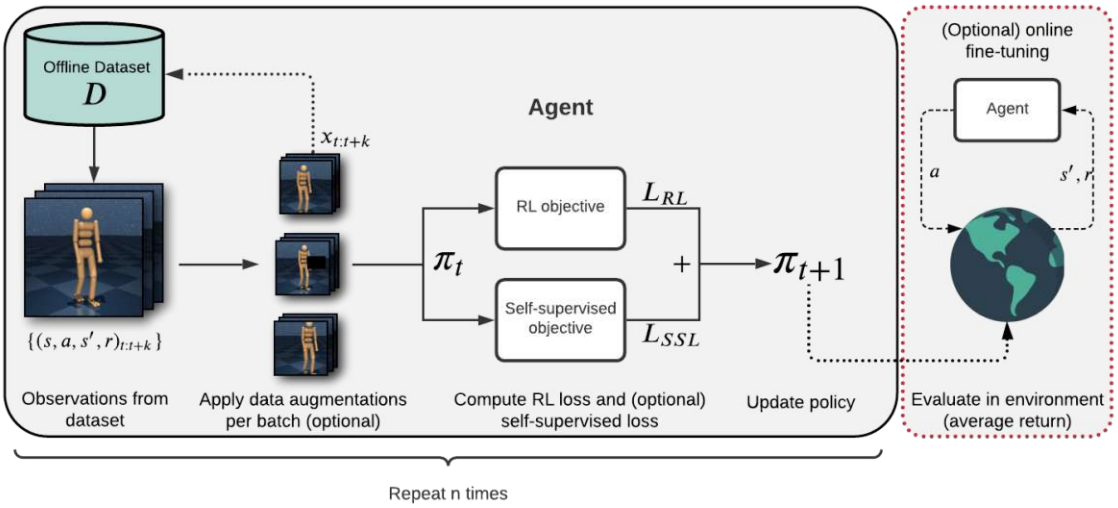


Figure 3.1: Methodology

The batch of (augmented) experience tuples is then passed on to the RL agent. The RL agent consists of the regular RL objective and a self-supervised objective. Depending on the offline RL algorithm selection, the contents of the RL objective box in Figure 3.1 can take different forms (e.g., Equation 2.12 with 2.16). Similarly, the self-supervised objective depends on the selected self-supervised method (e.g., Equations 2.18 or 2.20). As with data augmentations, the self-supervised method is optional and depends on the conducted experiment. Both the RL and self-supervised objective result in individual losses that are optimized jointly via SGD to improve the policy π.

All steps are repeated n times, where n is the number of update steps to perform. In our experiments, 10 thousand update steps constitute one epoch (or episode). After each episode (or every ith episode), the agent is evaluated within the actual environment based on average return to measure its learning progress. If desired, the agent can continue learning in the regular online fashion after an initial offline pre-training period.

3.2 Algorithm selection

In this section, we discuss the choice of algorithms we use in the subsequent experiments and the rationale for our selection.

3.2.1 Offline RL agents

First, we have to select the base offline RL algorithm for our subsequent experiments. The offline RL algorithm forms the basis of the agent and is extended with self-supervised methods and data augmentations. In principle, any kind of existing offline RL 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 Bibliothek Your knowledge hub WIEN

Algorithm 3.1: Workflow of methodology

Input: RL agent, environment, $\mathcal{D} = \{(s, a, s', r)_i\}$, set of augmentations $\mathcal{V}$, SSL objective, SSL weight λ, # of epochs, # of steps per epoch, eval freq

Initialize RL agent parameters θ
Initialize SSL objective parameters ϕ (optional)
for *epoch* $\in [0, ..., n\_epochs]$ **do**
  for *step* $\in [0, ..., steps\_per\_epoch]$ **do**
    $\mathcal{L} \leftarrow 0$
    $\mathcal{B} \sim \mathcal{D}$; // Sample batch $\mathcal{B} = \{(s, a, s', r)\}$ from $\mathcal{D}$
    if $\mathcal{V} \neq \emptyset$ **then**
      // Apply $v \in \mathcal{V}$ to current and next states s, s' in $\mathcal{B}$
      for $v \in \mathcal{V}$ **do**
        $s \leftarrow v(s)$
        $s' \leftarrow v(s')$
      end
    end
    // Compute RL loss, e.g., Equation 2.12 with 2.16
    $\mathcal{L} \leftarrow \mathcal{L} + \mathcal{L}_{RL}(\mathcal{B}; \theta)$
    if *SSL objective* **then**
      // Compute SSL loss, e.g., Equations 2.18, 2.20
      $\mathcal{L} \leftarrow \mathcal{L} + \lambda \mathcal{L}_{SSL}(\mathcal{B}; \phi)$
    end
    // Update θ, ϕ w.r.t. $\mathcal{L}$
    $\theta \leftarrow \theta + \alpha \nabla_\theta \mathcal{L}$
    $\phi \leftarrow \phi + \alpha \nabla_\phi \mathcal{L}$
  end
  if *eval_freq* % *epoch* $== 0$ **then**
    Evaluate agent in environment
    Compute performance metric m, report performance
  end
end

could be used for this purpose, because as we have seen in the last chapter, self-supervised learning and data augmentation plug into the RL framework relatively seamlessly. However, the selected algorithm should be as generally applicable as possible. Ideally, it should work with both discrete and continuous action spaces. The Atari test bed and Gym-MuJoco control tasks are examples for environments with discrete and continuous action spaces, respectively. Another obvious important criterion is performance: the algorithm should give state-of-the-art results and perform well across a wide range of tasks. Furthermore, to avoid additional architectural overhead, it should be relatively simple to implement. Hence, our decision is guided by three major criteria: **(1)** applicability to continuous and discrete action spaces, **(2)** performance, and **(3)** simplicity.

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 Your knowledge hub WIEN

Algorithm	Discrete control	Continuous	Performance	Simplicity
AWAC	✓	✓	✗	✓
BCQ	✓	✓	✗	✓
BEAR	✗	✓	✗	✓
REM	✓	✓	✗	✓
CRR	✗	✓	✓	✓
CQL	✓	✓	✓	✓
MoREL	✗	✓	✓	✗
MOPO	✗	✓	✓	✗
Combo	✗	✓	✓	✗

Table 3.1: Decision table for base offline RL algorithm.

In Table 3.1 we assess potential target algorithms according to the established criteria. For continuous control tasks (Gym-MuJoCo) the model-based method Combo has shown to outperform other approaches in terms of overall performance [Yu et al., 2021]. For the exact scores and a detailed analysis of the results, we refer to the original publications. The next best performing algorithms on these tasks were CQL [Kumar et al., 2020] and MOPO [Yu et al., 2020]. While Combo and MOPO are designed for continuous action space, CQL also works for discrete ones. Furthermore, CQL has also shown to deliver the best results for Atari benchmark games, outperforming REM [Agarwal et al., 2020]. In addition, CQL is relatively simple to implement, as it only requires minor adjustments to existing Q-learning or actor-critic methods. We already described the mathematical foundations of CQL in Section 2.2.3. Therefore, CQL is a robust choice, and we select it as our base offline RL agent for our subsequent experiments.

3.2.2 Self-supervised methods

Next, the self-supervised methods to extend the offline RL agent with have to be selected. We refer to Table 2.2 for a list of existing architectures. In principle, all self-supervised methods work for both discrete and continuous action spaces. However, not every task that works well for online RL, may also be suitable for the offline setup. For example, the Unreal agent employs a pixel control task, which requires online interaction [Jaderberg et al., 2017]. Similarly, tasks that aim to improve exploration, such as curiosity-based methods or methods that rely on disagreement [Pathak et al., 2017; Burda et al., 2018; Pathak et al., 2019], are better suited for online RL as in offline RL exploration is not possible. Hence, our decision is again guided by three criteria: **(1)** offline applicability, **(2)** performance, and **(3)** simplicity.

Curl [Laskin et al., 2020b], SPR [Schwarzer et al., 2021a], DIM [Mazoure et al., 2020], and SGI [Schwarzer et al., 2021b] remain. SPR, DIM and SGI were either applied to discrete or continuous control tasks, but not both. In contrast, Curl has been applied to both categories. Therefore, the first architecture we select is Curl, as it has shown

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 Your knowledge hub WIEN

to deliver excellent performance for both Atari (discrete) and DMControl (continuous). In addition, it is relatively simple to implement, as was discussed in Section 2.3. SPR outperforms Curl on Atari games and is also relatively straightforward to implement. Therefore, we also select SPR. Nevertheless, SPR has not been applied to continuous control tasks and requires minor adjustments that we discuss in the next chapter. Most importantly, both approaches are suitable for offline RL without major modifications. The mathematical background of Curl and SPR was examined in Section 2.3.

Algorithm	Applicable to Offline RL	Performance	Simplicity
Unreal	✓	✗	✓
[Pathak et al., 2017]	✗	✗	✓
[Burda et al., 2018]	✗	✗	✓
[Pathak et al., 2019]	✗	✗	✓
Curl	✓	✓	✓
SPR	✓	✓	✓
DIM	✓	✓	✗
SGI	✓	✓	✓

Table 3.2: Decision table for self-supervised methods.

Even though the underlying architecture of DIM is interesting, we discard this method, because it has thus far only been applied to Procgen and Ms.Pacman. SGI is based on the architecture of SPR, but instead relies on pre-training on massive amounts of data for a single Atari game. This is out of reach for our experiments. While SGI had to leverage a huge dataset, the performance improvements were of a similar magnitude. Therefore, we also select SGI, but restrict the amount of data to pre-train on. Just as SPR, SGI was developed for Atari and has not been applied to continuous control tasks. Therefore, we adjust the algorithm accordingly for continuous control tasks. We discuss our modifications to SGI in more detail in Section 4.

3.2.3 Data augmentations

Data augmentations typically work very well for image-based tasks, as images allow for a variety of manipulations to the pixel space, as was discussed in Section 2.4. But augmentations also can also be used with environments that emit state-based observations (e.g., proprioceptive information from robot joints) as demonstrated by RAD. In Section 2.4, we showed that prior work has evaluated a variety of augmentations. For our subsequent experiments with image-based tasks, we select the best-performing image-based data augmentations from RAD [Laskin et al., 2020a] and DrQ [Yarats et al., 2021b], namely random-cropping, random shifting and cutout. Likewise, for state-based environments we select random amplitude scaling and Gaussian noise, as was proposed by Laskin et al. [2020a].

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 Your knowledge hub WIEN

3.3 Evaluation

In this section, we describe how we evaluate our agents. This includes the actual tasks they have to perform and specific metrics to compare them against.

3.3.1 Offline Datasets & Benchmarks

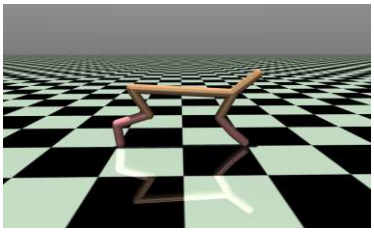
General. First, we have to select suitable tasks/offline datasets. As mentioned in Section 2.2.5, two benchmarking frameworks for offline RL were introduced recently, D4RL [Fu et al., 2020] and RL Unplugged [Gülçehre et al., 2020]. These frameworks define benchmark tasks, provide the corresponding offline datasets, and facilitate the evaluation of new architectures. D4RL provides datasets for seven different environments: Maze2D, AntMaze, Gym-MuJoCo, Adroit [Rajeswaran et al., 2018], FrankaKitchen [Gupta et al., 2020], Flow [Vinitsky et al., 2018], and CARLA (a simulator for autonomous driving) [Dosovitskiy et al., 2017]. RL Unplugged in contrast includes DMControl [Tassa et al., 2018], DMLocomotion, Atari [Bellemare et al., 2013a; Agarwal et al., 2020], and the Real-World Reinforcement Learning Suite [Dulac-Arnold et al., 2020]. Of all ten environments, only Carla (D4RL) and Atari (RL Unplugged) are image-based, i.e., the description of the environment state is provided in pixel-values.

As Table 2.1 illustrates, 13 of the 15 listed offline RL algorithms evaluate on Gym-MuJoCo. Recently, Atari has also emerged as an attractive benchmark for novel image-based offline RL architectures [Agarwal et al., 2020; Kumar et al., 2020; Schwarzer et al., 2021b]. In fact, Atari is one of the most established benchmarks for online RL agents [Bellemare et al. [2013b]; Machado et al. [2018b]]. While the Gym-MuJoCo benchmark consists of continuous control tasks, the Atari benchmark consists of discrete control tasks. Thus, we identify Gym-MuJoCo and Atari as the most suitable benchmark tasks for this work.

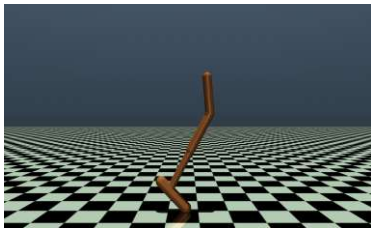
Continuous control. We select the Gym-MuJoCo benchmark from D4RL as our first target task. MuJoCo is a high-fidelity physics engine that is able to simulate a multitude of environments [Todorov et al., 2012]. It is widely used throughout the field of RL, which is why it was also included in D4RL. MuJoCo has been a commercial software and required a licence for a long time. During the course of writing, however, the MuJoCo simulator was acquired by DeepMind, open-sourced and is now free to use for any purpose. Due to this overlap, we initially used the PyBullet simulator [Coumans and Bai, 2016], an open-source alternative to MuJoCo, for our Gym experiments. When MuJoCo was open-sourced, however, we switched to MuJoCo and repeated all experiments on the new simulator. Therefore, all scores shown for continuous control tasks in Chapter 5 were obtained using MuJoCo as the underlying physics engine. Consequently, in this work, we use **Gym-MuJoCo** as the state-based benchmark.

Every Gym benchmark task dataset consists of 1 million transitions. Following the original D4RL publication [Fu et al., 2020], we train each agent for 500K gradient steps on Gym tasks, i.e., 0.5 × the dataset size. In this benchmark, the RL agent has to learn to walk with different robotic bodies/characters. We select the `medium` variants of the three most popular benchmark tasks: `Halfcheetah`, `Hopper`, and `Walker2d`.

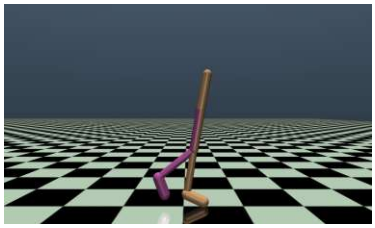
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN



(a) Halfcheetah

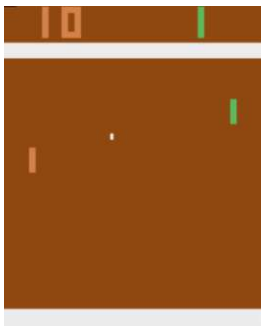


(b) Hopper

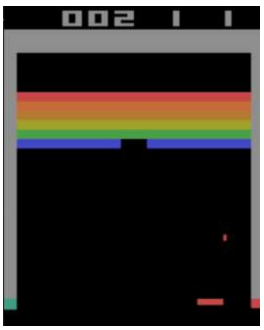


(c) Walker2d

Figure 3.2: The Gym-MuJoCo tasks used in this work.

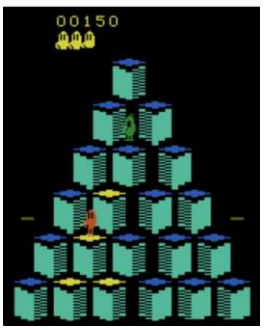


(a) Pong



(b) Breakout

(c) Seaquest



(d) Qbert

Figure 3.3: The Atari tasks used in this work.

Discrete control. To understand how well the effectiveness of self-supervised methods and data augmentations generalizes across domains, we also aim for the discrete-control Atari benchmark. The original DQN architecture proposed to limit environment interaction to 50 million frames for a single game, which corresponds to 38 days of real-time experience [Mnih et al., 2015]. Similarly, prior work in offline RL also employs a dataset consisting of 50 million transitions, which is equivalent to hundreds of gigabytes for a single Atari game [Agarwal et al., 2020; Kumar et al., 2020; Schwarzer et al., 2021b]. Given our limited resources, using such a huge amount of data is infeasible for our work.

Recently Agarwal et al. [2020] and Kumar et al. [2020] proposed to evaluate offline RL algorithms on 1%, 10%, 20% and 50% of the original dataset size. Using only 1% of the dataset considerably reduces the data size. Under these circumstances, our selected base-agent, CQL, has shown to deliver excellent results. We discussed, a similar setup 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 Bibliothek Your knowledge hub WIEN

Section 2.3, that is commonly used to evaluate the performance of data-efficient online RL architectures, such as Curl and SPR. These methods also evaluate on Atari games but limit the number of environment interaction steps to 100K, which corresponds to roughly two hours of real-time experience. This setup was proposed by Kaiser et al. [2020] and is known as Atari100K. For the purpose of this work, restricting the dataset size is ideal. Therefore, we adopt their setup and select **offline Atari1M** as the image-based benchmark for this thesis. Using 1 million frames ensures a fair comparison with the Gym tasks, which also consists of 1 million observations. Agarwal et al. [2020] and Kumar et al. [2020] set the number of gradient steps for each game to 5× the number of observations, i.e., 2.5 million steps. To ensure a fair comparison, we also set the number of gradient steps to 2.5 million. In total, there are 60 different Atari games. Unfortunately, a complete evaluation on all games is out of reach for this work. Therefore, we select four Atari games that were evaluated in detail by Agarwal et al. [2020] and Kumar et al. [2020], namely `Pong`, `Breakout`, `Seaquest`, and `QBert`.

Figure 3.2 and Figure 3.3 depict the Gym-MuJoCo and Atari tasks used in this work.

3.3.2 Metrics

Next, we introduce the evaluation metrics that we use to compare our architectures. Again, this decision is guided by the choices made in prior work. Evaluation metrics determine how well the agent is able to perform its task. In general, RL task success/performance is primarily measured in terms of the average return obtained, as was discussed in Section 2.1. To standardize the comparison, D4RL proposes an evaluation scheme that normalizes the obtained scores as follows:

$$\texttt{normalized\_score} = 100 * \frac{\texttt{score} - \texttt{random\_score}}{\texttt{expert\_score} - \texttt{random\_score}} \tag{3.1}$$

All scores correspond to the average return over 100 episodes. Here `score` is the score obtained by the agent to evaluate, `random_score` is the score obtained by a policy that selects actions uniformly at random, and `expert_score` reflects the score obtained by an expert policy. Hence, `normalized_scores` of 0 and 100 correspond to the average returns over 100 episodes of a random policy and an expert policy, respectively [Fu et al., 2020]. While for Gym-MuJoCo the expert policy is represented by an online SAC agent [Haarnoja et al., 2018], for offline Atari1M it is an online DQN agent [Mnih et al., 2015]. Additionally, prior work typically reports the mean or median of the normalized scores over four/five runs with different random seeds [Agarwal et al., 2020; Kumar et al., 2020]. We adopt the same evaluation procedure for this thesis with 3 random seeds due to computational constraints.

Normalized scores and learning curves. In Chapter 5 we always report the mean (± standard deviation) and normalized scores (in %) over all random seeds at the last training step and at the midpoint of training. For Atari, the last step and mid-point correspond to 2.5M steps and 1.25M steps, respectively. For Gym, they correspond 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 Bibliothek Your knowledge hub WIEN

500K steps and 250K steps. To illustrate the learning progress graphically, we also show the learning curves (raw scores at every time step) averaged across the seeds.

Interval estimates. However, due to the computational demand of RL training, it is common to report results over only a small number of runs (3 random seeds in our case). Thus, reproducibility and sound evaluation have become a major challenge in RL. To this end, Agarwal et al. [2021] recently proposed a statistically more robust approach to evaluate the performance of RL agents that accounts for the variability in reported results. In particular, the authors argue for interval estimates via stratified bootstrap confidence intervals (CI) [Efron, 1992] instead of point estimates, and the IQM instead of the mean or median. The interval estimates are computed over N independent runs (3 seeds) and M tasks (3 for Gym-MuJoCo, 4 for Atari) by re-sampling the runs with replacement independently for each task. The IQM only considers the middle 50% of the runs for computing the mean and provides a more robust estimate for the agent's performance. Thus, each sample uses $NM/2$ runs to compute the IQM.

Probability of improvement. In addition, Agarwal et al. [2021] propose to report the probability of improvement of an algorithm X over another algorithm Y. The probability of improvement is given by:

$$P(X > Y) = \frac{1}{M} \sum_{m=1}^{M} P(X_m > Y_m) \tag{3.2}$$

where M represents the number of tasks. $P(X_m > Y_m)$ is the probability that X outperforms Y on a particular task m and is computed using the Mann-Whitney U-statistic [Mann and Whitney, 1947], $P(X_m > Y_m) = \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} S(x_{m,i}, y_{m,j})$ with $S(x, y) = 1$ if $y < x$, $S(x, y) = \frac{1}{2}$ if $y = x$, and $S(x, y) = 0$ if $y > x$.

Therefore, in Chapter 5 we also report the interval estimate scores for IQM, mean and median (with 50K bootstrap samples) and probability of improvement (2K bootstrap samples) with a CI of 95%. The interval bounds are constructed using percentile bootstrap. To simplify the adoption of this new evaluation procedure for RL, Agarwal et al. [2021] released an open-source library called `rliable`[1] that we use in this work.

3.3.3 Baselines

Furthermore, the performance of our offline RL agents extended with self-supervised methods and data augmentations has to be compared against baseline methods to validate our experiments. The most natural baseline we select is the pure CQL agent, without any extensions. Hence, this choice allows for a comprehensive evaluation of the effectiveness of the proposed modifications.

In addition, we also compare against the simplest baseline, BC [Pomerleau, 1988]. BC is the standard baseline method for the offline setup, and most prior work compares against it [Wu et al., 2019; Kumar et al., 2020; Yu et al., 2021].

[1] Available at `https://github.com/google-research/rliable`

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 Your knowledge hub WIEN

Furthermore, for Atari, we compare against online/offline DQN and a random policy and against online/offline SAC and a random policy for Gym-MuJoCo tasks.

3.4 Implementation & Setup

Finally, we discuss the software and hardware setup for our subsequent experiments.

3.4.1 Software stack

We adhere to established software engineering guidelines for all our implementations and aim for easy reproducibility. To that effect, the project repository for this thesis is open-source and available on GitHub[2].

All RL algorithms are implemented in Python using the popular DL-framework PyTorch [Paszke et al., 2019b]. Our implementations are built on top of `d3rlpy` [Seno and Imai, 2021], a popular library for offline RL that provides high-quality implementations for a variety of offline RL algorithms, amongst them CQL. To obtain the datasets, we leverage the benchmarking frameworks D4RL[3] and RL Unplugged[4], as described in Section 3.3.1. To facilitate the usage of Atari datasets provided by RL Unplugged, we take advantage of an additional open-source library, `d4rl-atari`[5] that wraps RL Unplugged and provides a simple interface like D4RL. The library was created by the same author as `d3rlpy`. Thus, `d3rlpy` provides an easy-to-use and C-optimized interface for dataset interaction that we leverage as well.

During model training and evaluation all logs are written to TensorBoard[6], an easy-to-use browser-based visualization toolkit for tracking the learning progress of ML algorithms [Abadi et al., 2016]. TensorBoard can be executed using a commandline tool and visualized in the browser. Additionally, we also make use of Weights & Biases (`wandb`), another web-based tool for experiment tracking [Biewald, 2020].

We list all other dependencies and the exact version numbers in Table 3.3.

3.4.2 Hardware setup

In general, RL is among the most resource-intensive areas of ML and state-of-the-art RL algorithms often rely on excessive amounts of compute [Vinyals et al., 2019; Berner et al., 2019]. This characteristic may inhibit the progress of the entire field, as it favours teams or individuals with sufficient computing resources. Nevertheless, offline RL relaxes this prerequisite a bit, as was discussed in Section 2.2.5.

[2] Available at: `https://github.com/thomasschmied/offline_rl`
[3] D4RL is available at: `https://github.com/rail-berkeley/d4rl`
[4] RL Unplugged is available at: `https://github.com/deepmind/deepmind-research/tree/master/rl_unplugged`
[5] d4rl-atari is available at `https://github.com/takuseno/d4rl-atari`
[6] TensorBoard is open-source and available at: `https://github.com/tensorflow/tensorboard`

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 Your knowledge hub WIEN

Tool	Version	Description
Python [Van Rossum and Drake, 2009]	3.9.5	programming language
NumPy [Harris et al., 2020]	1.20.3	numerical computations
Pandas [McKinney et al., 2010]	1.2.4	post-processing of results
Seaborn [Waskom, 2021]	0.11.2	visualization of results
PyTorch [Paszke et al., 2019a]	1.8.1	neural network architectures & automatic differentiation
OpenAI Gym [Brockman et al., 2016]	0.18.3	interface for Atari and Gym-MuJoCo tasks
d3rlpy [Seno and Imai, 2021]	0.91	offline RL algorithms
tensorboardX	2.2	experiment tracking
wandb [Biewald, 2020]	0.12.4	experiment tracking
tqdm	4.61.0	progress bars
rliable [Agarwal et al., 2021]	1.0	reliable RL evaluation

Table 3.3: Tools used in this work.

For our subsequent experiments, we use a server equipped with four NVIDIA RTX 2080ti kindly provided by our university. As our agents do not require large amounts of memory, we always ran two experiments at a time on the same Graphics Processing Unit (GPU).

3.5 Summary

To summarize, we employ CQL [Kumar et al., 2020] as our base offline RL algorithm. We use Curl [Laskin et al., 2020b], SPR [Schwarzer et al., 2021a] and a modified version of SGI [Schwarzer et al., 2021b] as self-supervised methods. Moreover, following previous work on data augmentations in RL [Laskin et al., 2020a; Yarats et al., 2021b], we select random-cropping, random-shifting and random cutout as augmentations for discrete control tasks, and random amplitude scaling and Gaussian noise as data augmentations for continuous control tasks.

In the subsequent chapter, we evaluate our offline RL algorithms on both image-based as well as state-based tasks using offline Atari and Gym-MuJoCo datasets provided by RL Unplugged and D4RL, respectively. To this end, we select four Atari games (Pong, Breakout, Seaquest, QBert) and three Gym-MuJoCo tasks (Halfcheetah, Hopper, Walker-2d) for our experiments. Furthermore, we compare our implementations against five estaablished baselines, namely CQL without any extensions, BC, a random policy, online/offline DQN (Atari) and online/offline SAC (Gym-MuJoCo). All agents are evaluated based on average return, normalized scores w.r.t. to random and expert agents, interval estimates for IQM, mean and median, and probability of improvement as introduced in Section 3.3.2.

An overview of our methodology workflow was given in Figure 3.1 and Algorithm 3.1.

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 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.

CHAPTER 4

Experiments

In this section we define what experiments we conduct in this work and how we conduct them according to the methodology we established in the previous chapter. First, we briefly introduce all the subsequent experiments. Then each experiment category is described individually.

4.1 Overview & Experiment phases

To begin with, we give a general overview in chronological order of the experiments that we conduct. We structure this chapter and the subsequent chapter in four experiment phases: **(1)** Baselines, **(2)** Data augmentations, **(3)** Self-supervised methods, and **(4)** Online fine-tuning for offline RL. All architectures are applied to the image-based benchmark, Atari, as well as the state-based benchmark, Gym-MuJoCo, and trained for 2.5 million and 500 thousand gradient steps, respectively. Depending on the task at hand, different architectural choices are suitable. For example, while for an image-based task a CNN-encoder might be required, a fully connected architecture might suffice for state-based tasks. To ensure comparability of all methods, we retain the same choices for all tasks in the same task category across all experiment categories. We discuss our choices in the subsequent sections.

The first set of experiments aims to apply the selected baseline methods, online/offline DQN, online/offline SAC, CQL and BC to the benchmark tasks. On the one hand, these preliminary experiments result in the baseline performance scores to compare the other methods against. On the other hand, they serve as a general validation process of our experiment setup. As CQL forms the basis of all subsequent architectures, it is crucial that the implementation works as expected. Therefore, achieved performance measures should be comparable to the ones in the original publications. The other three experiment categories represent the core contributions of this thesis. They investigate how self-supervised methods, data augmentations and online fine-tuning can improve

the learning behaviour of offline RL agents. First, we evaluate our selection of data augmentations with CQL. Then, CQL is extended with the selected self-supervised architectures Curl, SPR and SGI. Finally, we investigate how CQL can be used effectively with offline pre-training and online fine-tuning.

4.2 General

First, we describe hyperparameters and architectural choices that remain constant throughout the different experiment phases.

4.2.1 Hyperparameters

General parameters. Table 4.1 gives an overview of all parameters that remain constant for both continuous control tasks (Gym-MuJoCo) and discrete control tasks (Atari). All datasets used in this work consist of 1 million samples, as was discussed in Chapter 3. As Fu et al. [2020], on Gym-MuJoCo we train all our algorithms for 500 thousand gradient steps. On Atari, we train all algorithms for 2.5 million steps, the same number of gradient steps as used by Agarwal et al. [2020] and Kumar et al. [2020]. Every epoch lasts for 10 thousand steps. For Gym-MuJoCo tasks, we evaluate the current agent on the actual environment after every epoch. For computational reasons, we evaluate the agent after every second epoch on Atari games. Therefore, each learning curve in Chapter 5 contains 50 and 125 data points for Gym-MuJoCo tasks and Atari games, respectively. Furthermore, we run each experiment with three random seeds. The first seed was set manually, the other two seeds were drawn at random from a discrete uniform distribution with lower bound of 0 and upper bound of 5000. All results we report in Chapter 5 are the aggregate scores over all seeds.

Training parameters. In all our experiments, we use a batch size of 256 for Gym-MuJoCo tasks and 32 for Atari games, just as Kumar et al. [2020]. The batch size of 32 is considered standard and widely used for the Atari benchmark [Mnih et al., 2015; Hessel et al., 2018; Agarwal et al., 2020]. However, this design decision might not be optimal for offline RL, and increasing the batch size could speed up training considerably by better taking advantage of the entire GPU. This issue was previously also discussed by Agarwal et al. [2020] in the context of offline RL. Indeed, prior work has shown that for online RL agents, increasing the batch size was beneficial on Atari [Stooke and Abbeel, 2018]. We note that during our experiments the GPU utilization for Atari was at 50% top, and trial experiments with a higher batch size indeed showed significant speed-ups. To ensure comparability, however, we stick with the standard batch size of 32.

Throughout most of our experiments we use a discount factor γ of 0.99, as used in Equation 2.1 (if an experiment uses a different value, this is indicated in the respective section). During evaluation, Gym-MuJoCo agents use an ϵ of 0 whereas Atari agents use an ϵ of 0.001 for ϵ-greedy exploration. For Atari this implies, that 0.1% of the time a random action is performed to explore the environment, instead of the greedy optimal

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 Your knowledge hub WIEN

action. The dilemma of exploration vs. exploitation was discussed in section 2.2.5. In addition, to reduce variance, each evaluation step consist of 5 evaluation trials for both Atari and Gym-MuJoCo tasks. The reported return is the mean return over these 5 trials.

Atari pre-processing. For Atari games, the observations are provided in image-space and require additional pre-processing steps. We perform the same pre-processing steps as prior work in online RL and offline RL for Atari [Mnih et al., 2015; Hessel et al., 2018; Agarwal et al., 2020; Kumar et al., 2020]. First, each frame is down-sampled to a resolution of 84×84 pixels. Each observation consists of 4 stacked frames. Stacking frames is useful for the agent to understand temporal relations. To illustrate, in Breakout, frame-stacking allows the agent to observe what direction the ball is moving in. By only perceiving a single frame, however, this would be impossible to infer. Thus, each action is also repeated 4 times. Furthermore, frames are grey-scaled and pixel-scaled. In addition, all rewards obtained by the agent during training are clipped to the range $[-1, 1]$.

Parameter	**Continuous control**	**Discrete control**
Dataset size	1M	1M
Gradient steps	500K	2.5M
# steps per epoch	10K	10K
Evaluation frequency (in epochs)	1	2
Seeds	42, 2509, 3333	42, 2509, 3333
Batch size	256	32
Discount factor (γ)	0.99	0.99
Evaluation (ϵ)	0.0	0.001
Evaluation trials	5	5
Observations down-sampling	-	84×84
Frames stacked	-	4
Frame skip (Action repetitions)	-	4
Grey-scaling	-	True
Pixel-scaling	-	True
Reward clipping	-	$[-1, 1]$
(Online) replay memory size	1M	1M
(Online) min replay size for sampling	1K	20K
(Online) sampling scheme	Uniform	Uniform
(Online) update interval	1	1

Table 4.1: General hyperparameters for continuous control tasks (Gym-MuJoCo) and discrete control tasks (Atari) that remain constant throughout all experiments.

Online training. For our online "expert" baselines (as described in Section 3.3.2) we fix the replay memory size at 1 million observations. The training only starts once 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 Bibliothek Your knowledge hub WIEN

replay memory is populated sufficiently. For Gym-MuJoCo tasks, we set the minimum replay memory size to 1 thousand observations. Similarly, for Atari we set it to 20 thousand observations, just as Agarwal et al. [2020]. Observations are drawn according to a uniform sampling scheme, i.e., we do not make use of more involved replay buffers such as Prioritized Experience Replay to ensure a fair comparison [Schaul et al., 2016]. For Atari, it is, however, not obvious how "long" online agents should be trained. All offline agents have access to 1M observations and perform 2.5M gradient steps. As discussed, for online Atari, it is common practice to stack four consecutive frames to a single observation and to set the update interval to four. To ensure a fair comparison with online agents, we consider a few options. Fist, if the training lasts for 2.5M environment steps, the online agent only observes 625K observations and performs the same number of gradient steps. To collect the same number of observations as offline agents (1M), the online agent requires 4M environment steps, but performs only 1M gradient steps. Similarly, to perform the same number of gradient steps as offline agents (2.5M), the online agent requires 10M environment steps. However, in this case it would also observe 2.5M observations, more than double the amount as the offline agent. Another alternative is to set the update interval to 1. Then the agent performs 2.5M gradient steps and collects 625K observations. To determine the best option, we conducted a short ablation study, which we present in Appendix A.1. Overall, we decided to set the update interval of online agents to 1. This choice gives the most balanced scores and simplifies our experimental setup for offline pre-training and online fine-tuning.

4.2.2 Dataset statistics

According to Fu et al. [2020], the Gym-MuJoCo datasets used in this work were generated by training an online SAC agent and collecting the first 1M observations. The Atari datasets used in this work were extracted from the Atari collection released by RL Unplugged [Gülçehre et al., 2020]. The Atari observations were generated by an online DQN agent over 50 million transitions [Gülçehre et al., 2020]. The "medium" datasets we use in this work contain 1 million transitions collected between 9 and 10 million[1]. In Table 4.2, we show the aggregate statistics for all Gym-MuJoCo tasks as well as for all Atari games. We note that all rewards in the Atari datasets provided by [Gülçehre et al., 2020] are clipped to range $[-1, 1]$. Thus, the aggregate statistics shown in Table 4.2 are significantly lower than the raw scores observed in our experiments.

In Table 4.3, we list the shapes of input observations as well as the number of actions available to the agent in each of the environments. For continuous control tasks, the input is a one-dimensional input vector that describes the robot's current state. In contrast, for discrete control tasks, each observation consists of a stack of four 84×84 image frames. The number of actions varies between 3 and 18 across all environments.

[1] As provided by `d4rl-atari`: `https://github.com/takuseno/d4rl-atari`

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 Your knowledge hub WIEN

Dataset	Mean ± Std	Min	Q_{25}	Q_{50}	Q_{75}	Max
			Gym-MuJoCo			
halfcheetah-medium-v2	4765.45 ± 355.43	-309.77	4689.34	4802.45	4904.05	5303.83
hopper-medium-v2	1422.05 ± 379.03	315.87	1231.71	1464.30	1621.47	3218.04
walker2d-medium-v2	2849.94 ± 1094.48	-6.61	2073.12	3495.89	3693.40	4226.94
			Atari			
Breakout	42.67 ± 15.89	1.00	31.00	41.00	53.00	91.00
Pong	18.19 ± 1.87	10.00	17.00	18.00	20.00	21.00
Seaquest	42.9 ± 14.41	1.00	33.00	43.00	53.00	101.00
QBert	132.06 ± 74.04	0.00	72.00	126.00	176.00	510.00

Table 4.2: Datasets statistics for Gym-MuJoCo and Atari tasks.

Dataset	Observation shapes	Action shapes/# of actions
	Gym-MuJoCo	
halfcheetah-medium-v2	(17,)	(6,)
hopper-medium-v2	(11,)	(3,)
walker2d-medium-v2	(17)	(6,)
	Atari	
Breakout	(4, 84, 84)	4
Pong	(4, 84, 84)	6
Seaquest	(4, 84, 84)	18
QBert	(4, 84, 84)	6

Table 4.3: Observation and action shapes for Gym-MuJoCo and Atari tasks.

4.2.3 Network architectures

In Figure 4.1, we show the neural network encoder backbones we employ in this work. The graphs were generated with Netron[2], an open-source tool for neural network visualization. The encoder backbones take input observations and return latent representations of them, that are further used by the policy, Q-function heads, and self-supervised methods.

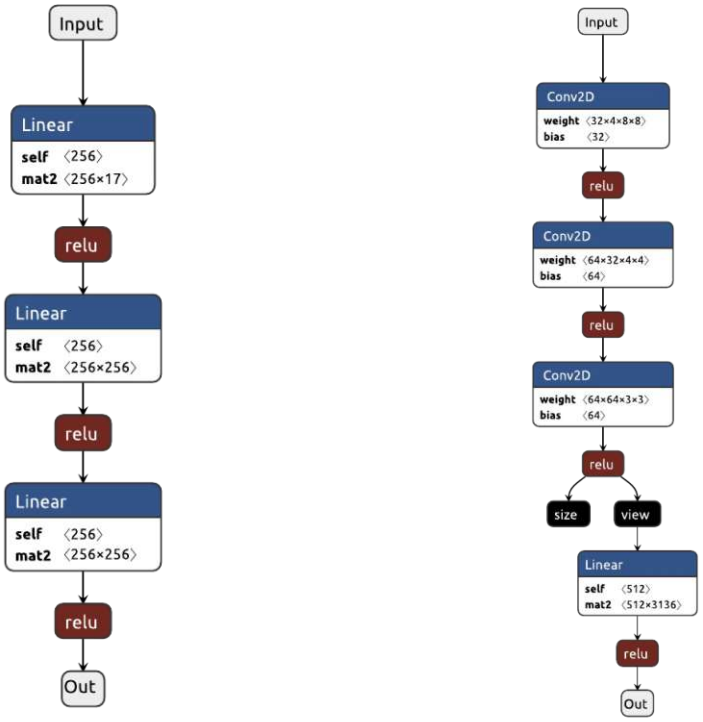
Continuous control. For most of our Gym-MuJoCo experiments, we employ a stack of three linear layers with hidden size 256 as our encoder backbone. The exception are experiments with online/offline SAC, for which we use a stack of two linear layers with the same hidden size, as was used by Fu et al. [2020]. The dimension of the input vector, as well as the output dimension, depend on the particular environment (as listed in Table 4.3).

Discrete control. For Atari environments we use the standard CNN encoder backbone as was used in prior work [Mnih et al., 2015; Hessel et al., 2018; Agarwal et al., 2020;

[2]Available at https://github.com/lutzroeder/netron

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 Your knowledge hub WIEN

Kumar et al., 2020. This network architecture consists of a stack of three convolutional layers followed by a linear output layer. The channel sizes, filter sizes and strides are set to $(32, 8 \times 8, 4)$, $(64, 4 \times 4, 2)$, and $(64, 3 \times 3, 1)$, for the three convolutional layers, respectively. Finally, the hidden size in the linear output layer is set to 512. As for Gym-MuJoCo tasks, the exact input and output shapes depend on the respective environment. The network backbones shown in Figure 4.1 depict the default neural network encoder backbones. For different algorithms the architectural choices might differ slightly, but this will be discussed in the respective sections.



(a) Gym encoder **(b)** Atari encoder

Figure 4.1: The neural network encoder backbones for continuous and discrete control tasks.

4.3 Baselines

Next, we describe our baseline experiments. This includes our experiments to acquire "expert" and "random" scores with online/offline DQN, online/offline SAC, and a random agent, as well as our experiments with CQL, and BC. These experiments are fundamental, as they not only aim to obtain baseline scores, but also serve as a validation process for the implemented source code. For our implementations many design decisions, including the neural network architectures, hyperparameter choices, and pre-processing steps are

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 Your knowledge hub WIEN

kept constant, as listed in Section 4.2.1. There are, however, differences between different algorithms. In this section, we discuss the parameter choices for our baseline experiments.

4.3.1 Online & offline SAC

For our online and offline experiments with SAC [Haarnoja et al., 2018], we use the same hyperparameters as used by Fu et al. [2020] and Kumar et al. [2020][3]. We list all hyperparameters in Table 4.4. Notably, we also make use of automatic entropy regularization (denoted as temperature) and the twin Q-function trick (# of critics). In contrast to other algorithms, online/offline SAC only uses a stack of two linear layers in the encoder instead of three layers.

Parameter	Value
Optimizer	Adam
Policy learning rate	3e−4
Q-function learning rate	3e−4
Temperature learning rate	3e−4
Initial temperature	1.0
Target smoothing coefficient (τ)	0.005
# of critics	2
# of hidden layers	2
Hidden units	256
Non-linearity	ReLU

Table 4.4: SAC hyperparameters for continuous control tasks (Gym-MuJoCo).

4.3.2 Online & offline DQN

For our implementation of online and offline DQN, we use the same hyperparameters as Agarwal et al. [2020], except for the learning rate, for which we use a slightly higher value. Agarwal et al. [2020] use similar hyperparameters for online/offline DQN as Gülçehre et al. [2020]. For the CNN encoder-backbone, we use the standard network architecture shown in Figure 4.1.

4.3.3 Random agent

As described in Section 3.3.2 we also require the `random_score` for the score normalization. Therefore, we use the random policy provided by `d3rlpy` and evaluate it on all environments. For discrete control tasks, the random policy simply selects an action uniformly at random from the action space, i.e., $a \sim \mathcal{U}(0, \text{num\_actions})$. Similarly, for continuous control tasks, the random policy selects n action values (Table 4.3), i.e., the action is an n-dimensional vector. Each action value is drawn from a continuous uniform

[3]Both works are based on this implementation: `https://github.com/vitchyr/rlkit`

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 Your knowledge hub WIEN

Parameter	Value
Optimizer	Adam
Learning rate	6.25e−5
Target network update interval	2000
(Online) Training ϵ	0.01
(Online) ϵ-decay schedule	250K

Table 4.5: DQN hyperparameters for discrete control tasks (Atari).

distribution, $a \sim \mathcal{U}(\min, \max)$, where min and max refer to the minimum and maximum action for the respective environment (e.g., $[-1, 1]$). Thus, this agent requires no training. The action shapes/number of actions for each environment are shown in Table 4.3.

4.3.4 Behaviour cloning (BC)

For BC, we employ the standard hyperparameters and network architectures as listed in Table 4.1 and Figure 4.1, respectively. All other hyperparameters are listed in Table 4.6. The two major differences to other algorithms are the learning rate and that BC does not employ discounting ($\gamma = 1$).

Parameter	Value
Optimizer	Adam
Learning rate	1e−3
Discount factor (γ)	1

Table 4.6: Behaviour Cloning hyperparameters for continuous control tasks (Gym-MuJoCo) and discrete control tasks (Atari).

4.3.5 CQL

In Table 4.7, we list the hyperparameters we use for our CQL experiments. CQL forms the basis of all subsequent experiments with self-supervised methods and data augmentations. For continuous control tasks CQL is built on top of SAC [Haarnoja et al., 2018]. Similarly, for discrete control tasks, CQL is built on top of QR-DQN, a distributional version of DQN with quantile regression [Dabney et al., 2018]. Thus, CQL requires different hyperparameter choices depending on the task at hand, as shown in Table 4.7.

For Atari games, we use the same hyperparameters as Kumar et al. [2020]. Also, for Gym-MuJoCo tasks, we use the same hyperparameters as Kumar et al. [2020], except for the conservative weight α. The conservative weight determines how strongly the Q-function is penalized for overestimating state-action pairs. Kumar et al. [2020] propose to keep α fixed for discrete control tasks. For continuous control tasks, however, they introduced two versions, one with a fixed α and another version for which α is tuned

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 Your knowledge hub WIEN

automatically during the training process. In their experiments, the latter outperformed the former slightly. However, in our preliminary experiments with CQL, we did not see a big difference in performance, despite high additional computational cost for the version with automatic tuning of α. Consequently, we discard the version with automatic tuning in favour of the version with a fixed α. For Gym-MuJoCo we thus set α to 5.0, as proposed by Kumar et al. [2020] for the small-data regime. For Atari games, α is set to 4.0 by default.

Parameter	Continuous control	Discrete control
Optimizer	Adam	Adam
Adam (ϵ)	1e−8	0.01/32
Policy learning rate	3e−5	-
Q-function learning rate	3e−4	5e-5
Temperature learning rate	3e−5	-
Fixed α	True	True
Initial α	5.0	4.0
Initial temperature	1.0	-
Target smoothing coefficient (τ)	0.005	-
# of action samples	10	-
# of critics	2	1
Q-function type	Mean	Quantile regression
# of Q-heads	-	200
Target network update interval	-	2000

Table 4.7: CQL hyperparameters for continuous control tasks (Gym-MuJoCo) and discrete control tasks (Atari).

4.4 Data augmentations

Next, we describe our experiments on continuous control tasks and discrete control tasks with the augmentations amplitude scaling, Gaussian noise, random cropping, random shifting and random cutout.

4.4.1 Discrete control: Atari

In Table 4.8, we show the parameters we use for our experiments with data augmentations on the Atari benchmark. There are a few common parameters across all augmentations that determine how the data augmentations are applied to the input observations.

Same augmentations. First of all, we have the option to apply either the same or different augmentations (e.g., the same random crop or a different random crop) to current

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 Your knowledge hub WIEN

observations s and next observations s'. By default, we use the same augmentations for the current and next observations, s and s' respectively (denoted "same" in Chapter 5).

Augmentation probability. Furthermore, we can apply augmentations with a certain augmentation probability, p_{aug}. If augmentations are used, we always augment the input observations by default (i.e., $p_{aug} = 1.0$) but we also experimented with different values for the augmentation probability (e.g., $p_{aug} = 0.5$), as will be discussed in Chapter 5.

Multiple target augmentations. As described in Section 2.4.2, the Q-estimates can be computed over K target augmentations as proposed by Yarats et al. [2021b] for the DrQ architecture. In our implementation, we set K to 1 by default, which is equivalent to the augmentation strategy employed by RAD [Laskin et al., 2020a]. However, we also experimented with $K = 2$ target augmentations.

Augment conservative. As we use CQL as the base offline RL algorithm, we have to decide whether the conservative loss should be computed from augmented observations or the original observations. By default, we use the augmented observations for the computation of the conservative loss.

Random crop. For the random crop augmentation, we first extract an 80×80 crop from the original input observation. Then we pad the resulting crop by two pixels using replication of the input boundary to the original dimension of 84×84. For Atari, each input observation is a stack of four consecutive frames. Thus, in line with prior work, we apply all augmentations consistently across the stack, but randomly across the batch [Laskin et al., 2020a; Yarats et al., 2021b].

Random shift. Random shift composes the same two augmentation operations as our version of random crop, but in reverse order. First, random shift pads the input observation by 4 pixels by replicating the image boundary. Then a random crop of dimension 84×84 is performed. The same parameters are used by Yarats et al. [2021b]. Again, the augmentations are applied consistently across the stack but at random across the batch.

Cutout. Finally, cutout constructs a randomly placed and randomly sized rectangle of maximum dimension $\text{cut}_{min} \times \text{cut}_{max}$ from the input image and sets all pixel values within that rectangle to zero. Thus, applying cutout results in an image with a black cut-out region (black $= 0$, white $= 255$ in pixel space).

4.4.2 Continuous control: Gym-MuJoCo

Next, we describe the data augmentations that we employ for our experiments on Gym-MuJoCo. The respective parameters are listed in Table 4.9. We omit the common parameters (p_{aug}, K target augmentations, etc.), as they remain constant and were already described in the previous section.

Amplitude scale. For amplitude scaling, the new observation is constructed according to $s_{new} = s * \mathcal{U}(\text{scale}_{min}, \text{scale}_{max})$. By default, we choose 0.8 and 1.2 for scale_{min} and scale_{max}, respectively. These are the same values as used by Laskin et al. [2020a].

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 Your knowledge hub WIEN

Parameter	Random crop	Random shift	Cutout
Same augmentations (s, s')	True		
Augmentation prob (p_{aug})	1.0		
K target augmentations	1		
Augment conservative	True		
Crop size	80	84	-
Padding	2	4	-
Minimum cut (cut_{min})	-	-	10
Maximum cut (cut_{max})	-	-	30

Table 4.8: Data augmentations for discrete control tasks (Atari).

Gaussian noise. Similarly, for Gaussian noise the augmented observation is constructed according to $s_{new} = s + \mathcal{N}(\mu, \sigma)$, with $\mu = 0$ and $\sigma = 1$, respectively. Again, we use the same parameters as Laskin et al. [2020a]. We also experimented with different values for σ, as will be presented in Chapter 5.

Parameter	Amplitude scale	Gaussian noise
Minimum scale (scale_{min})	0.8	-
Maximum scale (scale_{max})	1.2	-
Noise mean (μ)	-	0
Noise std (σ)	-	1

Table 4.9: Data augmentations for continuous control tasks (Gym-MuJoCo).

4.5 Self-supervised methods

In this section, we describe our experiments with the self-supervised methods Curl, SPR and SGI.

4.5.1 Offline Curl

For our implementation of offline Curl, we keep the same parameters as for the base CQL agent. Curl has a few additional hyperparameters that are listed in Table 4.10 for continuous and discrete control tasks.

Augmentations. To compute the contrastive loss (Equation 2.18 in Section 2.3.2), Curl requires to construct the queries and keys (q, and k) by encoding the augmented input observations s. For Atari, we use random cropping as the contrastive augmentation, as proposed by Laskin et al. [2020b]. In the original publication, Curl was only applied to vision-based tasks. We extend our implementation of Curl to environments with state-

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 Your knowledge hub WIEN

based inputs (Gym-MuJoCo). To this end, we use amplitude scaling as the contrastive augmentation for all Gym-MuJoCo tasks. Laskin et al. [2020b] proposed two ways of using augmentations in Curl. Either augmentations are employed only for the computation of the contrastive loss, or augmentations are applied globally to the observations that are fed into the regular the RL objective. By default, Laskin et al. [2020b] used the latter variant. We follow their methodology and use global augmentation variant by default with amplitude scaling and random cropping for continuous control tasks and discrete control tasks, respectively. We also experiment with different data augmentations (e.g., Gaussian noise instead of amplitude scaling), as we discuss in Section 5.3.

If the global augmentation variant is used, we can either apply different random augmentations to the current observation s and next observations s' or apply the same augmentations (e.g., the same random crops) to s and s'. By default, we always apply the same augmentations to s and s' but we also experiment with the other variant. Furthermore, as described in Section 2.4.2, we can compute the Q-estimates over K target augmentations as used in DrQ [Yarats et al., 2021b]. In our implementation, we set K to 1 by default.

Parameter	**Continuous control**	**Discrete control**
Contrastive augmentation	Amplitude scale	Random crop
Global augmentation	Amplitude scale	Random crop
Same augmentations (s, s')	True	
Augmentation prob (p_{aug})	1.0	
K target augmentations	1	
Encoder linear dim	256	
Contrastive latent dim	256	
EMA (m)	0.01	
EMA update frequency	1	
Renormalize	False	
Layer norm	True	
Batch norm	False	

Table 4.10: Offline Curl hyperparameters for continuous control tasks (Gym-MuJoCo) and discrete control tasks (Atari).

Curl components. To encode the augmented observations, we use the standard encoder backbones shown in Figure 4.1. For discrete control tasks, we reduce the dimension of the last linear layer from 512 to 256 as in the original publication of Curl. As shown in Table 4.7, we employ two critic networks for the continuous control base CQL agent. Thus, to ensure that both critic encoders are involved in the computation of the contrastive loss, we encode the input observation with both encoder networks and average their outputs. The latent dimension of the encoded queries and keys, and thus the bilinear weight matrix W (Equation 2.18) are of dimension 256 for both discrete and continuous

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 Your knowledge hub WIEN

control tasks. As described in Section 2.3.2 and depicted in Figure 2.6, the queries are constructed by the online encoder, while the keys are constructed by the momentum encoder. The weights of the momentum encoder are the EMA (Equation 2.19) of the online encoder ($m = 0.01$) and synced after every update step.

Normalization. Laskin et al. [2020b] employ LayerNorm [Ba et al., 2016] after the last layer of the encoder. By default, we apply the same normalization. However, we also experiment with a variant that replaces LayerNorm with BatchNorm [Ioffe and Szegedy, 2015] and a variant that normalizes the outputs of the encoder to lie in range $[0, 1]$ (re-normalize) as done by Schrittwieser et al. [2020] and Schwarzer et al. [2021a].

4.5.2 Offline SPR

Next, we describe our architectural and hyperparameter choices for offline SPR. All parameters are shown in Table 4.11. The overall architecture of SPR was examined in Section 2.3.2 and is depicted in Figure 2.7. The underlying idea of SPR is to predict future states in latent space via an explicit transition model, and to learn more effective representations as a result.

Augmentations. SPR can be used both with and without data augmentations. By default, Schwarzer et al. [2021a] employ random shifts as data augmentations, and we adopt the same strategy. However, SPR was only applied for image-based tasks. Therefore, we again extend our implementation of offline SPR to state-based tasks (Gym-MuJoCo) and use amplitude scaling as the respective data augmentation.

SPR components. Just as Schwarzer et al. [2021a], we use a prediction depth of 5 for the transition model and an SPR loss weight λ of 2 for both discrete and continuous tasks. For discrete control tasks, we employ the same convolutional transition model as Schwarzer et al. [2021a] that obtains the $64 \times 7 \times 7$ output from the last convolutional encoder layer (Figure 4.1). It comprises two $64 \times 3 \times 3$ layers with padding of 1, BatchNorm after the first convolutional layer, and ReLU activations after each layer. The transition model is action conditioned, and thus a one-hot vector that represents the action is appended to each location in the input Schwarzer et al. [2021a]. The last linear layer of the encoder backbone (with hidden dimension of 256) is used as the projection layer. The projection layer receives the flattened output of the transition model ($64 \times 7 \times 7 = 3136$) and projects down to dimension of 256. Furthermore, the SPR prediction head operates on the output of the projection layer and preserves the dimensions.

As SPR was only intended for pixel-based inputs, we extend the use of the transition model to state-based environments. To encode the input observations, we again use the same linear encoder as for other continuous control tasks. As mentioned in 4.5.1, we employ two critic networks for continuous control tasks. Thus, to involve both networks in the computation of the SPR loss, we encode the observations with both encoder networks and average their predictions. Our linear transition model consists of a stack of two linear layers each with hidden size 256, ReLU activations after each layer, and BatchNorm after the first linear layer. The transition model is action conditioned, thus

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 Your knowledge hub WIEN

Parameter	Continuous control	Discrete control
Global augmentation	Amplitude scale	Random shift
Transition model type	Linear	Convolutional
Same augmentations (s, s')	True	
Augmentation prob (p_{aug})	1.0	
SPR steps/depth	5	
SPR loss weight (λ)	2	
Encoder linear dim	256	
Projection dim	256	
Prediction head dim	256	
EMA (m)	1	
EMA update frequency	1	
Re-normalize	True	
Layer norm	False	
Batch norm	False	
Updates per step	1	

Table 4.11: Offline SPR hyperparameters for continuous control tasks (Gym-MuJoCo) and discrete control tasks (Atari).

we simply append the action to the encoded input. Finally, the projection and prediction heads are two separate linear layers of the same dimension. Similar to Curl, SPR employs separate momentum networks for the transition model and projection head for which the parameters are the EMA (Equation 2.19) of the online transition model and projection head, respectively. Schwarzer et al. [2021a] proposed to use $m = 1$ in case augmentations are enabled and $m = 0.01$ if no augmentations are used. We adopt the same choice. The parameters of the momentum networks are synced after every update step.

Normalization. As mentioned in Section 4.5.1, Schwarzer et al. [2021a] proposed to re-normalize the outputs of the encoder and the outputs of the transition model to lie in range $[0, 1]$. By default, we use the same normalization strategy. However, we also experiment with LayerNorm and BatchNorm, similar to our experiments with Curl.

Update steps. Finally, Schwarzer et al. [2021a] use two update steps per batch. However, to ensure a fair comparison, we stick with using only a single update step per batch, as this would double the number of gradient steps. The results for our experiments are described in Chapter 5.

4.5.3 Offline SGI

In this section, we describe our experiment setup for SGI [Schwarzer et al., 2021b]. SGI is based on SPR, thus we retain the same parameter choices as shown in Table 4.11. This

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 Your knowledge hub WIEN

includes the data augmentation strategy, SPR components and normalization strategy. Due to computational constraints, we could only conduct experiments on Gym-MuJoCo.

Training phases. SGI is trained in two phases [Schwarzer et al., 2021b]. First, the representation is pre-trained in a self-supervised, reward-free manner. Then, the pre-trained representation is used for online fine-tuning. In contrast, we use the pre-trained representation for offline RL. For Gym-MuJoCo, we pre-train for five times the number of offline gradient steps, i.e., 2.5M steps. Schwarzer et al. [2021b], pre-trained the representations on larger datasets (6M). To ensure a fair comparison with other algorithms, however, we pre-train on the same dataset as used during offline RL (1M).

Three SSL objectives. SGI consists of three self-supervised components: SPR, Goal-conditioned RL (GCRL) and Inverse modelling (IM) [Schwarzer et al., 2021b]. During pre-training, only the three self-supervised objectives are employed, while during fine-tuning the RL and SPR objectives are optimized together. The experiments by Schwarzer et al. [2021b] showed that most of the performance gains stem from SPR and IM (Table 2 in the original publication). Therefore, for simplicity, we do not employ GCRL in this work. The goal of IM is to predict the action a_t from two consecutive observations s_t and s_{t+1}. In fact, SGI integrates IM into SPR by predicting a_{t+k} from the latent projections/predictions $\hat{y}_{t+k}$ and $\tilde{y}_{t+k}$ (as used in Equation 2.20 and Figure 2.7). The IM consists of two linear layers (hidden size 256) and a ReLU activation in between.

Learning rates. Schwarzer et al. [2021b] found it essential to reduce the learning rates of the encoder and SPR transition model by a factor of 100 and the learning rates of other pre-trained weights by a factor of 3. For Gym-MuJoCo we found it useful to only reduce the learning rate of encoder and transition-model by a factor of 3 instead of 100, due to the dynamics of actor and critic in CQL.

Parameter	**Continuous control**
Pre-training steps	2.5M
Fine-tuning steps	500K
Global augmentation	Amplitude scale
Inverse model type	Linear
Goal-conditioned RL	False
SPR steps/depth	5
SPR loss weight (λ_{SPR})	2
Inverse modelling weight (λ_{IM})	1
EMA (m)	0.01
EMA update frequency	1
Lr reduction factor (encoder, transition model)	3
Lr reduction factor (other pre-trained weights)	3

Table 4.12: Offline SGI hyperparameters for continuous control tasks (Gym-MuJoCo).

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 Your knowledge hub WIEN

4.6 Online fine-tuning for offline RL

Finally, we describe our experiments with offline pre-training and online fine-tuning. We list the respective hyperparameter choices in Table 4.13. For both Gym-MuJoCo and Atari, we first pre-train offline for 30% of the total number of gradient steps. This corresponds to 150K and 750K gradient steps, respectively. Afterwards, the agents switch to online training for 350K and 1.75M gradient steps. We choose this proportion, as our experiments in Section 5.1 show, that after roughly 30% of the total number of gradient steps, CQL achieves a level of performance that is similar to its final performance.

By default, we turn off the conservative loss after the offline RL phase, set the conservative weight α to zero and initialize a fresh replay buffer. The new observations gathered during online training come from the same policy as the one being trained, and the optimization does not suffer from action distributional shift. However, the sudden change of the loss function may change the learning dynamics. Thus, we also experiment with variants that do not turn off the conservative loss abruptly, but only gradually over time. In particular, we experiment with halving or decaying the conservative weight after n epochs or gradient steps, respectively. Also, we experiment with a variant that turns off the conservative loss n epochs into the online training and another variant that automatically tunes the conservative weight α of CQL as proposed by Kumar et al. [2020].

We keep all other CQL parameters for offline pre-training (e.g., learning rates, network architectures, evaluation ϵ) and online fine-tuning (e.g., exploration, update start epoch, etc.) constant, as presented in the previous sections (Tables 4.1, 4.7).

Parameter	**Continuous control**	**Discrete control**
Total gradient steps	500K	2.5M
Offline gradient steps	150K	750K
Online gradient steps	350K	1.75M
Default online RL parameters		
Use conservative loss	False	False
Conservative weight α	0.0	0.0
Buffer init size	0	0
Alternative online RL parameters		
Use conservative loss	True	True
Conservative weight α	5.0	4.0
Half α every (in epochs)	5	5
Decay α every (in steps)	1000	1000
α decay	0.99	0.99
Zero α after (in epochs)	5	5

Table 4.13: Hyperparameters for offline pre-training and online fine-tuning for continuous control tasks (Gym-MuJoCo) and discrete control tasks (Atari).

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 Your knowledge hub WIEN

CHAPTER 5

Results

In this chapter, we present the results we obtained in our experiments. This includes a detailed empirical analysis and a discussion of the insights we gained.

5.1 Baselines

First, we present the results we obtained with our selected baseline algorithms on continuous and discrete control tasks.

5.1.1 Continuous control: Gym-MuJoCo

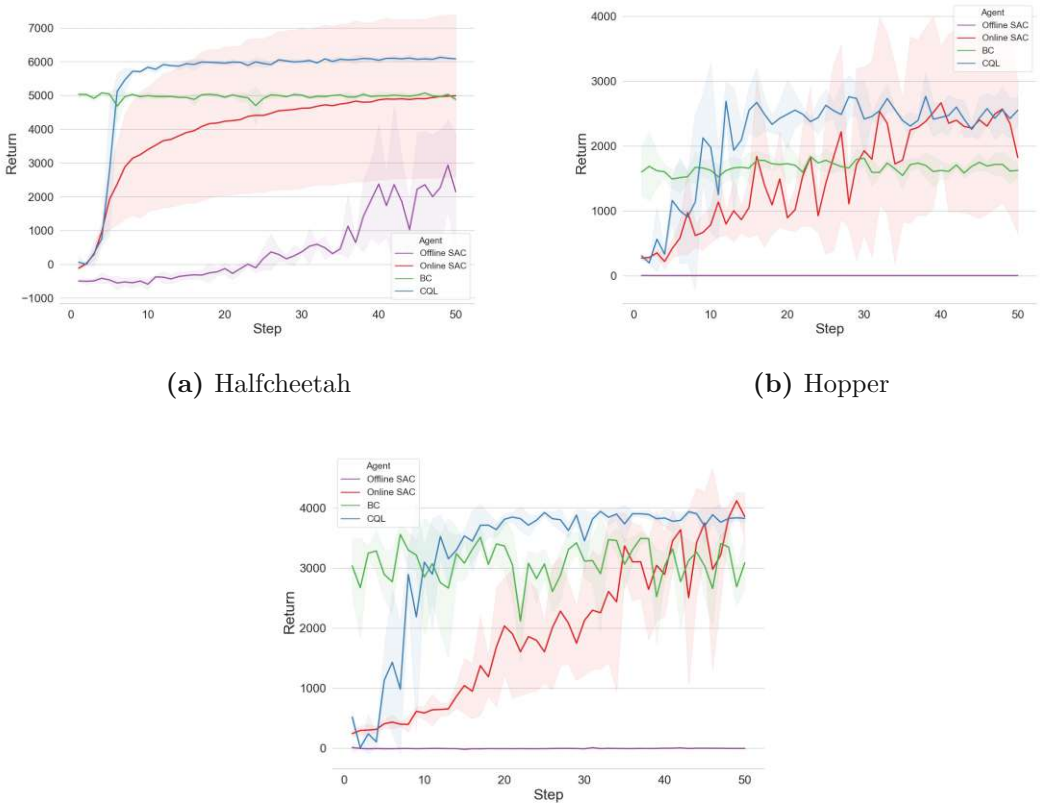
Learning curves. Figure 5.1 shows the learning curves for online/offline SAC, BC and CQL on continuous control tasks. Each curve represents the mean performance across the three seeds. The shaded area around the curve indicates the standard deviation across the three seeds.

We make a few interesting observations. First of all, we observe that the offline RL algorithm CQL (blue curve) outperforms all other baseline agents, including online SAC, on all three Gym-MuJoCo tasks. In contrast to online SAC, CQL achieves its final performance relatively quickly. After about 150K steps, the learning curve of CQL flattens out, indicating that the agent has leveraged all the information available in the dataset to extent. As environment interaction is not possible, CQL is not able to collect new information that could get rid of uncertainty and can therefore not improve further.

Online SAC, in comparison, requires much longer until it reaches its final performance in our experiments. The main reason for the slower initial learning progress is that, as an online RL algorithm, SAC initially performs actions at random. Consequently, the agent first has to discover high-reward behaviours by accident, and thus it takes much longer until the first learning progress materializes in the performance curve. The initial

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 Your knowledge hub WIEN

random behaviour of online SAC is also reflected in the high standard deviation across the three seeds, as indicated by the shaded red area in Figure 5.1. In contrast to CQL, however, the learning curve of online SAC points upwards towards the end of training and does not flatten out. If trained for longer, the performance of online SAC would arguably continue to improve. For CQL this would likely not be the case.



(a) Halfcheetah

(b) Hopper

(c) Walker2d

Figure 5.1: Learning curves for baseline agents on continuous control tasks, Halfcheetah, Hopper and Walker2d. CQL (blue) outperforms other baselines on all three tasks.

On the one hand, Offline RL is characterized by fast initial learning progress, but limited capacity for long-term improvement. Online RL, on the other hand, results in slow initial learning progress, but substantial capacity for long-term improvement. These contrary characteristics of online and offline RL suggest that combining offline with online training can be a promising research direction. It should be advantageous to start learning offline and to continue with online training once the offline dataset has been leveraged. For example, if we consider the learning curves for Walker2d in Figure 5.1, the agent could learn offline for 150K steps and then continue to learn online for the remaining 350K steps. Composing the red and blue curve would theoretically result in much better 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 Bibliothek Your knowledge hub WIEN

This initial observation further motivates the experiments we present in Section 5.4.

In contrast to online SAC, the offline version of SAC does not exhibit a similar learning progress over time. This is expected, as SAC is an online RL algorithm that relies on online interaction with the environment and thus is not able to learn effectively from a static dataset. The reason for the poor performance of offline SAC is action distributional shift in the critic, which was explained in Section 2.2.2. CQL, in contrast, mitigates the impact of action distributional shift by integrating the conservative penalty term (Section 2.2.3). This reflects the necessity for dedicated offline RL algorithms.

Finally, we observe that BC (in green) exhibits an almost horizontal learning curve. This behaviour is also expected, as BC simply imitates the behaviour available in the dataset.

Agent	halfcheetah-medium-v2	hopper-medium-v2	walker2d-medium-v2	Mean
		Final step (500K)		
Offline SAC	46.3	-0.4	-0.1	15.3
BC	98.0	89.0	79.9	89.0
CQL	120.7	140.2	99.2	120.0
		Midpoint (250K)		
Offline SAC	9.8	-0.5	-0.4	3.0
BC	110.8	124.9	191.2	142.3
CQL	132.7	184.7	244.3	187.3

Table 5.1: Normalized scores for baseline agents on continuous control tasks (in %).

Raw and normalized scores. In Table 5.1 we show the normalized scores (in %) at the final time step (500K) and at the midpoint of training (250K) of all baselines across the three Gym-MuJoCo tasks. Furthermore, we show the raw scores (mean ± standard deviation) of all experiments in Table A.2 in Appendix A.2. Machado et al. [2018a] advocate for reporting the RL agent's performance at multiple points in time during training to ensure a more robust comparison of architectures. This evaluation method was proposed in the context of Atari, but we think it is useful for RL in general. Therefore, we always additionally show the raw and normalized performance of all agents at the midpoint of training. The scores are normalized according to Equation 3.1, where the scores achieved by the random policy and online SAC represent the random and expert scores, respectively. Thus, the normalized scores for the random policy and online SAC correspond to 0% and 100% on all tasks, and we omit their scores.

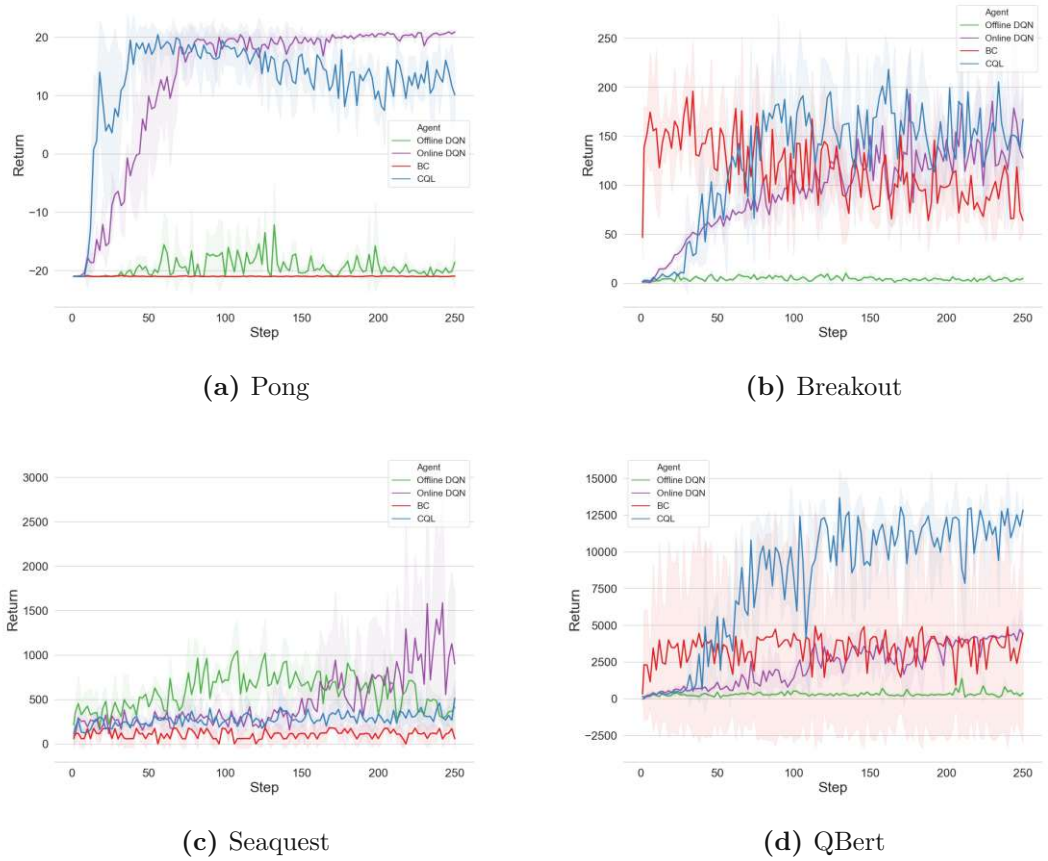
We note that the offline algorithms, CQL and BC, achieve on average 120% and 89% of the performance of the online SAC agent at the final step (500K). The offline version of SAC, however, performs poorly and is comparable to the random policy on two of the tasks, achieving only 15% of the performance of its online complement. Offline SAC only shows signs of learning for the Halfcheetah task. The scores at the midpoint of training, again, reflect the characteristics of offline and online RL established before, with CQL achieving 187.3% of the performance of the expert online algorithm. Similarly, BC achieves 142.3% of the expert's performance after 250K training 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 Bibliothek Your knowledge hub WIEN

To understand the impact of the conservative weight α on the learning performance of CQL, we conduct an ablation study with variable values for α. We present the results of these experiments in Section A.2.2 in Appendix A.2.

5.1.2 Discrete control: Atari

Next, we describe the results for our baseline experiments on the discrete control environments, Pong, Breakout, Seaquest and QBert.



(a) Pong **(b)** Breakout

(c) Seaquest **(d)** QBert

Figure 5.2: Learning curves for baseline agents on discrete control tasks, Pong, Breakout, Seaquest and QBert.

Learning curves. Figure 5.2 shows the learning curves for BC, CQL and online/offline DQN. Again, we observe fast initial learning progress of CQL (in blue) but a flattening of the learning curve as the training advances. In fact, on the game Pong, the performance of CQL even degrades towards the end of training. We did not observe this behaviour on Gym-MuJoCo environments. It is important to note that the learning curves for CQL shown in Table 5.2 were produced with $\alpha = 4.0$ for the conservative weight. We also

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.

experimented with other values for α, similar to our experiments with variable α on Gym-MuJoCo. We present the respective results for our ablation study in Section A.2.4 in Appendix A.2.

In contrast to the offline agent CQL, the online RL agent DQN again exhibits opposite characteristics: slower initial learning progress but continuous improvement throughout the course of training (except for QBert). This further motivates the necessity for agents that can learn both offline and online to combine the benefits of both paradigms.

The offline version of DQN performs poorly overall, but exhibits a somewhat stable learning progress on Seaquest. However, also on Seaquest the performance of offline DQN degrades towards the end of training, due to the impact of action distributional shift on the learned Q-function (as discussed in Section 2.2.2). Furthermore, BC also performs poorly on Pong and Seaquest, but does better on Breakout and QBert.

Raw and normalized scores. Again, we show the normalized scores and raw scores for all algorithms at the last step (2.5M) and at the midpoint of training (1.25M) in Table 5.2 and Table A.4 in Appendix A.2, respectively. Now, the online DQN agent represents the expert performance (100%) and we omit its scores.

Overall, CQL attains 139% the performance of online DQN at the final time step as shown in Table 5.2. However, the performance varies strongly between the four Atari games, and the percentual advantage of CQL stems mainly from high scores on QBert. On QBert, CQL achieves a score three times higher than that of online DQN. In contrast, for Seaquest it only performs half as well as the expert agent. Possibly, because Seaquest is considered harder than other games and requires longer term planning (e.g., returning to the surface before running out of oxygen) [Bellemare et al., 2013a]. The other offline algorithm, BC, achieves 37% of the expert's performance at the final time step.

Agent	Breakout	Pong	QBert	Seaquest	Mean
Final step (2.5M)					
Offline DQN	3.2	3.6	5.4	42.2	13.6
BC	49.8	-2.3	101.2	-0.2	37.1
CQL	130.7	73.6	297.5	54.2	139.0
Midpoint (1.25M)					
Offline DQN	3.1	16.8	2.0	783.7	201.4
BC	95.1	-2.6	111.0	97.8	75.3
CQL	129.8	96.7	461.6	134.8	205.8

Table 5.2: Normalized scores for baseline agents on discrete control tasks (in %).

However, if we consider the performance after only half the number of time steps (1.25M) the impression may be different. Under these circumstances, CQL achieves 205.8% of the performance of online DQN. Again, most of the percentual advantage stems from 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 Bibliothek Your knowledge hub WIEN

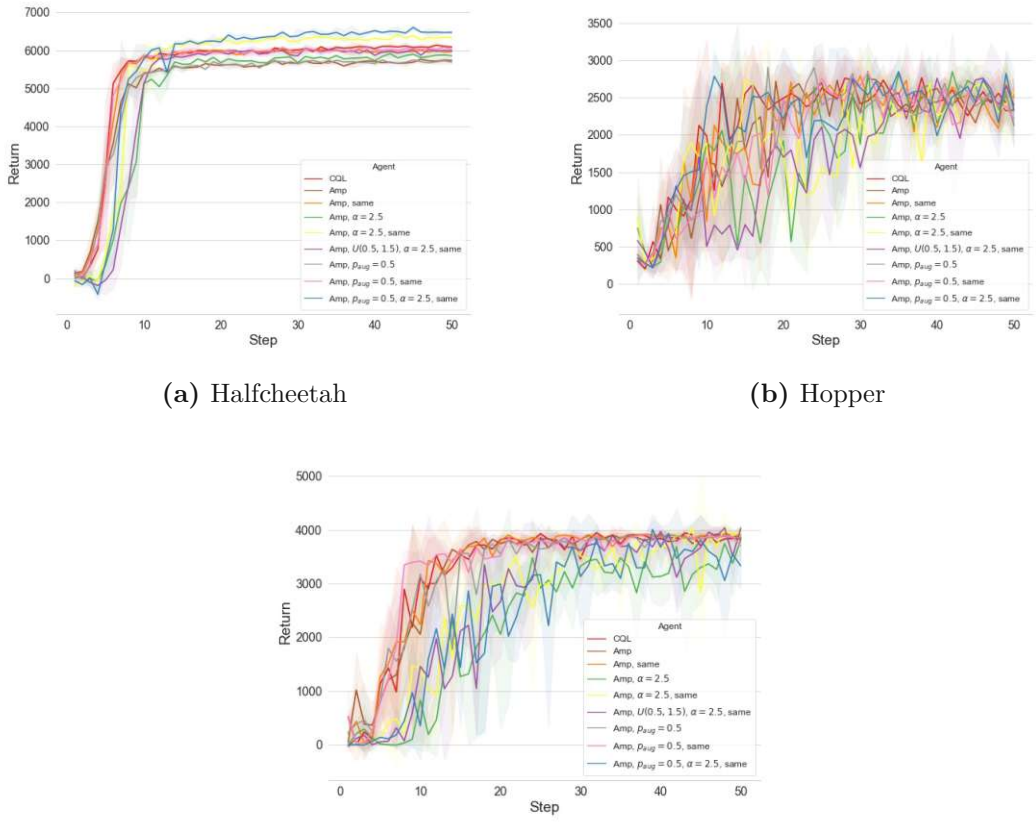
good performance on QBert (461.6%), but after 1.25M training steps CQL also performs better than the expert agent on Seaquest.

At the midpoint of training, the offline version of DQN also attains higher scores on average than its online counterpart, even though it performs poorly on three of the four games. Interestingly, offline DQN does, however, achieve the highest score observed so far on Seaquest. Over 1.25M steps BC achieves 75.3% the expert algorithm's performance.

5.2 Data augmentation

In this section, we present the results for our experiments with data augmentations.

5.2.1 Continuous control: Gym-MuJoCo



(a) Halfcheetah

(b) Hopper

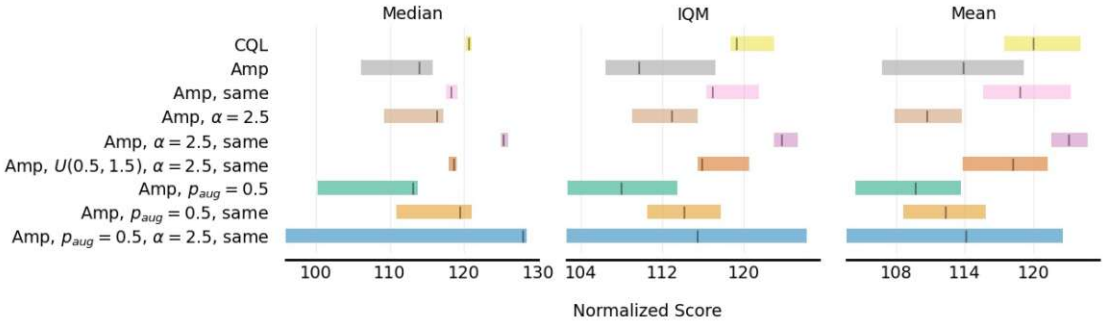
(c) Walker2d

Figure 5.3: Learning curves for CQL with amplitude scaling (Amp) on continuous control tasks.

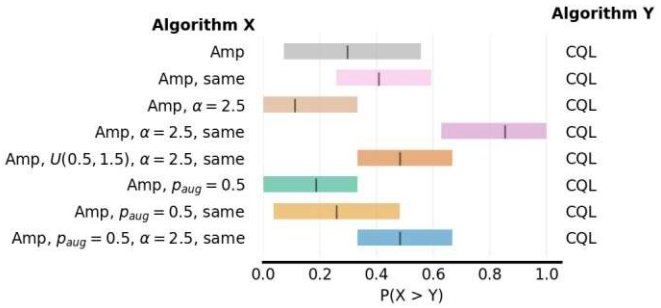
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

In Tables A.6 and A.7 in Appendix A.3 we show the raw and normalized scores for our experiments with data augmentations for a number of different configurations on continuous control tasks. As noted by Agarwal et al. [2021], the score tables can become quite overwhelming for a large number of experiments. Therefore, we focus our analysis on the figures for learning curves, interval estimates and probabilities of improvement.

Amplitude scale. In Figure 5.3 we show the learning curves for augmented CQL with amplitude scaling (Amp) in comparison to the base CQL agent. We ran experiments with different configurations for the parameters discussed in Section 4.4 (same augmentations for current & next observations, probability of augmentation p_{aug}, conservative weight α). Overall, we observe that most learning curves exhibit a similar learning progress. However, some configurations tend to be beneficial whereas others seem to be detrimental to the learning progress. For example, on Halfcheetah the blue ($p_{aug} = 0.5$, $\alpha = 2.5$, same) and yellow ($\alpha = 2.5$, same) learning curves clearly dominate the red CQL learning curve. On the same task, however, the green ($\alpha = 2.5$), grey ($p_{aug=0.5}$) and brown curve (default) are clearly worse than the base algorithm. On Walker2d the configurations with $\alpha = 2.5$ lead to more noisy learning curves, whereas all other parameters do not have much impact on the stability of the learning curves.



(a) Interval estimates



(b) Probability of improvement

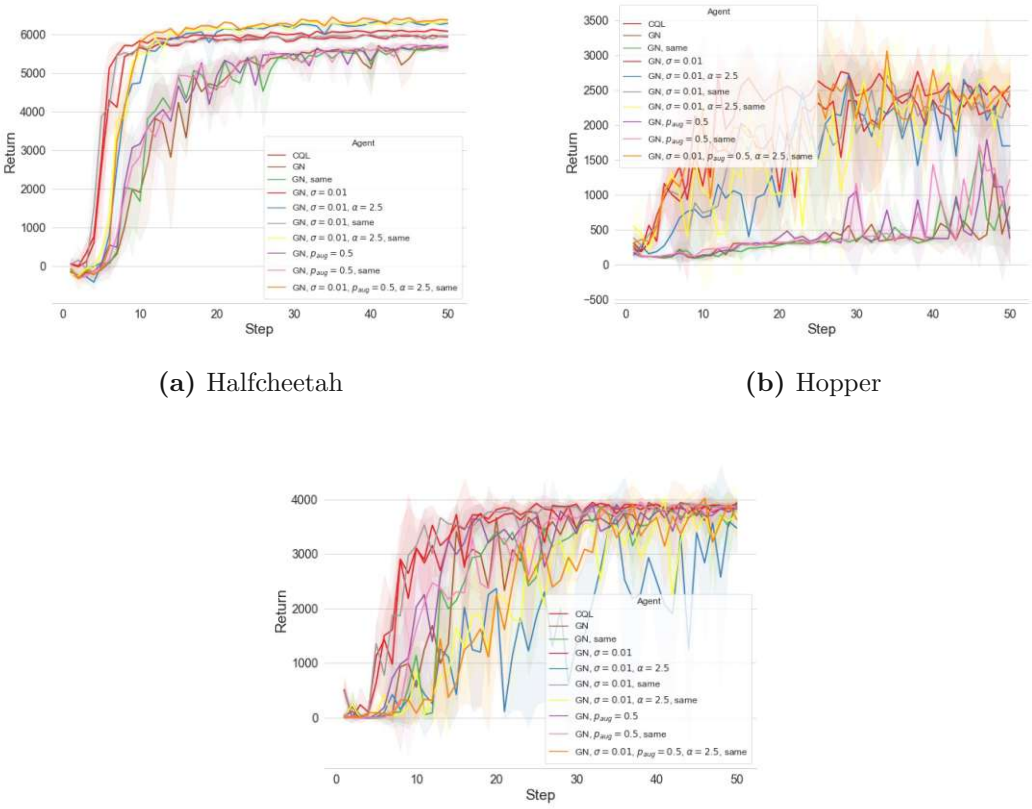
Figure 5.4: Interval estimates of normalized scores and probability of improvement for CQL with amplitude scaling (Amp) on continuous control tasks.

In terms of mean normalized performance (Table A.7), the variant with $\alpha = 2.5$ and

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.

same augmentations for s and s' reaches the highest score observed so far, 123.1% of the performance of the expert algorithm. This configuration outperforms the base CQL agent by 3.1%. These results are interesting, as CQL with $\alpha = 2.5$ performed much worse than the default variant with $\alpha = 5.0$, as we demonstrate in our ablation study in Section A.2.2 in Appendix A.2. The reason for this is that data augmentation acts as a form of regularization, and thus the conservative weight can be set to a lower value.

In Figure 5.4, we show the interval estimates and probabilities of improvement over base CQL for all variants of CQL with amplitude scaling. The graphical representation gives a clearer picture than the score tables (A.6, A.7). If we consider the IQM we observe that most configurations are clearly worse than the default CQL agent, except the configuration with $\alpha = 2.5$ and same augmentations. In fact, this configuration has a probability of improving over the default CQL agent of 85% given the observed scores.



(a) Halfcheetah

(b) Hopper

(c) Walker2d

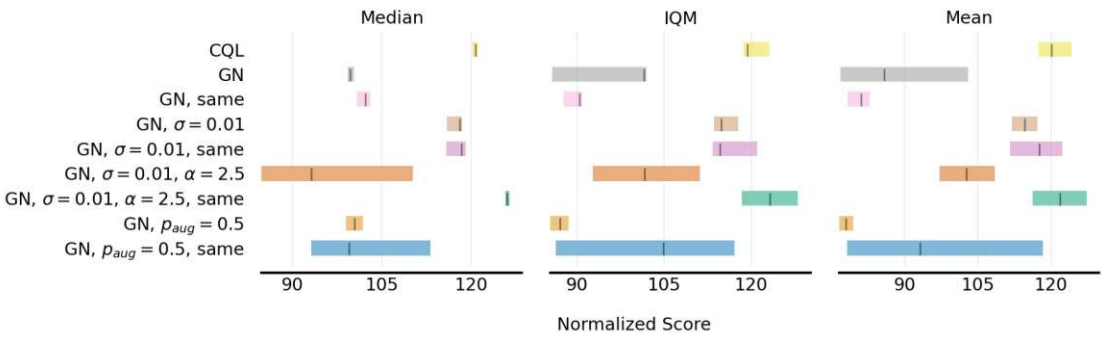
Figure 5.5: Learning curves for CQL with Gaussian noise (GN) on continuous control tasks.

Gaussian noise. Next, in Figure 5.5 we show the learning curves of all augmented CQL

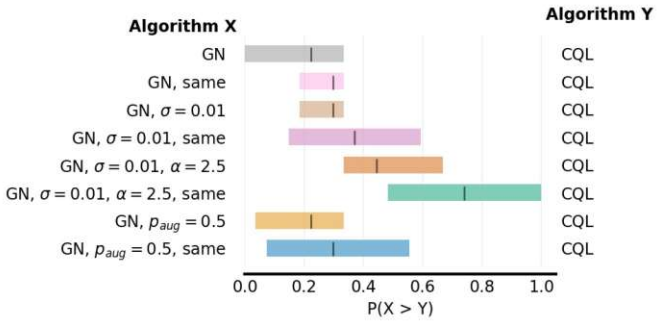
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

agents with Gaussian noise. Overall, we observe a similar behaviour as for agents with amplitude scaling. On some tasks, the learning curves of augmented agents are clearly above the learning curve of the base CQL agent, while for others they are clearly below. In particular, we observed poor learning performance with standard deviation $\sigma = 1.0$ (default) for the Gaussian noise (green, brown, pink, and purple curves). $\sigma = 1.0$ leads to a noisy training progress and lower overall scores. This indicates that the generated noise is too extreme for the underlying task. Therefore, we also experimented with $\sigma = 0.01$ and observed better learning progress with this setting. In particular, we observed that $\sigma = 0.01$ in combination with $\alpha = 2.5$ and same augmentations for s and s' works best, similar to the results with amplitude scaling. This configuration achieves a mean normalized score of 121.8% the performance of the expert, outperforming CQL by 1.8%.

In Figure 5.6 we show the interval estimates and probabilities of improvement over base CQL. Considering the IQM, the performance ranking remains the same as for the mean. The configuration with $\alpha = 2.5$ and same augmentations performs better than base CQL and has a mean probability of improvement of roughly 78% given the observed scores. All other configurations, in contrast, perform way worse than the base CQL agent.



(a) Interval estimates



(b) Probability of improvement

Figure 5.6: Interval estimates of normalized scores and probability of improvement for CQL with Gaussian noise (GN) on continuous control tasks.

Summary. In our experiments, we saw that data augmentations can lead to improve-

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 Your knowledge hub WIEN

ments (2-3%) in performance on continuous control Gym-MuJoCo tasks. However, extensive hyperparameter tuning may be necessary. If not tuned properly, the augmentations are more likely to be detrimental to the learning progress. This suggests that they might not be useful in every situation. For our experiments, however, we found that applying data augmentations with $\alpha = 2.5$ and using the same augmentation for current observations s and next observations s' performed best. Data augmentations implicitly regularize the value function, and thus we found it useful to reduce the conservative weight in return by half from $\alpha = 5.0$ to $\alpha = 2.5$.

5.2.2 Discrete control: Atari

In Tables A.8 and A.9 in Appendix A.3 we show the raw and normalized scores for all our experiments with data augmentations on Atari games. We again run a number of different configurations for each of the selected augmentations: random crop, random shift and cutout. As the learning curve figures become quite convoluted for many configurations, we only show a selection of learning curves in Figure A.6 in Appendix A.3. Also, the result tables and learning curves become quite overwhelming for a high number of configurations, and thus we primarily analyse our experiments based on the interval estimates in Figure 5.7. In Figure 5.7, we observe that all augmented variants of CQL are clearly worse than the base CQL agent in terms of mean normalized performance. If we consider the IQM instead of the mean, we notice that the confidence intervals overlap widely. The score tables and learning curves (Tables A.8, A.9, Figure A.6) also tell a similar story. Thus, in our experiments on discrete control Atari games, data augmentations did not improve the offline RL agent's learning performance. If anything, they rather hurt the agent's ability to learn high-reward behaviour. Why is this the case?

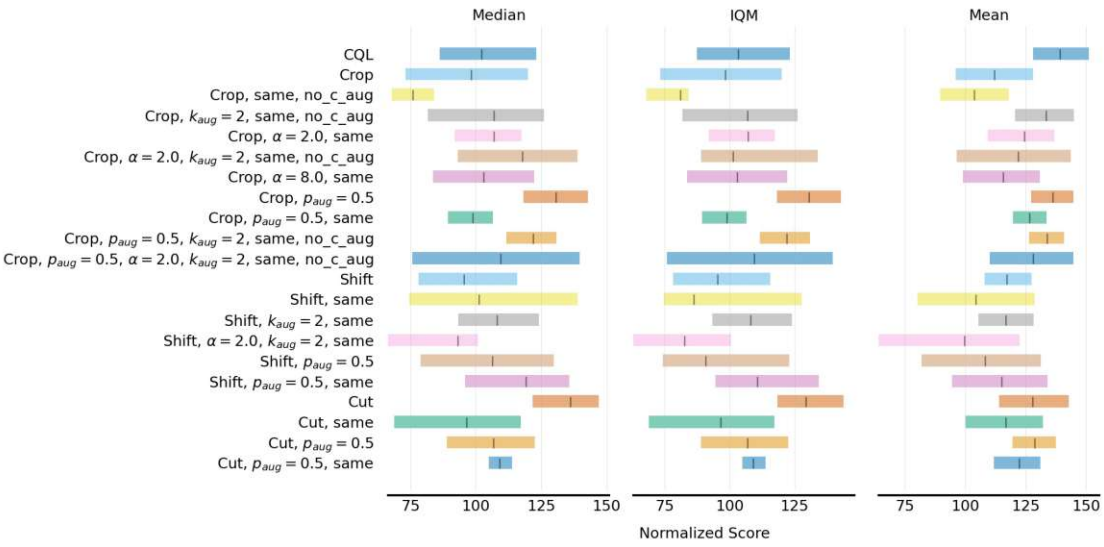


Figure 5.7: Interval estimates for CQL with data augmentations on Atari.

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 Your knowledge hub WIEN

Effect of data augmentation. In Figure 5.7 we show the aggregate scores over all four Atari games for all configurations with data augmentations. We do observe that augmentations affect the agent's learning performance differently on different games. For example, most augmented variants outperform the base agent on the Atari games Pong and Breakout, often by more than ten percentage points, as shown in Table A.9. However, all augmented variants are much worse on Seaquest and QBert. Pong and Breakout are relatively simple games: the agent only has to control a small pedal (up/down in Pong, left/right in Breakout) to hit a ball. Seaquest and QBert are comparatively much more complex. For example, in Seaquest, the agent controls a submarine, has to shoot enemies (multiple other submarines & sharks) and rescue divers from the water. Now, the data augmentation cutout would cut out a region from the current observation. If an enemy is not visible, the agent cannot perceive it. Thus, the performance might suffer. In contrast, for Pong and Breakout, data augmentations do not severely alter the game semantics. There is only one major object in both games and even if it would be covered by a cut-out region, it remains easy to guess the position.

Atari100K. Another potential reason for the poor performance of data augmentations in our experiments is that the RL architectures with data augmentations were originally applied to a low-data regime. Yarats et al. [2021b] developed their architecture, DrQ for Atari100K. Atari100K only allows for 100K gradient steps and is one of the most prominent benchmarks for data-efficient RL, as was discussed in Section 3.3.1. In this work, on the other hand, we train for 2.5M gradient steps, which is in line with prior work on Atari in offline RL. Thus, it is possible that the performance gains of data augmentations really only help in the beginning of training, but the benefits degrade if training for more steps. If we consider the scores after 100K steps, we do indeed observe substantial improvements over base CQL in some cases. However, as we study the impact of data augmentations on offline RL agents, we do not investigate the Atari100K benchmark further. In our Atari experiments, data augmentations are thus not beneficial.

5.3 Self-supervised methods

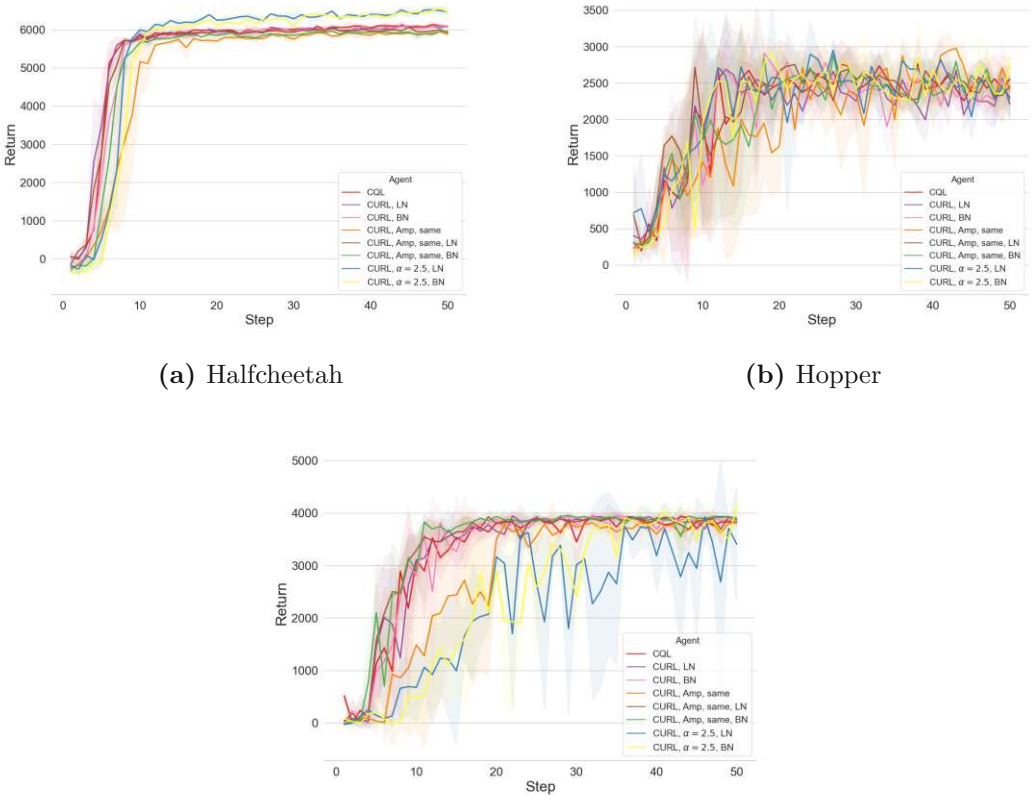
Next, we describe our experiments with the self-supervised architectures Curl, SPR and SGI.

5.3.1 Continuous control: Gym-MuJoCo

First, we present our experiments on Gym-MuJoCo.

Curl. In Tables A.10 and A.11 in Appendix A.4 we show the raw and normalized scores for all our experiments with the self-supervised architecture Curl. Again, we conduct a number of experiments with different configurations. In particular, we run experiments with different normalization variants (BN=BatchNorm, LN=LayerNorm) and augmentation strategies, as described in 4.5.1, as well as experiments with a lower value for the conservative weight α that are motivated by the previous section.

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 Your knowledge hub WIEN



(a) Halfcheetah

(b) Hopper

(c) Walker2d

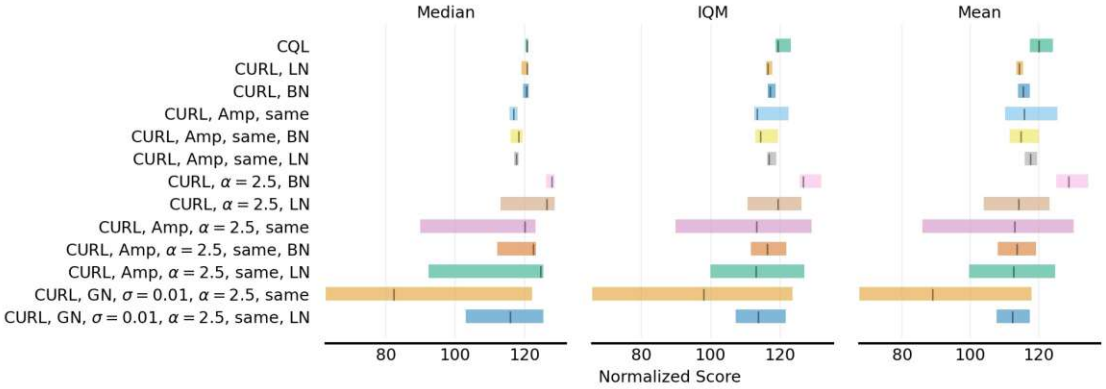
Figure 5.8: Learning curves for CQL + Curl on continuous control tasks.

In Figure 5.8 we show the learning curves for a selection of Curl agents. For visibility, we omit the curves of the five worst performing configurations. Overall, the learning curves exhibit a similar behaviour as the learning curves with data augmentations. In general, this does make sense, as all agents share the same base CQL algorithm. On Halfcheetah, we observe two curves (blue and yellow, the variants without augmentation, with $\alpha = 2.5$, and with BN/LN) that are clearly above the red CQL curve. The learning curves on the Hopper environment are, again, quite noisy. On Walker2d we also observe a similar pattern, namely that the variants with $\alpha = 2.5$ lead to noisier learning curves and slower initial learning progress, compared to variants with $\alpha = 5.0$.

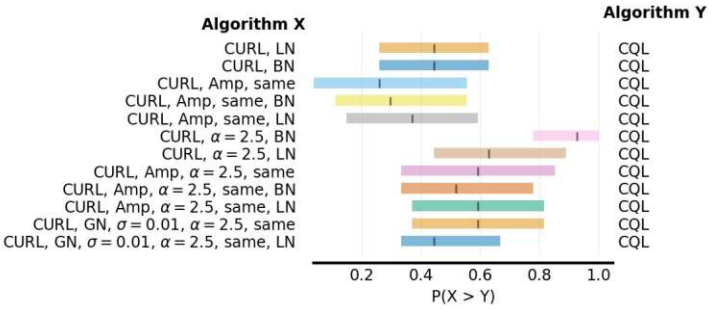
If we consider the mean normalized scores, we see that the variant without global augmentation (in which augmentations are only applied for the computation of the contrastive loss, as discussed in Section 4.5.1), with $\alpha = 2.5$ and BatchNorm achieves the highest scores observed so far, 128.8% the performance of the expert agent. However, this score should be treated with caution, as the learning curves for this configuration exhibit

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 Your knowledge hub WIEN

high variance. Nevertheless, this result is interesting, as in the original publication, the variants with global augmentation performed better. Also, the original publication employed LayerNorm instead of BatchNorm at the respective places. Laskin et al. [2020b] developed the Curl architecture on vision-based environments, but we adjusted and applied it to a state-based environment. Due to different environment dynamics, different choices might be appropriate.



(a) Interval estimates



(b) Probability of improvement

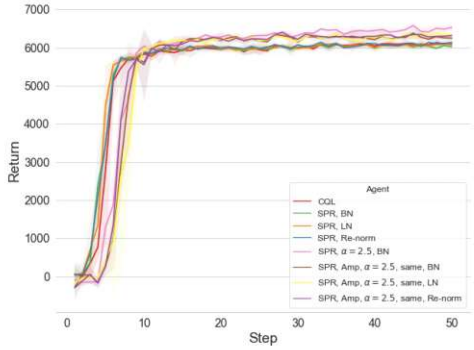
Figure 5.9: Interval estimates of normalized scores and probability of improvement for CQL + Curl on continuous control tasks.

In addition, we show the interval estimates and probabilities of improvement over CQL in Figure 5.9. If we consider the IQM, the Curl variant [$\alpha = 2.5$, BN] also outperforms the base CQL agent. For other configurations (e.g., [$\alpha = 2.5$, BN] or [Amp, $\alpha = 2.5$, same]) the confidence intervals clearly overlap with the CQL interval. Furthermore, the best variant [$\alpha = 2.5$, BN] has a probability of improvement over the base CQL agent of 92.6%, given the scores we observed. Interestingly, other variants also have a probability of improvement of 60%, even though their overall scores are worse. This, however, is due to the wide confidence intervals these configurations exhibit.

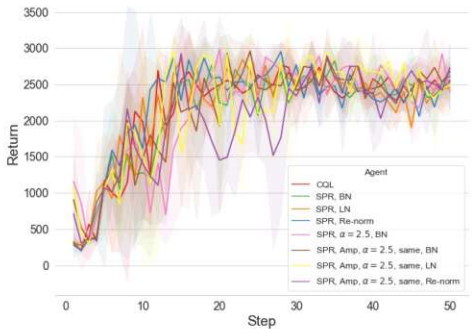
SPR. In Tables A.12 and A.13 we show the raw and normalized scores for all variants of SPR on Gym-MuJoCo. In Figures 5.10 and 5.11 we show the learning curves, interval

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.

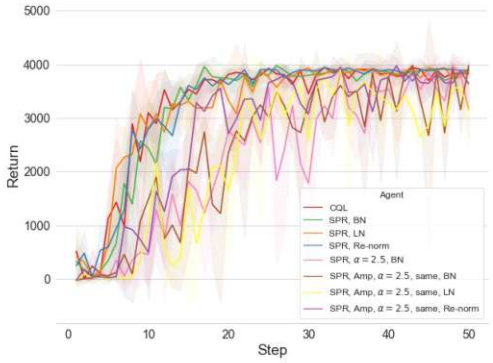
estimates and probabilities of improvement graphically. For visibility, we again omit the five worst configurations from the learning curve plots.



(a) Halfcheetah



(b) Hopper



(c) Walker2d

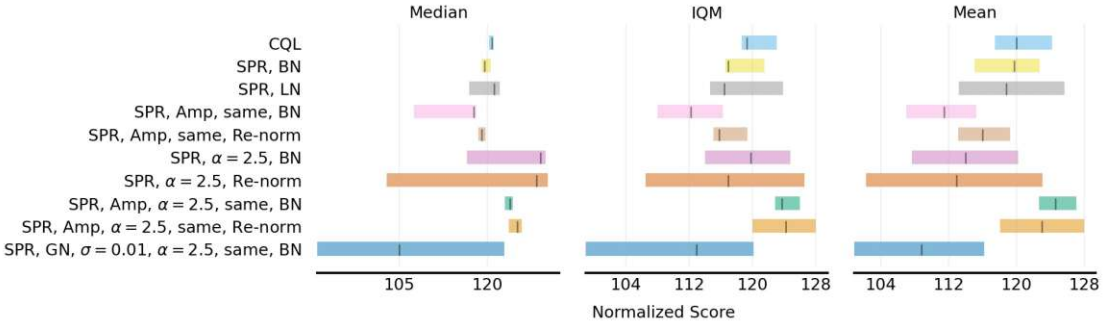
Figure 5.10: Learning curves for CQL + SPR on continuous control tasks.

In Figure 5.10, we observe that on Halfcheetah all SPR configurations perform at least as good or better than base CQL (red curve). However, on Hopper and Walker2d, the outcome is not as clear. As previously, the variants with $\alpha = 2.5$ perform particularly well on Halfcheetah, but lead to slow learning progress on Walker2d. Overall, the three SPR variants [Re-norm], [Amp, $\alpha = 2.5$, same, Re-norm] and [Amp, $\alpha = 2.5$, same, BN] outperform base CQL achieving a mean normalized score of 122%, 123%, and 124.6%, respectively, as shown in Table A.13.

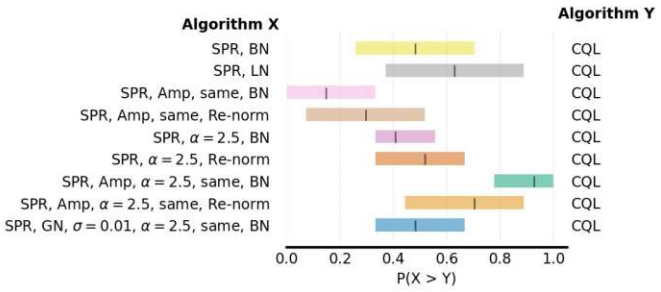
Finally, in Figure 5.11 we observe that the IQMs of [Amp, $\alpha = 2.5$, same, Re-norm] and [Amp, $\alpha = 2.5$, same, BN] are clearly higher than that of base CQL, even though their intervals overlap. Also, for other configurations the intervals overlap strongly with base CQL, despite being worse overall. The variants [Amp, $\alpha = 2.5$, same, Re-norm] and

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.

[Amp, $\alpha = 2.5$, same, BN] have probabilities of improvement over CQL of 70.4% and 92.5%, respectively.



(a) Interval estimates



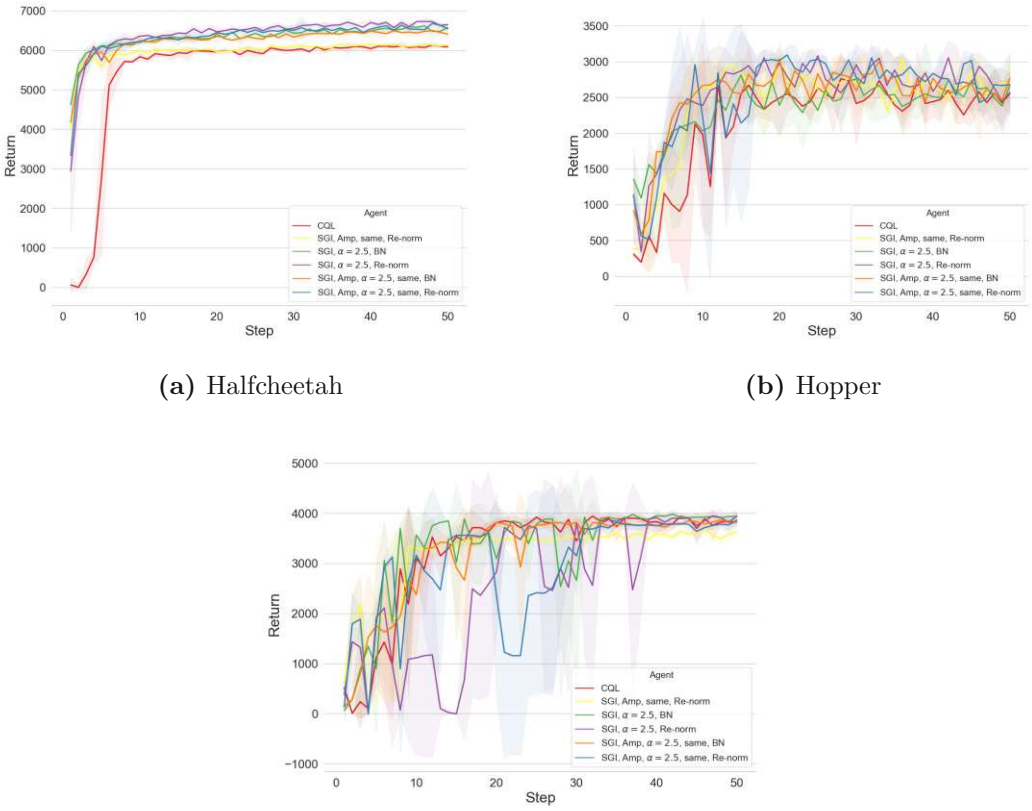
(b) Probability of improvement

Figure 5.11: Interval estimates of normalized scores and probability of improvement for CQL + SPR on continuous control tasks.

SGI. In Tables A.14 and A.15 in Appendix A.4, we show the raw and normalized scores at the final step and at the midpoint of training for our experiments with SGI on Gym-MuJoCo. In Figure 5.12 we show the learning curves for the five best variants on Halfcheetah, Hopper and Walker2d. On the Halfcheetah task, the learning curves for all variants of SGI already start at a higher level of performance compared to base CQL (red curve). The reason for this is that SGI pre-trains the encoder representations in a reward-free manner, as we discussed in Section 4.5.3. On Halfcheetah, the pre-trained representation facilitate the agent's interaction with the environment. Similarly, on the Hopper task the pre-trained representations lead to faster initial learning progress, even though the effect is not as noticeable as on Halfcheetah. SGI outperforms the base CQL agent by up to 6.2 percentage points at the final time step, as we show in Table A.15. All configurations with $\alpha = 2.5$ outperform the base CQL agent. Also, one configuration with $\alpha = 5.0$ achieves as higher mean normalized score as base CQL.

In Figure 5.13 we show the interval estimates and probabilities of improvements for SGI. In terms of the mean normalized score, the intervals of the best four configurations overlap with the interval of base CQL. However, this is not the case for the IQM and median (except for the last configuration). The four best configurations have high mean

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 Your knowledge hub WIEN



(a) Halfcheetah

(b) Hopper

(c) Walker2d

Figure 5.12: Learning curves for CQL + SGI on continuous control tasks.

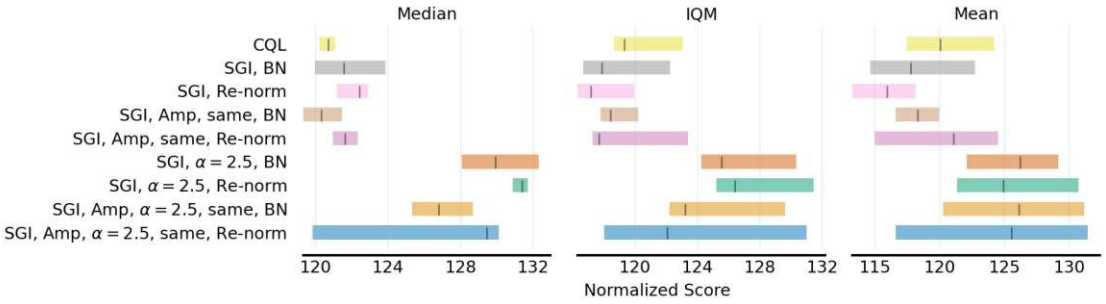
probabilities of improvement over the base CQL agent (up to 89%). Overall, these results highlight the effectiveness of reward-free pre-training of encoder representations. Compared to SPR, the final scores of architectures with SGI are higher, and we observe more consistent improvements across different configurations.

5.3.2 Discrete control: Atari

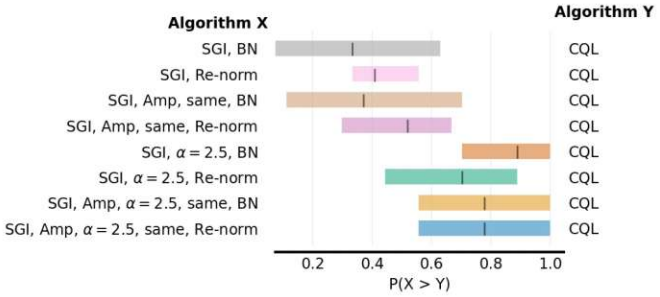
Next, we describe our experiments with self-supervised architectures on Atari games. To this end, we show the raw and normalized scores for all configurations with self-supervised architectures in Tables A.16 and A.17 in Appendix A.4.

In Figures A.7 and A.8 in Appendix A.4, we depict the learning curves for Curl and SPR, respectively. Overall, we observe a similar learning behaviour as with data augmentations on Atari games. The overall scores of all self-supervised architectures are clearly worse than the base CQL agent. Even though this is, again, not true for all Atari games. In terms of IQM, most variants do not differ much from the base CQL agent, as shown 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 Bibliothek Your knowledge hub WIEN



(a) Interval estimates



(b) Probability of improvement

Figure 5.13: Interval estimates of normalized scores and probability of improvement for CQL + SGI on continuous control tasks.

Figure 5.14. Curl and SPR were also developed for the Atari100K benchmark. Therefore, we again hypothesize that self-supervised methods benefit the agents in the beginning of training, but do not help if they are trained for more steps. If we consider their scores after 100K steps, we do indeed observe substantial improvements over the base agent, especially for Curl. For the benchmark we consider in this work, however, we conclude that they do not benefit the learning performance of the selected offline RL agent.

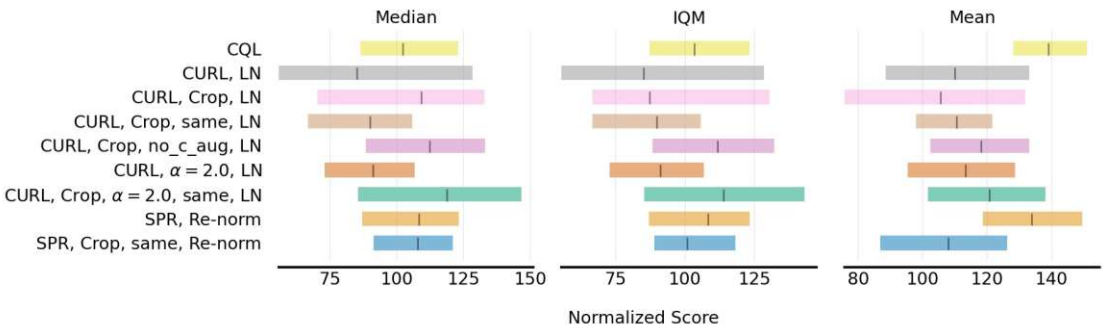


Figure 5.14: Interval estimates for CQL with self-supervised tasks on Atari.

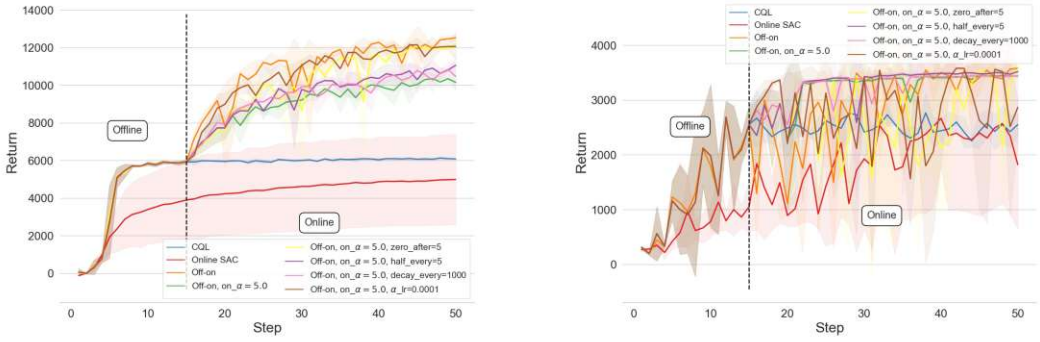
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

5.4 Online fine-tuning for Offline RL

Finally, we present our experiments with offline pre-training and online fine-tuning of CQL on Gym-MuJoCo and Atari.

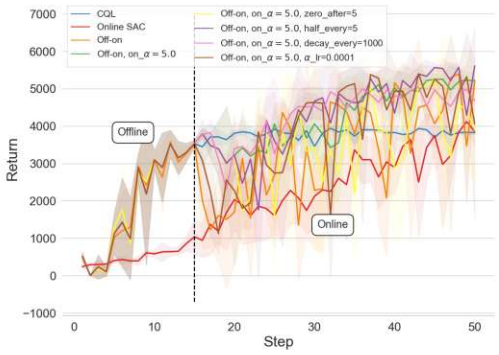
5.4.1 Continuous control: Gym-MuJoCo

In Figure 5.15 and Figure 5.16 we show the learning curves, interval estimates and probabilities of improvement for all variants of CQL with offline pre-training and online fine-tuning (denoted "Off-on" in the figures) on Gym-MuJoCo. Furthermore, in Tables A.18 and A.19 in Appendix A.6 we show the raw and normalized performance scores.



(a) Halfcheetah

(b) Hopper



(c) Walker2d

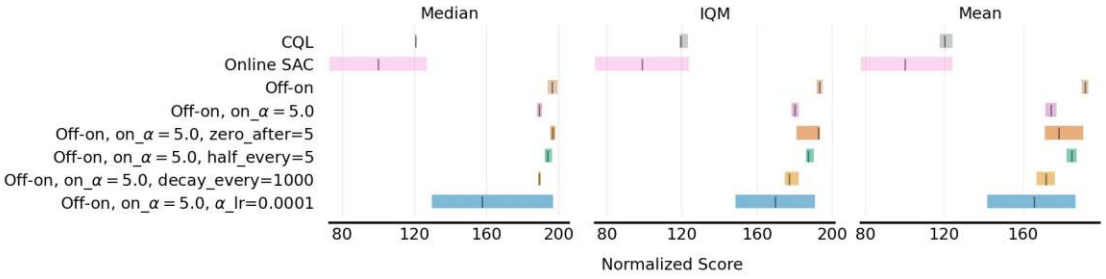
Figure 5.15: Learning curves for CQL with online fine-tuning on continuous control tasks.

Learning curves. In Figure 5.15 the dashed line indicates the shift from offline RL to online RL at 150K steps. The blue and red learning curves represent the regular offline and online agents, CQL and SAC. We observe that the variants with offline pre-training

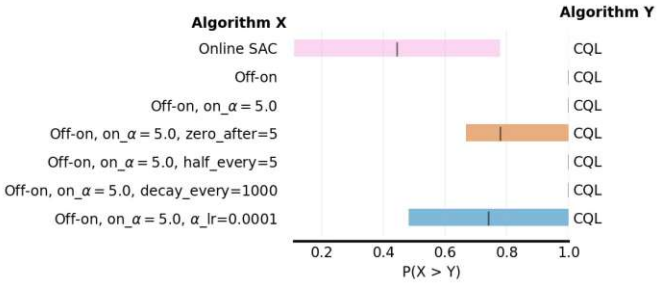
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

and online fine-tuning outperform the base agents by a large margin. On Halfcheetah the difference in performance is particularly apparent. On this task, the transition from offline training to online training is smooth, and all variants improve quickly. The best agent achieves 191.5% of the performance of the expert online agent and outperforms all other agents in this thesis. On Hopper and Walker2d the transition from offline to online training is not as smooth. For both tasks, we observe a big drop in performance at the inflection point, in particular for the variant that turns off the conservative loss immediately (Off-on). In contrast, for the variants that preserve/half/decay α, the drop in performance is not as severe. In particular, on the Hopper task, these methods eliminate this drop altogether. The variant with automatic tuning of α prevents the immediate drop in performance, but then leads to higher variance. Furthermore, the variant that preserves $\alpha = 5.0$ throughout the training process achieves the lowest score on Halfcheetah but performs well on the other two tasks.

In aggregate, the best variant (Off-on) achieves 191.5% the performance of the expert agent. Compared with the 120% of base CQL, these scores represent a large improvement. However, the variant also exhibits a large drop in performance at the inflection point. In practice, this could be problematic for many reasons. Therefore, methods that prevent this behaviour, e.g., by slowly decaying the conservative loss over time, might be preferable.



(a) Interval estimates



(b) Probability of improvement

Figure 5.16: Interval estimates of normalized scores and probability of improvement for CQL with online fine-tuning on continuous control tasks.

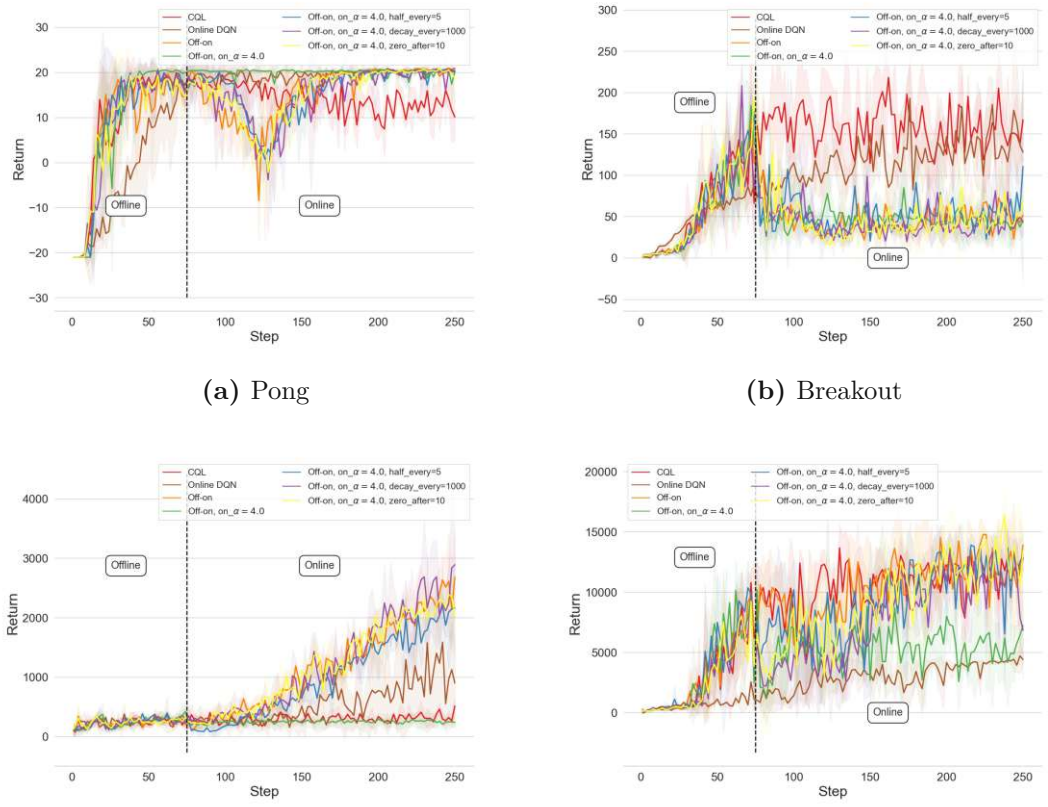
Interval estimates. Figure 5.16 illustrates the performance gains graphically. The four best variants significantly outperform the base agents and have a 100% chance of actually

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.

being better given the observed scores. Overall, these results suggest that RL algorithms that combine offline and online RL can be far superior to offline or online agents alone.

5.4.2 Discrete control: Atari

Finally, we present our results with offline pre-training and online fine-tuning on Atari. The raw and normalized scores for our experiments are available in Tables A.20 and A.21 in Appendix A.6.



(a) Pong **(b)** Breakout

(c) Seaquest **(d)** QBert

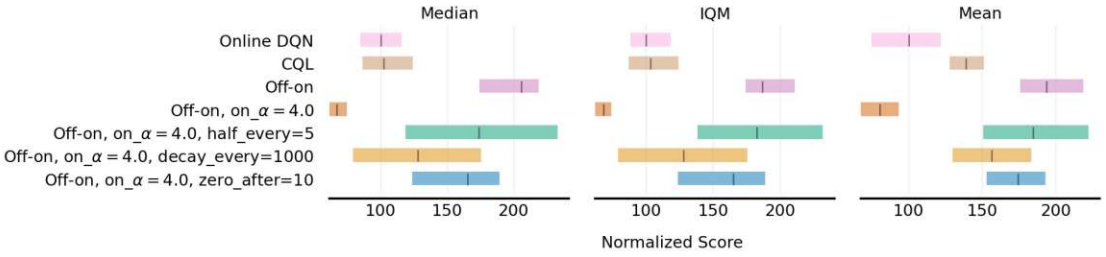
Figure 5.17: Learning curves for CQL with online fine-tuning on discrete control tasks.

Learning curves. In Figure 5.17 we show the learning curves for all agents. Again, the dashed line represents the shift from offline RL to online RL, now after 750K steps. The blue and brown curves represent the regular CQL and online DQN agents. Overall, we do observe considerable improvements over the base agents. Again, the variant that turns off the conservative loss abruptly obtains the highest score observed so far, 193.4% of the expert's performance at the last step (i.e., 93.4% improvement over DQN and 54% improvement over CQL). Other variants achieve similarly high scores. However,

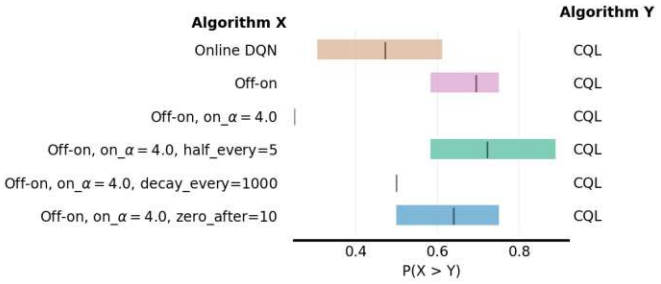
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

the learning curves reveal that the improvements are not distributed equally across the four games. On Pong, we observe a quite severe drop in performance at the inflection point from offline to online training. Only the variant that preserves $\alpha = 4.0$ throughout training avoids this drop. The variants that only decay/half/zero the conservative weight over time do, however, not prevent the drop in performance. Nevertheless, all agents recover from the performance drop on Pong. We observe similar behaviour on Breakout, but unlike on Pong, the agents do not recover and deliver poor performance after 2.5M steps. This suggests that naively manipulating the conservative loss during online fine-tuning is not enough in every situation, and dedicated algorithms need to be developed. In contrast, on Seaquest, all agents that turn off/decay/half/zero the conservative weight deliver a level of performance that is far superior to learning online or offline alone. Similarly, these agents perform well on QBert and outperform base CQL and online SAC.

Interval estimates. In addition, we show the interval estimates and probabilities of improvement in Figure 5.18. The intervals estimates confirm that all hybrid agents (except the versions with $\alpha = 4.0$ and decay_every $= 1000$) outperform their counterparts. Even though the CIs for some variants are wide, they do not overlap with the intervals of base CQL or online DQN. This holds for both the mean IQM interval estimates. However, the probabilities of improvement over base CQL are comparatively low (around 70%). The reason for this are the poor performance scores on the game Breakout. On Breakout, the scores of all individual runs are worse than the ones of CQL. Therefore, this is reflected in the relatively low probabilities of improvement (as computed in Equation 3.2).



(a) Interval estimates



(b) Probability of improvement

Figure 5.18: Interval estimates of normalized scores and probability of improvement for CQL with online fine-tuning on discrete control tasks.

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 Your knowledge hub WIEN

5.5 Discussion

Offline RL vs. online RL. Our baseline experiments in Section 5.1 highlighted the opposite characteristics of offline RL and online RL. Offline RL, on the one hand, is characterized by fast initial learning progress but limited potential for long-term improvement. Initially, offline RL agents, such as CQL, learn faster, as experience is readily available in the dataset. Once they have leveraged all the available information in the dataset, they cannot continue to improve, as there is no possibility to collect further experience. It is, however, important to note that the performance of offline RL agents heavily depends on the quality of the dataset [Schweighofer et al., 2021]. In this work, we did not address this limitation. Online RL, on the other hand, is characterized by slow initial learning progress but substantial potential for long-term improvement. Online RL agents, such as DQN or SAC, first need to discover high-reward behaviour by random interaction and thus the slower initial learning progress. However, environment interaction gives them the opportunity to discover the best possible behaviour over time. If we continued to run our experiments for longer, the performance of online RL agents would arguably continue to improve, whereas for offline agents this would likely not be the case. These opposite characteristics of offline RL and online RL motivated the experiments we presented in Section 5.4.

Self-supervision and data augmentation for offline RL. Our experiments in Sections 5.2 and 5.3 suggest that self-supervised methods and data augmentations can improve the learning performance of offline RL agents. On Gym-MuJoCo, they improved the agent's final performance by up to 9 percentage points. Overall, on Gym-MuJoCo we found it useful to decrease the conservative weight α to half the initial value ($\alpha = 2.5$) when data augmentations and self-supervised methods are employed. This characteristic is interesting, as our experiments in Section A.2.2 using CQL with the same parameter setting performed poorly. We made another interesting observation with the self-supervised architecture SGI. SGI employs reward-free pre-training of encoder representations. These pre-trained representations facilitate the interaction with the environment. In our experiments with SGI in Section 5.3.1, we observed very fast initial learning progress on the Halfcheetah and Hopper tasks.

In contrast, on Atari, none of our experiments with data augmentations or self-supervised methods resulted in performance improvements. Instead, they rather hurt the agent's ability to learn high-reward behaviour over 2.5M gradient steps. In online RL, however, the self-supervised methods and data augmentations we employ did indeed improve the learning performance. Prior work in online RL with the same architectures evaluated their performance in the low-data regime and only allowed for 100K observations. In contrast, the offline datasets we used contain 1M observations, and we trained for 2.5M gradient steps. Therefore, we hypothesized that these methods help if training for a small number of steps, but are not beneficial if training for longer. Indeed, if we consider the performance after 100K gradient steps, we did observe performance improvements. For offline RL, however, the Atari100K benchmark is not optimal, as under these circumstances simple methods, such as BC, can be far superior to more sophisticated offline RL agents.

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 Your knowledge hub WIEN

Overall, our results with self-supervised methods and data augmentations are mixed. Even though, we did observe substantial performance improvements on Gym-MuJoCo, the agents required extensive hyperparameter tuning. Nevertheless, our results on Gym-MuJoCo suggest that self-supervised methods and data augmentations can improve the learning performance of offline RL agents. Ultimately, self-supervised methods and data augmentations enable RL agents to acquire a better understanding of the environment, i.e., world knowledge. We believe that a deep understanding of how the environment works will be necessary for intelligent agents in the real world. Therefore, we believe that both approaches are a promising direction for future research in the field of offline RL.

Offline pre-training, online fine-tuning. Hybrid RL algorithms combine the benefits of offline RL and online RL by offline pre-training and online fine-tuning, as we have demonstrated in our experiments in Section 5.4. In our experiments, CQL with online fine-tuning was far superior to its offline and online counterparts, outperforming the base CQL agent on Gym-MuJoCo and Atari by 71.5 and 51.4 percentage points, respectively. However, our approach has limitations. Most importantly, the large performance drops at the inflection point from offline to online training that we observed in some environments. For real world applications, this would be problematic. To prevent these drops, we experimented with methods that turn off the conservative weight α smoothly over time. This naive approach works to some degree, but does not consistently eliminate performance drops across all environments. Simply turning off or decaying the conservative weight over time might, thus, not suffice in all situations, in particular if the agent would alternate between online RL and offline RL.

Future RL algorithms. In summary, our experiments suggest that future RL agents need to have the ability to learn from experiences that originate from a variety of sources, offline datasets and online environment interaction alike, in a data-efficient way. In addition, future RL agents need to acquire a rich understanding of how the world they operate in works and how it changes over time (e.g., via self-supervised learning methods). Such agents would empower several sophisticated RL strategies that are attractive for real-world applications. One notable example for such a strategy are RL agents that learn continuously throughout their lifetimes. This approach is known as open-ended/life-long/continual RL. Another closely related strategy is multi-task RL, which departs from learning only a single task at a time and instead aims to learn many tasks at once. Both approaches would strongly benefit from agents that can learn from all kinds of experience sources alike. In this thesis, we did not address open-ended RL and multi-task RL. In the long term, however, we should build RL agents that learn over an extended period of time in a sufficiently complex and open-ended environment (e.g., in the real world). Under these circumstances, RL agents may develop increasingly complex/intelligent behaviours. Overall, we believe that all these methods will be essential ingredients for general-purpose AI, and we look forward to exploring them in the near future.

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 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.

CHAPTER 6

Conclusion

In this chapter, we conclude this thesis. First, we summarize the work we conducted. Then, we highlight the contributions we make in this thesis.

6.1 Summary

In Chapter 2, we provided a comprehensive overview of the theoretical background for this thesis. This included a literature review on online RL and offline RL. We also discussed the individual strengths and weaknesses of both paradigms. In addition, we reviewed the state-of-the-art literature on self-supervised learning and data augmentation in RL.

In Chapter 3, we described our methodological approach for this thesis (algorithm selection, dataset selection, evaluation methodology, software, hardware, etc.). Overall, we selected a state-of-the-art offline RL algorithm, CQL, as our base agent for all experiments. We then selected three self-supervised RL architectures (Curl, SPR, SGI) and five data augmentations (random crop, random shift, cutout, amplitude scale, Gaussian noise) and adjusted them for offline RL. Furthermore, we experimented with CQL for offline pre-training and online fine-tuning. We evaluated all algorithms on four Atari games (Pong, Breakout, QBert, Seaquest) and three Gym-MuJoCo tasks (Halfcheetah, Hopper, Walker2d). All Atari games have a discrete action space and provide image-based observations. In contrast, all Gym-MuJoCo tasks have a continuous action space and emit state-based observations.

In Chapter 4, we detailed what experiments we conducted and how we conducted them. This includes a detailed description of the hyperparameters and adjustments we made to the learning algorithms.

Finally, in Chapter 5 we presented the results that we obtained during our experiments. Our results showed that self-supervised methods and data augmentations can outperform

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 Your knowledge hub WIEN

the baseline agents and considerably improved the learning performance of offline RL algorithms on Gym-MuJoCo, but were not beneficial on Atari. Furthermore, we demonstrated that with the right design decisions, CQL with offline pre-training and online fine-tuning and can be far superior to online learning and offline learning alone. Overall, our results suggest that there is a need for hybrid RL algorithms that can learn both offline and online in a data-efficient way.

6.2 Contributions

To conclude, we make the following **contributions** in this thesis:

- We provide a comprehensive overview of the state-of-the-art in the research field of offline RL.
- We evaluate the effectiveness of existing self-supervised methods and data augmentation techniques for offline RL and show that they can lead to considerable improvements on Gym-MuJoCo but are not beneficial on Atari.
- We demonstrate that the "offline pre-training, online-fine tuning" RL paradigm can be far superior to online or offline RL individually and advocate for hybrid RL algorithms that are designed to learn both offline and online.

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 Your knowledge hub WIEN

APPENDIX A

Full results

A.1 Online training

For Atari, it is not obvious how "long" online agents should be trained. To ensure a fair comparison with online agents, we consider a few options:

(1) environment steps = 2.5M, update interval = 4

(2) environment steps = 2.5M, update interval = 1

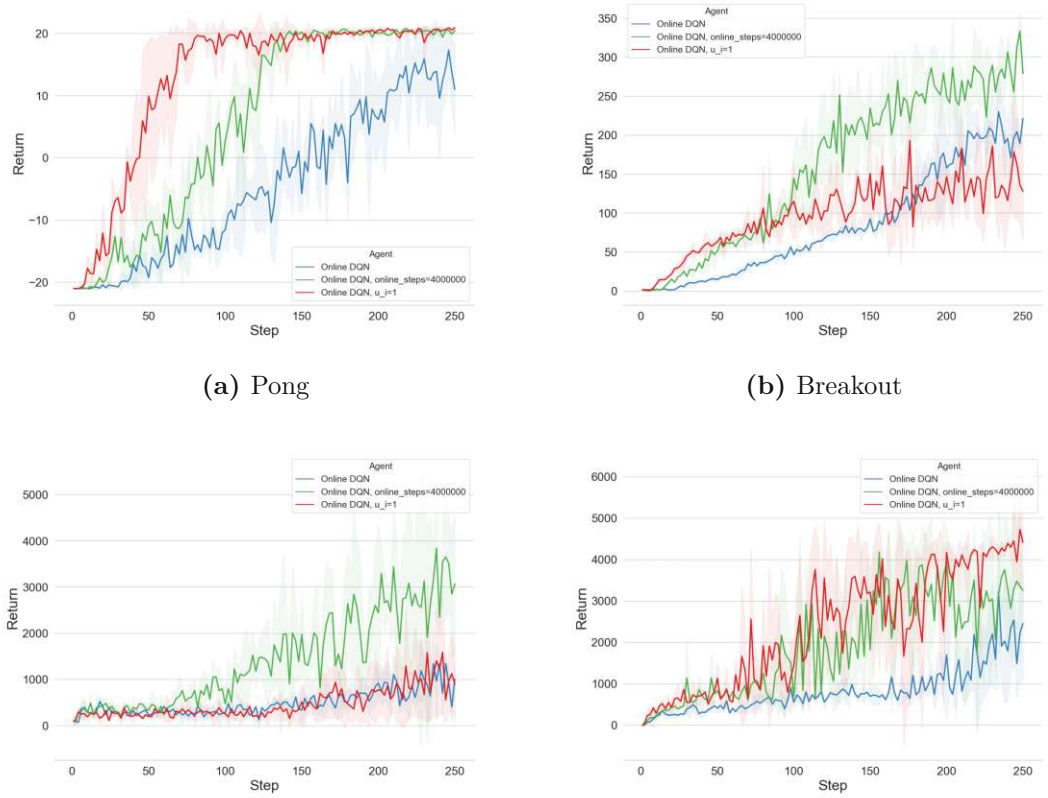
(3) environment steps = 4M, update interval = 4

Option (1) results in 625K observations & gradient steps. Option (2) results in 625K observations, 2.5M gradient steps. Option (3) results in 1M observations & gradient steps. We show the learning curves and normalized scores for all three options in Figure A.1 and Table A.1.

Agent	Breakout	Pong	QBert	Seaquest
Final step (2.5M)				
Online DQN	221.6 ± 29.0	11.0 ± 7.3	2458.3 ± 1358.4	1008.0 ± 358.6
Online DQN, u_i=1	128.2 ± 63.6	20.9 ± 0.1	4416.7 ± 227.8	901.3 ± 778.2
Online DQN, online_steps=4M	279.3 ± 45.4	20.5 ± 0.4	3258.3 ± 302.4	3068.0 ± 1452.3
Midpoint (1.25M)				
Online DQN	73.1 ± 6.9	-6.1 ± 6.5	747.5 ± 53.8	318.7 ± 123.8
Online DQN, u_i=1	122.3 ± 13.3	18.8 ± 1.8	2523.3 ± 828.7	182.7 ± 70.5
Online DQN, online_steps=4M	218.7 ± 63.4	16.4 ± 2.8	1058.3 ± 763.6	1448.0 ± 751.8

Table A.1: Raw scores for online training ablation.

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 Your knowledge hub WIEN



(a) Pong **(b)** Breakout

(c) Seaquest **(d)** QBert

Figure A.1: Learning curves for online ablation.

At the final step, we observe that variant (2) achieves the highest scores on Pong and QBert. In contrast, variant (3) scores best on Breakout and Seaquest. The same holds for the performance at the midpoint of training. Both variants are solid choices. For our subsequent experiments, we select variant (2) with update interval of 1 and the same number of gradient steps/environment steps as all offline RL agents. Variant (2) results in the most balanced scores across all environments. In addition, this decision simplifies the comparison of our agents in Section 5.4.

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 Your knowledge hub WIEN

A.2 Baselines

In this section, we list the full results for our experiments with baselines. The results for continuous control tasks (Gym-MuJoCo) are listed in Tables A.2 and A.3. The results for discrete control tasks (Atari) are listed in Tables A.4 and A.5.

A.2.1 Continuous control: Gym-MuJoCo

Agent	halfcheetah-medium-v2	hopper-medium-v2	walker2d-medium-v2
	Final step (500K)		
Random	-290.08	11.13	-1.66
Online SAC	4994.6 ± 2393.9	1825.0 ± 1173.1	3861.4 ± 386.5
Offline SAC	2154.1 ± 1731.5	4.3 ± 0.1	-4.6 ± 2.5
BC	4888.0 ± 61.7	1625.2 ± 108.7	3086.2 ± 442.6
CQL	6088.3 ± 22.7	2553.9 ± 202.4	3829.8 ± 70.2
	Midpoint (250K)		
Random	-290.08	11.13	-1.66
Online SAC	4410.1 ± 2267.3	1429.3 ± 1093.5	1605.4 ± 514.5
Offline SAC	172.3 ± 325.4	4.3 ± 0.1	-7.5 ± 2.3
BC	4915.9 ± 51.9	1781.9 ± 179.2	3070.7 ± 248.3
CQL	5948.1 ± 112.8	2630.9 ± 241.4	3924.8 ± 35.5

(a) Raw scores

Agent	halfcheetah-medium-v2	hopper-medium-v2	walker2d-medium-v2	Mean
	Final step (500K)			
Offline SAC	46.3	-0.4	-0.1	15.3
BC	98.0	89.0	79.9	89.0
CQL	120.7	140.2	99.2	120.0
	Midpoint (250K)			
Offline SAC	9.8	-0.5	-0.4	3.0
BC	110.8	124.9	191.2	142.3
CQL	132.7	184.7	244.3	187.3

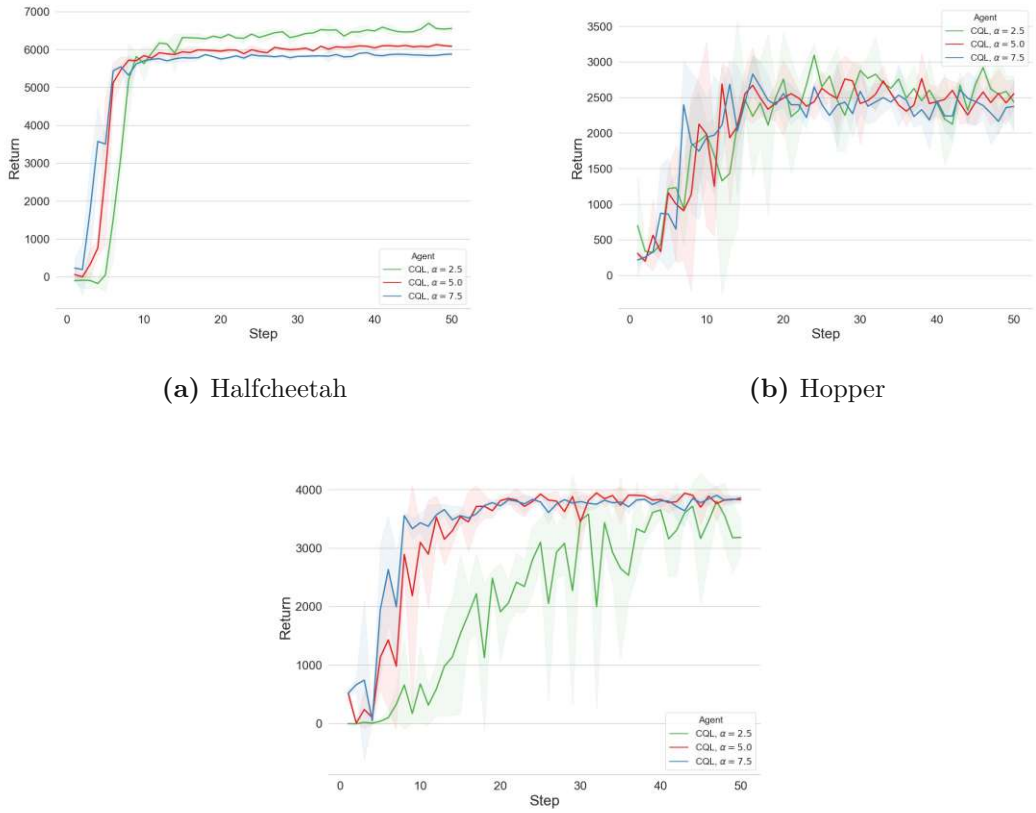
(b) Normalized scores

Table A.2: Raw and normalized scores for baseline agents on continuous control tasks. (a) Shows the mean ± standard deviation across the three random seeds, (b) shows the normalized scores 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 Bibliothek Your knowledge hub WIEN

A.2.2 CQL: Influence of α on Gym-MuJoCo

In this section, we present a preliminary ablation study on Gym-MuJoCo tasks to investigate the influence of the conservative weight parameter α on the learning performance of CQL. α determines how strongly the algorithm is penalized for overestimating the Q-values. A high value for α thus means strong penalization. We run CQL with $\alpha \in \{2.5, 5, 7.5\}$.



(a) Halfcheetah

(b) Hopper

(c) Walker2d

Figure A.2: Learning curves for CQL with variable α on continuous control tasks.

Learning curves and scores. The learning curves and raw/normalized scores at the final step for these experiments are shown in Figure A.2 and Table A.3, respectively. We note that the default value for α is 5.0 and all CQL curves/scores shown in Section 5.1 were obtained with this value. Overall, we observe that CQL with the default value of 5.0 for α performs best, achieving an average normalized score of 120% of the online SAC agent. In contrast, CQL with $\alpha = 2.5$ and $\alpha = 7.5$ only achieve 115% of the performance. The performance of the variants differs between the 3 tasks, however. The variant with small conservative weight ($\alpha = 2.5$) performs particularly well on the Halfcheetah task

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 Your knowledge hub WIEN

(green curve dominates red and blue curves) but performs poorly on the Walker2d task (high standard deviation, slow learning progress). This variant also exhibits the highest overall standard deviation of the scores across the three seeds. The contrary is true for the variant with $\alpha = 7.5$, which performs poorly on Halfcheetah, but well on Walker2d.

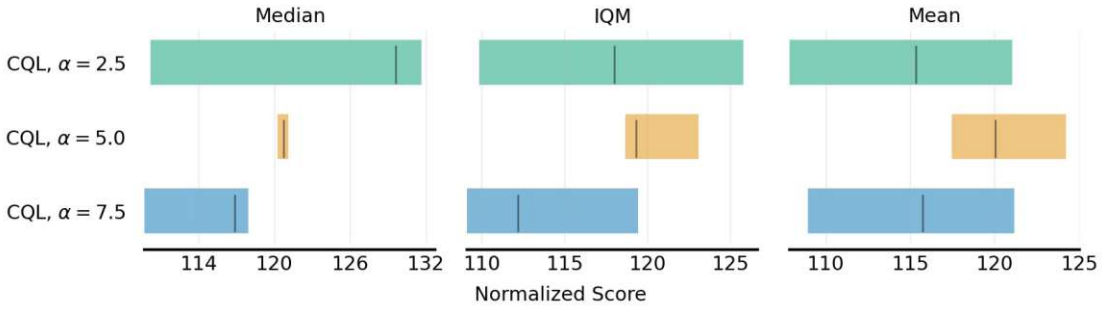
Agent	halfcheetah-medium-v2	hopper-medium-v2	walker2d-medium-v2
	Final step (500K)		
CQL, $\alpha = 2.5$	6557.4 ± 103.8	2439.2 ± 372.6	3185.1 ± 293.0
CQL, $\alpha = 5.0$	6088.3 ± 22.7	2553.9 ± 202.4	3829.8 ± 70.2
CQL, $\alpha = 7.5$	5883.6 ± 59.1	2375.8 ± 344.2	3860.0 ± 36.0

(a) Raw scores

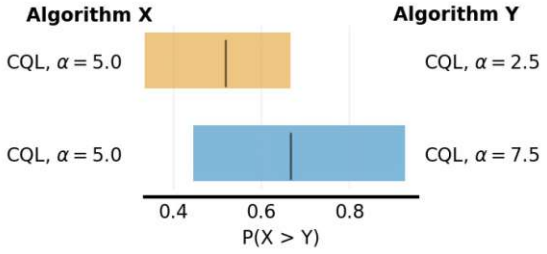
Agent	halfcheetah-medium-v2	hopper-medium-v2	walker2d-medium-v2	Mean
	Final step (500K)			
CQL, $\alpha = 2.5$	129.6	133.9	82.5	115.3
CQL, $\alpha = 5.0$	120.7	140.2	99.2	120.0
CQL, $\alpha = 7.5$	116.8	130.4	100.0	115.7

(b) Normalized scores

Table A.3: Raw and normalized scores for CQL with variable α on continuous control tasks.



(a) Interval estimates



(b) Probability of improvement

Figure A.3: Interval estimates of normalized scores and probability of improvement for CQL with variable α on continuous control tasks.

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 Your knowledge hub WIEN

Interval estimates. However, we do observe high standard deviations across the three seeds. In Figure A.3 we additionally show the interval estimates for median, IQM and mean, as well as the probability of improvement of $\alpha = 5.0$ over the other variants. We observe in Figure A.3 that the overall performance ranking remains the same for the median, IQM and the mean: the variant with $\alpha = 5.0$ performs best. Nevertheless, the intervals overlap strongly. The second graphic shows the probability of improvement of $\alpha = 5.0$ over $\alpha = 2.5$ and $\alpha = 7.5$. Indeed, we observe that $\alpha = 5.0$ only has a probability of improvement over $\alpha = 2.5$ of roughly 52%, with a very large confidence interval. The probability of improvement over $\alpha = 7.5$ is a bit larger (66%) but also exhibits a large confidence interval.

A.2.3 Discrete control: Atari

Agent	Breakout	Pong	QBert	Seaquest
Final step (2.5M)				
Random	0.8	-20	155	60
Offline DQN	4.9 ± 3.6	-18.5 ± 4.3	385.0 ± 96.6	414.7 ± 306.0
Online DQN	128.2 ± 63.6	20.9 ± 0.1	4416.7 ± 227.8	901.3 ± 778.2
BC	64.2 ± 12.1	-20.9 ± 0.1	4468.3 ± 7479.6	58.7 ± 101.6
CQL	167.3 ± 50.5	10.1 ± 4.9	12835.0 ± 1061.0	516.0 ± 132.2
Midpoint (1.25M)				
Random	0.8	-20	155	60
Offline DQN	4.5 ± 3.7	-13.5 ± 3.9	203.3 ± 75.2	1021.3 ± 382.2
Online DQN	122.3 ± 13.3	18.8 ± 1.8	2523.3 ± 828.7	182.7 ± 70.5
BC	116.4 ± 36.6	-21.0 ± 0.0	2783.3 ± 4561.1	180.0 ± 4.0
CQL	158.6 ± 44.5	17.5 ± 0.6	11088.3 ± 986.1	225.3 ± 29.5

(a) Raw scores

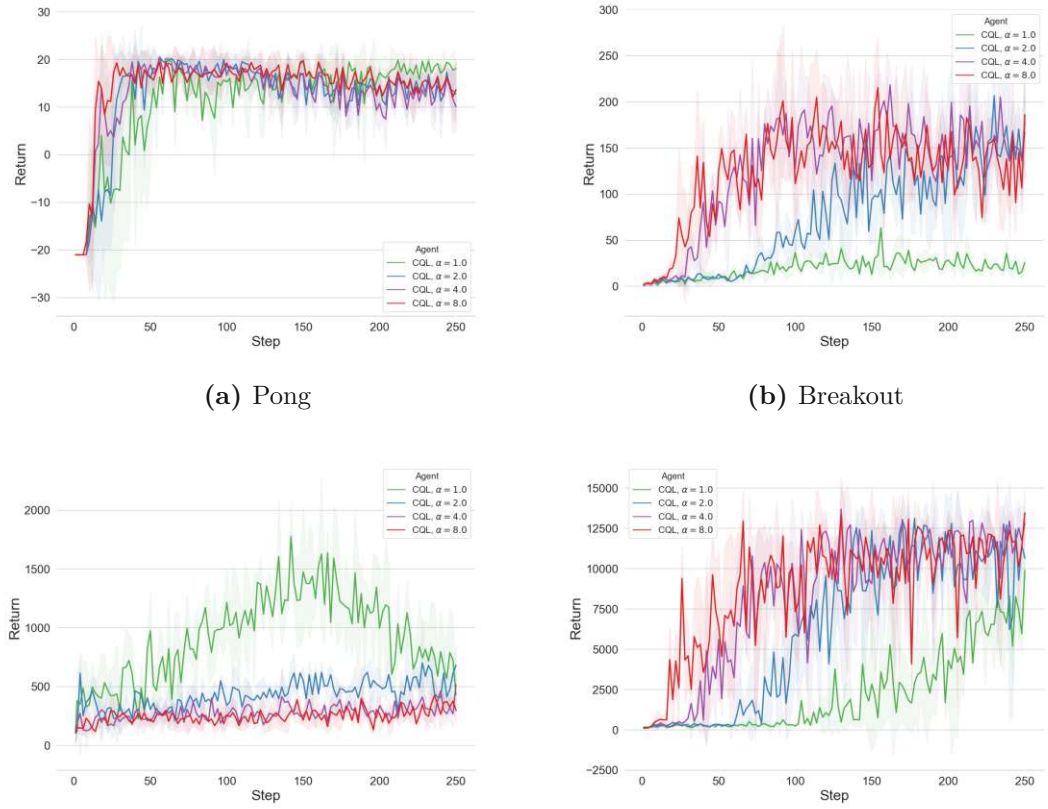
Agent	Breakout	Pong	QBert	Seaquest	Mean
Final step (2.5M)					
Offline DQN	3.2	3.6	5.4	42.2	13.6
BC	49.8	-2.3	101.2	-0.2	37.1
CQL	130.7	73.6	297.5	54.2	139.0
Midpoint (1.25M)					
Offline DQN	3.1	16.8	2.0	783.7	201.4
BC	95.1	-2.6	111.0	97.8	75.3
CQL	129.8	96.7	461.6	134.8	205.8

(b) Normalized scores

Table A.4: Raw and normalized scores for baseline agents on discrete control tasks.

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 Your knowledge hub WIEN

A.2.4 CQL: Influence of α on Atari



(a) Pong **(b)** Breakout

(c) Seaquest **(d)** QBert

Figure A.4: Learning curves for CQL with variable α agents on discrete control tasks.

Learning curves and scores. Again, we present a short ablation study on the influence of the conservative weight α on Atari games. We run CQL with $\alpha \in \{1.0, 2.0, 4.0, 8.0\}$ to see what value works best. The learning curves and raw/normalized scores of these experiments are shown in Figure A.4 and Table A.5. Overall, the scores differ strongly by game. On Pong $\alpha = 1.0$ performs best as the performance remains stable towards the end of training, while for all other variants the performance deteriorates towards the end of the training process. On Breakout, QBert and Breakout, $\alpha = 8.0$ clearly outperforms all other variants. On Seaquest $\alpha = 2.0$ obtains the highest score. However, overall, we observe that $\alpha = 4.0$ and $\alpha = 8.0$ deliver the best performance and the most stable learning curves across all games. Furthermore, on Seaquest we observe interesting behaviour for the CQL variant with $\alpha = 1.0$. Initially, the agent learns to play the game very well and improves until about 1.5M steps. However, afterwards, the performance quickly deteriorates. We previously observed such an "arc-shaped" learning curve in Figure 5.2 for offline DQN on Seaquest (even though with worse overall 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 Bibliothek Your knowledge hub WIEN

This performance curve indicates that the conservative regularization with $\alpha = 1.0$ was too small and led to strong over-estimation of the action-values. Thus, the quick deterioration of the agent's performance. In summary, the CQL variant with $\alpha = 8.0$ outperforms all other variants in terms of mean normalized score, achieving 142.1% of the expert agent's performance (online DQN). In contrast, $\alpha = 1.0$ and $\alpha = 2.0$ still outperform the online expert but showed clear signs of overestimation.

Agent	Breakout	Pong	QBert	Seaquest
		Final step (2.5M)		
CQL, $\alpha = 1.0$	25.7 ± 6.9	18.1 ± 1.9	9893.3 ± 2697.0	429.3 ± 99.6
CQL, $\alpha = 2.0$	177.5 ± 60.1	12.8 ± 6.5	10651.7 ± 1149.9	682.7 ± 160.8
CQL, $\alpha = 4.0$	167.3 ± 50.5	10.1 ± 4.9	12835.0 ± 1061.0	516.0 ± 132.2
CQL, $\alpha = 8.0$	186.1 ± 43.4	13.7 ± 6.2	13466.7 ± 1517.1	297.3 ± 102.3

(a) Raw scores

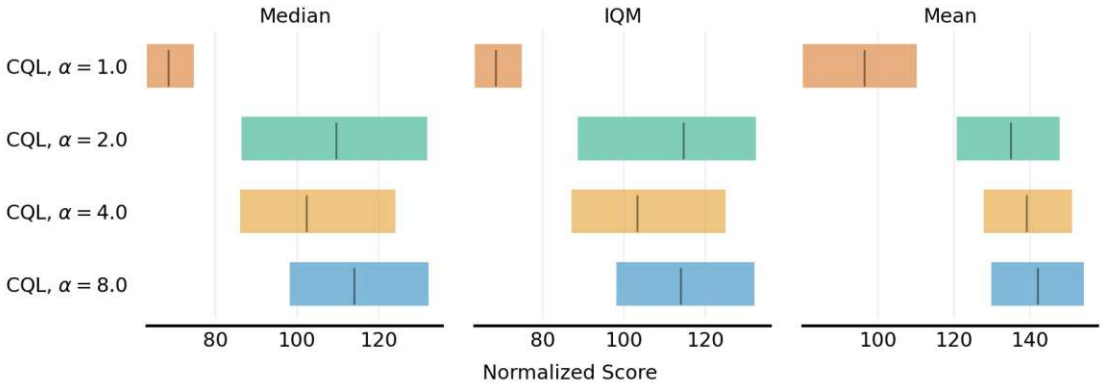
Agent	Breakout	Pong	QBert	Seaquest	Mean
		Final step (2.5M)			
CQL, $\alpha = 1.0$	19.6	93.2	228.5	43.9	96.3
CQL, $\alpha = 2.0$	138.7	80.1	246.3	74.0	134.8
CQL, $\alpha = 4.0$	130.7	73.6	297.5	54.2	139.0
CQL, $\alpha = 8.0$	145.4	82.2	312.4	28.2	142.1

(b) Normalized scores

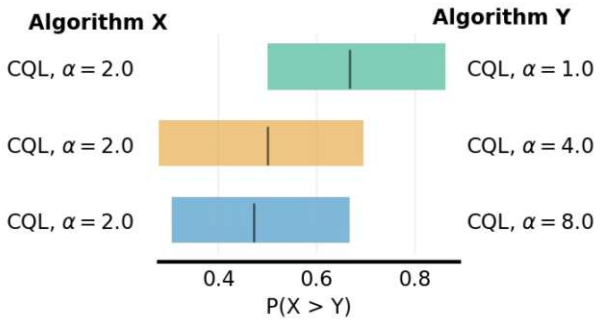
Table A.5: Raw and normalized scores for CQL with variable α on discrete control tasks.

Interval estimates. In addition, we also show the visual representations of the interval estimates and the probability of improvement in Figure A.5. If we consider the median and IQM instead of the mean normalized score, then $\alpha = 2.0$ should be considered the best variant. However, the confidence intervals are very wide and overlap strongly. If we consider the probability of improvement of $\alpha = 2.0$ over the other variants, we see that it exhibits a 50% chance of improving over $\alpha = 4.0$ and $\alpha = 8.0$. Thus, given our data, we cannot say that one variant is clearly better than all other variants. Therefore, we select the CQL variant with $\alpha = 4.0$ for all our subsequent experiments, as it is the default configuration in the original publication. $\alpha = 4.0$ was also used for the curves/scores in Figure 5.2 and Table 5.2 in Section 5.1.

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 Your knowledge hub WIEN



(a) Interval estimates



(b) Probability of improvement

Figure A.5: Interval estimates and probabilities of improvement for CQL with variable α on continuous control tasks.

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.

A.3 Data augmentations

In this section, we list the full results for our experiments with data augmentations. The results for continuous control tasks (Gym-MuJoCo) are listed in Tables A.6 and A.7. The results for discrete control tasks (Atari) are listed in Tables A.8 and A.9 and Figure A.6.

A.3.1 Continuous control: Gym-MuJoCo

Agent	halfcheetah-medium-v2	hopper-medium-v2	walker2d-medium-v2
	Final step (500K)		
CQL	6088.3 ± 22.7	2553.9 ± 202.4	3829.8 ± 70.2
Amp	5733.1 ± 82.5	2332.0 ± 356.2	3850.4 ± 36.4
Amp, same	5962.3 ± 41.0	2500.8 ± 221.7	3900.5 ± 59.1
Amp, $\alpha = 2.5$	5858.0 ± 38.9	2136.9 ± 131.0	3809.4 ± 256.8
Amp, $\alpha = 2.5$, same	6335.3 ± 28.5	2563.5 ± 81.4	3983.3 ± 96.7
Amp, min=0.5, max=1.5, $\alpha = 2.5$, same	5983.1 ± 32.7	2399.5 ± 217.6	4023.8 ± 66.6
Amp, $p_{aug} = 0.5$	5689.4 ± 50.0	2122.9 ± 263.2	3849.1 ± 29.9
Amp, $p_{aug} = 0.5$, same	6026.9 ± 69.0	2216.5 ± 167.8	3703.0 ± 309.6
Amp, $p_{aug} = 0.5$, $\alpha = 2.5$, same	6471.7 ± 42.6	2333.9 ± 518.6	3331.1 ± 429.1
GN	5668.8 ± 74.8	828.9 ± 815.2	3849.8 ± 21.6
GN, $\sigma = 0.01$	5947.9 ± 24.9	2262.3 ± 147.9	3920.2 ± 83.7
GN, same	5689.1 ± 27.4	515.6 ± 135.1	3948.5 ± 47.7
GN, $\sigma = 0.01$, same	5964.5 ± 38.3	2447.7 ± 305.3	3868.4 ± 82.5
GN, $\sigma = 0.01$, $\alpha = 2.5$	6292.6 ± 90.8	1700.7 ± 268.3	3476.8 ± 444.9
GN, $\sigma = 0.01$, $\alpha = 2.5$, same	6365.9 ± 19.0	2602.4 ± 236.4	3727.7 ± 543.3
GN, $p_{aug} = 0.5$	5696.9 ± 144.9	377.4 ± 66.0	3876.2 ± 56.5
GN, $p_{aug} = 0.5$, same	5702.2 ± 68.6	1215.7 ± 1206.9	3843.1 ± 219.0
GN, $\sigma = 0.01$, $p_{aug} = 0.5$, $\alpha = 2.5$, same	6382.0 ± 21.4	2474.6 ± 501.8	3629.8 ± 309.4
	Midpoint (250K)		
CQL	5948.1 ± 112.8	2630.9 ± 241.4	3924.8 ± 35.5
Amp	5580.3 ± 38.9	2700.0 ± 368.5	3822.7 ± 97.5
Amp, same	5958.0 ± 21.5	2526.3 ± 468.3	3836.5 ± 76.6
Amp, $\alpha = 2.5$	5671.4 ± 54.0	2159.6 ± 745.5	2916.6 ± 658.3
Amp, $\alpha = 2.5$, same	6233.0 ± 116.3	1564.3 ± 357.2	3004.4 ± 76.5
Amp, min=0.5, max=1.5, $\alpha = 2.5$, same	5897.0 ± 40.8	2105.9 ± 1124.4	3858.8 ± 136.4
Amp, $p_{aug} = 0.5$	5535.0 ± 19.2	2401.4 ± 284.3	3640.1 ± 486.9
Amp, $p_{aug} = 0.5$, same	5976.1 ± 79.6	2760.3 ± 377.8	3796.1 ± 117.0
Amp, $p_{aug} = 0.5$, $\alpha = 2.5$, same	6329.4 ± 32.6	2196.4 ± 252.1	3164.9 ± 578.8
GN	5252.2 ± 358.9	327.0 ± 24.2	3511.4 ± 416.6
GN, $\sigma = 0.01$	5875.1 ± 4.8	2314.5 ± 805.2	3754.2 ± 280.8
GN, same	5442.5 ± 97.8	342.0 ± 38.2	2580.2 ± 1158.7
GN, $\sigma = 0.01$, same	5908.7 ± 16.9	1647.8 ± 683.6	3727.8 ± 264.9
GN, $\sigma = 0.01$, $\alpha = 2.5$	6164.1 ± 65.9	1490.5 ± 1216.7	1854.7 ± 1598.0
GN, $\sigma = 0.01$, $\alpha = 2.5$, same	6162.6 ± 109.1	1490.3 ± 1114.1	2671.6 ± 893.5
GN, $p_{aug} = 0.5$	4687.0 ± 701.9	372.8 ± 38.2	3675.3 ± 238.0
GN, $p_{aug} = 0.5$, same	5446.3 ± 151.9	344.4 ± 18.9	3248.3 ± 503.3
GN, $\sigma = 0.01$, $p_{aug} = 0.5$, $\alpha = 2.5$, same	6235.5 ± 16.8	2458.8 ± 317.1	2777.7 ± 208.3

Table A.6: Raw scores for augmented agents on continuous control tasks.

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 Your knowledge hub WIEN

Agent	halfcheetah-medium-v2	hopper-medium-v2	walker2d-medium-v2	Mean
		Final step (500K)		
CQL	120.7	140.2	99.2	120.0
Amp	114.0	128.0	99.7	113.9
Amp, same	118.3	137.3	101.0	118.9
Amp, $\alpha = 2.5$	116.3	117.2	98.7	110.7
Amp, $\alpha = 2.5$, same	125.4	140.7	103.2	123.1
Amp, min=0.5, max=1.5, $\alpha = 2.5$, same	118.7	131.7	104.2	118.2
Amp, $p_{aug} = 0.5$	113.1	116.4	99.7	109.8
Amp, $p_{aug} = 0.5$, same	119.5	121.6	95.9	112.3
Amp, $p_{aug} = 0.5$, $\alpha = 2.5$, same	128.0	128.1	86.3	114.1
GN	112.8	45.1	99.7	85.8
GN, $\sigma = 0.01$	118.0	124.1	101.5	114.6
GN, same	113.1	27.8	102.3	81.1
GN, $\sigma = 0.01$, same	118.4	134.3	100.2	117.6
GN, $\sigma = 0.01$, $\alpha = 2.5$	124.6	93.1	90.0	102.6
GN, $\sigma = 0.01$, $\alpha = 2.5$, same	125.9	142.9	96.5	121.8
GN, $p_{aug} = 0.5$	113.3	20.2	100.4	78.0
GN, $p_{aug} = 0.5$, same	113.4	66.4	99.5	93.1
GN, $\sigma = 0.01$, $p_{aug} = 0.5$, $\alpha = 2.5$, same	126.3	135.8	94.0	118.7
		Midpoint (250K)		
CQL	132.7	184.7	244.3	187.3
Amp	124.9	189.6	238.0	184.2
Amp, same	132.9	177.4	238.8	183.0
Amp, $\alpha = 2.5$	126.8	151.5	181.6	153.3
Amp, $\alpha = 2.5$, same	138.8	109.5	187.0	145.1
Amp, min=0.5, max=1.5, $\alpha = 2.5$, same	131.6	147.7	240.2	173.2
Amp, $p_{aug} = 0.5$	123.9	168.5	226.6	173.0
Amp, $p_{aug} = 0.5$, same	133.3	193.9	236.3	187.8
Amp, $p_{aug} = 0.5$, $\alpha = 2.5$, same	140.8	154.1	197.0	164.0
GN	117.9	22.3	218.6	119.6
GN, $\sigma = 0.01$	131.2	162.4	233.7	175.8
GN, same	122.0	23.3	160.7	102.0
GN, $\sigma = 0.01$, same	131.9	115.4	232.1	159.8
GN, $\sigma = 0.01$, $\alpha = 2.5$	137.3	104.3	115.5	119.0
GN, $\sigma = 0.01$, $\alpha = 2.5$, same	137.3	104.3	166.3	136.0
GN, $p_{aug} = 0.5$	105.9	25.5	228.8	120.1
GN, $p_{aug} = 0.5$, same	122.0	23.5	202.2	115.9
GN, $\sigma = 0.01$, $p_{aug} = 0.5$, $\alpha = 2.5$, same	138.8	172.6	172.9	161.5

Table A.7: Normalized scores for augmented agents on continuous control tasks.

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 Your knowledge hub WIEN

A.3.2 Discrete control: Atari

Agent	Breakout	Pong	QBert	Seaquest
Final step (2.5M)				
CQL	167.3 ± 50.5	10.1 ± 4.9	12835.0 ± 1061.0	516.0 ± 132.2
Crop	150.3 ± 62.9	12.4 ± 4.7	10260.0 ± 1890.2	180.0 ± 156.5
Crop, same, no_c_aug	84.1 ± 28.5	15.4 ± 2.1	9373.3 ± 2305.1	450.7 ± 168.9
Crop, $k_{aug} = 2$, same, no_c_aug	166.9 ± 59.8	14.1 ± 4.2	12100.0 ± 688.9	392.0 ± 140.3
Crop, $\alpha = 2.0$, same	167.7 ± 33.5	13.9 ± 3.8	10381.7 ± 2314.0	430.7 ± 52.2
Crop, $\alpha = 2.0$, $k_{aug} = 2$, same, no_c_aug	204.9 ± 55.0	10.7 ± 7.8	9415.0 ± 3975.1	353.3 ± 100.0
Crop, $\alpha = 8.0$, same	170.6 ± 49.0	9.7 ± 8.6	10408.3 ± 2293.6	192.0 ± 69.4
Crop, $p_{aug} = 0.5$	232.4 ± 28.4	12.4 ± 6.0	11421.7 ± 1063.0	221.3 ± 81.7
Crop, $p_{aug} = 0.5$, same	140.9 ± 9.8	15.9 ± 7.2	12000.0 ± 339.1	316.0 ± 181.6
Crop, $p_{aug} = 0.5$, $k_{aug} = 2$, same, no_c_aug	196.8 ± 25.9	16.8 ± 2.8	11138.3 ± 887.3	342.7 ± 91.8
Crop, $p_{aug} = 0.5$, $\alpha = 2.0$, $k_{aug} = 2$, same, no_c_aug	166.0 ± 86.3	16.5 ± 5.0	10971.7 ± 1296.8	389.3 ± 6.1
Shift	145.7 ± 50.7	11.5 ± 4.7	11485.0 ± 506.9	161.3 ± 32.3
Shift, same	144.5 ± 93.9	16.6 ± 6.6	8616.7 ± 3366.4	198.7 ± 54.0
Shift, $k_{aug} = 2$, same	183.3 ± 40.1	9.7 ± 5.6	10106.7 ± 1539.9	206.7 ± 30.3
Shift, $\alpha = 2.0$, $k_{aug} = 2$, same	116.9 ± 31.6	18.9 ± 0.5	8181.7 ± 5426.1	260.0 ± 46.1
Shift, $p_{aug} = 0.5$	177.6 ± 63.7	10.2 ± 8.5	8673.3 ± 4063.0	230.7 ± 40.3
Shift, $p_{aug} = 0.5$, same	191.3 ± 52.3	16.3 ± 6.3	8860.0 ± 3169.7	205.3 ± 92.7
Cut	239.6 ± 34.2	14.6 ± 4.8	9208.3 ± 2280.9	285.3 ± 135.4
Cut, same	146.1 ± 65.7	12.2 ± 6.9	11075.0 ± 1864.5	205.3 ± 77.7
Cut, $p_{aug} = 0.5$	163.1 ± 44.0	15.2 ± 3.2	11770.0 ± 30.4	298.7 ± 140.1
Cut, $p_{aug} = 0.5$, same	168.6 ± 4.9	15.3 ± 3.6	10440.0 ± 1676.0	305.3 ± 104.1
Midpoint (1.25M)				
CQL	158.6 ± 44.5	17.5 ± 0.6	11088.3 ± 986.1	225.3 ± 29.5
Crop	174.9 ± 47.2	15.0 ± 5.0	11191.7 ± 876.9	202.7 ± 194.6
Crop, same, no_c_aug	164.0 ± 50.2	14.6 ± 3.3	8978.3 ± 2326.2	306.7 ± 112.5
Crop, $k_{aug} = 2$, same, no_c_aug	138.7 ± 45.6	15.5 ± 2.9	7816.7 ± 1547.6	225.3 ± 40.5
Crop, $\alpha = 2.0$, same	119.7 ± 54.6	18.7 ± 0.9	6563.3 ± 3231.8	320.0 ± 62.9
Crop, $\alpha = 2.0$, $k_{aug} = 2$, same, no_c_aug	208.8 ± 68.9	17.5 ± 3.4	10965.0 ± 867.5	253.3 ± 67.2
Crop, $\alpha = 8.0$, same	182.1 ± 82.8	15.5 ± 7.8	10260.0 ± 1264.4	136.0 ± 14.4
Crop, $p_{aug} = 0.5$	146.7 ± 73.1	19.3 ± 1.2	11243.3 ± 908.2	246.7 ± 34.0
Crop, $p_{aug} = 0.5$, same	203.7 ± 79.1	19.7 ± 0.8	9253.3 ± 3935.9	228.0 ± 44.5
Crop, $p_{aug} = 0.5$, $k_{aug} = 2$, same, no_c_aug	220.8 ± 5.8	18.7 ± 1.6	9600.0 ± 4112.3	216.0 ± 24.3
Crop, $p_{aug} = 0.5$, $\alpha = 2.0$, $k_{aug} = 2$, same, no_c_aug	78.9 ± 47.8	18.1 ± 1.1	7985.0 ± 4462.6	309.3 ± 173.0
Shift	87.3 ± 16.0	13.5 ± 8.5	8850.0 ± 2415.8	141.3 ± 20.1
Shift, same	174.6 ± 70.8	8.1 ± 18.3	8196.7 ± 2917.7	181.3 ± 90.0
Shift, $k_{aug} = 2$, same	139.3 ± 50.5	15.8 ± 4.5	8516.7 ± 1123.7	180.0 ± 55.6
Shift, $\alpha = 2.0$, $k_{aug} = 2$, same	86.9 ± 40.3	18.1 ± 4.4	5598.3 ± 2795.8	270.7 ± 62.3
Shift, $p_{aug} = 0.5$	172.5 ± 54.4	18.7 ± 1.5	11690.0 ± 1586.8	160.0 ± 10.6
Shift, $p_{aug} = 0.5$, same	149.1 ± 79.2	15.5 ± 5.3	6930.0 ± 1542.0	146.7 ± 26.6
Cut	222.9 ± 54.0	17.1 ± 1.9	8975.0 ± 1550.5	268.0 ± 142.0
Cut, same	149.9 ± 92.4	16.8 ± 1.8	10321.7 ± 3500.0	216.0 ± 80.3
Cut, $p_{aug} = 0.5$	136.5 ± 57.7	16.3 ± 3.0	11780.0 ± 26.5	288.0 ± 161.1
Cut, $p_{aug} = 0.5$, same	122.9 ± 21.8	16.4 ± 2.1	9170.0 ± 2666.3	284.0 ± 69.7

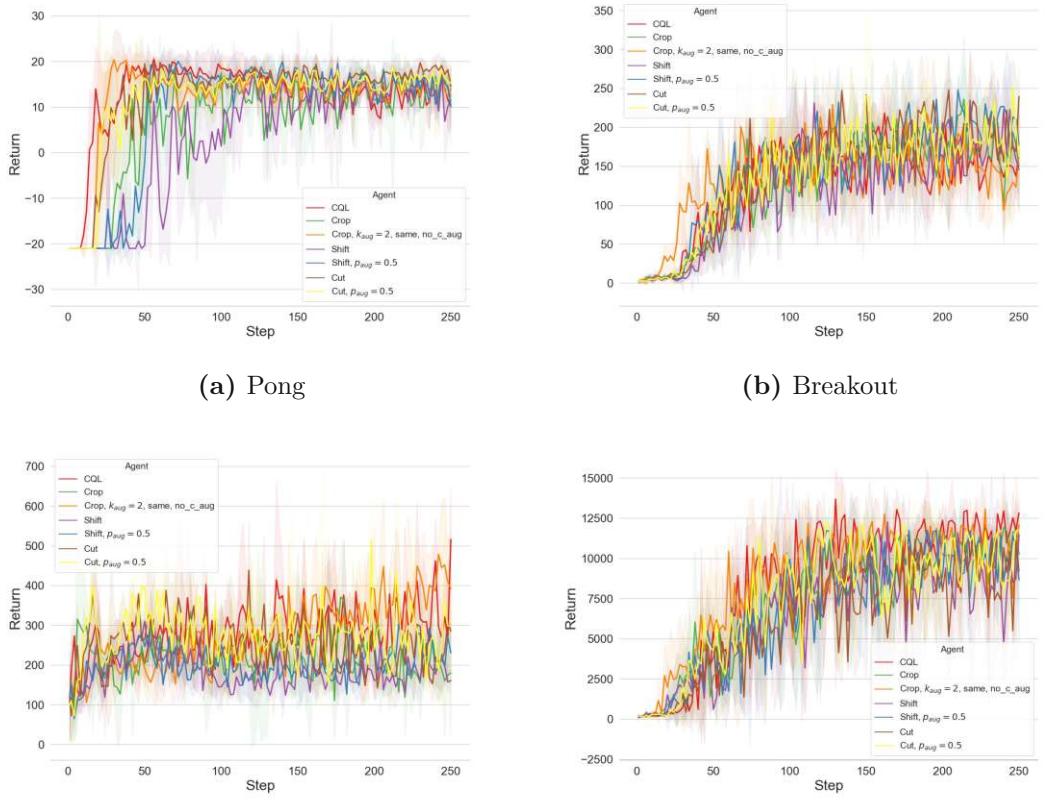
Table A.8: Raw scores for augmented CQL on discrete control tasks.

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 Your knowledge hub WIEN

Agent	Breakout	Pong	QBert	Seaquest	Mean
Final step (2.5M)					
CQL	130.7	73.6	297.5	54.2	139.0
Crop	117.3	79.2	237.1	14.3	112.0
Crop, same, no_c_aug	65.4	86.5	216.3	46.4	103.6
Crop, $k_{aug} = 2$, same, no_c_aug	130.4	83.4	280.3	39.5	133.4
Crop, $\alpha = 2.0$, same	131.0	82.7	240.0	44.1	124.4
Crop, $\alpha = 2.0$, $k_{aug} = 2$, same, no_c_aug	160.2	75.1	217.3	34.9	121.9
Crop, $\alpha = 8.0$, same	133.3	72.5	240.6	15.7	115.5
Crop, $p_{aug} = 0.5$	181.8	79.2	264.4	19.2	136.1
Crop, $p_{aug} = 0.5$, same	109.9	87.6	277.9	30.4	126.5
Crop, $p_{aug} = 0.5$, $k_{aug} = 2$, same, no_c_aug	153.8	89.9	257.7	33.6	133.8
Crop, $p_{aug} = 0.5$, $\alpha = 2.0$, $k_{aug} = 2$, same, no_c_aug	129.7	89.1	253.8	39.1	127.9
Shift	113.7	76.9	265.9	12.0	117.1
Shift, same	112.8	89.4	198.6	16.5	104.3
Shift, $k_{aug} = 2$, same	143.3	72.6	233.5	17.4	116.7
Shift, $\alpha = 2.0$, $k_{aug} = 2$, same	91.1	95.1	188.3	23.8	99.6
Shift, $p_{aug} = 0.5$	138.8	73.8	199.9	20.3	108.2
Shift, $p_{aug} = 0.5$, same	149.5	88.6	204.3	17.3	114.9
Cut	187.4	84.5	212.4	26.8	127.8
Cut, same	114.1	78.7	256.2	17.3	116.6
Cut, $p_{aug} = 0.5$	127.4	86.0	272.5	28.4	128.6
Cut, $p_{aug} = 0.5$, same	131.7	86.3	241.3	29.2	122.1
Midpoint (1.25M)					
CQL	129.8	96.7	461.6	134.8	205.8
Crop	143.3	90.2	466.0	116.3	204.0
Crop, same, no_c_aug	134.3	89.2	372.6	201.1	199.3
Crop, $k_{aug} = 2$, same, no_c_aug	113.4	91.4	323.5	134.8	165.8
Crop, $\alpha = 2.0$, same	97.9	99.8	270.6	212.0	170.1
Crop, $\alpha = 2.0$, $k_{aug} = 2$, same, no_c_aug	171.1	96.6	456.4	157.6	220.4
Crop, $\alpha = 8.0$, same	149.2	91.6	426.7	62.0	182.3
Crop, $p_{aug} = 0.5$	120.0	101.4	468.2	152.2	210.4
Crop, $p_{aug} = 0.5$, same	166.9	102.4	384.2	137.0	197.6
Crop, $p_{aug} = 0.5$, $k_{aug} = 2$, same, no_c_aug	181.0	99.7	398.8	127.2	201.7
Crop, $p_{aug} = 0.5$, $\alpha = 2.0$, $k_{aug} = 2$, same, no_c_aug	64.2	98.1	330.6	203.3	174.1
Shift	71.1	86.3	367.1	66.3	147.7
Shift, same	143.0	72.3	339.5	98.9	163.5
Shift, $k_{aug} = 2$, same	113.9	92.3	353.1	97.8	164.3
Shift, $\alpha = 2.0$, $k_{aug} = 2$, same	70.8	98.3	229.8	171.7	142.7
Shift, $p_{aug} = 0.5$	141.3	99.8	487.1	81.5	202.4
Shift, $p_{aug} = 0.5$, same	122.1	91.4	286.1	70.7	142.5
Cut	182.8	95.5	372.4	169.6	205.1
Cut, same	122.7	94.8	429.3	127.2	193.5
Cut, $p_{aug} = 0.5$	111.7	93.5	490.9	185.9	220.5
Cut, $p_{aug} = 0.5$, same	100.5	93.8	380.6	182.6	189.4

Table A.9: Normalized scores for augmented CQL on discrete control tasks.

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 Your knowledge hub WIEN



(a) Pong

(b) Breakout

(c) Seaquest

(d) QBert

Figure A.6: Learning curves for augmented CQL with discrete control tasks.

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 Your knowledge hub WIEN

A.4 Self-supervised tasks

In this section, we list the full results for our experiments with data augmentations. The results for continuous control tasks (Gym-MuJoCo) are listed in Tables A.10, A.11, A.12 and A.13. The results for discrete control tasks (Atari) are listed in Tables A.16 and A.17 and Figure A.7 as well as A.8.

A.4.1 Continuous control: Gym-MuJoCo

Agent	halfcheetah-medium-v2	hopper-medium-v2	walker2d-medium-v2
	Final step (500K)		
CQL	6088.3 ± 22.7	2553.9 ± 202.4	3829.8 ± 70.2
CURL, LN	6088.6 ± 12.0	2210.0 ± 58.2	3898.4 ± 48.8
CURL, BN	6075.6 ± 41.6	2273.8 ± 98.8	3900.5 ± 58.6
CURL, $\alpha = 2.5$, BN	6463.8 ± 70.5	2753.6 ± 278.8	4146.8 ± 52.5
CURL, $\alpha = 2.5$, LN	6465.7 ± 63.7	2303.9 ± 273.6	3408.4 ± 1123.7
CURL, Amp, same	5882.2 ± 58.7	2400.9 ± 464.0	3818.2 ± 111.7
CURL, Amp, same, BN	5956.0 ± 101.6	2292.5 ± 252.5	3870.1 ± 6.3
CURL, Amp, same, LN	5922.9 ± 32.7	2455.6 ± 106.8	3879.7 ± 42.6
CURL, Amp, $\alpha = 2.5$, same	6160.6 ± 90.3	2188.7 ± 1304.4	3727.3 ± 434.3
CURL, Amp, $\alpha = 2.5$, same, BN	6184.2 ± 67.0	2256.5 ± 245.5	3651.6 ± 461.7
CURL, Amp, $\alpha = 2.5$, same, LN	6291.5 ± 67.0	2411.4 ± 668.3	3132.6 ± 713.6
CURL, GN, $\sigma = 0.01$, $\alpha = 2.5$, same	6163.2 ± 35.2	1139.7 ± 1377.0	3174.7 ± 1298.6
CURL, GN, $\sigma = 0.01$, $\alpha = 2.5$, same, LN	6328.8 ± 60.1	2111.5 ± 263.4	3702.6 ± 252.7
	Midpoint (250K)		
CQL	5948.1 ± 112.8	2630.9 ± 241.4	3924.8 ± 35.5
CURL, LN	5959.5 ± 70.1	2318.1 ± 334.9	3913.1 ± 97.7
CURL, BN	5934.9 ± 86.6	2561.6 ± 342.6	3864.8 ± 76.7
CURL, $\alpha = 2.5$, BN	6157.9 ± 136.5	2542.9 ± 229.6	2593.8 ± 1424.2
CURL, $\alpha = 2.5$, LN	6294.3 ± 63.7	2827.9 ± 175.8	2665.4 ± 783.9
CURL, Amp, same	5771.0 ± 26.8	2521.8 ± 371.1	3555.8 ± 334.1
CURL, Amp, same, BN	5788.6 ± 82.8	2481.4 ± 297.7	3849.8 ± 177.4
CURL, Amp, same, LN	5941.7 ± 178.5	2594.5 ± 417.2	3877.2 ± 90.8
CURL, Amp, $\alpha = 2.5$, same	6009.8 ± 34.5	1245.5 ± 522.9	2569.2 ± 1829.2
CURL, Amp, $\alpha = 2.5$, same, BN	5958.0 ± 56.0	2668.3 ± 181.4	3819.4 ± 139.6
CURL, Amp, $\alpha = 2.5$, same, LN	6135.0 ± 73.0	2803.1 ± 390.9	3194.2 ± 683.6
CURL, GN, $\sigma = 0.01$, $\alpha = 2.5$, same	5968.8 ± 55.6	881.4 ± 855.3	1224.4 ± 1335.0
CURL, GN, $\sigma = 0.01$, $\alpha = 2.5$, same, LN	6136.4 ± 127.1	2030.8 ± 1399.6	2369.4 ± 1046.5

Table A.10: Raw scores for CQL + Curl on continuous control tasks.

A.4.2 Discrete control: Atari

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.

Agent	halfcheetah-medium-v2	hopper-medium-v2	walker2d-medium-v2	Mean
		Final step (500K)		
CQL	120.7	140.2	99.2	120.0
CURL, LN	120.7	121.2	101.0	114.3
CURL, BN	120.5	124.7	101.0	115.4
CURL, $\alpha = 2.5$, BN	127.8	151.2	107.4	128.8
CURL, $\alpha = 2.5$, LN	127.8	126.4	88.3	114.2
CURL, Amp, same	116.8	131.8	98.9	115.8
CURL, Amp, same, BN	118.2	125.8	100.2	114.7
CURL, Amp, same, LN	117.6	134.8	100.5	117.6
CURL, Amp, $\alpha = 2.5$, same	122.1	120.1	96.5	112.9
CURL, Amp, $\alpha = 2.5$, same, BN	122.5	123.8	94.6	113.6
CURL, Amp, $\alpha = 2.5$, same, LN	124.5	132.3	81.1	112.7
CURL, GN, $\sigma = 0.01$, $\alpha = 2.5$, same	122.1	62.2	82.2	88.9
CURL, GN, $\sigma = 0.01$, $\alpha = 2.5$, same, LN	125.2	115.8	95.9	112.3
		Midpoint (250K)		
CQL	132.7	184.7	244.3	187.3
CURL, LN	133.0	162.7	243.6	179.7
CURL, BN	132.4	179.8	240.6	184.3
CURL, $\alpha = 2.5$, BN	137.2	178.5	161.5	159.1
CURL, $\alpha = 2.5$, LN	140.1	198.6	166.0	168.2
CURL, Amp, same	129.0	177.0	221.4	175.8
CURL, Amp, same, BN	129.3	174.2	239.7	181.1
CURL, Amp, same, LN	132.6	182.2	241.4	185.4
CURL, Amp, $\alpha = 2.5$, same	134.0	87.0	160.0	127.0
CURL, Amp, $\alpha = 2.5$, same, BN	132.9	187.4	237.8	186.0
CURL, Amp, $\alpha = 2.5$, same, LN	136.7	196.9	198.9	177.5
CURL, GN, $\sigma = 0.01$, $\alpha = 2.5$, same	133.2	61.4	76.3	90.3
CURL, GN, $\sigma = 0.01$, $\alpha = 2.5$, same, LN	136.7	142.4	147.5	142.2

Table A.11: Normalized scores for CQL + Curl on continuous control tasks.

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 Your knowledge hub WIEN

Agent	halfcheetah-medium-v2	hopper-medium-v2	walker2d-medium-v2
	Final step (500K)		
CQL	6088.3 ± 22.7	2553.9 ± 202.4	3829.8 ± 70.2
SPR, BN	6027.0 ± 46.1	2524.9 ± 226.8	3903.0 ± 23.7
SPR, LN	6112.9 ± 56.8	2442.0 ± 347.7	3909.3 ± 25.8
SPR, Re-norm	6138.5 ± 129.2	2616.2 ± 459.0	3884.4 ± 40.5
SPR, $\alpha = 2.5$, BN	6529.5 ± 66.2	2394.3 ± 262.8	3152.6 ± 538.8
SPR, $\alpha = 2.5$, Re-norm	6489.7 ± 90.5	2444.6 ± 497.8	2950.3 ± 784.5
SPR, Amp, same, BN	5930.6 ± 28.9	2202.2 ± 210.0	3702.1 ± 279.0
SPR, Amp, same, LN	5943.7 ± 19.3	2525.1 ± 216.5	3939.0 ± 35.4
SPR, Amp, same, Re-norm	5997.2 ± 32.4	2352.1 ± 174.8	3863.4 ± 62.0
SPR, Amp, $\alpha = 2.5$, same, BN	6252.5 ± 39.1	2675.8 ± 126.0	3977.2 ± 80.2
SPR, Amp, $\alpha = 2.5$, same, LN	6345.5 ± 14.0	2494.2 ± 200.6	3118.1 ± 721.7
SPR, Amp, $\alpha = 2.5$, same, Re-norm	6316.8 ± 63.9	2728.2 ± 193.1	3636.7 ± 455.4
SPR, GN, $\sigma = 0.01$, $\alpha = 2.5$, same, BN	6316.7 ± 72.3	1914.7 ± 391.6	3724.2 ± 496.5
	Midpoint (250K)		
CQL	5948.1 ± 112.8	2630.9 ± 241.4	3924.8 ± 35.5
SPR, BN	5985.2 ± 30.3	2070.4 ± 581.1	3798.2 ± 70.0
SPR, LN	5976.5 ± 106.0	2349.7 ± 135.7	3763.6 ± 104.5
SPR, Re-norm	5979.2 ± 164.9	2587.5 ± 95.3	3901.6 ± 66.8
SPR, $\alpha = 2.5$, BN	6317.8 ± 62.9	2270.9 ± 748.0	3567.4 ± 297.7
SPR, $\alpha = 2.5$, Re-norm	6272.9 ± 114.2	2637.5 ± 245.1	2174.9 ± 245.5
SPR, Amp, same, BN	5984.8 ± 59.0	2422.0 ± 463.4	3682.7 ± 278.5
SPR, Amp, same, LN	5937.2 ± 77.3	2596.4 ± 18.3	3751.1 ± 33.5
SPR, Amp, same, Re-norm	5986.8 ± 99.1	2725.3 ± 230.6	3792.7 ± 29.0
SPR, Amp, $\alpha = 2.5$, same, BN	6185.3 ± 39.5	2451.1 ± 499.8	3021.1 ± 554.8
SPR, Amp, $\alpha = 2.5$, same, LN	6183.1 ± 44.4	2098.6 ± 59.4	3042.0 ± 176.5
SPR, Amp, $\alpha = 2.5$, same, Re-norm	6229.2 ± 72.7	2318.3 ± 798.5	3655.8 ± 171.0
SPR, GN, $\sigma = 0.01$, $\alpha = 2.5$, same, BN	6175.9 ± 124.5	693.5 ± 329.4	3017.1 ± 875.3

Table A.12: Raw scores for CQL + SPR on continuous control tasks.

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.

Agent	halfcheetah-medium-v2	hopper-medium-v2	walker2d-medium-v2	Mean
Final step (500K)				
CQL	120.7	140.2	99.2	120.0
SPR, BN	119.5	138.6	101.1	119.7
SPR, LN	121.2	134.0	101.2	118.8
SPR, Re-norm	121.6	143.6	100.6	122.0
SPR, $\alpha = 2.5$, BN	129.0	131.4	81.7	114.0
SPR, $\alpha = 2.5$, Re-norm	128.3	134.2	76.4	113.0
SPR, Amp, same, BN	117.7	120.8	95.9	111.5
SPR, Amp, same, LN	118.0	138.6	102.0	119.5
SPR, Amp, same, Re-norm	119.0	129.1	100.1	116.0
SPR, Amp, $\alpha = 2.5$, same, BN	123.8	146.9	103.0	124.6
SPR, Amp, $\alpha = 2.5$, same, LN	125.6	136.9	80.8	114.4
SPR, Amp, $\alpha = 2.5$, same, Re-norm	125.0	149.8	94.2	123.0
SPR, GN, $\sigma = 0.01$, $\alpha = 2.5$, same, BN	125.0	104.9	96.4	108.8
Midpoint (250K)				
CQL	132.7	184.7	244.3	187.3
SPR, BN	133.5	145.2	236.4	171.7
SPR, LN	133.3	164.9	234.3	177.5
SPR, Re-norm	133.4	181.7	242.9	186.0
SPR, $\alpha = 2.5$, BN	140.6	159.3	222.1	174.0
SPR, $\alpha = 2.5$, Re-norm	139.6	185.2	135.4	153.4
SPR, Amp, same, BN	133.5	170.0	229.3	177.6
SPR, Amp, same, LN	132.5	182.3	233.5	182.8
SPR, Amp, same, Re-norm	133.5	191.4	236.1	187.0
SPR, Amp, $\alpha = 2.5$, same, BN	137.8	172.1	188.1	166.0
SPR, Amp, $\alpha = 2.5$, same, LN	137.7	147.2	189.4	158.1
SPR, Amp, $\alpha = 2.5$, same, Re-norm	138.7	162.7	227.6	176.3
SPR, GN, $\sigma = 0.01$, $\alpha = 2.5$, same, BN	137.6	48.1	187.8	124.5

Table A.13: Normalized scores for CQL + SPR on continuous control tasks.

Agent	halfcheetah-medium-v2	hopper-medium-v2	walker2d-medium-v2
Final step (500K)			
SGI, BN	6133.8 ± 107.9	2439.0 ± 244.0	3778.2 ± 47.8
SGI, Re-norm	6180.3 ± 23.7	2358.7 ± 139.2	3701.9 ± 49.2
SGI, Amp, same, BN	6068.4 ± 57.2	2491.6 ± 90.2	3773.8 ± 79.4
SGI, Amp, same, Re-norm	6137.1 ± 35.5	2688.5 ± 290.5	3628.6 ± 22.8
SGI, $\alpha = 2.5$, BN	6575.9 ± 114.7	2672.1 ± 208.2	3936.4 ± 102.3
SGI, $\alpha = 2.5$, Re-norm	6652.4 ± 23.3	2569.8 ± 278.6	3949.7 ± 114.0
SGI, Amp, $\alpha = 2.5$, same, BN	6410.7 ± 90.3	2772.2 ± 312.9	3834.6 ± 112.4
SGI, Amp, $\alpha = 2.5$, same, Re-norm	6550.8 ± 44.6	2681.6 ± 436.9	3859.9 ± 136.6
Midpoint (250K)			
SGI, BN	6013.1 ± 25.5	2212.9 ± 310.2	3754.3 ± 53.8
SGI, Re-norm	6119.0 ± 27.8	2553.8 ± 216.4	3690.3 ± 182.4
SGI, Amp, same, BN	6084.0 ± 80.8	2558.9 ± 273.2	3694.6 ± 45.0
SGI, Amp, same, Re-norm	6047.2 ± 61.0	2613.6 ± 406.0	3469.4 ± 104.9
SGI, $\alpha = 2.5$, BN	6431.0 ± 90.8	2320.4 ± 162.4	3787.6 ± 74.6
SGI, $\alpha = 2.5$, Re-norm	6580.0 ± 29.3	3083.9 ± 103.8	3704.1 ± 102.4
SGI, Amp, $\alpha = 2.5$, same, BN	6305.5 ± 82.8	2832.0 ± 178.6	3766.0 ± 102.2
SGI, Amp, $\alpha = 2.5$, same, Re-norm	6517.1 ± 79.9	3034.6 ± 24.1	2416.2 ± 2040.3

Table A.14: Raw scores for CQL + SGI on continuous control tasks.

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 Your knowledge hub WIEN

Agent	halfcheetah-medium-v2	hopper-medium-v2	walker2d-medium-v2	Mean
Final step (500K)				
SGI, BN	121.6	133.9	97.8	117.8
SGI, Re-norm	122.4	129.4	95.9	115.9
SGI, Amp, same, BN	120.3	136.7	97.7	118.3
SGI, Amp, same, Re-norm	121.6	147.6	94.0	121.1
SGI, $\alpha = 2.5$, BN	129.9	146.7	101.9	126.2
SGI, $\alpha = 2.5$, Re-norm	131.4	141.1	102.3	124.9
SGI, Amp, $\alpha = 2.5$, same, BN	126.8	152.2	99.3	126.1
SGI, Amp, $\alpha = 2.5$, same, Re-norm	129.4	147.2	100.0	125.5
Midpoint (250K)				
SGI, BN	134.1	155.3	233.7	174.4
SGI, Re-norm	136.4	179.3	229.7	181.8
SGI, Amp, same, BN	135.6	179.7	230.0	181.8
SGI, Amp, same, Re-norm	134.8	183.5	216.0	178.1
SGI, $\alpha = 2.5$, BN	143.0	162.8	235.8	180.5
SGI, $\alpha = 2.5$, Re-norm	146.2	216.7	230.6	197.8
SGI, Amp, $\alpha = 2.5$, same, BN	140.3	198.9	234.4	191.2
SGI, Amp, $\alpha = 2.5$, same, Re-norm	144.8	213.2	150.4	169.5

Table A.15: Normalized scores for CQL + SGI on continuous control tasks.

Agent	Breakout	Pong	QBert	Seaquest
Final step (2.5M)				
CQL	167.3 ± 50.5	10.1 ± 4.9	12835.0 ± 1061.0	516.0 ± 132.2
CURL, LN	117.8 ± 99.1	12.0 ± 7.6	10676.7 ± 2392.9	254.7 ± 104.4
CURL, $\alpha = 2.0$, LN	128.5 ± 41.6	13.5 ± 6.4	10260.0 ± 2628.4	344.0 ± 124.2
CURL, Crop, LN	194.3 ± 83.3	7.2 ± 6.6	8233.3 ± 4371.1	176.0 ± 72.8
CURL, Crop, same, LN	127.5 ± 53.9	12.9 ± 4.3	10420.0 ± 1149.5	237.3 ± 92.7
CURL, Crop, no_c_aug, LN	178.8 ± 59.1	14.7 ± 4.0	9910.0 ± 2011.1	218.7 ± 66.0
CURL, Crop, $\alpha = 2.0$, same, LN	192.5 ± 87.0	15.5 ± 7.9	9415.0 ± 1449.7	294.7 ± 151.5
SPR, Re-norm	165.3 ± 46.8	15.7 ± 6.1	12640.0 ± 2216.6	281.3 ± 161.1
SPR, Crop, same, Re-norm	171.8 ± 29.5	14.1 ± 9.4	8698.3 ± 3431.5	190.7 ± 22.7
Midpoint (1.25M)				
CQL	158.6 ± 44.5	17.5 ± 0.6	11088.3 ± 986.1	225.3 ± 29.5
CURL, LN	90.7 ± 36.4	18.0 ± 2.8	7500.0 ± 1265.2	221.3 ± 54.6
CURL, $\alpha = 2.0$, LN	43.7 ± 10.8	19.7 ± 1.1	4748.3 ± 484.3	292.0 ± 101.4
CURL, Crop, LN	148.1 ± 33.0	11.9 ± 8.2	8188.3 ± 2774.6	188.0 ± 49.2
CURL, Crop, same, LN	121.9 ± 68.6	18.1 ± 1.8	8296.7 ± 3535.1	205.3 ± 6.1
CURL, Crop, no_c_aug, LN	137.9 ± 22.6	19.1 ± 1.9	8495.0 ± 3476.7	172.0 ± 76.3
CURL, Crop, $\alpha = 2.0$, same, LN	57.1 ± 40.9	19.7 ± 1.1	5315.0 ± 448.9	244.0 ± 62.9
SPR, Re-norm	150.7 ± 53.7	20.2 ± 0.5	8940.0 ± 3496.5	288.0 ± 50.0
SPR, Crop, same, Re-norm	119.3 ± 33.0	20.8 ± 0.0	7090.0 ± 1140.4	202.7 ± 84.3

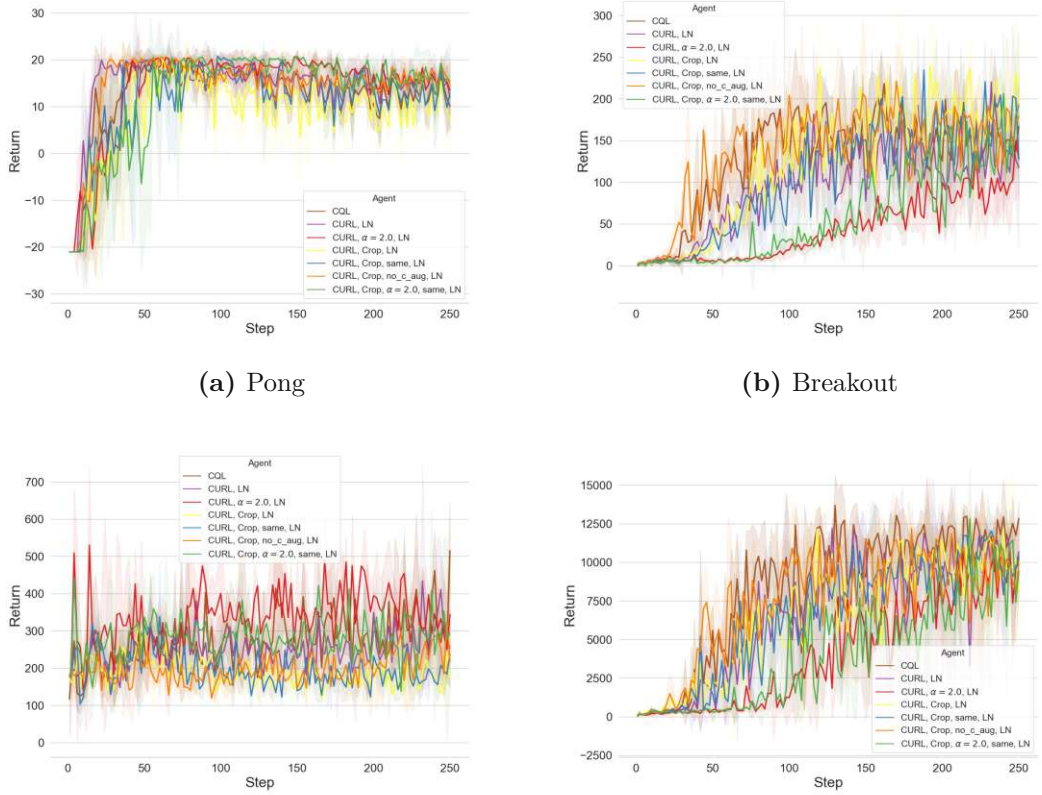
Table A.16: Raw scores for self-supervised agents on discrete control tasks.

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 Your knowledge hub WIEN

Agent	Breakout	Pong	QBert	Seaquest	Mean
Final step (2.5M)					
CQL	130.7	73.6	297.5	54.2	139.0
CURL, LN	91.8	78.2	246.9	23.1	110.0
CURL, $\alpha = 2.0$, LN	100.3	81.9	237.1	33.8	113.3
CURL, Crop, LN	151.9	66.5	189.6	13.8	105.4
CURL, Crop, same, LN	99.4	80.3	240.9	21.1	110.4
CURL, Crop, no_c_aug, LN	139.7	84.9	228.9	18.9	118.1
CURL, Crop, $\alpha = 2.0$, same, LN	150.5	86.8	217.3	27.9	120.6
SPR, Re-norm	129.1	87.1	293.0	26.3	133.9
SPR, Crop, same, Re-norm	134.2	83.2	200.5	15.5	108.4
Midpoint (1.25M)					
CQL	129.8	96.7	461.6	134.8	205.8
CURL, LN	74.0	97.9	310.1	131.5	153.4
CURL, $\alpha = 2.0$, LN	35.3	102.2	193.9	189.1	130.1
CURL, Crop, LN	121.2	82.1	339.2	104.3	161.7
CURL, Crop, same, LN	99.7	98.3	343.8	118.5	165.1
CURL, Crop, no_c_aug, LN	112.8	100.9	352.1	91.3	164.3
CURL, Crop, $\alpha = 2.0$, same, LN	46.3	102.4	217.9	150.0	129.1
SPR, Re-norm	123.4	103.6	370.9	185.9	195.9
SPR, Crop, same, Re-norm	97.5	105.2	292.8	116.3	153.0

Table A.17: Normalized scores for self-supervised agents on discrete control tasks.

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 Your knowledge hub WIEN

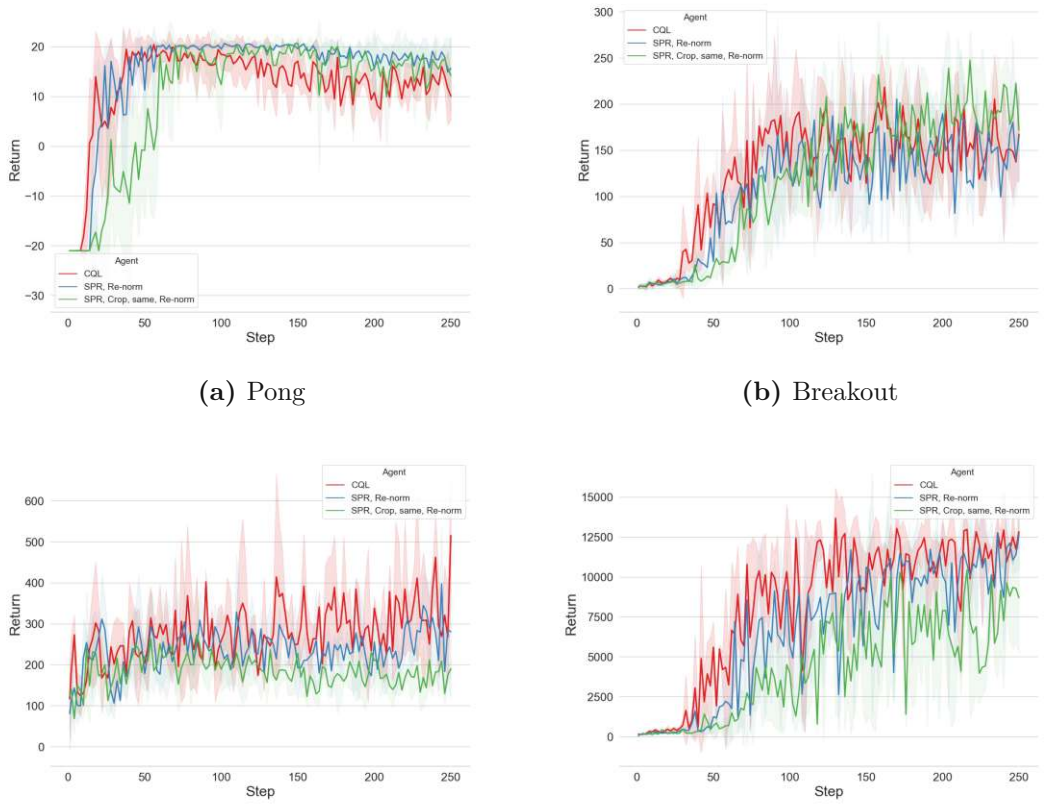


(a) Pong **(b)** Breakout

(c) Seaquest **(d)** QBert

Figure A.7: Learning curves for Curl on discrete control tasks.

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 Your knowledge hub WIEN



(a) Pong **(b)** Breakout

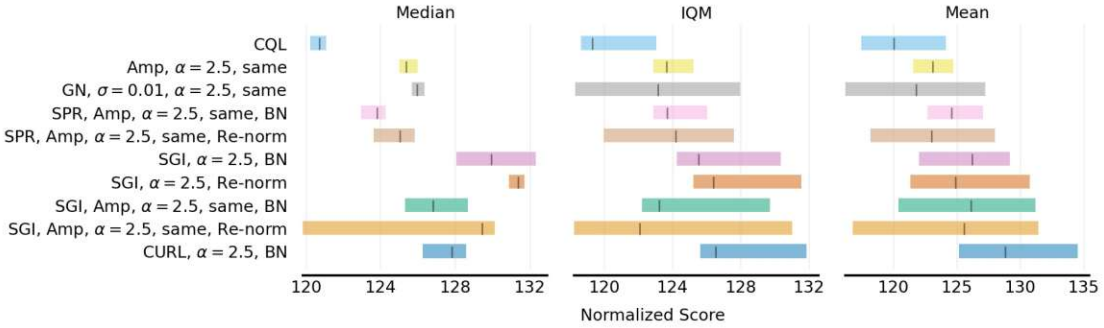
(c) Seaquest **(d)** QBert

Figure A.8: Learning curves for SPR on discrete control tasks.

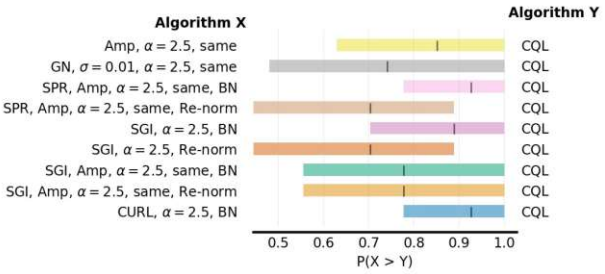
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

A.5 Best agents: Gym-MuJoCo

In Figure A.9 we show the interval estimates and probabilities of improvements for all agents that outperformed the base CQL agent on Gym-MuJoCo.



(a) Interval estimates



(b) Probability of improvement

Figure A.9: Interval estimates of normalized scores and probability of improvement for best agents on continuous control tasks.

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.

A.6 Online fine-tuning for offline RL

Finally, in this section, we list the full results for our experiments with offline pre-training and online fine-tuning. The results for continuous control tasks (Gym-MuJoCo) are listed in Tables A.18 and A.19. The results for discrete control tasks (Atari) are listed in Tables A.20 and A.21.

A.6.1 Continuous control: Gym-MuJoCo

Agent	halfcheetah-medium-v2	hopper-medium-v2	walker2d-medium-v2
	Final step (500K)		
CQL	6088.3 ± 22.7	2553.9 ± 202.4	3829.8 ± 70.2
Online SAC	4994.6 ± 2393.9	1825.0 ± 1173.1	3861.4 ± 386.5
Off-on	12538.6 ± 178.4	3567.6 ± 50.4	5236.3 ± 124.3
Off-on, on_$\alpha = 5.0$	10162.4 ± 377.0	3439.3 ± 26.1	5222.9 ± 224.7
Off-on, on_$\alpha = 5.0$, zero_after=5	12204.1 ± 133.4	3573.9 ± 24.3	3892.0 ± 1274.5
Off-on, on_$\alpha = 5.0$, half_every=5	11066.3 ± 384.3	3521.8 ± 39.2	5614.3 ± 98.8
Off-on, on_$\alpha = 5.0$, decay_every=1000	10481.8 ± 723.0	3440.7 ± 15.0	4688.1 ± 272.6
Off-on, on_$\alpha = 5.0$, α_lr=0.0001	12074.2 ± 716.4	2867.5 ± 637.8	4063.9 ± 2555.0
	Midpoint (250K)		
CQL	5948.1 ± 112.8	2630.9 ± 241.4	3924.8 ± 35.5
Online SAC	4410.1 ± 2267.3	1429.3 ± 1093.5	1605.4 ± 514.5
Off-on	11235.8 ± 149.6	2898.3 ± 932.1	2129.6 ± 1928.9
Off-on, on_$\alpha = 5.0$	8635.1 ± 167.3	3353.4 ± 26.4	4060.8 ± 457.2
Off-on, on_$\alpha = 5.0$, zero_after=5	10036.9 ± 144.6	2668.0 ± 747.0	1574.7 ± 1435.3
Off-on, on_$\alpha = 5.0$, half_every=5	8657.7 ± 480.5	3373.9 ± 24.7	3600.9 ± 1340.7
Off-on, on_$\alpha = 5.0$, decay_every=1000	9128.5 ± 44.5	3359.2 ± 11.3	3263.9 ± 2052.6
Off-on, on_$\alpha = 5.0$, α_lr=0.0001	10126.2 ± 863.0	3129.9 ± 678.9	2746.0 ± 866.4

Table A.18: Raw scores for CQL with online fine-tuning on continuous control tasks.

Agent	halfcheetah-medium-v2	hopper-medium-v2	walker2d-medium-v2	Mean
	Final step (500K)			
CQL	120.7	140.2	99.2	120.0
Online SAC	100.0	100.0	100.0	100.0
Off-on	242.8	196.1	135.6	191.5
Off-on, on_$\alpha = 5.0$	197.8	189.0	135.2	174.0
Off-on, on_$\alpha = 5.0$, zero_after=5	236.4	196.4	100.8	177.9
Off-on, on_$\alpha = 5.0$, half_every=5	214.9	193.5	145.4	184.6
Off-on, on_$\alpha = 5.0$, decay_every=1000	203.8	189.1	121.4	171.4
Off-on, on_$\alpha = 5.0$, α_lr=0.0001	234.0	157.5	105.2	165.6
	Midpoint (250K)			
CQL	132.7	184.7	244.3	187.3
Online SAC	100.0	100.0	100.0	100.0
Off-on	245.2	203.6	132.6	193.8
Off-on, on_$\alpha = 5.0$	189.9	235.7	252.8	226.1
Off-on, on_$\alpha = 5.0$, zero_after=5	219.7	187.3	98.1	168.4
Off-on, on_$\alpha = 5.0$, half_every=5	190.4	237.1	224.2	217.2
Off-on, on_$\alpha = 5.0$, decay_every=1000	200.4	236.1	203.2	213.2
Off-on, on_$\alpha = 5.0$, α_lr=0.0001	221.6	219.9	171.0	204.2

Table A.19: Normalized scores for CQL with online fine-tuning on continuous control tasks.

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 Your knowledge hub WIEN

A.6.2 Discrete control: Atari

Agent	Breakout	Pong	QBert	Seaquest
Final step (2.5M)				
CQL	167.3 ± 50.5	10.1 ± 4.9	12835.0 ± 1061.0	516.0 ± 132.2
Online DQN	128.2 ± 63.6	20.9 ± 0.1	4416.7 ± 227.8	901.3 ± 778.2
Off-on	51.6 ± 7.5	20.5 ± 0.5	13913.3 ± 1012.1	2685.3 ± 769.5
Off-on, on_$\alpha = 4.0$	45.7 ± 17.9	20.3 ± 0.3	7233.3 ± 2287.8	241.3 ± 84.0
Off-on, on_$\alpha = 4.0$, half_every=5	110.7 ± 142.8	20.0 ± 0.6	13081.7 ± 1640.2	2162.7 ± 881.9
Off-on, on_$\alpha = 4.0$, decay_every=1000	43.5 ± 19.3	20.3 ± 0.1	6851.7 ± 4113.4	2892.0 ± 538.3
Off-on, on_$\alpha = 4.0$, zero_after=10	68.9 ± 36.4	18.7 ± 3.1	13448.3 ± 1635.9	2048.0 ± 613.5
Midpoint (1.25M)				
CQL	158.6 ± 44.5	17.5 ± 0.6	11088.3 ± 986.1	225.3 ± 29.5
Online DQN	122.3 ± 13.3	18.8 ± 1.8	2523.3 ± 828.7	182.7 ± 70.5
Off-on	21.3 ± 19.2	2.4 ± 20.2	9598.3 ± 3507.2	612.0 ± 337.9
Off-on, on_$\alpha = 4.0$	56.3 ± 9.8	20.3 ± 0.1	5740.0 ± 2719.5	257.3 ± 119.5
Off-on, on_$\alpha = 4.0$, half_every=5	40.6 ± 10.0	0.7 ± 2.7	9536.7 ± 1088.9	322.7 ± 39.5
Off-on, on_$\alpha = 4.0$, decay_every=1000	21.6 ± 4.7	3.7 ± 6.0	6890.0 ± 2969.2	554.7 ± 128.3
Off-on, on_$\alpha = 4.0$, zero_after=10	27.3 ± 13.5	5.7 ± 5.0	4723.3 ± 1812.8	660.0 ± 218.1

Table A.20: Raw scores for CQL with online fine-tuning on discrete control tasks.

Agent	Breakout	Pong	QBert	Seaquest	Mean
Final step (2.5M)					
CQL	130.7	73.6	297.5	54.2	139.0
Online DQN	100.0	100.0	100.0	100.0	100.0
Off-on	39.9	98.9	322.8	312.0	193.4
Off-on, on_$\alpha = 4.0$	35.2	98.5	166.1	21.6	80.3
Off-on, on_$\alpha = 4.0$, half_every=5	86.2	97.7	303.3	249.9	184.3
Off-on, on_$\alpha = 4.0$, decay_every=1000	33.5	98.5	157.1	336.6	156.4
Off-on, on_$\alpha = 4.0$, zero_after=10	53.5	94.5	311.9	236.3	174.0
Midpoint (1.25M)					
CQL	129.8	96.7	461.6	134.8	205.8
Online DQN	100.0	100.0	100.0	100.0	100.0
Off-on	16.8	57.7	398.7	450.0	230.8
Off-on, on_$\alpha = 4.0$	45.6	103.8	235.8	160.9	136.5
Off-on, on_$\alpha = 4.0$, half_every=5	32.7	53.4	396.1	214.1	174.1
Off-on, on_$\alpha = 4.0$, decay_every=1000	17.1	61.0	284.4	403.3	191.4
Off-on, on_$\alpha = 4.0$, zero_after=10	21.8	66.2	192.9	489.1	192.5

Table A.21: Normalized scores for CQL with online fine-tuning on discrete control tasks.

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 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.

List of Figures

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 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.
TU Bibliothek Your knowledge hub WIEN

List of Tables

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 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.
TU Bibliothek Your knowledge hub WIEN

List of Algorithms

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 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.

Bibliography

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283.

Achiam, J. (2018). OpenAI SpinningUp. https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html.

Agarwal, R., Schuurmans, D., and Norouzi, M. (2020). An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, pages 104–114. PMLR.

Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A., and Bellemare, M. G. (2021). Deep reinforcement learning at the edge of the statistical precipice. In *Thirty-Fifth Conference on Neural Information Processing Systems*.

Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., et al. (2019). Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., and Zaremba, W. (2017). Hindsight experience replay. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 5055–5065.

Aytar, Y., Pfaff, T., Budden, D., Paine, T. L., Wang, Z., and Freitas, N. d. (2018). Playing hard exploration games by watching youtube. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 2935–2945.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *stat*, 1050:21.

Baevski, A., Zhou, Y., Mohamed, A., and Auli, M. (2020). wav2vec 2.0: A framework for self-supervised learning of speech representations. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

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 Your knowledge hub WIEN

Bellemare, M. G., Candido, S., Castro, P. S., Gong, J., Machado, M. C., Moitra, S., Ponda, S. S., and Wang, Z. (2020). Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588(7836):77–82.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013a). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013b). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279.

Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al. (2019). Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.

Biewald, L. (2020). Experiment tracking with weights and biases. Software available from wandb.com.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *CoRR*, abs/1606.01540.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Burda, Y., Edwards, H., Pathak, D., Storkey, A., Darrell, T., and Efros, A. A. (2018). Large-scale study of curiosity-driven learning. In *International Conference on Learning Representations*.

Chebotar, Y., Handa, A., Makoviychuk, V., Macklin, M., Issac, J., Ratliff, N., and Fox, D. (2019). Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8973–8979. IEEE.

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., and Mordatch, I. (2021). Decision transformer: Reinforcement learning via sequence modeling. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. (2020a). A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR.

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 Your knowledge hub WIEN

Chen, T., Kornblith, S., Swersky, K., Norouzi, M., and Hinton, G. E. (2020b). Big self-supervised models are strong semi-supervised learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Cobbe, K., Hesse, C., Hilton, J., and Schulman, J. (2020). Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pages 2048–2056. PMLR.

Coulom, R. (2006). Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer.

Coumans, E. and Bai, Y. (2016). Pybullet, a python module for physics simulation for games, robotics and machine learning.

Dabney, W., Rowland, M., Bellemare, M. G., and Munos, R. (2018). Distributional reinforcement learning with quantile regression. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In Burstein, J., Doran, C., and Solorio, T., editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., and Koltun, V. (2017). Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR.

Dubey, R., Agrawal, P., Pathak, D., Griffiths, T., and Efros, A. (2018). Investigating human priors for playing video games. In *International Conference on Machine Learning*, pages 1349–1357. PMLR.

Dulac-Arnold, G., Levine, N., Mankowitz, D. J., Li, J., Paduraru, C., Gowal, S., and Hester, T. (2020). An empirical investigation of the challenges of real-world reinforcement learning. *arXiv preprint arXiv:2003.11881*.

Dulac-Arnold, G., Levine, N., Mankowitz, D. J., Li, J., Paduraru, C., Gowal, S., and Hester, T. (2021). Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Mach. Learn.*, 110(9):2419–2468.

Efron, B. (1992). Bootstrap methods: another look at the jackknife. In *Breakthroughs in statistics*, pages 569–593. Springer.

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 Your knowledge hub WIEN

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al. (2018). Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pages 1407–1416. PMLR.

Fu, J., Kumar, A., Nachum, O., Tucker, G., and Levine, S. (2020). D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*.

Fujimoto, S., Meger, D., and Precup, D. (2019). Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062. PMLR.

Grill, J., Strub, F., Altché, F., Tallec, C., Richemond, P. H., Buchatskaya, E., Doersch, C., Pires, B. Á., Guo, Z., Azar, M. G., Piot, B., Kavukcuoglu, K., Munos, R., and Valko, M. (2020). Bootstrap your own latent - A new approach to self-supervised learning. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Gülçehre, Ç., Wang, Z., Novikov, A., Paine, T., Colmenarejo, S. G., Zolna, K., Agarwal, R., Merel, J., Mankowitz, D. J., Paduraru, C., Dulac-Arnold, G., Li, J., Norouzi, M., Hoffman, M., Heess, N., and de Freitas, N. (2020). RL unplugged: A collection of benchmarks for offline reinforcement learning. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Gupta, A., Kumar, V., Lynch, C., Levine, S., and Hausman, K. (2020). Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. In *Conference on Robot Learning*, pages 1025–1037. PMLR.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR.

Hafner, D., Lillicrap, T., Ba, J., and Norouzi, M. (2019). Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*.

Hafner, D., Lillicrap, T. P., Norouzi, M., and Ba, J. (2021). Mastering atari with discrete world models. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Hansen, N. and Wang, X. (2021). Generalization in reinforcement learning by soft data augmentation. In *IEEE International Conference on Robotics and Automation, ICRA 2021, Xi'an, China, May 30 - June 5, 2021*, pages 13611–13617. IEEE.

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 Your knowledge hub WIEN

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., et al. (2020). Array programming with numpy. *Nature*, 585(7825):357–362.

He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. (2020). Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9729–9738.

He, P., Liu, X., Gao, J., and Chen, W. (2021). Deberta: decoding-enhanced bert with disentangled attention. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2018). Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Hoffman, M., Shahriari, B., Aslanides, J., Barth-Maron, G., Behbahani, F., Norman, T., Abdolmaleki, A., Cassirer, A., Yang, F., Baumli, K., et al. (2020). Acme: A research framework for distributed reinforcement learning. *arXiv preprint arXiv:2006.00979*.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. (2017). Reinforcement learning with unsupervised auxiliary tasks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Jakobi, N., Husbands, P., and Harvey, I. (1995). Noise and the reality gap: The use of simulation in evolutionary robotics. In *European Conference on Artificial Life*, pages 704–720. Springer.

Janner, M., Li, Q., and Levine, S. (2021). Offline reinforcement learning as one big sequence modeling problem. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*.

Jaques, N., Shen, J. H., Ghandeharioun, A., Ferguson, C., Lapedriza, A., Jones, N., Gu, S., and Picard, R. (2020). Human-centric dialog training via offline reinforcement learning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3985–4003.

Jing, L. and Tian, Y. (2020). Self-supervised visual feature learning with deep neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*.

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 Your knowledge hub WIEN

Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., Mohiuddin, A., Sepassi, R., Tucker, G., and Michalewski, H. (2020). Model based reinforcement learning for atari. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., Quillen, D., Holly, E., Kalakrishnan, M., Vanhoucke, V., et al. (2018). Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning*, pages 651–673. PMLR.

Kalashnikov, D., Varley, J., Chebotar, Y., Swanson, B., Jonschkowski, R., Finn, C., Levine, S., and Hausman, K. (2021). Scaling up multi-task robotic reinforcement learning. In *5th Annual Conference on Robot Learning*.

Kapturowski, S., Ostrovski, G., Quan, J., Munos, R., and Dabney, W. (2018). Recurrent experience replay in distributed reinforcement learning. In *International conference on learning representations*.

Kidambi, R., Rajeswaran, A., Netrapalli, P., and Joachims, T. (2020). Morel: Model-based offline reinforcement learning. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Kostrikov, I., Nair, A., and Levine, S. (2021). Offline reinforcement learning with implicit q-learning. In *Deep RL Workshop NeurIPS 2021*.

Kumar, A., Fu, J., Soh, M., Tucker, G., and Levine, S. (2019). Stabilizing off-policy q-learning via bootstrapping error reduction. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 11761–11771.

Kumar, A., Zhou, A., Tucker, G., and Levine, S. (2020). Conservative q-learning for offline reinforcement learning. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Laskin, M., Lee, K., Stooke, A., Pinto, L., Abbeel, P., and Srinivas, A. (2020a). Reinforcement learning with augmented data. *Advances in Neural Information Processing Systems*, 33.

Laskin, M., Srinivas, A., and Abbeel, P. (2020b). CURL: contrastive unsupervised representations for reinforcement learning. In *Proceedings of the 37th International*

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 Your knowledge hub WIEN

Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event, volume 119 of *Proceedings of Machine Learning Research*, pages 5639–5650. PMLR.

Lazic, N., Lu, T., Boutilier, C., Ryu, M., Wong, E., Roy, B., and Imwalle, G. (2018). Data center cooling using model-predictive control. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 3818–3827.

Lee, A. X., Devin, C. M., Zhou, Y., Lampe, T., Bousmalis, K., Springenberg, J. T., Byravan, A., Abdolmaleki, A., Gileadi, N., Khosid, D., et al. (2021). Beyond pick-and-place: Tackling robotic stacking of diverse shapes. In *5th Annual Conference on Robot Learning*.

Legg, S. and Hutter, M. (2007). Universal intelligence: A definition of machine intelligence. *Minds and machines*, 17(4):391–444.

Levine, S., Kumar, A., Tucker, G., and Fu, J. (2020). Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*.

Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Goldberg, K., Gonzalez, J., Jordan, M., and Stoica, I. (2018). Rllib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning*, pages 3053–3062. PMLR.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In *ICLR 2016 : International Conference on Learning Representations 2016*.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Lu, J., Gong, P., Ye, J., and Zhang, C. (2020). Learning from very few samples: A survey. *arXiv preprint arXiv:2009.02653*.

Lu, Y., Hausman, K., Chebotar, Y., Yan, M., Jang, E., Herzog, A., Xiao, T., Irpan, A., Khansari, M., Kalashnikov, D., et al. (2021). Aw-opt: Learning robotic skills with imitation andreinforcement at scale. In *5th Annual Conference on Robot Learning*.

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., and Bowling, M. (2018a). Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562.

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M. J., and Bowling, M. (2018b). Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *J. Artif. Intell. Res.*, 61:523–562.

Mann, H. B. and Whitney, D. R. (1947). On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, pages 50–60.

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 Your knowledge hub WIEN

Marcus, M., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of english: The penn treebank.

Mazoure, B., Tachet des Combes, R., DOAN, T. L., Bachman, P., and Hjelm, R. D. (2020). Deep reinforcement and infomax learning. *Advances in Neural Information Processing Systems*, 33.

McKinney, W. et al. (2010). Data structures for statistical computing in python.

Mirhoseini, A., Goldie, A., Yazgan, M., Jiang, J., Songhori, E., Wang, S., Lee, Y.-J., Johnson, E., Pathak, O., Bae, S., et al. (2020). Chip placement with deep reinforcement learning. *arXiv preprint arXiv:2004.10746*.

Mirowski, P., Pascanu, R., Viola, F., Soyer, H., Ballard, A., Banino, A., Denil, M., Goroshin, R., Sifre, L., Kavukcuoglu, K., Kumaran, D., and Hadsell, R. (2017). Learning to navigate in complex environments. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR.

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

Nair, A., Dalal, M., Gupta, A., and Levine, S. (2020). Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*.

Nie, X., Brunskill, E., and Wager, S. (2020). Learning when-to-treat policies. *Journal of the American Statistical Association*, pages 1–18.

Oord, A. v. d., Li, Y., and Vinyals, O. (2018). Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*.

Panayotov, V., Chen, G., Povey, D., and Khudanpur, S. (2015). Librispeech: an asr corpus based on public domain audio books. In *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5206–5210. IEEE.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019a). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 8024–8035.

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 Your knowledge hub WIEN

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019b). Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32:8026–8037.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning*, pages 2778–2787. PMLR.

Pathak, D., Gandhi, D., and Gupta, A. (2019). Self-supervised exploration via disagreement. In *International Conference on Machine Learning*, pages 5062–5071. PMLR.

Peng, X. B., Andrychowicz, M., Zaremba, W., and Abbeel, P. (2018). Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3803–3810. IEEE.

Peng, X. B., Kumar, A., Zhang, G., and Levine, S. (2019). Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*.

Perez, L. and Wang, J. (2017). The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*.

Pham, H., Guan, M., Zoph, B., Le, Q., and Dean, J. (2018). Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning*, pages 4095–4104. PMLR.

Pomerleau, D. (1988). ALVINN: an autonomous land vehicle in a neural network. In Touretzky, D. S., editor, *Advances in Neural Information Processing Systems 1, [NIPS Conference, Denver, Colorado, USA, 1988]*, pages 305–313. Morgan Kaufmann.

Raileanu, R., Goldstein, M., Yarats, D., Kostrikov, I., and Fergus, R. (2021). Automatic data augmentation for generalization in reinforcement learning. In Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*.

Rajeswaran, A., Kumar, V., Gupta, A., Vezzani, G., Schulman, J., Todorov, E., and Levine, S. (2018). Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. In Kress-Gazit, H., Srinivasa, S. S., Howard, T., and Atanasov, N., editors, *Robotics: Science and Systems XIV, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, June 26-30, 2018*.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). Squad: 100, 000+ questions for machine comprehension of text. In Su, J., Carreras, X., and Duh, K., editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 2383–2392. The Association for Computational Linguistics.

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 Your knowledge hub WIEN

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2016). Prioritized experience replay. In Bengio, Y. and LeCun, Y., editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. (2020). Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR.

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

Schwarzer, M., Anand, A., Goel, R., Hjelm, R. D., Courville, A., and Bachman, P. (2021a). Data-efficient reinforcement learning with self-predictive representations. In *International Conference on Learning Representations*.

Schwarzer, M., Rajkumar, N., Noukhovitch, M., Anand, A., Charlin, L., Hjelm, R. D., Bachman, P., and Courville, A. (2021b). Pretraining reward-free representations for data-efficient reinforcement learning. In *Self-Supervision for Reinforcement Learning Workshop-ICLR 2021*.

Schweighofer, K., Hofmarcher, M., Dinu, M.-C., Renz, P., Bitto-Nemling, A., Patil, V. P., and Hochreiter, S. (2021). Understanding the effects of dataset characteristics on offline reinforcement learning. In *Deep RL Workshop NeurIPS 2021*.

Seno, T. and Imai, M. (2021). d3rlpy: An Offline Deep Reinforcement Learning Library. In *35th Conference on Neural Information Processing Systems, Offline Reinforcement Learning Workshop, 2021*, NeurIPS 2021 Offline Reinforcement Learning Workshop.

Shelhamer, E., Mahmoudieh, P., Argus, M., and Darrell, T. (2017). Loss is its own reward: Self-supervision for reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net.

Shorten, C. and Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):1–48.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2018). A general reinforcement learning

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 Your knowledge hub WIEN

algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of go without human knowledge. *nature*, 550(7676):354–359.

Silver, D., Singh, S., Precup, D., and Sutton, R. S. (2021). Reward is enough. *Artificial Intelligence*, page 103535.

Sinha, S., Mandlekar, A., and Garg, A. (2021). S4rl: Surprisingly simple self-supervision for offline reinforcement learning in robotics. In *5th Annual Conference on Robot Learning*.

Stooke, A. and Abbeel, P. (2018). Accelerated methods for deep reinforcement learning. *CoRR*, abs/1803.02811.

Sutton, R. S. (1991). Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., McAllester, D. A., Singh, S. P., Mansour, Y., et al. (1999). Policy gradient methods for reinforcement learning with function approximation. In *NIPs*, volume 99, pages 1057–1063. Citeseer.

Tan, J., Zhang, T., Coumans, E., Iscen, A., Bai, Y., Hafner, D., Bohez, S., and Vanhoucke, V. (2018). Sim-to-real: Learning agile locomotion for quadruped robots. In *Robotics: Science and Systems*.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al. (2018). Deepmind control suite. *arXiv preprint arXiv:1801.00690*.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE.

Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30.

van Hasselt, H., Hessel, M., and Aslanides, J. (2019). When to use parametric models in reinforcement learning? In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 14322–14333.

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 Your knowledge hub WIEN

Van Rossum, G. and Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA.

Vinitsky, E., Kreidieh, A., Le Flem, L., Kheterpal, N., Jang, K., Wu, C., Wu, F., Liaw, R., Liang, E., and Bayen, A. M. (2018). Benchmarks for reinforcement learning in mixed-autonomy traffic. In *Conference on Robot Learning*, pages 399–409. PMLR.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354.

Wang, J., Kurth-Nelson, Z., Soyer, H., Leibo, J. Z., Tirumala, D., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. M. (2017). Learning to reinforcement learn. In Gunzelmann, G., Howes, A., Tenbrink, T., and Davelaar, E. J., editors, *Proceedings of the 39th Annual Meeting of the Cognitive Science Society, CogSci 2017, London, UK, 16-29 July 2017*. cognitivesciencesociety.org.

Wang, Z., Novikov, A., Zolna, K., Merel, J. S., Springenberg, J. T., Reed, S. E., Shahriari, B., Siegel, N., Gulcehre, C., Heess, N., et al. (2020). Critic regularized regression. *Advances in Neural Information Processing Systems*, 33.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR.

Waskom, M. L. (2021). Seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Wu, Y., Tucker, G., and Nachum, O. (2019). Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*.

Yarats, D., Fergus, R., Lazaric, A., and Pinto, L. (2021a). Reinforcement learning with prototypical representations. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 11920–11931. PMLR.

Yarats, D., Kostrikov, I., and Fergus, R. (2021b). Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

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 Your knowledge hub WIEN

Yu, T., Kumar, A., Rafailov, R., Rajeswaran, A., Levine, S., and Finn, C. (2021). Combo: Conservative offline model-based policy optimization. In *Self-Supervision for Reinforcement Learning Workshop-ICLR 2021*.

Yu, T., Thomas, G., Yu, L., Ermon, S., Zou, J. Y., Levine, S., Finn, C., and Ma, T. (2020). MOPO: model-based offline policy optimization. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Zoph, B. and Le, Q. V. (2017). Neural architecture search with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

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 Your knowledge hub WIEN