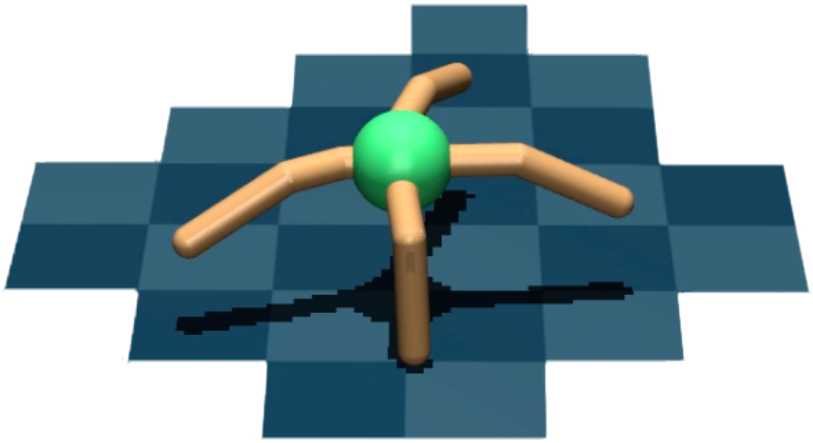
Johann Wolfgang Goethe-University Frankfurt am Main
Faculty 12 - Computer Science and Mathematics
Thesis - Bioinformatics (M.Sc.)

Intrinsically Motivated Agents for Goal Discovery in High Dimensional State Spaces



Author: Nico Bohlinger
Duration: 25.11.2022 - 25.05.2023

Zusammenfassung

Goal-Conditioned Reinforcement Learning (GCRL) ist ein beliebtes Framework, um Agenten zu trainieren, welche mehrere Aufgaben in einer Lernumgebung lösen können. Es ist entscheidend einen Agenten mit vielfältigen Zielen zu trainieren, sodass er lernt zu generalisieren, um auch unbekannte Ziele erreichen zu können. Deshalb versuchen aktuelle Algorithmen, den Agenten so zu trainieren, dass er lernt Ziele zu erreichen, während er gleichzeitig die Umgebung nach neuen Zielen erkundet (Aubret et al., 2021; Mendonca et al., 2021). Dies erzeugt eine Form des prominenten Exploration-Exploitation Dilemmas. Um den Druck von einem einzelnen Agenten zu nehmen, der zwei konkurrierende Ziele gleichzeitig optimieren muss, schlägt diese Thesis die neue Algorithmenfamilie Goal-Conditioned Reinforcement Learning with Prior Intrinsic Exploration (GC-π) vor, welche die Exploration und das Ziel-Lernen in getrennte Phasen aufteilt. In einer ersten Explorationsphase erkundet ein intrinsisch motivierter Agent die Umgebung und sammelt einen großen Datensatz von Umgebungszuständen und Aktionen. Dieser Datensatz wird dann verwendet, um einen Repräsentationsraum zu lernen, welcher als Distanzmetrik für das zielbedingte Belohnungssignal dient. In der letzten Phase wird eine zielbedingte Policy mit Hilfe des Repräsentationsraums trainiert und die Trainingsziele werden zufällig aus dem Datensatz ausgewählt, der während der Explorationsphase gesammelt wurde. Mehrere Variationen dieser drei Phasen werden ausführlich in der klassischen Ant-Maze MuJoCo Umgebung (Nachum et al., 2018) evaluiert. Die finalen Ergebnisse zeigen, dass die vorgeschlagenen Algorithmen in der Lage sind, die Lernumgebung vollständig zu erkunden und alle Evaluierungsziele zu erreichen, während sie jede Dimension des Zustandsraums für den Zielraum verwenden. Dies macht den Ansatz flexibler im Vergleich zu anderen GCRL Algorithmen aus der Literatur, welche nur eine geringe Anzahl der Dimensionen für die Ziele verwenden (S. Li et al., 2021a; Pong et al., 2020).

Abstract

Goal-Conditioned Reinforcement Learning (GCRL) is a popular framework for training agents to solve multiple tasks in a single environment. It is crucial to train an agent on a diverse set of goals to ensure that it can learn to generalize to unseen downstream goals. Therefore, current algorithms try to learn to reach goals while simultaneously exploring the environment for new ones (Aubret et al., 2021; Mendonca et al., 2021). This creates a form of the prominent exploration-exploitation dilemma. To relieve the pressure of a single agent having to optimize for two competing objectives at once, this thesis proposes the novel algorithm family Goal-Conditioned Reinforcement Learning with Prior Intrinsic Exploration (GC-π), which separates exploration and goal learning into distinct phases. In the first exploration phase, an intrinsically motivated agent explores the environment and collects a rich dataset of states and actions. This dataset is then used to learn a representation space, which acts as the distance metric for the goal-conditioned reward signal. In the final phase, a goal-conditioned policy is trained with the help of the representation space, and its training goals are randomly sampled from the dataset collected during the exploration phase. Multiple variations of these three phases have been extensively evaluated in the classic AntMaze MuJoCo environment (Nachum et al., 2018). The final results show that the proposed algorithms are able to fully explore the environment and solve all downstream goals while using every dimension of the state space for the goal space. This makes the approach more flexible compared to previous GCRL work, which only ever uses a small subset of the dimensions for the goals (S. Li et al., 2021a; Pong et al., 2020).

Contents

List of Figures

List of Tables

List of Abbreviations

AE Autoencoder.

BIC Bayesian Information Criterion.

CLTT Contrastive Learning Through Time.

CQL Conservative Q-Learning.

D4RL Datasets for Deep Data-Driven Reinforcement Learning.

DISCERN Discriminative Embedding Reward Network.

DQN Deep Q-Network.

DRL Deep Reinforcement Learning.

E3B Exploration via Elliptical Episodic Bonuses.

GC-π Goal-Conditioned RL with Prior Intrinsic Exploration.

GCRL Goal-Conditioned Reinforcement Learning.

GMM Gaussian Mixture Model.

HER Hindsight Experience Replay.

HRL Hierarchical Reinforcement Learning.

IM Intrinsic Motivation.

InfoNCE Information Noise-Contrastive Estimation.

IQL Implicit Q-Learning.

JEM Joint Embedding Method.

LSTM Long Short-Term Memory.

MDP Markov Decision Process.

ML Machine Learning.

MSE Mean Squared Error.

MuJoCo Multi-Joint dynamics with Contact.

NN Neural Network.

NovelD Novelty Difference.

NT-Xent Normalized Temperature-Scaled Cross Entropy.

PPO Proximal Policy Optimization.

ReLU Rectified Linear Unit.

RL Reinforcement Learning.

RND Random Network Distillation.

SAC Soft Actor-Critic.

SB3 Stable-Baselines3.

SimCLR A Simple Framework for Contrastive Learning of Visual Representations.

SOTA State-of-the-Art.

SSL Self-Supervised Learning.

TD Temporal Difference.

TQC Truncated Quantile Critics.

UCB Upper-Confidence-Bound.

VAE Variational Autoencoder.

1. Introduction

Developing fully autonomous agents with the ability to solve arbitrary tasks in complex environments is one of the ultimate goals of Machine Learning (ML). Such agents are the continuation of the never-ending automation of the 21st century. Real-world agents can take the form of household robots to improve the quality of life of humans by performing tedious tasks (Driess et al., 2023). They can be used as autonomous drivers to reduce the number of accidents on the road and to improve the efficiency of the transportation system (Kiran et al., 2021). Future medical robots will be able to perform surgeries and assist doctors in their daily work (Yu et al., 2021). In general, highly capable real-world agents can help with many tasks and are especially useful in environments where continuous physical and mental strain, fast and precise execution, or safety concerns might be an issue for human workers. Furthermore, digital agents can be used to navigate the internet (Shi et al., 2017) and assist in software development (M. Chen et al., 2021) or even plan financial strategies for businesses (Ozbayoglu et al., 2020). No matter the environment, future agents will be able to execute all kinds of complex behavior. Nevertheless, they should remain fully controllable and safe to operate by humans to prove their usefulness. Researching how to build and train these increasingly powerful but controllable agents is therefore crucial for current and future applications.

1.1 Motivation

Reinforcement Learning (RL) describes a framework that allows an agent to learn how to solve a task defined by a reward signal in an environment. However, the real world and even simulated environments consist of a possibly infinite number of tasks and each of those tasks could be important for a human operator during downstream use. Training a specialized agent

for each single task is therefore not a feasible approach.

Goal-Conditioned Reinforcement Learning (GCRL) tackles the problem of having multiple tasks to care about by giving the agent a description of the goal that is associated with the current task, thereby conditioning the agent's behavior on the goal. Some GCRL approaches are based on the assumption that the set of relevant goals is known during training (Schaul et al., 2015). In many applications, this set of downstream tasks/goals is not known beforehand, so other algorithms continue to explore the environment for possible new goals while simultaneously trying to reach already encountered ones (Warde-Farley et al., 2018). The latter approach optimizes for two rivaling objectives at the same time. This is an example of the prominent exploration-exploitation dilemma, which is pervasive to the core of RL. To alleviate this dilemma, a stricter separation of exploration and goal learning could be a promising approach, which allows for a more precise focus on the respective objectives.

For hard exploration problems, the incorporation of Intrinsic Motivation (IM) into RL algorithms has been adapted more recently (Badia et al., 2020a). Adding intrinsic rewards enables a more extensive exploration of the environment, which can be crucial for escaping local reward optima. However, choosing a good mixing coefficient for intrinsic and extrinsic rewards is not trivial. If the intrinsic reward is too high, the agent will not learn to solve the original task. If the extrinsic reward is too high, the agent will tunnel vision on the extrinsic signal and not explore enough. It is conceivable that selecting different mixing coefficients during different parts of the training process or even in different parts of the state space might be necessary for especially hard exploration tasks. When the agent can be fully focused on exploring using only intrinsic reward, this might lead to faster and more widespread exploration.

Combining the motivation for separating goal learning and exploration with the approach of using pure intrinsic reward for the exploration phase forms the main idea of this thesis.

1.2 Aim and Objectives

The general aim of this thesis is to investigate the potential of combining IM with GCRL. This aim is divided into the multiple objectives, which act as the fundamentals of the development of a novel algorithm, that ...

1. ... is able to solve a suite of different tasks in a single environment, by utilizing the **concept of goals**.

2. ... can efficiently **discover new goals** during training with the help of a pure IM objective.

3. ... splits the objectives of exploration and goal learning into **separately optimizable processes**.

4. ... is able to reach a set of **unknown downstream goals** during the evaluation phase.

5. ... is capable of learning in a **high dimensional state space** without any prior information about the dimensions, e.g., the agent is not told which dimensions to focus on for exploration or goal learning.

6. ... can work with a complex 3-dimensional **physics-based environment** and an agent with a **continuous action space**.

7. ... should run with a **reasonable computational budget** that can be expected from standard consumer hardware. This means that orthogonal ideas, such as the expensive training of world models and using them for exploration or goal learning, are out of the scope (Mendonca et al., 2021).

1.3 Outline

The thesis is structured through the following chapters:

- **Fundamentals & Related Work**: This chapter introduces the fundamental topics for the following chapters and gives an overview of their related work.

- **Problem Setup**: This chapter illustrates the concrete problem setup, including the used environment, agent and the developed Deep Reinforcement Learning (DRL) framework.

- **Goal-Conditioned RL Baseline**: This chapter explains the GCRL baseline algorithm, which is a mix of previously developed methods from the literature. It is used as a reference for the following developed algorithms.

- **Goal-Conditioned RL with Prior Intrinsic Exploration**: This chapter contains the main contributions of this thesis. It deeply explains the developed methods and showcases them through several experiments.

- **Conclusion**: This chapter summarizes the work of the thesis, discusses its limitations and finally suggests potential future work.

2. Fundamentals & Related Work

This chapter introduces all the fundamental concepts that are used as the basis throughout the rest of the thesis. Each section explains the respective topics and gives a brief overview of related work in the literature. This literature review is not exhaustive, but rather focuses on the most relevant work for the methods discussed and developed in the thesis.

2.1 Reinforcement Learning

The Reinforcement Learning framework is a paradigm of Machine Learning and is concerned with training an agent to make decisions by interacting with the environment to solve a reward-defined task. The fundamentals of RL are well defined in Sutton & Barto, 2018.

RL problems are typically modeled as a Markov Decision Process (MDP), which is a mathmatical formalism used to describe sequential decision-making processes.

A specific MDP can be defined by its 5-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$:

- $\mathcal{S}$ is the set of all possible states s and forms the state space (also called observation space).
- $\mathcal{A}$ is the set of all possible actions a and forms the action space.
- $\mathcal{P}$ is the transition function that maps the state-action pair (s, a) to the next state s'.
- $\mathcal{R}$ is the reward function that maps the state-action pair (s, a) to the scalar reward r.

- γ is the scalar discount factor that determines the importance of future rewards.

With the formalism of MDPs in place, the goal of an RL algorithm is to learn a policy $\pi(a_t|s_t)$ which takes, at timestep t, a state s_t as input and outputs an action a_t. The action a_t is then executed in the environment, and the agent receives a reward r_{t+1} and the next state s_{t+1} from the environment. This leads to the standard agent-environment interaction loop of Reinforcement Learning, as shown in Figure 2.1.

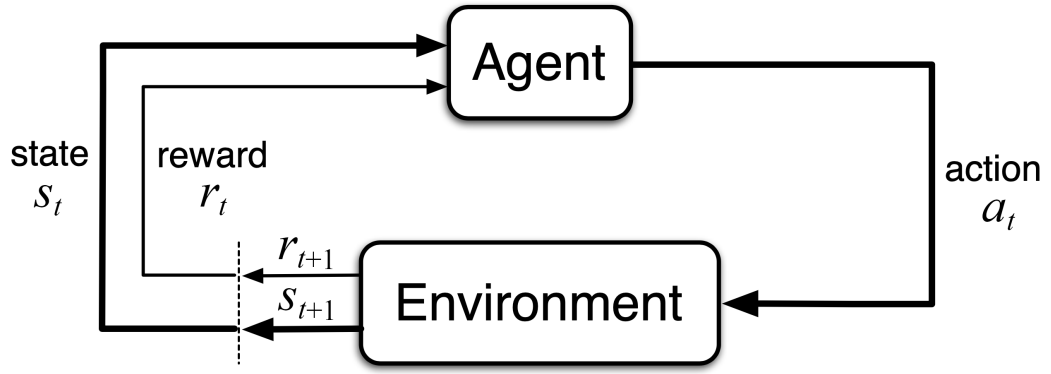


Figure 2.1: The standard agent-environment interaction loop of Reinforcement Learning. Modified from Sutton & Barto, 2018.

For the agent, the key value to optimize is the sum of discounted future rewards, known as the return G, and is defined as follows:

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k \tag{2.1}$$

where T is the terminal timestep and marks the end of the episode.

To help the agent learn a good policy, value functions are used to estimate the expected return for a given state or state-action pair when acting according to the current policy.
The state-value function is defined as:

$$V(s) = \mathbb{E}_\pi[G_t|s_t = s] \tag{2.2}$$

The action-value function (also called Q-function) is defined as:

$$Q(s,a) = \mathbb{E}_\pi[G_t | s_t = s, a_t = a] \tag{2.3}$$

With the definition of the return and the value function, the famous Bellman equation gives rise to the fundamental Temporal Difference (TD) update rule of the Q-learning algorithm (Watkins, 1989):

$$Q(s,a) \leftarrow Q(s,a) + \eta \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right) \tag{2.4}$$

where η is the learning rate. A Q-learning agent learns only the action-value function. The policy is then derived by selecting the action with the highest value for a given state. Q-learning is a model-free algorithm, which means that it does not require a model of the environment. It is also an off-policy algorithm, which means that it can learn from experiences that were not generated by the current policy.

2.2 Deep Reinforcement Learning Algorithms

In environments with high-dimensional state spaces, simple tabular methods such as Q-learning are not applicable, and the usage of function approximators like a Neural Network (NN) is required. The field of Deep Reinforcement Learning combines RL with NNs to solve more complex problems.

One of the first successful algorithms was the Deep Q-Network (DQN) (Mnih et al., 2015). Besides using a NN to approximate the Q-function Q_ϕ, DQN introduces two important ideas:

- A replay buffer to store experiences that can be sampled for training. This increases the sample efficiency and learning stability by smoothing out changes in the data distribution.

- A target network $Q_{\bar{\phi}}$ to increase stability by smoothing out changes in target values, which are used for the loss function.

DQN is made for environments with a discrete action space such as the Atari games. It can only support continuous actions by discretizing the action space, which makes it not ideal for many complex environments and agents. Because of this limitation, the Soft Actor-Critic (SAC) algorithm (Haarnoja et al., 2018a) was developed.

Besides action-space limitations, classic Q-learning-based algorithms also notoriously suffer from an overestimation bias, which means that the Q-function tends to overestimate the true value of the action. This results from using the maximum action-value as the approximation of the expected action-value in the Q-learning update rule (see Eq. 2.4) (Hasselt, 2010). Because of that the SAC algorithm trains two Q-functions Q_{ϕ_1}, Q_{ϕ_2} and takes their minimum to reduce the overestimation bias, with the following loss function:

$$\nabla_{\phi_i}(Q_{\phi_i}(s,a) - (r + \gamma(\min_{i=1,2} Q_{\bar{\phi}_i}(s',\tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s'))))^2 \tag{2.5}$$

where $\tilde{a}'$ is a sampled action from the policy and α is the temperature parameter of the entropy regularization term. This regularization encourages exploration by pushing up the entropy of the policy. The coefficient α can be either fixed or automatically tuned (Haarnoja et al., 2018b).

In addition to the Q-functions, SAC trains a NN policy $\pi_\theta(a|s)$, with the following loss function:

$$\nabla_\theta \min_{i=1,2} Q_{\phi_i}(s,\tilde{a}) - \alpha \log \pi_\theta(\tilde{a}|s) \tag{2.6}$$

The policy outputs mean and standard deviation of Gaussian distributions for every action dimension. The reparameterization trick (Kingma & Welling, 2013) is then used to sample the action and compute the gradient.

The DRL algorithm chosen for all the experiments in this thesis is Trun-

cated Quantile Critics (TQC) (Kuznetsov et al., 2020). TQC uses SAC as its core and improves on it by using quantile regression to learn the distribution of the return rather than its mean for the Q-function.

The return distribution Z is defined as:

$$Z_{\phi_i}(s,a) = \frac{1}{M}\sum_{m=1}^{M} \delta(z^m_{\phi_i}(s,a)) \tag{2.7}$$

where M is the number of quantiles (also called atoms), δ is the Dirac delta function and $z^m_{\phi_i}(s,a)$ is the return at the m-th quantile.

Instead of using the minimum over two Q-functions to reduce overestimation, TQC combines the distributions of N Q-functions, as illustrated in Figure 2.2. Only the first k atoms are kept to cut off the probably overestimated upper tail of the combined return distribution.

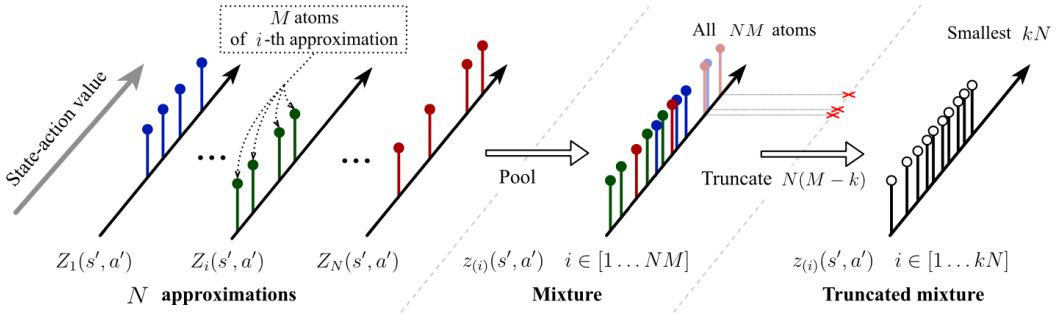


Figure 2.2: The mixing process of N Q-functions into a single truncated estimation of the return distribution in TQC. Taken from Kuznetsov et al., 2020.

For an extensive overview of further distributional RL algorithms see Bellemare et al., 2023.

2.3 Exploration-Exploitation Dilemma

As the name Reinforcement Learning suggests, an agent should learn to reinforce good actions and avoid bad ones indicated by the given reward signal. Naturally, the agent does not know which actions are good or bad

at the beginning of the learning process. Even after a long time of learning, the agent can never be sure that its evaluation, e.g., through its value function, is correct. This is because environments can be stochastic, non-stationary or partially observable. Additionally, the agent might use function approximators such as NNs, which introduce approximation errors. Looking at it from the optimization perspective, the agent wants to find the global optimum but can never be sure that it has found it. This means agents need to explore by using actions that seem bad in the short term but might lead to better long-term rewards. The core of the dilemma is that neither pure exploration nor pure exploitation leads to desirable performance; thus, a not clearly decidable mixture of the two is needed (Sutton & Barto, 2018).

Finding the right amount of exploration is an open problem in RL and has been studied for decades. Classical approaches to this problem are the ϵ-greedy (Watkins, 1989) or Upper-Confidence-Bound (UCB) (Lai, Robbins, et al., 1985) action selection schemes. DRL algorithms can achieve exploration through multiple sources and concepts:

- Noise and uncertainty in stochastic, non-stationary or partially observable environments
- Errors in function approximators
- Noisy gradient estimations through batches or mini-batches (Schulman et al., 2017)
- Added exploration noise to the actions (Lillicrap et al., 2015)
- Gaussian action distributions (Haarnoja et al., 2018a)
- Entropy regularization of action distributions (Haarnoja et al., 2018a)
- Noise in the parameter space (Fortunato et al., 2017)
- Mixing intrinsic and extrinsic rewards (Badia et al., 2020a)
- The same concepts on a higher hierarchical level (Hafner et al., 2022)

All the listed sources of exploration can be reduced to differently structured noise around the current policy or rather the extrinsic reward signal. Exploration free of bias from the extrinsic reward or different exploration modes (Pislar et al., 2021) might be needed to solve exceedingly complex tasks with sparse rewards or reward landscapes filled with strongly attracting but poorly performing local optima.

The exploration-exploitation dilemma is typically not considered in other ML paradigms such as Supervised, Self-Supervised or Unsupervised Learning. Although newer research in the fields of Continual and Active Learning suggests that balancing the two might be key for these paradigms in the future as well (Jiang et al., 2022; Mundt et al., 2023). The real world is naturally open-ended and through that provides an endless stream of data. One can imagine how even the simplest predictive models would first need to figure out which data they should focus on and which they should ignore during learning.

2.4 Intrinsic Motivation

The idea of Intrinsic Motivation comes from the field of psychology and introduces a clear distinction between extrinsic and intrinsic sources that drive behavior. Extrinsic sources are rewards, punishments or other types of external feedback. Intrinsic sources are inherent preferences and values. These are a key driver of human curiosity and creativity (Ryan & Deci, 2000).

IM can be found in different forms in current DRL algorithms and is typically used to improve the exploration capabilities of agents or enable diverse skill learning (Aubret et al., 2019). The focus in this thesis lies on the exploration aspect of IM and utilizes the concept of state novelty.

State novelty is a measure of how novel a state is to the agent compared to previously visited states. Computationally, this can be achieved by using

prediction errors of designated state feature encoder networks (Pathak et al., 2017). The main IM algorithm used in this thesis is Random Network Distillation (RND) (Burda et al., 2018).

RND initializes two NNs $f_\psi(s)$ and $\hat{f}_\varphi(s)$. The former is a feature encoder that maps states to a feature vector, $f_\psi : \mathcal{S} \to \mathbb{R}^z$. It is randomly initialized and never trained during the learning process. The latter is a predictor network that tries to predict the feature vector of the random feature encoder, so it maps to the same output dimensions $\hat{f}_\varphi : \mathcal{S} \to \mathbb{R}^z$. The predictor network is trained on the Mean Squared Error (MSE) of its prediction compared to the feature vector:

$$\|f_\psi(s) - \hat{f}_\varphi(s)\|^2 \tag{2.8}$$

This same error is used as the measure of state novelty and acts as the intrinsic reward given to the agent:

$$r_t^i = \|f_\psi(s_{t+1}) - \hat{f}_\varphi(s_{t+1})\|^2 \tag{2.9}$$

To keep the scale of the intrinsic reward the same for different environments, RND divides the intrinsic reward by its running average. This type of normalization is also performed on the input states for the feature encoder and predictor to control their variance.

Besides life-long exploration bonuses (also called across-episode bonuses), like in RND, there are also episodic exploration bonuses (also called within-episode bonuses), which try to maximize state novelty within a single episode. These can either be combined with life-long exploration bonuses (Badia et al., 2020a; Zhang et al., 2021) or used on their own, like in the Exploration via Elliptical Episodic Bonuses (E3B) algorithm (Henaff et al., 2022)

Many DRL algorithms achieve State-of-the-Art (SOTA) performance by mixing intrinsic reward with extrinsic reward (Badia et al., 2020b; Jarrett et al., 2022). The choice of the mixing coefficient is crucial but often fixed during training (Burda et al., 2018). Different mixing coefficients might be

needed for different parts of the learning process (Badia et al., 2020a) or in different parts of the state space to achieve more precise exploration. Dynamically adjusting the mixing coefficient can be done by using a measure of long-term learning progress of the agent. If the learning progress slows down, higher mixing coefficients can be used to increase exploration.

2.5 Goal-Conditioned Reinforcement Learning

Goal-Conditioned Reinforcement Learning is a subfield of RL in which an agent learns to reach a specific goal or set of goals. GCRL introduces a framework to tackle the problem of learning multiple tasks with the same agent. If the language for goal specification is sufficiently expressive, goals can be used to describe any given task. To add the notion of goals to an existing RL algorithm, the agent's policy and value function are conditioned on the goal: $\pi(a|s,g)$ and $V(s,g)$. In practice, state and goal are concatenated and form the input to the respective NNs.

Natural language is an obvious candidate to define the goal space, especially at the highest abstraction level where human operators (or teachers) and agents can communicate (Sigaud et al., 2022). But this requires the agent to learn a mapping from language to the state space, which is an open problem in the domain of language grounding. One current approach is to use automatically generated language annotations from simulated environments (Mu et al., 2022), but this is not yet scalable to real-world applications.

To circumvent the language problem, the goal space $\mathcal{G}$ is often defined in an equal manner as the state space $\mathcal{S}$ (Schaul et al., 2015). In this way, progress on a goal can be measured by comparing the current state to the goal state. To calculate the rewards when trying to reach a goal, a distance metric, e.g., the euclidean distance, can be used:

$$r_t = -\|s_t - g_t\|^2 \tag{2.10}$$

State spaces in complex environments can be high dimensional and non-linear, i.e., euclidean distances in this raw state and goal space might not be a good distance measure considering the dynamics of the environment. Learning a feature representation z for those raw states can disentangle the information in the state dimensions and subsequently make the distance metric more meaningful:

$$r_t = -\|z_{s_t} - z_{g_t}\|^2 \tag{2.11}$$

Learning feature representation is performed in a variety of different ways in the literature for GCRL algorithms (Mendonca et al., 2021; Pong et al., 2020; Warde-Farley et al., 2018). For an overview of the different Representation Learning approaches, see section 2.6.

During the learning process, GCRL algorithms have to deal with the question of how goals are acquired and which ones should be used next. A range of different methods have been proposed over time (Colas et al., 2022). To give an overview, some of them are described below with their advantages and drawbacks:

HER: Hindsight Experience Replay (HER) (Andrychowicz et al., 2017) uses the encountered states in a trajectory of an episode as examples for succesfully reached goals for every state prior to it in the trajectory. This is often referred to as goal-relabeling. Unlike other methods, HER does not inherently encourage the exploration of the goal space.

DISCERN: Discriminative Embedding Reward Network (DISCERN) (Warde-Farley et al., 2018) stores a goal buffer of previously encountered states. Goals are sampled randomly from the buffer. New states are added to the buffer either randomly or by checking if the proposed goal is further away from the goal buffer than the proposed removal goal. The latter variant is used to maximize the entropy of the goal buffer, which encourages exploration. DISCERN focuses on making the distribution of the goal buffer slowly more uniform rather than strongly exploring the edges of the distribution, which contain the areas of the goal space that are not yet mastered (Pitis et al., 2020).

Skew-Fit: Skew-Fit (Pong et al., 2020) trains a Variational Autoencoder (VAE) on previously seen states to generate new goals. The sampling process is skewed toward lower probability goals to maximize entropy and encourage exploration. Skew-Fit explores the edges of the goal distribution more aggressively compared to DISCERN but is only ever trained on already encountered states, which might slow down the exploration (Pitis et al., 2020). Additionally, making goals potentially unreachable through the generation of the VAE can lead to an inefficient training process.

Hierarchical Reinforcement Learning (HRL) algorithms use the notion of goals for higher level policies that learn to propose subgoals to lower level policies (Hafner et al., 2022; S. Li et al., 2021b; S. Li et al., 2021a). The introduced subgoal space can be used to communicate only the necessary information to the lower level controllers while enabling exploration on a more abstract higher level.

2.6 Representation Learning

Representation Learning is a field of ML that aims to learn representations (also called embeddings, features, latent vectors or hidden vectors) of raw data. The goal is to learn representations that capture the underlying structure of the data. This means disentangling the complex and non-linear information present in the data in a way that is useful for potential downstream tasks or further Transfer Learning.

Unsupervised Learning and especially Self-Supervised Learning (SSL) methods use forms of Representation Learning as core components in their algorithms. For instance in Unsupervised Learning, training compact representations can be done with the family of Autoencoders (AEs) through their reconstruction loss (Ballard, 1987; Kingma & Welling, 2014). In SSL, Joint Embedding Methods (JEMs) are typically used among others to explicitly learn representations (Balestriero & LeCun, 2022; Bordes et al., 2023).

In general, JEMs all train different forms of Siamese networks (Bromley et al., 1993), which are NNs that share their weights for multiple inputs in their objective function. These networks are trained to produce similar representations for similar inputs. To prevent the collapse of the representation space, i.e., the network learns to produce the same representation for all inputs, different approaches have been proposed, which can be divided into contrastive and non-contrastive methods.

The training process of contrastive JEMs (also known as Contrastive Learning) uses positive and negative pairs of data. The positive pairs are used to pull the representations of similar data points together, while the negative pairs are used to push dissimilar data points apart in the representation space (T. Chen et al., 2020). On the other hand, non-contrastive JEMs only use positive pairs and rely on different tricks to prevent the representation collapse, e.g., using a momentum encoder and batch normalization (Grill et al., 2020), data clustering (Caron et al., 2020) or regularizing statistics of the data (Bardes et al., 2022).

A Simple Framework for Contrastive Learning of Visual Representations (SimCLR) (T. Chen et al., 2020) is a popular contrastive algorithm and the main method of choice in this thesis. A flow through the network architecture is illustrated in Figure 2.3. SimCLR uses the Normalized Temperature-Scaled Cross Entropy (NT-Xent) loss, which is a variant of the Information Noise-Contrastive Estimation (InfoNCE) loss (Oord et al., 2018):

$$\text{sim}_{z_i,z_j} = \frac{z_i^T z_j}{\|z_i\| \|z_j\|} \tag{2.12}$$

$$\ell_{i,j} = -\log \frac{\exp\left(\text{sim}\left(z_i, z_j\right)/\tau\right)}{\sum_{k=1}^{2N} \mathbf{1}_{[k\neq i]} \exp\left(\text{sim}\left(z_i, z_k\right)/\tau\right)} \tag{2.13}$$

$$\mathcal{L}_{\text{NT-Xent}} = \frac{1}{2N} \sum_{k=1}^{N} [\ell(2k-1, 2k) + \ell(2k, 2k-1)] \tag{2.14}$$

where z_i is the representation of the ith data point, N is the number of data points, τ is a temperature parameter and sim is the cosine similarity. The positive pairs are the ith and jth data points used in the numerator. The negative pairs are all other data points and their pairs used in the

denominator.

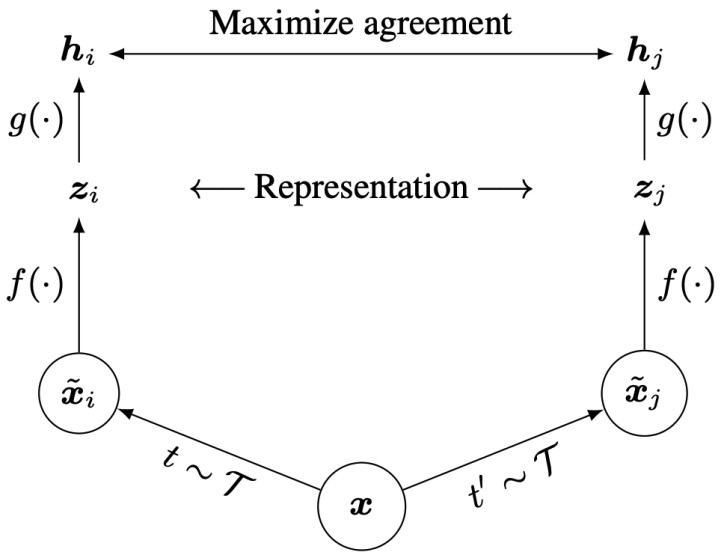


Figure 2.3: Flow through the architecture of SimCLR. A positive pair for x is sampled and first encoded by the encoder network $f(\cdot)$ to produce the representations z, which can later be used for downstream tasks. For training, the representations are run through the projection head $g(\cdot)$ to bring them into the representations space used for the contrastive NT-Xent loss. Taken from T. Chen et al., 2020.

DRL algorithms that leverage contrastive learning typically use state and next-state pairs (s_t, s_{t+1}) as positive pairs, and other collected states as negative pairs (Aubret et al., 2021; S. Li et al., 2021a). This type of temporal learning is reminiscent of the Contrastive Learning Through Time (CLTT) paradigm (Schneider et al., 2021) used for computer vision and object detection. Algorithms that use AEs or VAEs rely on the reconstruction loss and do not need positive or negative pairs (Hafner et al., 2022; Pong et al., 2020). A third option is to predict the temporal distance in timesteps between states (Mendonca et al., 2021):

$$\mathcal{L} = |\|z_t - z_{t+k}\|^2 - k| \tag{2.15}$$

The temporal distance space can make for a natural representation space if the downstream application is to use the representations for measuring the distance between states, e.g., current state and goal state in GCRL.

3. Problem Setup

This chapter introduces the problem setup defined to build and test algorithms that are in line with the aim and objectives of this thesis (see chapter 1.2). The setup contains the agent, the environment and the DRL framework used to conduct the experiments. Experiments with real-world robots are notoriously difficult, so this thesis focuses on 3D simulated environments. As the underlying physics engine, the research proven and newly open-sourced Multi-Joint dynamics with Contact (MuJoCo) engine is used to model and simulate the agent and its environment. (Todorov et al., 2012).

3.1 Agent: Ant

To make interactions possible in 3D simulated environments, the RL agent needs a suitable body. The robotic representation of the agent used for all the following experiments is called ant, a simple quadrupedal 3D robot often used in RL benchmark environments (Schulman et al., 2015).

The ant's body is made of a spherical torso in the center with four legs attached to it (see Figure 3.1). Each leg consists of two links, a cylindrical thigh and a cylindrical shank, modeled with MuJoCo capsules. The ant is controlled by 8 motors, one for each hinge joint between the torso-thigh and thigh-shank connections. This results in an 8-dimensional action space, where each dimension consists of the torque applied to the respective motor. The torques are mapped to the continuous range of $[-1, 1]$.

The observation space consists of 29 dimensions: The global x,y,z position of the torso (3), the orientation of the torso as a quaternion (4), the angles of the joints (8), the velocity of the torso (3), the angular velocity of the torso (3) and the angular velocities of the joints (8).

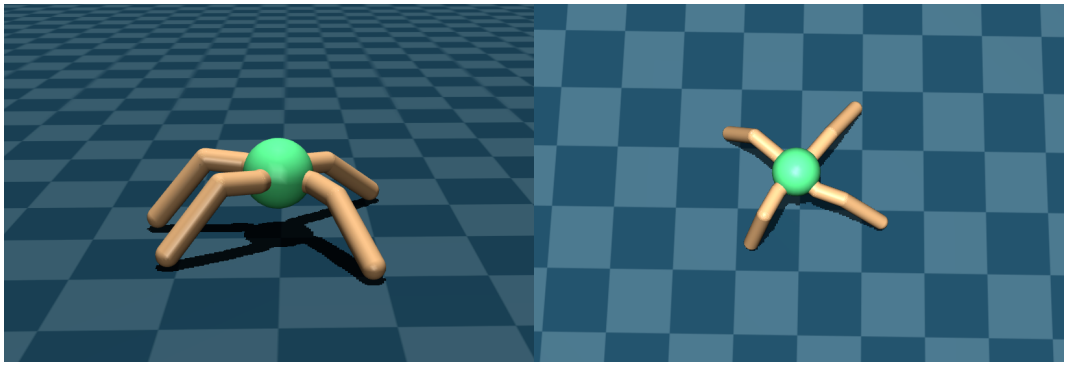


Figure 3.1: MuJoCo rendering of the ant robot consisting of a green torso and four golden legs. (Left) Side view of the ant. (Right) Top view of the ant.

3.2 Environment: Maze

Designing an environment that is challenging enough to not be solved by simple RL algorithms but not too complex so that making progress is still possible is a crucial part of the problem setup. For all experiments in this thesis, a 3D maze environment is used, which is typically found in GCRL research (Nachum et al., 2018). In combination with the ant agent, it is referred to as the AntMaze environment in the literature. The Datasets for Deep Data-Driven Reinforcement Learning (D4RL) (Fu et al., 2020) framework provides three sizes of the maze, which are shown in Figure 3.2. The u-shaped maze is chosen for the main experiments in this thesis, as it is the smallest one but sufficient to develop and test algorithms and it is consistently used in the literature (S. Li et al., 2021a; Nachum et al., 2018).

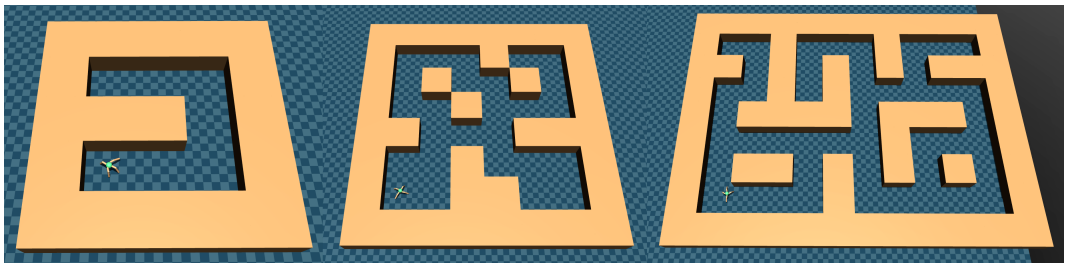


Figure 3.2: MuJoCo rendering of the maze environment in its three sizes as defined in D4RL. (Left) Small u-shaped maze. (Middle) Medium sized maze. (Right) Largest maze.

Most of the GCRL or HRL algorithms use a two-dimensional goal space

for the AntMaze environment, consisting only of the targeted global x,y position for the ant (Aubret et al., 2021; S. Li et al., 2021b; S. Li et al., 2021a; Pong et al., 2020). As stated as one of the objectives of this thesis, there should not be any external prioritization or limit on the goal space. A general learning agent should be able to reach goals in the complete state space. Therefore, in all the following experiments, the goal space will be the same as the 29-dimensional observation space. This makes the developed methods harder to compare to previous work but also more general. Even though the global positions remain for now, they are generally not available to natural agents in the real world, so being able to reach goals also defined by other dimensions of the observation space makes for a more realistic setup.

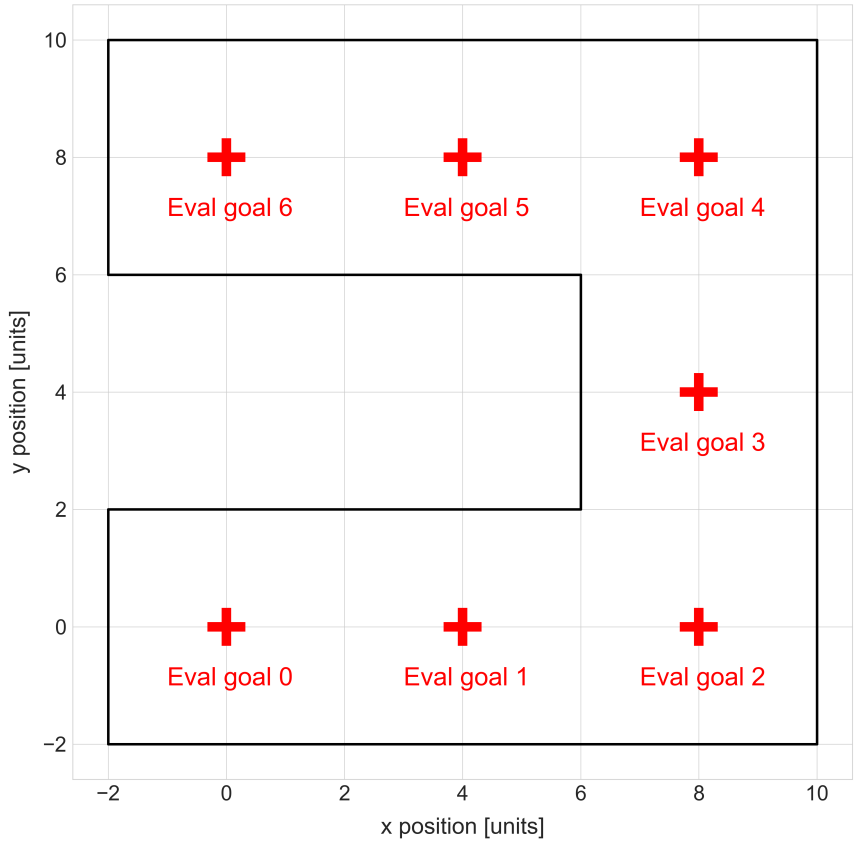


Figure 3.3: Seven evaluation goals in the u-shaped AntMaze environment.

To test whether the agent is able to reach unknown downstream goals, a set of seven evaluation goals is defined that can be used in evaluation phases during training. The seven goals are nearly identical and differ only in their global x,y positions. All other dimensions are taken from an upright standing position of the ant with minimal velocity. The 7th evaluation goal is the hardest to reach, as the agent always starts at position $(0, 0)$ (bottom left of the maze) with a small amount of noise. Analyzing the agent's progress in the global x,y part of the state/goal space is the easiest to illustrate and the most challenging for the agent because of the non-linear shape of the maze. As noted above, all other dimensions are still part of the goal space and can be used to analyze the learning progress of those dimensions as well.

An episode in the AntMaze consists of 5000 raw simulation steps, where the agent's action is repeated 4 additional times until the next observation is received. This results in a total of 1000 steps per episode. For the developed algorithms, it is a further design choice if an episode can end before the time limit, e.g., when the agent gets close enough to its goal.

3.3 Framework: RL-X

Experiments in DRL are notoriously compute intensive, as the training loop consists not only of optimization steps, like in standard ML, but also of many simulation steps. This makes it important to use a framework that can leverage the available hardware, especially accelerators like GPUs, to its fullest while remaining flexible for research purposes. Therefore, this section presents RL-X, a DRL framework developed during this thesis, which focuses on speed and flexibility.

Prominent open-source frameworks, such as Stable-Baselines3 (SB3) (Raffin et al., 2021), RLlib (Hoffman et al., 2020) or Acme (Liang et al., 2018), offer community-proven implementations with robust hyperparameter choices. However, they are not fully designed for pure research purposes, because

they are not easily extendable and often lack flexibility through extensive use of modularity, which harms research development speed and prototyping. Additionally, SB3 and RLlib use PyTorch (Paszke et al., 2019) as the underlying Deep Learning framework, while faster frameworks built on JAX (Bradbury et al., 2018) are available, as used in Acme. CleanRL (Huang et al., 2022) is a good example of a research-driven framework with its simple and clean code and single-file implementations. As much as it is a great resource for research, it also results in the fact that it needs to provide the same algorithm multiple times to support different environment types and variations, e.g., the popular Proximal Policy Optimization (PPO) algorithm is implemented 12 times with small alterations. RL-X is a compromise between the two, offering fully independent and self-containing single directory implementations of algorithms, but also providing a generic interface for environments and algorithms to allow for easy mixing and matching.

Table 3.1 and 3.2 show the currently implemented algorithms and supported environments in RL-X.

Table 3.1: Algorithms in RL-X

Algorithm	Deep Learning framework	Category	Reference
AQE	Flax	Off-policy	Wu et al., 2022
DroQ	Flax	Off-policy	Hiraoka et al., 2021
ESPO	PyTorch, TorchScript, Flax	On-policy	Sun et al., 2022
MPO	Flax	Off-policy	Abdolmaleki et al., 2018
PPO	PyTorch, TorchScript, Flax	On-policy	Schulman et al., 2017
REDQ	Flax	Off-policy	M. Chen et al., 2021
SAC	PyTorch, TorchScript, Flax	Off-policy	Haarnoja et al., 2018a
TQC	Flax	Off-policy	Kuznetsov et al., 2020

Table 3.2: Environments in RL-X

Environment type	Providing framework	References
Atari	EnvPool	Bellemare et al., 2013; Weng et al., 2022
Classic control	EnvPool	Brockman et al., 2016; Weng et al., 2022
DeepMind Control Suite	EnvPool	Tassa et al., 2018; Weng et al., 2022
MuJoCo	Gym, EnvPool	Brockman et al., 2016; Todorov et al., 2012; Weng et al., 2022
Custom environments with socket communication		

Some algorithms are implemented with PyTorch and TorchScript (PyTorch + JIT compilation), but all of them have a Flax (Heek et al., 2023) version available. Flax is a NN library that builds on top of JAX, which is a library for automatically differentiating and JIT compiling Python/NumPy code, which can then be run on CPUs, GPUs and TPUs. JAX was developed with fast execution on GPUs and TPUs in mind. It offers a range of vectorization mappings that allow for fast and parallelized execution of the compiled code on these devices.

To benchmark computational performance of RL-X, the hardware components listed below were used and are also the same for all further experiments in this thesis:

- CPU: Intel Core i9-9900K
- GPU: NVIDIA GeForce RTX 2080 Ti
- RAM: Corsair VENGEANCE 4x16GB, DDR4, 3200MHz

RL-X can offer the biggest speedups when using a Flax implementation of

an off-policy algorithm running on a GPU in a computationally fast environment such as MuJoCo. This is because on-policy algorithms calculate their loss function and update their NNs only after a certain number of steps, while off-policy algorithms can do this after every step, which leads to a lot of computation that can be optimized by using JAX. Speedups are less significant for environments with high computational costs attached because the agent's acting time becomes the bottleneck of the training loop. Figure 3.4 shows the relative computational performance of RL-X's SAC implementations over a SB3 baseline in the Gym Humanoid-v4 MuJoCo environment (Brockman et al., 2016). The results are the average of measuring the frames per second over 3 runs of 50k training steps and are normalized by the SB3 CPU version. This experiment resembles a similar setup to the one used for the thesis (MuJoCo + off-policy algorithm + GPU available). RL-X's Flax version achieves a significant speedup of 4.5x over the SB3 baseline. The Flax version is also the only one that benefits from the GPU because for PyTorch the overhead cost for the data transfer between CPU and GPU is higher than the computational gains, at least for this regime of batch size (512) and overall loss function complexity.

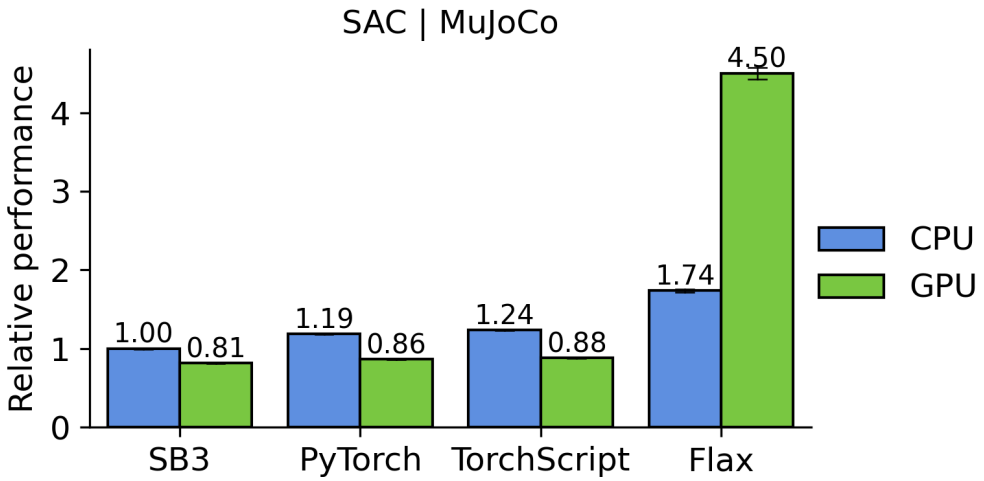


Figure 3.4: Relative computational performance of the RL-X versions for SAC in PyTorch, TorchScript and Flax compared to the SB3 baseline.

RL-X is developed fully open-source under the MIT license and can be found on GitHub: https://github.com/nico-bohlinger/RL-X

4. Goal-Conditioned RL Baseline

This chapter describes the used GCRL baseline algorithm. It does not use any ideas of IM and relies only on a mix of current methods from the literature for exploration and goal reaching. The sections illustrate the iterative development of the algorithm with plots from the experiments. Each learning run trains either directly on the evaluation goals for development purposes or is evaluted on them separately every 100k steps with 10 episodes per goal. For all experiments, three seeds are used to show the variance of the results. The standard deviation over the seeds can be seen through the shaded area in the plots, where the mean is shown as a solid line in the middle.

TQC is used as the core DRL algorithm for all experiments. The critic and policy consist both of 2x256 (layer x neurons) NNs with Rectified Linear Unit (ReLU) activations and their respective policy and critic head on top. The networks are trained with a batch size of 512 and a learning rate of 0.0003.

4.1 Simple Reward

To highlight the importance of using a good representation space for the reward calculation later on, the first experiments in Figure 4.1 are performed using simple reward functions. The reward is calculated as defined in equation 2.11 in x,y space, state space or a random representation space, where the latter is represented with a randomly initialized 3x256 NN. The goal during training is only the evaluation goal $(8, 0)$. It can be seen that the agent has no trouble reaching the goal in 1M steps in either of the three spaces. Using the x,y space performs the best as it corresponds directly to the evaluated distance in the x,y space to the goal. The complete state space and the random representation need slightly more steps as the agent

has to learn to distill the relevant dimensions or rather simply does not know that it is evaluated only in the x,y space.

The maze consists of a strong non-linearity in the x,y space through the u-shaped corridor, which blocks the direct path to goals in the upper part of the maze. Therefore, the agent has to learn to traverse along the corridor to reach those goals. The raw state space does not capture this non-linearity and the agent falls into a local minimum trying to reach the upper goals by walking against the blocks in the middle. Going around the blocks would give temporary worse rewards for a too long time, so the agent does not learn to do so. Figure 4.2 shows the results of the agent trained on all evaluation goals with distances in state space. The goals up to $(8, 8)$ (in the top right corner of the maze) are reachable, but the agent is unable to learn to go beyond that for the reasons given above.

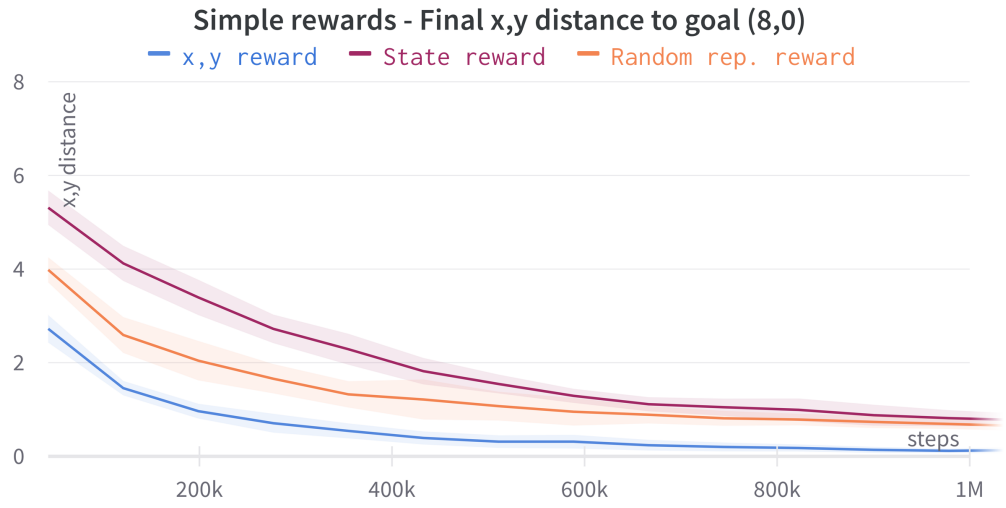


Figure 4.1: Simple bases for the reward calculation are tested with either the x,y space, the state space or a random representation space. Note that all three versions start with a similar random policy. The difference in the first distances is an artifact of the plotting tool as it cuts some of the initial steps for smoothing purposes.

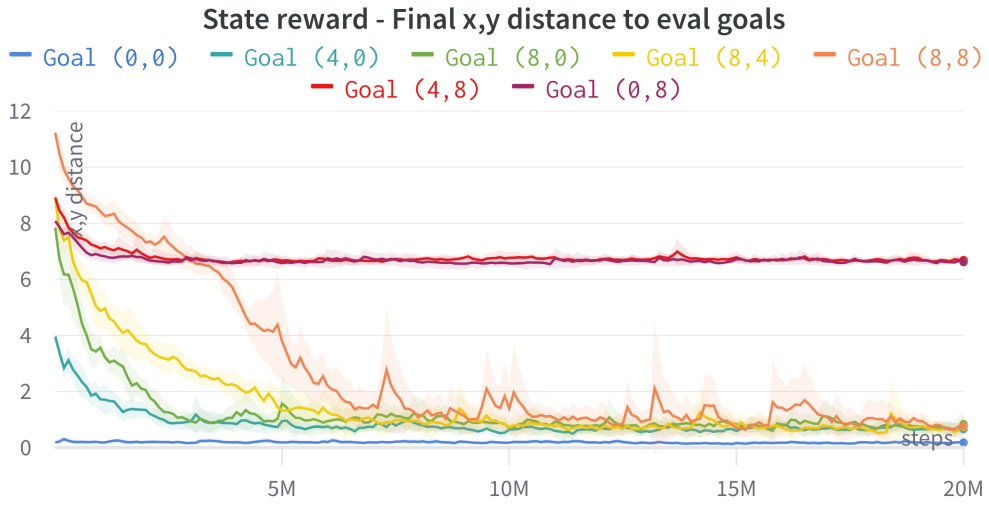


Figure 4.2: Learning all seven evaluation goals with state space reward.

4.2 Representation-based Reward

A trained representation space can capture the dynamics of the environment and distill the information in its latent space. This can be done in different ways as described in chapter 2.6. The SimCLR method is trained with batches of 512 (s_t, s_{t+1}) pairs, a 3x256 encoder network with ReLU activations, a 1x256 projection head and different temperature values. A representation learned with an inverse dynamics model (Pathak et al., 2017) is also tested. This inverse dynamics model encodes the state space into a latent space and from there learns to predict the used action between two consecutive states. The experiments in Figure 4.3 show that the temperature parameter in SimCLR is highly volatile and crucial to get right for the representation to work as the base of the reward calculation. The best result is achieved with a temperature of 0.1, which is the only setup that beats out the random representation and is going to be used for the rest of the experiments in this chapter. Using an inverse dynamics model does not work out of the box, but it might still lead to good results with sufficient tuning.

Training the agent with the SimCLR representations on all seven evaluation

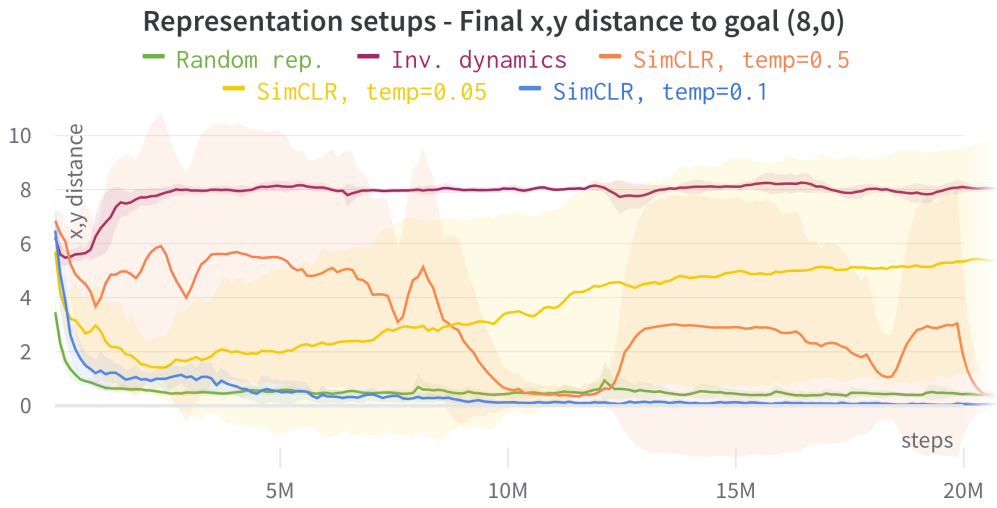


Figure 4.3: Different representation learning setups. The inverse dynamics model and SimCLR with different temperature values are tested.

goals leads to non-optimal results as shown in Figure 4.4. Even reaching the easier goals $(4, 0)$ and $(8, 0)$ is not reliably achieved during the training process. The key issue is the ever-growing representations, as illustrated in Figure 4.6. They result from the fact that during training the agent sees more and more states from the same part of the state space, especially when its converging to a less stochastic policy. This means that the entropy of the replay buffer is becoming smaller and so the available negative pairs become more similar to the positive pairs. The SimCLR loss keeps increasing the distance between the positive and negative pairs, which finally leads to the ever-growing representations. This constant representation growth causes instability in the training process, but it can be mitigated in different ways, e.g., by using a tanh activation for the last layer in the encoder network. The tanh squashes each dimension to the range $[-1, 1]$. With more stable representations, the agent in Figure 4.5 is able to reach goals until $(8, 8)$ reliably and $(4, 8)$ more likely than with the state space reward (compare to Figure 4.2).

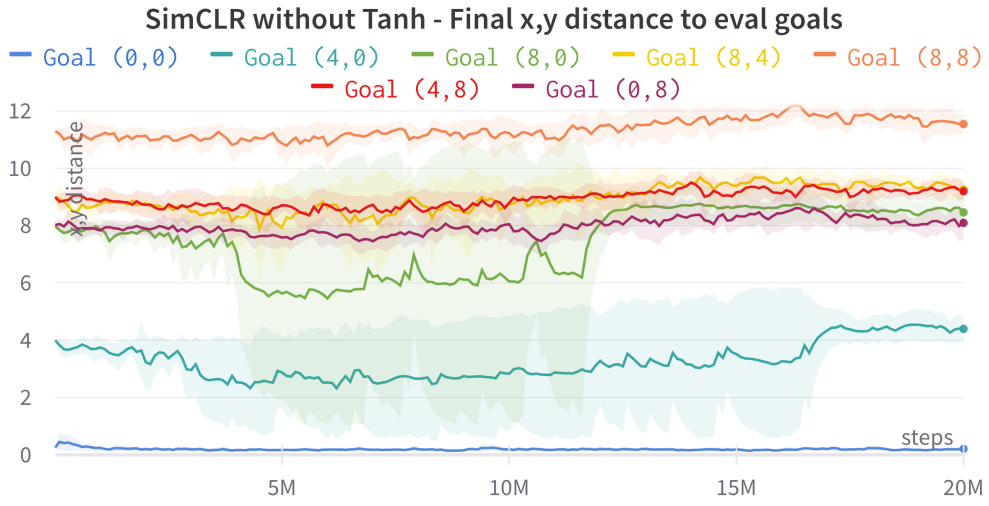


Figure 4.4: Learning all seven evaluation goals with SimCLR representations without any squashing of a tanh activation leads to ever-growing representations.

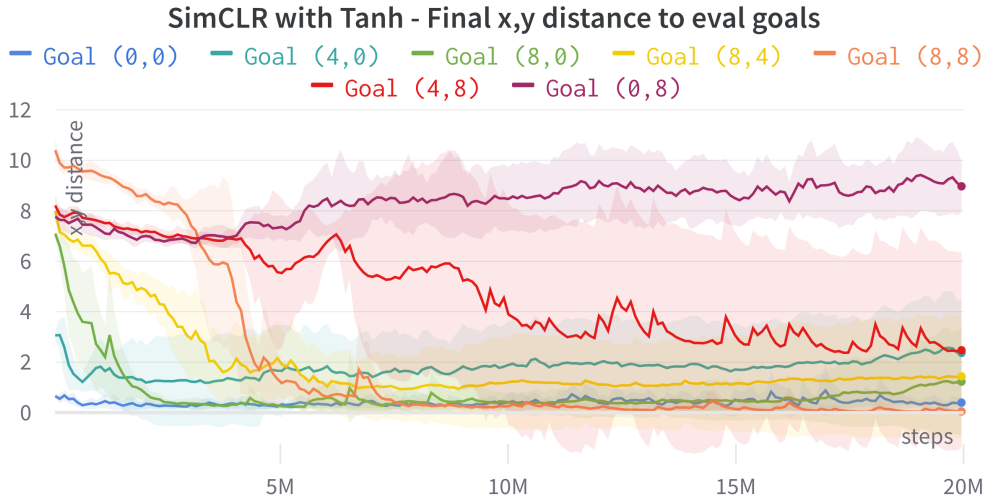


Figure 4.5: Learning all seven evaluation goals with SimCLR representations with a tanh activation on the last layer keeps the size of the representations in check.

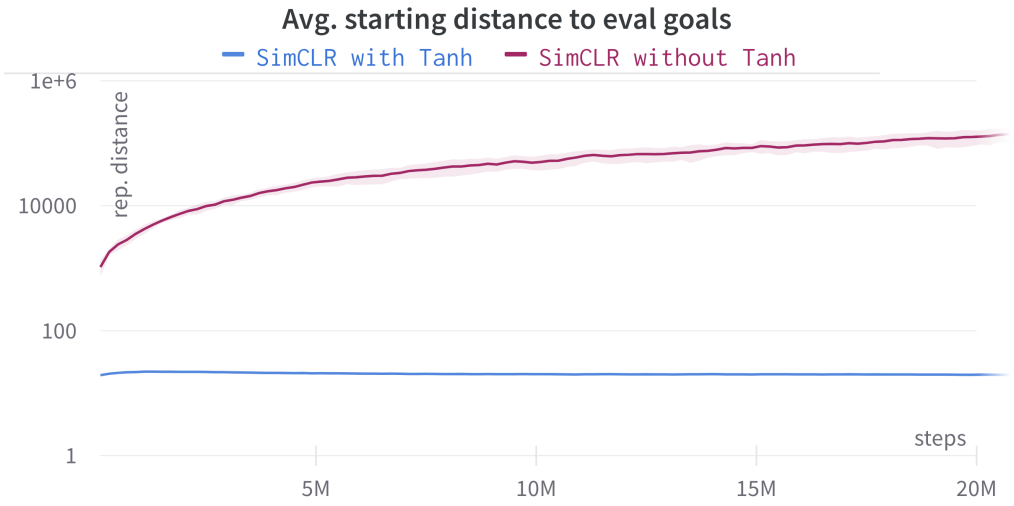


Figure 4.6: SimCLR leads to ever-growing representations without squashing. The plot shows the average distance between the starting state and the seven evaluation goals. Note that the y-axis is logarithmic.

4.3 Goal Buffer

Goals for downstream applications are often not known in advance. To simulate this, the set of evaluation goals should not be used during training. Chapter 2.5 introduced different algorithms to store or generate goals for the agent. The goal buffer as described in DISCERN is used in this chapter to test the performance of this family of algorithms. DISCERN has two different goal buffer setups:

- **Random**: Newly encountered states are added to the goal buffer if the buffer is not full yet. If the buffer is full, a random goal is replaced by the new state with the probability of p_{replace}.

- **Diverse**: Newly encountered states are added to the goal buffer if the buffer is not full yet. If the buffer is full, a random goal is replaced by the new state if the distance of the new state to the mean of the buffer is larger than the distance of the random goal to the mean of the buffer. Replacement in the described case happens with the

probability of p_{replace} or in the opposite case with the probability of $p_{\text{add-non-diverse}}$.

Both setups are tested with $p_{\text{replace}} = 0.001$, $p_{\text{add-non-diverse}} = 0.001$ and a goal buffer size of 1024, as those values were found to work well in the original DISCERN paper. Figure 4.7 shows that with both goal buffer setups the agent learns to reach the goals in the buffer but it only slowly expands the active goal space. The agent is visiting the same states over and over again to learn to reach the goals in the buffer, but this means that the states that are added to the buffer are also more of the same. This feedback loop makes it hard for the agent to explore more of the goal space.

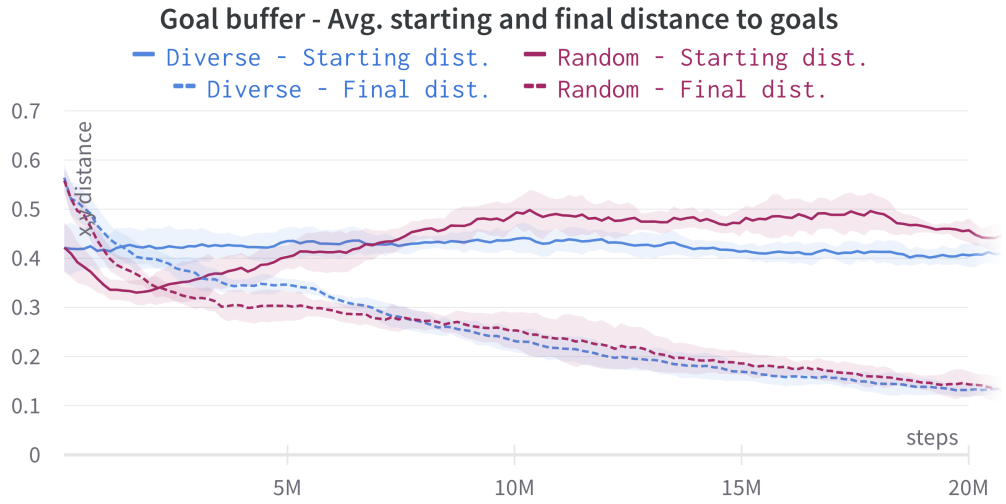


Figure 4.7: Average starting and final distances to the goals sampled from the goal buffer. The random and diverse setups perform roughly the same.

Disentangling the process of exploring the environment for goals from the process of learning to reach those goals will be investigated in the following chapters. The results from Figure 4.5 will be the baseline for the developed algorithms. It should be noted that the training process is not yet independent of the evaluation goals, as the experiments with the goal buffer were unsuccessful.

5. Goal-Conditioned RL with Prior Intrinsic Exploration

This chapter develops a novel family of algorithms called Goal-Conditioned RL with Prior Intrinsic Exploration (GC-π) that combines IM, representation learning and GCRL to separate the two processes of exploration and goal learning. An algorithm of this family consists of three distinct phases:

1. Train an agent with IM that explores the environment and collects a dataset of (s, a, s') transitions.

2. Train a representation space offline on the collected dataset, which can be used as the distance metric for later reward calculations.

3. Train a goal-conditioned policy by utilizing the learned representations and the collected dataset. This can be done offline or online.

The following sections of the chapter match the three phases and explain them in detail with experiments on different algorithmic design choices.

5.1 Prior Intrinsic Exploration

The goal of the prior intrinsic exploration phase is to explore the environment as much as possible and collect a dataset of (s, a, s') transitions. The exploring agent has to discover all parts of the state space that might be relevant for the later training of the goal-conditioned policy. This is necessary so that the learned representation space is accurate and the dataset contains enough states that can act as representative training goals for the downstream evaluation goals.

All experiments use the same TQC setup as described in chapter 4 as the core DRL algorithm. For the IM method used to calculate the reward, four different setups are tested:

- **RND**: The RND algorithm (Burda et al., 2018) is used as described in chapter 2.4 with the intrinsic reward:
  $r_{\text{RND}} = r_t^i = \|f_\psi(s_{t+1}) - \hat{f}_\varphi(s_{t+1})\|^2$ (see equation 2.9). In all experiments, the random feature encoder $f_\psi(s)$ is a 4x256 NN and the predictor $\hat{f}_\varphi(s)$ is a 6x256 NN. Both networks have ReLU activations in-between the layers. To warm up the running average for the observation normalization, 100 random steps are taken at the beginning of a learning run.

- **NovelD**: Novelty Difference (NovelD) (Zhang et al., 2021) builds on RND and computes the intrinsic reward as:
  $r_{\text{NovelD}} = r_t^i = \max[r_{\text{RND}}(s_{t+1}) - \alpha \cdot r_{\text{RND}}(s_t), \beta] \cdot \mathbb{I}[N_e(s_{t+1}) = 1]$, where $r_{\text{RND}}(s)$ is the RND reward and $N_e(s)$ is the visitation count for a state. Originally, NovelD only rewards states that were seen for the first time, but as the state space in the AntMaze environment is continuous, naively counting visitations is problematic and therefore omitted in the following experiments. That means every state is eligible for the reward. The parameters α and β are set to 0.5 and 0.0 respectively based on the original NovelD paper.

- **E3B**: E3B (Henaff et al., 2022) tries to generalize the raw visitation counts, which are needed to compute episodic exploration rewards, to continuous state spaces. The within-episode intrinsic reward is calculated as:
  $r_{\text{E3B}} = r_t^i = f_\psi(s_t)^\top C_{t-1}^{-1} f_\psi(s_t)$, where the f_ψ is a feature encoder. The matrix C represents the generalized visitation count and is updated with $C_{t-1} = \sum_{i=1}^{t-1} f_\psi(s_i) f_\psi(s_i)^\top + \lambda I$. This means that the term C_{t-1}^{-1} in the reward calculation is the inverse of the generalized visitation count, so the final E3B reward corresponds to $1/N_e(s_t)$ in the raw counting approach. The parameter λ is set to 0.1 for regularization and f_ψ is a 2x256 NN with ReLU activations.

- **NovelD + E3B**: As explained above, the standard visitation count used in NovelD is not applicable to the continuous state space in AntMaze, but the E3B approach can function as a drop-in replacement. Mixing E3B with intrinsic across-episode rewards like NovelD might lead to a better exploration performance, but this was only suggested and not tested in the original E3B paper. Therefore, the intrinsic reward for this combination is calculated as:
  $r_{\text{NovelD+E3B}} = r_t^i = \max[r_{\text{RND}}(s_{t+1}) - \alpha \cdot r_{\text{RND}}(s_t), \beta] \cdot r_{\text{E3B}}(s_t)$

The plots in Figure 5.1, Figure 5.2, Figure 5.3 and Figure 5.4 show the visitation maps of the four IM methods after 10M steps of training. Each dot in the map represents a state visited by the agent in x,y coordinates. The color of a dot shows in which episode the state was visited for the first time during training. The lighter the color, the earlier the state was visited. It should be kept in mind that the map only shows the exploration of the agent in the x,y part of the state space, i.e., an algorithm that explores those dimensions particularly well does not necessarily explore the other dimensions to the same extent.

Figure 5.1 shows the visitation map of the RND setup. The agent explores the environment fully and visits basically all parts inside of the maze. Compared to the other methods, RND explores the maze the most extensively and also the fastest, as the agent covers huge parts of the maze already in the first 1000-2000 episodes. With more episodes, the agent extends its exploration to the edges of the maze. This can even lead to the agent escaping the maze. One such escape can be seen in the bottom left corner of the map. The space outside the maze is obviously novel and therefore produces a strong intrinsic reward signal when reached, but it is hard for the agent to reliably build enough momentum to jump over the wall and explore the outside space.

Figure 5.2 shows the visitation map of the NovelD setup. The agent explores the maze in a similar way to RND, but its exploration is more sparse and not as far reaching. NovelD differs from RND by giving the difference in novelty between two consecutive states as the intrinsic reward instead of only using the novelty of the next state. Playing around with the coefficient

for the subtraction, α, and the minimum reward, β, changes the exploration behavior slightly. When reducing α to 0.0 and β to a small enough negative value, the RND method and its performance can be fully recovered. NovelD's improvements did not help in the tested AntMaze environment, but it has to be noted that the visitation count was removed, which might be the leading factor for NovelD's original performance gains (Henaff et al., 2022).

Figure 5.3 shows the visitation map of the E3B setup. The E3B agent explores less and especially slower compared to the other methods. Letting the agent train for longer than 10M steps (up until 100M steps in further experiments) showed that the reward kept increasing linearly, so the agent is not done exploring but does so very slowly. It can be hypothesized that the agent might be able to discover the maze just as much as the other methods, but training it for that long was not a viable option in this thesis. An explanation for the slow exploration could be the fact that the original E3B uses a Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997) module for its policy and value networks. The LSTM enables the agent to reason about the states it has already visited during an episode; therefore, it can optimize better for the within-episode reward defined by E3B.

Figure 5.4 shows the visitation map of the NovelD + E3B setup. Naturally, the agent explores the environment similar to NovelD, but the addition of the E3B reward leads to a slightly more extensive exploration as the agent is able to reach further into the maze. Through all experiments, it was also observed that this combination resulted in the most escapes from the maze. Nevertheless, the agent's exploration is still lower compared to the results from RND. For future work, it would be interesting to see if the performance gain of combining NovelD and E3B can be used to improve RND as well. Also adding an LSTM to the networks, similar to plain E3B, could enhance the exploration performance.

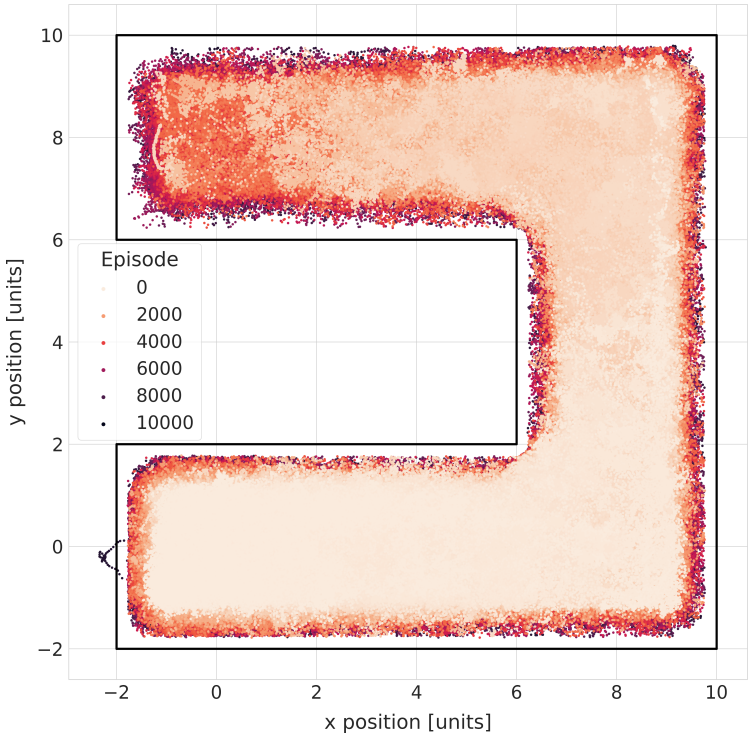


Figure 5.1: Visitation map of the RND agent after 10M steps of training.

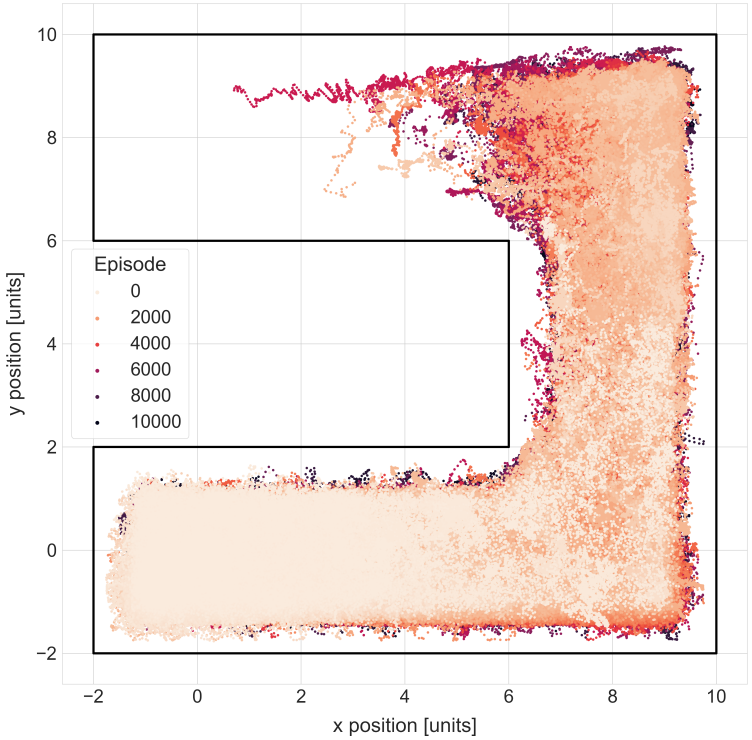


Figure 5.2: Visitation map of the NovelD agent after 10M steps of training.

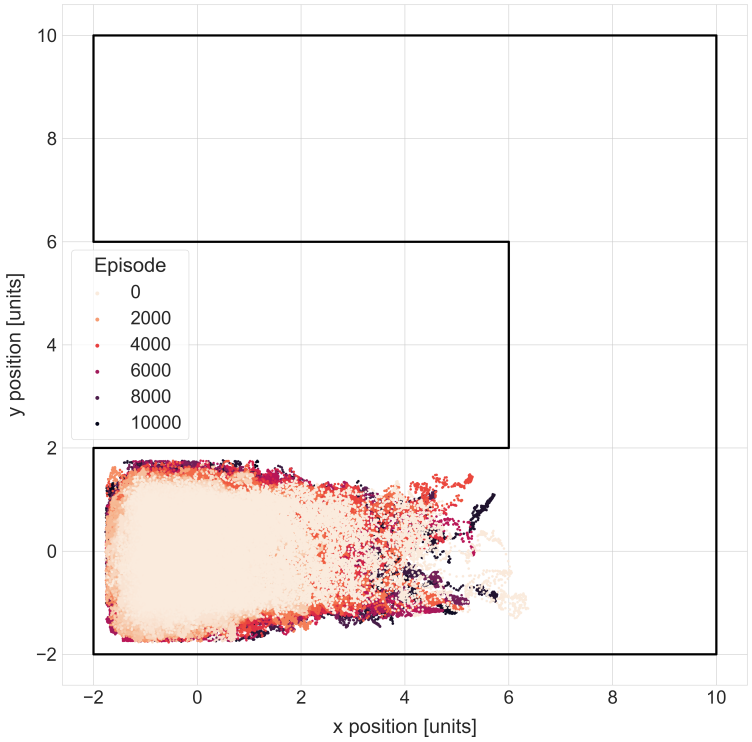


Figure 5.3: Visitation map of the E3B agent after 10M steps of training.

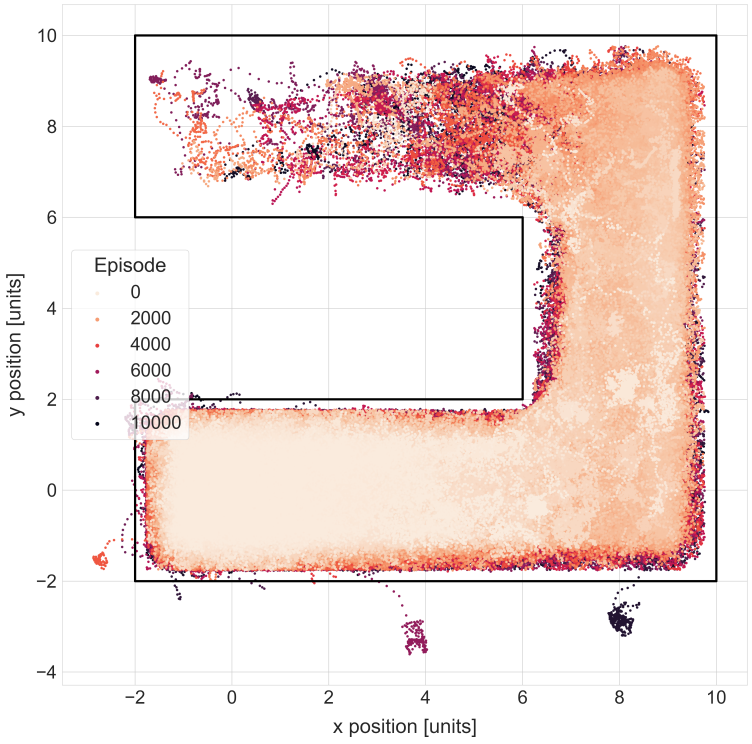


Figure 5.4: Visit. map of the NovelD+E3B agent after 10M steps of training.

RND shows the best exploration performance after 10M steps, so it is selected as the IM method of choice for the rest of the thesis. The prior intrinsic exploration phase for the GC-π algorithms has to provide an exhaustive dataset of (s, a, s') transitions, that covers the most important parts of the state space. To curate the dataset, an RND agent is trained for 100M steps and every transition seen during the training is stored. The produced visitation map is shown in Figure 5.5. The inside of the maze is densely covered by the agent and also some more regular escapes from the maze can been seen in the top right corner.

Figure 5.6 shows the density map for the states in the dataset. The lighter the color, the more often this part of the state space was visited. The bottom left corner is mostly bright yellow and white, which is expected as the agent starts every episode around the position $(0, 0)$. Toward the bottom right the colors fade more into the darker reds, but a yellow trail can be spotted, which is the shortest path to reach further into the maze and was therefore preferred by the agent. The top right corner is strongly yellow and white, which is the area where the agent has escaped the maze somewhat regularly and subsequently got high rewards by the RND algorithm. Going further to the end of the maze, the densities get darker and show even some holes in their coverage. The whole dataset is quite unbalanced, which is to be expected from an unbiased IM method, as it does not directly care about a balanced state space coverage, especially not only in two of the 29 dimensions, which are the x,y positions presented in the maps. This imbalance could be a problem for the representation learning methods later on, which are the basis for the goal-conditioned policy learning at the end. Skewing the sampling processes during the learning procedures in favor of less dense areas could be a solution to this problem. Nevertheless, the dataset covers most parts of the maze and is therefore good enough for the prior exploration phase, as the areas around every evaluation goal are represented in the dataset.

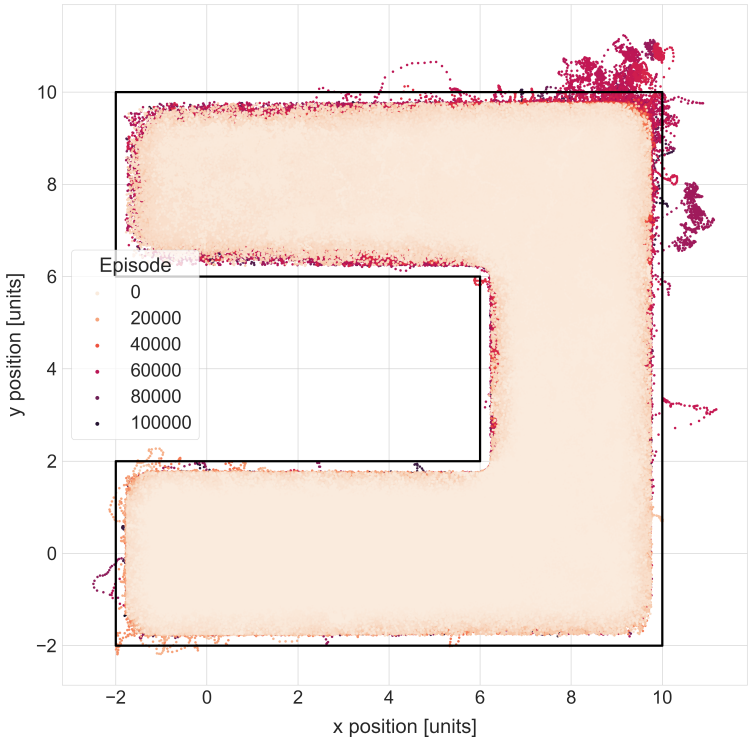


Figure 5.5: Visitation map of the RND agent after 100M steps of training.

Figure 5.6: State density map of the RND agent after 100M steps of training.

5.2 Offline Representation Learning

Learning good representations of the state space is a crucial part of the GC-π algorithms, as the distances in the representation space are later used to calculate the reward for the goal-conditioned agent. In contrast to chapter 4.2, the representations are learned offline. This means that the transitions are sampled from the dataset collected during the prior exploration phase, instead of from recently collected transitions from a replay buffer. Every transition in the dataset is used exactly once during an optimization epoch and multiple of those epochs are run for the training process. Besides SimCLR, the prediction of temporal distances, as explained in chapter 2.6, is tested as another possible representation learning method. A small set of experiments with a triplet loss setup like in S. Li et al., 2021b were also conducted, but the results did not show promising distances in the learned representation space and are therefore not included here.

The evaluation of the quality of a learned representation space is done with distance heatmaps. The distance heatmaps show the distance in representation space between a given x,y position (marked in the title of the plot) and all the positions defined by the grid in the heatmap. Only the x,y parts of the state space are varied and the rest are kept constant, exactly like for the evaluation goals. All distances are normalized by the maximum distance in the plot and multiplied by 10 so that changes in relative distances are easy to spot and the heatmaps can be compared to each other.

5.2.1 SimCLR

The default setup for SimCLR is the same as the one used in chapter 4.2. The following experiments show ablation on the three hyperparameters: temperature, batch size and number of consecutive states for the positive pairs, as they have a big impact on the learned representations. To spare space, only the results of the largest and smallest values tested are compared. Intermediate values typically show interpolation between the two

extremes and do not provide any further insights. More experiments with different network sizes and the euclidean distance loss instead of the cosine similarity loss were also conducted but did not show any improvements in their results.

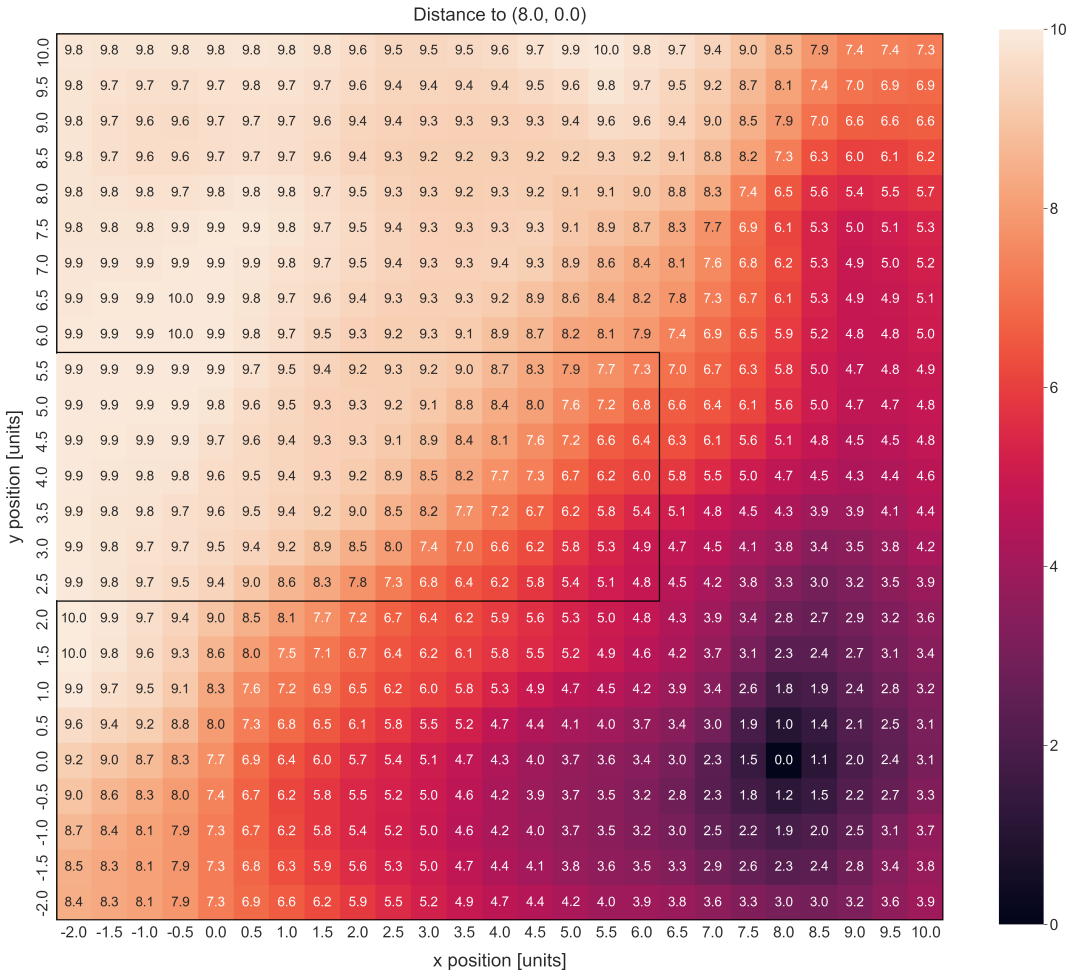


Figure 5.7: Distance heatmap for the position $(8, 0)$ for the default setup with one consecutive state (next-state) as positive pair.

Instead of using only state, next-state pairs (s_t, s_{t+1}), more consecutive states can be used to create more positive pairs for the contrastive loss, which might result in a faster learning process. To evaluate this setup, Figure 5.7 shows the representation distances for the position $(8, 0)$ for the default setup with one consecutive state (only the next-state) and Figure 5.8 shows the same when using four consecutive states. Using more than one consecutive state deteriorates the representation quality and unnaturally placed borders/lines in the representation space can be observed, e.g., cre-

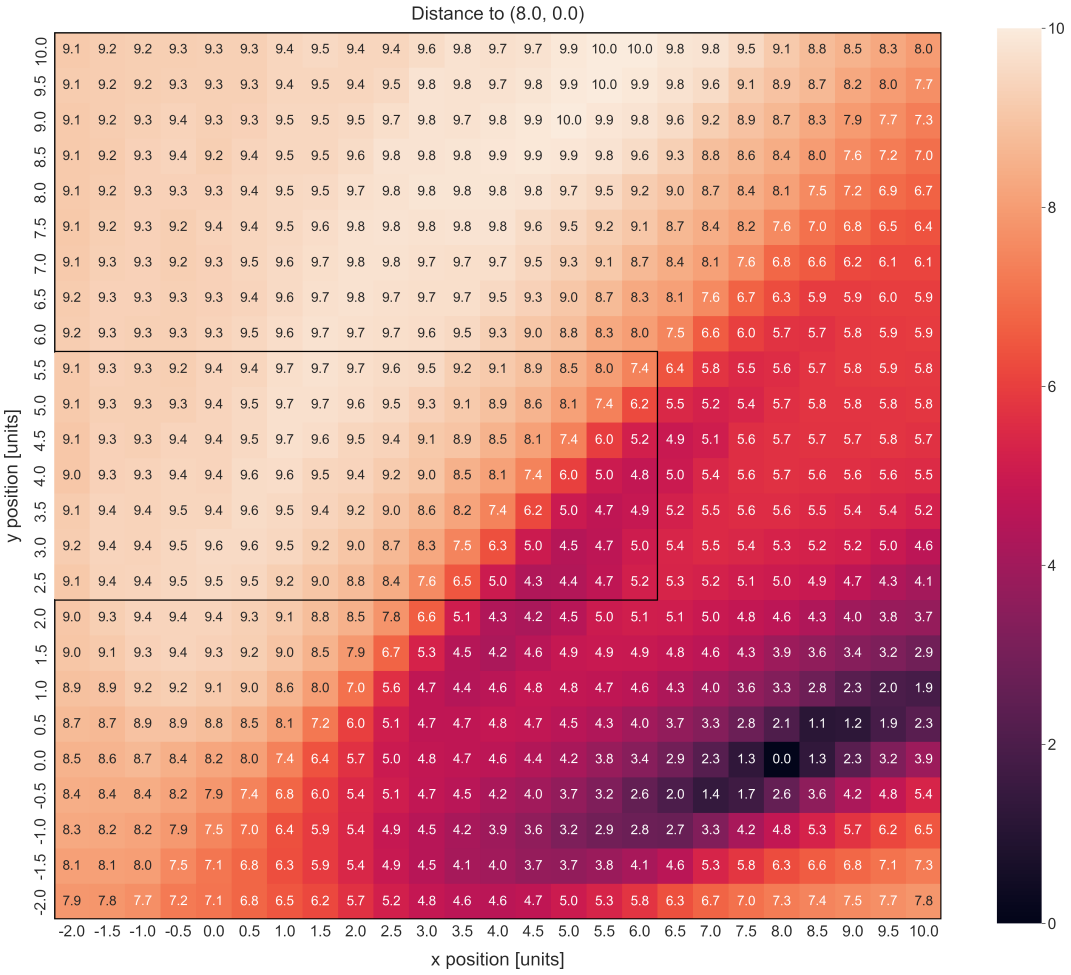


Figure 5.8: Distance heatmap for the position $(8, 0)$ when using four consecutive states as positive pairs.

ating a v-shaped valley of low distances in the middle right of the maze in Figure 5.8. The reason for that might be that the objective of bringing multiple consecutive states closer together in the loss is far harder to optimize for. An experiment with three consecutive states showed less of this effect. In theory, using only one consecutive state should be enough to learn accurate distances as long as the model is trained long enough and generalizes well.

Chapter 4.2 showed that the temperature parameter of the softmax function in the contrastive loss has a big impact on the quality of the learned representations. To investigate the effects of different temperatures, Figure 5.9 illustrates the distance heatmap for the position $(0, 0)$ for the default

setup with temperature 0.1 and Figure 5.10 shows the same when using a temperature of 1.0. It can be seen that the higher temperature stops the proliferation of lower distances to the top right corner of the maze, which is initially caused by the high density of transitions in the dataset for that area.

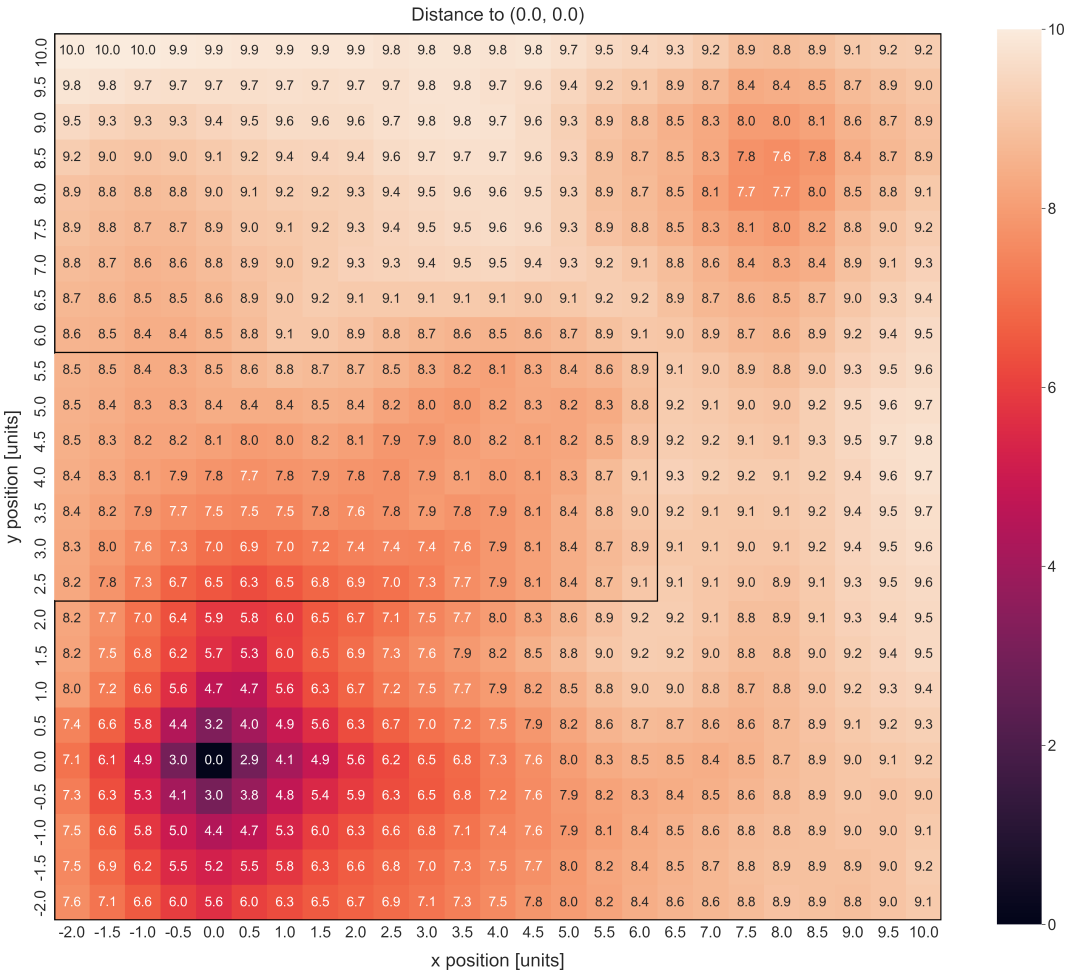


Figure 5.9: Distance heatmap for $(0, 0)$ for the default setup with a temperature of 0.1 for the softmax function in the contrastive loss.

Further distance leakage can be spotted in the blocked middle part of the maze. The dataset contains no transitions there as the agent cannot reach that area during the exploration phase or only through super rare escapes. SimCLR generalizes the learned representations through this part of the maze because there are no positive or negative pairs for it. Combined with further approximation errors, this results in the observed distance leakage. The high temperature in Figure 5.10 can reduce this effect such that the

top part of the maze shows more adequate distances. In general, high temperatures effectively smooth out the representations, which helps in this case, but should only be used with caution as it also smooths over correctly learned structures in the representation space.

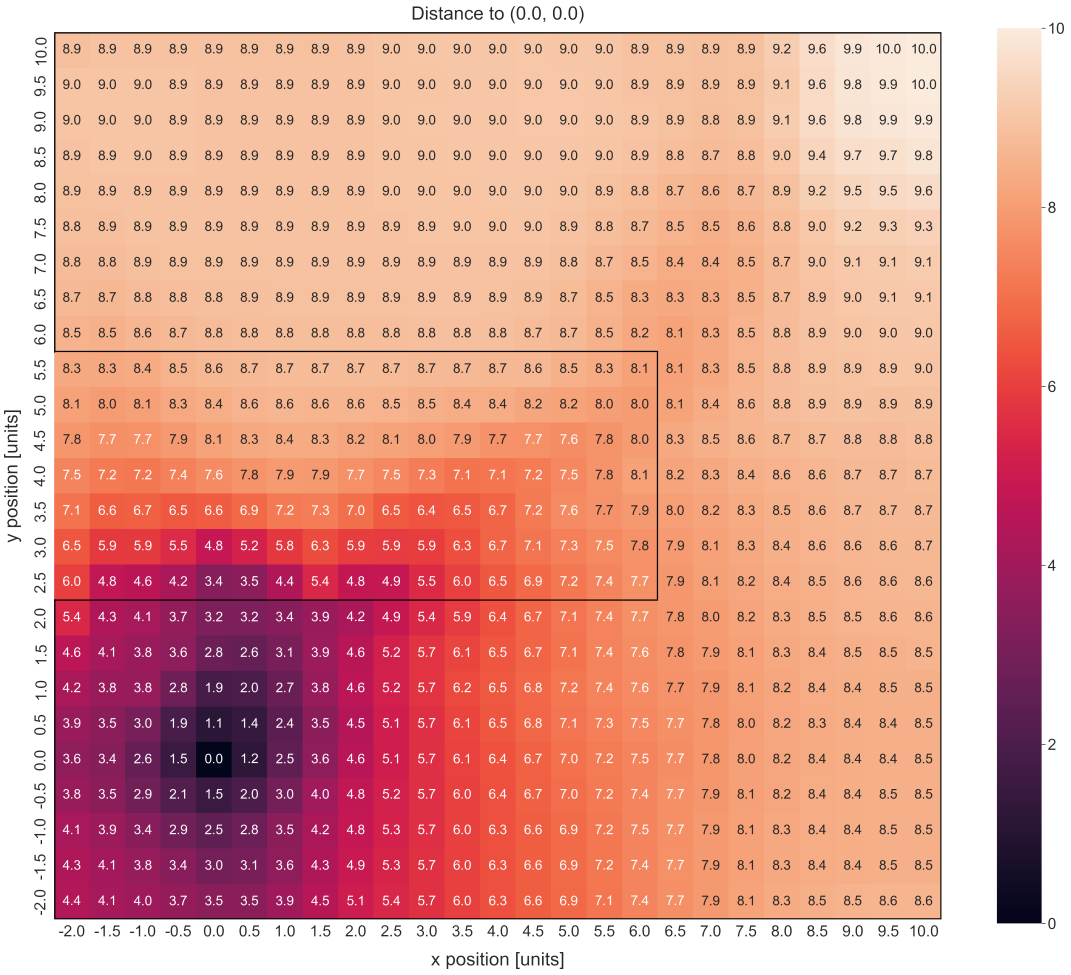


Figure 5.10: Distance heatmap for $(0, 0)$ when using a temperature of 1.0 for the softmax function in the contrastive loss.

SimCLR typically benefits from larger batch sizes. The original paper uses batch sizes up to 8192 samples. Even though the application and setting are different, the same effect might apply here as well. Figure 5.11 shows the distance heatmap for the position $(8, 4)$ for the default setup with a batch size of 512 and Figure 5.12 shows the same when using a batch size of 16382. With the larger batch size, there is a more prominent curve in distances in the middle right of the maze. This curve illustrates that the method learns the underlying structure of this part of the maze better with

the bigger batch size than with the smaller one. All the batch sizes 512, 1024, 2048, 4096, 8192 and 16382 were tested and this effect gradually increased with the batch size. With the size increase, the computational cost increases as well and as even bigger batch sizes do not fit into the GPU VRAM, there were no further experiments with even larger batch sizes. Going from 8192 to 16382 samples in the batch more than doubled the training time but only marginally improved the results. The 100 epochs used for each training run took 19 hours with the batch size of 8192 and 39 hours with the batch size of 16382.

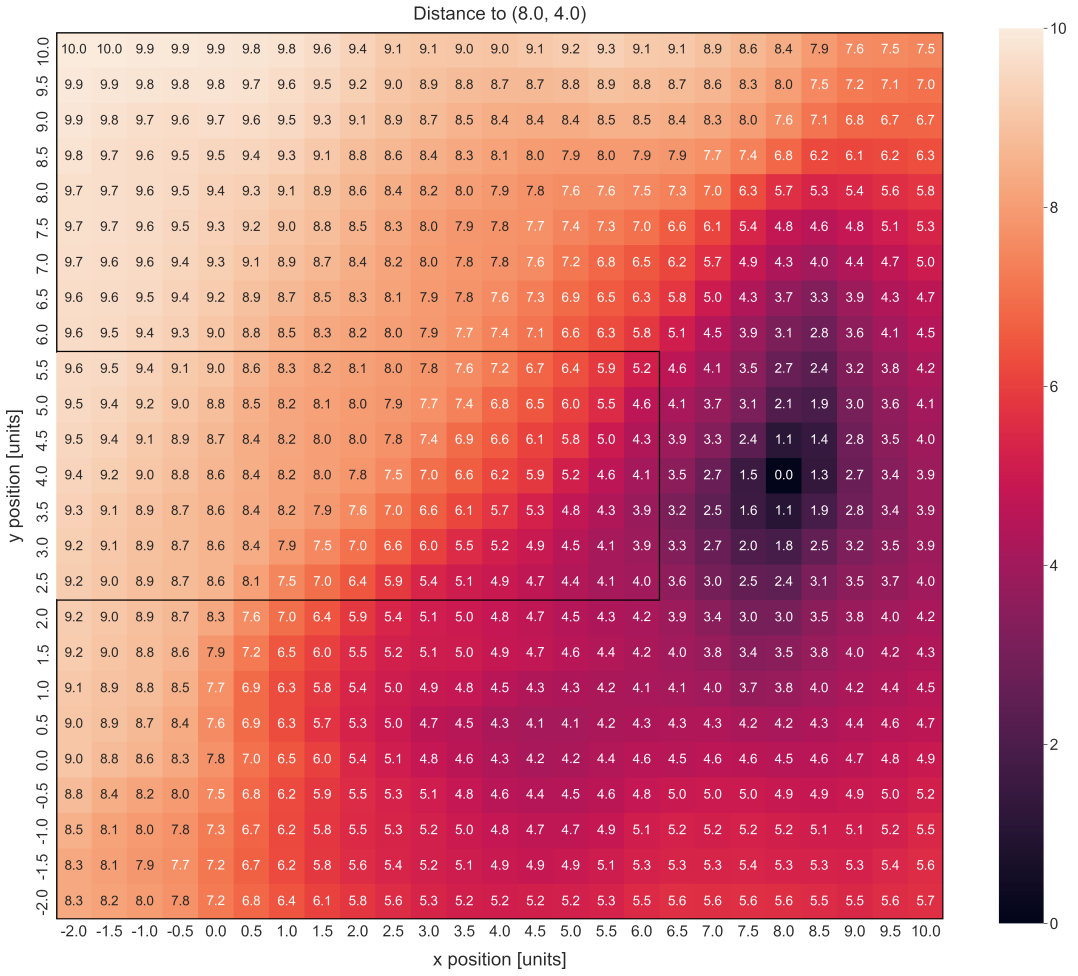


Figure 5.11: Distance heatmap for $(8, 4)$ for the default setup with a batch size of 512.

For the final SimCLR model a batch size of 8192, a temperature of 1.5 and one consecutive state were used. The model was trained for 300 epochs,

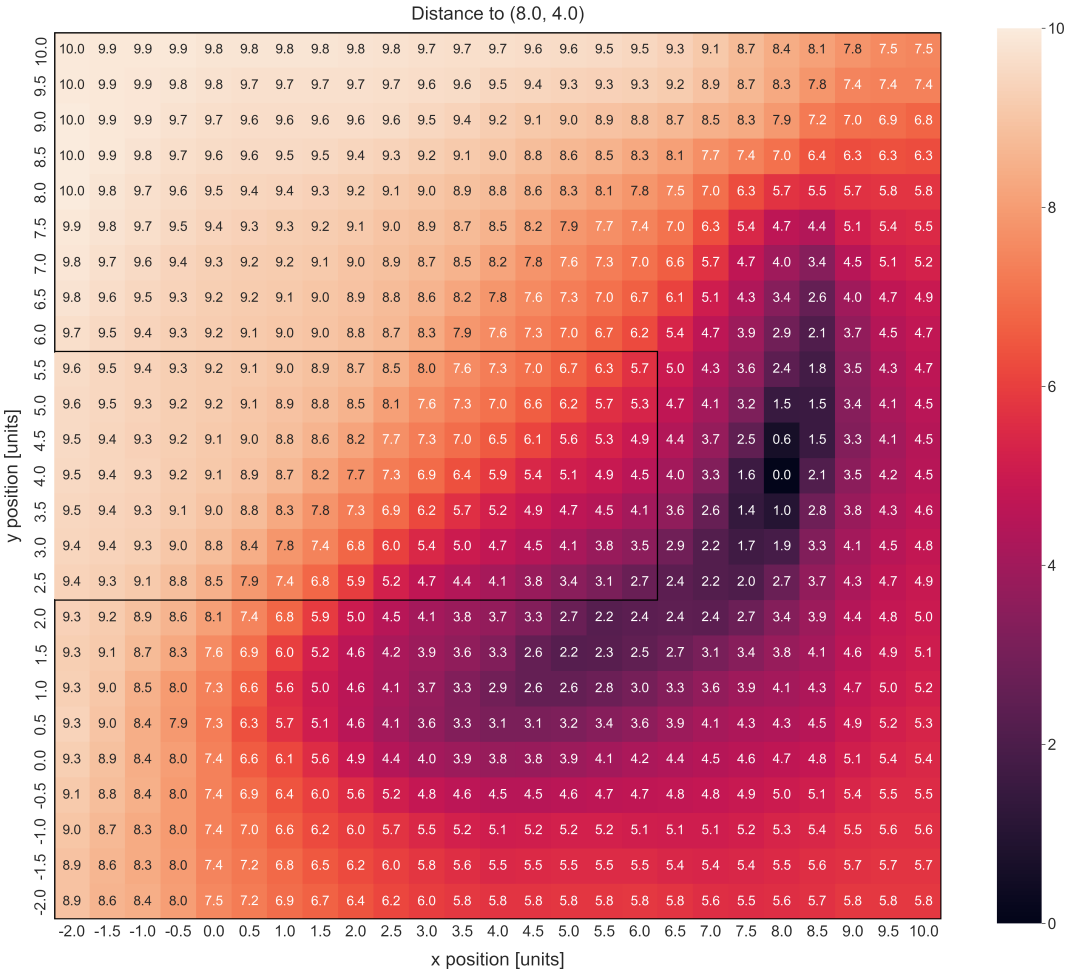


Figure 5.12: Distance heatmap for $(8, 4)$ when using a batch size of 16382.

which is why only a batch size of 8192 was used as the 16382 batch size would have taken too long to train. The temperature of 1.5 was chosen for experimental reasons, but its representation did not look any different to the ones with temperature 1.0. Figure 5.13 depicts the resulting distance heatmap for the position $(0, 8)$ for the final SimCLR model. Even though the representations do not show any strange artifacts anymore from the unbalanced dataset, the distances still leak through the middle part of the maze and more importantly do not capture the underlying structure of the maze too well. The learned distances should fully bend around the corners of the maze to represent the actual distances between the states but this is either not the case or barely visible as can be seen in Figure 5.12. The representation space seems accurate in local neighborhoods, but the global structure is not captured well enough to be useful for guiding an agent

from the starting point to the end of the maze. Local minima just below the middle part in Figure 5.13 would attract the agent and it would be unable to learn to walk around the blocked middle as going further to the right and top would only result in a large space of worse reward. Nevertheless, the learned representations and their distances might still be useful if the local accuracy can be used to navigate from subgoal to subgoal until the agent reaches its final goal.

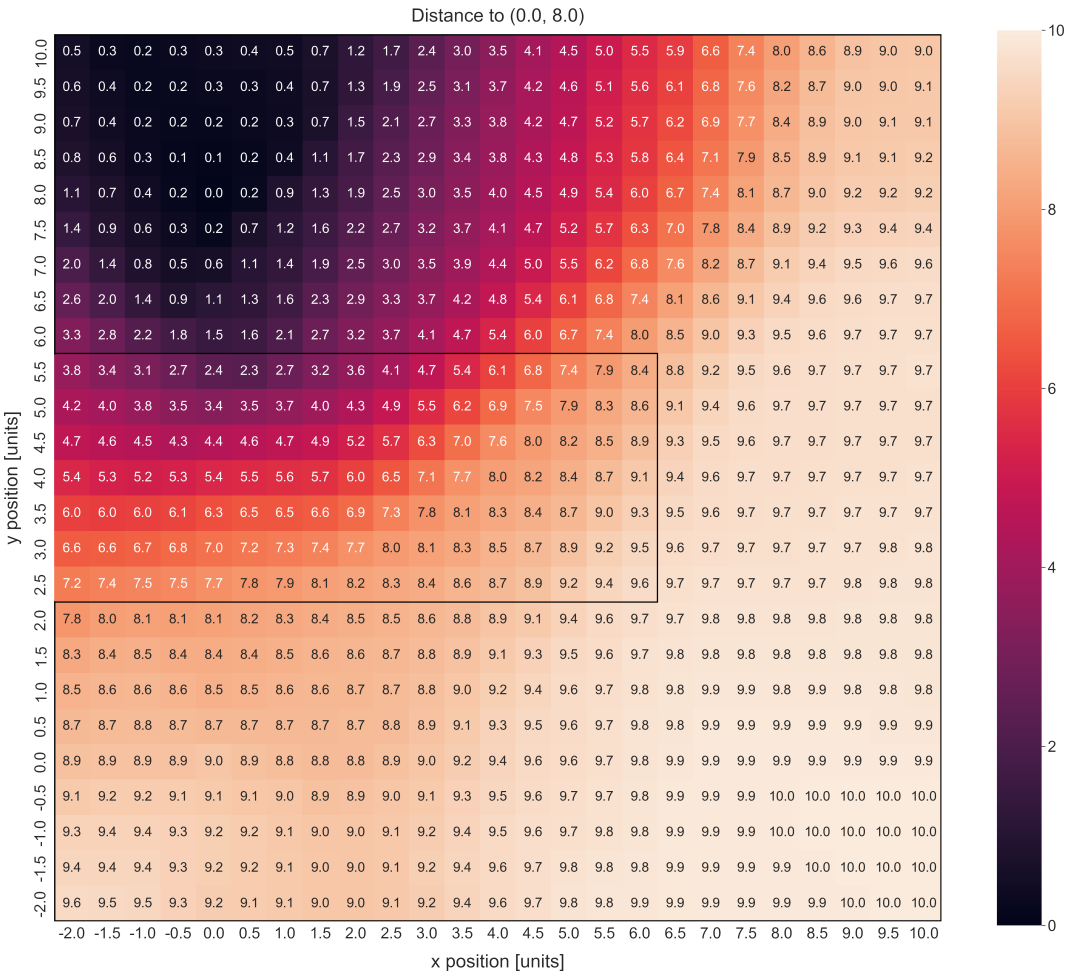


Figure 5.13: Distance heatmap for $(0, 8)$ for the final SimCLR model with a batch size of 8192, a temperature of 1.5 and one consecutive state as the positive pair.

5.2.2 Temporal Distance

Instead of relying on a contrastive objective and inferring the distances from the learned representations, a NN that learns to predict the number of timesteps between two states optimizes for a distance metric directly. In the default training setup, one batch equals one episode of 1000 timesteps sampled from the dataset. Using every state pairing results in a 1000x1000 matrix, which is optimized with the loss defined in equation 2.15.

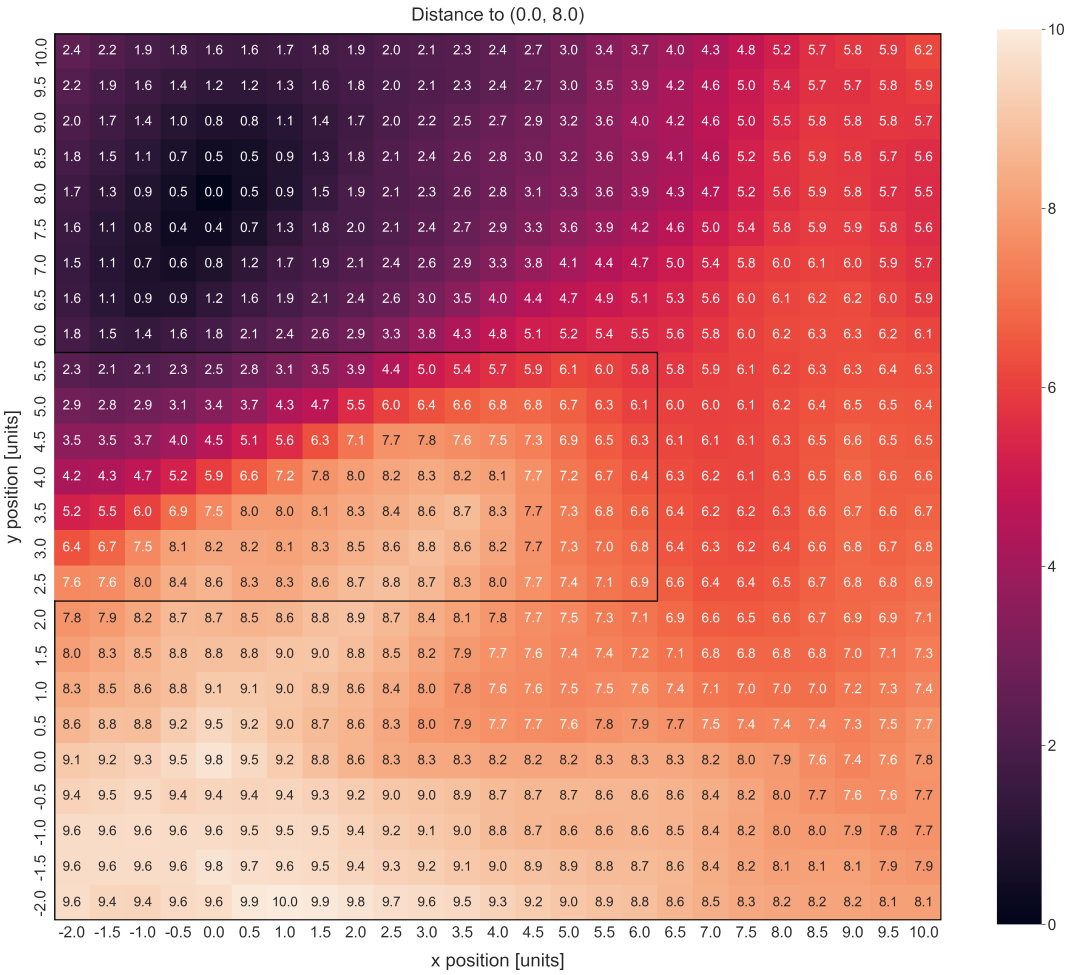


Figure 5.14: Distance heatmap for $(0, 8)$ for the default temporal distance setup.

Figure 5.14 shows the distance heatmap for the position $(0, 8)$ for this default setup. When comparing the results to the distance heatmap from the best SimCLR setup in Figure 5.13, it can be seen that the learned distances strongly curve around the corners of the maze and resemble the shape of

the maze a lot more accurately. Unfortunately, there is still a leakage of small distances on the left boarder of the maze, making the bottom left closer to the $(0, 8)$ position than it should be. Nevertheless, the resulting local minimum is far smaller, and if the agent explores enough to the right, it can quickly find a good path of continuously decreasing distances toward the goal, which would be a better reward signal during learning.

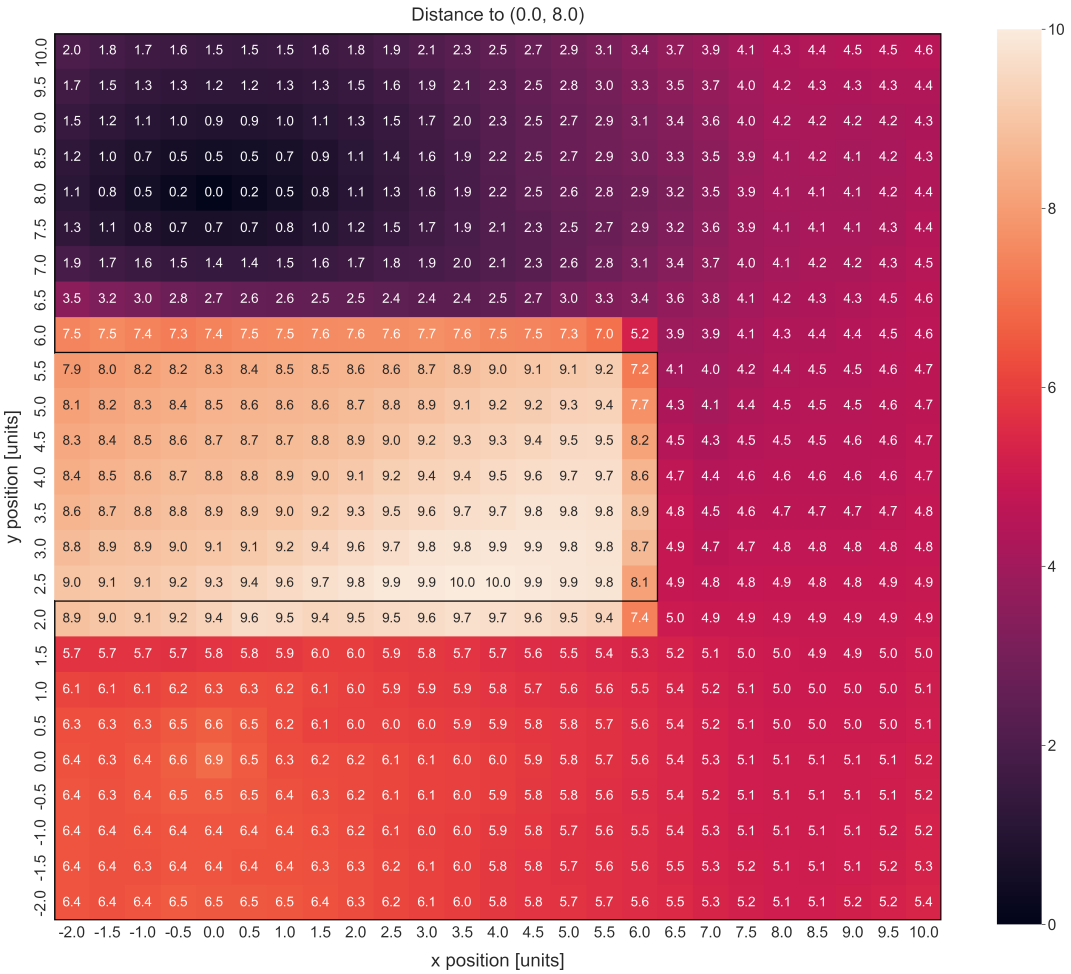


Figure 5.15: Distance heatmap for $(0, 8)$ when adding 10 cheated states with x,y positions sampled from the blocked middle part of the maze with a target distance of 1000.

As already the experiments with SimCLR showed, the leakage of distances through the blocked middle part in the maze causes problems for a NN to accurately learn the underlying u-shape. Transitions in this part of the maze are completely missing in the dataset, but the tested methods are unable to leverage the fact that missing transitions for a part of the state space indicate the impossibility of reaching that area. States without transitions

near them should therefore be far away from the rest of the state space that was actually visited and is part of the dataset. Sampling states that are explicitly not part of the dataset and using them as negative pairs or in the temporal distance setup with the maximum distance of 1000 steps might be a way to enforce this property.

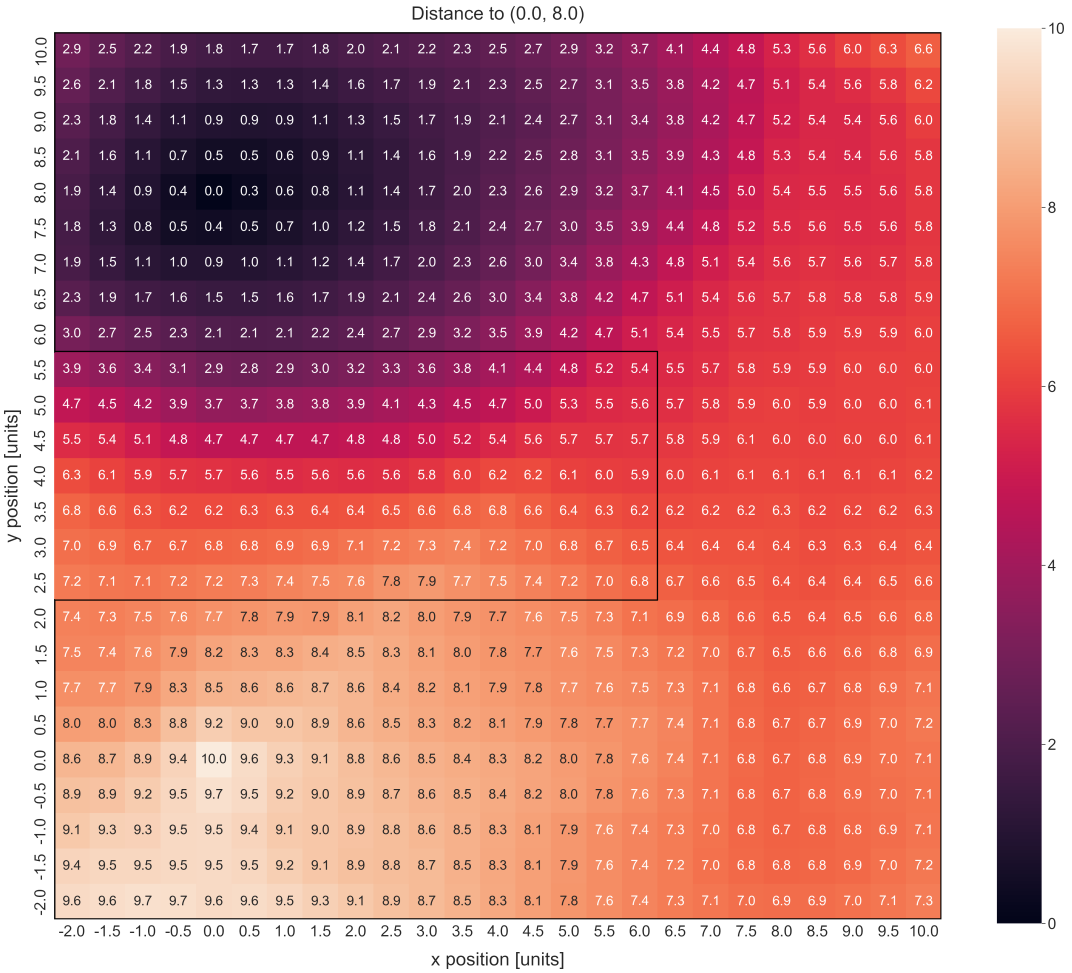


Figure 5.16: Distance heatmap for $(0, 8)$ when adding 50 randomly sampled states based on the complete visited state space with a target distance of 1000.

To start testing the mentioned hypothesis in an optimal but somewhat cheated way, for every batch, 10 x,y positions are sampled uniformly from the blocked middle part of the maze and every other dimension is filled with random configurations from the dataset. These cheated states are added to the current batch and every proper state gets a target distance of 1000 timesteps to the cheated ones. Learning with this cheated setup, the resulting distance heatmap for the position $(0, 8)$ is shown in Figure 5.15.

The leakage through the middle part can now barely be seen anymore and the distances curve perfectly around the maze. It should be kept in mind that this way of generating negative/unreachable samples is only possible because the x and y dimensions can be easily observed and reasoned about by a human. All other 27 dimensions can have the same problem resulting from unreachable parts of their state space that are not in the dataset. Figure 5.15 highlights what is possible if sampling from the distribution of states that are not part of the dataset is possible and the remaining experiments in this section try to find a way to do this in a more general approach.

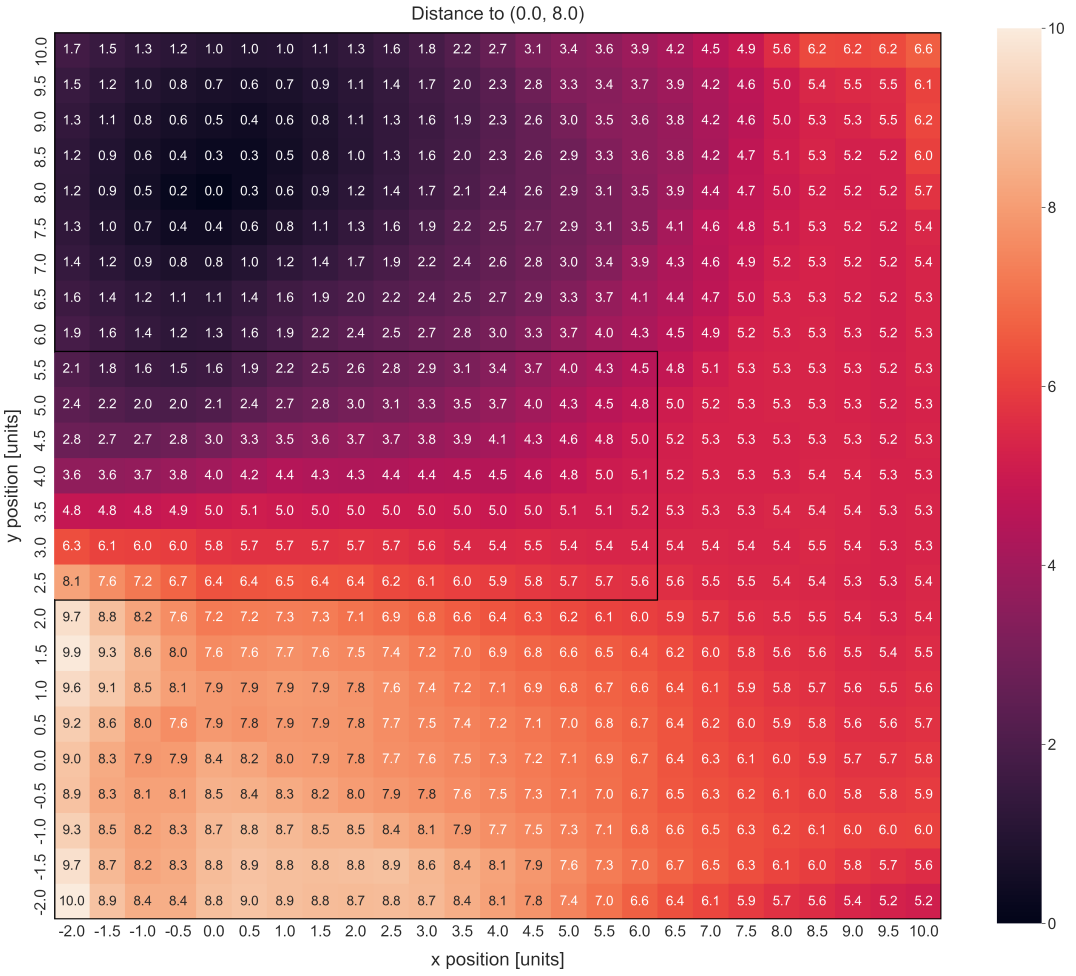


Figure 5.17: Distance heatmap for $(0, 8)$ when adding 50 randomly sampled states from the dataset with a target distance of 1000.

Instead of selectively sampling states, another approach can be to determine the minimum and maximum values for each dimension and then

sample uniformly from the space between those values. An additional 10 percent of the min and max values is added to them to be able to also sample states at the borders of the visited state space. The target distances for the sampled negative states are set to 1000 again. The correct distances for states that are sampled this way but are also part of the dataset can still be learned when there are enough samples of them in the dataset to counteract the 1000 distance. On the other hand, the unreachable states should keep their distance of 1000 as there are no samples in the dataset to counteract it. Figure 5.16 shows the resulting distance heatmap for the position $(0, 8)$ when adding 50 randomly sampled negative states based on the complete visited state space to the batches. The random samples smooth out the distance heatmap but do not stop the leakage through the middle of the maze. There is still a big local minimum around the bottom of the blocked middle part, which looks even slightly worse as the distances are lower there compared to Figure 5.14.

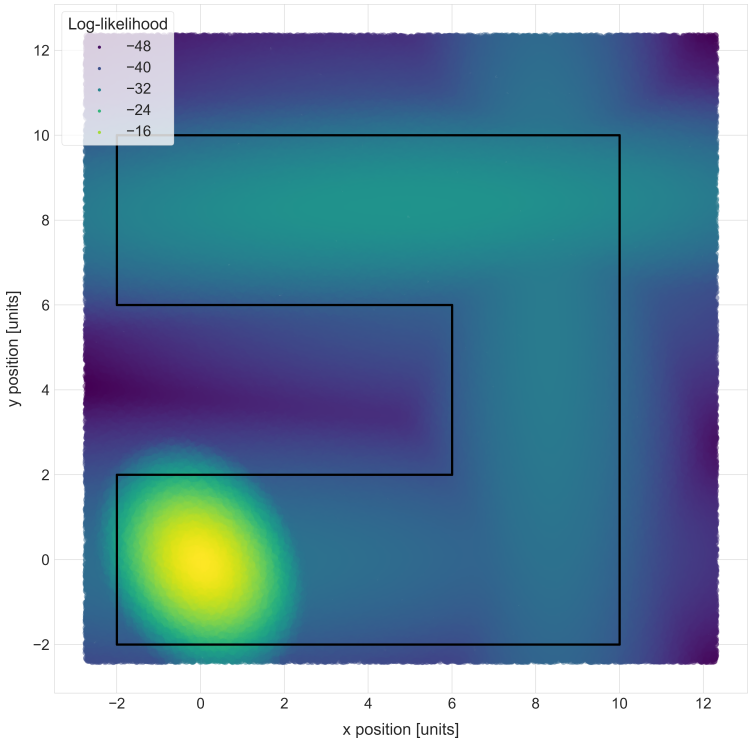


Figure 5.18: Heatmap of the log-likelihood of the best Gaussian Mixture Model (GMM) model for uniformly random sampled states, where only the x and y dimensions are varied.

Rather than sampling based on the complete visited state space, Mendonca

et al., 2021 proposes to sample the negative states from the actual dataset and label them with the maximum distance as well. Figure 5.17 presents the resulting distance heatmap for the position $(0, 8)$ when adding 50 randomly sampled negative states from the collected dataset to the batches. The leakage is still not stopped, but the local distance minimum is much smaller. There is a distance valley of around 7.2, which is followed by only a slightly higher plateau of around 7.3 to the right before going down again and revealing a path of continuously decreasing distances toward $(0, 8)$. Unfortunately, another local minimum arose in the bottom right corner which can be misleading for the agent.

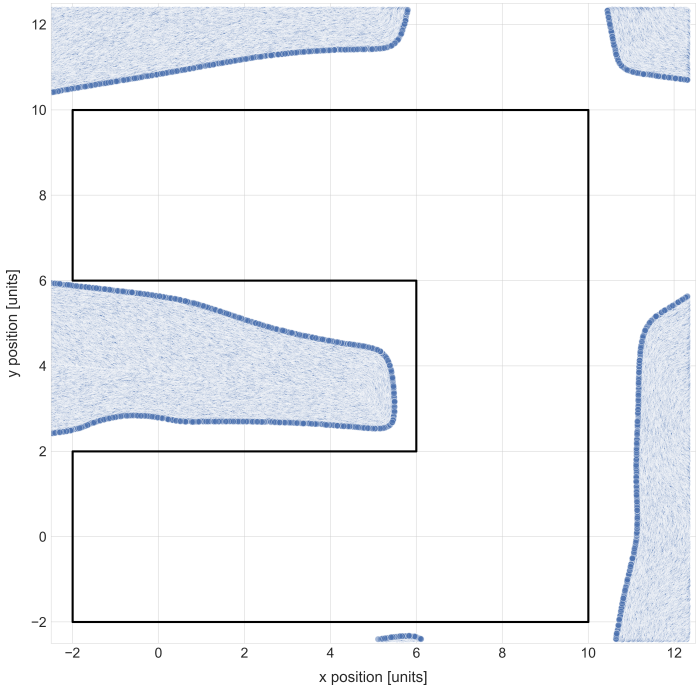


Figure 5.19: 20% of the states with the lowest log-likelihood of the GMM for uniformly random sampled states, where the x, y dimensions are varied.

To generalize the idea of explicitly sampling states that are not part of the dataset, like with the cheated setup, a model that can capture the distribution of the dataset can be trained and used to score uniformly random sampled states based on how likely they are to be part of the dataset. For this setup, a GMM (Bishop & Nasrabadi, 2006) is trained to approximate the probability distribution of all 29 dimensions with the samples from the dataset. The GMM implementation used is the one from the Python package scikit-learn (Pedregosa et al., 2011). To get the best parameters

for the model, a grid search is performed over the number of components (100, 300, 500) and the covariance type (full, spherical, diag, tied). The maximum number of iterations is fixed to 200 and 1M samples from the dataset are used for training. After training, the runs are evaluated and ranked based on the Bayesian Information Criterion (BIC). The best model uses 500 components, the full covariance type and takes around 54 hours to train. The long training time is the main reason why not more samples from the dataset are used for training. For future experiments, it would be possible to stick with the diag covariance type as the performance is slightly worse but only takes around 17 hours to train.

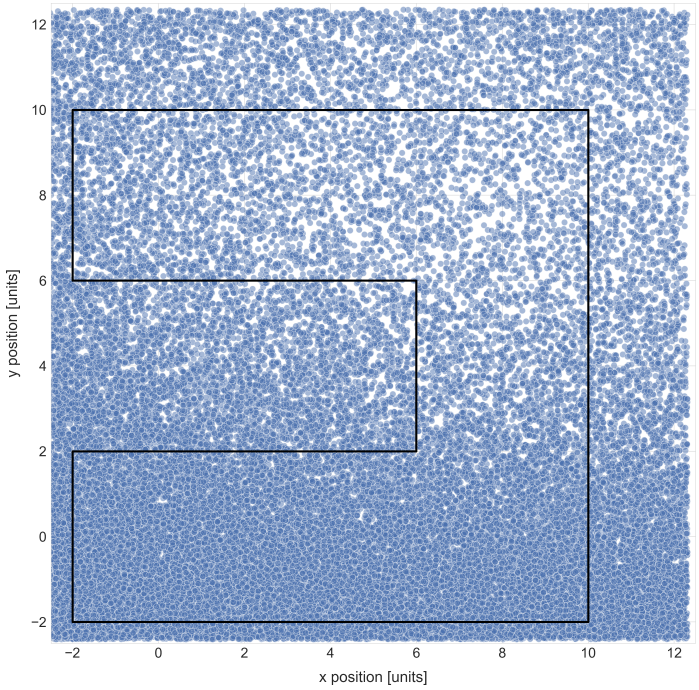


Figure 5.20: 20% of the states with the lowest log-likelihood of the GMM for uniformly random sampled states, where all dimensions are varied.

Figure 5.18 illustrates uniformly random sampled states from the range of min and max values of the visited state space and colored by the log-likelihood of the best GMM model. The more yellow the color is, the more likely it is that the state is part of the dataset. To sample states that are unlikely to be in the dataset, only the 20% of the states with the lowest log-likelihood are kept, which results in Figure 5.19. Both figures show that the GMM captures the distribution of the dataset quite well and the produced states with low log-likelihood are around the borders of the maze and most

importantly in the blocked middle part as well. These plots are produced by only varying the x and y positions of the randomly sampled states and keeping the other dimensions fixed. Figure 5.20 presents the same plot but uniformly random sampling every dimension and again keeping the 20% of the states with the lowest log-likelihood. Unfortunately, now all previously seen structure is gone and only a slight focus on lower y values can be seen.

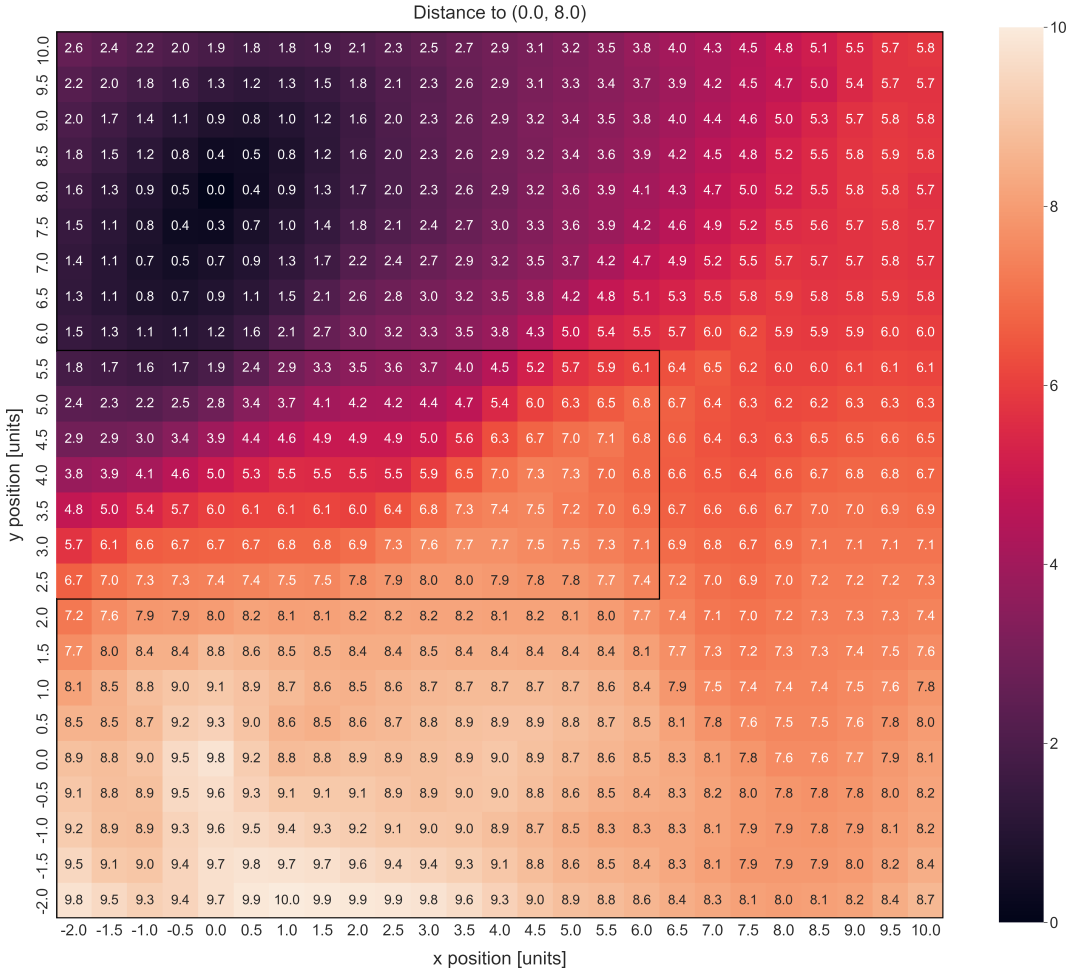


Figure 5.21: Distance heatmap for $(0, 8)$ when adding 50 states sampled from the 20% of states with the lowest log-likelihood scored by the GMM with a target distance of 1000.

Figure 5.21 shows the resulting distance heatmap for the position $(0, 8)$ when adding 50 states sampled from the 20% of states with the lowest log-likelihood to the batches. As already highlighted Figure 5.20, the sampled negative states are not very useful anymore and the final setup does not perform any better compared to the added random states in Figure 5.16.

The GMM shows potential when only considering the x and y dimensions, so making this work for all dimensions might still be possible and an interesting avenue for future work.

5.2.3 Information Retention

The final quality measure of the learned representations, that a GC-π algorithm actually cares about, is the performance of the goal-conditioned policy trained with them. Doing this downstream evaluation for every trained representation is too time consuming. Therefore, another way to assess the quality besides visually inspecting the distance heatmap is to look at the information contained in the representations. For a final comparison of the SimCLR and temperoral distances methods, a supervised learning experiment is conducted, where the goal is to predict the original state from its representation. The performance on this task indicates how much information about the original states is contained in their respective representations. The loss is defined as the MSE between the original state s and the output of a 3x256 NN $f_\psi(z_s)$, which tries to convert the representation z_s back to the original state:

$$\mathcal{L} = \|s - f_\psi(z_s)\|^2 \tag{5.1}$$

After training for 200 epochs on the complete dataset, the resulting training loss curves can be seen in Figure 5.22. The representations learned with the temporal distance method perform in general better than those learned with the SimCLR method. To evaluate single dimensions, the absolute difference between the original and predicted value of each dimension is also tracked during training. Surprisingly, the SimCLR representations perform significantly better on the x and y dimensions, which is highlighted in Figure 5.23. The temporal distance method shows better results on the other dimensions, but those might be not as crucial for guiding the goal-conditioned policy in its learning phase.

To determine which representation is better overall is hard to say with only looking at the distance heatmaps and the information retention experiment. In the end, the performance of the goal-conditioned policy is what matters the most and this will be evaluated extensively in the next section.

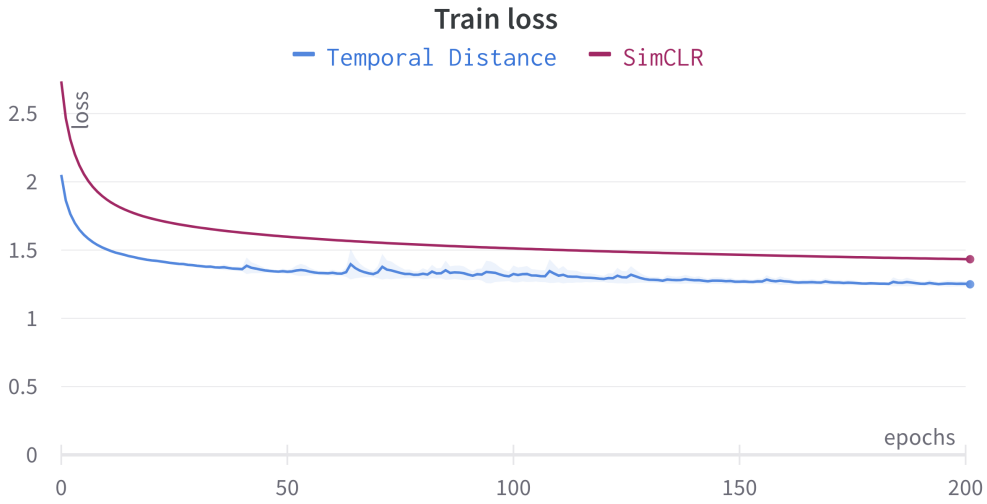


Figure 5.22: Training loss curves for the NNs trained with the SimCLR and temporal distance representations in the information retention task.

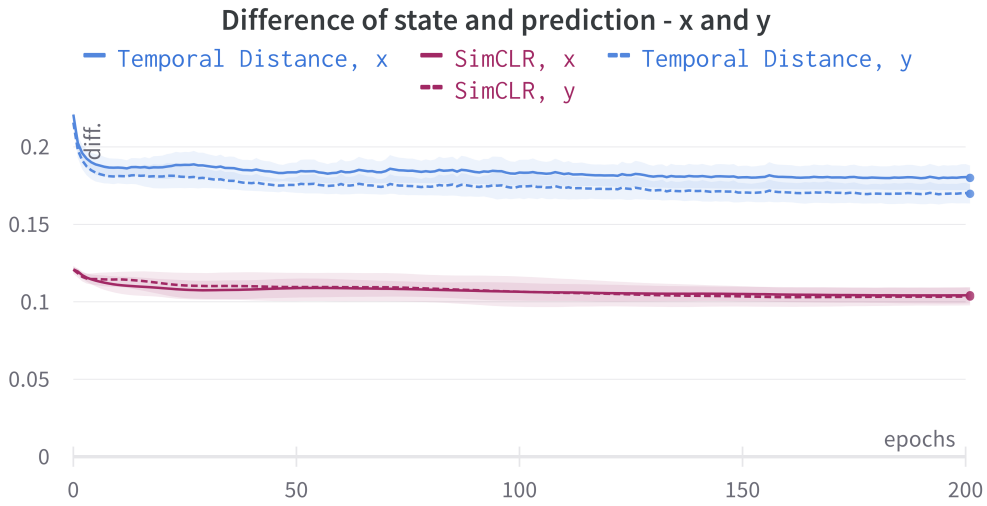


Figure 5.23: Absolute differences between the original and the predicted values of the x and y dimensions for the NNs trained with the SimCLR and temporal distance representations in the information retention task.

5.3 Goal-Conditioned Policy Learning

Creating a goal-conditioned policy is the last step in the GC-π algorithm pipeline. The training process can be done with either offline or online RL methods. In the offline case, the policy is trained with the dataset collected from the prior intrinsic exploration phase. This means that no additional data points are collected for policy training. During online learning, the agent can collect its own data by interacting with the environment. In both cases, the policy is trained with goals sampled from the collected dataset. Additionally, the previously offline learned representation space is used to calculate the reward as defined in equation 2.11. This means that the goal-conditioned policy can only ever be as good as the collected dataset and the representations it is trained with. Using offline RL is the more compelling and elegant option as the complete GC-π agent does only have to interact with the environment once for pure exploration purposes, but it is also notoriously hard to get right and train successfully (Kumar et al., 2020). Therefore, the online learning case is evaluated more extensively and further improvements, e.g., using subgoals to guide the policy, are only investigated for the online case as well.

5.3.1 Offline Learning

In general, offline RL is a lot trickier to train than its online counterpart because the agent cannot safely reinforce its own actions anymore. Sampling actions from the policy is necessary for most types of DRL objective functions such as equation 2.5 for SAC's version of the Q-learning objective or for policy gradient methods (Schulman et al., 2017). This means that the policy can generate actions or rather state-action pairs that are not covered by the dataset. Sampling such out-of-distribution actions can lead to catastrophic overestimation of their value (Kumar et al., 2020). The agent does not get any reality checks by actually trying the actions out in the environment, as would be the case in online RL. This problem led to the development of many offline-specific algorithms that try to reduce the im-

pact of out-of-distribution actions, two of which are tested in this section. For all the following experiments, the "cheated" temporal distance representation is used (see Figure 5.15) as it is visually the best performing one and its capabilities were checked in preliminary online RL experiments. Besides the distance in representation space, the distance in x,y space is also tested. During the training process, the policy is conditioned only on the evaluation goal $(8, 0)$. This training setup was chosen to initially make the learning problem as easy as possible. The difficulty of the task can always be increased later.

Implicit Q-Learning (IQL) (Kostrikov et al., 2021) circumvents the problem of overestimating out-of-distribution actions by simply never using them. It trains a state-value function $V(s)$ that regresses to an upper expected value:

$$\mathcal{L}_V = \mathcal{L}_2^\tau(Q_{\bar{\phi}}(s,a) - V_\psi(s)) \tag{5.2}$$

$$\mathcal{L}_2^\tau(u) = |\tau - \mathbb{1}(u < 0)|u^2 \tag{5.3}$$

where τ defines the targeted expectile. The action-value function $Q(s,a)$ can then be trained by using $V(s')$ as the bootstrap target instead of taking $Q(s', \tilde{a}')$ for a sampled action of the policy:

$$\mathcal{L}_Q = (r + \gamma V_\psi(s') - Q_\phi(s,a))^2 \tag{5.4}$$

After training the two value functions, the policy can be extracted with simple advantage-weighted regression (Peng et al., 2019):

$$\mathcal{L}_\pi = \exp\Big(\beta(Q_{\bar{\phi}}(s,a) - V_\psi(s))\Big) \log \pi_\theta(a|s) \tag{5.5}$$

where β is a temperature parameter. To closely follow the recommendations of the original paper and official implementation, the targeted expectile is set to $\tau = 0.8$ and for the temperature a small set of values were initially tested $\beta = [10.0, 1.0, 0.1]$. The results of the training process are shown in Figure 5.24. Neither the representation distance nor the x,y distance can enable the learning of a good policy. More extensive hyperparameter tuning might be needed to make the algorithm work, but the obtained results were not promising enough to further investigate IQL.

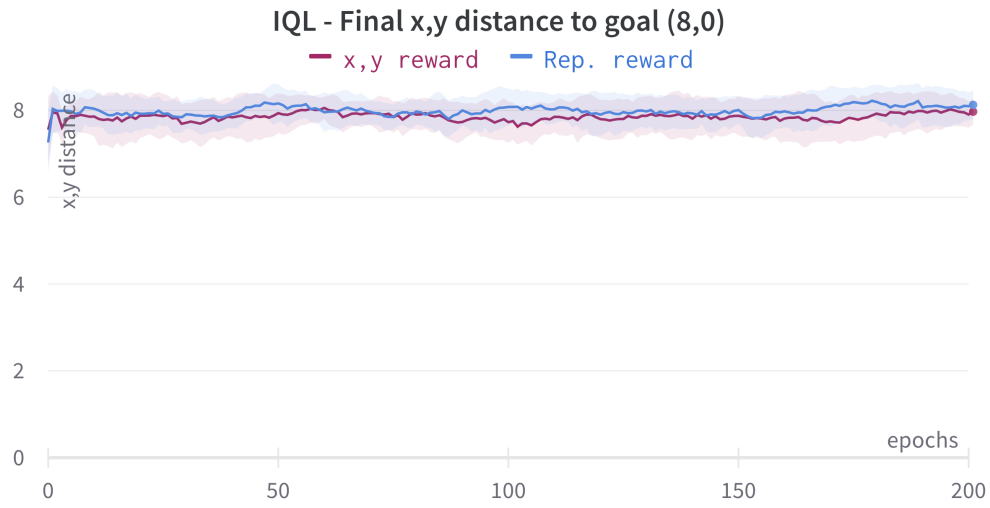


Figure 5.24: IQL conditioned on the goal $(8, 0)$ and the reward calculated in representation space or x,y space.

Conservative Q-Learning (CQL) (Kumar et al., 2020) is another offline RL algorithm that learns a lower-bound Q-function around the actions in the dataset to reduce the impact of out-of-distribution actions. In practice, CQL implements a simple Q-value regularizer on top of an SAC-style algorithm, which weights down the Q-values of actions sampled from the policy compared to actions from the dataset. For the experiments, the exact hyperparameters are used as in the original paper, but the number of sampled actions is reduced from 10 to 4 to make the training process faster (from 20 to 10 minutes per epoch). More sampled actions did not improve the results in preliminary experiments running for 20 epochs. The training curve for CQL is illustrated in Figure 5.25. Again, neither the representation distance nor the x,y distance show good performance. Compared to IQL, the agent at least learns to move slightly to the right and reaches a final distance of 6 units, but there are no signs of it learning to get out of this poor local minimum.

Both offline RL algorithms showed disappointing results. For the last attempt, only pure TQC is used without any offline specific modifications. This corresponds to the previously used CQL setup without the Q-value regularizer. The results are presented in Figure 5.26. The agent learns substantially better and gets in the 2-unit range of the goal with the reward in

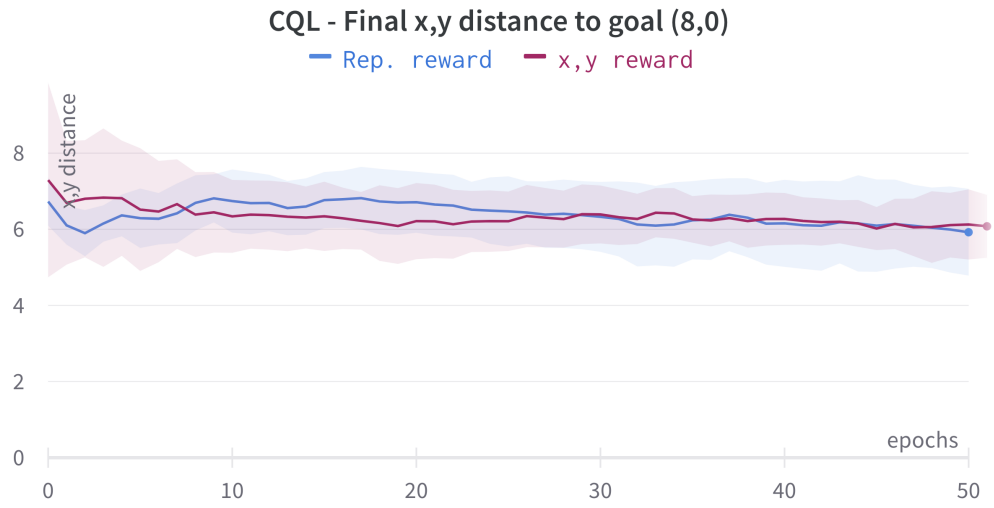


Figure 5.25: CQL conditioned on the goal $(8, 0)$ and the reward calculated in representation space or x,y space.

x,y space. Calculating the reward in the representation space still leads to an improvement over the other offline methods, but its performance shows high variance and is not as good as the x,y space reward. Even though the results are better in general, they are not good enough to be considered a success compared to the baseline online RL results in chapter 4.

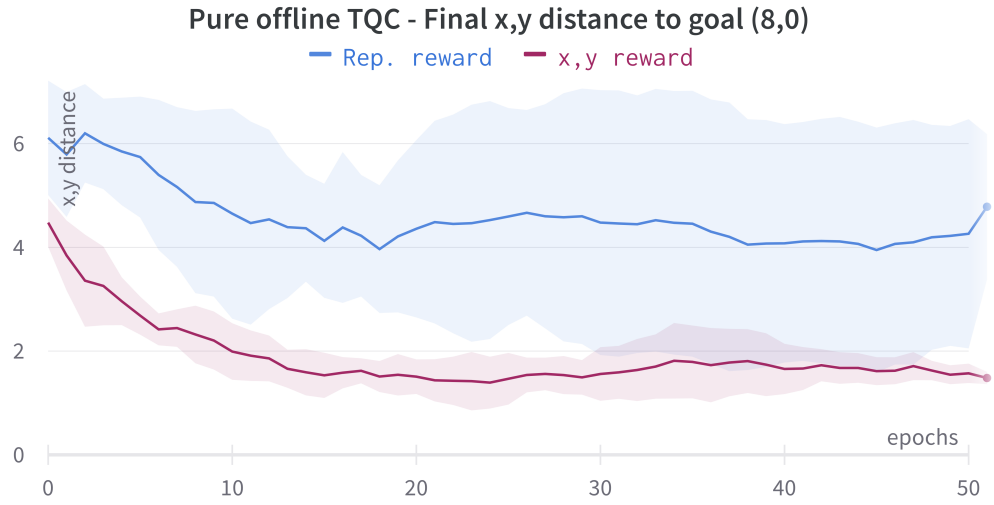


Figure 5.26: Pure offline TQC conditioned on the goal $(8, 0)$ and the reward calculated in representation space or x,y space.

Unfortunately, both IQL and CQL show poor results already in the easy

learning setup, even to the point that a standard TQC agent performs better than any offline specific adaptation. Different algorithms, further hyperparameter tuning, better representations or a more balanced dataset might help to alleviate the underlying problems and should be investigated more thoroughly in the future. Using offline RL algorithms is still a promising idea due to its ability to learn from a completely different behavior policy. This means that every interaction with the environment can fully focus on exploration or even be from a completely different type of agent. Nevertheless, to make progress in the AntMaze task, the focus for the rest of the thesis lies on online RL algorithms.

5.3.2 Online Learning

In the online learning setup, the agent is trained with the help of the previously offline learned representation space, but can interact with the environment to collect its own data and reinforce its policy in the typical trial-and-error RL fashion. This section starts with experiments sampling the evaluation goals for the training process, like in chapter 4. Later on, the states collected in the dataset are used as the goals during training. To learn only from the collected states is a tougher problem, as the agent has to generalize much more and the dataset is not balanced, but it allows the algorithm to be fully free of any knowledge about possible downstream goals, which is one of the main objectives of this thesis (see chapter 1.2).

The first experiment is conducted with the "cheated" temporal distance representation. Using this representation is not the end goal, but a good starting point. The results can be seen in Figure 5.27. The agent is able to consistently reach all seven evaluation goals after only around 5M steps and outcompete the baseline performance by a large margin (compare to Figure 4.5). It should be noted that this is the first time in this thesis that the agent could reach the last evaluation goal $(0, 8)$. This experiment demonstrates that the desired performance in the proposed AntMaze setup is possible, but the use of the "cheated" representation and the incorporation of the evaluation goals during the training process remain undesirable.

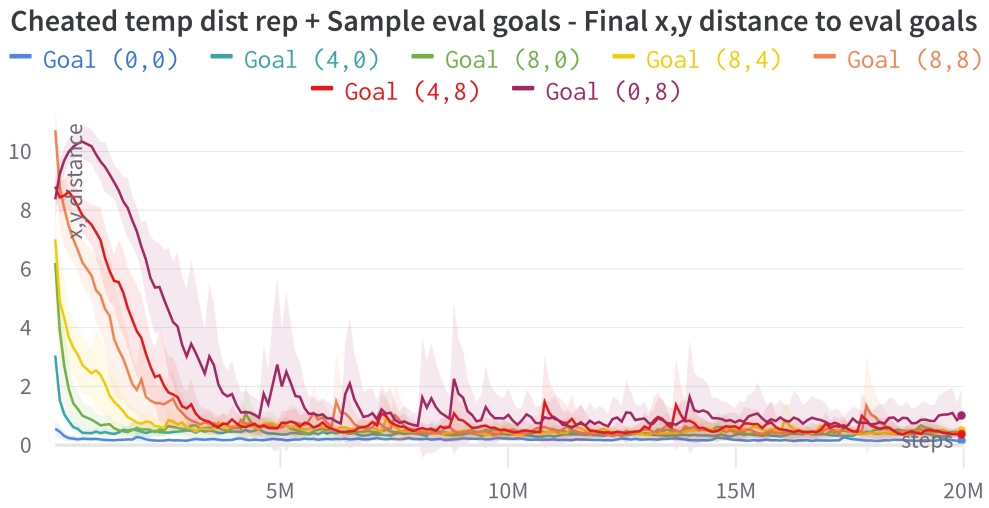


Figure 5.27: Performance on the evaluation goals when using the "cheated" temporal distance representation for the reward calculation and sampling the evaluation goals for training.

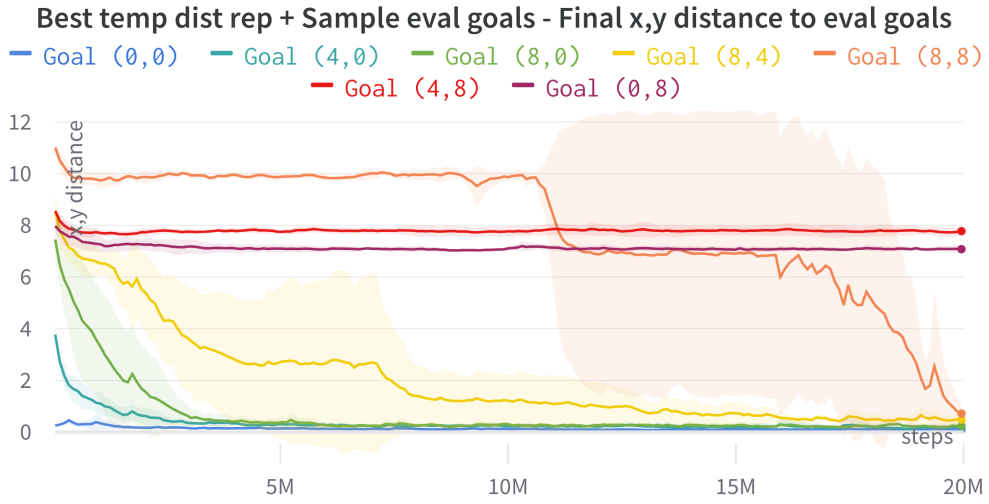


Figure 5.28: Performance on the evaluation goals when using the basic temporal distance representation for the reward calculation and sampling the evaluation goals for training.

To remove the need for the "cheated" representation, the next experiment is conducted with a standard temporal distance representation. Both the basic setup (Figure 5.14) and the addition of negative samples from the dataset (Figure 5.17) were evaluated in preliminary experiments. The basic setup performed better and its resulting learning curve is depicted in

Figure 5.28. The agent can only barely reach the goal $(8, 8)$ after 20M steps and makes no progress in going deeper into the maze. Using a standard temporal distance representation is not enough to solve the task and performs worse than the baseline with a SimCLR representation (compare to Figure 4.5). On the upside, the significant performance difference between the normal and the "cheated" representation space highlights the potential of future representation learning methods, which can generate and use out-of-distribution data as negative samples, like the "cheated" representation did in a first attempt.

The best previously trained SimCLR representation (Figure 5.13) is tested in the next experiment and its results are illustrated in Figure 5.29. The agent is able to reach all seven evaluation goals and performs substantially better than the normal temporal distance representation. During training, the ability to reach the last two evaluation goals emerges roughly 2M steps later than in Figure 5.27, but in general the performance is comparable to the "cheated" temporal distance representation. This means that standard representation learning methods such as SimCLR can be enough to solve the task, at least as long as the evaluation goals are used during training.

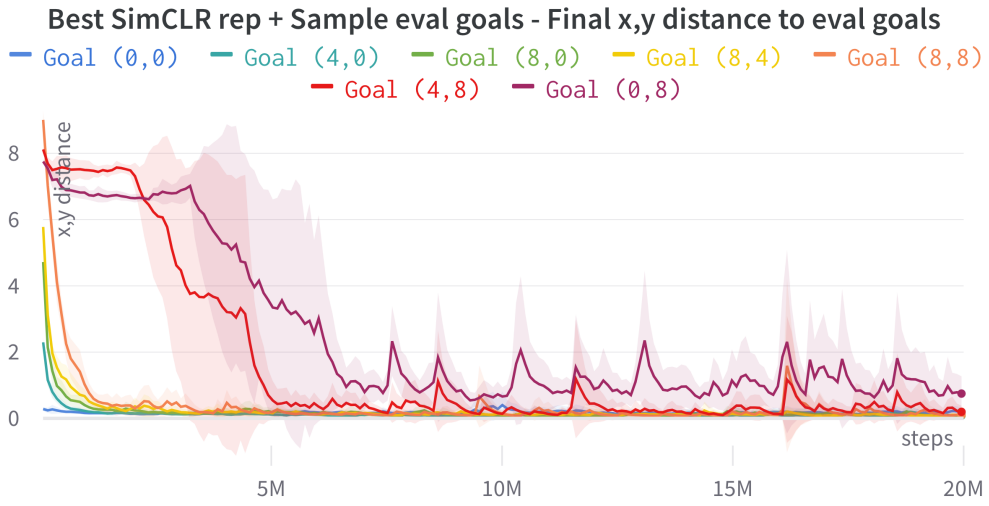


Figure 5.29: Performance on the evaluation goals when using the SimCLR representation for the reward calculation and sampling the evaluation goals for training.

The collected dataset provides a rich set of states, which can be used dur-

ing training to remove the need for knowledge about the evaluation goals. It can be expected that the agent will perform worse or at least needs more time to show the same performance on the evaluation goals when not directly training on them. The states in the dataset are also not uniformly distributed over the maze and the deeper sections of the maze are less represented (see Figure 5.6). To initially make the task simpler, the "cheated" temporal distance representation is used, as it showed the best performance in the previous experiment. Figure 5.30 displays the agent's performance when using the states sampled from the dataset as the training goals. The learning shows high variance and in the end the agent is not able to consistently reach even the more easier goals such as $(8,0)$. A lack of good generalization in the goal space might be the main reason for the poor performance as the agent cannot simply memorize seven paths to the seven evaluation goals anymore, but rather has to infer the underlying structure of the state/goal space. Another reason can be the other 27 dimensions besides the global x,y position in the goal state. The states/goals sampled from the dataset have different values in all dimensions, whereas the evaluation goals only differ in the x,y position. These factors might be too much for the agent to learn a good policy and even the "cheated" temporal distance representation is not good enough to compensate for them.

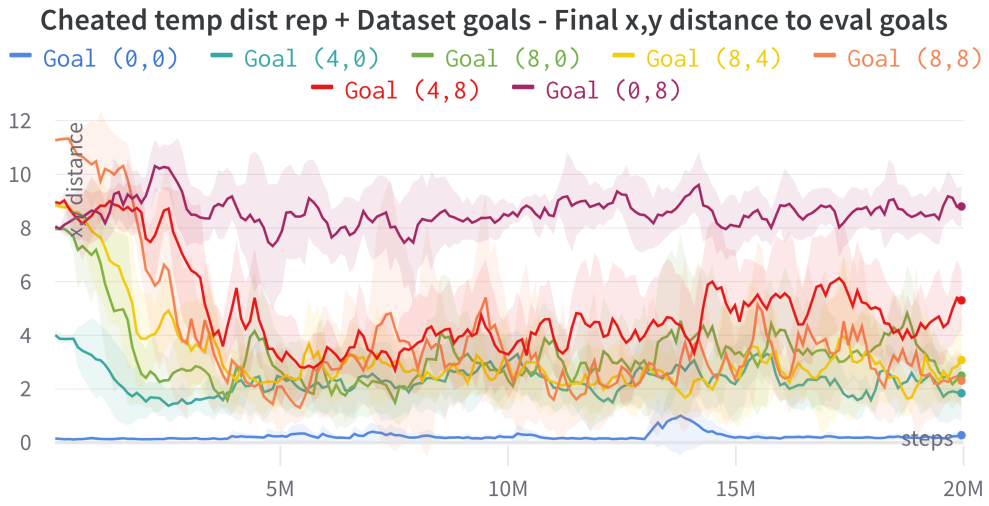


Figure 5.30: Performance on the evaluation goals when using the "cheated" temporal distance representation for the reward calculation and sampling the states in the dataset as training goals.

5.3.3 Subgoal Trajectories

HRL algorithms leverage a higher level policy that generates subgoals to guide a lower level policy that is responsible for the actual movements of the agent (Hafner et al., 2022). This setup can make it easier for the agent to reach far away goals, as it only needs to navigate from subgoal to subgoal, but training such a high level policy adds another layer of complexity and more wall clock time to the training process. To evaluate the potential of subgoals for a GC-π algorithm, the trajectories in the collected dataset can be used as the source of the subgoals, rather than letting them be produced by a higher level policy, which is typically parameterized with a NN. To heuristically find the best subgoals in the dataset for a given goal, the following procedure is used:

1. Calculate the representations for all states in the dataset and the goal state.
2. Loop over all states and determine the closest one to the goal by calculating the euclidean distance between the representations.
3. Take the trajectory containing the closest state and cut it off at the closest state.
4. Choose every n-th state from the trajectory as a potential subgoal, where n is a hyperparameter.
5. Loop over the potential subgoals in reverse order with tuples of three consecutive subgoals and remove the second subgoal in the tuple if the distance between the first and the third subgoal is smaller than the distance between the second and the third subgoal.

When training on goals sampled from the dataset, the subgoals are calculated on the fly, whereas the subgoals for the evaluation goals are calculated beforehand. The pruning step (5.) is used to remove subgoals that do not bring the agent closer to the next subgoal, which is helpful as the trajectories from the prior intrinsic exploration can be chaotic and full of loops or backtracking. The hyperparameter n is set to 50 in the following exper-

iments, which means that every 50th state in a trajectory is considered a potential subgoal. For the representation space, the SimCLR representation is used, as it proved the best performance in the previous section (exluding the "cheated" representation). SimCLR showed poor distances for far away states but rather good ones for neighboring states, as illustrated in its distance heatmap (Figure 5.13). Furthermore, SimCLR reached better performance on the information retention task for the x and y dimensions than the temporal distance representation. All of this points to accurate local distances, which is all that is needed in this setup as the agent only has to navigate from subgoal to subgoal.

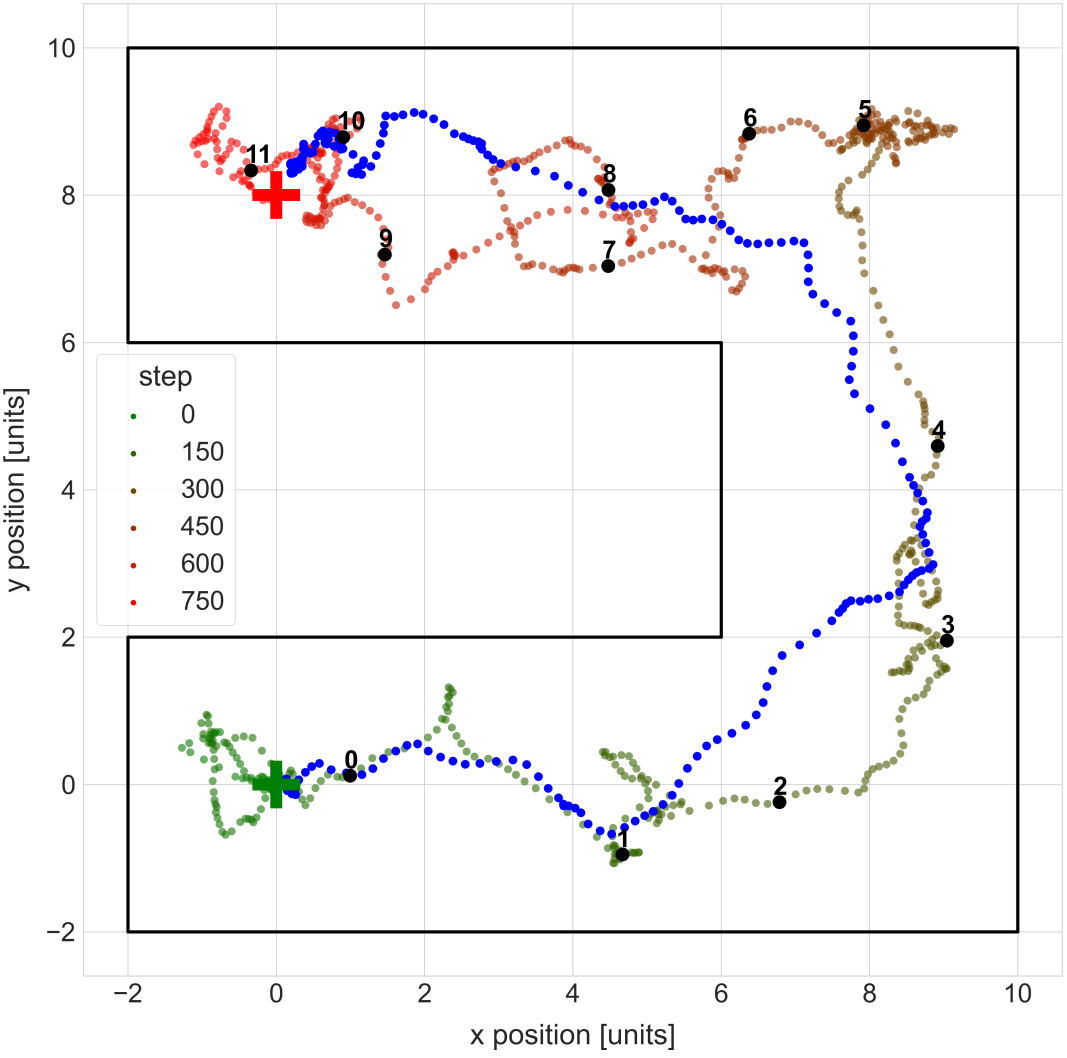


Figure 5.31: Illustration of the best trajectory in the dataset for the evaluation goal $(0, 8)$ in green and red, the calculated and pruned subgoals in black and an example trajectory from a trained agent in blue.

The best trajectory and the corresponding subgoals for the evaluation goal $(0, 8)$ are illustrated in Figure 5.31. The subgoals are already pruned and show a good path to the end goal with no backtracking. As an example for the pruning, if subgoal 7 would be a bit further to the left, the subgoal 8 would be removed as well. The figure also shows an example trajectory of a trained agent, which is guided by the subgoals and comes very close to the final goal at the end. As it can be seen, the agent does not need to precisely reach every subgoal, because during the training process, the subgoals are switched before the agent reaches them. The agent starts with the subgoal sg_1, i.e., the second subgoal in the list. A current subgoal is switched to the next one when the distance in the representation space between the agent's state s_t and the current subgoal sg_i is smaller than the distance between the previous subgoal sg_{i-1} and the current subgoal sg_i. This heuristic ensures that the agent does not get stuck trying to precisely reach the current subgoal and it also avoids the need for any distance cutoff hyperparameter.

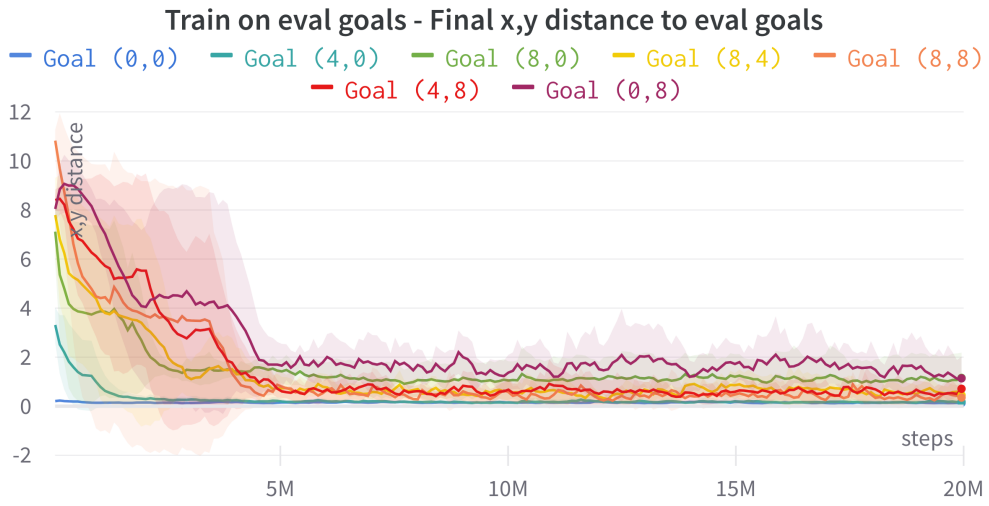


Figure 5.32: Performance on the evaluation goals with the subgoal setup when using the SimCLR representation for the reward calculation and sampling the evaluation goals for training.

For the first experiment, the agent is trained directly on the seven evaluation goals. Figure 5.32 displays the performance of the agent and can be compared to the setup without subgoals in Figure 5.29. The agent is able to reach the last two evaluation goals faster than previously and its

performance is even comparable to the "cheated" temporal distance representation in Figure 5.27. The spikes for the goal $(0,8)$ during the middle and end of training in Figure 5.29 are far more damped in comparison and the agent now reaches the goal consistently.

In the setup without subgoals, the agent struggled to reach even the simplest evaluation goals when trained on states sampled from the dataset as training goals (Figure 5.30). Figure 5.33 shows the performance of the subgoal setup in this scenario while also leveraging the SimCLR representation instead of the "cheated" temporal distance representation. The final performance after training for 20M steps is not as clean and consistent as when training on the evaluation goals directly and the standard deviation for the goal $(0,8)$ is still quite high, but the agent is now able to reach or come close to all evaluation goals without directly training on them for the first time. The main reasons for the worse performance compared to directly training on the evaluation goals, as explained in the previous section, still apply, but the addition of subgoals helps the agent significantly navigating and generalizing in the narrower subgoal space. Longer training times and additional hyperparameter tuning might improve the performance even further.

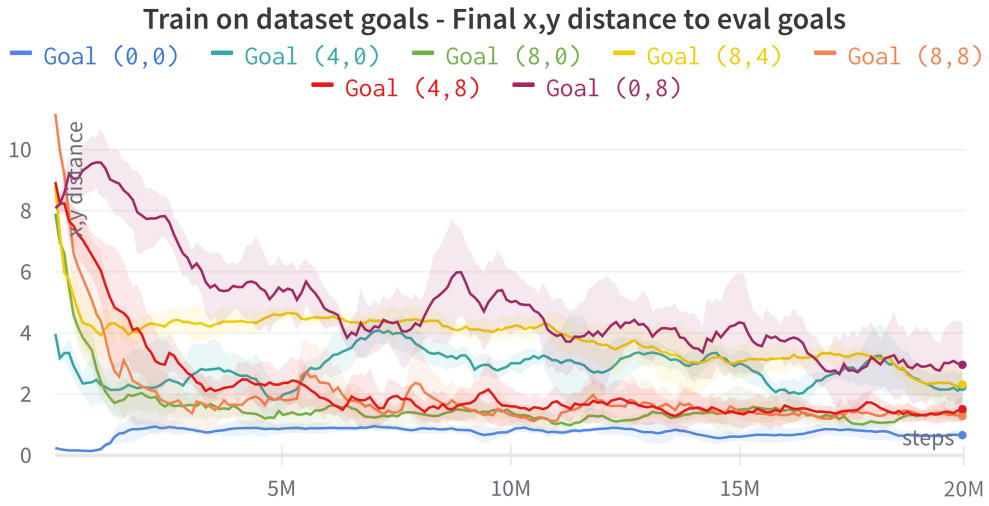


Figure 5.33: Performance on the evaluation goals with the subgoal setup when using the SimCLR representation for the reward calculation and sampling the states in the dataset as training goals.

The pruning step in the subgoal calculation helps to disentangle the chaotic trajectories in the dataset, but it does not guarantee the best possible subgoals for the goal-conditioned agent. It seems natural to let the agent improve the subgoal generation during training on its own, so it can optimize for reaching speed of the end goals and consistency. To do this, the previously defined subgoal procedure is distilled into a high level policy parameterized by a NN. The high level policy $\pi_\vartheta(sg_t|s_t, g)$ takes the current state and the end goal as input and produces the next subgoal as its output. Both the original state space and the representation space were tested for the inputs and the output, but the original state space performed better in preliminary experiments, i.e., the lower level policy was able to learn faster and reach better performance. To pre-train the high level policy, training and test datasets of 50M and 5M samples each are curated with (state, goal, subgoal) tuples from the previously collected dataset. The states and goals are randomly sampled from all the states in the dataset and have to be part of the same trajectory. The corresponding subgoal is then calculated using the subgoal procedure described above. A 3x256 NN with a 29-dimensional output head is used for the high level policy and trained with the following MSE loss:

$$\mathcal{L} = \|sg - \pi_\vartheta(s, g)\|^2 \tag{5.6}$$

As in all other experiments, the learning rate is set to 0.0003 and a batch size of 512 is used. Figure 5.34 shows the training and test loss curves for the pre-training of the high level policy. The losses decrease steadily and the area of the standard deviation can barely be seen over the 3 seeds, which indicates a stable learning process. More epochs of training were not possible due to time constraints but would most certainly improve the loss further.

Using the pre-trained high level policy during the online goal-conditioned training allows for generating subgoals on the fly. This removes the need of coming up with a heuristic to switch the subgoals during an episode as the high level policy can simply generate a new subgoal in every timestep, which makes for smoother subgoal transitions and learning signals, as long as the learned high level policy can generalize well. Figure 5.35 illustrates the potential of the high level policy. The agent performs more consistently

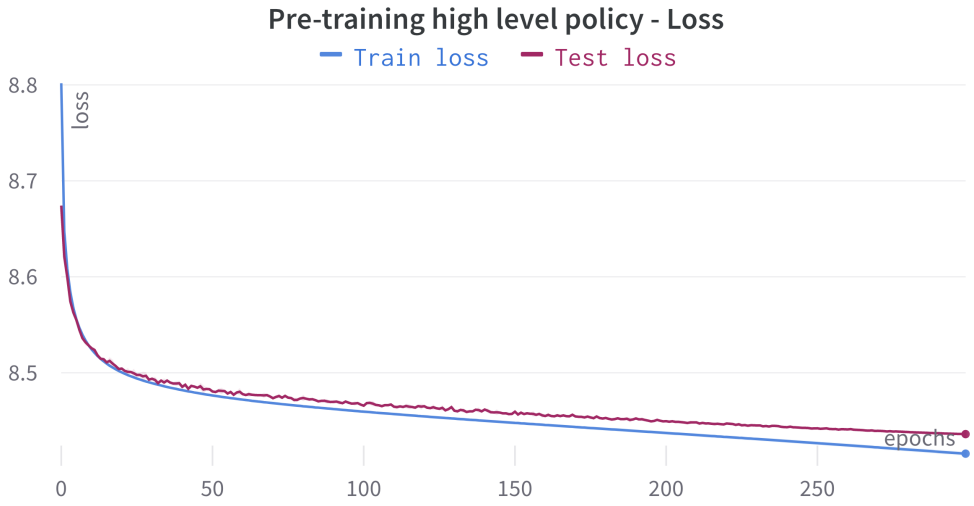


Figure 5.34: Train and test loss curves for the pre-training of the high level policy.

on some of the easier goals such as $(4,0)$ and $(8,4)$ compared to using the heuristic for the subgoal generation directly (see Figure 5.33). Unfortunately, the performance on the last subgoal $(0,8)$ is slightly worse than before. This is caused by the fact that the deeper parts of the maze are underrepresented in the dataset, which makes it harder for the pre-training process to generalize well in those areas, whereas the heuristic generation can rely on finding one single good trajectory in the dataset toward the goal.

Fine tuning the high level policy during the online training process requires many non-trivial design decisions. The first one is the used reward signal. The generated subgoals should assure that the agent reaches the end goal as fast and consistent as possible. Therefore, the reward might be higher the faster the agent reaches the end goal. This opens the question of how to define when a goal was reached, which might be answered with a distance threshold in the representation space. With this reward setup, the high level policy would only receive a non-zero reward signal when reaching the end goal. This can be too sparse in the early stages of training and it can then overwrite the carefully pre-trained parameters. Updating the policy based on a randomly initialized Q-function at the start of the training process can also lead to catastrophic overwriting of the parameters. An-

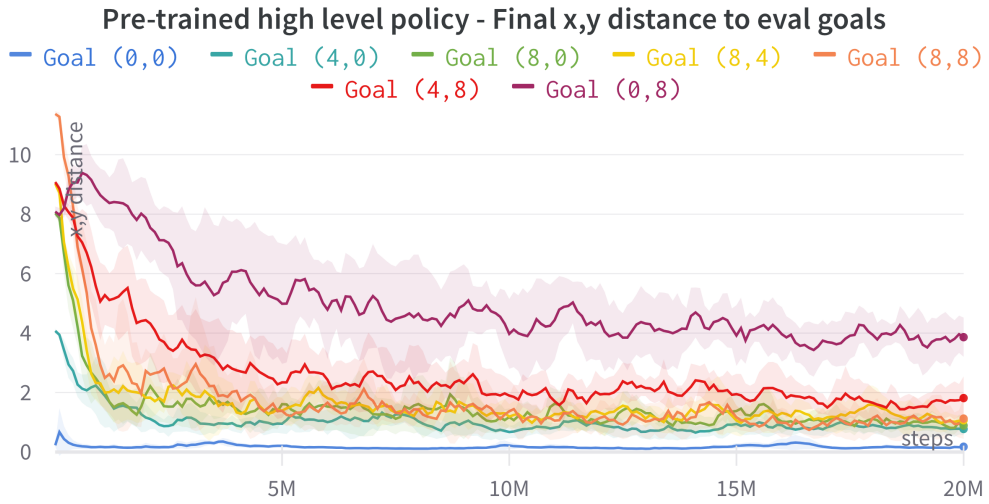


Figure 5.35: Performance on the evaluation goals with the subgoal setup when using the SimCLR representation for the reward calculation and sampling the states in the dataset as training goals. The subgoals are generated with the pre-trained high level policy.

other problem is the action space parameterization of the high level policy. Typically, continuous action space compatible DRL algorithms such as SAC or TQC parameterize the action space with a Gaussian distribution. The mean of the distribution can be the output head of the pre-trained policy, but how the standard deviation should be initialized is unclear. All those mentioned problems and a lack of experimentation time lead to only worse results than before and are not further investigated in this thesis They remain great avenues for future work.

6. Conclusion

To conclude this thesis, the first section of this chapter will summarize the results and main contributions described in the previous chapters. The second section will discuss the limitations of the developed GC-π algorithms and propose interesting directions for future work.

6.1 Summary

The aim of this thesis was to develop a novel algorithm that follows the objectives defined in chapter 1.2 with the specific goal-conditioned AntMaze problem setup described in chapter 3. The main contribution is the GC-π algorithm family, which was proposed in chapter 5. The following list summarizes the approaches of the GC-π algorithms on how the defined objectives were achieved:

- **Concept of goals**: The GC-π algorithms utilize the GCRL framework to define goals. Contrary to previous work, every dimension of the state space is used in the goal space, which allows for a completely flexible goal selection.

- **Discover new goals**: A dataset containing states, which act as possible goals, is collected through a prior intrinsic exploration phase. The goal-conditioned agent samples those goals during training and therefore learns to deliberately reach the parts of the state/goal space that were explored by the intrinsic agent previously.

- **Separately optimizable processes**: The exploration and goal learning objectives are separated into two distinct phases and executed one after another. This means that the objectives are optimized by a fully

intrinsic motivated agent and a goal-conditioned agent respectively and do not compete with each other, as is the case in previous work.

- **Unknown downstream goals**: When the intrinsic agent is able to explore the state space extensively, the goal-conditioned agent is free to learn only about the collected states/goals. It can generalize to unseen downstream goals as long as there are enough similar states in the dataset.

- **High dimensional state space**: The GC-π algorithms learn a representation of the state space that can handle the high dimensionality by reweighting and compressing (if necessary) the influence of the dimensions with a NN and therefore focus on the relevant ones and disentangle the underlying dynamics of the learning problem.

- **Physics-based environment and continuous action space**: All experiments were conducted with the AntMaze environment, which uses the MuJoCo physics engine and the simulated ant robot, which consists of an 8-dimensional continuous action space. Both pose no problems for the developed algorithms because they use TQC as the underlying DRL algorithm, which handles continuous action spaces and high-dimensional inputs easily through the use of NNs.

- **Reasonable computational budget**: A single complete 20M step training run takes between 17 and 19 hours with the hardware described in chapter 3.3. This learning speed is made possible with the newly developed DRL framework RL-X. All experiments that were conducted for this thesis used two PCs with the same hardware setup. The total number of tracked experiments is 371, with an average of 12.18 hours per experiment, which amounts to a total compute time of 4519 hours or 188 days for all the research done in this thesis.

6.2 Limitations & Future Work

The developed GC-π algorithms in this thesis are a promising approach for goal-conditioned learning problems by combining multiple areas of RL and ML research. This also means that progress in each of those areas can be used to improve the GC-π algorithms further and make them more general. Each of the three phases (prior intrinsic exploration, offline representation learning, goal-conditioned policy learning) can be individually improved and an improvement in one phase helps the subsequent phases as well.

The initially collected dataset is the basis of the learned representation space, goal sampling, potential offline RL and subgoal trajectories. The used RND agent was able to explore the AntMaze environment extensively but struggled to produce a balanced dataset, which is to be expected as the algorithm was not designed for this purpose. For future research, it might be interesting to bring the balancedness criterion directly into the intrinsic reward function or postprocess the dataset to make it more balanced. Besides that, general improvements in the field of IM can be used to improve the exploration capabilities of the intrinsic agent, which will be needed for even bigger and more complex environments.

The offline learned representation space lies at the heart of the GC-π algorithms because it forms the reward signal for the goal-conditioned policy learning. The representations learned with the temporal distance approach showed promising results as accurate distances for far away states were learned. Nevertheless, in the final usage as the distance metric for the reward signal, the temporal distance method fell short and the representations learned with SimCLR performed better. Investigating this discrepancy and also trying to apply newer non-contrastive JEMs that do not need negative samples (Bardes et al., 2022) are interesting directions for future work. Improvements to the representations might as well be the key to unlocking the potential of offline goal-conditioned policy learning, as current methods such as IQL and CQL were unable to learn good policies. In the offline learning setup, the agent would only need to interact with the environment during exploration to collect the dataset, or the data could even

come from completely different sources such as human demonstrations (Rajeswaran et al., 2017).

Interweaving the exploration and goal learning phases more often than only once per training run could be an interesting way to share the knowledge between the two agents and speed up the learning process (Pislar et al., 2021). This could be done by alternating between the two phases every n episodes or even every n steps, where the parameter n might be learned based on a measure of the current learning progress or based on the part of the state space the agent is currently in.

The AntMaze environment uses only a 29-dimensional state space, which is high dimensional but still relatively small compared to environments where the agent has thousands of sensors or even camera images as input. The applied representation learning methods have to be able to disentangle and compress such high dimensions or the resulting reward signal will be too noisy and full of local minima to be useful. Besides extremely high-dimensional state spaces, hidden state spaces produced by LSTMs or similar recurrent NNs also pose a challenge not only for the representation learning methods. The GCRL framework itself makes it hard to define downstream goals as a human operator when the state space is too large or its dimensions are simply not interpretable. A combination or translation of language representations and state space representations might be necessary to make goal-conditioned RL feasible for complex real-world applications, where humans define abstract goals in their natural language (Colas et al., 2022).

Bibliography

Abdolmaleki, A., Springenberg, J. T., Tassa, Y., Munos, R., Heess, N., & Riedmiller, M. (2018). Maximum a posteriori policy optimisation, In *International conference on learning representations*.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Pieter Abbeel, O., & Zaremba, W. (2017). Hindsight experience replay. *Advances in neural information processing systems*, *30*.

Aubret, A., Matignon, L., & Hassas, S. (2019). A survey on intrinsic motivation in reinforcement learning. *arXiv preprint arXiv:1908.06976*.

Aubret, A., Matignon, L., & Hassas, S. (2021). Distop: Discovering a topological representation to learn diverse and rewarding skills. *arXiv preprint arXiv:2106.03853*.

Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z. D., & Blundell, C. (2020b). Agent57: Outperforming the atari human benchmark, In *International conference on machine learning*. PMLR.

Badia, A. P., Sprechmann, P., Vitvitskyi, A., Guo, D., Piot, B., Kapturowski, S., Tieleman, O., Arjovsky, M., Pritzel, A., Bolt, A., Et al. (2020a). Never give up: Learning directed exploration strategies, In *International conference on learning representations*.

Balestriero, R., & LeCun, Y. (2022). Contrastive and non-contrastive self-supervised learning recover global and local spectral embedding methods, In *Advances in neural information processing systems*.

Ballard, D. H. (1987). Modular learning in neural networks., In *Aaai*.

Bardes, A., Ponce, J., & Lecun, Y. (2022). Vicreg: Variance-invariance-covariance regularization for self-supervised learning, In *Iclr 2022-10th international conference on learning representations*.

Bellemare, M. G., Dabney, W., & Rowland, M. (2023). *Distributional reinforcement learning* [http://www.distributional-rl.org]. MIT Press.

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

Bishop, C. M., & Nasrabadi, N. M. (2006). *Pattern recognition and machine learning* (Vol. 4). Springer.

Bordes, F., Balestriero, R., & Vincent, P. (2023). Towards democratizing joint-embedding self-supervised learning. *arXiv preprint arXiv:2303.01986*.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., & Zhang, Q. (2018). *JAX: Composable transformations of Python+NumPy programs* (Version 0.4.9). http://github.com/google/jax

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., & Shah, R. (1993). Signature verification using a" siamese" time delay neural network. *Advances in neural information processing systems*, *6*.

Burda, Y., Edwards, H., Storkey, A., & Klimov, O. (2018). Exploration by random network distillation, In *International conference on learning representations*.

Caron, M., Misra, I., Mairal, J., Goyal, P., Bojanowski, P., & Joulin, A. (2020). Unsupervised learning of visual features by contrasting cluster assignments. *Advances in neural information processing systems*, *33*, 9912–9924.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Et al. (2021). Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

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

Colas, C., Karch, T., Sigaud, O., & Oudeyer, P.-Y. (2022). Autotelic agents with intrinsically motivated goal-conditioned reinforcement learning: A short survey. *Journal of Artificial Intelligence Research*, *74*, 1159–1199.

Driess, D., Xia, F., Sajjadi, M. S. M., Lynch, C., Chowdhery, A., Ichter, B., Wahid, A., Tompson, J., Vuong, Q., Yu, T., Huang, W., Chebotar, Y., Sermanet, P., Duckworth, D., Levine, S., Vanhoucke, V., Hausman, K., Toussaint, M., Greff, K., … Florence, P. (2023). Palm-e: An embodied multimodal language model, In *Arxiv preprint arxiv:2303.03378*.

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Hessel, M., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Et al. (2017). Noisy networks for exploration, In *International conference on learning representations*.

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

Grill, J.-B., Strub, F., Altché, F., Tallec, C., Richemond, P., Buchatskaya, E., Doersch, C., Avila Pires, B., Guo, Z., Gheshlaghi Azar, M., Et al. (2020). Bootstrap your own latent-a new approach to self-supervised learning. *Advances in neural information processing systems*, *33*, 21271–21284.

Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018a). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, In *International conference on machine learning*. PMLR.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., Et al. (2018b). Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*.

Hafner, D., Lee, K.-H., Fischer, I., & Abbeel, P. (2022). Deep hierarchical planning from pixels, In *Advances in neural information processing systems*.

Hasselt, H. (2010). Double q-learning. *Advances in neural information processing systems*, *23*.

Heek, J., Levskaya, A., Oliver, A., Ritter, M., Rondepierre, B., Steiner, A., & van Zee, M. (2023). *Flax: A neural network library and ecosystem for JAX* (Version 0.6.8). http://github.com/google/flax

Henaff, M., Raileanu, R., Jiang, M., & Rocktäschel, T. (2022). Exploration via elliptical episodic bonuses, In *Advances in neural information processing systems*.

Hiraoka, T., Imagawa, T., Hashimoto, T., Onishi, T., & Tsuruoka, Y. (2021). Dropout q-functions for doubly efficient reinforcement learning, In *International conference on learning representations*.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Hoffman, M. W., Shahriari, B., Aslanides, J., Barth-Maron, G., Momchev, N., Sinopalnikov, D., Stańczyk, P., Ramos, S., Raichuk, A., Vincent, D., Et al. (2020). Acme: A research framework for distributed reinforcement learning. *arXiv preprint arXiv:2006.00979*.

Huang, S., Dossa, R. F. J., Ye, C., Braga, J., Chakraborty, D., Mehta, K., & Araújo, J. G. (2022). Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *The Journal of Machine Learning Research*, *23*(1), 12585–12602.

Jarrett, D., Tallec, C., Altché, F., Mesnard, T., Munos, R., & Valko, M. (2022). Curiosity in hindsight, In *Deep reinforcement learning workshop neurips 2022*.

Jiang, M., Rocktäschel, T., & Grefenstette, E. (2022). General intelligence requires rethinking exploration. *arXiv preprint arXiv:2211.07819*.

Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kingma, D. P., & Welling, M. (2014). Auto-encoding variational bayes. *stat*, *1050*, 1.

Kiran, B. R., Sobh, I., Talpaert, V., Mannion, P., Al Sallab, A. A., Yogamani, S., & Pérez, P. (2021). Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, *23*(6), 4909–4926.

Kostrikov, I., Nair, A., & Levine, S. (2021). Offline reinforcement learning with implicit q-learning, In *Deep rl workshop neurips*.

Kumar, A., Zhou, A., Tucker, G., & Levine, S. (2020). Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, *33*, 1179–1191.

Kuznetsov, A., Shvechikov, P., Grishin, A., & Vetrov, D. (2020). Controlling overestimation bias with truncated mixture of continuous distributional quantile critics, In *International conference on machine learning*. PMLR.

Lai, T. L., Robbins, H. Et al. (1985). Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, *6*(1), 4–22.

Li, S., Zhang, J., Wang, J., Yu, Y., & Zhang, C. (2021b). Active hierarchical exploration with stable subgoal representation learning, In *International conference on learning representations*.

Li, S., Zheng, L., Wang, J., & Zhang, C. (2021a). Learning subgoal representations with slow dynamics, In *International conference on learning representations*.

Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Goldberg, K., Gonzalez, J., Jordan, M., & Stoica, I. (2018). Rllib: Abstractions for distributed reinforcement learning, In *International conference on machine learning*. PMLR.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Mendonca, R., Rybkin, O., Daniilidis, K., Hafner, D., & Pathak, D. (2021). Discovering and achieving goals via world models. *Advances in Neural Information Processing Systems*, *34*, 24379–24391.

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.

Mu, J., Zhong, V., Raileanu, R., Jiang, M., Goodman, N., Rocktäschel, T., & Grefenstette, E. (2022). Improving intrinsic exploration with language abstractions, In *Advances in neural information processing systems*.

Mundt, M., Hong, Y., Pliushch, I., & Ramesh, V. (2023). A wholistic view of continual learning with deep neural networks: Forgotten lessons and the bridge to active and open world learning. *Neural Networks*.

Nachum, O., Gu, S. S., Lee, H., & Levine, S. (2018). Data-efficient hierarchical reinforcement learning. *Advances in neural information processing systems*, *31*.

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

Ozbayoglu, A. M., Gudelek, M. U., & Sezer, O. B. (2020). Deep learning for financial applications: A survey. *Applied Soft Computing*, *93*, 106384.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, *32*.

Pathak, D., Agrawal, P., Efros, A. A., & Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction, In *International conference on machine learning*. PMLR.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

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

Pislar, M., Szepesvari, D., Ostrovski, G., Borsa, D. L., & Schaul, T. (2021). When should agents explore?, In *International conference on learning representations*.

Pitis, S., Chan, H., Zhao, S., Stadie, B., & Ba, J. (2020). Maximum entropy gain exploration for long horizon multi-goal reinforcement learning, In *International conference on machine learning*. PMLR.

Pong, V. H., Dalal, M., Lin, S., Nair, A., Bahl, S., & Levine, S. (2020). Skew-fit: State-covering self-supervised reinforcement learning, In *Proceedings of the 37th international conference on machine learning*.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., & Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *The Journal of Machine Learning Research*, *22*(1), 12348–12355.

Rajeswaran, A., Kumar, V., Gupta, A., Vezzani, G., Schulman, J., Todorov, E., & Levine, S. (2017). Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*.

Ryan, R. M., & Deci, E. L. (2000). Intrinsic and extrinsic motivations: Classic definitions and new directions. *Contemporary educational psychology*, *25*(1), 54–67.

Schaul, T., Horgan, D., Gregor, K., & Silver, D. (2015). Universal value function approximators, In *International conference on machine learning*. PMLR.

Schneider, F., Xu, X., Ernst, M. R., Yu, Z., & Triesch, J. (2021). Contrastive learning through time, In *Svrhm workshop@ neurips*.

Schulman, J., Moritz, P., Levine, S., Jordan, M., & Abbeel, P. (2015). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

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

Shi, T., Karpathy, A., Fan, L., Hernandez, J., & Liang, P. (2017). World of bits: An open-domain platform for web-based agents, In *International conference on machine learning*. PMLR.

Sigaud, O., Akakzia, A., Caselles-Dupré, H., Colas, C., Oudeyer, P.-Y., & Chetouani, M. (2022). Towards teachable autotelic agents. *IEEE Transactions on Cognitive and Developmental Systems*.

Sun, M., Kurin, V., Liu, G., Devlin, S., Qin, T., Hofmann, K., & Whiteson, S. (2022). You may not need ratio clipping in ppo. *arXiv preprint arXiv:2202.00079*.

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

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., & Tassa, Y. (2012). Mujoco: A physics engine for model-based control, In *2012 ieee/rsj international conference on intelligent robots and systems*. IEEE.

Warde-Farley, D., Van de Wiele, T., Kulkarni, T., Ionescu, C., Hansen, S., & Mnih, V. (2018). Unsupervised control through non-parametric discriminative rewards, In *International conference on learning representations*.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. King's College, Cambridge United Kingdom.

Weng, J., Lin, M., Huang, S., Liu, B., Makoviichuk, D., Makoviychuk, V., Liu, Z., Song, Y., Luo, T., Jiang, Y., Et al. (2022). Envpool: A highly parallel reinforcement learning environment execution engine, In *Thirty-sixth conference on neural information processing systems datasets and benchmarks track*.

Wu, Y., Chen, X., Wang, C., Zhang, Y., & Ross, K. W. (2022). Aggressive q-learning with ensembles: Achieving both high sample efficiency and high asymptotic performance, In *Deep reinforcement learning workshop neurips*.

Yu, C., Liu, J., Nemati, S., & Yin, G. (2021). Reinforcement learning in healthcare: A survey. *ACM Computing Surveys (CSUR)*, *55*(1), 1–36.

Zhang, T., Xu, H., Wang, X., Wu, Y., Keutzer, K., Gonzalez, J. E., & Tian, Y. (2021). Noveld: A simple yet effective exploration criterion. *Advances in Neural Information Processing Systems*, *34*, 25217–25230.