Novelty Search Using Generalized Behavior Metrics

University of Southern Denmark - Faculty of Engineering

Master's Thesis in Advanced Robotics

Author

Benjamin B. Longet

belon18@student.sdu.dk

Student ID: 167596

Supervisor

Anders L. Christensen

andc@mmmi.sdu.dk

Date of submission: June 3, 2024

Number of characters: 107682

Number of standard pages: 44.87

Abstract

In the domain of evolutionary computation and reinforcement learning, novelty search encompasses algorithms aimed at enhancing exploration by emphasizing diversity of the phenotypes encountered by a model. By prioritizing novel behavior, these algorithms enable the model to break free from the monotonous, preconceived behaviors often enforced by the fitness function, thereby overcoming deception in complex domains. The effectiveness of novelty search hinges entirely on the careful design of the behavior metric, which typically must be tailored specially to a task. Expert knowledge is therefore required to reap the benefits of novelty search.

In this project, we propose to generalize the behavior metric based on sequential state data. We investigate the application of novelty search across a diverse set of environments, with this generalized metric. To accomplish generalization, two types of novelty search are applied. An off-policy version, which attempts to train multiple novel policies by adding a novelty score to the reward, based on the reconstruction error from a linear autoencoder. An on-policy version, meant to overcome immediate deception by training a LSTM-autoencoder on all state trajectories encountered during exploration.

We test the two algorithms, separately and in combination, across four distinct environments: Bipedal Walker, Swimmer, Deceptive Maze, and UR5. To visualize and measure the efficacy of the proposed approach, an open version of the maze without a target position is utilized. Using the Fréchet distance on resulting models in an open version of the maze environment, we determine the diversity of intermediate policies found with the on-policy algorithm. This measurement is also used to determine the diversity between the final models, found with the novel policy-seeking algorithm.

Generalized behavior metrics enables the off-policy version to find a wide array of distinct behaviors across all domains, while maintaining the ability to complete the specific tasks. Numerous behaviors are found for the Walker, one of the found behaviors for the Swimmer even outperforms the existing benchmarks, multiple inverse kinematics solutions are found for the UR5, and multiple distinct solutions are found for the Deceptive Maze.

The on-policy version allowed the Deceptive Maze to be solved consistently, usually within just a handful of iterations. The combination of both algorithms find both of the two distinct paths through the maze using two separate policies.

Contents

1 Introduction

Traditional machine learning approaches often focus on improving the performance for a static, task-related objective. Such methods are susceptible to deception and premature convergence to local optima of the reward function, the level of which scales with the complexity of the task [1]. Novelty search [2] is a concept that deviates from the traditional objective-based optimization. This method aims to increase the behavioral diversity of genomes in evolutionary algorithms (EA). By scoring genomes based on their behavioral novelty, the genetic algorithm is incentivized to cover a vast area of the behavioral space, effectively overcoming the challenges of deceptive or sparse reward environments. When used in conjunction with single-agent reinforcement learning (RL), the novelty score can be applied by replacing the step-wise reward with some measure of step-wise novelty [3]. By using the dynamic objective of seeking novel behavior, both EA and RL algorithms are able to avoid premature convergence to specific areas of the search space, by promoting thorough exploration of the resulting behaviors.

This thesis details the development of a novel policy-seeking algorithm that find multiple distinct policies by applying novelty search between separate policies. Furthermore, an on-policy version is applied with the aim of improving exploration. The primary objective of the project is to attempt to generalize the behavior metric, and prove this generalization by applying the same algorithms to a diverse set of environments.

Throughout the report, visual representations are employed to illustrate the differences in behavior for dynamical models. Some of these images struggle to fully capture the essence of the behaviors, as such, the project's repository contains a selection of GIFs that provide a comprehensive view of the different behaviors. The repository is found at https://github.com/BenjaminLonget/Novelty-Search-Master-Thesis/tree/main [4].

The report has the following structure:

- **Introduction**: Primarily focussing on the concept and benefits of novelty search, as well as the issue concerning multi-objective optimization.
- **Generalizing the Behavior Metric**: Covering the need for generalization, describes the problem statement, and examines a selection of previous work in the field.
- **Preliminary Theory**: Explaining the theory that leads up to the underlying algorithm used for training the models.
- **Environments**: Gives an explanation of the different environments, along with the changes made to some of these.
- **Fréchet Distance**: Where a few different distance metrics for measuring the similarities between trajectories in a metric space are explored, and the calculation of the discrete Fréchet distance is covered.

- **Discovering Novel Policies using Autoencoders**: The theory, implementation, and results of the novel policy seeking algorithm.
- **Improving Exploration with LSTM-Autoencoders**: The theory, implementation, and results of the on-policy, novelty search algorithm.
- **Combination of On-policy and Between-policy Novelty Search**: Showing the results of combining the two algorithms.
- **Discussion**: Containing a comprehensive analysis of the achieved results, the effectiveness, and implications of the work done, as well as the potential for future work.
- **Conclusion**: Summarizing the discoveries in the report and listing the success of the implemented algorithms and their potential contribution to the reinforcement learning field.

1.1 Novelty Search

The term Novelty Search was first introduced by Lehman and Stanley in [2]. In this work, they address one of the fundamental challenges in machine learning: the potential for deception in objective-based optimization. The relevant insight regarding this is that traditional objective-based approaches might inadvertently impede the evolutionary progress, by favoring known solutions and overlooking novel ones which could potentially contain more effective strategies. When following the gradient of the objective, the algorithm is prone to trap itself in the immediate local optimum of non-convex problems.

Instead of seeking to optimize for a predefined objective, they propose to simply search for novelty within the solution space. This involves optimizing for behaviors that diverge from the known solutions, while disregarding performance according to the objective function. Since the amount of simplistic behaviors are limited, more complex behaviors will appear as training advances. According to their experiments, when ascending the ladder of behavioral complexity, at least one solution to the objective is likely to be encountered.

$$\rho(x) = \frac{1}{k}\sum_{i=1}^{k} \text{dist}(x, \mu_i) \tag{1}$$

A common way of visualizing the effect of novelty search is by comparing it with objective based algorithms in a Deceptive Maze. In fig. 1, two versions of a Deceptive Maze are seen from the experiment in [2]. The starting position is indicated by the large circle, and the desired end position by the smaller circle. For the medium map, the agent starts in the upper left section and should end in the lower right. For the hard map, the start is in the lower left and the goal in the upper left. NeuroEvolution of Augmenting Topologies (NEAT) [5] is used as the underlying algorithm in this experiment, stopping early whenever a genome lands on the desired

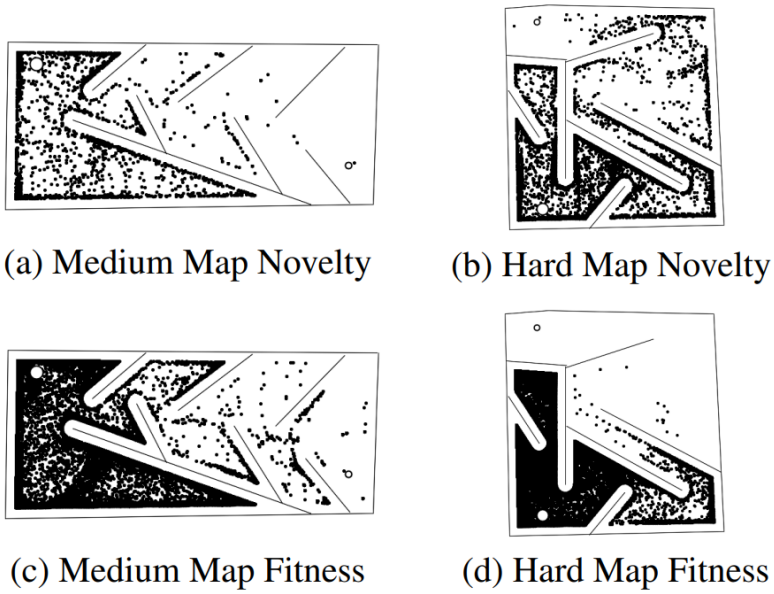


Figure 1: The Deceptive Maze used in [2]

position. The final position of each genome is used as the behavior metric, all of which are indicated on the maps. The novelty score for every position is calculated based on the average distance to the k-nearest neighbors, according to eq. (1). An archive containing the resulting final positions of the most novel past genomes is used when computing the novelty score for new genomes, along with the positions of the current generation. Their results showed that by ignoring the objective in favor of behavioral diversity, novelty search solved the hard map in 39 out of 40 attempts, where the fitness-based NEAT algorithm only found 3 solutions in 40 attempts.

By looking at the densities of the final coordinates in fig. 1, it becomes apparent that novelty search performs better in environments where deception is present. Deception, in this context, refers to scenarios where the model is required to go against the fitness gradient in order to succeed.

While handcrafting a fitness function for the maze could enable objective-based search to overcome the specific local optima, such an approach would be unlikely to generalize across similar tasks. Here, the open-ended objective of novelty search would allow for generalization across similar tasks, by encouraging diverse behaviors while disregarding the objective.

1.2 Multi-objective Optimization

Where the original paper on novelty search [2] completely ignores the task-specific objective for the Deceptive Maze, other tasks require a more nuanced approach that incorporate both fitness and novelty. In [6], they address the issue that arises when the behavior metric is decoupled from the fitness measurement. They illustrate this effect on the Deceptive Tartarus Problem, where the task is to make a bulldozer move blocks to the corners of

a grid environment. When defining novelty based on the Hamming distance of action sequences, the behavior space becomes excessively vast, such that novelty search alone is unable to cover even a fraction of this space. By applying a linearly normalized, weighted sum of rewards between fitness and novelty, they greatly increase the performance for this specific task, with their best results having a weight around 0.8. They further theorize that a dynamically adjusted weight based on fitness improvements might lead to an effective balance between novelty and performance.

Another approach is followed in [7]. Here, they utilize a multi-objective optimization method that aims to estimate the Pareto Frontier. The frontier is made up of non-dominated combinations of the different scores, where a solution is said to dominate another when it is better in at least one metric, and not worse in any other metrics. The Pareto front is then composed of all solutions which are not dominated by any other solutions, indicating the best trade-offs between the measured metrics. In the paper, they go on to show that an evolutionary algorithm which focus on estimating the Pareto front (Nondominated Sorting Genetic Algorithm 2, NSGA-2 [8]) for fitness and novelty, outperform pure novelty search in terms of the achieving the best fitness and rate of convergence in the original Deceptive Maze.

In [9], they modify the original Deceptive Maze to introduce a vast behavioral space, by removing some outer walls of the maze. This enables an endless number of possible novel behaviors to be found, most of which are completely irrelevant in terms of solving the task. They show that the fitness-based solutions evolve to the immediate local optimum, and the novelty search solutions drift around outside the bounds of the maze. In order to successfully complete the task, they introduce a minimal criteria version of novelty search, where the novelty score is only applied if the final coordinate is within the boundaries of the maze. In [10], the authors extend upon the minimal-criteria algorithm, by defining the criteria as a progressively increasing fitness threshold. By only applying a score to genomes that meet this criterion, their algorithm avoids spending time exploring the less fit sections of the behavior space. This shows that while minimal-criteria does explicitly optimize multiple objectives, it does allow for clever ways of incorporating the objective function into the novelty search algorithm.

1.3 Benefits of Novel Behavior Exploration

Besides automating the design of model controllers, RL is useful when trying to discover control strategies that outperform those designed with human expertise. Often, such control strategies involve behaviors that diverge from what developers might have anticipated. The pursuit of novel policies stems from the recognition that these unexpected approaches can offer alternative solutions or prove more effective in accomplishing the task at hand.

While the intended effect of the fitness function is to guide the models towards high-quality solutions, it often

results in very similar outcomes. This convergence is an inherent effect of a well-devised fitness function, as new models generally start from similar initial conditions, and therefore often discover the same immediate evolutionary stepping stones which lead them toward the same paths of learning.

In cases where the objective function is vaguely defined, or the task contains a great deal of stochasticity, different seeding of the models or the environment can allow multiple solutions to be found. An example of this is seen in [11], where the authors use random seeding of a stochastic PixelWorld environment with reoccurring content, to lead multiple agents toward diverse behavior defined by sensorimotor contingencies. These behaviors are subsequently combined in a hierarchical policy, able to recognize and combine the learned concepts.

Exploration of different behaviors would often require some type of reward augmentation, guiding the behaviors to optimize different or dynamic fitness functions. In [12], reward augmentation is applied through human preferences. Instead of following the typical state, action, reward process, the agent generates trajectories, before a human operator then decides which of two trajectories are preferred. The reward signal returned to the agent is based on a preference-predictor, which allow the agent to learn a vast array of human-guided behaviors. Besides training agents to complete various tasks entirely based on feedback, it also allowed them to guide the model toward complex novel behaviors, such as making the MuJoCo Hopper do consistent backflips.

Different behaviors might occur as a byproduct of optimizing for different objectives. One instance where this is the case is seen in [13]. In this paper, different bipedal "animals" are controlled with muscle contractions instead of regular joint actuation. On top of learning how to walk, the different models are also trained to walk at specific velocities. Walking at these different velocities is what develops distinctly different gaits, ranging from a slow-speed trot to a medium-speed hop to a high-speed run.

Novelty search has proven its worth in a wide array of tasks. In [14], novelty search is incorporated in a genetic programming framework to improve a vision based Simultaneous Localization and Mapping (SLAM) algorithm using KinectFusion.

Novelty-driven exploration is used to guide the learning of damage recovery for a six-legged robot and a robotic arm in [15]. By first mapping a set of high-performing behaviors, the robots are able to use these behaviors as intuition to guide trial-and-error learning towards some compensatory behavior whenever the robots are damaged.

In [16], minimal-criteria novelty search is used to generate diverse and feasible game levels, aiming to improve procedural content generation for games. They accomplish this by dividing a population into feasible and infeasible levels, and compare the effect of applying novelty search on the feasible genomes alone versus applying it on both sets.

By introducing diversity into the RL process, either through explicit reward shaping or through mechanisms

such as novelty search, the agent explores a broader range of behaviors and solutions. This approach encourages the discovery of novel strategies that may not have been apparent based on human expertise or traditional optimization methods.

In [17] a few common objections about novelty search are mentioned:

- Novelty search is not general.
- Ineffective if the behavioral space becomes too vast.
- Maintaining a growing archive of past behaviors becomes too computationally expensive.
- Novelty search is itself an objective and therefore still an objective-based search.

While they go on to give examples that oppose these objections, this thesis focuses on the generality of novelty search methods. The examples given in [17] all use specific handcrafted novelty metrics, whereas this work aims to demonstrate broader applicability across a wide range of environments and tasks.

2 Generalizing the Behavior Metric

Novelty Search has proven to be an effective method to circumvent deceptive local optima, as well as improving sample efficiency for certain tasks. It does however rely on a precise behavior metric for the given task, which limits the applicability across diverse domains. Designing a behavior metric is often labor-intensive and requires expert-knowledge of the domain. This section covers a few examples where expert knowledge is required, before listing a series of previous work regarding generalization of novelty search.

In [18], novelty search is utilized on two different environments with different behavior metrics. The first environment is the same Deceptive Maze as mentioned in section 1.1. Here, the behavior metric is the final coordinate of the genome after a fixed amount of environment interactions. The behavior metric in the Deceptive Maze is often designed this way, as it gives an intuitive understanding of the effect of novelty search while maintaining a perfect correlation with the fitness score. Defining behavior based entirely on the end result does not truly capture the nature of the behavior. Two different trajectories could have end positions near each other, granting them a low novelty score, even though they resulted from vastly different trajectories. This highlights a challenge in various tasks where the goal is to optimize for the same objective: multiple distinct solutions can lead to the same desired outcome.

The second environment used in [18] is a 3-Dimensional Bipedal Walker. For this environment, the behavior metric is based on the offset of center of mass at 1 second intervals. This proved to be a clever way of training the model, as novelty in terms of offset over time leads to oscillations of different frequencies, which in turn is the result of different gaits. By focussing on novel behavior described like this, they saw a prominent increase in

sample efficiency, finding a model with a natural-looking gait, which also greatly outperformed a fitness-based solution trained for the same amount of time-steps. This paper shows how novelty search in conjunction with a clever behavior metric might be beneficial, but it also showcases how domain-specific knowledge is required to design the behavior metric.

In [19], they attempt to apply novelty search with NEAT on the Humanoid Walker environment. For this challenge, they defined the behavior metric as the final coordinate of the humanoid, much like the metric from the original maze environments. Their results show that novelty search is able to teach the model to walk, as a result of rewarding different end coordinates, but the objective-based version generally outperforms the algorithm. When a deceptive trap was introduced in the environment, however, their algorithm was able to avoid the local optimum and obtain a better result than the fitness-based counterpart. They hypothesize that a more sophisticated behavior metric, aiming to encourage diversity for all relevant behavioral dimensions, might enable novelty search to perform as well as NEAT, but such a metric would be hard to construct and would likely exaggerate the amount of required computation.

Behavior, in the context of RL, refers to the actions that the agent decides upon based on the state it is in, driven by the goal of maximizing the cumulative reward. The agents' behavior can be said to be the manifestation of its policy, how it interacts with its surroundings, makes decisions, and adapts to achieve its goals.

Whether solving a maze, standing on one leg, or completing complex robotics tasks, the behavior of an RL agent should encapsulate its responses to environmental stimuli, effectively summarizing the movement patterns of the agent. For example, if we consider a maze-solving task, one agent might take a long, winding path to the goal, and another might take a more direct path. Both achieve the same end result but exhibit different behaviors, demonstrating the importance of a metric that captures these nuances rather than the final state.

2.1 Problem Statement

In light of the challenges posed by requiring task-specific knowledge to design behavior metrics, the concept of generalizing the metric emerges as a promising solution to broaden the application of novelty search. To achieve generalization, the resulting algorithm should utilize an all-encompassing behavior metric based entirely on the available information. Since the available information across different tasks lies solely within the state, action, and reward spaces, we need to find an approach based on these, in order to mitigate the need for domain-specific knowledge. Such an approach should allow for the algorithm to be deployed across a diverse set of environments, ideally with minimal to no manual adjustments.

Based on the aforementioned issues regarding the need for generalization, we formulate the following research questions:

- How can we generalize the behavior metric?

- How can we use this generalized metric to determine novelty?
- How can we combine the novelty score with the objective function?
- How can we prove that the found methods are successful in generalizing the behavior metric?

2.2 Previous Work

Devising an algorithm of the desired nature requires the knowledge from previous work in the field. A selection of different methods to generalize the behavior, measure the novelty, and combine novelty with fitness to assure solutions of high quality are examined.

In [20], they combine the exploration aspects of Goal Exploration Processes (GEP) with Deep Deterministic Policy Gradient (DDPG). They show how decoupling exploration and exploitation yields a large improvement to the performance of the trained models on two different environments. GEP differs slightly from novelty search in the sense that the algorithm tries to accomplish preset goals, and uses behavior encoded as state-action trajectories to achieve these goals. GEP first runs as a random policy, saving all new outcomes it encounters, along with the parameters that resulted in said outcome. In the second stage, a vector of the random outcomes are selected, and the agent attempts to achieve these outcomes by searching the k-nearest neighbors of discovered outcomes, and applying exploration noise to the parameters for these. At the goal exploration stage, the model is rewarded intrinsically based on its ability to reach the goal. Since all parameter-outcome pairs are saved in a replay buffer, DDPG are able to effectively generate different goal-oriented policies.

Diversity via Determinants (DvD) [21], generalizes the task-agnostic behavioral metric, by encoding the actions of all policies on a fixed set of states, and subsequently applying a squared exponential kernel on these encoded actions. The diversity in a set of policies is then measured by the determinant of all kernel combinations, and this measurement of diversity is added to the loss function. Finding a new policy with this loss function effectively maximizes novelty against all previous policies at once.

Interior Policy Differentiation (IPD) [22], measures the novelty between policies as the Wasserstein distance between their respective state distributions. This metric replaces the fitness once the difference of the behaviors are below some similarity threshold, making it a constrained optimization problem rather than multi-objective optimization. Switching to novelty optimization once the behavior is near previous behaviors, allows for the same initial stepping stones to be followed, before applying behavior repulsion from known solutions.

In [23], they devise the Diversity is All You Need (DIAYN) algorithm. DIAYN focuses on learning a set of diverse, distinct skills, without the need of a reward function. Just like novelty search, the core idea is to maximize behavioral diversity. An information-theoretic method is used to measure this diversity by maximizing the mutual information between a latent skill variable and the states encountered by the agent. A

discriminator is employed to predict the skill given the encountered states. The accuracy of this discriminator determines how distinguishable the skills are, thereby providing a measure of diversity. By maximizing the accuracy of the discriminator and the state entropy, they ensure that each skill leads to distinct and recognizable states, effectively distinguishing different behaviors.

In [24], they exploit the learning process of a neural network, trained to predict the next state from a current state-action pair. The idea is that the more times a specific combination is seen, the better the network is to predict the outcome. They expand upon this idea with Random Network Distillation (RND), where the novelty metric is defined as the prediction error of the output of a random network. Using a random network allows for this method to be used in stochastic environments, where similar state-action pairs would result in different next states. To combine the novelty metric with the fitness score, they use the weighted sum of rewards, where the weight is dynamically adjusted based on the change in quality of the solution.

In [25], a behavior learner network is trained to characterize the behavior for specific task types. The input for their behavior learner is the exploration fitness paired with the output of an "underlying" behavior metric. The underlying behavior metric encodes previously observed state-action pairs into a matrix, containing probabilities of the agent taking the actions given the states, similar in nature to state-count algorithms. The input for the learner is a set of initial behaviors and successful behaviors, and the output is a matrix containing importance weights of the state-action combinations. Once the learner has been trained, the resulting importance matrix is applied with the Hadamard product to the behavior of new genomes, where the novelty is measured with the Euclidean distance.

A generalized behavior metric is defined with a combined state count in [26]. Here, all state-action pairs encountered from a swarm of robots are discretized into values of high, medium, or low, before being hashed. A hash table is used to track the number of times a swarm encounters the different behavior states, where rare states are filtered. To measure the novelty, they use the Bray-Curtis dissimilarity between two tables before combining the score with a weighted sum of rewards.

In [27], the reconstruction error of sequential state data from an autoencoder is used as a novelty metric. To ensure that all policies are consistently influenced by the fitness gradient, they modify the task-novel bisector based on the angle between the two gradients. If the angle between the gradients exceeds $90°$, the fitness gradient is projected onto the line perpendicular to the novelty gradient, resulting in the fitness gradient always having an impact on the policy update. Conversely, if the angle between the two separate gradients are below $90°$, the conventional task-novel bisector is used. This conventional bisector is what results from doing a $50/50$ weighted sum of rewards. The resulting vector, conventional or projected, is then utilized as the response to the underlying algorithm.

An overview of the covered methods is seen in table 1. While numerous alternative approaches exist, a broad

Behavior metrics	Novelty metrics	Application of the novelty score
[20] Encoded state-action trajectories	Prediction error of distance to goal	Replacing fitness
[21] Encoding actions through fixed states	Determinant of Gaussian kernel	Replacing fitness
[22] State distributions as Gaussians	Wasserstein distance	Constrained
[23] Encountered states	Discriminator accuracy	Replacing fitness
[24] State-action pairs	Prediction error of next state	Dynamical weighted sum of rewards
[25] State-action probabilities	Euclidean distance	Replacing fitness
[25] Hadamard of learned importance matrix	Euclidean distance	Replacing fitness
[26] Hash tables with state-action counts	Bray-Curtis dissimilarity	Fixed weighted sum of rewards
[27] Sequential state data	Reconstruction error	Task-novel bisector

Table 1: Potential options for generalization

spectrum of possibilities is considered. We decided to use sequential state-data, as it encapsulates the behavior in terms of the results of the agents' actions. The actions themselves can also be said to be encoded within the state-transition dynamics, which is maintained with sequential state data. Autoencoders meant to reconstruct the behavior metric will serve as our method of measuring novelty, as they should learn the important patterns present in the input, and simultaneously functions as a replacement of the archive. Integration of the novelty score will be achieved with a dynamically weighted sum of rewards, as this allows the agent to regulate the importance of novelty dependent on the results it achieves.

3 Preliminary Theory

This section covers the basic theory about RL, including value-based methods, policy-based methods, policy-gradient methods, and actor-critic methods. The section ends with an introduction to Proximal Policy Optimization (PPO), which is the RL algorithm used throughout the project. The entire section is based on the Deep RL course from Hugging Face [28].

3.1 Introduction to Reinforcement Learning

RL covers one of the three main machine learning (ML) techniques, alongside supervised and unsupervised learning. While supervised learning focuses on mimicking human-designed outputs, and unsupervised learning seeks patterns within data, RL uses an agent to learn through interactions with an environment [29].

The framework of RL has four key components:

- Agent, the learner or decision-maker that interacts with the environment.
- The environment in which the agent operates.
- The policy that governs the decision-making process of the agent, dictating the actions taken based on the observed state.

- The reward signal returned to the agent, indicating the desirability of its actions and guiding the learning process.

Compared to the other ML techniques, RL learns through trial and error, with the main goal of finding the optimal policy π^* which maximizes the expected cumulative return.

Value-Based Methods

With value-based RL methods, the aim is to learn a value function which leads to the optimal policy π^*. The objective is to minimize the loss between the predicted and target values to approximate the true action value function. Methods such as Q-Learning does not have a policy, it is however implicit within the value function, in such a case the policy could be ϵ-greedy applied to the values stored in the Q-table.

Policy Based Methods

Policy based methods opt to directly learn the approximation of π^*, without the need of learning the value function. The general idea is to parameterize the policy, usually as a neural network π_θ, where θ would be the parameters. The output of such a network would be a probability distribution over actions given the states.

$$\pi_\theta(s) = P[A|s;\theta] \tag{2}$$

The objective is then to maximize the performance of the parameterized policy by using gradient ascent on the objective function $J(\theta)$, which defines the expected cumulative discounted reward. The parameters are optimized indirectly by maximizing the local approximation of the objective function.

Policy Gradient Methods

Policy gradient methods is a subclass of policy-based methods. These are generally on-policy as the trajectories are gathered with the most resent version of π_θ. Compared to the standard policy-based method, policy gradient methods optimize the parameters of the policy directly by performing gradient ascend on the performance of the objective function.

$$J(\theta) = \mathbb{E}_{\tau\sim\pi}[R(\tau)] = \sum_{\tau} P(\tau;\theta)R(\tau) \tag{3}$$

$$\theta \leftarrow \theta + \alpha \cdot \nabla_\theta J(\theta) \tag{4}$$

The objective function to maximize is seen in eq. (3). Gradient ascent is used to find the parameters that maximize this function. The update step of the parameters would then follow eq. (4).

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)] \tag{5}$$

Calculating the true gradient of the objective function would require probability calculations of all possible trajectories. The state distribution also needs to be differentiated to obtain the gradient. Since this is unfeasible in most cases, the Policy Gradient Theorem, eq. (5), is used to approximate the gradient.

Actor-Critic Networks

Actor-Critic methods combine value-base and policy-based methods. The actor is what controls how the agent behaves and is in itself a policy-based method. The critic is value-based, and measures the performance of the actions taken by the Actor.

$$\Delta\theta = \alpha \nabla_\theta (\log \pi_\theta(s,a)) \hat{q}_w(s,a) \tag{6}$$

$$\Delta w = \beta(R(s,a) + \gamma \hat{q}_w(s_{t+1}, a_{t+1}) - \hat{q}_w(s_t, a_t)) \nabla_w \hat{q}_w(s_t, a_t) \tag{7}$$

The update of the actor, eq. (6), relies on the action value estimate from the critic. Update of the critic, eq. (7), uses the temporal difference and the gradient of the value function.

3.2 Introduction to Proximal Policy Optimization

> *OpenAI: "We're releasing a new class of reinforcement learning algorithms, Proximal Policy Optimization (PPO), which perform comparably or better than state-of-the-art approaches while being much simpler to implement and tune. PPO has become the default reinforcement learning algorithm at OpenAI because of its ease of use and good performance."*[30]

PPO [31], based on Trust Region Policy Optimization (TRPO) [32], is a model-free, online, on-policy RL algorithm. Like TRPO, the core principle of PPO lies in its cautious approach to policy updates, ensuring that adjustments are made gradually to prevent overshooting potential optima.

$$L_t^{clip}(\theta) = \hat{\mathbb{E}}_t[min(r_t(\theta) \cdot \hat{A}_t, clipped(r_t(\theta), 1-\epsilon, 1+\epsilon) \cdot \hat{A}_t)] \tag{8}$$

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \tag{9}$$

$$\hat{A}_t = Q(s,a) - V(s) \approx r_t + \gamma V(s') - V(s) \tag{10}$$

$$\hat{A}_t^{GAE} = \delta_t + (\gamma\lambda)\delta_{t+1} + ... + ... + (\gamma\lambda)^{T-t+1}\delta_{T-1} \tag{11}$$

PPO constrains the policy updates by using the Clipped Surrogate Objective depicted in eq. (8). Here, r_t represents the probability ratio between the current and the old policy, of taking some action in some state. This ratio replaces the log probability used in the policy objective function. The ratio is clipped between $[1-\epsilon, 1+\epsilon]$ to ensure that policy updates are not excessive if the ratio happens to be disproportionately large. The estimated advantage of a single step is seen in eq. (10), and is computed based on the difference between the discounted return and the state's estimated value. To ensure the application of advantage is applied over a fixed time horizon, the Generalized Advantage Estimator (GAE) is applied as seen in eq. (11), where $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$, γ is the temporal discount factor, λ is a bias-variance tradeoff parameter, and r_t is the observed reward.

$$L_t^{\text{CLIP+VF+S}} = \hat{\mathbb{E}}_t[L_t^{\text{CLIP}}(\theta) - c_1 L_t^{\text{VF}}(\theta) + c_2 S[\pi_\theta](s_t)] \tag{12}$$

The final objective function to be maximized is seen in eq. (12). Here c_1 and c_2 are coefficients, L_t^{VF} is the squared-error value loss $(V_\theta(s_t) - V_t^{\text{target}})$, and $S[\pi_\theta](s_t)$ is an entropy bonus that rewards effective exploration. Since PPO enforces a limit on the change in actions of the policy, it can be said to apply the same limit in the behavioral space, as the behavior is the result of all the actions the model takes. This should mean that PPO pairs well with the concept of novelty search.

Algorithm 1 PPO with Actor-Critic Structure

for Iteration = $1, 2, ...$ **do**
  for Actor = $1, 2, ..., N$ **do** ▷ One environment per actor running in parallel
    Run policy $\pi_{\theta_{old}}$ for T time steps
    Compute the estimated advantages $\hat{A}_1, ..., \hat{A}_T$
  end for
  Optimize surrogate L with respect to θ for K epochs and minibatch size $M \leq NT$
  $\theta_{old} \leftarrow \theta$
end for

Implementation of the PPO algorithm is done with StableBaselines3 (SB3) [33], a robust library for reinforcement learning algorithms. Minimal adjustments are made to increase computational efficiency. Specifically, the minibatch size has been reduced from 64 to 32, and the number of training epochs has been decreased from 10 to 3. As generalization is a core concept of this thesis, all other aspects of the base algorithm remains untouched. This decision enables the work of this thesis to be combined with other underlying algorithms that use the same

callback structure as SB3's PPO.

4 Environments

Gymnasium [34] from OpenAI, also known as Gym, is used as the primary environment framework. Its appeal lies in the unified interface, enabling seamless interaction across a diverse set of environments. It offers a vast library of classical control tasks, robotics simulations, image-based, and text-based challenges. Numerous third-party environments use Gymnasium, many of which are available through public repositories.

The environments used to test the effectiveness of generalized novelty search should follow some or more basic criteria:

- Environmental complexity regarding model behavior.

  Some environments contain tasks that are too simple to solve with different behaviors. The complexity should therefore be adequate to apply novelty search.

- The generic fitness-based solution should usually result in similar behaviors.

  Not necessarily a requirement, but it does help to cement both the need and the effectiveness of novelty search.

- Deceptive solutions.

  Environments with one or more local optima are ideal use cases for novelty search, and are therefore preferred.

- The state-space should primarily contain information that are relevant to describe the behavior.

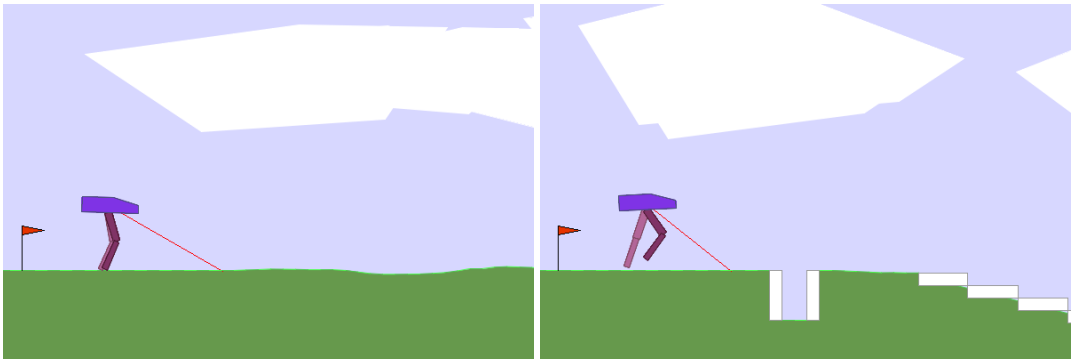
  Some Gymnasium environments use a dictionary-style observation space that include additional information such as desired and achieved target. Including these in the behavioral metric could influence the calculation of the novelty score. Some desired position will, for example, have nothing to do with the novelty of the models' behavior, but would still have an influence on the calculated novelty score.

- The range of the step-wise fitness should be approximately equal to the range of the novelty-score, since the weighted sum of rewards will be used as the reward feedback.

Four different environments using three different physics engines are picked according to the above criteria. Different tasks in different engines are used in an attempt to showcase the ability to generalize the behavior metric for vastly different scenarios.

4.1 Bipedal Walker

One of the basic environments that are part of the Gymnasium library is the 2D Bipedal Walker [35], powered by the Box2D physics engine [36]. This environment features a simple yet challenging 4-joint, two-dimensional walker traversing randomly generated terrain.



(a) Box2D BipedalWalker-v3 (b) Hardcore version of the BipedalWalker environment

The objective in this environment is for the walker to progress to the right by applying torque to its four joints. The hardcore version of the environment contains the additional challenge of stairs, stumps and pitfalls, as well as even more rugged terrain.

Rewards are based on the horizontal velocity, with a small penalty for applying torque, such that efficient movement yields higher scores. Additionally, a large penalty is applied when the hull contacts the ground.

This fitness function often results in the final models converging to very similar behaviors, thus making it an ideal environment for the objective of this thesis.

Joint name	Data type	Range
Hip_1	float32	[-1, 1]
Knee_1	float32	[-1, 1]
Hip_2	float32	[-1, 1]
Knee_2	float32	[-1, 1]

Table 2: Action space of the Walker

Observation	Data type	Range
Hull angle	float32	$[-\pi, \pi]$
Hull angular velocity	float32	$[-5, 5]$
Horizontal velocity	float32	$[-5, 5]$
Vertical velocity	float32	$[-5, 5]$
Joint angles	float32	$[-\pi, \pi]$
Joint angular velocities	float32	$[-5, 5]$
Legs ground contact	float32	$[0, 5]$
Lidar measurements	float32	$[-1, 1]$

Table 3: Observation space of the Walker

The action space encompasses the torque applied to the four joints, ranging from -1.0 to 1.0. The observation space consists of 24 measurements, and contains the positional velocity, angular velocity and current angle of the hull. Angle and angular velocities of the four joints. A ground contact flag for each of the feet. As well as 10 LIDAR measurements at different angles from the hull to the ground.

User	Version	Episodes before solve	Write-up	Video
timurgepard	3.0	28 (Symphony🤖 ver 3.0)	writeup	video
timurgepard	3.0	40 (Symphony🎹 ver 2.0)	writeup	video
Benjamin & Thor	3.0	57 (TRPO with OU action noise)	writeup	
timurgepard	3.0	100 (Monte-Carlo🌊 & Temporal Difference🔥)	writeup	
Lauren	2.0	110	writeup	Video
Mathias Åsberg 😎	2.0	164	writeup	Video
liu	2.0	200 (AverageEpRet:338)	writeup	

Figure 3: BipedalWalker challenge, leaderboard [37] top 7 as of 23-4-2024

This environment is also part of a challenge issued by OpenAI. The challenge consists of getting an 100-episode average score of above 300. Successfully completing this challenge makes one eligible for an entry on the leaderboard, where the ranking is based on the number of iterations before the challenge is completed. The current top 7 is seen in fig. 3.

4.2 Swimmer

The second environment is the Swimmer, running on the Multi-Joint dynamics with Contact (MuJoCo) [38] physics engine.

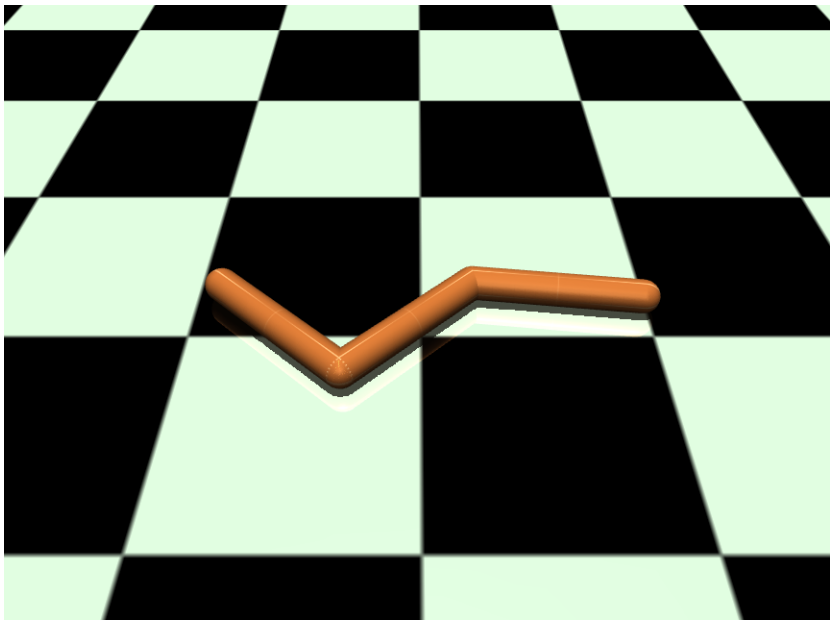


Figure 4: MuJoCo Swimmer-v4

The model consist of two (or more) joints, controlling three links connected as a chain, where one joint is always connected to two links. The task of the Swimmer is to move right as fast as possible in a viscous, fluid-like environment. It is possible to change the number of links, the mass, and the length of each link, as well as the viscous coefficient by passing a custom MuJoCo XML file to the environment.

Joint name	Data type	Range
Torque joint 1	float32	[-1, 1]
Torque joint 2	float32	[-1, 1]
...	float32	[-1, 1]

Table 4: Action space of the Swimmer

Observation	Unit	Range
Angle of the front tip	rad	$[-\inf, \inf]$
Angle of the first rotor	rad	$[-\inf, \inf]$
Angle of the second rotor	rad	$[-\inf, \inf]$
...	rad	$[-\inf, \inf]$
Velocity of the tip along the x-axis	m/s	$[-\inf, \inf]$
Velocity of the tip along the y-axis	m/s	$[-\inf, \inf]$
Angular velocity of front tip	rad/s	$[-\inf, \inf]$
Angular velocity of first rotor	rad/s	$[-\inf, \inf]$
Angular velocity of second rotor	rad/s	$[-\inf, \inf]$
...	rad/s	$[-\inf, \inf]$

Table 5: Observation space of the Swimmer

The model is rewarded based on the velocity of the tip along the x-axis, with the addition of a control cost penalty for applying torque.

As the movement of the entire structure directly affects the forward velocity, the behavior of the model is tightly correlated with the resulting performance, rendering it a good contender for novelty search. Furthermore, models resulting from training with the predetermined fitness function, seem to always converge to a sideways "frog-leg" swim, as this gains a consistent, though minor, positive reward.

4.3 UR5

The third environment is a generic reach task with a UR5 robotic arm. The environment is based on the public repository found at [39], which is a Gymnasium implementation of Hierarchical Actor-Critic [40]. The environment is built upon the PandaReach environment from Panda-Gym [41], which uses the PyBullet physics engine [42].

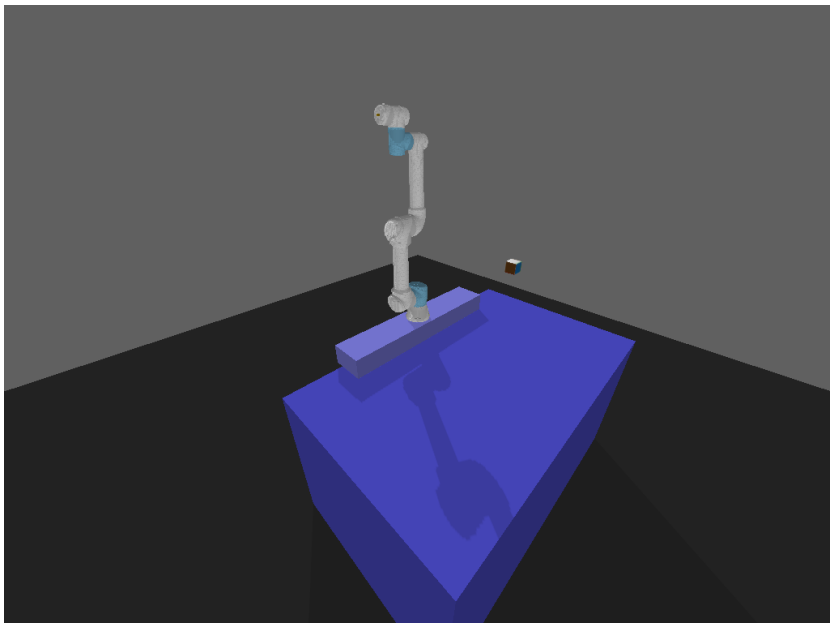


Figure 5: Bullet UR5 reach task

Joint name	Data type	Range
Base	float32	[-1, 1]
Shoulder	float32	[-1, 1]
Elbow	float32	[-1, 1]
Wrist_1	float32	[-1, 1]
Wrist_2	float32	[-1, 1]
Wrist_3	float32	[-1, 1]

Table 6: Action space of the UR5

Observation	Unit	Range
EE x-position	m	$[-10, 10]$
EE y-position	m	$[-10, 10]$
EE z-position	m	$[-10, 10]$
EE yaw (z)	rad	$[-10, 10]$
EE pitch (y)	rad	$[-10, 10]$
EE roll (x)	rad	$[-10, 10]$
Base joint angle	rad	$[-10, 10]$
Shoulder joint angle	rad	$[-10, 10]$
Elbow joint angle	rad	$[-10, 10]$
Wrist_1 joint angle	rad	$[-10, 10]$
Wrist_2 joint angle	rad	$[-10, 10]$
Wrist_3 joint angle	rad	$[-10, 10]$

Table 7: Observation space of the UR5

Several adjustments have been made to the original environment to ensure its functionality and alignment with specific requirements:

- The action implementation has been updated to utilize the **setJointMotorControlArray** method from the physics engine, replacing the previous method using **resetJointState**. This change enables the simulation of the actual robot dynamics.
- A via point and a randomly moving obstacle is removed from the environment. Additionally, the initial and desired end-effector (EE) position is set to fixed values.
- The observation space has been reduced from a dictionary format, containing the additional coordinate of the via point, desired EE position, and distance to the obstacle. The new state space simply contains the current joint values and EE coordinates. This ensures that the state-space only consists of information that is relevant regarding novel behavior.
- The fitness function is revised to include EE position and orientation, as well as a slight bonus for decreasing the distance to the target. Earlier versions never converged to achieving the desired position, but rather opted for hovering right next to the target, as this achieved a high cummulative reward.

$$r_t = \exp(-d_t) + o_t \cdot w_o + (d_{t-1} - d_t) \cdot w_{\Delta d} \tag{13}$$

The final fitness score for the UR5 is calculated according to eq. (13), where d_t is the distance between current and desired end EE position, o_t is the normalized angle between the current and desired EE orientation, and w is weights applied to the orientation and distance change bonus. A substantial additional reward is granted when the distance and orientation is below a specified threshold, signifying successful task completion.

The decision to utilize this environment stems from the recognition that the underlying algorithm effectively

mirrors gradient ascent, akin to the iterative nature of numerical inverse kinematics (IK) solvers. Using a numerical IK solver yields the single joint configuration that is nearest the initial joint values, the same applies when solving with a policy optimization algorithm. To find all eight configurations, the analytical IK solution is usually required.

If subsequent solutions are simultaneously fit and novel compared to their predecessors, some of the eight IK configurations should be ascertainable with the final algorithm, in a numerical-like fashion.

4.4 Deceptive Maze

The maze domain is a commonly used environment when dealing with exploration strategies such as novelty search, as it effectively showcases the effect of generic algorithms getting stuck in local optima. The two-dimensional view of the maze further serves as an excellent proxy for describing behavior in a comprehensible manner.

The environment is from [43], and is implemented through Gymnasium-Robotics [44]. It runs on MuJoCo, and allows for custom mazes, defined as a list of lists, to be passed to the initializer. Two controllable models are available; a two degree of freedom, force controlled ball, and the classic MuJoCo ant model.

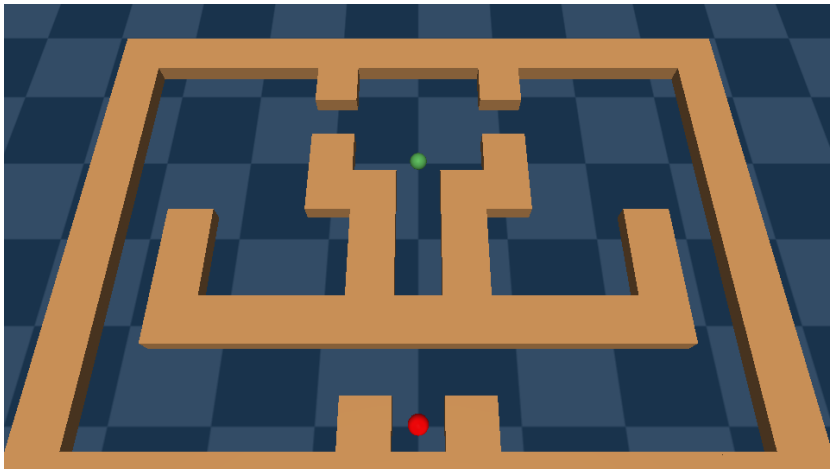


Figure 6: Double Deceptive Maze

The designed deceptive maze is seen in fig. 6. The starting position is indicated by the green ball, and the goal is indicated by the red ball. The maze features one prominent local optimum on the direct path between the start and end position, and two secondary local optima to capture policies that might get out of the starting room. The lips on the outer wall that help define the starting room, as well as the ones on either side of the goal, serves to ensure that an outer wall hugging policy is unable to complete the maze. The maze is mirrored such that two novel policies should be obtainable.

Joint name	Unit	Range
Force x	N	[-1, 1]
Force y	N	[-1, 1]

Table 8: Action space of the Ball model

Observation	Unit	Range
X-position	m	$[-\inf, \inf]$
Y-position	m	$[-\inf, \inf]$
X-velocity	m/s	$[-\inf, \inf]$
Y-velocity	m/s	$[-\inf, \inf]$

Table 9: Observation space of the Ball model

The observation space is once again changed from a dictionary to a box observation to comply with the environmental requirements.

The fitness function is changed from $\exp(-\text{distance})$, as this allows for exploring away from the goal without much of a penalty, to instead be the linear distance, normalized to this specific maze, such that being in one of the far corners yield a score of -1 per step, and being on the goal yields a score of 1. An additional reward is applied when reaching the goal. The linearization of the score greatly increases the difficulty of finding a proper policy, and further cements the convergence to the immediate local optimum, which happens within just a few policy iterations.

Open Maze Variant

A second variant of the maze is created, this one simply being an open space where the reward is set to either 0 or some factor of the velocity in the x direction. This environment is used to give an intuitive understanding of the implemented novelty search algorithms, as well as enabling a 2D trajectory comparison.

5 Fréchet Distance

Measuring the likeness between trajectories is essential to validate the effectiveness of the applied novelty search algorithm on the Open Maze environment. A few methods to accomplish this in a metric space are considered in this section.

A weighted cosine similarity could be used to measure the similarities between trajectories. This approach would use both the angle and the distance to each point in both trajectories to calculate a similarity metric. By using the combination of spatial and directional information, this method could serve as a decent comparison.

The Jensen-Shannon distance measures the similarity between two probability distributions. This could be used on the final coordinates found during exploration to validate that novelty search results in more diversity.

Subsets of the exploration coordinates could also be compared to show the evolution of exploration.

The Hausdorff distance is another metric capable of quantifying dissimilarities between trajectories or shapes. It measures the maximum distance from a point in one trajectory to the closest point in the other trajectory, enabling it to capture the overall difference between trajectories regardless of their lengths or shapes.

The Fréchet distance is a metric used to compare both distance and similarity of shape between two trajectories. The intuitive interpretation of the Fréchet distance can be expressed as a man walking his dog. In this analogy, the man walks along one trajectory and the dog along the other. Both are allowed to walk at varying speeds, but none are allowed to backtrack. The Fréchet distance in such a scenario would represent the shortest possible leash that allow both the man and the dog to traverse their respective trajectories from start to finish.

5.1 Implementation

Algorithm 2 Recursive Discrete Fréchet Distance

Input: curves $P = [p_1, p_2, ...]$ and $Q = [q_1, q_2, ...]$
Initialize $i = len(P) - 1$, $j = len(Q) - 1$, $ca = \text{ones}(i, j) * (-1)$
function $c(ca, i, j, p, q)$:
  if $ca[i, j] > -1$ **then**
    return $ca[i, j]$
  else if $i == 0$ **and** $j == 0$ **then**
    $ca[i, j] = \text{dist}(p_i, q_j)$
  else if $i > 0$ **and** $j == 0$ **then**
    $ca[i, j] = \max(c(ca, i - 1, 0, P, Q), \text{dist}(p_i, q_j))$
  else if $i == 0$ **and** $j > 0$ **then**
    $ca[i, j] = \max(c(ca, 0, j - 1, P, Q), \text{dist}(p_i, q_j))$
  else if $i > 0$ **and** $j > 0$ **then**
    $ca[i, j] = \max(\min(c(ca, i - 1, j, P, Q), c(ca, i, j - 1, P, Q), c(ca, i - 1, j - 1, P, Q)), \text{dist}(p_i, q_j))$
  else
    $ca[i, j] = \inf$
    return $ca[i, j]$

The discrete Fréchet distance is implemented based on algorithm 2 [45] according to the pseudocode described in [46]. For trajectories existing in a metric space, the distance is calculated as the Euclidean distance. It is possible to calculate the Fréchet distance on more complex state-spaces by using other distance functions.

5.2 Example

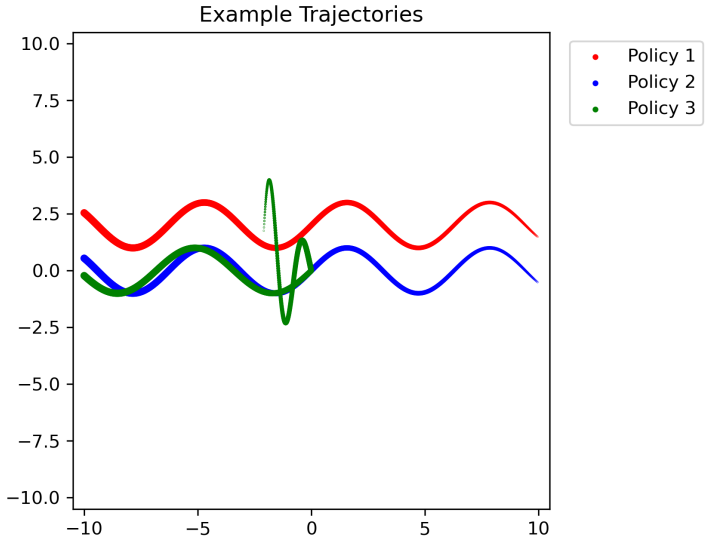
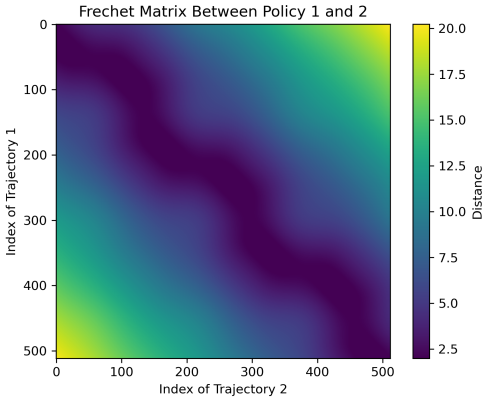
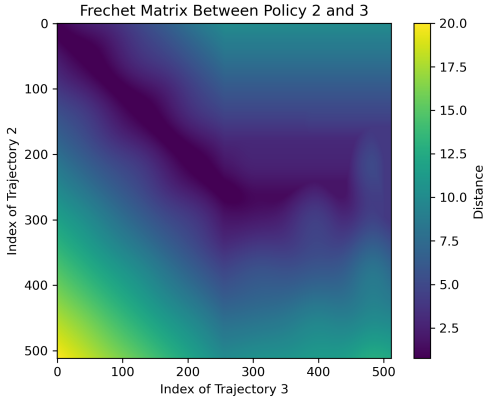


Figure 7: Example trajectories

The Fréchet distance between three example trajectories is illustrated in fig. 7. The final calculated Fréchet distance corresponds to the value at the last step of both trajectories, indicated by the lower right corner of the matrix.



(a) Fréchet matrix between the two similar trajectories offset by 2.5



(b) Fréchet matrix between two different trajectories

Figure 8: Example Fréchet matrices

By examining the heat maps of the Fréchet matrices depicted in fig. 8, it becomes apparent that these matrices

contain general information regarding similarities between the shape of the trajectories, even when the trajectories are spatially distant. A consistent value along the diagonal of the matrix suggests a high similarity between the trajectories, with the final Fréchet distance providing a quantitative measure of their distance.

6 Discovering Novel Policies using Autoencoders

This section covers the theory, implementation, and results of a novel policy-seeking algorithm, aimed at training multiple policies that all exhibit distinct behavior. The approach of using autoencoders (AEs) to generalize and quantify behavior differences draws significant inspiration from [27].

AEs, described in detail in chapter 14 of [47], are an unsupervised learning method commonly used for dimensionality reduction and representation/feature learning. The primary objective of an AE is to encode the input data into a lower-dimensional representation, and then reconstruct the original data by decoding from this latent space.

Tested in [48], encoding to a lower dimension effectively applies a bottleneck on the information flow. When this latent dimension is sufficiently small, the AE learns to capture the most essential features that are required for accurate data reconstruction. The AE is said to be **undercomplete** when the latent space have a lower dimension than the input data.

Similarities can be drawn between undercomplete AEs and Principal Component Analysis (PCA). When the decoding layer is linear and trained on the mean squared reconstruction error, the AE effectively learns the principal subspace of the training data. In [49], an AE is trained to function like PCA, by training it with one dimension in the latent space at a time, increasing the dimension as training progresses much like the general procedure of PCA.

By encoding behavioral information into a compact latent space, patterns within the behavior become recognizable. If an AE is trained on the behavior of a specific policy, it should be able to reconstruct similar data with high accuracy. The AEs ability to reconstruct the behavioral data can subsequently be used to measure the novelty of new behavior.

6.1 Rewarding based on reconstruction error

Since the range of reconstruction errors might vary greatly between different environments, it is necessary to design a reward function that both normalizes the score and scales it according to the general distribution of reconstruction errors.

$$f_\sigma(x) = \frac{1}{1 + e^{-x}} \tag{14}$$

The logistic sigmoid function eq. (14) is used as the base premise for the novelty reward function. $2 \cdot f_\sigma(x) - 1$ adjusts the range of the output to be $[-1.0, 1.0]$. The midpoint of the curve can be determined by subtracting $(x - m)$, and the desired steepness of the slope by division $\frac{x-m}{s}$. Using the mean, μ, and the standard deviation, σ, directly results in eq. (15).

$$f_n(x) = \frac{2}{1 + e^{-(1/(\sigma+\epsilon)) \cdot (x-\mu)}} - 1 \tag{15}$$

Where x is the mean-squared reconstruction error from the AE, σ and μ are the mean and standard deviation of all previous reconstruction errors. A small ϵ is added to avoid potential zero-division. This equation effectively utilizes the standard deviation to determine the slope of the function, where a smaller deviation will result in a more aggressive slope centered around the mean. Measurements that are precisely one standard deviation above the mean will result in a novelty score of 0.462. Using the distribution to calculate the score ensures that the scale of the score remains the same, even for different environments with potentially large differences in reconstruction error.

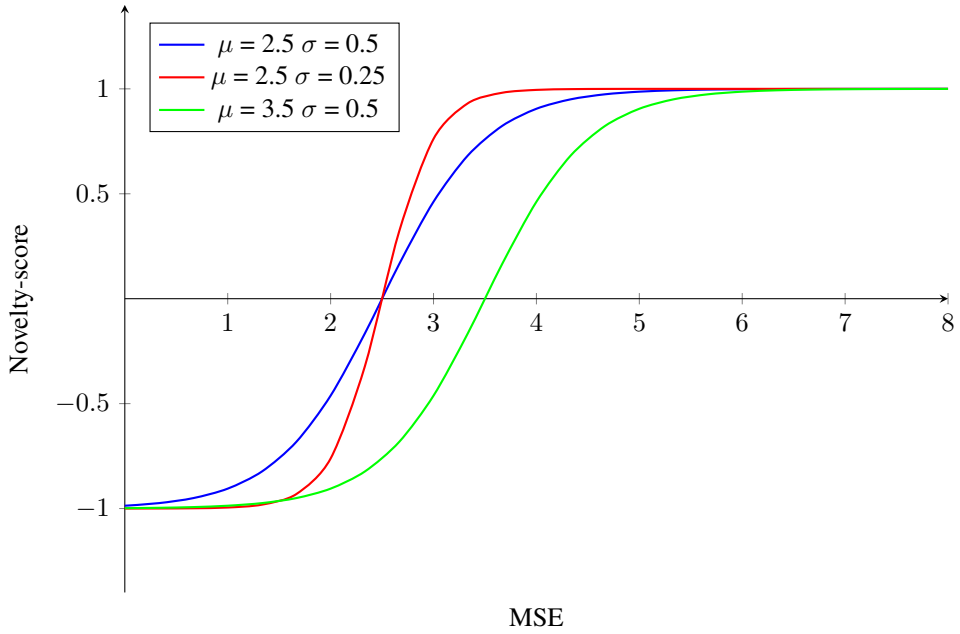


Figure 9: Novelty score based on the distribution of mean squared reconstruction error and eq. (15).

The graph in fig. 9 shows how eq. (15) is affected by different means and standard deviations.

A consequence of the novelty score being based on the mean and standard deviation is that the score tends to converge towards 0. As the algorithm follows the novelty gradient, the means will increase over time, leading to a decrease in the novelty score. This decay over time is deemed acceptable as it gives the policy a chance to focus more on the fitness score, whose gradient would be comparably more prominent towards the end of

training.

6.2 Two-Objective Trade-off

Whenever the aim is to optimize multiple objectives simultaneously, such as when optimizing for both fitness and novelty, finding an optimal trade-off between these objectives is crucial. Simply applying a uniform weighted sum of rewards, such as a $50/50$ split, could inadvertently prioritize one gradient over the other if this gradient happens to contradict the other while being more prominent, leading to suboptimal results.

Inspired by [24], we implement a dynamically adjusted weighted sum of rewards, which is based on the progression of the evaluation fitness score. If fitness seems to increase, the weight is shifted towards favoring the fitness score. If fitness does not see an increase, the algorithm weights the novelty score higher. One noteworthy issue that might arise from the weight adjustment is the fact that updating the policy towards preferring novelty does not necessarily yield an increase in fitness. In such a case, the policy would converge towards an entirely novel behavior, with the weight anchored for novelty, hindering its ability to accomplish the task at hand. In an attempt to avoid this self-perpetuating weighting, we decide to reset the weight whenever it reaches a minimum or maximum value.

$$r = (1 - \alpha) * r_{task} + \alpha * r_{nov} \tag{16}$$

$$\alpha = \begin{cases} 0.5 & \text{if } \alpha \leq \alpha_{min} \text{ or } \alpha \geq \alpha_{max} \\ \alpha - c & \text{if mean}(fitness_{prev}) > fitness \\ \alpha + c & \text{if mean}(fitness_{prev}) < fitness \end{cases} \tag{17}$$

The final reward is seen in eq. (16), where the weight is determined periodically based on eq. (17). Here $fitness_{prev}$ is the average episodic evaluation fitness from the 5 previous iterations, and $fitness$ is the current episodic evaluation fitness. The step-size constant c is set to 0.05, and the limits, α_{min} and α_{max}, is set to 0.1 and 0.9 respectively. Having a step-size of 5% allows the weight to be reset multiple times over a single run, enabling the algorithm to potentially switch between novelty and fitness.

6.3 Implementation Details

The entire implementation is carried out in Python, leveraging its extensive libraries and frameworks for machine learning. The AE network is implemented with PyTorch [50].

The topology of the implemented AE is seen in fig. 10. Both the encoder and decoder are composed of 5 layers.

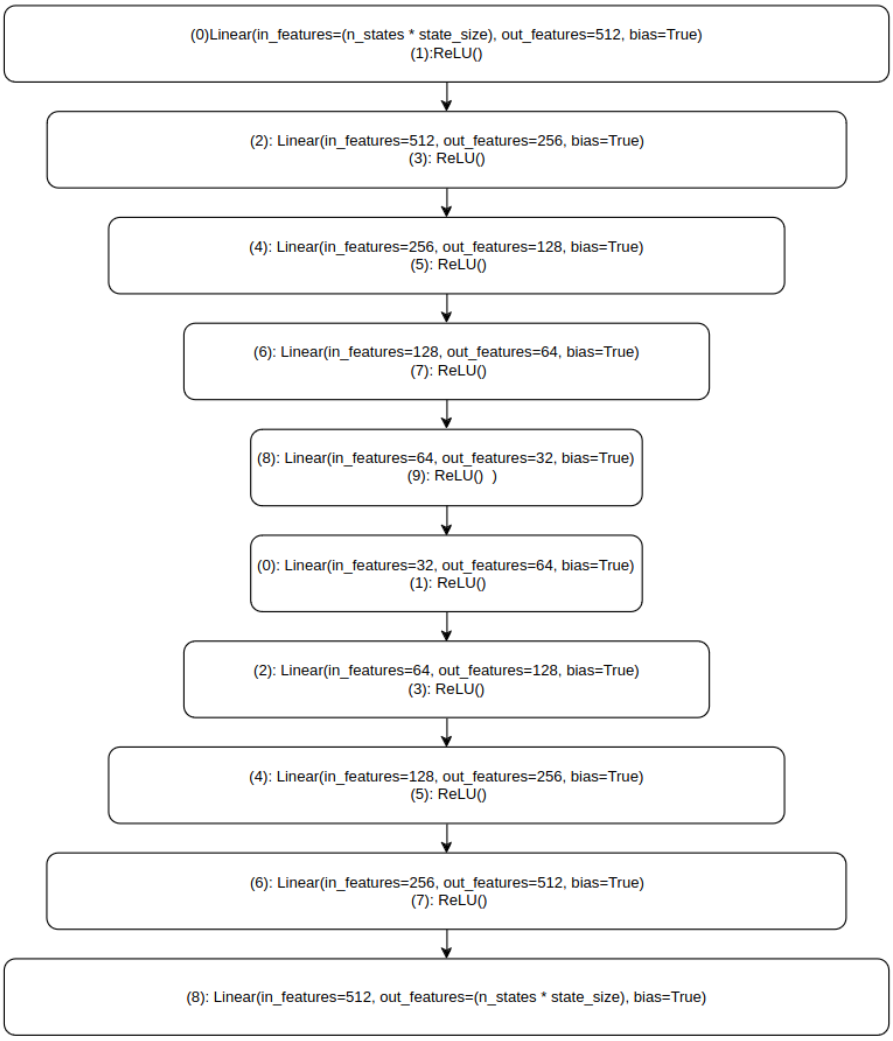


Figure 10: Topology of the Linear Autoencoder

The input size is determined based on the size of the state-window multiplied with the size of the states. The outer layers contain 512 neurons, and each subsequent layer is half the size of its predecessor, going down to 32 in the latent dimension. Currently, the state-window size is set to contain 32 states, the same size as the latent dimension, to ensure that information bottlenecking is present for all environments that have a state-space larger than 1.

The pseudocode for novel policy seeking is illustrated in algorithm 3. The initial policy is trained to optimize the fitness score, covering the environments' generic solution. Once the policy has been trained for the designated number of time steps, determined by the desired number of policy iterations, number of parallel environments, and steps per environment, an AE is trained using state sequences gathered with the models from the 10 final policy updates. The number of environments is set to equal the number of available cores, to enable effective

Algorithm 3 Finding Novel Policies with Autoencoders

Require: n-models, n-env, n-steps, iterations,
  for i in range(n-models) **do**
    $\boldsymbol{D} \leftarrow \{D_1, D_2, ..., D_n\}$ ▷ Load existing AEs
    Initialize n-env parallel environments
    for iterations $= 1, 2, ..., n$-iterations **do**
      Gather state-sequence τ using $\pi_{i_{\theta_{old}}}$ with n-steps per env
      if $\boldsymbol{D}$ **then**
        Calculate step-wise novelty-score r_{nov} based on $f_n(\min_{D \in \boldsymbol{D}} \text{MSE}(\tau, D(\tau)))$ ▷ eq. (15)
        Update rewards based on eq. (16)
        Update μ and σ for eq. (15)
        Update α based on eq. (17)
      end if
      Compute estimated advantages $\hat{A}$
      Update policy parameters θ with PPO
    end for
    Gather state-sequences from the 10 last policy updates of π_i
    Train AE D_i on collected state-sequences.

parallel computing. The number of iterations is estimated based on trial and error of the individual environments, where it is set high enough to devise an effective policy for the fitness based solution. The number of steps per environment is likewise estimated as the minimum steps required for potential environment completion. A list of the final used variables is seen in table 10.

Environment	Iterations	Steps per environment	Total interactions
BipedalWalker	200	1536	2457600
Swimmer	200	1024	1638400
UR5	250	512	1024000
Deceptive Maze	100	1024	819200
Open Maze	100	512	409600

Table 10: Environmental variables

A fixed-size time window containing 32 states are used as the input for the AE. The AEs are trained across 150 epochs with a batch size of 256. When collecting the states for training, the models are run non-deterministically to capture the general behavior of the policy. A total of $10 \cdot 32 \cdot 150 \cdot 256 = 1228800$ states are collected from the 10 models. Although such a high quantity of states might lead to overfitting, in the context of discovering novel policies, overfitting is advantageous. The objective of the AEs is to identify the degree of dissimilarity between the training data and newly encountered data, and not to discover some dynamical associations of the environment.

The calculations of the novelty score are performed within a custom environment wrapper. When the wrapper is initialized, it loads all previously trained AEs. Once a rolling state buffer reaches the input size of the AEs, all AEs attempt to recreate this state sequence. The lowest resulting reconstruction error is used to calculate the

novelty score, and is stored in a vector to calculate the mean and standard deviation for eq. (15) at the end of every rollout. A custom SB3 callback is created to handle logging to a Tensorboard log file, and to save various desired models. The callback also runs 8 evaluations of the current policy after every rollout, where the mean evaluation fitness is used to update α.

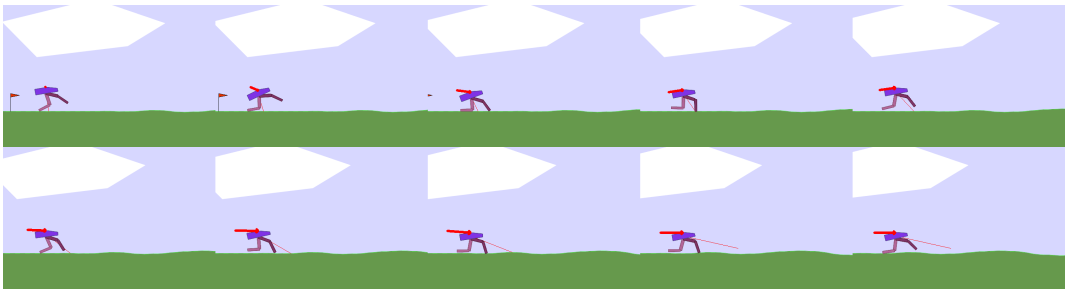
6.4 Resulting Novel Policies

Bipedal Walker

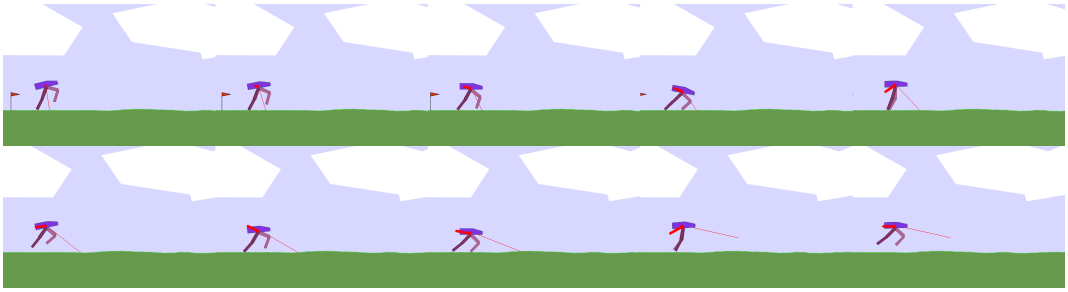
Novel policies of the walker sees no real advantage in terms of the resulting fitness score. The evaluation in this environment is therefore entirely subjective. A typical run usually finds at least two policies that primarily follow the novelty gradient, before finding upwards of nine distinct behaviors that also follow the fitness gradient. Training even more policies results in previously encountered behaviors reoccurring with more aggressive movement. The quality of the novel behaviors vary greatly between runs, but some are often capable of completing the 300 score challenge.

An attempt at visualizing some of the more interesting behaviors encountered across multiple runs is seen in fig. 11, where frames are captured at an interval of 13. The red arrow indicates the movement of the walker between the captured frames.

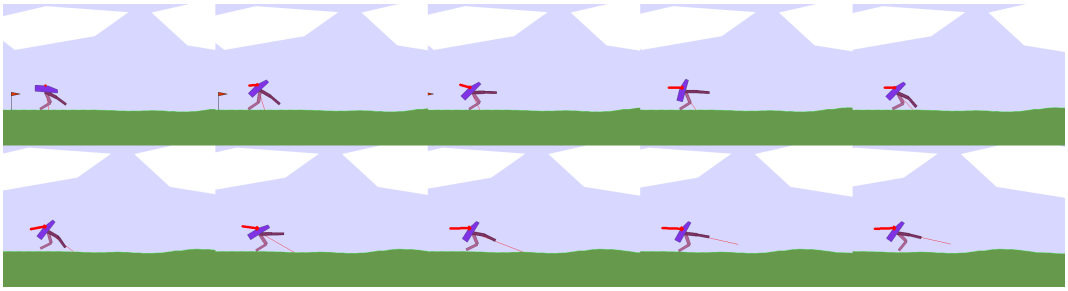
GIFs of a larger collection of behaviors can be found at https://github.com/BenjaminLonget/Novelty-Search-Master-Thesis/tree/main [4].



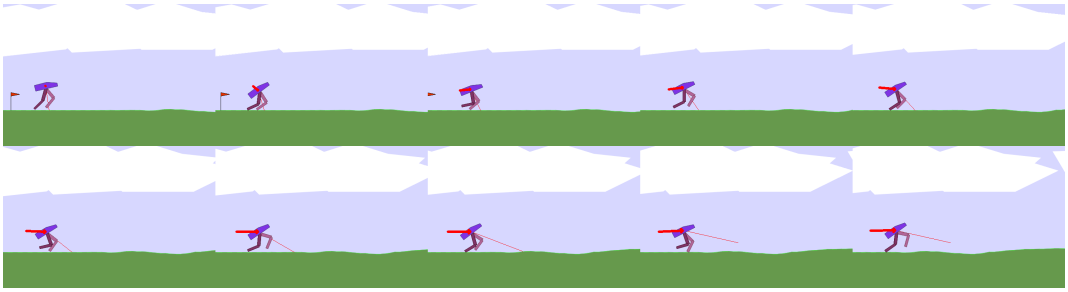
(a) Pure fitness



(b) Forward jumper



(c) One legged jumper



(d) Runner

Figure 11: Collection of novel walking strategies

Three policies are selected from a single run to compare the mean evaluation reward, the α values, and the achieved novelty score, between a fitness-based policy, a purely novel policy, and a novel policy of high quality.

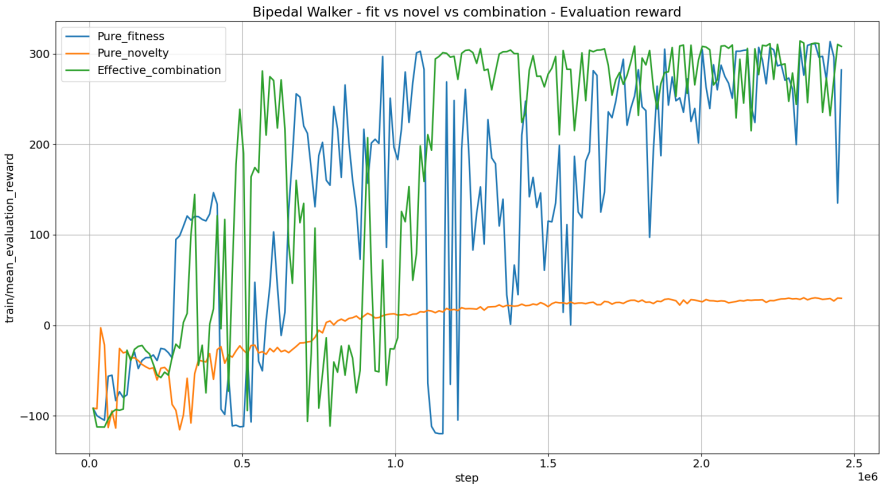


Figure 12: Mean evaluation rewards of a novel, fit, and combined policy

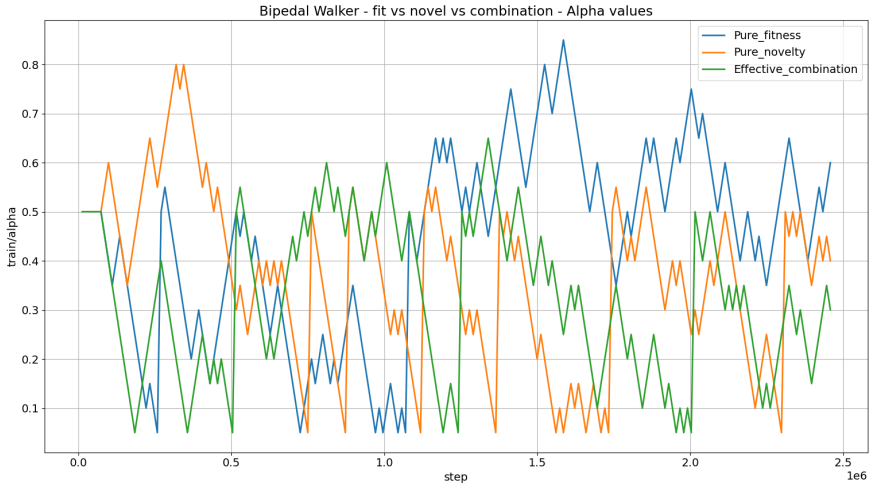


Figure 13: Graph showing the evolution of the α value for a novel, fit, and combined policy

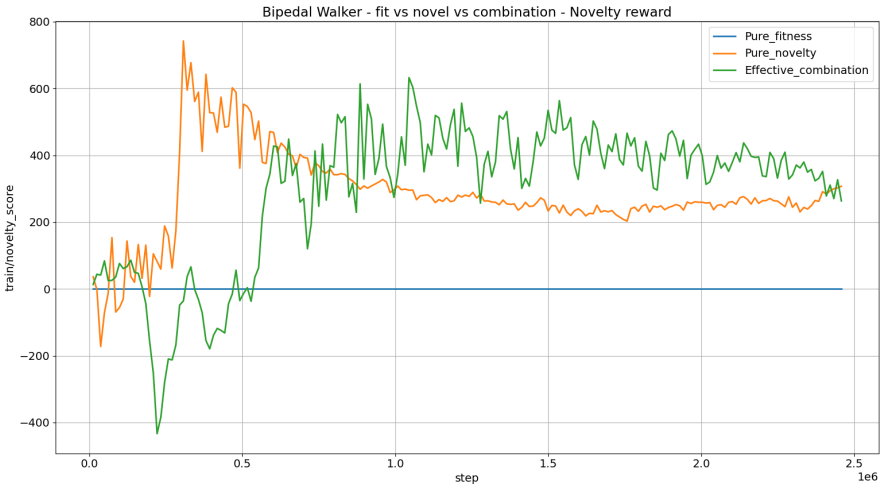


Figure 14: Mean novelty score of a novel, fit, and combined policy

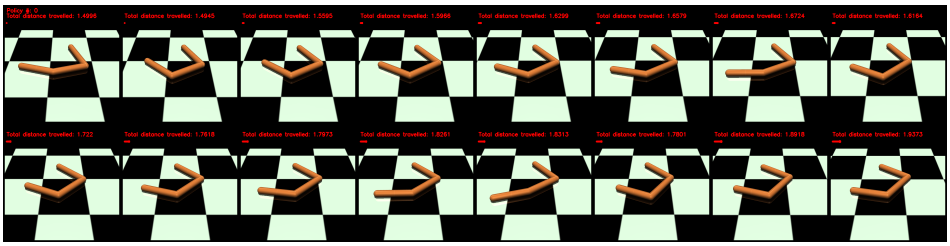
The low quality of the pure novel solution is seen in fig. 12. The progression of the weight as a result of the quality progression is seen in fig. 13. The pure novel solution in fig. 14 shows how the novelty score decays over time, when novelty is the only gradient of importance.

Swimmer

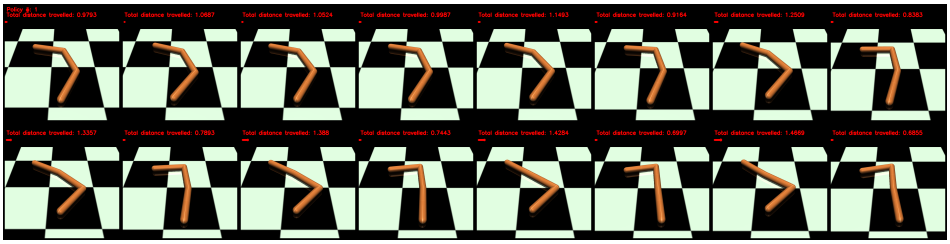
An attempt at visualizing the behavior of a typical run on the swimmer environment is seen in fig. 15, GIFs of the same run can be found at https://github.com/BenjaminLonget/Novelty-Search-Master-Thesis/tree/main [4]. For the visualization in fig. 15, the trained policy is first run for 100 steps to allow the swimmer to settle in the behavior specific orientation. Once settled, 16 frames are captured at a 10 frame interval. At the top of each frame, the total distance travelled is shown, as well as an arrow that indicates the exaggerated distance travelled along the x-axis since the initial frame was captured.

The fitness based solution fig. 15a consistently converges to a "frog-leg" type of movement. The only behavioral difference between fitness-based runs is which direction the swimmer turns before doing a similar movement.

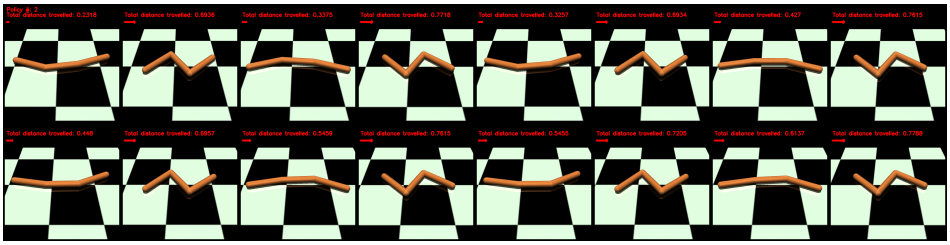
Both fig. 15b and fig. 15c follow a largely novel behavior, where forward motion is limited.



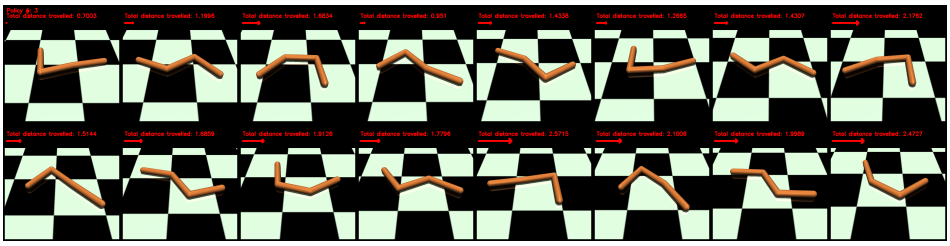
(a) Policy 0



(b) Policy 1



(c) Policy 2



(d) Policy 3

Figure 15: Resulting novel policies from 4 iterations of novelty seeking

An interesting behavior occurred in fig. 15d. Here, the model learned to swim in a manner reminiscent of how a 3-linked eel would move. This behavior is found in roughly 1 out of 4 runs, and is able to outperform the fitness based solution by a large margin.

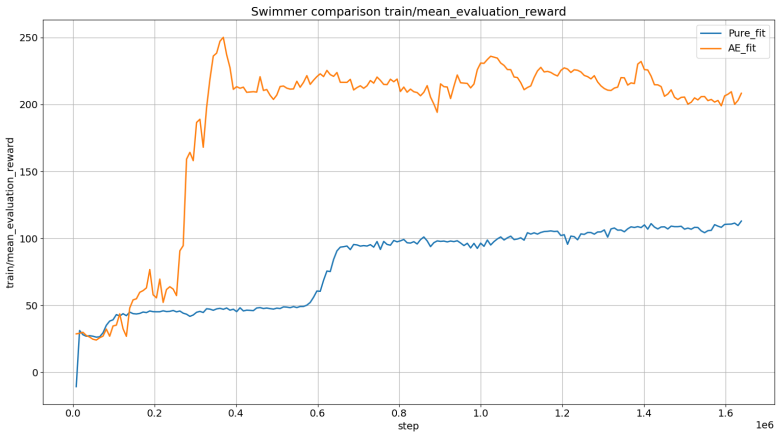


Figure 16: Comparison between a pure fitness based Swimmer and one found after training 4 novel policies.

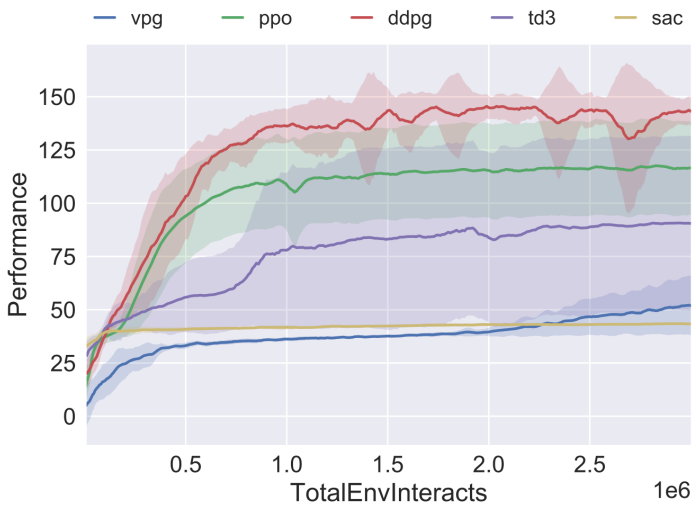


Figure 17: Benchmark comparisons of Swimmer performance for different algorithms from SpinningUp [51]

In fig. 16 a comparison between the model that occurs from optimizing for fitness and the one that learns to swim efficiently is seen. The mean episodic reward for the swimmer without novelty converges after 0.8m time steps to a score just above 100, where the novel solution converge after 0.5m time steps, achieving a score around 220.

The benchmark results from SpinningUp are shown in fig. 17. Their best results are with Deep Deterministic Policy Gradient, which converges at around 1m time steps to a score of 135.

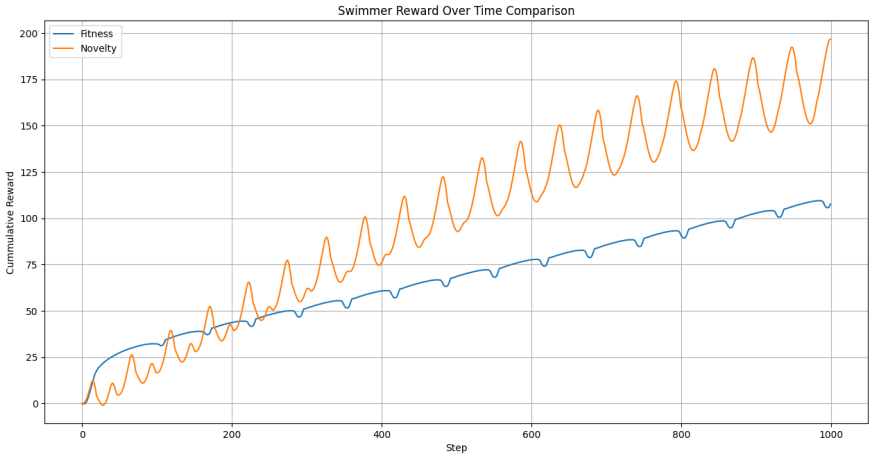


Figure 18: Comparison of the cumulative reward between the optimal behavior and the fitness-based model across a single environmental episode

As seen in fig. 18, comparing the fitness solution with the behavior that achieves a higher score, it is noted that the fitness behavior has an almost constant positive step-wise fitness score, where the novel behavior oscillates between positive and negative step-wise scores.

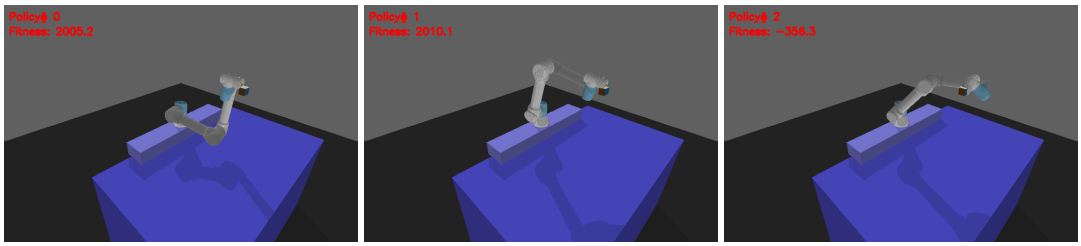
UR5

The UR5 is tested over five runs, each run training 8 policies. This is done to measure how many of the eight IK solutions are found on average.

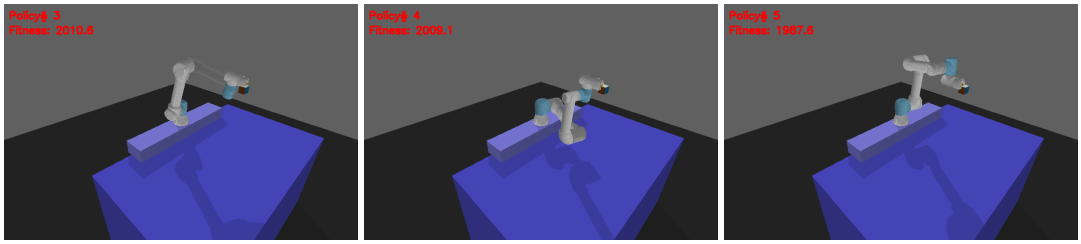
Test number	Unique configurations
1	5/8
2	4/8
3	4/8
4	4/8
5	2/8
Average	3.8/8

Table 11: UR5 test results

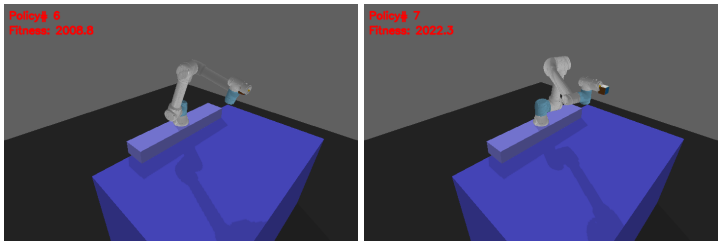
The final configurations from the best run are seen in fig. 19. Here, the found configuration is defined from the position of the shoulder (S), elbow (E), and wrist (W), totaling eight distinct possible configurations. Novelty is still somewhat notable for the policies that obtain similar end positions. Here, novelty is achieved with the trajectory the model takes before hitting the final configuration. The final average number of configurations found are seen in table 11.



(a) S - right, E - down, W - up (b) S - right, E - up, W - up (c) Novelty



(d) S - right, E - up, W - up (e) S - left, E - down, W - up (f) S - left, E - down, W - down



(g) S - right, E - up, W - up (h) S - left, E - up, W - up

Figure 19: Test run that found five of the eight configurations

Deceptive Maze

To test the effectiveness of the novel policy-seeking algorithm on the Deceptive Maze, five runs are conducted, each training five policies. Ideally, only 3 policies are required to find both paths, but more are trained to allow for entirely novel behaviors. For a behavior to be considered successful, it needs to reach the goal while taking a unique path from the starting area.

Test number	Unique successful paths
1	1/2
2	2/2
3	2/2
4	2/2
5	1/2
Average	1.6/2

Table 12: Deceptive Maze results

In fig. 20, only one policy was able to successfully reach the goal, as the second novel policy were trapped in the secondary optimum. This result show the difficulty of the maze even though it is simple. An example of an

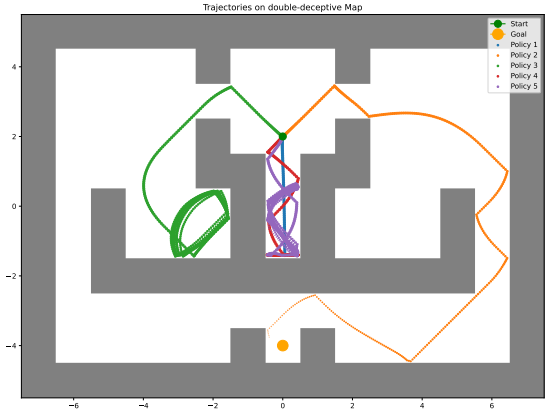


Figure 20: Result showing the effect of the secondary local optima

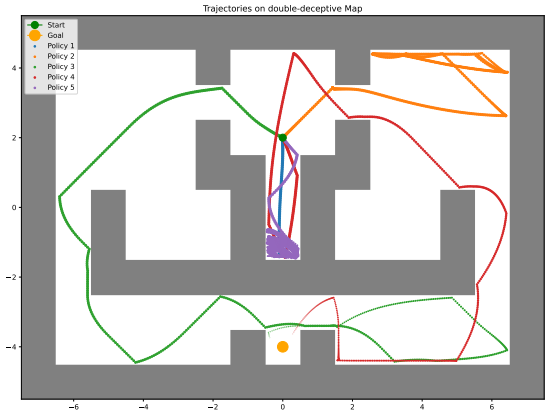


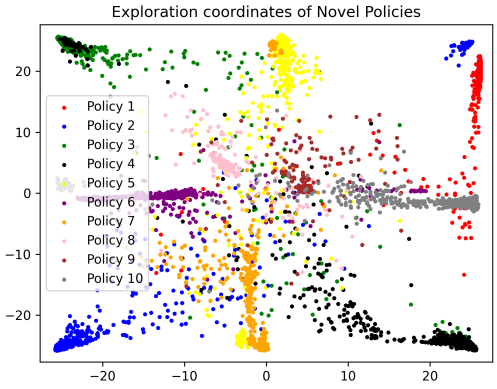
Figure 21: Test result showing both paths are found, even when policy number 2 goes for pure novelty

entirely novel behavior is seen in fig. 21, where the first novel behavior rolled around in the upper corner. The final results are seen in table 12, showing the average number of achieved unique trajectories being 1.6.

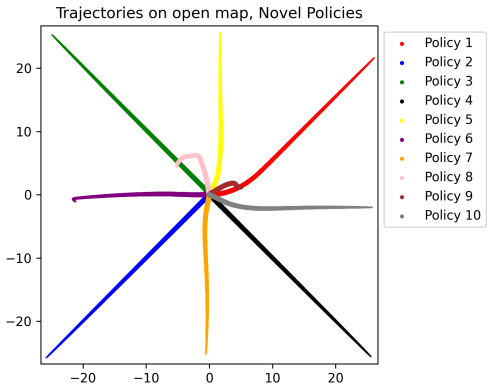
Open Maze

The open maze is used to visualize the novelty in a two-dimensional space. Three different tests are conducted, each composed of five runs where 10 policies are trained. The first test is to show the effect of AE-novelty on its own, where only the first trained policy receives a fitness score for moving right. The second test show the result without novelty, where intermediate policies are used to generate the trajectories, to show the evolution of a generic solution. The final test shows the results of combining fitness and novelty. To obtain an objective

evaluation of the results, the Fréchet distance is calculated between all novel trajectories, and between the intermediate trajectories for the pure fitness policies. A threshold of 2.5 is applied to these distances to determine the amount of unique trajectories.

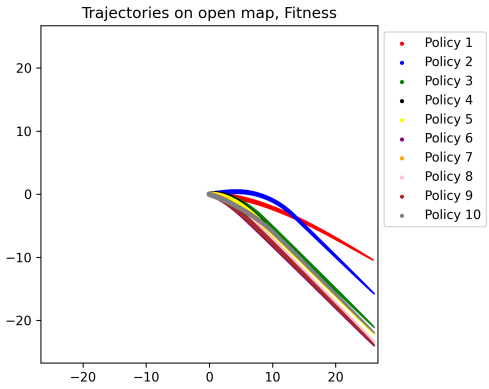


(a) Scatter plot of exploration when searching for novel policies

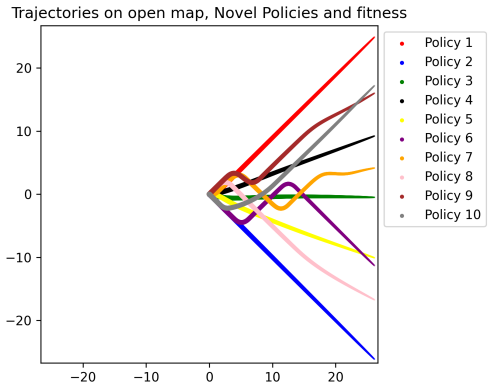


(b) Final trajectories of novel policies

Figure 22: Resulting final positions and trajectories when seeking novel policies



(a) Intermediate trajectories of a fitness based policy



(b) Resulting policies from combining fitness and novelty

Figure 23: Results of optimizing for fitness while seeking novel policies

Test	Average unique trajectories
Fitness	3.0/10
AE novelty	9.6/10
AE novelty and fitness	9.6/10

Table 13: Unique trajectories according to Fréchet with a threshold of 2.5

By looking at the resulting trajectories in the order they appear, we can see how new policies are trained to be as different as all previous policies at the same time. For example, in fig. 22b, the first four policies maximize

acceleration in different directions, covering the corners of the map. Subsequent policies then either cover the space between these or does some novel movement around the center of the map. The final result, according to a Fréchet distance threshold of 2.5, is seen in table 13.

6.5 Thoughts on Resulting Novel Policies

Overall, the performance of novel policy seeking using linear AE appears promising, where newly discovered solutions often exhibit significantly different behavior compared to their predecessors.

Novel policies for the swimmer environment tend to have mixed results. Besides the occasional high performing behavior, most other novel behaviors often fall short of the generic fitness based solution.

While only somewhat successful, finding multiple IK configurations for the UR5 is quite notable, as the algorithm is not specifically designed with this task in mind, and no changes are made to the algorithm between the environments.

While both solutions are found in the Deceptive Maze for most of the runs, room for improvement still persists. One improvement could be to avoid having to train an entire policy for fitness, just for it to be trapped in the local optimum. Another obvious improvement would be to find both solutions consistently.

The open maze shows how the algorithm is able to maximize the difference in trajectories between policies, where every new policy is as distinct as possible compared to all the previous policies. When combined with a simple fitness function, this ability is maintained.

A notable drawback, for most of the environments, is the necessity of initially training for pure fitness before encountering novel solutions. To address this limitation, the next step involves integrating on-policy novelty, a strategy aimed at promoting novelty exploration within the training process itself.

7 Improving Exploration with LSTM-Autoencoders

In the previous section, we determined the novelty of exploration steps by measuring the reconstruction error of sequential state data applied to AEs trained on previous policies. While this method effectively found novel behaviors, it required training an entire policy for fitness optimization before novelty could be introduced. Furthermore, since the behavior is described with fixed size time windows, the AEs would only capture the novelty of the immediate previous behavior. This could hinder the discovery of novelty if behavioral patterns are found at larger temporal differences.

To maintain possible connections between states that are far apart, Recurrent Neural Networks (RNNs) could be utilized [52]. Given that the state data collected is structured in a time-series manner, RNNs offer the capability

to maintain possible connections between sequential states. This ability to capture sequential dependencies makes them particularly suitable for modeling behavior over time. However, standard RNNs still struggle to maintain information across larger temporal spans due to issues such as the vanishing gradient problem.

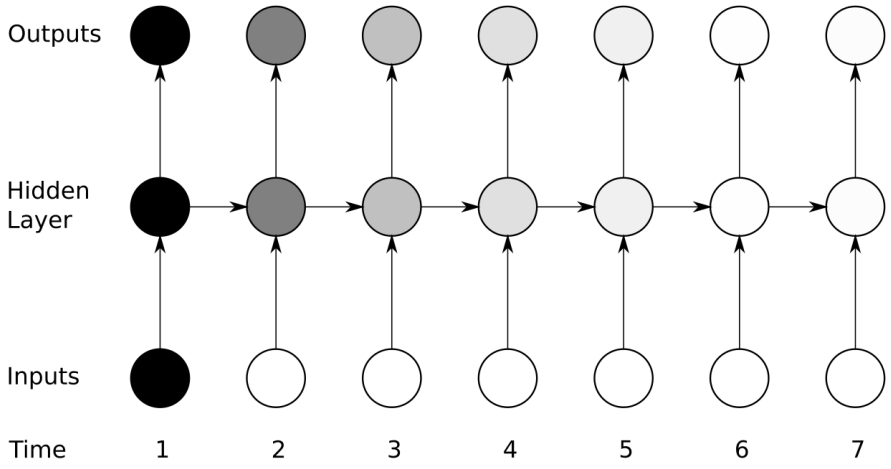


Figure 24: Visualization of the vanishing gradient problem from [53]. The shading indicates the sensitivity to the first input over time

In [54], the vanishing gradient problem in RNNs is discussed. Depicted in fig. 24, this problem describes how the effect of an input on the hidden layers, and subsequently on the output, either decays or explodes exponentially over time when recurrency is introduced. As a result, RNNs struggle to maintain context between inputs that are distant in time, limiting their ability to retain long-term dependencies. This leads to RNNs only being suitable when short-term memory is necessary, making them less effective for processing sequential data where earlier inputs might impact later ones.

Long Short-Term Memory (LSTM) [53] is one of numerous methods devised to circumvent this issue. The idea of LSTM is to replace the neurons in the hidden layers with memory blocks, known as LSTM blocks. Each LSTM block consists of several components:

- Input activation: Takes the current input along with the previous output to generate the value for the memory cell.
- Input gate: Responsible for determining the relevance or importance of the current input.
- Constant Error Carousel (CEC): An internal *memory-cell* with a recurrent connection. If no changes are seen at its input and the forget gate remains active, the CEC will contain a constant looping value. This cell is what constitutes the long-term memory of the block.
- Forget gate: Introduced in [55], this component evaluates the current input and previous output to decide

whether to reset the memory. A lower output from the forget gate indicates a reduced retention of the information looping in the CEC.

- Output gate: Regulates the output of the LSTM block based on the input and the previous output.

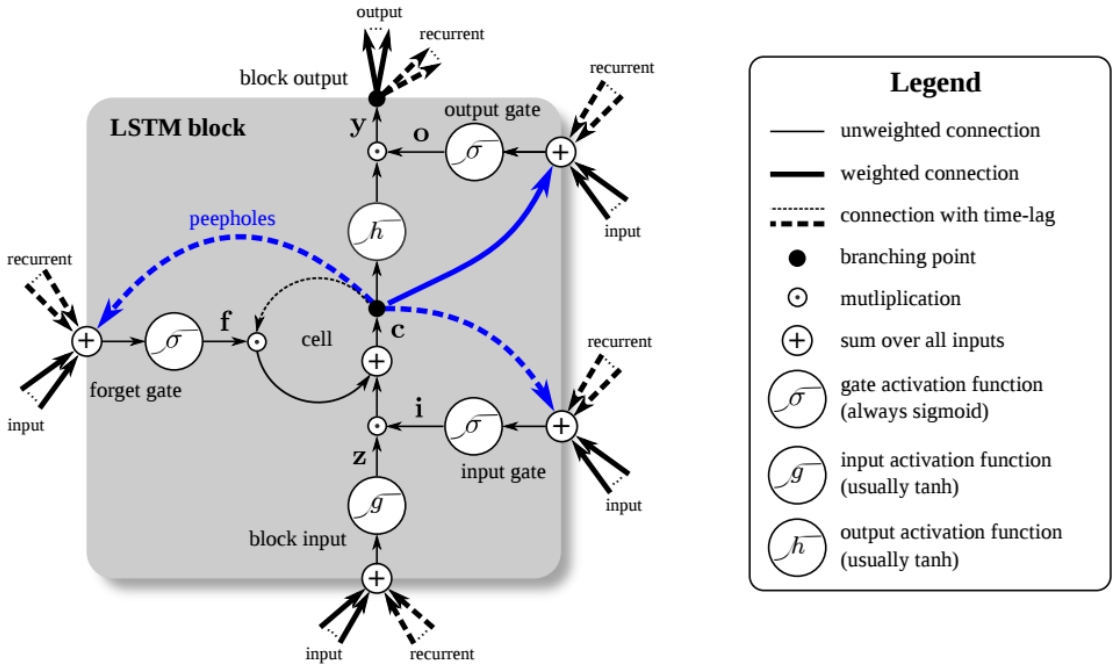


Figure 25: Image from [56] shows a single LSTM-block with a forget-gate. In this figure, the CEC is seen as the looping connection named "cell". According to [57], the peephole connections rarely increase performance and are therefore often omitted

$$
\begin{aligned}
i_t &= \sigma(W_{ii} + b_{i_i} + W_{yi} y_{t-1} + b_{yi}) && \text{input gate} \\
z_t &= \tanh(W_{iz} x_t + b_{iz} + W_{yz} y_{t-1} + b_{yz}) && \text{input activation} \\
f_t &= \sigma(W_{if} + b_{if} + W_{yf} y_{t-1} + b_{yf}) && \text{forget gate} \\
o_t &= \sigma(W_{io} + b_{io} + W_{yo} y_{t-1} + b_{yo}) && \text{output gate} \\
c_t &= f_t \cdot c_{t-1} + i_t \cdot z_t && \text{cell state} \\
y_t &= o_t \cdot \tanh(c_t) && \text{output activation}
\end{aligned}
\tag{18}
$$

An overview of the LSTM cell is seen in fig. 25, and the corresponding equations are seen in eq. (18). For the equations, W indicates the different weights, and b the different biases. All control gates are activated with the sigmoid function to give a value of relevance between 0 and 1, and the input and output is activated with the hyperbolic tangent function.

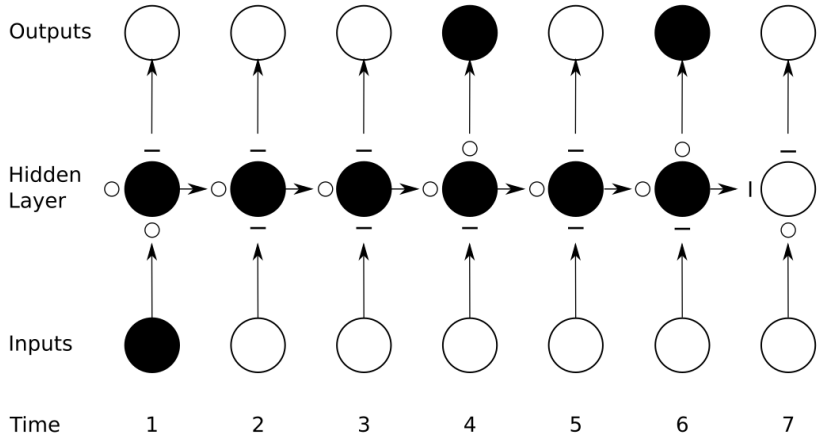


Figure 26: Image from [53]
showing the LSTM cell attaining information over time, circumventing the vanishing gradient.

A visualization of the hidden cell attaining information over time is seen in fig. 26. Here the state of the different gates are denoted with $\circ$, denoting the gate is fully open, and $-$, denoting the gate is closed. The CEC state is maintained from the input at the first time-step, up until the forget gate is closed.

Since LSTM networks are adept at capturing long range dependencies, they are used in a variety of tasks concerning time-series data.

In [58], sentiment and question classification are achieved by combining Convolutional Neural Networks (CNN) with LSTM. Here, the CNN extract sequences of sentence features, which the LSTM use to generate a sentence representation. Text prediction and generation is done in [59], by applying LSTM as an AE. End-to-end speech recognition from acoustics is achieved in [60] by training an LSTM network with connectionist temporal classification loss.

A multitude of research papers show promising results for anomaly detection with LSTM-Autoencoders (LSTM-AE), many of which are compared in [61]. One example is seen in [62], where LSTM-AE are used for anomaly detection in time series data. By training the LSTM-AE to reconstruct non-anomalous data and subsequently modelling the reconstruction error as a multivariate Gaussian, the likelihood of new sequential data coming from the same distribution is estimated. An anomaly is said to be detected when this likelihood is less than some threshold.

Anomaly detection is closely correlated with novelty search when applied to data that describes behavior. In both cases, the objective is to identify unusual or unexpected patterns within a given dataset. Furthermore, the aforementioned LSTM-AE used for anomaly detection uses the reconstruction error to determine anomalies, much like the linear AEs from section 6 used this error to determine the level of novelty.

We predict that training an LSTM-AE on the encountered state-trajectories, while using the reconstruction error to score new state transitions, will serve as an effective proxy for determining the novelty score in an on-policy manner.

7.1 Implementation details

The LSTM module from PyTorch [63] is used to create the LSTM-AE class, based on the all-round implementation found at [64].

Both decoder and encoder consist of 8 LSTM modules. Additionally, the decoder incorporates a final linear layer. When the LSTM-AE receives a sequence, the encoder traverses the entire sequence, appending all outputs to a vector. Each element in this vector contains the encoded value up to that specific time-step. The final value in the vector consequently contains the entire sequence encoded into a fixed size feature, which constitutes the information bottleneck required by undercomplete AEs as mentioned in section 6. The decoder attempts to reconstruct the entire input sequence based entirely on this single encoded feature.

Instead of modifying the reward in an environment wrapper, as done in section 6.3, everything is handled within the custom callback, where a simple method is constructed to detect the end of an episode, as the rollout buffer contains the trajectories from all parallel environments at this time.

A method is created in the LSTM-AE class that preprocess the data, before attempting the reconstruction. This method returns the absolute error for all the reconstructed states, which is used to modify the corresponding reward in the rollout buffer. The length of the sequences to be reconstructed is set to the same value as the length of an environmental episode. The mean absolute error for every reconstructed step is normalized based on the largest reconstruction error of the rollout. The novelty score is multiplied with a decaying epsilon, before being applied linearly as a $50/50$ weighted sum of rewards. After applying the novelty score, the LSTM-AE is trained on the same data, enabling it to recognize these sequences, and forcing the underlying algorithm to continually find behavior that is different from that of the current policy.

7.2 Results

Since the LSTM-AE is meant to guide the search for novel behavior against the current policy, 10 intermediate policies are saved during training on the maze maps to show the progression of the achieved trajectories.

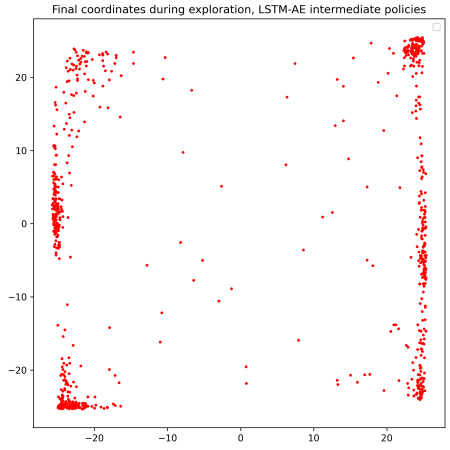
First, the algorithm is tested entirely with LSTM-AE on the open maze, before the simple fitness function is introduced.

The Fréchet distance is calculated between subsequent intermediate policies to determine whether LSTM novelty helps diverge the policy across the 2D behavior space. Since the on-policy novelty search tries to

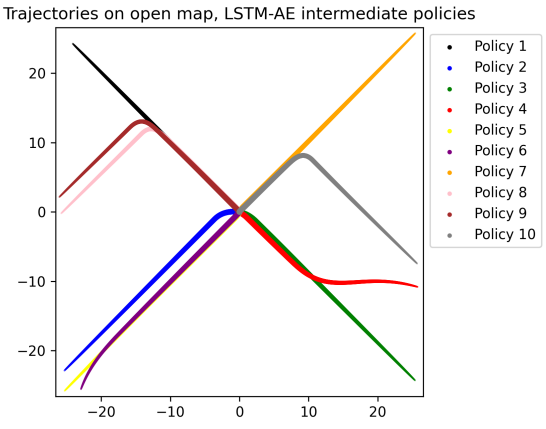
be novel to the immediate previous update steps, the distance is calculated between subsequent policies instead of between all other intermediate policies at once. For example, policy number 2 is only compared against policy 1, policy 3 against policy 2, et cetera.

Finally, the LSTM-AE is used along with the fitness score in the Deceptive Maze environment, counting the number of successful runs. Again, 5 runs of each experiment are conducted, and a distance threshold of 2.5 is used to determine novel trajectories in the open maze variant.

Open Maze

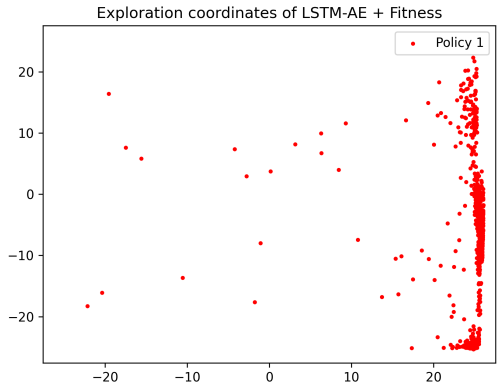


(a) Scatter plot from exploration with LSTM

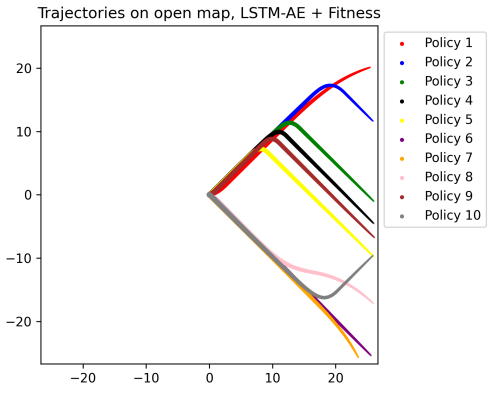


(b) Trajectory of 10 intermediate policies during LSTM exploration

Figure 27: Final positions and intermediate policy trajectories when training entirely with the LSTM-AE



(a) Scatter plot from exploration with LSTM and fitness



(b) Trajectory of 10 intermediate policies during LSTM and fitness exploration

Figure 28: Example of resulting final positions and intermediate policy trajectories when training with both the LSTM-AE and fitness

Test	Average unique trajectories
Fitness	3.0/10
LSTM-AE novelty	8.6/10
LSTM-AE novelty and fitness	5.4/10

Table 14: Unique intermediate trajectories according to Fréchet with a threshold of 2.5

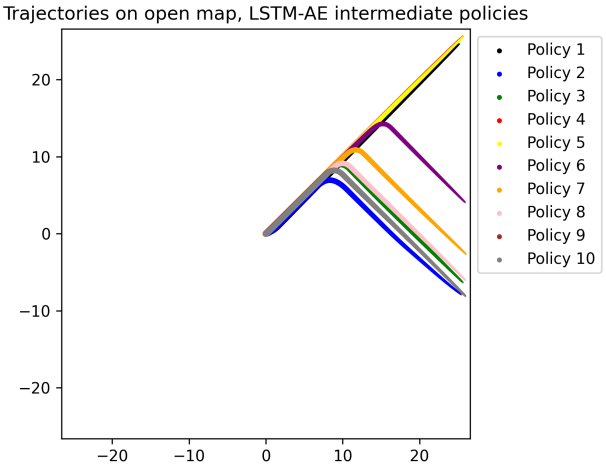
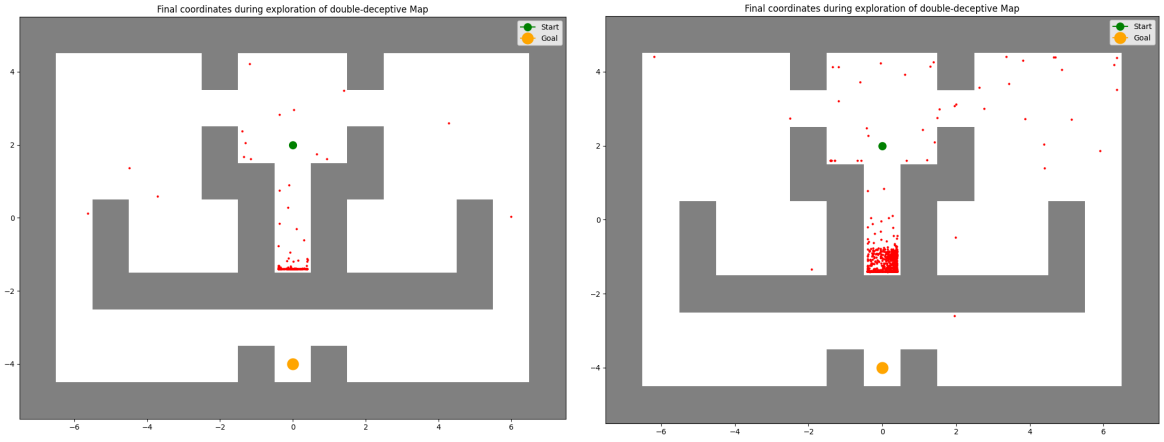


Figure 29: Intermediate policies of a run that achieved just 6 unique trajectories

The final average number of unique trajectories are seen in table 14. An example of the runs that cause the low number of average unique trajectories when combined with fitness is seen in fig. 29. Upon closer inspection, it is noted that the trajectories exhibit oscillations over time. This observation is further supported by fig. 27b,

where similarities between policy 2, and 5 are evident, while the policies in between display distinct differences. This observation hints at the catastrophic forgetting phenomenon, a phenomenon where the network gradually overwrites the weights associated with previously learned behaviors. Catastrophic forgetting is plausible, as the LSTM-AE is trained entirely on newly gathered data. One possible way to counteract this issue could involve preserving a subset of previous trajectories and include these as additional batches when training on new data.

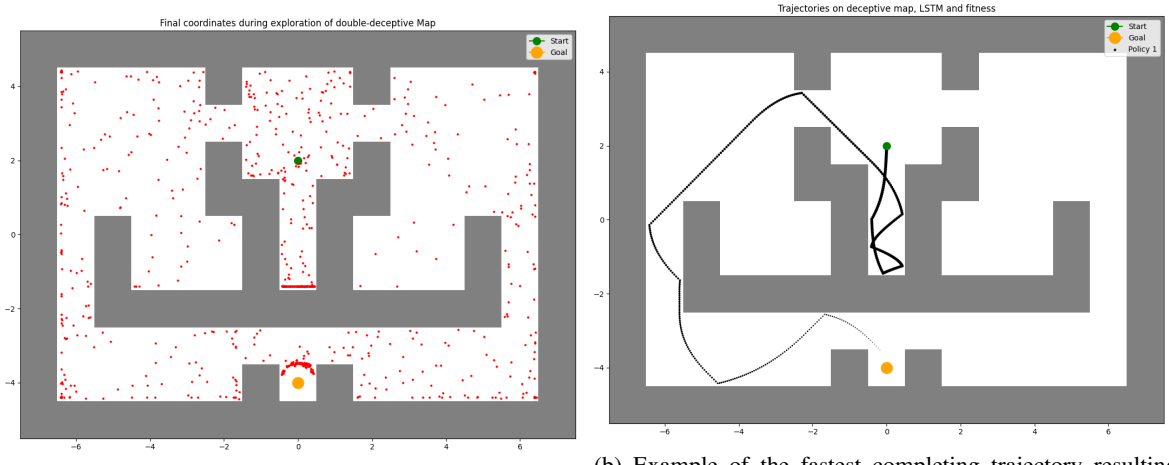
Deceptive Maze



(a) Scatter plot from pure fitness exploration

(b) Scatter plot from fitness with additional reward noise

Figure 30: Resulting final positions during exploration with fitness



(a) Scatter plot from LSTM and fitness exploration

(b) Example of the fastest completing trajectory resulting from successful LSTM-AE exploration

Figure 31: Resulting final positions from exploration with LSTM-AE and the resulting successful trajectory

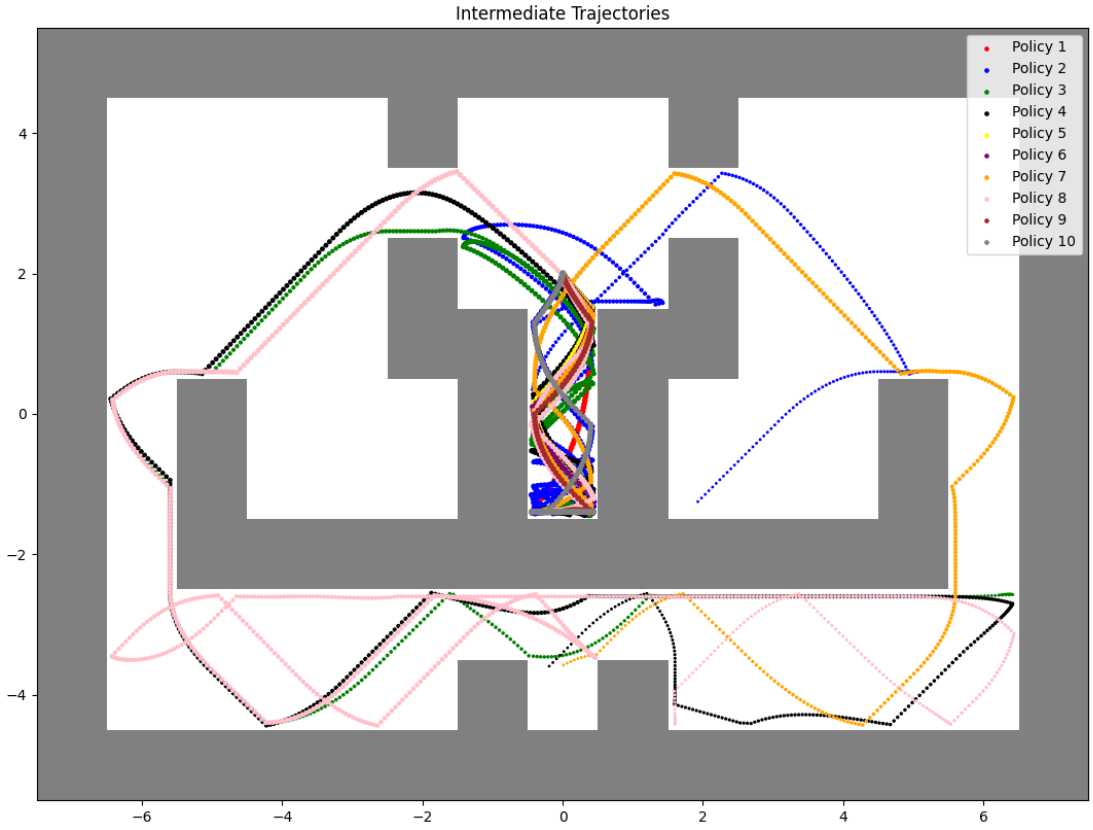


Figure 32: 10 intermediate trajectories found when exploring with LSTM-AE

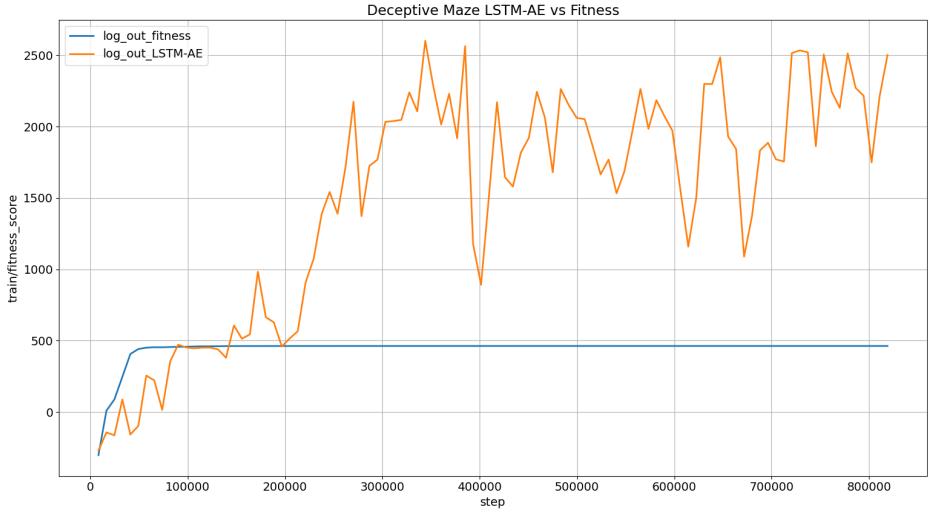


Figure 33: Fitness score when exploring with and without LSTM-AE

LSTM-AE consistently achieved map completion in all attempts, often obtaining a valid policy after just 20 iterations. In contrast, when running without novelty search, the algorithm converges completely to the immediate local optima within 10 iterations.

Multiple intermediate policies that complete the map are saved for comparisons, including the fastest, highest scoring, final successful, and the final policy. Upon inspection, it was observed that LSTM-AE typically discovers both separate paths through the maze. To visualize this, 10 intermediate trajectories are plotted in fig. 32, showcasing the oscillatory nature of the algorithm.

To assess whether the success was merely a consequence of the LSTM-AE introducing stochastic behavior by adding large amounts of noise to the reward, a test was conducted where a zero-mean Gaussian with a standard deviation of 10 was applied to the fitness score. The result of this test is illustrated in fig. 30. While the high Gaussian noise did show a slight increase in the spread of the final coordinates during exploration, the impact was far less pronounced when compared to LSTM-AE as seen in fig. 31.

The difference in mean fitness score between being stuck in the local optima, and effectively exploring the map, is seen in fig. 33. When using novelty search, the model consistently achieves a higher score than the models that are stuck in the local optimum, indicating that the goal is reached in at least some parallel environments.

8 Combination of On-policy and Between-policy Novelty Search

In an attempt to discover both unique paths in the deceptive maze, a combination of both novelty search algorithms is employed. On-policy novelty should allow the agent to escape the local optima, and the off-policy novelty should allow the agent to discover the secondary path through the maze.

8.1 Implementation details

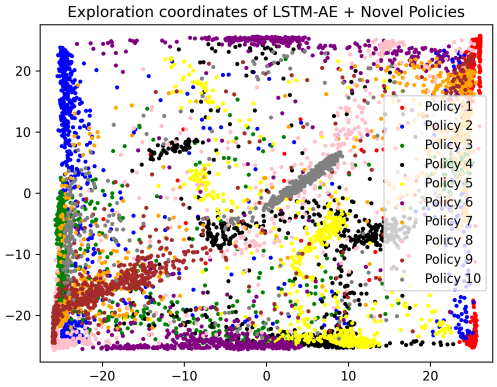
Having the linear AEs implemented in the environment wrapper, and the LSTM-AE in the custom callback results in a modular structure. Minor adjustments are made to variable names, and additional initialization variables are added to activate the different types of novelty.

The weighted sum of rewards is based on $1/3$ novelty score from the LSTM-AE, and $2/3$ of the α-regulated sum of fitness and off-policy novelty.

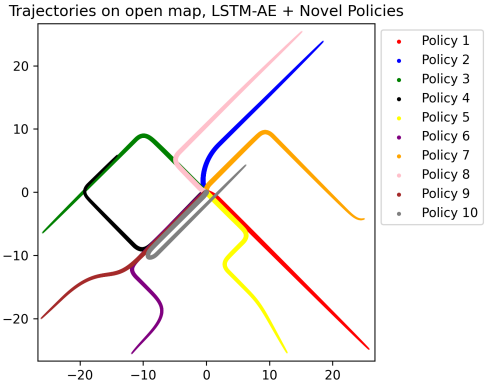
Again, 5 tests are conducted on the open map, with and without fitness, as well as on the deceptive map.

8.2 Results

Open Maze



(a) Scatter plot of exploration with the combined algorithm



(b) Resulting novel trajectories from combining AEs with LSTM-AE

Figure 34: Open map with both types of novelty

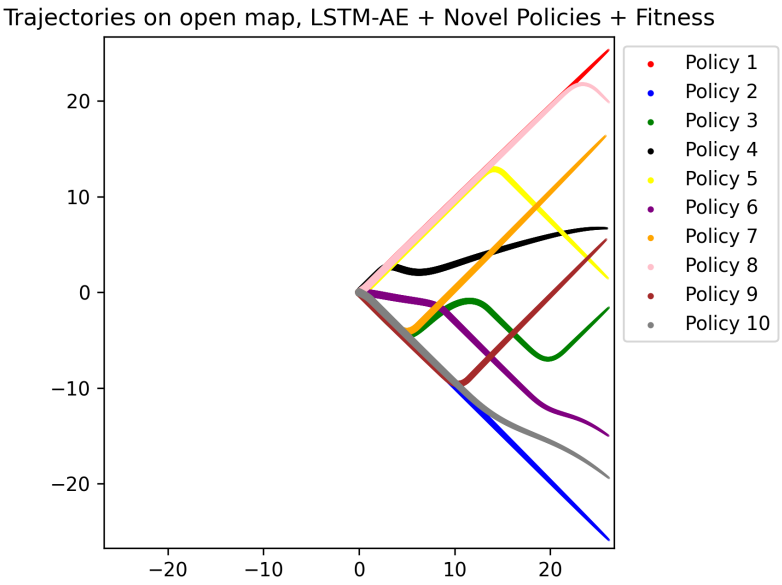


Figure 35: Resulting trajectories from combining AEs, LSTM-AE, and fitness

Test	Average unique trajectories
Fitness	3.0/10
Linear AE	9.6/10
Linear AE and fitness	9.6/10
LSTM-AE	8.6/10
LSTM-AE and fitness	5.4/10
Linear AE and LSTM-AE	9.8/10
Linear AE, LSTM-AE, and fitness	8.8/10

Table 15: Final results of all tests on the open maze

Examining the coordinates and trajectories depicted in fig. 34, it is apparent that the influence of both types of novelty remains noticeable. The linear AEs contribute to the dispersion of trajectories, while the LSTM-AE induces the sharp turns observed in most of the trajectories.

The final average number of unique trajectories for all tests on the open map is seen in table 15.

Deceptive Maze

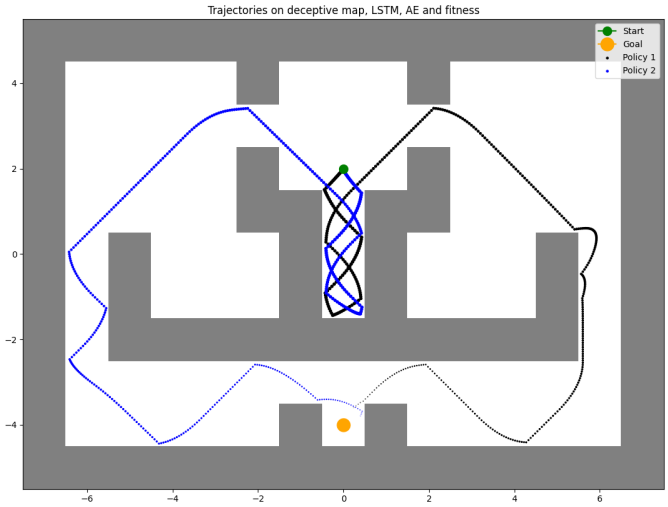


Figure 36: Typical result of the combined algorithm on the deceptive map

(a) Highest scoring policies

(b) Final successful policies

Figure 37: Different policies of the same run

All runs found both paths to the goal with just 2 policies using the combined algorithm, resulting in trajectories similar to fig. 36. Another depiction of LSTM-AE finding both paths while training a single policy, as mentioned in section 7.2, is seen in fig. 37.

The remaining environments

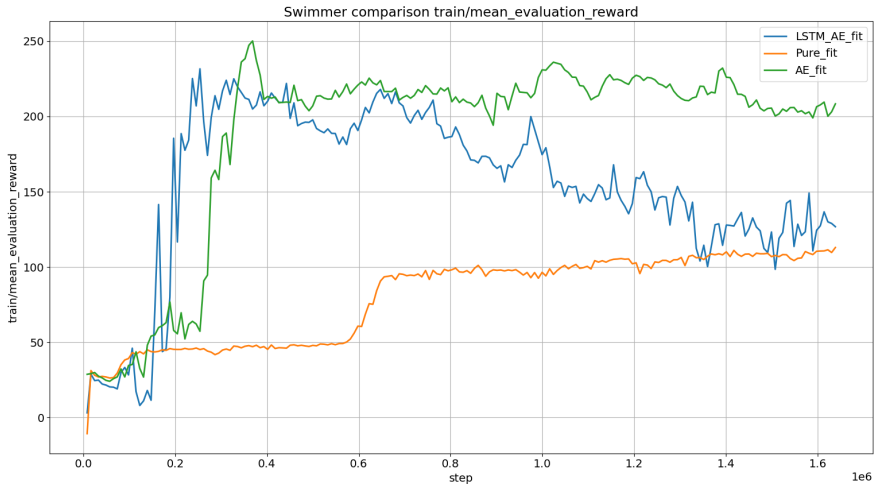


Figure 38: Evaluation scores of Swimmer. Benchmark vs. linear AE vs. Combined

Most of the other environments see no benefit from the addition of the LSTM-AE. The Swimmer did manage to find the specific behavior that results in a good score. However, as seen in fig. 38, this behavior was short-lived,

as the LSTM-AE forced the policy to diverge from the learned behavior. A series of GIFs from a combined run of the Swimmer and the Walker can be found at https://github.com/BenjaminLonget/Novelty-Search-Master-Thesis/tree/main [4].

9 Discussion

In this section, we reflect upon the strengths, weaknesses, and potential future work for the novel policy-seeking algorithm using linear AEs, and the effectiveness of LSTM-AEs regarding exploration.

9.1 Novel Policy-Seeking with Linear Autoencoders

General Considerations

Overall, using linear AEs to find novel policies demonstrated some promising results, where most policies resulted in clearly novel behavior.

One drawback of this method is the need to train and load separate AEs for every behavior, which increased training time considerably as training progressed. Using the UR5 tests as an example, training the first policy entirely for fitness took about 60 minutes on a GTX 970 when running eight environments in parallel. The total time for training all eight policies was above 11.5 hours, meaning 57.5 hours were spent running the five usable tests for this single environment, not accounting for the additional time spent collecting states and training the AEs.

Walker Environment

As indicated by fig. 11 and the GIFs on the repository [4], the linear AEs proved highly effective in the BipedalWalker environment. Even after extensive testing during the implementation and testing phase, the algorithm was still able to discover new distinct behaviors. This was not the case for the other environments, indicating that the algorithm might be better suited for environments where the behavioral space is vast. It might also be due to this environment being the most complex of all, both in terms of the size of the state space, and the complexity required to solve the task due to the stochastic elements of the environment. The stochastic nature of the environment also meant that the final policy trajectories were likely to be more varied than the non-stochastic environments, which could result in the AEs describing the behavior in an even more general way.

Swimmer Environment

The performance in the Swimmer environment was mixed. On one hand, the novel policy-seeking algorithm occasionally discovered a specific behavior that significantly outperformed the benchmark, both in terms of the final score and the rate of convergence. As seen in fig. 16 and fig. 17, the best result from the benchmark converged after 1m time steps to a score between 125 and 150, and the optimal novel behavior converged after 0.5m time steps to a score around 220. This showcased the potential for novelty search in general, where the obvious path of learning might not yield the ideal outcome. As of May 30, 2024, the best entry on the Gymnasium leaderboard [37] achieved a score of 205 after 55 policy iterations. If we ignore the fact that multiple previous policies were required, we achieved an even higher score in less policy iterations, beating the current world record.

For many runs, however, most of the discovered behaviors failed to follow the fitness gradient at all. A reason for the primarily novel behaviors could be due to the generally low step-wise fitness score obtainable in the environment. This could potentially be remedied by normalizing the fitness score in the environmental wrapper, which would further generalize the algorithm across environments. Early attempts at a generic linear normalization showed little to no promise, however, but more complex methods of normalization could be worth investigating.

UR5 Environment

Despite not being designed for this specific task, the novel policy-seeking algorithm managed to find multiple IK solutions. If more time was spent working with this environment, the algorithm could potentially be designed as a complete numerical IK solver, removing the need to calculate the analytical IK solution.

The reason for the mediocre results seen in table 11 might have to do with the high reward applied when obtaining the desired EE position and orientation. This reward was set very high to enable the detection of success in the log files, but might inadvertently guide the policies to the same direct behavior. Testing for a better suited fitness function would have been optimal, but were omitted due to the time required to run a single test.

The most significant issue regarding the UR5 environment was the time spent searching for a functional Gym-enabled version. Although many public repositories contain UR environments, few utilize actual joint control. Instead, they often implemented or mimicked inverse kinematics control, with the action and state spaces defined by the EE position and orientation. In the end, more time was spent searching for and attempting to fix various robotic arm environments than it would have taken to create one from scratch.

Open Maze Environment

The trajectories found in the Open Maze environment aligned well with the expected results, especially when viewing the trajectories in the sequence they are trained. As indicated by fig. 22b, if the first policy goes right and slightly up, the next policy went left and down, the third left and up, et cetera. When the corners were covered, the trajectories that maximized just a single directional velocity usually followed, sometimes preceded by a single trajectory that hovers around the starting area. The same effect is seen in fig. 23b, where the first policy went up, the second went down, and the third went almost perfectly between the two others. Once the straight trajectories covered most of the high quality solution space, more advanced trajectories occurred, aligning well with the statement from [2] about novelty search ascending the ladder of behavioral complexity.

In two separate runs, there were instances where two trajectories were measured to be identical. This was quite an unexpected result, especially in the test where the novelty score was the only response sent to the underlying algorithm. This could be attributed to the fact that the variance of the reconstruction error becomes smaller as more AEs are introduced, making the slope for the novelty score very steep. This would introduce uncertainties when calculating the score, as the region for the gradient becomes very narrow. Upon further inspection of the unused fitness score in the log files, we noticed that it changed a great deal towards the end of training one of these policies, indicating that half of the final policy updates, used to gather states to train the AE, went entirely right as opposed to entirely left like the final version of the model. This means that the corresponding AE was not trained to perfectly recognize the final behavior, instead granting a high score to behavior that was equal to its final trajectory.

Deceptive Maze Environment

The results in the Deceptive Maze environment were relatively successful, with three out of five tests finding both paths to the goal. The two tests that failed still found one path before blocking the second path with a novel behavior that happened to exit the starting room. An example of this is seen in fig. 20. Once both exits have been discovered, the chances of a new policy going in the same direction diminish, resulting in new policies mostly ignoring the novelty gradient.

9.2 On-Policy Novelty Search with LSTM-Autoencoders

General Considerations

The use of LSTM-AE for on-policy novelty search showed some promising results in terms of exploration. The primary drawback is the lack of convergence, resulting from the incentive to continuously find behaviors that are different from the current one.

Using LSTM-AE in combination with an on-policy algorithm like PPO might have some unintended effects.

One challenge is the effect it has on the critic network, and subsequently the generalized advantage, as the final score for the state transitions will change depending on how often the sequence is encountered in the immediate previous iterations. This will cause the critic to be unable to accurately estimate the values of different states, potentially hindering the optimization process.

On-policy novelty search might be more efficient when combined with an evolutionary algorithm. One such possibility could be NEAT, where the LSTM-AE would score entire genomes rather than modifying single-step rewards. Using NEAT also opens the possibility for modifying the species definition, from being dependent on genetic distance, to instead rely on behavioral distance. By doing so, while maintaining the idea that species compete internally, it might allow for diverse behaviors to be discovered, where the quality within the species could be explored.

Due to the seemingly very efficient exploration in the Deceptive Maze, another approach could involve using PPO with on-policy novelty to gather environment interactions. A secondary, off-policy, algorithm could then be used to construct a final policy based on the encountered observations, effectively decoupling exploration and exploitation. A similar methodology was mentioned in section 2.2, where [65] similarly decoupled the objectives. This hybrid method might exploit the efficient exploration of LSTM-AE novelty to effectively cover the behavioral space, and use the broadly gathered information in a more effective manner.

Open Maze Environment

As seen in fig. 27b, the LSTM-AE often resulted in behaviors that maximized acceleration in the Open Maze environment. This is due to movements in opposite directions inherently being the most novel relative to each other.

The oscillating trajectories might be circumvented by including batches of previous trajectories in the training stage of the LSTM-AE. This approach would likely require testing of the complexity of the network to ensure it can handle the additional data without a loss in performance.

By comparing the scatter plots from novel policy-seeking, fig. 22a, and the combined algorithm, fig. 34a, it becomes apparent that the clustering effect of the novel policy-seeking algorithm slightly diminishes, while the overall coverage of the behavior space becomes more prominent when the on-policy novelty is introduced. If this effect is maintained as the dimensionality of the behavior space increases, the combined algorithm could prove to be an efficient way of exploring and diversifying potential solutions.

Deceptive Maze Environment

The LSTM-AE demonstrated a strong ability to reach the goal in the Deceptive Maze environment, often finding a capable policy within just 20 iterations.

The non-converging nature of novelty search with LSTM-AE often results in both of the separate paths being found within the 100 total iterations that the model is trained for, this is seen both fig. 32 and fig. 37. If a method were designed to determine which path is found, the novel policy-seeking algorithm would become unnecessary to solve this specific environment.

The effect of LSTM-AE seems to become more prominent as the trajectory length increases. This is seen in the Open Maze maps, where the model learns to take sharp turns somewhere along the path. In the Deceptive Maze, this effect is seen in most, if not all, successful trajectories, as they typically dip into the local optimum before exiting the starting room. This phenomenon is likely inherent in the RNN structure, where the reconstruction error might escalate over time due to smaller errors at the beginning of the trajectory. One could train the LSTM-AE to predict future states from the current, and use this error as the additional reward in the current step, instead of using the direct reconstruction of the trajectories.

A simple tests were conducted, where the LSTM-AE was only trained when an optimum were detected based on the average change in fitness between iterations. In this test, clusters of final positions were located in all three local optima, and the policy still managed to complete the map, though it required more iterations. The local optima detection could also be applied in the behavior space by using the loss of the LSTM-AE. If a secondary LSTM-AE trains on all sequences encountered, we could copy its parameters to the one used to calculate the novelty score, whenever the secondary networks' loss is below some threshold.

Remaining Environments

The effect of the on-policy novelty algorithm is seen in the first of the trained models, which, for most environments, usually result in a more "aggressive" of the pure fitness based solution. As seen in the repository [4], the subsequent policies, when both algorithms are active, show even more novelty while struggling with the fitness score. This is likely due to fitness only accounting for around $1/3$ of the weighted sum of rewards. It would likely be ideal to combine the two novelty scores, before applying the α regulation between the fitness and summed novelty.

The model that found the high-performing swimmer clearly decayed in performance, as visualized in fig. 38. This decline is a logical side effect of the constant pressure to continually exhibit novel behavior.

9.3 Potential Future Work

Regarding the current implementation, most parameters for both types of novelty search remains to be tested. Some of these include the length of trajectories, topology of the networks, and the different weights or calculations of the applied scores. It would also have been beneficial to apply the LSTM-AE for novel policy-seeking, such that the two methods could have been directly compared.

LSTM-AE could be compared to the generic novelty search algorithm in the designed Deceptive Maze. If measuring the required amount of iterations before a solution is found, the viability of using LSTM-AEs to generalize novelty search could be determined.

To further prove that generalization was achieved, a broader spectrum of deceptive environments should have been utilized. If solutions were found for all the deceptive environments, without changes to the algorithm, the effectiveness of generalization could be decided.

Customizing Behavior Selection

By having metrics of all previous behaviors saved as AEs, one could handpick unwanted behaviors to manually guide the path of learning. One way to test this could be an attempt at guiding the model towards the optimal Swimmer behavior. As the optimal Swimmer behavior is somewhat rare, it could be interesting to see whether blocking the same specific previous behaviors would always result in a model of similar quality.

Manually choosing behaviors could also be used for behavior mimicry by inverting the novelty score signal. This could be used to either combine found behaviors, or to directly mimic a single behavior. A preliminary test with a running version of the Walker was conducted in the hardcore environment, as the stance of the runner seems like it would be optimal for jumping over the obstacles. This test showed that some tuning would be required for this to work, as the model ended up standing in a specific pose that were predominant for the running behavior. A possible reason for this poor result could simply be the overall difficulty of the hardcore environment.

Fréchet Distance

The Fréchet distance was used to measure the difference between the trajectories in the Open Maze. Since the matrix contains a lot of information regarding similarities, it might be worth testing whether one could use it to generalize the behavioral difference metric. While this might be computationally complex, it could prove to be effective, if a proper state-distance function were determined.

LSTM-VAE for Novelty Search

In the initial stages of developing on-policy novelty, we considered both LSTM-AE and variational autoencoders (VAE). The key idea of VAEs are to encode the variables into probability distributions rather than a latent dimension. Describing the latent space probabilistically allows for the clustering of high-dimensional data.

In [66] the authors combine VAE with the time-series efficiency of LSTM modules to effectively detect anomalies on multimodal input data. Since anomaly detection were used as the basis for the implementation of novelty search with LSTM-AE, the implementation from this paper might translate just as well.

Variational Option Discovery Algorithms (VALOR), as described in [67], utilize the reconstruction error of sequential state data from an LSTM-VAE, to guide the agents' discovery of a variety of different behaviors. This approach aligns perfectly with the implementations used in this thesis, and could have been used as a core concept.

Two-Objective Optimization

For harder environments, such as the Walker and the Swimmer, the algorithm initially finds some entirely novel solutions for the earlier policies. An example of such policies is the two most common novel behaviors of the Walker. One of these simply wiggles on the ground, which would be an unlikely contender for a stepping stone towards ideal behavior. Another novel behavior, however, stand with its legs straight while also wiggling, this one could potentially block future behaviors from ever standing completely upright, which might be part of some desired behavior. Even when using the dynamically adjusted weights, the first few policies tend to prioritize novelty while mostly ignoring fitness. The conditional resetting of the weight was implemented with this issue in mind, but as indicated by fig. 13, the weight actually goes toward preferring fitness, since small increases in quality are present, even when the resulting behavior is mostly novel. This indicates that dynamical weight should be replaced by some other multi-objective method. An attempt to circumvent this specific issue is seen in [27], where the authors check the angle between the two gradients. To ensure that the fitness gradient is always used, they modify the task-novel gradient bisector whenever the angle between these gradients are more than $90°$. Even with this method, they still see a few mostly novel behaviors, hinting at the fact that a more advanced two-objective optimization method might be required to ensure that all policies are of high quality.

If we found a way to estimate the Pareto front for fitness and novelty, like the method used in [7], we could score the environment interactions based on their distance to this front to give an incentive for found policies to be close to the optimal combinations. The novelty score could be computed directly from this distance, which might bypass the oscillatory nature of on-policy novelty if the front were to be estimated with all measurements in mind. The policy-seeking algorithm might also benefit from this, as the fitness score would be part of the novelty calculation, which might avoid the purely novel solutions.

10 Conclusion

In this thesis, we explored the concept and benefits of novelty search, and the intricacies of generalizing the behavior metric to broaden the applicability of novelty search. A generalized metric, novelty score, and method of combining novelty with fitness were employed to effectively find multiple novel behaviors, while having the ability to overcome immediate deception. To prove generalization, four highly different environments were

implemented, where no changes were required for the employed methods.

We devised two distinct algorithms. One for seeking novelty between policies, and one for applying novelty to increase exploration efficiency.

The novel policy-seeking algorithm used the reconstruction error from autoencoders, trained to reconstruct the sequential state data from the final update steps of a policy. This allowed a single autoencoder to give a generalized novelty measurement against this trained policy. New policies were scored based on the lowest mean squared reconstruction error from all previously trained autoencoders, ensuring that new policies were novel against all previous behaviors at once. A method of calculating and normalizing the score based on the mean and standard deviation of the reconstruction errors were employed. To combine the novelty score with the fitness, a weighted sum of rewards were used, where the weight was dynamically adjusted based on the progress along the fitness gradient.

After testing the novel policy-seeking algorithm, a few issues were identified, leading to the development of an on-policy novelty search algorithm. This algorithm intended to improve the general exploration of the agent, by applying pressure towards procedural novel behavior. To accomplish this, an LSTM-autoencoder was trained to reconstruct all the encountered state trajectories as the policy training were progressing. Here, the novelty score was applied as a fixed weighted sum of rewards of the mean absolute reconstruction error, linearly normalized based on the current rollout.

The result from the BipedalWalker environment demonstrated how novel behaviors could still have high quality. A wide array of distinct behaviors were discovered across multiple runs. While none of the discovered behaviors outperformed the fitness-based solution by any margin of relevance, a few did converge slightly faster, and occasionally score slightly higher. Due to the stochastic nature of this environment, the faster convergence was presumed to be the result of chance rather than the algorithm discovering a better path of learning.

In the Swimmer environment, a single novel behavior that obtained a high fitness score were found. This behavior outperformed existing benchmarks by a significant margin, both in terms of rate of convergence and final score. This same optimal behavior was also found when incorporating the on-policy novelty. Here, the behavior was discovered even earlier in the training process, but convergence to this behavior were avoided due to the nature of on-policy novelty.

In the UR5 environment, the tests showed that, on average, close to half of the possible inverse kinematics configuration were obtained. This environment served to prove a real-world application for the novel policy-seeking algorithm, the potential of which these results seem to support.

The Deceptive Maze environment were successfully solved in every attempt with the on-policy novelty search, and both paths were consistently found with the combination of both algorithms. The Open Maze were intended

to visualize and measure the effect of both algorithms by plotting the resulting trajectories. In order to measure the similarities between these two-dimensional trajectories, a recursive method was employed to calculate the discrete Fréchet distance.

Overall, the methods tested in this thesis demonstrate the potential of generalized behavior metrics for novelty search. By leveraging two types of autoencoders, we have shown that it is possible to generalize novelty search across a wide spectrum of reinforcement learning tasks, without the need for environment-specific knowledge usually required to define the behavior metrics. This approach further broadens the applicability of novelty search, and paves the way for future research aimed at enhancing similar techniques.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Anders L. Christensen, for always steering me in the right direction whenever I felt lost about the direction to take the project, and for calming me when I stressed about something insignificant.

I would also like to thank my family, especially my fiancée, for their support, love, and acceptance of me being less than present during the development of this thesis.

References

[1] L. D. Whitley, "Fundamental principles of deception in genetic search," in *Foundations of genetic algorithms*. Elsevier, 1991, vol. 1, pp. 221–241.

[2] J. Lehman, K. O. Stanley *et al.*, "Exploiting open-endedness to solve problems through the search for novelty." in *ALIFE*, 2008, pp. 329–336.

[3] N. Bougie and R. Ichise, "Fast and slow curiosity for high-level exploration in reinforcement learning," *Applied Intelligence*, vol. 51, pp. 1086–1107, 2021.

[4] B. Longet, "Repository containing the code used in this thesis, as well as gifs and images that better show dynamical behavior," [Online; accessed 20-05-2024]. [Online]. Available: https://github.com/BenjaminLonget/Novelty-Search-Master-Thesis/tree/main

[5] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[6] G. Cuccu and F. Gomez, "When novelty is not enough," in *European conference on the applications of evolutionary computation*. Springer, 2011, pp. 234–243.

[7] J.-B. Mouret, "Novelty-based multiobjectivization," in *New Horizons in Evolutionary Robotics: Extended Contributions from the 2009 EvoDeRob Workshop*. Springer, 2011, pp. 139–154.

[8] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii," in *Parallel Problem Solving from Nature PPSN VI: 6th International Conference Paris, France, September 18–20, 2000 Proceedings 6*. Springer, 2000, pp. 849–858.

[9] J. Lehman and K. O. Stanley, "Revising the evolutionary computation abstraction: minimal criteria novelty search," in *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, 2010, pp. 103–110.

[10] J. Gomes, P. Urbano, and A. L. Christensen, "Progressive minimal criteria novelty search," in *Advances in Artificial Intelligence–IBERAMIA 2012: 13th Ibero-American Conference on AI, Cartagena de Indias, Colombia, November 13-16, 2012. Proceedings 13*. Springer, 2012, pp. 281–290.

[11] N. Hay, M. Stark, A. Schlegel, C. Wendelken, D. Park, E. Purdy, T. Silver, D. S. Phoenix, and D. George, "Behavior is everything: Towards representing concepts with sensorimotor contingencies," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[12] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," *Advances in neural information processing systems*, vol. 30, 2017.

[13] T. Geijtenbeek, M. Van De Panne, and A. F. Van Der Stappen, "Flexible muscle-based locomotion for bipedal creatures," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 6, pp. 1–11, 2013.

[14] V. R. López-López, L. Trujillo, and P. Legrand, "Novelty search for software improvement of a slam system," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2018, pp. 1598–1605.

[15] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret, "Robots that can adapt like animals," *Nature*, vol. 521, no. 7553, pp. 503–507, 2015.

[16] A. Liapis, G. N. Yannakakis, and J. Togelius, "Enhancements to constrained novelty search: Two-population novelty search for generating game content," in *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, 2013, pp. 343–350.

[17] J. Lehman and K. O. Stanley, "Novelty search and the problem with objectives," *Genetic programming theory and practice IX*, pp. 37–56, 2011.

[18] ——, "Abandoning objectives: Evolution through the search for novelty alone," *Evolutionary computation*, vol. 19, no. 2, pp. 189–223, 2011.

[19] E. Conti, V. Madhavan, F. Petroski Such, J. Lehman, K. Stanley, and J. Clune, "Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents," *Advances in neural information processing systems*, vol. 31, 2018.

[20] C. Colas, O. Sigaud, and P.-Y. Oudeyer, "Gep-pg: Decoupling exploration and exploitation in deep reinforcement learning algorithms," in *International conference on machine learning*. PMLR, 2018, pp. 1039–1048.

[21] J. Parker-Holder, A. Pacchiano, K. M. Choromanski, and S. J. Roberts, "Effective diversity in population based reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 18 050–18 062, 2020.

[22] H. Sun, Z. Peng, B. Dai, J. Guo, D. Lin, and B. Zhou, "Novel policy seeking with constrained optimization," *arXiv preprint arXiv:2005.10696*, 2020.

[23] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine, "Diversity is all you need: Learning skills without a reward function," *arXiv preprint arXiv:1802.06070*, 2018.

[24] K. Xu, Y. Ma, and W. Li, "Dynamics-aware novelty search with behavior repulsion," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2022, pp. 1112–1120.

[25] E. Meyerson, J. Lehman, and R. Miikkulainen, "Learning behavior characterizations for novelty search," in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, 2016, pp. 149–156.

[26] J. Gomes and A. L. Christensen, "Generic behaviour similarity measures for evolutionary swarm robotics," in *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, 2013, pp. 199–206.

[27] Y. Zhang, W. Yu, and G. Turk, "Learning novel policies for tasks," in *International Conference on Machine Learning*. PMLR, 2019, pp. 7483–7492.

[28] H. Face, "Hugging face - deep reinforcement learning course," [Online; accessed 31-05-2024]. [Online]. Available: https://huggingface.co/learn/deep-rl-course/unit0/introduction

[29] R. S. Sutton, A. G. Barto *et al.*, "Reinforcement learning," *Journal of Cognitive Neuroscience*, vol. 11, no. 1, pp. 126–134, 1999.

[30] OpenAI, "Proximal policy optimization," [Online; accessed 20-04-2024]. [Online]. Available: https://openai.com/research/openai-baselines-ppo

[31] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[32] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*. PMLR, 2015, pp. 1889–1897.

[33] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.

[34] OpenAI, "Gymnasium documentation," [Online; accessed 06-03-2024]. [Online]. Available: https://gymnasium.farama.org/

[35] ——, "Bipedal walker documentation," [Online; accessed 06-03-2024]. [Online]. Available: https://gymnasium.farama.org/environments/box2d/bipedal_walker/

[36] E. Catto, "Box2d, a 2d physics engine for games," [Online; accessed 23-04-2024]. [Online]. Available: https://box2d.org/

[37] OpenAI, "Gymnasium challenge leaderboard," [Online; accessed 23-04-2024]. [Online]. Available: https://github.com/openai/gym/wiki/Leaderboard

[38] G. DeepMind, "Multi-joint dynamics with contact, mujoco homepage," [Online; accessed 23-04-2024]. [Online]. Available: https://mujoco.org/

[39] W. Xia, "Hits dynamic," [Online; accessed 23-04-2024]. [Online]. Available: https://github.com/WanqingXia/HiTS_Dynamic/tree/main

[40] A. Levy, G. Konidaris, R. Platt, and K. Saenko, "Learning multi-level hierarchies with hindsight," *arXiv preprint arXiv:1712.00948*, 2017.

[41] Q. Gallouédec, N. Cazin, E. Dellandréa, and L. Chen, "panda-gym: Open-Source Goal-Conditioned Environments for Robotic Learning," *4th Robot Learning Workshop: Self-Supervised and Lifelong Learning at NeurIPS*, 2021.

[42] E. Coumans, "Bullet physics engine homepage," [Online; accessed 23-04-2024]. [Online]. Available: https://pybullet.org/wordpress/

[43] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine, "D4rl: Datasets for deep data-driven reinforcement learning," 2020.

[44] R. de Lazcano, K. Andreas, J. J. Tai, S. R. Lee, and J. Terry, "Gymnasium robotics," 2023. [Online]. Available: http://github.com/Farama-Foundation/Gymnasium-Robotics

[45] U. spiros, "Discrete fréchet distance," [Online; accessed 4-04-2024]. [Online]. Available: https://github.com/spiros/discrete_frechet/tree/master

[46] T. Eiter and H. Mannila, "Computing discrete fréchet distance," 1994.

[47] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[48] N. I. Tapia and P. A. Estévez, "On the information plane of autoencoders," in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–8.

[49] S. Ladjal, A. Newson, and C.-H. Pham, "A pca-like autoencoder," *arXiv preprint arXiv:1904.01277*, 2019.

[50] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[51] OpenAI, "Spinning up in deep rl," [Online; accessed 8-04-2024]. [Online]. Available: https://spinningup.openai.com/

[52] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[53] A. Graves and A. Graves, "Long short-term memory," *Supervised sequence labelling with recurrent neural networks*, pp. 37–45, 2012.

[54] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[55] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," *Neural computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[56] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2222–2232, 2016.

[57] NVidia, "Long short-term memory (lstm)," [Online; accessed 15-03-2024]. [Online]. Available: https://developer.nvidia.com/discover/lstm

[58] C. Zhou, C. Sun, Z. Liu, and F. Lau, "A c-lstm neural network for text classification," *arXiv preprint arXiv:1511.08630*, 2015.

[59] S. Santhanam, "Context based text-generation using lstm networks," *arXiv preprint arXiv:2005.00048*, 2020.

[60] H. Soltau, H. Liao, and H. Sak, "Neural speech recognizer: Acoustic-to-word lstm model for large vocabulary speech recognition," *arXiv preprint arXiv:1610.09975*, 2016.

[61] B. Lindemann, B. Maschler, N. Sahlab, and M. Weyrich, "A survey on anomaly detection for technical systems using lstm networks," *Computers in Industry*, vol. 131, p. 103498, 2021.

[62] P. Malhotra, L. Vig, G. Shroff, P. Agarwal *et al.*, "Long short term memory networks for anomaly detection in time series." in *Esann*, vol. 2015, 2015, p. 89.

[63] P. Foundation, "Pytorch lstm documentation," [Online; accessed 15-03-2024]. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html

[64] hellojinwoo, "Torchcoder," [Online; accessed 12-03-2024]. [Online]. Available: https://github.com/hellojinwoo/TorchCoder/blob/master/autoencoders/rae.py

[65] S. Forestier, R. Portelas, Y. Mollard, and P.-Y. Oudeyer, "Intrinsically motivated goal exploration processes with automatic curriculum learning," *Journal of Machine Learning Research*, vol. 23, no. 152, pp. 1–41, 2022.

[66] D. Park, Y. Hoshi, and C. C. Kemp, "A multimodal anomaly detector for robot-assisted feeding using an lstm-based variational autoencoder," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1544–1551, 2018.

[67] J. Achiam, H. Edwards, D. Amodei, and P. Abbeel, "Variational option discovery algorithms," *arXiv preprint arXiv:1807.10299*, 2018.

List of Figures

List of Tables