Universiteit Leiden
The Netherlands

BSc Data Science & AI

Application of Knowledge Transfer in Reinforcement Learning

Avan Salman

Supervisors:
Aske Plaat & Álvaro Serra-Gómez

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

1/7/2025

Abstract

This thesis report presents an investigation into knowledge transfer in Deep Reinforcement Learning (Deep-RL) with continuous state-action space robotic locomotion tasks. The goal is to determine whether transferring knowledge from a model trained on a base environment could improve performance on similar tasks. A set of custom variations of the Ant environment are made by altering the Ant's morphology or surroundings. The training progress of models trained from scratch to those pretrained on the Ant base task is assessed and the final model performances are evaluated and compared. Results show that knowledge transfer is beneficial when morphological differences are small. Large morphological changes resulted in failure to find a viable policy in some cases. For changes made in the environment knowledge transfer was not beneficial, but did not result in drastically adverse effects. Therefore it is concluded that knowledge transfer in Deep-RL can be effective when the source and target tasks are adequately similar, especially in terms of morphology.

Contents

1 Introduction **1**
1.1 Thesis overview . . . 2

2 Related Work **3**
2.1 Deep Reinforcement Learning Methods . . . 3
2.1.1 Actor Critic Algorithms . . . 3
2.2 Proximal Policy Optimization . . . 4
2.3 Soft Actor Critic . . . 5
2.4 Knowledge Transfer . . . 6
2.5 The Ant Environment . . . 7

3 Experiment Setup **9**
3.1 Algorithm Implementation & Experiment Execution . . . 9
3.2 Ant Variations . . . 9
3.2.1 Ant Body Variations . . . 10
3.2.2 Ant Environment Variations . . . 10
3.2.3 Ant Variant Selection . . . 11
3.3 Experiments . . . 14
3.3.1 Performance Comparison SAC, PPO . . . 14
3.3.2 Knowledge Transfer Experiments . . . 15

4 Results **17**
4.1 Results SAC vs PPO . . . 17
4.2 Results Knowledge Transfer Experiments . . . 18
4.2.1 Training Results . . . 18
4.2.2 Model Evaluation Results . . . 21

5 Conclusions and Further Research **24**

References **27**

Appendix **28**

A **28**
A.1 Links: Results, Code and Experimentation . . . 28
A.1.1 CleanRL PPO & SAC . . . 28

A.1.2 WandB 28
A.1.3 Github 28
A.2 PPO & SAC networks and parameter settings 29
A.2.1 PPO settings 29
A.2.2 SAC settings 30

Chapter 1

Introduction

Reinforcement Learning (RL) is a type of machine learning where an agent learns to perform a task by interacting with its environment. The learning process involves the agent observing the current state of the environment, selecting an action based on that observation and receiving feedback in the form of a reward. This reward indicates how beneficial the action was in terms of progressing toward a desired goal or completing the task objective. As the agent gathers experience, a policy mapping state observations to actions is learned. The learning objective is to find a policy that maximizes cumulative rewards over time [SB18]. Initial RL algorithms relied on tabular methods that have proven successful in discrete state action space settings but have been limited in continuous environments. The introduction of Deep-RL methods, combining RL with Deep Neural Networks enabled learning in complex continuous state and action space environments.

Deep-RL still presents many challenges, such as poor sampling and computational efficiency. In continuous environments usually millions of agent environment interactions are required for learning a useful policy and these interactions can only be performed sequentially per agent environment pair. A possible improvement can be achieved by instead of training an agent from a zero knowledge starting point for each new task, to employ previously learned knowledge form similar tasks. With knowledge transfer the efficiency limitations may be addressed to some extent by enabling agents to reuse previously learned skills or representations. Applied to similar tasks, this may reduce the amount of training required, resulting in accelerated learning and possibly improved performance. Knowledge transfer therefore can play a crucial role in expanding Deep-RL application.

The focus of this research project is on knowledge transfer in a RL setting. Specifically to investigate how well knowledge acquired in one task can be adapted to improve performance in a similar task. This will be done in application for continuous state and action space robotic locomotion environments. The main objectives of this project are represented in the following research questions:

- RQ 1. How well does learned knowledge generalize to similar but not identical continuous state-action space robotic locomotion tasks?
- RQ 2. Is knowledge from the source task preserved after training on the target task?

To answer these questions it must be determined what algorithms and environments are suitable for the knowledge transfer application. This presents the following preliminary tasks:

- What environments and tasks are suitable for implementing and evaluating knowledge transfer? The aim here is to select an environment and design variations of that environment suited for testing the effect of knowledge transfer.
- What algorithms are suitable for training on the chosen continuous state-action space tasks? Here the aim is to identify and test a suitable Deep-RL algorithm for training on the chosen tasks.

1.1 Thesis overview

With this thesis project the potential of Knowledge Transfer application with Deep Reinforcement Learning is investigated. Where the focus is on continuous state-action space robotic locomotion environments. Experimentation is done to investigate how knowledge learned from a source task improves learning and final model performance of a selected target task.

In chapter 2 relevant concepts are presented to provide background knowledge for the conducted research. Here Deep-RL and the used algorithms PPO and SAC are discussed, an overview of the applied knowledge transfer approach is given and the environment used for experimentation is discussed.

Following the experiment setup is explained. To answer the research questions, variations of the chosen environment are designed and configured. In chapter 3 the process of setting up the environment variations is described along with the algorithm implementation and experiment execution and finally the conducted experiments are presented.

In chapter 4 the experiment outcomes are presented and an analysis of the findings is given. Based on these findings in the final chapter 5 a conclusions is given providing the answers to the research questions.

Chapter 2

Related Work

2.1 Deep Reinforcement Learning Methods

Deep Reinforcement Learning (Deep RL) methods can be broadly characterized by three categories. Deep value-based methods, aimed at learning a state-value (V-value) or state-action (Q-value) function parameterized by a neural network and from this value function the policy is then derived. There are also policy gradient methods that directly learn the policy function for mapping state observations to actions also parameterized by a neural network. And finally model-based methods, where a transition model of the environment is initially learned and can be used for the policy optimization [FLHI+18].

For this project the focus is on policy gradient based model-free methods specifically actor-critic algorithms as they have proven to perform well in continuous state-action space environments and robotic locomotion tasks [TAH+24]. Two algorithms have been considered for the experimentation Proximal Policy Optimization (PPO) and Soft Actor Critic (SAC) which will be described in the following sections.

2.1.1 Actor Critic Algorithms

Actor Critic algorithms are a class of policy gradient methods that combine a policy function (the actor) and a value function (the critic). The actor selects actions based on the state observation, while the critic estimates the state-value function $V(s)$ or a state-action value function $Q(s, a)$ used for the policy evaluation. The policy and value functions are both learned simultaneously during training. The critic is applied for the optimization of the actor but is not required for inference after training is concluded. The use of the actor critic method stabilizes learning by reducing the variance in gradient estimations[Ben19].

In the case of PPO, the critic provides a $V(s)$ function estimation that serves as a baseline for calculating the generalized advantage estimate (GAE) [SML+15]. This is an estimation of the advantage of taking a certain action in any given state compared to the average expected return for that state under the current policy.

For the SAC algorithm the critic (or actually critics further explained in section 2.3) estimates the $Q(s, a)$ function. The Q-value provided by the critic is then used in the policy loss computation as a target value for the expected returns. As such with SAC the critic provides a more stable return estimation based on which the policy optimization is performed.

2.2 Proximal Policy Optimization

Proximal Policy Optimization is an on-policy actor-critic algorithm that has proven to perform well on many types of tasks for both continuous and discrete action spaces. The main feature of PPO is the objective function with clipped probability ratios [SWD+17]. The common policy gradient objective with policy parameters ϕ is to maximize:

$$J^{PG}(\phi) = E_t \left[\log \pi_\phi(a_t \mid s_t) \hat{A}_t \right] \tag{2.1}$$

where the advantage estimate $\hat{A}_t$ with GAE is given by:

$$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l} = \delta_t + \gamma\lambda \hat{A}_{t+1} \tag{2.2}$$

With δ_t being the temporal difference error (TD-error), the error between the expected state-value and the improved estimation of the state-value based on the observed reward at timestep t and state-value estimation at $t+1$ discounted by $\gamma \in [0, 1]$:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{2.3}$$

As advantages are calculated recursively a trajectory is first sampled from the current policy π_θ. For the reversed trajectory the advantages over time-steps are calculated. With GAE the bias variance tradeoff in the advantage estimation can be controlled with parameter $\lambda \in [0, 1]$. As with lower λ the estimate becomes closer to the one-step TD-error and variance is reduced in exchange for increased bias. While with higher λ more weight is given to future TD-errors lowering bias in trade for increased variance[SML+15].

For the actor optimization, PPO uses the clipped surrogate objective function where not the log probability but the action probability ratio between the current and previous policy $r(\phi)$ is used:

$$r(\phi) = \frac{\pi_\phi(a_t|s_t)}{\pi_{\phi_{old}}(a_t|s_t)} \tag{2.4}$$

The ratio is a measure of how far the current policy has moved from the old policy. To control the policy update size $r_t(\phi)$ is clipped, forcing it between bounds. The clip size is controlled by the parameter ϵ to a maximum of $1 + \epsilon$ and minimum $1 - \epsilon$:

$$J^{CLIP}(\phi) = E_t \left[min(r_t(\phi)\hat{A}_t,\ clip(r_t(\phi), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right] \tag{2.5}$$

With PPO after sampling multiple trajectories and calculating the advantages for these trajectories multiple update epochs are typically performed on the samples. For the first epoch

$\pi_\phi(a_t|s_t) = \pi_{\phi_{old}}(a_t|s_t)$ the objective simplifies to $\hat{A}_t$. As with following epochs the current and old policies diverge this results in a larger probability ratio. The clipped surrogate objective by bounding the probability ratio restricts the update size preventing excessively large updates thus stabilizing learning.

In implementation one loss is calculated for both the actor and critic. This is done to allow for shared neural network parameters between the actor and critic. Also an entropy bonus term is added awarding higher policy entropy and as such stochastic behavior for increased exploration. Resulting in the combined objective to be maximized at each optimization step:

$$J_t^{CLIP+VF+S}(\phi) = E_t\left[J_t^{CLIP}(\phi) - c_1 J_t^{VF}(\phi(\cdot|s_t) + c_2 S(\pi_\phi(\cdot|s_t)\right] \tag{2.6}$$

Here the parameters c_1 and c_2 weight the importance of the critic loss and entropy bonus.

2.3 Soft Actor Critic

The Soft Actor Critic is an off-policy actor-critic algorithm designed for continuous state-action space environments[HZAL18]. SAC is based on the maximum entropy RL framework aimed at improving exploration by promoting policy stochasticity. This is achieved by extending the standard RL objective that is to maximize the expected cumulative return under policy π:

$$\pi^* = \arg\max_\pi \sum_t E_{(s_t,a_t)\sim\rho^\pi}\left[r(s_t, a_t)\right] \tag{2.7}$$

with the addition of an entropy term that rewards policy stochasticity. As such, with SAC the objective becomes to maximize for expected returns as well as policy entropy as seen in the following equation:

$$\pi^* = \arg\max_\pi \sum_t E_{(s_t,a_t)\sim\rho^\pi}\left[r(s_t, a_t) + \alpha\mathcal{H}(\pi(\cdot \mid s_t))\right] \tag{2.8}$$

Here $\mathcal{H}(\pi(\cdot \mid s_t))$ is the policy entropy at state s_t. The temperature parameter α controls the importance of the entropy in relation to the reward. In implementation optionally α can be tuned during training against a fixed target entropy based on the size of the action space. The entropy term itself is the expectation of the negative log probability of the action under policy π:

$$\mathcal{H}(\pi(\cdot \mid s)) = -E_{a\sim\pi}\left[\log \pi(a \mid s)\right] \tag{2.9}$$

With the extended objective the policy is optimized to encourage actions that lead to states with higher future entropy, states where multiple actions are possible for achieving desirable results. This approach results in more robustness in stochastic environments and improved convergence. The maximum entropy objective can be especially useful for environments that have multiple solution trajectories. For example navigating a maze that has multiple possible routes to the goal location and can be beneficial in knowledge transfer from general to more specific behaviors [HTAL17].

As stated SAC is an actor-critic algorithm. The actor - policy function - and critic - Q-function - are approximated by neural networks that are trained to minimize a loss. For the critic network the loss is the expected mean squared error defined as:

$$J_Q(\theta) = E_{(s_t,a_t)\sim\mathcal{D}}\left[\frac{1}{2}\left(Q_\theta(s_t, a_t) - \hat{Q}(s_t, a_t)\right)^2\right], \tag{2.10}$$

where $\mathcal{D}$ is the replay buffer containing the sampled trajectories, $Q_\theta(s_t, a_t)$ is the current Q-value estimation and $\hat{Q}$-value is the target Q-value calculated as:

$$\hat{Q}(s_t, a_t) = r(s_t, a_t) + \gamma \, E_{s_{t+1} \sim p} \left[V_{\bar{\psi}}(s_{t+1}) \right] \tag{2.11}$$

with state-value function $V_{\bar{\psi}}$ incorporating the entropy term given by:

$$V(s_t) = E_{a_t \sim \pi} \left[Q(s_t, a_t) - \log \pi(a_t \mid s_t) \right] \tag{2.12}$$

As $V_{\bar{\psi}}$ can be estimated by the Q-function a separate network is not needed. Also the SAC implementation uses two critic networks that are intended to reduce Q-value overestimation bias by taking the minimum of the two network outputs [FHM18]. As such the target Q-value in implementation is calculated as:

$$\hat{Q}(s_t, a_t) = r_t + \gamma \, E_{a_{t+1} \sim \pi} \left[\min \left(Q_1(s_{t+1}, a_{t+1}), \, Q_2(s_{t+1}, a_{t+1}) \right) - \alpha \log \pi(a_{t+1} \mid s_{t+1}) \right] \tag{2.13}$$

For the actor network the optimization objective to minimize is:

$$J_\pi(\phi) = E_{s_t \sim \mathcal{D}, \, \epsilon_t \sim \mathcal{N}} \left[\alpha \log \pi_\phi(f_\phi(\epsilon_t; s_t) \mid s_t) - Q_\theta(s_t, f_\phi(\epsilon_t; s_t)) \right] \tag{2.14}$$

As the network output is modeled as a Gaussian probability distribution, to enable stochasticity. The action needs to be parameterized for the gradient calculation. This is achieved using the parametrization trick [KW13] with function $f_\phi(\epsilon_t; s_t)$ where $f_\phi(\epsilon_t; s_t) = \mu_\phi(s_t) + \sigma_\phi(s_t) \cdot \epsilon$, with $\epsilon = \mathcal{N}(0, 1)$ enabling gradient descent optimization.

Another important feature is that SAC is an off-policy algorithm, this is achieved by incorporating a replay buffer. As training commences the buffer is filled with gathered experiences. With each update step a batch of experiences is randomly sampled resulting in batches containing experiences from past policies. This is off-policy learning and can produce improved sampling efficiency.

2.4 Knowledge Transfer

Although the introduction of Deep-RL has vastly expanded the application domain for RL algorithms, challenges such as the requirement of large amounts of data, computational inefficiency and poor generalization still limit RL application in more complex tasks. With the application of knowledge transfer or Transfer Learning (TL) in RL, these limitations could be addressed to some extent. The aim of TL in RL can generally be defined as the following: Given a source domain and a target domain the aim is to learn an optimal policy for the target domain by leveraging exterior information from the source domain as well as interior information from the target domain [ZLJZ20]. More specifically with TL it is attempted to reuse learned knowledge such as learned policy functions, value functions or environment models from the source task to improve performance for the target task.

For this project several considerations needed to be made for applying knowledge transfer. As the SAC actor critic algorithm is used, the knowledge gained during training is represented in the learned policy and value functions. Although only the policy is needed for action selection. During

training the actor's performance is measured based on the double critics Q-values estimations, therefore both functions are needed in the knowledge transfer.

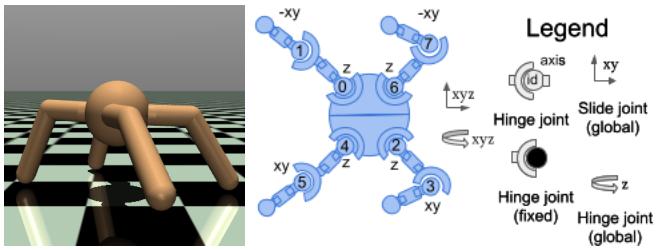
For the choice of tasks, the focus is on robotic locomotion tasks with continuous state and action spaces. An important considerations for the function transfer are source and target environments state and/or action space sizes. Differences in these sizes for the source and target task present additional challenges for transferring the learned policy and value functions.

Finally the effect of the knowledge transfer is assessed by comparing the training progress and final model performance of the pretrained source task and a non pretrained version of the same task. The performance is measured as the episodic returns achieved both during training and for the fully trained models.

2.5 The Ant Environment

Multiple continuous state and action space environments from the Gymnasium library [TKT+24] were considered and tested for this project. This selection process will not be discussed as it deviates from the project objective. For this project the latest version of the Ant environment, the Ant-v5 is used. The Ant environment was chosen as it showed the best performance with the PPO algorithm from the tested options and also because it is a robotic agent simulation in 3-dimensional space, it offered the most flexibility for designing interesting environment variations.

The Ant represents a quadruped robot simulated in 3-dimensional space using the MuJoCo (Multi-Joint dynamics with Contact) general purpose physics engine [TET12], often used for robotic simulations and RL-research. MuJoCo uses the XML markup language to define the environment setup. As required for the experiments changes to the Ant simulation such as morphology adjustments or adding objects to the surroundings can be made by adjusting the XML file.



Source: https://gymnasium.farama.org

As can be seen in the illustrations, the Ant is composed of a spherical body with four legs each having two joints. Resulting in an action space of 8 continuous elements, each ranging from -1 to 1. The observation space consists of 105 continuous elements with infinite ranges. These are the positions, velocities and center-of-mass based external forces for the individual body parts. The goal of the Ant is to move as fast as possible along the positive direction of the x-axis. The reward function is set up to promote this behavior as a forward reward equal to the Ant's velocity along the x-axis is given at each time-step. Another component is the Health reward, which is given at

each time step if the center of the Ant's main body remains between the hight values of $[0.2, 1.0]$ on the z-axis. Moving out of this range terminates the episode. Other components are the control cost and the contact cost, which penalize large actions and large external forces.

Chapter 3

Experiment Setup

3.1 Algorithm Implementation & Experiment Execution

For the SAC and PPO algorithm implementations the CleanRL open-source library [HDYB21] is used, links to the original implementations can be found in the Appendix sectionA.1.1. CleanRL is chosen as it provides comprehensive single file RL algorithm implementations. These are set up for compatibility with Gymnasium environments and add integration of Weights and Biasses (WandB) experiment tracking. This enables tracking and real time visualization of selected metrics during code execution[Bie20]. The WandB online dashboard provides tools for visualizing and comparing experiment results. Several projects are set up in WandB for logging the experiments conducted during this research. WandB reports are made for the projects containing the experiment results discussed and also additional experiments performed. Report links can be found in the Appendix sectionA.1.2.

For code execution Google Colab is used [Goo24] as in addition to providing access to capable computing resources, it has the main advantage of being able to run multiple instances simultaneously, enabling more experiments to be performed. However to fully utilize Colab's capabilities compute units are purchased. Compute units are spent at an hourly rate depending on how many computational resources are used. Also, as Colab executes code through Python Notebook files it requires adapting the CleanRL standard Python script to run in a Notebook file format. Other additions and changes are made to the code for adding and running the different Ant versions. The final implementations as used for the performed experiments can be found on the GitHub project page, link found in the Appendix sectionA.1.3.

3.2 Ant Variations

As required for the experiments multiple variations of the Ant environment are set up. The purpose is to have a set of similar environments or tasks derived from the original Ant which will be referred to as the 'BaseAnt' task. Most importantly the variations should promote different locomotion strategies for successful task execution. For this project two methods are considered for making the Ant variations. The first is to make changes to the Ant's body such as adding or removing legs or making changes to the body dimensions. The second is changing the Ant's surroundings with the addition of obstacles. Multiple such variations are made and tested. A selection of the variants

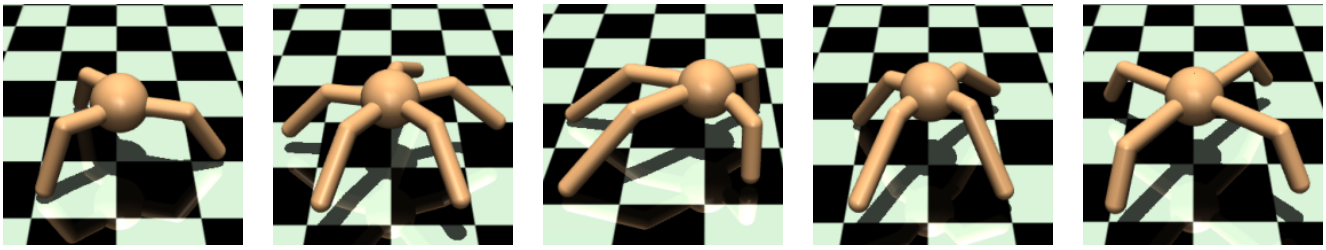
is chosen for further experimentation. The XML files for the variants can also be found on the GitHub project page.

3.2.1 Ant Body Variations

For the changes in the Ant body variations, 5 versions are made as seen in figure 4.2. The first two images 3.1a and 3.1b show the 3-legged and 5-legged Ant variants. In these cases besides removing or adding a leg, to keep the Ant stable the appendage locations are repositioned to ensure that all legs are equally spaced apart. Secondly adding or removing a leg will obviously alter the action space as each leg possesses two movable joints, the hip and ankle. As such the original environment implementation is adjusted to account for the changed action space.

For the ShortLongAnt 3.1d and LongShortAnt 3.1e the lengths of the leg sections are altered. For the ShortLongAnt the upper leg sections are halved in length and the lower sections extended by a quarter of the original size. For the LongShortAnt the upper sections are doubled in length while the lower sections are decreased by one quarter of their original size.

With the HopperAnt 3.1c the main purpose is to enforce a completely different locomotion strategy therefore multiple changes are made. Firstly the back legs are moved closer to one another at a 50 degree angle instead of 90. The hip joints rotation axis is altered to rotate around the x-axis moving the leg up and down instead of side to side and both leg sections are doubled in length. For the front legs only the lower sections are shortened by a quarter of their original size.



(a) 3LegAnt: Original Ant with one leg removed.

(b) 5LegAnt: Original Ant with one leg added.

(c) HopperAnt: Ant variant intended to resemble a grasshopper.

(d) ShortLongAnt: Original Ant with shortened upper and lengthened lower legs.

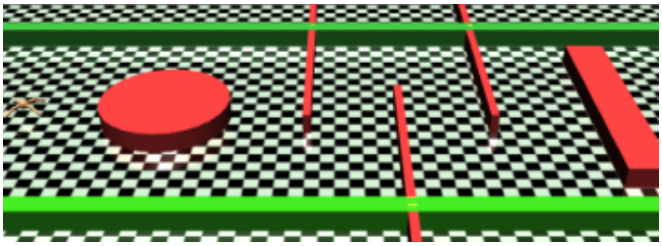
(e) LongShortAnt: Original Ant with lengthened upper and shortened lower legs

Figure 3.1: The figures show the Ant Body-Variants with the variant name and description.

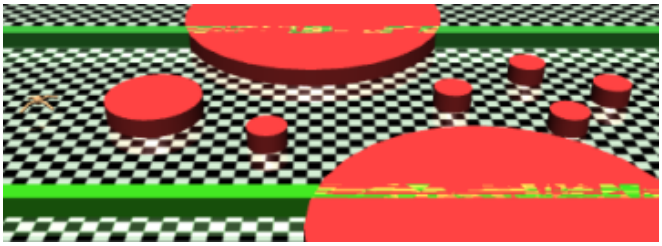
3.2.2 Ant Environment Variations

For the Ant environment variations two different types of obstacle courses are designed. As the original Ant is not aimed at navigating to a specific location, but to move as fast as possible in the positive direction of the x-axis. To have the Ant navigate the obstacle course without having to radically change the environment objective. The obstacles are placed in the way of the Ant's movement direction. The designed obstacle courses as seen in figure 4.4 also have green borders to prevent the Ant from navigating around the obstacles completely. The first type are obstacles that the Ant needs to move around. This variant is named the GoAroundAnt, with the

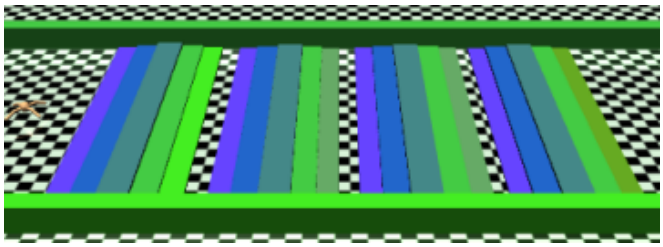
obstacle courses seen in figures 3.2a and 3.2b. The initial GoAroundAnt version presented certain challenges leading to a final course design, which will be further discussed. For the second obstacle type the ClimberAnt is designed with obstacles the Ant can only climb over as seen in figure 3.2c.



(a) Initial version GoAroundAnt: BaseAnt with the addition of the initially designed obstacle course with obstacles the Ant must navigate around.



(b) Final version GoAroundAnt: BaseAnt with the addition of the obstacle course used for experimentation with obstacles the Ant must navigate around.



(c) ClimberAnt: BaseAnt with addition of the obstacle course used for experimentation with obstacles the Ant must climb over.

Figure 3.2: The figures show the designed obstacle courses for the Ant Environment-Variants with the variant name and description.

3.2.3 Ant Variant Selection

For the variant selection the versions are trained with the SAC algorithm as it will also be used for the following experiments. Based on the performance during training, observed as the development of averaged episodic returns and the trained agent behavior as visually observed by video rendering a total of four variants are selected. The main purpose of these test runs is to see if the variants function at all, what performance measured in average episodic returns can be expected and to observe the methods of locomotion found for the variants. As compute units are limited and no performance comparison is made at this time a single training run per version test is performed for this selection process. Each variant is trained for 10^6 SAC algorithm optimization steps where with each step five environment instances are sampled in parallel making for $5 \cdot 10^6$ environment steps.

As seen in figure 3.3 the Ant body variants achieved similar or in the case of the LongShortAnt significantly better performance than the BaseAnt. Examining the video renderings of the Ant's performance it is found that all Ant variants except for the HopperAnt produced locomotion strategy very similar to that of the BaseAnt. The HopperAnt in contrast showed a completely novel method of locomotion. Although it does not resemble the grasshopper like movement aimed for, still it is selected for further experimentation as it can be interesting for testing knowledge preservation with the HopperAnt when trained from a pretrained BaseAnt model. From the remaining four variants

the ShortLongAnt is chosen to contrast the HopperAnt as it showed both similar performance and locomotion to the BaseAnt. The 3Leg and 5LegAnt were not further considered because of the added difficulty these would present with the knowledge transfer experiments due to the action-space size difference with the BaseAnt. Although this can be dealt with by only transferring the hidden layer weights for the actor network excluding the output layer and for the critics the first hidden layer would be excluded. Another possibility, for example in the case of the 3LegAnt, the variants action space can be set to 8, equal to the BaseAnt instead of the required size of 6. Where the two added action elements simply do not control any joints. However as the possible effects of these changes is unclear it restricts informative comparison with other versions and would require additional experimentation to be informative.

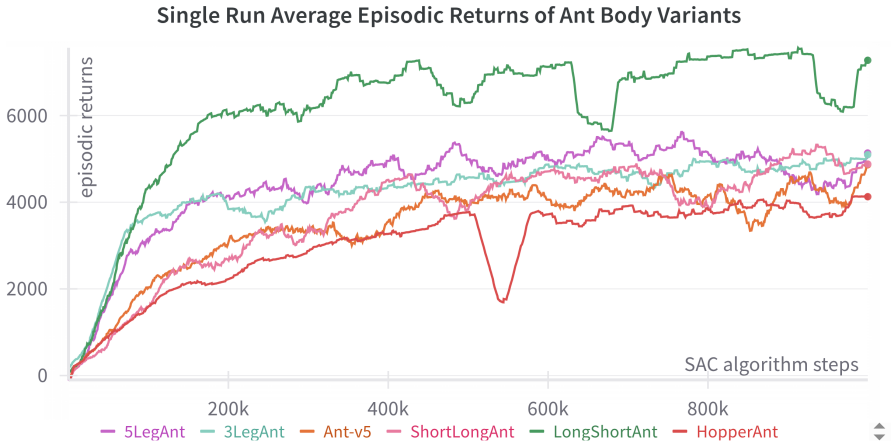


Figure 3.3: Mean episodic returns Ant Body-Variants with outliers removed and running average smoothing window of 50 time-steps. The plot shows that all variants achieved performance similar to the BaseAnt except for the LongShortAnt which shows notably higher returns.

For the ClimberAnt and the GoAroundAnt multiple tests were performed. The training steps were reduced to half the amount used for the Body-Variant testing as beyond $5 \cdot 10^5$ SAC optimization steps or $2.5 \cdot 10^6$ environment steps very little improvement in episodic returns is observed if any. The final Environment-Variant versions required accounting for obstacle dimensions and shapes as well as making changes to the reward function as defined for the BaseAnt.

The track as seen in figure 3.2a excluding the added green borders -as initially no borders were added under the assumption that the Ant would not deviate too far from the x-axis- was the first GoAroundAnt version tested. With this version episodic returns achieved during training clearly showed that the Ant was not learning as seen in figure 3.4. Running the trained model for multiple episodes with video rendering showed that the Ant never got past the first two obstacles getting blocked behind them. As the original reward function is aimed at rewarding speed an attempt was made to shift away from this goal with the following changes in reward calculation: The forward reward weight is reduced from 1 to 0.3 and an additional reward of 2 points is given at each timestep the Ant moves in the positive x-axis direction with a velocity greater than 0.3. The purpose is to promote speed to a lesser extent and instead to reward just moving forward in the intended

direction. These values were chosen based on a rough estimation of what episodic reward could be expected if the Ant performs well and if this reward does not deviate excessively from the original reward range. As the BaseAnt model trained on SAC scores an average episodic return of approximately 5500, of which 1000 are given for remaining healthy during the entire episode. The remaining 4500 return must have been achieved for the forward movement. Although this does not account for the control cost and the contact cost, but as these costs are relatively low and are neglected for the estimation. With the reduced forward weight the forward 'speed' reward would at least be reduced to $0.3 \cdot 4500 = 1350$ and would probably become lower as speed has become of less importance. The additional forward reward has a max of 2000 for an entire episode. That is if at each timestep the Ant reaches the required velocity. This should result in a total forward reward of roughly 3350. Although this is lower than estimated for the BaseAnt it is within an acceptable range to produce interpretable results and is applied for further testing.

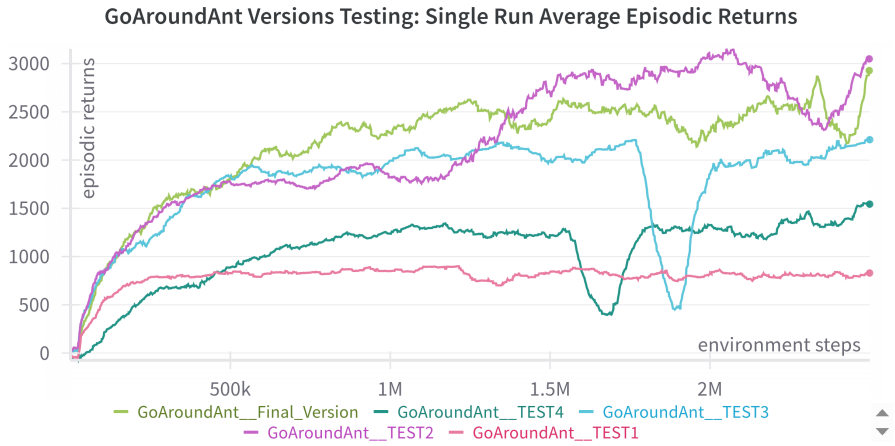


Figure 3.4: GoAroundAnt training progress results for the 5 versions tested: Mean episodic returns with outliers removed and running average smoothing window of 50 time-steps.

The second test with the adjusted reward version achieved promising results in terms of rewards but visualizing the model behavior revealed that the Ant avoided the obstacles completely by walking around them and then continue moving in the positive x-axis direction. This resulted in the addition of the green borders for the third test. Testing the bordered GoAroundAnt track again showed promising results during training but the final model behavior revealed that instead of navigating the track the Ant would move to the first wall and rock back and forth against it to achieve the 2 points granted when reaching a forward velocity greater that 0.3. To prevent this behavior the reward function was again adjusted to include a negative reward of -1 for moving in the negative x-axis direction. Unfortunately this did not change the behavior and only resulted in a reduction of episodic returns. Although with further reward function tuning a working version might be possible. This can potentially require extensive tuning of the reward calculation, possibly not achievable within the available time and resources. Therefore a different strategy is tried first, the reward function is reverted to that of the previous test and the obstacle course is adjusted to only include cylindrical obstacles for the final version on the GoAroundAnt as seen in 3.2b. As the previous reward function showed promising results with the not bordered version, using this

with round obstacles may prevent getting stuck behind a flat surface and possibly enable the Ant to learn the track. This resulted in the final version of the GoAroundAnt as the Ant was able to navigate past all obstacles and achieve adequate episodic returns.

The ClimberAnt course consists of 4 sets of bordered ascending and then descending steps that the Ant must traverse. For the initial testing only a single set was used. The first test produced low performance in terms of returns as seen in figure 3.5. Running the model showed that the Ant never got past the first 2 ascending steps. The cause was the steps were simply too high as the Ant's body must remain within the range of $[0.2, 1]$ on the z-axis, otherwise it is considered 'unhealthy' and the episode is terminated. For the second test the steps were simply lowered. Although the Ant showed good performance in this case, observing the trained model behavior the steps were too low as any traversing was hardly required. For the third test the step height was increased again and two additional changes were made. The first was to increase the vertical range limit to $[0.2, 1.6]$ enabling the Ant to climb higher. Additionally the reward function used with the GoAroundAnt was adopted for the ClimberAnt as it showed good performance both on returns as well as locomotion behavior for the GoAroundAnt and this also allows for performance comparison between the two versions. The third version showed positive results both in performance as behavior. For the fourth test three additional sets of steps were added, each set had different height dimensions and as there seemed to be room for further increasing the step height even larger steps where added. This was chosen as the final ClimberAnt version as seen in figure 3.2c as it again showed good performance and locomotion behavior.

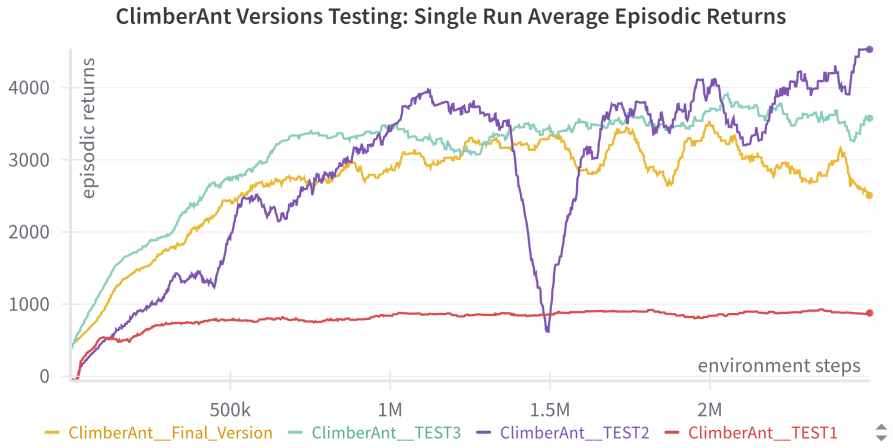


Figure 3.5: ClimberAnt training progress results for the 4 versions tested: Mean episodic returns with outliers removed and running average smoothing window of 50 time-steps.

3.3 Experiments

3.3.1 Performance Comparison SAC, PPO

Previous to the SAC and PPO performance comparison, initially with PPO several adjustments to both parameter settings and the actor critic network architectures were tried to achieve bet-

ter performance on the BaseAnt task. Although some performance gains were made, extensive hyperparameter tuning is needed for further improvement. Due to time and resource restrictions and the large amount of tunable parameters for PPO this was not achievable. The final parameter changes that showed some improvement were to sample from 4 environments in parallel instead of 1, to decrease the number of update epochs per algorithm iteration from 10 to 6 and to use a entropy coefficient of $1 \cdot 10^{-4}$ for better exploratory behavior. Additionally for both the actor and critic networks the size of the first hidden layer is increased from 64 to 128. For training the PPO algorithm was run for $5 \cdot 10^6$ environment time-steps in total. The used parameter settings and network architectures for both PPO and SAC can be found in the Appendix sectionA.2.

As PPO performance was inadequate, the SAC algorithm was tried as it has shown to perform well for most environments with the default parameter settings due to the improved exploratory behavior and high sampling efficiency. For the SAC algorithm to match the number of environment steps with that of PPO either $5 \cdot 10^6$ SAC optimization steps would be performed with a single environment sampled at each step, or 10^6 steps with 5 environments sampled in parallel. The latter was chosen as it showed better performance. The PPO, SAC comparison is done by completing 5 model training runs with 5 different random seeds for both SAC and PPO. The average episodic returns development during training are tracked and plotted for the performance comparison.

Due to an anomaly seen in the SAC training progress an additional SAC run is performed where besides the already tracked training losses and episodic returns, the policy entropy and the separate cost and reward components are also tracked in search of an explanation.

3.3.2 Knowledge Transfer Experiments

The knowledge transfer experiments are all performed using the SAC algorithm. For each of the 4 chosen Ant-Variants and the BaseAnt 5 training runs with different random seeds are performed for $5 \cdot 10^5$ optimization steps, over 5 environments in parallel, for a total of $2.5 \cdot 10^6$ environment steps. An additional 5 training runs for each variant are performed where the networks starting points are a pretrained BaseAnt model parameter weights for the actor and both critic networks and critic target networks. For all runs and variants the same BaseAnt pretrained model is used, which is also trained for $2.5 \cdot 10^6$ environment steps, with the random seed set to 1. During training in addition to the average episodic returns and training losses again the separate reward components and policy entropy are tracked. The main purpose of these runs is to determine what effect using the pretrained network has with the different variants. For this the development of episodic returns will be compared.

A second set of comparison experiments is performed with the trained models. In this case the main goal is to examine if it can be said that after training on a variant task from the pretrained model. Knowledge from the original task is preserved and in comparison to other variants to what extent it is preserved. A selection of model evaluation runs is performed for several combinations of the trained BaseAnt and variant models and tasks, for 100 episodes per run. For each run the mean and standard deviation of the episodic returns are calculated.

For each variant the following runs will be performed:

- The BaseAnt model run on the Variant environment.
- The Variant model on the Variant environment.
- The pretrained Variant model on the Variant environment.
- The pretrained Variant model on the BaseAnt environment.
- The Variant model on the BaseAnt environment.

Except for the HopperAnt the evaluation results for the variant models or pretrained variant models are the averaged mean and standard deviation aggregated over the 5 models trained with different seeds per model. The HopperAnt pretrained models showed strongly varying performance during training. Aggregating the results for the 5 models may be less informative in this case. Therefore it is decided to test both the best and worst performing pretrained models, these are the models trained on a random seed of 1 (best) and 2 (worst). To see if a significant difference can be found in terms of knowledge preservation. For completeness the 2 non pretrained variant models with corresponding seeds are also added.

A run is also performed for the BaseAnt model on the BaseAnt environment and an untrained agent on the BaseAnt environment for a baseline. To determine if the found performance differences are statistically significant, analysis is performed in the form of an independent 2-sided t-test [LY10]. The threshold value for the statistical significance is chosen to be $\alpha = 0.05$. Additionally for each variant the results are plotted in a barplot. The evaluations are not tracked in WandB, for reviewing the evaluation experiment setup the experiment Notebook file with experiment outcomes including used models and required XML files are found in the project GitHub page.

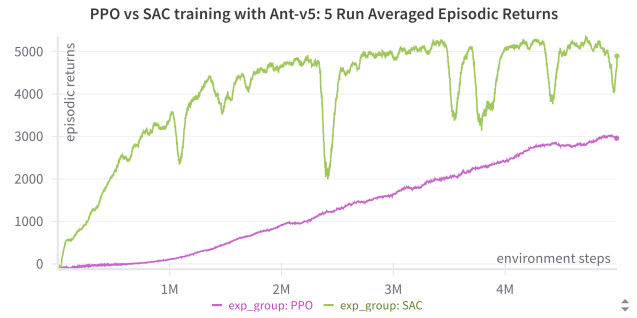
Chapter 4

Results

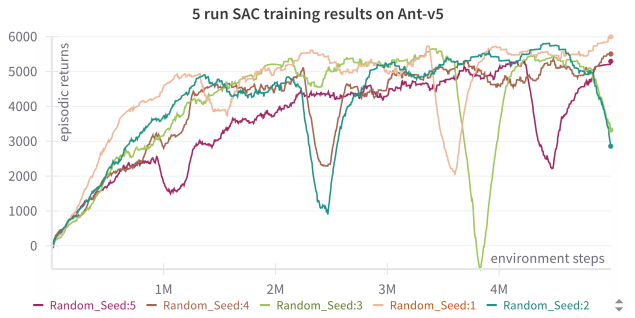
4.1 Results SAC vs PPO

As can be seen from figure 4.1a SAC clearly outperforms PPO on the BaseAnt task, reaching a performance equal to the final performance of PPO after $\pm 7 \cdot 10^5$ environment steps and training ends with a performance of ± 5000 episodic return per episode vs. ± 3000 for PPO based on the aggregated results. Although the SAC training progress shows instability in the performance progression during training as several dips in episodic returns can be seen for the aggregated results.

In figure 4.1b the plots for the separate SAC runs show that during each run a large dip in rewards occurs at some point. Possible causes may involve the temperature parameter α as results from the original SAC paper show a less prominent but similar development in SAC training for Ant-v2 and Cheetah-v2 Gymnasium environments when the parameter is tuned but not when a fixed α is used [HZAL18]. Although this is not further discussed by the authors. Another possibility is the development of an overestimation bias for the value function. However the double Q-function implemented for the critic should mitigate this to some extent as the probability of overestimation is reduced by taking the minimum of the two independent Q-function outcomes [HGS16]. The run performed with SAC additionally tracking the policy entropy and separate reward components as discussed in section3.3.1 did not show any interesting results as none of the tracked metrics showed an anomalous development preceding the dip in rewards. This observation renders the possible causes mentioned to be unlikely. Another possibility may involve the replay buffer, as the replay buffer size can severely affect learning[ZS17]. If the buffer is to large stored transitions can become outdated, resulting in a discrepancy in learned behavior and experiences sampled from the buffer in future update steps. Newly sampled, possibly critical transitions have a lower probability of being replayed with increasing buffer size and this may result in slower learning. On the other hand if the replay buffer is too small it may result in highly correlated samples causing overfitting to more recent experiences, destabilizing learning. To find a conclusive explanation for the irregular training progression will require a more in depth investigation. However as previously stated the sudden dip in reward does not seem to affect the overall learning progression, as the policy relatively quickly recovers after the dip and learning continues positively. Therefore if a similar progression is found in following experiments this should not prevent reliable inference from the found results. SAC is therefore chosen for further experimentation as it performs significantly better than PPO with the Ant environment.



(a) PPO, SAC performance comparison for the BaseAnt: 5 Runs Averaged Mean episodic returns with outliers removed and running average smoothing window of 50 time-steps. The plot shows that SAC is superior to PPO in terms of performance but may be lacking in stability.



(b) SAC training results for 5 runs with different random seeds: Run Averaged Mean episodic returns with outliers removed and running average smoothing window of 50 time-steps. The plots show that the 5 runs each have prominent dips in episodic returns during training but overall show a similar progression in performance.

Figure 4.1

4.2 Results Knowledge Transfer Experiments

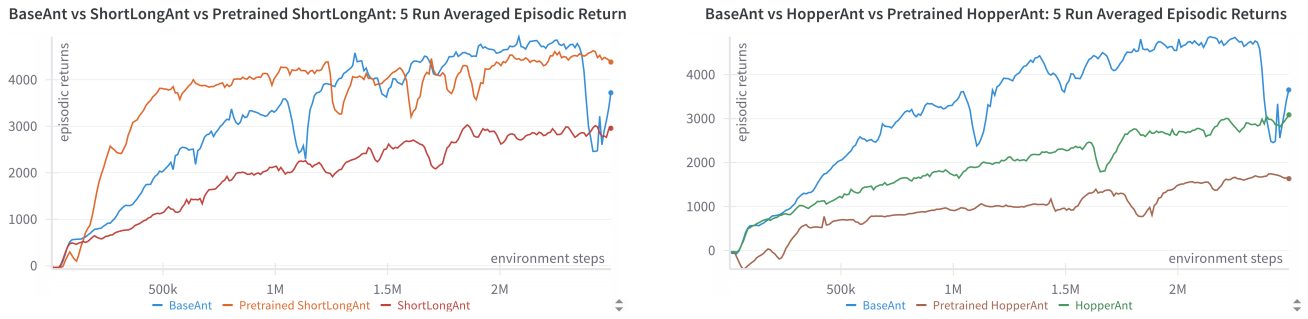
4.2.1 Training Results

In this section the training results for the chosen variants -ShortLongAnt, HopperAnt, ClimberAnt and GoAroundAnt- will be discussed. Comparing the training progress of the agents when pretrained on the BaseAnt environment against the progress when no pretraining is applied. Additionally the locomotion behavior of the resulting models will also be discussed.

Figure 4.2 shows the plots with the aggregated training results for the ShortLongAnt and HopperAnt. For the ShortLongAnt as seen in plot 4.2a pretraining has resulted in accelerated learning. With pretraining the final performance of the non pretrained model of ± 3000 episodic return is reached after $\pm 3.5 \cdot 10^5$ environment steps and training ends with returns of ± 4500. The pretrained version even outperforms the BaseAnt for approximately the first half of training and maintains a similar progression but slightly lower returns afterwards. The ShortLongAnt locomotion behavior as observed during testing in section 3.2.3 was practically identical to that of the BaseAnt. Among the 5 following models trained for experimentation, for 3 of the 5 runs a different method of locomotion is observed than seen with the BaseAnt. For the BaseAnt models trained sofar in all cases the same overall locomotion strategy is observed. And for the pretrained version in all 5 cases the Ants move in a way practically identical to the BaseAnt. This can be interpreted as that the BaseAnt has the highest probability to end up in a certain local or possibly the global optimum. For the ShortLongAnt another local optimum has a possibly similar probability to be found. Therefore with Pretraining the ShortLongAnt is forced into a (in this case better) optimum resulting in a similar mode of locomotion as found for the BaseAnt.

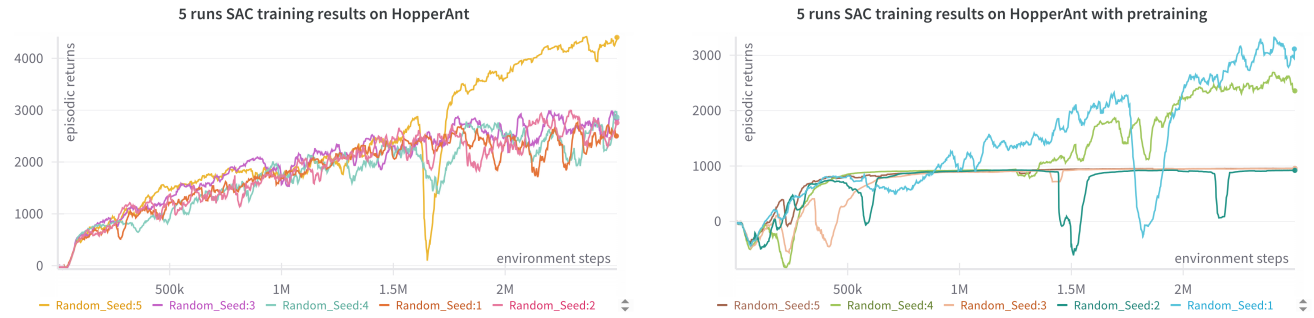
For the HopperAnt as seen in plot 4.2b pretraining has not been beneficial. The pretrained HopperAnt reaches returns of ±1500 whereas with no pretraining returns of ±3000 are achieved. The BaseAnt finishes training with returns of ±4800 (roughly extrapolating the results before the dip in returns occurs at the end of training). Viewing the results for the separate HopperAnt training runs shown in figure 4.3. It is found that for the non pretrained HopperAnt as seen in plot 4.3a all runs have a similar progression except for Random_Seed:5 where after $\pm 1.7 * 10^6$ environment steps performance increases more rapidly reaching higher final returns. Observing the locomotion behavior of the different models it is seen that for seeds: 1 to 4 a similar strategy is found. Only the best performing seed: 5 shows distinctly different movements. So far three different locomotion methods are observed for the HopperAnt as the initial model trained for the variant selection showed yet another distinct method. The BaseAnt sofar has produced a single means of locomotion and achieves far better performance than the HopperAnt. This could either mean that the HopperAnt's morphology is less suited for the environment objective than the BaseAnt. Or that better performing policies exist in the policy space, but have a low probability of being discovered.

For the pretrained HopperAnt a larger difference in performance is seen between runs as shown in plot 4.3b. With seeds: 2, 3 and 5 improvement is seen during early training, after which the performance flattens below ±1000 episodic returns. With seeds: 1 and 4 the performance is similar to the majority of the non pretrained runs. Examining the different models behaviors it is found that in the case of the three worst performers the Ant simply does not move from its starting position for the entire episode. Achieving only the health reward minus the control and contact costs with each step. For seeds: 1 and 4 again a new locomotion strategy is seen. Most interestingly with the non pretrained models in all cases at the very start the Ant flips on its back before moving forward while with pretraining this is not the case. This can be interpreted as that policies with greater similarity to the BaseAnt policy used for pretraining now have a higher probability of being learned even if they produce significantly poorer results.



(a) (b)

Figure 4.2: Plots (a) ShortLongAnt vs. Pretrained ShortLongAnt vs. BaseAnt and (b) HopperAnt vs. Pretrained HopperAnt vs. BaseAnt, SAC training progress performance comparison: 5 Runs per version Averaged Mean episodic returns with outliers removed and running average smoothing window of 50 time-steps. Plot (a) shows that the pretrained ShortLongAnt clearly outperforms the non pretrained version, showing better performance than the BaseAnt for the first half of the training and similar performance for the second half. Plot (b) shows that pretraining for the HopperAnt overall has not been beneficial and results in lower returns.

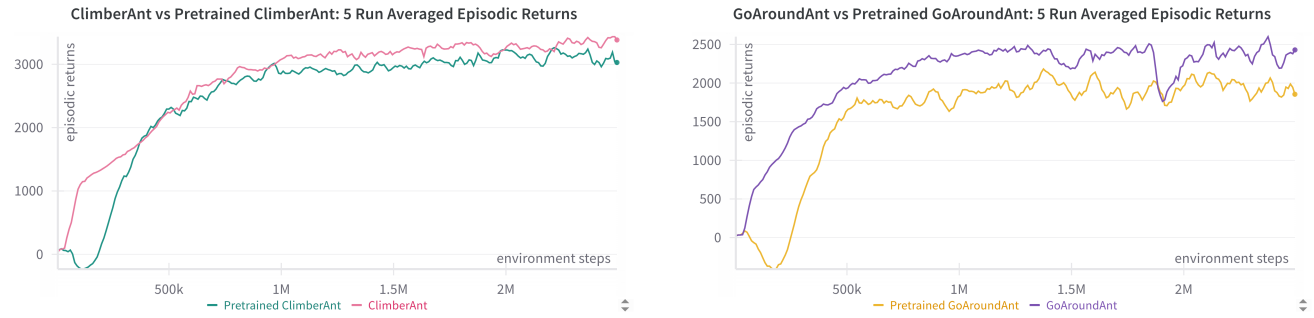


(a) (b)

Figure 4.3: Plots (a)HopperAnt, (b)Pretrained HopperAnt : 5 separate runs mean episodic returns with outliers removed and running average smoothing window of 50 time-steps. Plot (a) shows that without pretraining a similar progression is seen for the different seeds except for Random_seed:5 where after recovering from a sudden dip in rewards learning is accelerated. Plot (b) shows that with pretraining the HopperAnt for seeds 2, 3 and 5 achieves poor performance, while for seeds 1 and 4 the performance is similar to what is seen without pretraining.

For the ClimberAnt and the GoAroundAnt figure 4.4 shows the aggregated training results for the 5 training runs. The BaseAnt performance is not added to the plots as the reward function for these variants is different and a direct comparison with the BaseAnt would not be informative. The plots show that for both environment variants pretraining has not been beneficial. With pretraining both variants show negative returns at the start of training, after which the returns rapidly improve. In both cases this is mainly caused by high control costs. For the ClimberAnt the performance even seems to match that of the runs with no pretraining but after $\pm 5 \cdot 10^5$ it shows slightly lesser performance for following training steps. The GoAroundAnt has clearly lower returns during the entire run when pretrained. Comparing the two variants the ClimberAnt reaches higher returns overall. As both Ants use the same reward function this could imply that walking around obstacles is simply a more difficult task to perform. Although it is important to account for the fact that the Ants have no direct observation of the added obstacles. Any information about the track must be inferred from the original observation space. It can be argued that observing the obstacles is of lesser importance for the ClimberAnt as its track can be walked in a straight line while the GoAroundAnt must avoid obstacles by moving along the y-axis direction. Although the required sideway movement itself will also cause lower forward rewards.

Assessing the locomotion behavior of the final models. For the ClimberAnt both the pretrained and non pretrained models show very similar movements to the BaseAnt especially when propelling forward on flat areas of the track. The non pretrained Ant shows the more unique movement when moving over the obstacles while the pretrained version here also moves in a way very similar to the BaseAnt. The GoAroundAnt shows similar results as the only noticeable difference in this case, is that the non pretrained models seem more intentional in avoiding the obstacles. For both variants it seems that some residual behavior specific to the BaseAnt is retained and has negatively impacted learning.



(a) (b)

Figure 4.4: (a)ClimberAnt vs. Pretrained ClimberAnt, (b) GoAroundAnt vs. Pretrained GoAroundAnt, SAC training progress performance comparison: 5 Runs per version, Averaged Mean Returns with outliers removed and running average smoothing window of 50 time-steps. The plots show that for both the ClimberAnt and GoAroundAnt pretraining results in lower performance.

4.2.2 Model Evaluation Results

In this section the models resulting from the training runs previously discussed are evaluated. The variant models are also evaluated on the BaseAnt environment and the BaseAnt model used for pretraining is also evaluated on all variant environments. The goal is to assess the effect of starting training using model weights of a model trained for the BaseAnt task. Secondly the non pretrained and pretrained variant models are also run on the BaseAnt task to asses wether any evidence of the model being able to generalize to both tasks and as such if it can be said that knowledge from the BaseAnt model is preserved after training on the variant task. The evaluation results are found in table 4.1 and visualized with the barplots in figure 4.5. Table 4.2 contains calculated statistics, assessing the confidence of the results withing a 95% confidence interval.

With the HopperAnt a large variation in performance was seen for the pretrained models. Therefore only the best and worst performing models are evaluated as combining the results for all 5 would not be informative. The barplot 4.5b shows that the 'best performer' pretrained model has slightly higher returns than the non pretrained models. However as the found differences are relatively small and also for example comparing the two chosen non pretrained models, the model with seed:2 shows negative episodic returns with a mean of -932.73 while seed:1 shows mean returns of 43.17, although these two models have shown very similar performance otherwise. Therefore single model results are considered unreliable for direct one on one comparison and further statistics are also not added. The main interesting feature of the results in this case is how the variant models perform when evaluated on the BaseAnt task. As for the HopperAnt for these evaluations very poor performance is found, which is even more evident when compared to the other variants. The difference in morphology in this case seems to restrict possible overlap in policy with the BaseAnt model. For both the training and evaluation experiment pretraining has shown least beneficial for the HopperAnt even though in terms of task definition, it can be considered closer to the BaseAnt task than the ClimberAnt and GoAroundAnt.

In contrast the ShortLongAnt has shown very promising results. In this case pretraining results in a significant improvement of 46.76% in episodic returns. The pretrained model also showed 13.65% better performance on the BaseAnt task as seen in table 4.2. Thus pretraining not only improved performance but also produced a model that proved more capable on the BaseAnt task as without pretraining. For the evaluation of the BaseAnt model (which was used for the pretraining) on the ShortLongAnt task, the barplot 4.5a shows a mean performance more or less equal to that of an untrained agent although compared to the other variants it is the best result found for this case. However this still indicates that the BaseAnt model used was in no way suitable for the ShortLongAnt task but still resulted in a significant performance improvement.

For the ClimberAnt and GoAroundAnt a very similar results pattern is seen in the barplots 4.5c and 4.5d. In both cases pretraining resulted in a slight but statistically significant decrease in performance. Without pretraining the ClimberAnt achieved 8.26% higher episodic returns and for the GoAroundAnt 16.95% better returns are found. For the variant models evaluation on the BaseAnt task the performance difference of the models with vs. without pretraining is small and statistically insignificant based on the p-values calculated. Also note worthy is that for both variants, evaluation on the BaseAnt task for pretrained and non pretrained models show results similar to that of the ShortLongAnt. This can be evidence that as a consequence of the shared morphology, the policies learned on these variants naturally generalize to the BaseAnt task, despite the different reward functions and the changes made to the environment.

Evaluation Scenario	**BaseAnt**	**ClimberAnt**	**HopperAnt**	**GoAroundAnt**	**ShortLongAnt**
Untrained Agent	644.82 ± 188.28	-	-	-	-
BaseAnt Model	5607.20 ± 650.96	-	-	-	-
BaseAnt Model on Variant Env	-	-195.77 ± 474.53	-199.36 ± 347.70	-160.15 ± 1069.22	683.74 ± 609.80
Variant Model	-	3254.68 ± 1190.69	2887.35 ± 696.50 (V1) / 3090.16 ± 903.00 (V2)	2354.80 ± 1347.36	3249.82 ± 1182.66
Pretrained Variant Model	-	2985.76 ± 1373.65	3268.69 ± 905.20 (P1) / 927.76 ± 3.33 (P2)	1955.56 ± 1387.56	4769.47 ± 1007.28
Pretrained Variant Model on BaseAnt Env	-	3878.96 ± 1438.61	36.64 ± 160.65 (P1) / 547.00 ± 8.39 (P2)	3414.16 ± 752.71	3347.60 ± 1102.35
Variant Model on BaseAnt Anv	-	4007.02 ± 1167.40	43.17 ± 196.41 (V1) / -932.73 ± 304.55 (V2)	3513.58 ± 954.07	2945.43 ± 965.44

Table 4.1: Table with the performance evaluation results Mean ± Standard Deviation of Episodic Returns over 100 episodes per model evaluation. Where for the ShortLongAnt, ClimberAnt and GoAroundAnt evaluations involving the Variant and Pretrained-Variant models show aggregate results for 5 separate models trained with different seeds thus based on a total of 500 samples. Al other evaluations (untrained agent, BaseAnt and HopperAnt evaluations) are single model evaluations. Each row corresponds to a specific evaluation scenario: untrained agent on the BaseAnt environment, BaseAnt model on the variant environments, variant models trained from scratch or with pretraining, and models evaluated outside their original training environment. For the HopperAnt, results are shown for both the best (P1) and worst (P2) performing pretrained models and the regular Hopper models (V1 and V2) with corresponding seeds: 1 and 2.

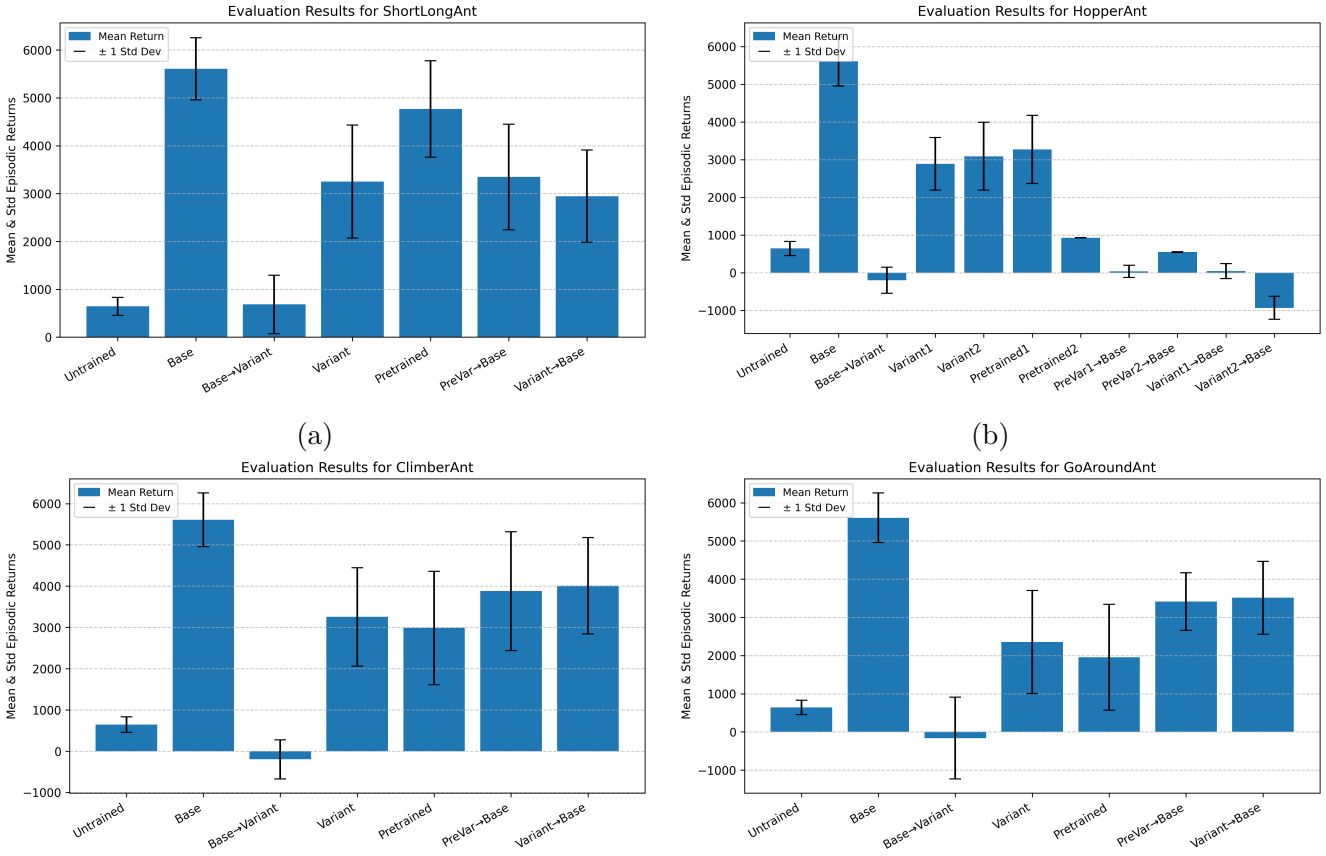


Figure 4.5: Model performance comparison for the selected Ant variant models. The barplots show the mean an the 1 standard deviation range for each evaluation. Where the BaseAnt Variant and Pretrained Variant models are evaluated on their origin environments and also evaluated outside the original environment. Where pretrained variant (PreVar) and Variant are evaluated on the BaseAnt environment and the BaseAnt model on the Variant environment. From these barplots it can be seen that only for the ShortLongAnt pretraining resulted in a notable performance improvement and also resulted in better performance of the variant model on the BaseAnt task.

Scenario	**Comparison**	**Mean A**	**Mean B**	**Mean Diff**	**% Diff**	**95% CI**	**95% CI %**	**t-stat**	**p-value**
ClimberAnt	PreVar vs Var	2985.76	3254.68	-268.92	-8.26%	[-428.46, -109.38]	[-13.16%, -3.36%]	-3.31	0.00097
ClimberAnt	PreVar vs Var on BaseEnv	3878.96	4007.02	-128.06	-3.20%	[-290.66, 34.54]	[-7.25%, 0.86%]	-1.55	0.12253
GoAroundAnt	PreVar vs Var	1955.56	2354.80	-399.24	-16.95%	[-568.97, -229.51]	[-24.16%, -9.75%]	-4.62	4.43×10^{-6}
GoAroundAnt	PreVar vs Var on BaseEnv	3414.16	3513.58	-99.42	-2.83%	[-206.08, 7.24]	[-5.87%, 0.21%]	-1.83	0.06766
ShortLongAnt	PreVar vs Var	4769.47	3249.82	1519.65	46.76%	[1383.31, 1655.99]	[42.57%, 50.96%]	21.87	1.38×10^{-86}
ShortLongAnt	PreVar vs Var on BaseEnv	3347.60	2945.43	402.17	13.65%	[264.08, 540.26]	[8.97%, 18.34%]	5.57	6.16×10^{-8}

Table 4.2: Statistical comparison results between Pretrained Variant (PreVar) and Variant (Var) across different Ant environments. **Mean A** and **Mean B** are the average returns for PreVar and Var. **Mean Diff** is the difference between the means (A - B), and **% Diff** is this difference expressed as a percentage of Mean B. **95% CI** gives the 95% confidence interval for the mean difference, and **95% CI %** expresses the same interval as a percentage of Mean B. **t-stat** is the t-statistic from Welch's t-test, and **p-value** indicating the statistical significance of the difference. The found p-values show that all comparison results are statistical significant except for the GoAroundAnt and ClimberAnt evaluations on the BaseAnt environment and that only for the ShortLongAnt the pretrained model showed an improvement in performance.

Chapter 5

Conclusions and Further Research

With this project the possible benefits and limitations of knowledge transfer in Deep-RL are researched in context of robotic locomotion tasks with continuous state-action spaces. Specifically, the effectiveness of transferring knowledge from a model trained on the BaseAnt environment to variant tasks with morphological changes or environment alterations is tested.

Comparing the performance of the PPO and SAC RL algorithms on the BaseAnt environment, SAC was chosen for further experimentation as it achieved significantly better performance. Although PPO showed more stable training progress and with structured parameter finetuning may achieve improved results.

The experiment results show evidence that the effectiveness of knowledge transfer is mainly determined by morphological similarities between the Ant variants and the BaseAnt. The ShortLongAnt, for which only slight changes were made in the Ant's leg dimensions showed improved model performance with pretraining on the BaseAnt environment. The pretrained ShortLongAnt models also showed improved performance when applied to the BaseAnt environment. In contrast the HopperAnt, for which the morphology was significantly altered pretraining was not beneficial. As three of the five models trained with pretraining performed very poorly and the other two showed performance similar to the non pretrained models. Even though pretraining showed promising results with the ShortLongAnt, comparing the behaviors produced by the pretrained and non pretrained models it is seen that pretraining potentially enforced a policy specific to a certain region of the policy space to be learned. Although this has proved beneficial for the ShortLongAnt, it can be argued that in some cases the limited exploration of the policy space can result in suboptimal policies.

For the ClimberAnt and GoAroundAnt despite having the same morphology as the BaseAnt a reduction in performance is seen when pretraining is applied. However there were no cases where pretraining resulted in a policy that completely failed to produce viable behavior as with the HopperAnt. As for the performance of the trained models on the BaseAnt environment no significant effect was found. These observations further support the claim that prior knowledge can direct learning towards a policy that retains behaviors better suited for the source task but results in suboptimal performance on the target task.

This leads to the following answers to the research questions:

- RQ 1. How well does learned knowledge generalize to similar but not identical continuous state-action space robotic locomotion tasks?

  Knowledge transfer can be effective when the source and target tasks are similar. Both in terms of the robotic agents morphology as the performed task, as seen with the ClimberAnt. The HopperAnt for which the morphology was more severely altered, produced policies that failed to perform any method of locomotion in some cases. For the GoAroundAnt and ClimberAnt the morphology was not changed but significant changes to the environment and reward function where made. Although these variants showed no improvement with pretraining, the performance was not strongly diminished either. This shows that for the performed experiments mainly the robotic agents morphology has been determinant for the effectiveness of the knowledge transfer application.

- RQ 2. Is knowledge from the source task preserved after training on the target task?

  An indication of knowledge being preserved from the source task after training on the target task is only seen with the ClimberAnt. For the HopperAnt, none of the variant models (pretrained or not) showed performance exceeding that of an untrained agent. Whereas for the GoAroundAnt and ClimberAnt no significant difference was found for the pretrained and non pretrained models. Thus it is concluded that some knowledge from the source task can be preserved, achieving better applicability to the source task with a pretrained target task model. Although this is only achieved when source and target task are adequately similar both in terms of morphology and the performed task.

The experiment results have shown that transferring learned knowledge in context of the robotic locomotion task can be beneficial depending on task similarity. Where changed morphology has shown the strongest effect on performance. However the experiments performed have proven limited in producing conclusive answers for the proposed research questions. In future research for more conclusive results a larger set of variants and more extensive testing of the training progression and model performance is required. Also a measure of the variant task similarity to the base environment is needed for adequate interpretation of the results [CS05]. As for the experiment with the GoAroundAnt and ShortLongAnt a shortcoming of these variants is that there was no direct observation of the obstacles added in the environment. Particularly for the GoAroundAnt this has limited the Ant's adaptation to the changed environment. For future research the addition of visual observation using Convolutional Neural Networks (CNNs)[ON15], for tasks that require interaction with objects in the environment can be considered.

Bibliography

[Ben19] Eric Benhamou. Variance reduction in actor critic methods (ACM). *ArXiv*, 2019.

[Bie20] Lukas Biewald. Experiment tracking with weights and biases, 2020. Software available from https://www.wandb.com.

[CS05] James L. Carroll and Kevin Seppi. Task similarity measures for transfer in reinforcement learning task libraries. *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005*, 2:803–808, 2005.

[FHM18] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *ArXiv*, 2018.

[FLHI+18] Vincent Francois-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *ArXiv*, 2018.

[Goo24] Google. Google colaboratory, 2024. Accessed: 2025-05-30.

[HDYB21] Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, and Jeff Braga. CleanRL: High-quality single-file implementations of deep reinforcement learning algorithms. *ArXiv*, 2021.

[HGS16] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), 2016.

[HTAL17] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. *ArXiv*, 2017.

[HZAL18] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *ArXiv*, 2018.

[KW13] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *ArXiv*, 2013.

[LY10] Zhenqiu Lu and Ke-Hai Yuan. *Welch's t test*, pages 1620–1623. 01 2010.

[ON15] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *ArXiv*, 2015.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2nd edition, 2018.

[SML+15] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *ArXiv*, 2015.

[SWD+17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *ArXiv*, 2017.

[TAH+24] Chen Tang, Ben Abbatematteo, Jiaheng Hu, Rohan Chandra, Roberto Martín-Martín, and Peter Stone. Deep reinforcement learning for robotics: A survey of real-world successes. *ArXiv*, 2024.

[TET12] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[TKT+24] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G Younis. Gymnasium: A standard interface for reinforcement learning environments. *ArXiv*, 2024.

[ZLJZ20] Zhuangdi Zhu, Kaixiang Lin, Anil K Jain, and Jiayu Zhou. Transfer learning in deep reinforcement learning: A survey. *ArXiv*, 2020.

[ZS17] Shangtong Zhang and Richard S Sutton. A deeper look at experience replay. *ArXiv*, 2017.

Appendix A

A.1 Links: Results, Code and Experimentation

A.1.1 CleanRL PPO & SAC

- Link to CleanRL PPO for continuous action space implementation: https://github.com/vwxyzjn/cleanrl/blob/master/cleanrl/ppo_continuous_action.py
- Link to CleanRL SAC implementation: https://github.com/vwxyzjn/cleanrl/blob/master/cleanrl/sac_continuous_action.py

A.1.2 WandB

Experiment results can be found via the following WandB report links:

- Results Testing of Ant Body-Variants and additional SAC, PPO runs. https://api.wandb.ai/links/s0772151-leiden-university/qy5w5byq
- Ant Environment-Variants Testing and Final PPO SAC comparison. https://api.wandb.ai/links/s0772151-leiden-university/tnypc8hv
- Selected Ant variants training and training from pretrained BaseAnt model experiment results: https://api.wandb.ai/links/s0772151-leiden-university/167lpr93

A.1.3 Github

Experiment code, models, evaluation experiment results and video visualizations can be found at: Github repository: https://github.com/Ovitch87/Knowled_transfer_RL_Ant#

A.2 PPO & SAC networks and parameter settings

A.2.1 PPO settings

PPO actor critic networks:

```
Agent(
  (critic): Sequential(
    (0): Linear(in_features=105, out_features=128, bias=True)
    (1): Tanh()
    (2): Linear(in_features=128, out_features=64, bias=True)
    (3): Tanh()
    (4): Linear(in_features=64, out_features=1, bias=True)
  )
  (actor_mean): Sequential(
    (0): Linear(in_features=105, out_features=128, bias=True)
    (1): Tanh()
    (2): Linear(in_features=128, out_features=64, bias=True)
    (3): Tanh()
    (4): Linear(in_features=64, out_features=8, bias=True)
  )
)
```

PPO parameter settings:

- anneal_lr: true
- batch_size: 8,192
- clip_coef: 0.2
- clip_vloss: true
- cuda: true
- ent_coef: 0.0001
- gae_lambda: 0.95
- gamma: 0.99
- learning_rate: 0.0003
- max_grad_norm: 0.5
- minibatch_size: 256
- norm_adv: true
- num_envs: 4

- num_iterations: 610
- num_minibatches: 32
- num_steps: 2,048
- target_kl: None
- torch_deterministic: true
- total_timesteps: 5,000,000
- update_epochs: 6
- vf_coef: 0.5

A.2.2 SAC settings

SAC actor critic networks:

```
Actor(
  (fc1): Linear(in_features=105, out_features=256, bias=True)
  (fc2): Linear(in_features=256, out_features=256, bias=True)
  (fc_mean): Linear(in_features=256, out_features=8, bias=True)
  (fc_logstd): Linear(in_features=256, out_features=8, bias=True)
)
SoftQNetwork(
  (fc1): Linear(in_features=113, out_features=256, bias=True)
  (fc2): Linear(in_features=256, out_features=256, bias=True)
  (fc3): Linear(in_features=256, out_features=1, bias=True)
)
```

SAC parameter settings:

- alpha: 0.2
- autotune: true
- batch_size: 1,024
- buffer_size: 1,000,000
- gamma: 0.99
- learning_starts: 5,000
- num_envs: 5
- policy_frequency: 2
- policy_lr: 0.0003

- q_lr: 0.001
- target_network_frequency: 1
- tau: 0.005
- total_timesteps: 1,000,000