METRA: Scalable Unsupervised RL with Metric-Aware Abstraction

Leonard Yu (ly4431@princeton.edu), Eugene Choi (ec0342@princeton.edu), Alice Hou (ah5087@princeton.edu)
GitHub Repository

May 27, 2024

1 Introduction

Reinforcement learning (RL) algorithms have risen to prominence due to their capabilities in training agents in complex tasks like video games (Mnih et al., 2015) or robot locomotion (Schulman et al., 2015) when provided with extrinsic reward functions. However, supervised rewards are either engineered or provided as labels by human operators, which makes it hard to scale RL agents (Kostrikov et al., 2020). Additionally, these agents are typically only able to solve their intended task, with insufficient generalization to other downstream tasks . In the fields of computer vision and natural language processing, unsupervised pre-training has seen successes in enabling few-shot learning and more generalizable algorithms (Laskin et al., 2021). Similarly, unsupervised RL could enable a more efficient process of discovering behaviors, which would allow agents to learn a wider range of downstream tasks. The development of generalist RL agents paves a path for the broad application of RL on real world problems, where environment are volatile and agents must operate in diverse and dynamic settings.

Current approaches in unsupervised RL can be categorized into two main groups: pure exploration and unsupervised skill discovery. However, pure exploration methods may not be scalable to large or complex state spaces, and unsupervised skill discovery methods have limited state coverage and may fail to completely explore the environment (Park et al., 2024). As such, formulating unsupervised RL remains a major open problem.

In this paper, we will reproduce METRA, which is a novel unsupervised RL objective that can scale to environments with high intrinsic dimensionality such as complex image- and pixel-based environments. METRA aims to learn diverse behaviors that cover a compact latent metric space instead of the original state space. To ensure scalability in complex environments, this latent space is based on the temporal distance metric (the minimum number of steps between two states) (Park et al., 2024).

First, we will propose a novel, metric-aware unsupervised RL objective based on the Wasserstein dependency measure (WDM). Second, we will define the temporal distance metric and provide a more tractable method to maximize the objective by introducing a practical simplification. Lastly, by performing experiments in a variety of locomotive environments, we will demonstrate that our implementation of METRA is able to learn diverse tasks without having to learn a separate set of policies.

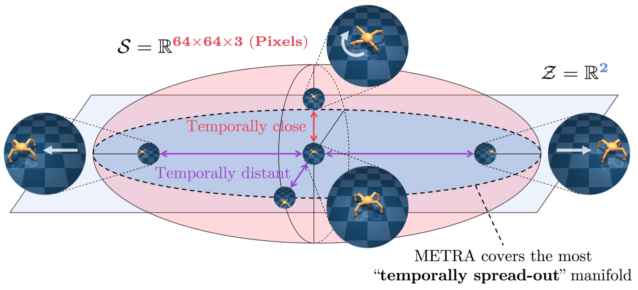

$\mathcal{S} = \mathbb{R}^{64\times64\times3}$ (Pixels)
$\mathcal{Z} = \mathbb{R}^2$
Temporally close
Temporally distant
METRA covers the most "**temporally spread-out**" manifold

Figure 1. Rather than focusing on covering every possible configuration of leg poses, METRA focuses on an approximate coverage of the state space. As depicted above in the pixel-based Quadruped environment, a 2-dimensional $\mathcal{Z}$ space can capture behaviors that run in all directions.

2 Related Work

Previous work can be categorized into two main groups: pure exploration and unsupervised skill discovery. Broadly, these attempts aim to tackle the unsupervised RL problem, with the objective of

Algorithm 1 Metric-Aware Abstraction (METRA)

```
1: Initialize skill policy $\pi(a|s,z)$, representation function $\phi(s)$, Lagrange multiplier $\lambda$, replay buffer $\mathcal{D}$
2: for $i \leftarrow 1$ to (# epochs) do
3:     for $j \leftarrow 1$ to (# episodes per epoch) do
4:         Sample skill $z \sim p(z)$
5:         Sample trajectory $\tau$ with $\pi(a|s,z)$ and add to replay buffer $\mathcal{D}$
6:     end for
7:     Update $\phi(s)$ to maximize $\mathbb{E}_{(s,z,s')\sim\mathcal{D}}[(\phi(s') - \phi(s))^\top z + \lambda \cdot \min(\varepsilon, 1 - \|\phi(s) - \phi(s')\|_2^2)]$
8:     Update $\lambda$ to minimize $\mathbb{E}_{(s,z,s')\sim\mathcal{D}}[\lambda \cdot \min(\varepsilon, 1 - \|\phi(s) - \phi(s')\|_2^2)]$
9:     Update $\pi(a|s,z)$ using SAC (Haarnoja et al., 2018a) with reward $r(s,z,s') = (\phi(s') - \phi(s))^\top z$
10: end for
```

Figure 2. Overview of the METRA algorithm pseudocode.

obtaining useful knowledge of the environment in an unsupervised manner. This knowledge can then be applied to solve downstream tasks. However, to the best of our knowledge, there currently does not exist an unsupervised RL method that is able to successfully discover locomotive behaviors in pixel-based environments due to the size and complexity of the state spaces.

Latent Explorer Achiever (LEXA) is a pure exploration method that leverages both an explorer and achiever to maximize epistemic uncertainty. Based on an imagined model of the environment, the explorer maps out action sequences with the goal of finding novel states that would maximize the information learned. The achiever is then trained on these rollouts (Mendonca et al., 2021). Pure exploration methods like LEXA attempt to cover the entire state space to learn a complete model of the environment and its transition dynamics, but this may be infeasible in larger, more complex spaces.

At present, the most common unsupervised skill discovery approach is to maximize the mutual information (MI) between states and skills. However, because MI is defined using the KL divergence, this objective is overly reliant on how distinguishable behaviors are, which results in limited state coverage (Park et al., 2024). There have been attempts to combine MI with exploration bonuses, such as the Explore, Discover and Learn (EDL) approach. EDL operates in three stages: exploration, skill discovery, and skill learning; in contrast to other MI methods, EDL encourages the agent to reinforce already-discovered behaviors (Campos et al., 2020). However, like many other unsupervised skill discovery methods, EDL attempts to either capture an entire MDP or assume a compact state space, making it averse to scaling to larger and more complex environments (Park et al., 2024).

The novel METRA objective attempts to address the limitations of both aforementioned methods by introducing the WDM objective (Park et al., 2024). There also exists previous work that use similar constrained objectives to that of the concise METRA objective. For instance, Lipschitz-constrained Skill Discovery (LSD) encourages the agent to maximize traveled Euclidean distances and, similar to METRA, provides a method for the agent to reach the goal-state in a zero-shot manner (Park et al., 2022). However, LSD fails to scale to larger and more complex environments due to limitations imposed by the Euclidean distance metric. METRA aims to address this by using the latent temporal distance metric.

3 Methods

To create a scalable unsupervised RL objective, we will address two main objectives: first, taking into consideration the complexity of the environment, METRA leverages a compact latent space $\mathcal{Z}$ and a latent-conditioned policy. Second, to maximize state coverage, METRA aims to ensure that behaviors between different latent vectors are also different.

Table 1
Hyperparameters

Hyperparameter	**Ant**	**HalfCheetah**	**Quadruped**
Latent low	-1	1	-1
Discrete	False	True	False
Latent high	1	16	1
Latent dimension	2	1	4
Number of epochs	10,000	10,000	10,000
Batch size	256	256	256
Environment name	Ant-v4	HalfCheetah-v4	Quadruped
Lambda (λ)	30.0	30.0	30.0
Learning rate (lr)	0.0001	0.0001	0.0001
Optimizer	Adam	Adam	Adam
Episodes per epoch	8	8	8
Gradient steps per epoch	50	50	200
Minibatch size	256	256	256
Epsilon (ϵ)	0.001	0.001	0.001
Checkpoint epoch	100	100	100
Encoder	False	False	True
Layers	N/A	N/A	3
Pixel	False	False	True

3.1 Problem Setting

Consider a MDP without a reward function $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mu, p)$ where $\mathcal{S}$ denotes the state space, $\mathcal{A}$ denotes the action space, μ denotes the initial state distribution, and p denotes the transition dynamics kernel. Also consider a set of latent vectors $z \in \mathcal{Z}$ and a latent-conditioned policy $\pi(a|s, z)$, where a latent vector and its corresponding policies are jointly denoted as *skills.*

To sample a trajectory, first sample a skill from the prior distribution $z \sim p(z)$ and roll out the trajectory using $\pi(a|s, z)$ (Park et al., 2024).

3.2 Objective

The novel METRA objective is given by:

$$I_{\mathcal{W}}(S; Z) = \mathcal{W}(p(s, z), p(s)p(z)), \tag{1}$$

where $I_{\mathcal{W}}(S; Z)$ is the WDM and $\mathcal{W}$ is the 1-Wasserstein distance on the metric space. WDM is a metric-aware quantity, meaning that it discovers different skills, but simultaneously maximizes the distance d between different skill trajectories (Ozair et al., 2019).

3.3 Optimization

To make the objective more straightforward to maximize in practice, we can implement a simplification by introducing the Kantorovich-Rubenstein duality:

$$I_{\mathcal{W}}(S; Z) = \sup_{\|f\|_L \le 1} \mathbb{E}_{p(s,z)}[f(s, z)] - \mathbb{E}_{p(s)p(z)}[f(s, z)] \tag{2}$$

where f represents a score function that assigns larger values to (s, z) sampled from the joint distributions and smaller values to (s, z) sampled independently from their marginal distributions. $||f||_L$ represents the Lipschitz constant for the function $f : \mathcal{S} \times \mathcal{Z} \to \mathbb{R}$. The duality in equation 2 can be used to reformulate the Wasserstein distance as the maximization over a 1-Lipschitz function (Ozair et al., 2019; Villani, 2009).

However, training the 1-Lipschitz function would require a formulation of the reward function that requires sampling N zs for each data point. This would be too computationally expensive, we can parametrize the score function f as:

$$f(s, z) = \phi(s)^T \psi(z) \tag{3}$$

where the 1-Lipschitz functions $\phi : \mathcal{S} \rightarrow \mathbb{R}^D$ maps the state space and $\psi : \mathcal{Z} \rightarrow \mathbb{R}^D$ maps the skill space.

We can also consider a variant of the WDM that only depends on the last state, which allows us to use a telescoping sum to further decompose the objective. Rewriting the objective using the above simplifications:

$$I_{\mathcal{W}}(S_T; Z) \approx \sup_{\|\phi\|_L \leq 1} \mathbb{E}_{p(\tau,z)} \left[\sum_{t=0}^{T-1} (\phi(s_{t+1}) - \phi(s_t))^T (z - \bar{z}) \right] \tag{4}$$

where $\bar{z} = \mathbb{E}_{p(z)}[z]$. The term $z - \bar{z}$ centralizes the skills to have a mean of 0, for ease of computation.

Finally, we can incorporate the temporal distance metric, which depends only on the MDP's inherent transition dynamics. We can rewrite equation 4 as:

$$\sup_{\pi,\phi} \mathbb{E}_{p(\tau,z)} \left[\sum_{t=0}^{T-1} (\phi(s_{t+1}) - \phi(s_t))^T z \right] \tag{5}$$

such that $||\phi(s) - \phi(s')||_2 \leq 1, \forall (s, s') \in S_{\text{adj}}$. Intuitively, a policy that maximizes the METRA objective should be able to move as far as possible in a variety of directions in the latent space. Thus, the learned latent space should attempt to spread out the manifolds (subsets of state space that are related due to the system dynamics) to make the shortest paths as long as possible (Park et al., 2024).

3.4 Implementation

We based our SAC model on the source Tabor, 2024, but we heavily modified the source code to store skills in the replay buffer and use them for inference. For the METRA algorithm, we implemented this from scratch only using the paper as a reference. In building the model, we wanted to ensure each component was modular and self-contained for easier debugging. Here is the breakdown of the division of components:

1. **`networks.py`:** Contains the definitions for each individual neural network (Actor, Critic, Phi)
2. **`agent.py`:** Assembles training updates and initialization for SAC.
3. **`metra.py`:** Implements METRA training loop and includes testing tools for downstream tasks.
4. **`test_graph.py`, `test_skills.py`:** Testing scripts that call metra with different `kwargs`.

One key difference between our METRA implementation and the pseudo-code in figure 2 is the addition of mini-batches. Instead of calculating the gradient over the entire replay buffer at each epoch, we first roll out `episodes_per_epoch` trajectories and sample `grad_steps_per_epoch` mini-batches for efficiency and more stable gradient updates.

We focused on optimization to ensure that training time was not a limitation for our model. Notably, we vectorize every operation, with a key aspect of this vectorization being the use of `torch.einsum` to efficiently compute the mini-batch losses. Additionally, we used Modal Labs (a serverless GPU computing service) to deploy GPUs for faster training. Since the models were relatively small, this provided minor performance boosts, as the main bottleneck was trajectory rollouts, which could not be parallelized.

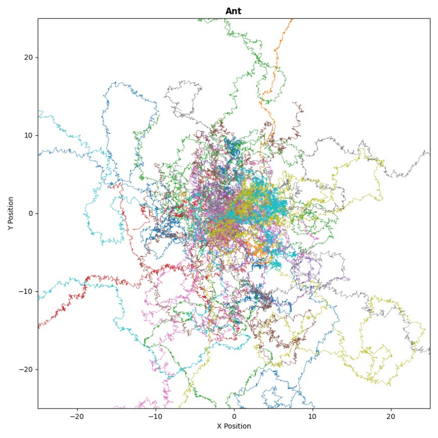
Debugging was challenging, as it was difficult to monitor the model's performance during training because most of the paper's model evaluation was done post-training. We monitored the model's performance by outputting the phi and lambda losses and qualitatively rendering the agent at each step to observe its behaviors. This approach helped us debug the negative signs in our loss functions in order to maximize the phi loss.

Since directly inputting the pixel values into our model for pixel-based environments would result in very high dimensionality for our state space, we train a CNN autoencoder to efficiently represent states during trajectory rollouts. We then use the encoded states as the observations for training.

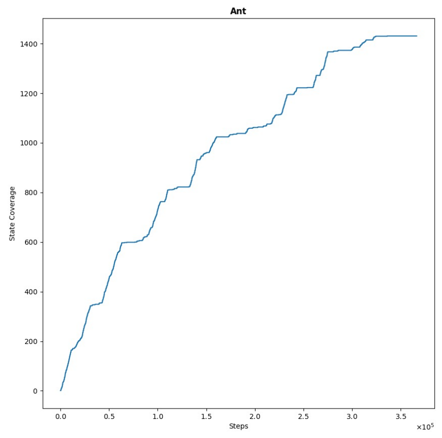
We present the corresponding hyperparameters in table 1. We noticed that the losses plateaued around 2500 epochs. We opted for smaller model architectures with three layers and 256 neurons to allow for faster experimental turnaround, given the relatively small state and action spaces.

4 Results

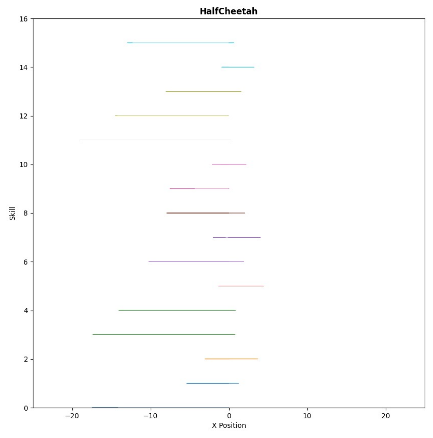
To evaluate the performance of METRA, we performed a series of experiments in complex state- and pixel-based environments, then collected the quali-



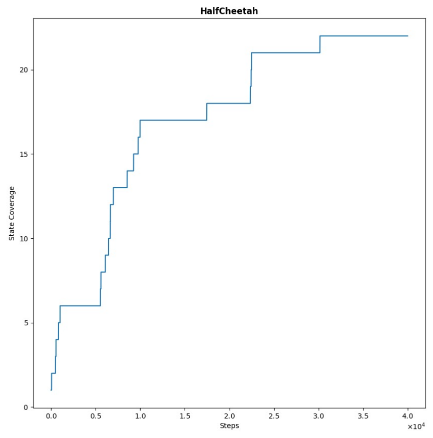
(a) Ant (States)



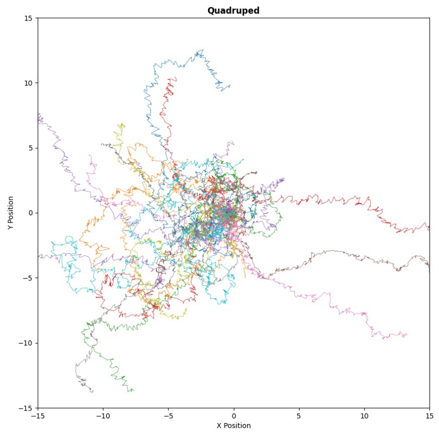
(a) Ant (States)



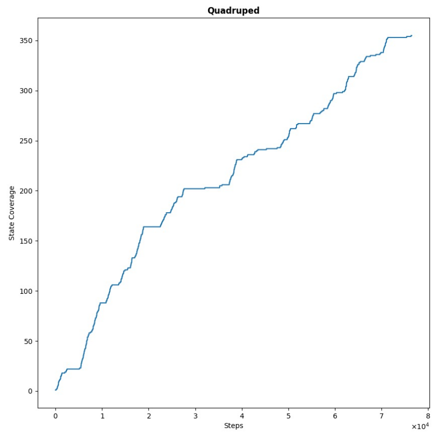
(b) Half-Cheetah (States)



(b) Half-Cheetah (States)



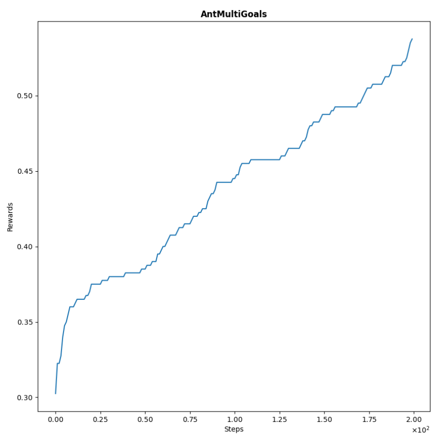
(c) Quadruped (Pixels)



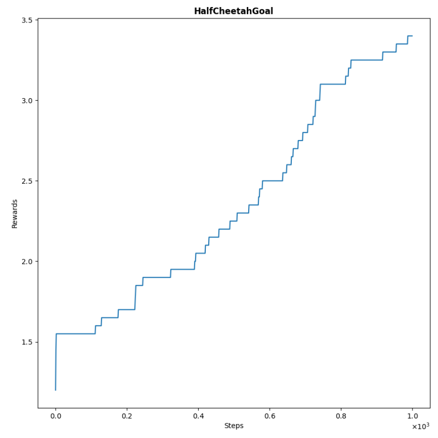
(c) Quadruped (Pixels)

Figure 3. METRA learns diverse locomotion behaviors across different environments.

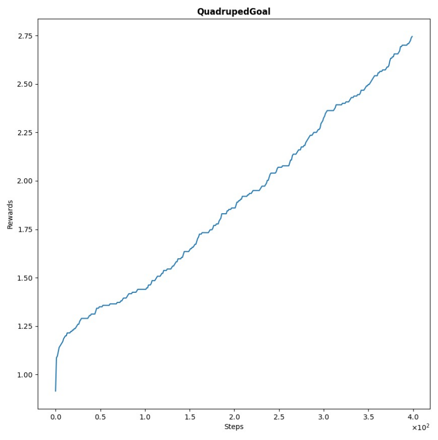
Figure 4. METRA covers a wide variety of states across different environments.



(a) Ant (States)



(b) Half-Cheetah (States)



(c) Quadruped (Pixels)

Figure 5. METRA performs well on downstream tasks across different environments.

tative and quantitative results. For our experimental setup, we used the MuJoCo Ant and HalfCheetah environments (Gym, 2024), as well as the pixel-based Quadruped environment from the DeepMind Control Suite (Technologies, 2024). Ant has a 29-dimensional state space and HalfCheetah has an 18-dimensional state space, while Quadruped has an observation space of $64 \times 64 \times 3$ and we do not use any proprioceptive state information. To help the agent infer its location from pixels, we use a gradient colored floor. Ant is trained on a 2-dimensional latent space, Quadruped is trained on a 4-dimensional latent space, and HalfCheetah is trained on a discrete latent space of size 16.

In each environment, we tested the performance of METRA in the diversity of the locomotion behaviors discovered, performance on downstream tasks, and the amount of state coverage. We aimed to address three main areas of interest: (1) if METRA is able to scale to complex, high-dimensional environments, (2) if METRA is able to discover meaningful behaviors in an unsupervised manner, and (3) if the behaviors learned are applicable to downstream tasks.

4.1 Behaviors

For the Ant and Quadruped environments, we randomly sample skills from the latent space and plot the trajectories. For HalfCheetah, we plot the trajectories of all 16 skills.

Figures 3a, 3b, and 3c show the locomotion behaviors that METRA learns. We see that different skills correspond to unique trajectories, indicating that each skill covers a distinct region of the state space.

4.2 State Coverage

To quantify state coverage, we count the number of 1×1-sized x-y bins for Ant and Quadruped and 1-sized x bins for HalfCheetah which are visited by any trajectories. We compute the state coverage after every training epoch by using the trajectories from the rollouts of 48 randomly sampled skills.

Figures 4a, 4b, and 4c show that METRA is able to approximately cover the state space.

4.3 Downstream Tasks

To quantify the performance of METRA on downstream tasks, we use the tasks AntMultiGoals, HalfCheetahGoal, and QuadrupedGoal defined by the original METRA paper (Park et al., 2024). In AntMultiGoals, the task is to reach a target goal randomly sampled from $[s_x - 7.5, s_x + 7.5] \times [s_y - 7.5, s_y + 7.5]$, where (s_x, s_y) is the agent's current x-y position. The agent receives a reward of 2.5 whenever it reaches the goal. A new goal is sampled when the agent either reaches the previous goal or fails to reach it within 50 steps. In HalfCheetahGoal and QuadrupedGoal, the task is to reach a target goal randomly sampled from $[-25, 25]$ and $[-7.5, 7.5]$ respectively. The agent receives a reward of 10 whenever it reaches the goal. We restrict the agent to 200 environment steps in AntMultiGoals and HalfCheetahGoal, and 400 environment steps in QuadrupedGoal. We average the total accumulated rewards across 1,000 rollouts at each time step.

To control the agent, we use zero-shot goal-reaching. By setting $z = \frac{\phi(g)-\phi(s)}{||\phi(g)-\phi(s)||_2}$ for continuous skills and $z = \arg\max(\phi(g) - \phi(s))$ for discrete skills, we can find the skill that leads to the goal state (Park et al., 2024). This allows METRA to solve goal-conditioned tasks without learning a separate goal-conditioned policy.

The results are depicted in figures 5a, 5b, and 5c. We show that METRA is able to reach goal states relatively consistently and quickly.

5 Limitations

As demonstrated, METRA is a novel unsupervised skill discovery method that is able to outperform current methods in complex environments with high intrinsic state dimensionality (Park et al., 2024). However, even though METRA exhibits state-of-the-art performance across several environments, it still has intrinsic limitations that provides direction for future work. Due to the nature of inherent differences in experimental configurations and resource constraints between our project and the original METRA implementation, there are also several reproducibility limitations that we will outline.

5.1 METRA Limitations

Similar to other unsupervised skill discovery methods, METRA is still lacking in terms of sample efficiency. METRA uses a relatively small update-to-data ratio, so each gradient step uses a large amount of data. Furthermore, METRA is implemented using a vanilla SAC backbone, so exploring other RL backbones would be a worthwhile consideration (Haarnoja et al., 2018). For instance, Data-regularized Q (DrQ) is a novel model-free RL technique that uses data augmentation to circumnavigate the need for pre-training or auxillary losses, which improves sample efficiency (Kostrikov et al., 2020). METRA also assumes a fixed MDP, which is not extendable to non-Markovian environments. This poses a limitation on the performance of METRA in environments other than the locomotion and manipulation ones tested (Park et al., 2024). Further work would explore non-Markovian dynamics to allow METRA to perform in more complex and diverse environments.

Furthermore, there are limitations stemming from the METRA objective itself. In equation 5, the temporal distance, which could be asymmetric, is incorporated into the symmetric Euclidean distance. The temporal distance abstraction is given by the minimum of both temporal distances $\min(d_{\text{temp}}(s_1, s_2), d_{\text{temp}}(s_2, s_1))$, which may be an overly conservative simplification for asymmetric environments. In future work, a potential solution to this issue is to replace the Euclidean distance $||\phi(s_1) - \phi(s_2)||_2$ with an asymmetric quasimetric. For instance, the quasimetric value function obtained using Quasimetric Reinforcement Learning (QRL) preserves local distances, but maximally spreads out other states (Wang et al., 2023). A similar notion could be adapted to mitigate the effect of the temporal distance assumption in asymmetric environments.

5.2 Reproducability Limitations

Due to time and resource constraints, we were not able to perform experiments on a wider variety of environments beyond state-based and pixel-based environments. While we evaluated METRA on two state-based environments (Ant, HalfCheetah) and a pixel-based environment (Quadruped), given additional time, future directions for research would entail also exploring Atari games and a wider variety of pixel-based environments. To enable successful performance on these additional environments, we would explore adapting METRA to perform successfully on non-Markovian dynamics. Additionally, in comparison to the original METRA implementation, the trajectories explored by our agent have slightly more overlap, and are therefore less diverse.

Compared to the original METRA implementation, our reproduction did not reach the same level of downstream task performance. Due to time constraints, we did not implement a hierarchical high-level controller on the frozen policy, which would have helped maximize rewards and improve downstream task performance (Park et al., 2024).

6 Conclusion

In this work, we presented a reproduction of METRA, which is a state-of-the-art unsupervised RL method that uses the coverage of a temporal distance-based latent skill space. While METRA still holds limitations in terms of its adaptability to non-Markovian environments, sample inefficiency, and potential oversimplification of the objective, it out-performs current unsupervised RL methods in complex state-based and pixel-based environments.

Acknowledgements

We would like to thank Professor Benjamin Eysenbach, Professor Mengdi Wang, and all of the COS 435 TAs for their feedback and guidance both throughout the semester and on this project.

References

Campos, Víctor et al. (2020). "Explore, Discover and Learn: Unsupervised Discovery of State-Covering Skills". In: *CoRR* abs/2002.03647. arXiv: 2002.03647. URL: https://arxiv.org/abs/2002.03647.

Gym, OpenAI (2024). *MuJoCo Environments*. https://www.gymlibrary.dev/environments/mujoco/index.html. Accessed: 2024-05-07.

Haarnoja, Tuomas et al. (2018). "Learning to Walk Via Deep Reinforcement Learning". In: *Robotics: Science and Systems XV*. Robotics: Science and Systems Foundation. DOI: 10.15607/rss.2019.xv.011.

Kostrikov, Ilya, Denis Yarats, and Rob Fergus (2020). "Image Augmentation Is All You Need: Regularizing Deep Reinforcement Learning from Pixels". In: *CoRR* abs/2004.13649. arXiv: 2004.13649. URL: https://arxiv.org/abs/2004.13649.

Laskin, Michael et al. (2021). "URLB: Unsupervised Reinforcement Learning Benchmark". In: *CoRR* abs/2110.15191. arXiv: 2110.15191. URL: https://arxiv.org/abs/2110.15191.

Mendonca, Russell et al. (2021). "Discovering and Achieving Goals via World Models". In: *CoRR* abs/2110.09514. arXiv: 2110.09514. URL: https://arxiv.org/abs/2110.09514.

Mnih, Volodymyr et al. (Feb. 25, 2015). "Human-level control through deep reinforcement learning". In: *Nature* 518.7540, pp. 529–533. DOI: 10.1038/nature14236.

Ozair, Sherjil et al. (2019). "Wasserstein Dependency Measure for Representation Learning". In: *CoRR* abs/1903.11780. arXiv: 1903.11780. URL: http://arxiv.org/abs/1903.11780.

Park, Seohong, Oleh Rybkin, and Sergey Levine (2024). *METRA: Scalable Unsupervised RL with Metric-Aware Abstraction*. arXiv: 2310.08887 [cs.LG].

Park, Seohong et al. (2022). "Lipschitz-constrained Unsupervised Skill Discovery". In: *CoRR* abs/2202.00914. arXiv: 2202.00914. URL: https://arxiv.org/abs/2202.00914.

Schulman, John et al. (2015). "Trust Region Policy Optimization". In: *CoRR* abs/1502.05477. arXiv: 1502.05477. URL: http://arxiv.org/abs/1502.05477.

Tabor, Phil (2024). *Soft Actor-Critic (SAC) Implementation in PyTorch*. https://github.com/philtabor/Youtube-Code-Repository/tree/master/ReinforcementLearning/PolicyGradient/SAC.

Technologies, DeepMind (2024). *DeepMind Control Suite*. https://github.com/google-deepmind/dm_control.

Villani, Cedric (2009). In: *Optimal transport: old and new*. Springer.

Wang, Tongzhou et al. (2023). *Optimal Goal-Reaching Reinforcement Learning via Quasimetric Learning*. arXiv: 2304.01203 [cs.LG].