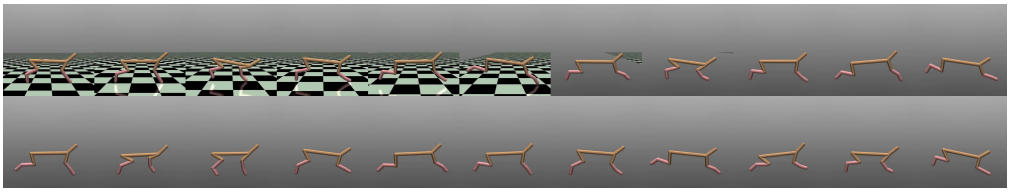
A Rendering of Discovered Skills

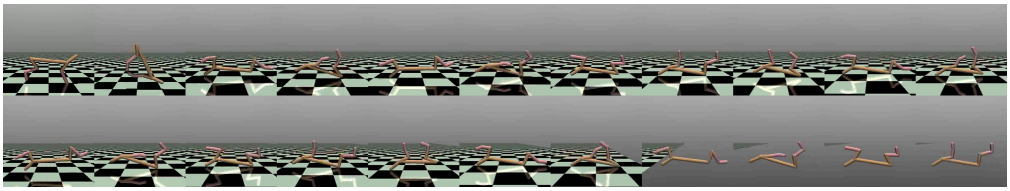
We present the rendered robotic locomotion tasks in this section. Videos of different locomotion skills can be found on our project website `https://sites.google.com/view/neurips22-rest`.

A.1 HalfCheetah

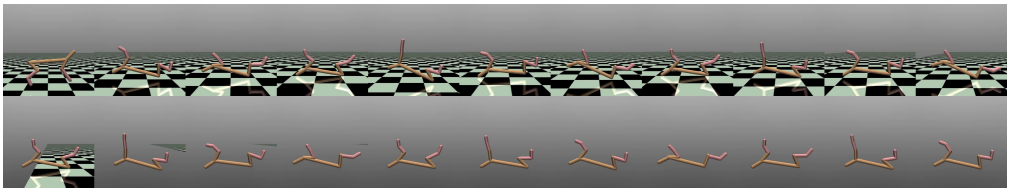
We used `HalfCheetah-v3` environment in OpenAI Gym [16].



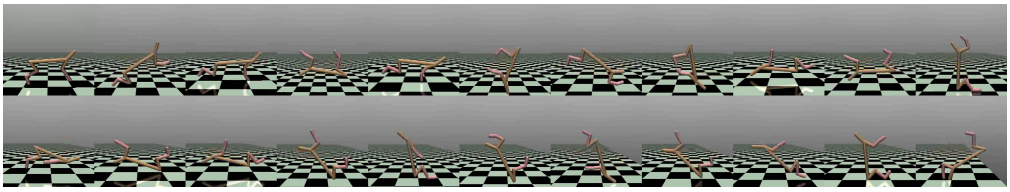
(a) `HalfCheetah` Backward Running



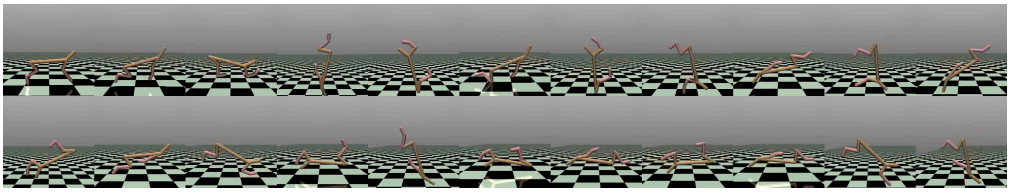
(b) `HalfCheetah` Flipped Forward Running



(c) `HalfCheetah` Flipped Backward Running



(d) `HalfCheetah` Rolling Forward

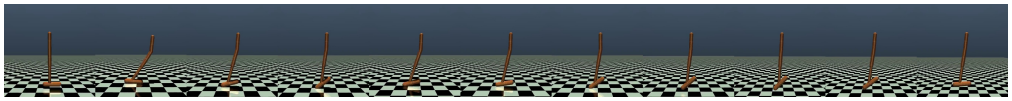


(e) `HalfCheetah` Rolling Backward

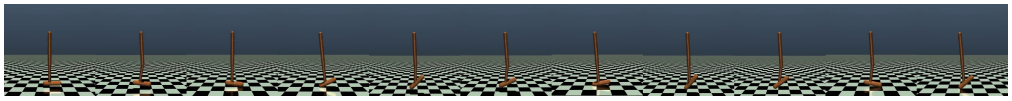
Figure 7: Rendering of skills discovered with robot `HalfCheetah`.

A.2 Hopper

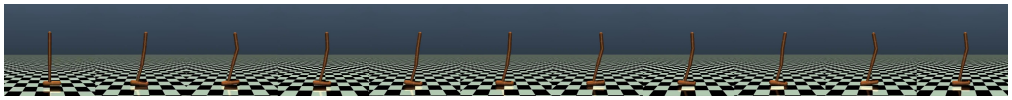
We used `Hopper-v3` environment in OpenAI Gym [16].



(a) `Hopper` Hopping Forward



(b) `Hopper` Hopping Backward

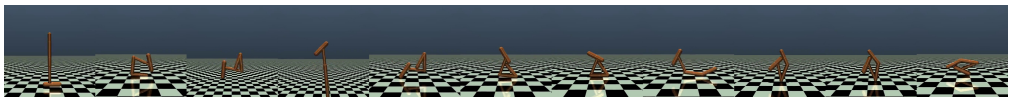


(c) `Hopper` Small-Step Hopping Forward

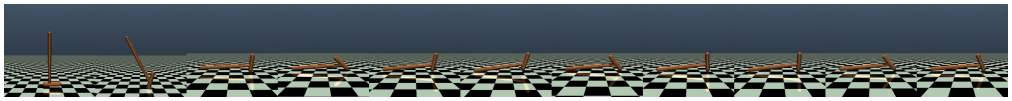
Figure 8: Rendering of skills discovered with robot `Hopper` with termination.

A.3 Hopper Without Termination

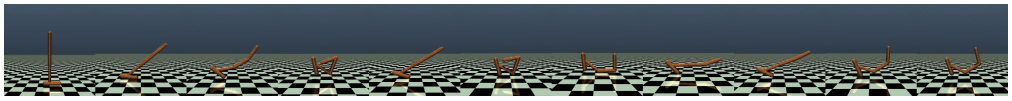
We set the `terminate_when_unhealthy` parameter to `False` in the `Hopper-v3` environment in OpenAI Gym [16].



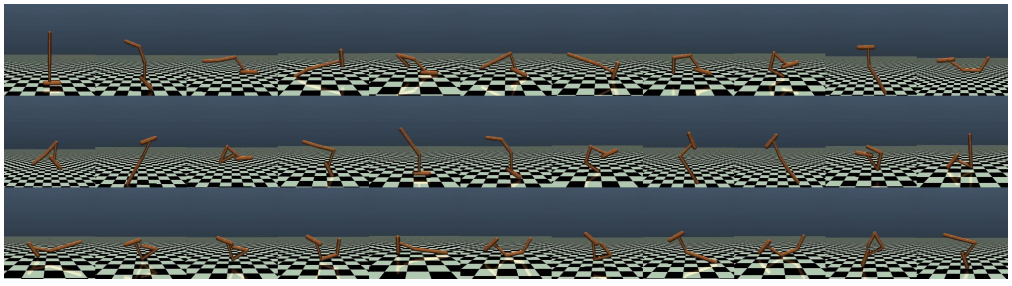
(a) `Hopper` Back Flip



(b) `Hopper` Rubbing Backward



(c) `Hopper` Crawling Forward

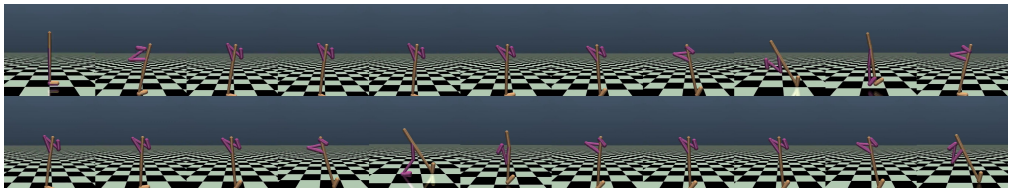


(d) `Hopper` Kung Fu

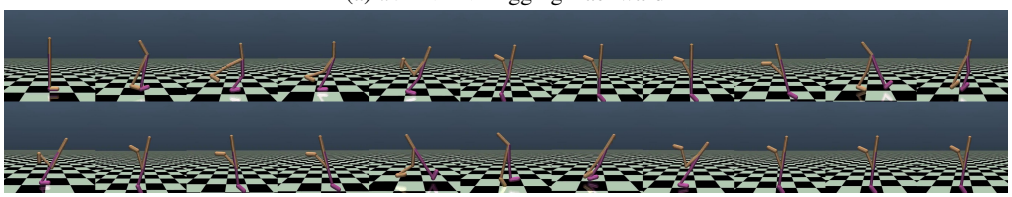
Figure 9: Rendering of skills discovered with robot `Hopper` without termination.

A.4 Walker2d

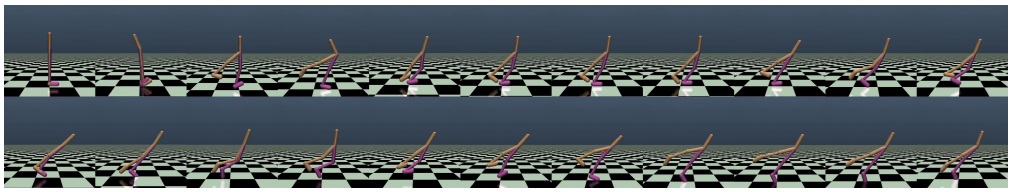
We used `Walker2d-v3` environment in OpenAI Gym [16]



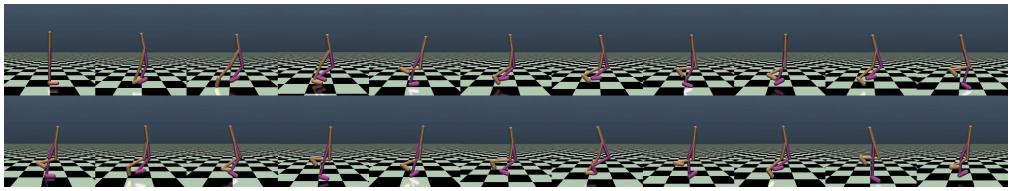
(a) `Walker2d` Digging Backward



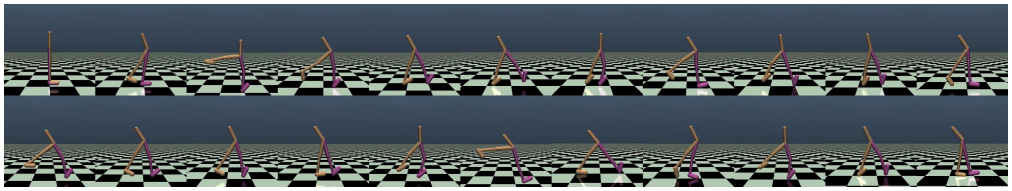
(b) `Walker2d` Standing on One Foot



(c) `Walker2d` Ninja Running



(d) `Walker2d` Trotting



(e) `Walker2d` Walking Backward

Figure 10: Rendering of skills discovered with robot `Walker2d`.

B Implementation Details

In this section, we introduce our implementation details of our proposed ReST algorithm and our comparison baselines.

B.1 ReST

Our proposed ReST algorithm can incorporate with any reinforcement learning algorithm to maximize the intrinsic reward. We choose Proximal Policy Optimization (PPO) [13] as our algorithm to maximize intrinsic reward. Detailed PPO related hyper-parameters can be found in Table 1.

We set the temperature coefficient $\alpha = 10$ for 2D navigation environments and $\alpha = 1$ for MuJoCo environments. We train each skill for $M = 20$ epochs in all the environments. We used $k = 5$ output channels for the RND networks. We update the RND networks 10 times per epoch and the learning rate is same as the learning rate of the critic. For the RND networks to better record the visitation of a certain skill, it needs to slightly overfit and we set the hidden dimensions for the RND networks as $[500, 500]$.

B.2 DIAYN

We use our own reproduction of DIAYN in the experiments talked about in this paper. For fair comparison, we also used PPO to optimize the intrinsic reward of DIAYN. We basically followed the hyper-parameter setting from the original DIAYN paper and added entropy term $\alpha = 0.1$ to the original DIAYN intrinsic reward. We update the discriminator 10 times per epoch during training and the learning rate is the same as the learning rate of the critic.

For the independent neural network parameterization version of DIAYN, we used same initialization for all the skills for fair comparison. We used exactly the same network architecture as our proposed algorithm.

Detailed PPO hyper-parameters can be found in Table 1.

B.3 DADS

We use our own reproduction of DADS in the experiments mentioned above. For a fair comparison, we used PPO to optimize the intrinsic reward of DADS. We basically followed the hyper-parameter setting, network architectures, and implementation details from the original DADS paper, like predicting Δs in the skill-dynamics and using a Mixture-of-Experts [40] with 4 Gaussian experts to model the output distribution. We don't let the dynamics to predict $\Delta x, \Delta y$ if the global coordinates are excluded from the state s, as there is no goal-based navigation problems in our experiments. The input (s, z) first goes through a two hidden layers MLP, whose capacity is the same as that of the policy and critic networks, then the output will be linear tranformed to be the means of the Gaussian experts and a discrete distribution over them. We fix the covariance matrix of each expert to be an Identity Matrix. The skill dynamics is updated 32 times per epoch during training and the learning rate is the same as that of the critic.

For the independent neural network parameterization version of DADS, we used same initialization for all the skills for fair comparison. We used exactly the same network architecture as our proposed algorithm.

Detailed PPO hyper-parameters can be found in Table 1.

B.4 Other Implementation Details

We implement all the algorithm using PyTorch 1.10.1 and Python 3.9.7. All the networks are trained using NVIDIA RTX 3090 GPU and the CPU used for MuJoCo simulation is AMD EPYC 7H12.

C Details of Evaluation Metrics

C.1 State Coverage

We particlize the X-Y plane of the 2D navigation environments into 20×20 particles. We rollout 20 trajectories for each skill each algorithm. Therefore, 200 trajectories are generated for each algorithm. We count the particles visited by any or the 200 trajectories and record the number as n_v, then the state coverage rate is $\frac{n_v}{400}$.

C.2 Mutual Information

We particlize the 4 dimensional state space, which is x, y, v_x, v_y into $20 \times 20 \times 20 \times 20$ particles. We rollout 20 trajectories for each skill each algorithm. We first count the visitation frequency of particles $p(s)$ where s is a 4 dimensional particle and calculate $H(S) = -\sum_s p(s) \log p(s)$. We then calculate the visitation frequency of particles for each skill $p(s|z)$ and calculate $H(S|Z) = -\sum_s p(s|z) \log p(s|z$. Therefore, we derive the mutual information between skills and states: $I(S; Z) = H(S) - \sum_{i=1}^{n} \frac{1}{N} H(S|z_i)$.

Table 1: Hyper-parameter Settings of PPO

Hyper-parameter	DIAYN	DIAYN-i	DADS	DADS-i	ReST (ours)
Hidden Layers	2	2	2	2	2
Hidden Nodes	256	64	256	64	64
Activation	tanh	tanh	tanh	tanh	tanh
Log std init	-0.5	-0.5	-0.5	-0.5	-0.5
Discount factor γ	0.99	0.99	0.99	0.99	0.99
Batch size (2D Navigation)	10^5	10^5	10^5	10^5	10^5
Batch size (MuJoCo)	2×10^4	2×10^4	2×10^4	2×10^4	10^4
Policy updates per epoch	10	10	10	10	10
Critic updates per epoch	10	10	10	10	10
Max episode length	1000	1000	1000	1000	1000
GAE λ	0.95	0.95	0.95	0.95	0.95
Learning rate for π	3×10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}
Learning rate for V	1×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-3}
Clip factor	0.2	0.2	0.2	0.2	0.2

D Environment Details

D.1 2D Navigation Environments

We use our own implementation for the 2D navigation environments in Section 4.1. The environment is a GPU based environment implemented using PyTorch 1.10.1 and Python 3.9.7. The state space is the concatnation of (x, y, v_x, v_y) and the action space is the acceleration (a_x, a_y). We use a fixed episode length of 1000 in all 2D navigation environments. The X-Y plane of these environments is $[0.1] \times [0, 1]$ whereas the velocity space is $\{(v_x, v_y) | v_x^2 + v_y^2 \leq 0.01\}$.

D.2 MuJoCo Environments

We use OpenAI Gym [16] as our evaluation environment in Section 4.2. The maximum episode length of the environments are set to 1000. We follow the setting of OpenAI gym where `HalfCheetah-v3` does not terminate and run for 1000 time steps each episode whereas `Hopper-v3` and `Walker2d-v3` terminates when unhealthy or reaches the maximum time step. We expose all the state information to the discriminator or dynamic model or RND modules and exclude current position from the observation of the agents.

We also added `Ant-v3` to our rendering of the learned skills. The `Ant-v3` environments are adopted from OpenAI Gym but does not terminate when unhealthy. Moreover, some of the skills demonstrated in the rendering using `Hopper-v3` also do not terminate when unhealthy. These are done by setting the `terminate_when_unhealthy=False`. This modification makes some flipping skills and other interesting ones emerged during training.

E Why ReST does not have the lack of exploration issue mentioned by EDL [9]

Our proposed algorithm ReST not only addresses the parallel training issue, but can also solve the issue mentioned by EDL [9]. EDL argues that the reason for the poor state coverage of previous information theoretic unsupervised skill discovery approaches is that the mutual information reward inherently do not encourage exploration. For a visited state, the intrinsic reward for the skill that visited the state would be $\log N$ while for an unexplored state, the reward for arbitrary skill would be 0. In this section, we demonstrate that our proposed algorithm doew not have this undesirable property.

We make the same assumption as in Equation 8, which means we have perfect RND modules. For a state s already visited by skill z_i, the intrinsic reward would be:

$$r(s, z_i) = -\log \left[\frac{\sum_{j \in \{1,2,\ldots,N\}, j \neq i} e^{\left(-\alpha \cdot ||\hat{f}_j(s_{t+1}) - f_j(s_{t+1})||^2\right)}}{N-1} \right] \tag{9}$$

whereas for an unexplored state s', the reward function stays the same. We note that the above reward function does not contain any term related to skill z_i, which indicates that the reward function for z_i at a state has nothing to do with whether state s is an explored state by z_i.

F More Comparison Baselines

We compare our proposed algorithm to several more comparison baselines in the 2D navigation environments. The added comparison baselines are discussed in the following subsections.

F.1 DIAYN/DADS with state coverage-based intrinsic reward

We trained agents using the parallel training paradigm as DIAYN or DADS did with state coverage-based intrinsic reward as shown in Equation (7). We added this comparison baseline to show the effect of the recurrent training paradigm and the state coverage-based intrinsic reward respectively. The result shows that without the recurrent training paradigm, the discovered skills perform worse than those with the recurrent training paradigm while still performs better than those using mutual information as intrinsic reward. As shown in Fig 11, we denote this baseline as ReST-p.

F.2 ReST with MI as intrinsic reward

We also trained agents using the reverse formulation of the mutual information reward in Equation (1) as suggested by [9] together with the recurrent training paradigm. We added this comparison baseline to demonstrate the effect of the recurrent training paradigm and the state coverage-based intrinsic reward respectively. As shown in Fig 11, we denote this baseline as MIrecurrent.

F.3 LEXA

We trained agents using LEXA [22]. The empirical results show that LEXA does not perform well in the 2D navigation environments. The LEXA framework does not work well in the 2D navigation settings mainly because of it's goal-conditioned setting. The LEXA framework takes in an image as a goal and the intrinsic reward for the achiever π^g is encouraging the agent to go to and *stay at* the goal. This is helpful for many real-world robotic tasks, for instance, pick and place manipulation. However, despite the effectiveness, this goal based setting introduces some limitations in other domains. For instance, the goal based intrinsic reward is one of the reasons why the DeepMind Control experiments demonstrated by LEXA contain mainly 'posing' skills, since the reward encourages the agent to stay at the state provided by the goal image. This setting is more problematic in our setting since the observation space of the 2D navigation environment is a vector composed of the position in the x-y plane and velocities v_x, v_y. If the achiever is encouraged to stay at the goal, suppose it reached the goal position and velocity at a certain timestep, then in the next time step, if v_x and v_y are not 0, the agent would run away from the goal state immediately, which makes the agent get low reward. This kind of scenario is common in robotics. For instance, if we use the LEXA framework in the legged

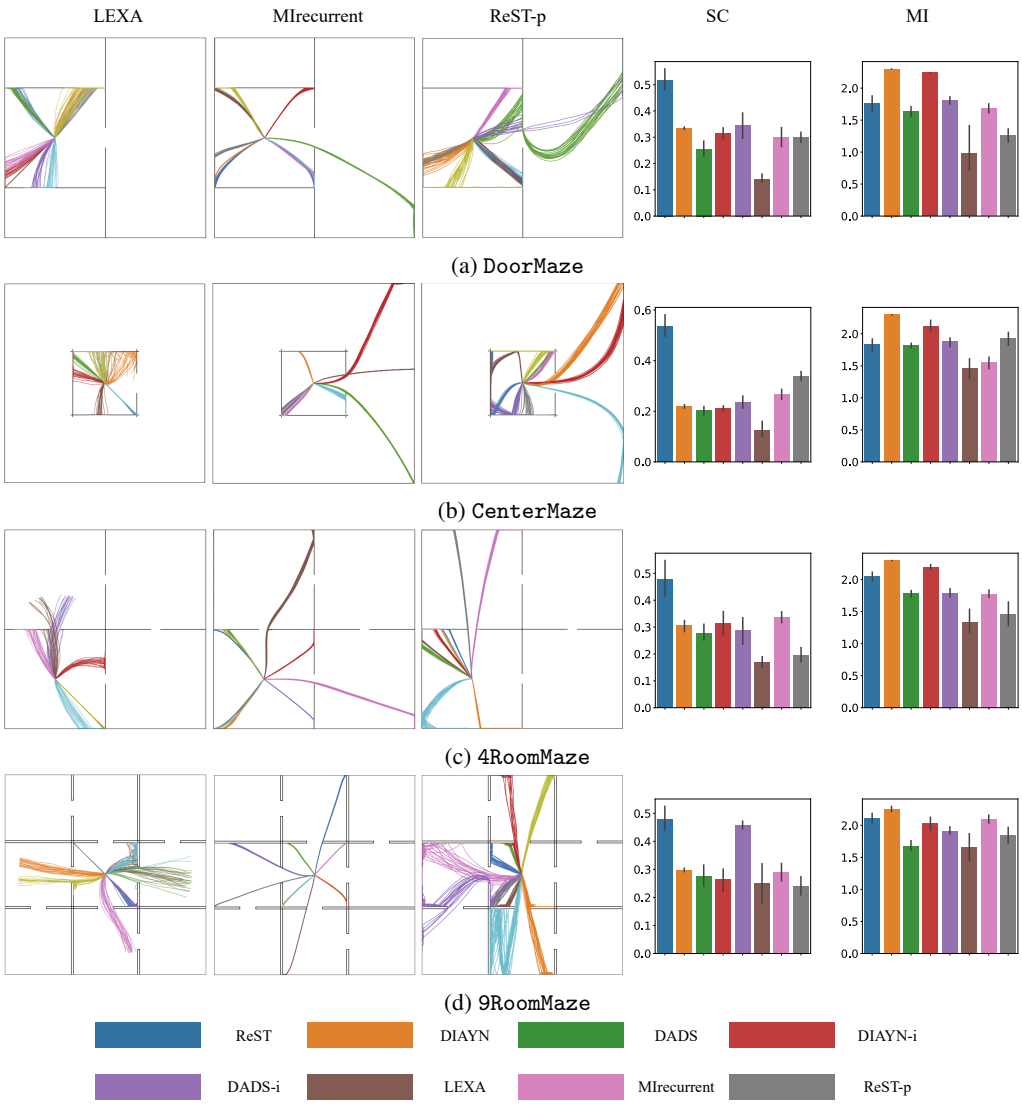


Figure 11: Additional Results for 2D navigation experiments.

locomotion tasks, the observation space would contain joint positions and joint velocities, which might bring trouble to the training process. We believe this is one of the reasons why LEXA does not work in our low dimensional state settings. Our proposed approach ReST, however, does not have such issues since it directly learns a set of skills and no explicit goal is given for the agent to reach and stay during the training process.

G Quantitative Results for MuJoCo

We quantitatively evaluated the MuJoCo robotic locomotion tasks by calculating the variance of the ending x position of each skill. The results are shown in Table 2.

H Broader Impact

Being able to acquire useful skills in the absence of reward functions is beneficial for a wide range of applications. To this end, we introduce our Recurrent Skill Training algorithm to effectively explore the state space of the environment without external supervision. This enables a fully autonomous acquisition of skills, which might bring both positive and negative social impacts. On the positive side, for instance, the fully autonomous procedure of acquiring diverse and state covering skills makes

Table 2: Quantitative Results for MuJoCo

Hyper-parameter	DIAYN	DIAYN-i	DADS	DADS-i	ReST (ours)
`Hopper`	0.454	0.01	0.40	0.37	76.49
`HalfCheetah`	54.47	294.89	45.59	0.55	3792.13
`Walker2d`	0.22	1.05	1.70	1.23	227.94

it possible for robots to explore in dangerous, unknown and unstructured environments to cover all valid states, preventing human from being exposed to dangers. However, on the negative side, when deployed incorrectly in the real world, the novelty-seeking nature of the proposed approach might autonomously discover skills novel yet dangerous, which might be harmful for both the agent and the environment. This indicates that we should be careful on the usage of such algorithms in the real world.