OPEN-ENDED REINFORCEMENT LEARNING WITH NEURAL REWARD FUNCTIONS

Robert Meier *
Department of Computer Science
ETH Zürich
Zürich, Switzerland
romeier@inf.ethz.ch

Asier Mujika *
Department of Computer Science
ETH Zürich
Zürich, Switzerland
asierm@inf.ethz.ch

ABSTRACT

Inspired by the great success of unsupervised learning in Computer Vision and Natural Language Processing, the Reinforcement Learning community has recently started to focus more on unsupervised discovery of skills. Most current approaches, like DIAYN or DADS, optimize some form of mutual information objective. We propose a different approach that uses reward functions encoded by neural networks. These are trained iteratively to reward more complex behavior. In high-dimensional robotic environments our approach learns a wide range of interesting skills including front-flips for HALF-CHEETAH and one-legged running for HUMANOID. In the pixel-based Montezuma's Revenge environment our method also works with minimal changes and it learns complex skills that involve interacting with items and visiting diverse locations. A web version of this paper which shows animations for the different skills is available in https://as.inf.ethz.ch/research/open_ended_RL/main.html

1 INTRODUCTION

Deep reinforcement learning (RL) has proven to be very successful in many challenging tasks (Mnih et al., 2015; Silver et al., 2017b; Berner et al., 2019). These were considered intractable just a few years ago. However, current methods require enormous amounts of compute to achieve great performance in individual tasks. This compute just goes to waste very often when new tasks are considered, even if the environment does not change. Not long ago this was also the case in Computer Vision and Natural Language Processing, until unsupervised learning took off. By using task agnostic pre-training schemes, generalist models have revolutionized both fields (Caron et al., 2021; Tenney et al., 2019; Brown et al., 2020). They can solve most tasks with minimal or even no fine-tuning, Unsupervised Reinforcement Learning aims to bring similar successes to the Reinforcement Learning community. By discovering a wide range of skills that cover diverse behaviours in an environment, we will be able to solve new tasks, beyond what could be imagined when the model was trained, with minimal or even no training.

Here, we propose a new method for open-ended, unsupervised skill discovery. We devise an iterative process which creates pairs of neural reward functions together with policies that can solve the corresponding reward function. Each policy corresponds to a skill that the agent has learnt. The neural reward function is a neural network that maps the current observation to a scalar reward. In each iteration, the neural reward function is modified to differ from the previous one. This results in increasing the complexity of the encoded task. Then, a new skill is learnt that optimizes this reward function. We devise several techniques to transfer the knowledge from previously learnt skills. These mechanisms enable learning of the most complex reward functions that our method creates. In fact, we show that some of the functions are impossible to learn from scratch.

We empirically test our framework in a diverse set of environments. First, we apply it to a simple 2d navigation task. This lets us perform experiments quickly and gain a solid empirical understanding of the different components of our method. Then, we apply it to three robotic environments where we

*equal contribution

have continuous high-dimensional observations and actions. Finally, we apply it on the challenging Montezuma's Revenge Atari game which has visually rich pixel based observations and discrete actions. Despite their differences, our method manages to learn useful and interesting skills in all of them. This shows that neural reward functions are equipped to encode meaningful tasks in very diverse environments. We highlight the most important findings of this paper:

- Solve a 2d maze that cannot be solved by using random exploration.
- Acquire complex skills including performing front- and back-flip in the HALF-CHEETAH, running in all directions in ANT and HUMANOID, standing and jumping on one leg in HUMANOID.
- Learn to run as fast in HUMANOID as a supervised RL agent trained for tens of millions of steps.
- Achieve a higher Particle-Based Mutual Information Metric (Gu et al., 2021) than approaches that use expert knowledge for feature engineering.
- Collect the first key and reach several rooms in Montezuma's Revenge.

2 RELATED WORK

Our work fits best into the unsupervised skill discovery literature (Mohamed & Rezende, 2015; Gregor et al., 2016; Florensa et al., 2017a; Achiam et al., 2018; Eysenbach et al., 2018; Sharma et al., 2019; Choi et al., 2021). Compared to DIAYN (Eysenbach et al., 2018) and similar approaches, we do not set the number of skills to be learnt at the beginning of training. This allows us to learn new skills in an open-ended fashion. On top of that, maximizing mutual information can lead to degenerate behaviours in high-dimensional environments. By manipulating a small subset of the dimensions, a lot of information can be encoded, without exploring the rest of the state space.

Open-ended learning (Wang et al., 2019; Campero et al., 2020; Dennis et al., 2020; Wang et al., 2020; Ecoffet et al., 2021; Stooke et al., 2021) is closely related to unsupervised skill discovery. However, most approaches require either a parameterizable environment (Wang et al., 2019; 2020), some fixed encoding of tasks (Stooke et al., 2021) or self-competition (Silver et al., 2017a; Baker et al., 2019). This limits the applicability to environments that are engineered with these restrictions in mind. In contrast, neural networks are universal function approximators and thus, our approach can encode any possible task in any possible environment, as long as the input is chosen appropriately.

Another related line of research to our approach is intrinsic motivation. (Stadie et al., 2015; Bellemare et al., 2016; Pathak et al., 2017; Burda et al., 2018b;a; Raileanu & Rocktäschel, 2020). These approaches have managed great success in hard-exploration Atari games. However, these approaches do not learn discrete skills that can be composed or fine-tuned for fast learning of new tasks. This has limited their applicability to robotic environments where exploration is not usually the limiting factor.

There has been previous work using neural networks to output reward to RL agents. Compared to our work, they use supervision to train the reward function. Many of them train a reward function trained with states/trajectories labeled by supervisors in some fashion (Abbeel & Ng, 2004; Fu et al., 2017; Singh et al., 2019; Li et al., 2021). Other approaches train auxiliary rewards with meta-learning (Zheng et al., 2018; Du et al., 2019; Veeriah et al., 2019) to enhance the learning of the original reward function.

Another approach to train multiple behaviours is goal-conditioned learning (Kaelbling, 1993; Schaul et al., 2015; Andrychowicz et al., 2017; Rauber et al., 2017; Nair et al., 2018; Veeriah et al., 2018; Warde-Farley et al., 2018; Pong et al., 2019; Choi et al., 2021). In automated curriculum learning (Bengio et al., 2009; Florensa et al., 2017b; Forestier et al., 2017; Graves et al., 2017; Sukhbaatar et al., 2017; Florensa et al., 2018; Matiisen et al., 2019; Narvekar et al., 2020; Portelas et al., 2020a;b; Zhang et al., 2020), a sequence of goals is created such that each of them is not too hard nor too easy for the current agent. These approaches mostly rely on low-dimensional goal embeddings. When dealing with high-dimensional observations, they must use dimensionality reduction techniques. These techniques can introduce instabilities or destroy relevant information from the input. Our approach, on the other hand, can deal with high-dimensional inputs directly. On top of that, goals encode a narrow region of the state space, while each of our reward functions can be rewarding in a large region. This speeds-up the exploration in 'easy' regions of the state space.

3 METHOD

We introduce a method that performs open-ended, unsupervised skill discovery. It iteratively creates pairs of neural reward functions R_ψ and policies π_θ trained to maximize the corresponding R_ψ. Our proposed method alternates between increasing the complexity of the reward function R_ψ and leveraging the previously learnt skills to learn a policy π_θ that can solve the new R_ψ. This yields a general learning procedure that learns complex skills in a diverse set of environments. See Figure 1 for a high level overview.

3.1 INCREASING THE COMPLEXITY OF R_ψ

The reward function in a fully observable Markov Decision Process (MDP) is by definition a function of the current observation, the next one and the action that was performed. However, in many cases this can be reduced to a function of just the current observation and because of this, we opt for these simpler reward functions as the basis of our R_ψ. In our method, the reward function R_ψ is a neural network which takes observations o_t as input and outputs a single scalar value, the instantaneous reward r_t^ψ.

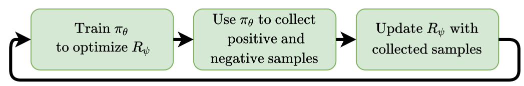


Figure 1: Main steps of our algorithms open-ended training loop.

Assume that we already have a policy π_θ that can reach states in the MDP which are rewarding under R_ψ. To increase the complexity of the reward function, we want to do the following:

- **Decrease the reward** of states which are visited by π_θ, as we already have a policy that can reach these states. We will create a dataset O_{neg} of such states, which we will refer to as *negative samples*.
- **Increase the reward** of states that can almost be reached by the current policy. This will allow us to leverage π_θ to learn the new reward function. We will create a dataset O_{pos} of such states, which we will refer to as *positive samples*.

To generate the negative samples, we run π_θ for a given number of steps (ideally until it reaches rewarding states) and store the visited states. To then generate positive samples, we change to performing random actions[1] for a fixed number of steps. To ensure that the new reward function is different from all previous ones, we also keep track of all the negative samples that we have collected for all skills in a dataset O_{neg_all}.

Finally, we set target values a and $-a$ for the positive and negative samples respectively, and train the reward network using standard supervised learning on the following loss:

$$\mathcal{L}_\psi = \sum_{o \in O_{neg}} \frac{(R_\psi(o) + a)^2}{|O_{neg}|} + \sum_{o \in O_{neg\_all}} \frac{(R_\psi(o) + a)^2}{|O_{neg\_all}|} + \sum_{o \in O_{pos}} \frac{(R_\psi(o) - a)^2}{|O_{pos}|}$$

This loss ensures that positive samples that have never been seen before, will have positive reward in the next R_ψ, while all other samples that have been seen before will decrease their reward. In the Reinforcement Learning phase we clip rewards to the $[0, a]$ range. This ensures that the agent seeks only positive samples, rather than less negative ones.

3.2 FORWARD TRANSFER FOR π_θ

Given the procedure presented in Section 3.1, we create increasingly complex reward functions. While this is great for open-ended learning, it eventually leads to skills that are too complex and cannot be learnt from scratch. In order to learn these skills, we must leverage previous knowledge about the environment. In this section we present several forward transfer mechanisms that are necessary for the most complex skills.

Our method so far can be combined with any standard Reinforcement Learning (RL) technique but we will focus on Actor-Critic methods like Advantage Actor-Critic (Mnih et al., 2016) or Proximal

[1] In MDPs with discrete actions we sample actions u.a.r. and in the continuous case, we keep the mean of π_θ and increase the standard deviation.

Policy Optimization (Schulman et al., 2017) for learning π_θ. These methods have a value network that's separate from the policy one which will allow us to transfer the most knowledge from the previous agent. Also, these approaches work both for continuous and discrete action spaces, allowing us to use the same technique for robotic environments or for 2d navigation tasks. Finally the learnt policies are stochastic, which increases the diversity of negative samples and speeds up the skill discovery process. See Section 4.1.2.

We present our three forward transfer mechanisms below. They all exploit the similarity between consecutive reward functions to ensure that even very complex reward functions can be solved by the RL agent in a reasonable amount of environment interactions.

- **Value Reuse:** Initialize the value network to the final value network of the previous agent. While two consecutive reward functions are different, both still reward close-by regions of the state space. Thus, by keeping the previous value function, the policy network will be nudged towards that region of the state space from the very first gradient updates.
- **Policy Feature Reuse:** Initialize the policy network to the final policy of the previous agent but setting the weights of the final layer to 0. This keeps the features learnt before, but outputs a uniform policy over all actions (or mean 0 and a fixed standard deviation in the continuous case), which allows for proper learning and exploration[2].
- **Guiding Policy:** Act with the previous policy for a random number of steps at the beginning of each episode. This heavily simplifies the exploration problem. The agent will start exploring from states that are much closer to the rewards defined by R_ψ. This is because the previous policy could already solve the previous reward function. In contrast to the other two mechanisms, this one does not rely on initialization. This means that it is the most effective in very sparse reward functions that will need many parameter updates to be learnt.

In Section 4.1.1 we individually evaluate these three techniques and show that their combination is necessary in complex environments.

Putting everything together we get an algorithm that can both, learn reward functions that encode increasingly complex behaviours and learn RL agents that solve those reward functions. Figure 1 illustrates the main steps of our training loop and Algorithm 1 shows the steps in more detail.

4 EXPERIMENTS

We now proceed to experimentally test our method. First, in Section 4.1, we thoroughly test all different components of our model in a 2d navigation task. This task allows us to verify the function of each component and also to explicitly visualize what each reward function is encoding.

Then, in Section 4.2, we move to BRAX robotic environments (Freeman et al., 2021). These have the most flexibility and thus allow the agent to learn very complex tasks. In these tasks, we evaluate the complexity of our skills by measuring their zero-shot transfer ability to the environment rewards. In the HUMANOID environment, our unsupervised skills outperform supervised agents trained for tens of millions of time steps. We also compute the one dimensional particle-based mutual information metric that has been proposed in the literature before (Gu et al., 2021) and show that our method outperforms previous approaches, even when other approaches only consider handcrafted feature dimensions in their objective.

Finally, in Section 4.3, we apply our method to Montezuma's Revenge. We show that the learnt reward functions keep getting increasingly complex and we are mostly limited by the amount of compute that it takes to learn each new reward function.

4.1 2D NAVIGATION TASK

The task consists of a 32 by 32 maze with several walls and a "danger zone". pppThe observation is given as a 32x32x1 image with all values set to 0, except a 1 in the current position of the agent. The agent always starts in the top left corner and can always perform 5 actions, either move in one of

[2]Policies at the end of learning can be very deterministic, which slows-down or completely stops learning of new reward functions.

the cardinal directions or stay in the current position. If the agent moves into a wall it will stay at its current position instead. If the agent is in the "danger zone" and moves up, moves down or stays, the episode is terminated and the agent is moved back to the starting position. Figure 2 shows the layout of the maze. The "danger zone" ensures that random exploration will not work to reach many parts of the environment and lets us easily test both the increasing complexity of the reward functions and the importance of forward transfer. We computed the expected number of steps to reach the bottom right corner with a random walk using Dynamic Programming. In expectation, $7 \cdot 10^{27}$ episodes are needed to do so. This shows that this maze is difficult to navigate.

To train our agent we use the Advantage Actor Critic (A2C) (Mnih et al., 2016) algorithm. To learn the rewards we use the full algorithm presented in Section 3. We use the same architecture for the reward, policy and value networks, but do not share any parameters. The architecture is a ReLU network with 2 convolutional layers followed by 2 fully connected layers. For the exact hyper-parameters see Table 3 in Appendix B. Figure 2 plots the most visited locations for each skill of one run of our algorithm. The first few skills visit points near the origin, later skills start moving to harder to reach parts of the state space. After roughly 40 iterations they reach the bottom right part. As stated before, this would take unreasonably long when using only random exploration.

4.1.1 Forward transfer of skills

As pointed out in Section 3.2, our reward functions become too complex to be learned from scratch with random exploration in a reasonable number of steps. When this happens, our agent must rely on transferring knowledge from previous generations. Our navigation task is specifically designed to test this transfer ability, as random exploration would never reach the bottom right corner ($7 \cdot 10^{27}$ episodes in expectation).

We experimentally evaluate the three forward transfer mechanisms proposed in Section 3.2: **Value reuse**, **Policy feature reuse** and **Guiding policy**. The reward functions from Figure 2 serve as tasks, sorted according to creation order. We train ablations of the three mechanisms sequentially on these tasks. This allows us to ignore the skill discovery process and only measure the forward transfer of skills. We repeat each experiment three times. The agent with all mechanisms, always manages to solve[3] all reward functions. On the other hand, the Policy, Value and Guiding ablations fail to learn after solving 29.3 ± 9.5, 18.7 ± 11.6 and 19 ± 8.5 reward functions, respectively.

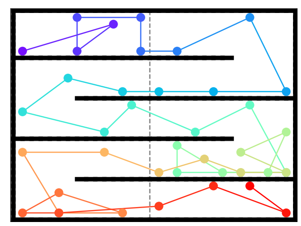


Figure 2: 40 skills discovered by our method in the 2d maze environment. The circles represent the average position of the most visited locations for each skill. Colors change from early skills in purple to late skills in red. Consecutive reward functions are connected by a line. The agent learns increasingly complex skills, until it reaches the bottom right corner of the maze; the hardest part to reach in the whole maze. The black bars represent the walls in the maze and the dotted line represents the danger zone.

4.1.2 Speeding-up deep exploration

One key parameter when training actor critic methods is entropy regularization. In our method, policies with a lot of entropy will generate a diverse set of negative samples. Diverse negative samples lead to reward functions that evolve more in each generation. This is especially beneficial in environments where many steps are necessary to reach certain states, like in this 2D navigation task or Montezuma's Revenge. We empirically verify this claim by training a set of agents with varying levels of entropy regularization. Figure 6 shows the coverage of the state space after 60 generations as a function of the entropy regularization. We observed that higher entropy leads to a faster coverage of the state space up to a certain threshold. However, too much entropy leads to poor policies that do not learn to reach the rewarding states when these are far enough from the origin. Observe that this problem arises independent of the training procedure as the entropy regularization changes what the optimal policy is.

In order to have policies that gather diverse samples to change the reward function and that can also be deterministic in dangerous parts of the trajectory that already have no reward, we use adaptive entropy regularization. That is, we apply a small entropy regularization term everywhere. We increase

[3] We consider a reward function as solved if the agent manages to find positive reward.

the entropy regularization in states where the reward function is positive (states in which we want to decrease the reward function). By doing this, we manage to consistently visit all states in the maze. This technique was necessary to create the reward functions from Figure 2.

4.2 Robotic Environments

Quantitatively measuring Unsupervised Reinforcement Learning progress remains an open problem and active area of research. Because of this, it is still important to visualize and qualitatively study the learned skills. We do this in Section 4.2.1. In the next two Sections, 4.2.2 and 4.2.3, we quantitatively measure the performance of our algorithm on downstream task performance and with the so-called particle-based mutual information metric, respectively.

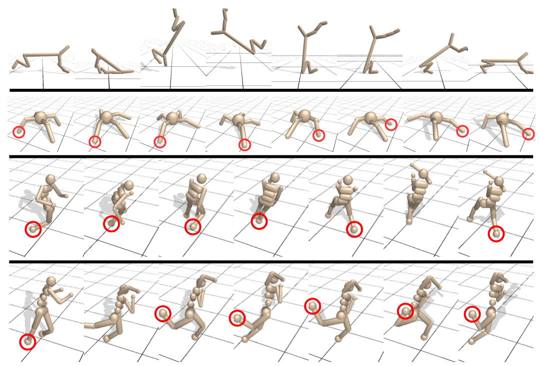


Figure 3: Visualization of four learned robotic skills. One of the legs is tracked with a red circle when it is visible. The Half-Cheetah learns to do a front-flip. The Ant does a partial rotation around its torso and then starts running. The upper Humanoid also does a partial rotation and then runs backwards. The lower Humanoid jumps on one leg while moving forward.

In contrast to the previous experiments, the action space is continuous and the input modality is now a vector of features, like relative position, angle and speed of the different joints. We do not use adaptive entropy here, as we need deterministic policies to maximize the downstream task performance and the mutual information metric. Given that the environments do not have far away regions to reach, we did not implement the guiding algorithm here. Note that the agent does not see the x-y position. Hyperparemeters and architecture can be found in Appendix C.

4.2.1 Qualitative Analysis

In Figure 3 we illustrate a selection of particularly interesting skills. In Figure 4 we see how the velocity of learnt skills in Ant and Humanoid evolve over training. See Appendix E for the other runs. One can see that consecutive skills slightly change the direction and speed of moving. However, it is important to realize that the observations contain tens to hundreds of dimensions and thus, the skills can encode much more complex behaviors than speed/direction of movement. For example, Figure 3 shows that the skill involves keeping one leg in the air on top of moving in the right angle/speed.

4.2.2 Zero-Shot Transfer

After discovering 50 unsupervised skills, we identify the skill that aligns best with the reward given by the environment. We report this zero-shot[4] performance in Table 1. As a baseline, we train an agent from scratch and measure how long it takes it to reach an equivalent performance. We use the Optuna hyperparameter optimizer (Akiba et al., 2019) to find hyperparameters which maximize the speed of learning beforehand. The exact procedure can be found on Appendix D.

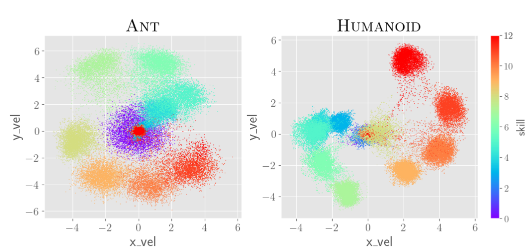


Figure 4: Scatter-plots of the x- and y-velocity of the states visited by the first 13 skills of one run in Ant and Humanoid. The second half of the trajectory is shown. Colors change from early skills in purple to late skills in red.

[4]Technically, we query the true reward function to identify the skills, but we do not perform any training on it.

The simplest environment, HALF-CHEETAH , benefits the least from unsupervised learning, while the hardest one, HUMANOID, benefits the most. This is in line with what is observed in Computer Vision and Natural Language Processing, of more complex tasks benefiting the most from a longer pre-training phase.

4.2.3 PARTICLE-BASED MUTUAL INFORMATION METRIC

Measuring how well the agent can control relevant state dimensions is another way to track progress in Unsupervised Reinforcement Learning. This is measured using the mutual information between state dimensions and skills. While high-dimensional estimation of mutual information an active area of research, sampling can be an effective form of estimation in the 1-dimensional case, c.f. Algorithm 1 in (Gu et al., 2021). We report the particle-based mutual information for the x-velocity in Table 2, using the same bucketing strategy[5] as in (Gu et al., 2021).

Table 1: Zero-shot environment reward of our algorithm and the number of steps a supervised PPO agents needs to match it. Both columns averaged over 10 repetitions.

Task	Zero-shot reward	Steps from scratch
Cheetah	1094 ± 1130	$340K \pm 50K$
Ant	2506 ± 511	$1.2M \pm 0.24M$
Humanoid	9092 ± 1063	$55M \pm 27M$

Our method far outperforms DIAYN, when both methods look at the whole observation state. DIAYN achieves diversity by learning a set of skills that can be correctly labeled by a neural network. In high-dimensional spaces this is easy to do by relying on a small subset of all state dimensions. This leads to non-diverse behaviours across most state dimensions. Even when expert knowledge about relevant dimensions is supplied to other methods, i.e. only taking the x- and y-velocity into account, our method still fares well. Particularly, in the most complex environment, HUMANOID, our method comes on top. We believe the fact that our method does not partition the space, leads to a narrower coverage of the state space per skill (see Figure 4). Also the iterative increase in complexity leads to better coverage of hard to reach regions of the state, e.g. high velocity. With all this, our approach achieves greater controllability of the x-velocity without any kind of feature engineering.

4.3 MONTEZUMA'S REVENGE

To show the generality of our approach we evaluate it on the notoriously hard Montezuma's Revenge Atari game. In this game, the agent controls a character in a complex 2d world with several rooms. Appendix G shows the initial room and various items with which the agent may interact. Same as in (Mnih et al., 2015), the observation is a stack of the last 4 frames. This gives the agent information about speed and direction of movement. This is done for all networks, that is, the value, policy and neural reward networks. We use the same simple CNN architecture as in (Mnih et al., 2015) for all three networks. For exact learning details see Appendix H.

Table 2: Particle-based mutual information metric for the x-velocity. Results are averaged over 10 runs. Algorithms with feature engineering only consider x-y velocities. Baselines taken from (Gu et al., 2021)

Task	Method	Feature Engineering	MI(s, z)
Cheetah	Ours	✗	1.40 ± 0.21
Cheetah	DIAYN	✗	0.49 ± 0.16
Cheetah	**DIAYN**$_p$	✓	$\mathbf{1.82 \pm 0.20}$
Cheetah	GCRL	✓	1.63 ± 0.16
Ant	**Ours**	✗	$\mathbf{1.33 \pm 0.11}$
Ant	DIAYN	✗	0.03 ± 0.01
Ant	$DIAYN_p$	✓	1.12 ± 0.27
Ant	GCRL	✓	1.22 ± 0.19
Humanoid	**Ours**	✗	$\mathbf{1.29 \pm 0.25}$
Humanoid	DIAYN	✗	0.07 ± 0.01
Humanoid	$DIAYN_p$	✓	0.93 ± 0.13
Humanoid	GCRL	✓	0.77 ± 0.15

Our algorithm uses finite episode lengths because once it reaches a rewarding state, the agent can stay there forever. Because

[5]We split the dimension in 1000 bins in the $[-10, 10]$ range. As in (Gu et al., 2021), we only take the second half of each trajectory. The initial state is independent of the skill and thus the beginning of the trajectory does not tell anything about the gained controllability.

of this, we reset the environment every 500 steps.

One of the main difficulties when dealing with Montezuma's Revenge is that it cannot be simulated as fast as the other studied environments. On top of that, an agent can learn hundreds of *different* skills without ever leaving the first room. Finally, skills learnt by our agent evolve from simple to very complex and extended in time. In the beginning, the agent just needs to stay close to the initial position. By the end of training, the agent learns to collect a key, open the door, avoid several enemies and visit four different rooms. In the most complex skills, it takes the agent several hundred steps to reach a rewarding state. This means that experiments can take several days to visit a different room.

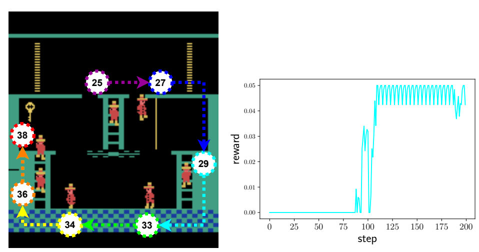
In order to save computation, we adapt the number of training steps to ensure the agent has learnt each skill. See Appendix F for details.

We found out that the first thing the agent learns is to get the life counter to 0. Then it starts exploring the room, with 0 lives already. This is because losing a life happens very easily during random exploration and because once the agent reaches 0 lives, all future positive samples will have 0 lives. This has two side effects. On the one hand, the initial $100 - 200$ steps of an episode are spent losing lives. On the other hand, exploration is harder, as the agent will reach a terminal state very easily. Because of this, we cut out the part of the image that shows the remaining lives, see Appendix G.

Figure 5(a) visualizes different skills. The first observations which get the maximum possible reward are overlaid. All the shown skills go to the bottom and kill the skull first. Then, they move to a specific location. It can be seen that later skills explore states further and further away from the initial state. Figure 5(b) shows that the skill receives no reward for many steps. Only after the agent has killed the skull, being on the ladder becomes rewarding[6]. This shows that both the reward network and the agent have learnt a lot of the concepts that are necessary to tackle Montezuma's Revenge. This includes controlling the agent in the 2d environment, killing enemies and interacting with different objects. All of this without ever accessing the original reward function. Eventually, after learning a few hundred skills, the neural reward function pushes the rewarding states all the way to the fourth room.

5 CONCLUSION

We have presented an unsupervised Reinforcement Learning algorithm that uses reward functions encoded by neural networks. Our algorithm alternates between increasing the complexity of the reward function and transferring previous knowledge to learn a new skill that finds rewarding states. This allows it to learn an unbounded number of skills that is mostly limited by the available compute power.



(a) (b)

Figure 5: (a) The position of the agent in the most rewarding state of each skill. The skills are numbered with the creation order. (b) The rewards received by skill 29 during part of its trajectory. This shows the long term nature of the discovered skills.

We have thoroughly tested the different components of our model in a 2d navigation task. This has allowed us to better understand our method in practice. We have shown that our method works both with high dimensional feature inputs, in robotic environments, and pixel inputs, in Montezuma's Revenge. Our algorithm learned a diverse set of skills in both settings. In HUMANOID and Montezuma's Revenge, skills found by our method achieve a zero-shot performance that takes millions of steps to learn in the classical Reinforcement Learning setup.

Our approach is very general and opens up many possibilities for future research. The forward transfer mechanism can be replaced by a more complex meta-learning or off-policy relabelling techniques. Collecting data for reward function training can benefit from smarter exploration strategies. We

[6]Using the ladder is needed to kill the skull, but the reward function gives no reward when the agent is on the ladder before killing the skull.

believe our algorithm is one step in a direction that may one day allow Reinforcement Learning agents to fully understand an environment without making use of any predefined reward function. Just like in Computer Vision and Natural Language Processing, this will lead to agents that need very few labels from the task at hand to be able to solve it and will drastically expand the applicability of Reinforcement Learning.

6 ACKNOWLEDGMENTS

We thank the reviewers for their useful comments and will incorporate them in a future revision of the paper.

Robert Meier and Asier Mujika were supported by grant no. CRSII5 173721 of the Swiss National Science Foundation.

REFERENCES

Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 1, 2004.

Joshua Achiam, Harrison Edwards, Dario Amodei, and Pieter Abbeel. Variational option discovery algorithms. *arXiv preprint arXiv:1807.10299*, 2018.

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2623–2631, 2019.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *arXiv preprint arXiv:1707.01495*, 2017.

Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula. *arXiv preprint arXiv:1909.07528*, 2019.

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29:1471–1479, 2016.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, 2009.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*, 2018a.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018b.

Andres Campero, Roberta Raileanu, Heinrich Küttler, Joshua B Tenenbaum, Tim Rocktäschel, and Edward Grefenstette. Learning with amigo: Adversarially motivated intrinsic goals. *arXiv preprint arXiv:2006.12122*, 2020.

Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. *arXiv preprint arXiv:2104.14294*, 2021.

Jongwook Choi, Archit Sharma, Honglak Lee, Sergey Levine, and Shixiang Shane Gu. Variational empowerment as representation learning for goal-based reinforcement learning. *arXiv preprint arXiv:2106.01404*, 2021.

Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. *arXiv preprint arXiv:2012.02096*, 2020.

Yali Du, Lei Han, Meng Fang, Ji Liu, Tianhong Dai, and Dacheng Tao. Liir: Learning individual intrinsic reward in multi-agent reinforcement learning. 2019.

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. First return, then explore. *Nature*, 590(7847):580–586, 2021.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

Carlos Florensa, Yan Duan, and Pieter Abbeel. Stochastic neural networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1704.03012*, 2017a.

Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. In *Conference on robot learning*, pp. 482–495. PMLR, 2017b.

Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In *International conference on machine learning*, pp. 1515–1528. PMLR, 2018.

Sébastien Forestier, Rémy Portelas, Yoan Mollard, and Pierre-Yves Oudeyer. Intrinsically motivated goal exploration processes with automatic curriculum learning. *arXiv preprint arXiv:1708.02190*, 2017.

C Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax-a differentiable physics engine for large scale rigid body simulation. 2021.

Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. *arXiv preprint arXiv:1710.11248*, 2017.

Alex Graves, Marc G Bellemare, Jacob Menick, Remi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. In *international conference on machine learning*, pp. 1311–1320. PMLR, 2017.

Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. Variational intrinsic control. *arXiv preprint arXiv:1611.07507*, 2016.

Shixiang Shane Gu, Manfred Diaz, Daniel C Freeman, Hiroki Furuta, Seyed Kamyar Seyed Ghasemipour, Anton Raichuk, Byron David, Erik Frey, Erwin Coumans, and Olivier Bachem. Braxlines: Fast and interactive toolkit for rl-driven behavior engineering beyond reward maximization. *arXiv preprint arXiv:2110.04686*, 2021.

Leslie Pack Kaelbling. Learning to achieve goals. In *IJCAI*, pp. 1094–1099. Citeseer, 1993.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kevin Li, Abhishek Gupta, Ashwin Reddy, Vitchyr H Pong, Aurick Zhou, Justin Yu, and Sergey Levine. Mural: Meta-learning uncertainty-aware rewards for outcome-driven reinforcement learning. In *International Conference on Machine Learning*, pp. 6346–6356. PMLR, 2021.

Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. Teacher–student curriculum learning. *IEEE transactions on neural networks and learning systems*, 31(9):3732–3740, 2019.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937. PMLR, 2016.

Shakir Mohamed and Danilo Jimenez Rezende. Variational information maximisation for intrinsically motivated reinforcement learning. *arXiv preprint arXiv:1509.08731*, 2015.

Ashvin Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. *arXiv preprint arXiv:1807.04742*, 2018.

Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *arXiv preprint arXiv:2003.04960*, 2020.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pp. 2778–2787. PMLR, 2017.

Vitchyr H Pong, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine. Skew-fit: State-covering self-supervised reinforcement learning. *arXiv preprint arXiv:1903.03698*, 2019.

Rémy Portelas, Cédric Colas, Katja Hofmann, and Pierre-Yves Oudeyer. Teacher algorithms for curriculum learning of deep rl in continuously parameterized environments. In *Conference on Robot Learning*, pp. 835–853. PMLR, 2020a.

Rémy Portelas, Cédric Colas, Lilian Weng, Katja Hofmann, and Pierre-Yves Oudeyer. Automatic curriculum learning for deep rl: A short survey. *arXiv preprint arXiv:2003.04664*, 2020b.

Roberta Raileanu and Tim Rocktäschel. Ride: Rewarding impact-driven exploration for procedurally-generated environments. *arXiv preprint arXiv:2002.12292*, 2020.

Paulo Rauber, Avinash Ummadisingu, Filipe Mutz, and Juergen Schmidhuber. Hindsight policy gradients. *arXiv preprint arXiv:1711.06006*, 2017.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International conference on machine learning*, pp. 1312–1320. PMLR, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills. *arXiv preprint arXiv:1907.01657*, 2019.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017a.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017b.

Avi Singh, Larry Yang, Kristian Hartikainen, Chelsea Finn, and Sergey Levine. End-to-end robotic reinforcement learning without reward engineering. *arXiv preprint arXiv:1904.07854*, 2019.

Bradly C Stadie, Sergey Levine, and Pieter Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015.

Adam Stooke, Anuj Mahajan, Catarina Barros, Charlie Deck, Jakob Bauer, Jakub Sygnowski, Maja Trebacz, Max Jaderberg, Michael Mathieu, et al. Open-ended learning leads to generally capable agents. *arXiv preprint arXiv:2107.12808*, 2021.

Sainbayar Sukhbaatar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. *arXiv preprint arXiv:1703.05407*, 2017.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. Bert rediscovers the classical nlp pipeline. *arXiv preprint arXiv:1905.05950*, 2019.

Vivek Veeriah, Junhyuk Oh, and Satinder Singh. Many-goals reinforcement learning. *arXiv preprint arXiv:1806.09605*, 2018.

Vivek Veeriah, Matteo Hessel, Zhongwen Xu, Richard Lewis, Janarthanan Rajendran, Junhyuk Oh, Hado van Hasselt, David Silver, and Satinder Singh. Discovery of useful questions as auxiliary tasks. *arXiv preprint arXiv:1909.04607*, 2019.

Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O Stanley. Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions. *arXiv preprint arXiv:1901.01753*, 2019.

Rui Wang, Joel Lehman, Aditya Rawal, Jiale Zhi, Yulun Li, Jeffrey Clune, and Kenneth Stanley. Enhanced poet: Open-ended reinforcement learning through unbounded invention of learning challenges and their solutions. In *International Conference on Machine Learning*, pp. 9940–9951. PMLR, 2020.

David Warde-Farley, Tom Van de Wiele, Tejas Kulkarni, Catalin Ionescu, Steven Hansen, and Volodymyr Mnih. Unsupervised control through non-parametric discriminative rewards. *arXiv preprint arXiv:1811.11359*, 2018.

Yunzhi Zhang, Pieter Abbeel, and Lerrel Pinto. Automatic curriculum learning through value disagreement. *arXiv preprint arXiv:2006.09641*, 2020.

Zeyu Zheng, Junhyuk Oh, and Satinder Singh. On learning intrinsic rewards for policy gradient methods. *arXiv preprint arXiv:1804.06459*, 2018.

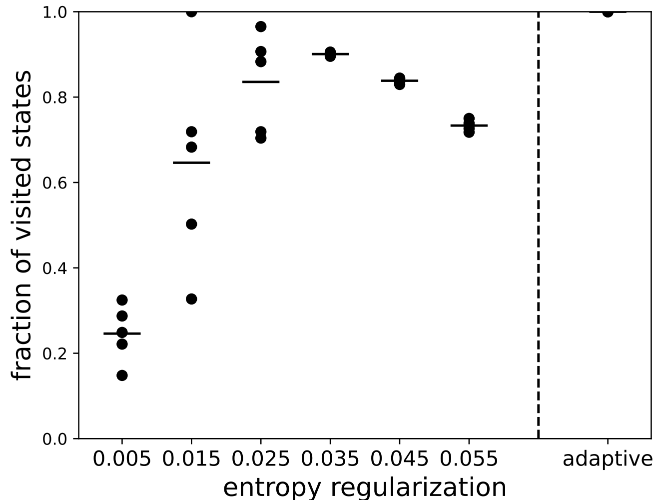


Figure 6: Total number of states visited by the learnt policies after 60 generations as a function of the entropy regularization used by the underlying A2C agent. The adaptive entropy model manages to visit all states in all runs.

A PSEUDO-CODE

Algorithm 1 Open-Ended Neural Reward Functions

Initialize θ_0 and ψ_0 and O_{neg_all}.
$i \leftarrow 0$
while TRUE **do**
  $O_{neg_i}, O_{pos_i} \leftarrow \emptyset, \emptyset$
  Set R_{ψ_i} as the reward of the MDP
  Train π_{θ_i} and V_{θ_i} using an actor-critic algorithm
  for $j = 0$ **to** b **do**
    Reset the MDP to the initial state
    Follow π_{θ_i} for k steps and add the observed states to O_{neg}
    Follow a random policy for k' steps and add the observed states to O_{pos}
  end for
  $O_{neg\_all_i} \leftarrow O_{neg\_all_{i-1}} \cup O_{neg_i}$
  $\psi_{i+1} \leftarrow \psi_i$
  Train ψ_{i+1} with $\mathcal{L}_\psi$, O_{pos_i}, O_{neg_i} and $O_{neg\_all_i}$
  $\theta_{i+1} \leftarrow \theta_i$
  Set last layer of $\pi_{\theta_{i+1}}$ to 0
  $i \leftarrow i + 1$
end while

B 2D NAVIGATION: EXPERIMENTAL DETAILS

The episode length in the 2d navigation task is 250 steps. The guiding phase lasts for $2/3$ of the total steps of the 1-steps A2C agent learning. We sample the guiding length uniformly at random in the 0 to 200 range. For the negative samples we follow the learnt policy for 200 steps. For the positive samples, we take random actions for 50 steps after following the policy for 200 steps.

Table 3: Hyperparameters for the 2d navigation task.

HYPERPARAMETER	
BASE ENTROPY REGULARIZATION	0.005
EXTRA ENTROPY REGULARIZATION	0.05
EPISODE LENGTH	250
A2C LEARNING RATE	0.0001
A2C DISCOUNT FACTOR	0.99
BATCH-SIZE	2048
STEPS PER SKILL	$2048 \cdot 60000$
POSITIVE/NEGATIVE SAMPLE TARGET VALUE	$0.05/-0.005$
REWARD NETWORK UPDATES PER SKILL	500
REWARD NETWORK LEARNING RATE	0.001
REWARD NETWORK TRAINING BATCH SIZE	$3 \cdot 256$

Table 4: Hyperparameters for PPO shared in all three environments.

BRAX PPO HYPERPARAMETERS	
POLICY HIDDEN LAYER SIZES	$[512, 512]$
VALUE HIDDEN LAYER SIZES	$[512, 512]$
BODY LEARNING RATE MULTIPLIER	0.5
EPISODE LENGTH	1000
ACTION REPEAT	1
NUMBER OF MINI-BATCHES	32
BATCH-SIZE	1024
PARALLEL ENVIRONMENTS	2048

C ROBOTIC ENVIRONMENTS: ARCHITECTURE AND HYPERPARAMETERS

We use the PPO (Schulman et al., 2017) implementation from (Freeman et al., 2021) with modifications such that it allows our training method to work. In Table 4 and Table 5 we list the hyperparemeters we used. Note that we use a smaller learning rate for the bodies of the policy and value network. We found that this was beneficial for transferring more knowledge from the previous skill. The environment specific parameters were found using Optuna (Akiba et al., 2019). For each environment we ran a search to optimize final performance on the environment rewards (running in positive x-direction). We used 200 runs and trained each of them for the same number of steps as a skill in our method. This yielded hyperparameters which are able to learn tasks in the corresponding environment. We did not take the downstream performance metrics (zero-shot perforamce and particle-based information) into account. Doing so would have exceeded our compute budget. In Table 6 the architecure and hyperparameters used for the neural reward functions are listed. For the supervised reward training we use the Adam optimizer (Kingma & Ba, 2014). All code can be found in the supplementary material.

D ROBOTIC ENVIRONMENTS: ZERO-SHOT TRANSFER

We use the Optuna hyperparameter tuning library (Akiba et al., 2019) to create Table 1. We ran multiple optimization procedures. For each one, we fix a number of environment steps, then optimize the final performance on the environment reward over 200 runs. We iteratively increased the number of timesteps until the highest score of the 200 runs beats our averaged zero-shot performance. We take the hyperparemters of that run and train 10 agents until they outperform our averaged zero-shot performance. The average number of training steps needed for this is reported in Table 1. In the HUMANOID environment three of the runs did not reach the score in $100M$ steps, at which point we stopped training. We nonetheless took $100M$ into the average. The code for the hyperparameter search and the results of our conducted studies can be found in the supplementary material.

Table 5: Environment specific PPO hyperparemeters.

	HALF-CHEETAH	ANT	HUMANOID
LEARNING RATE	0.00010	0.00029	0.00017
REWARD SCALING	0.24532	5.58242	0.15326
UNROLL LENGTH	3	5	6
UPDATES PER EPOCH	15	14	10
DISCOUNT FACTOR	0.99109	0.92318	0.99114
ENTROPY COST	0.00062	0.00200	0.02087
ENVIRONMENT STEPS PER SKILL	$10M$	$10M$	$20M$

Table 6: Neural reward function hyperparemeters for the BRAX environments.

NEURAL REWARD FUNCTION HYPERPAREMETERS IN BRAX ENVIRONMENTS	
HIDDEN LAYER SIZES	[87]
HIDDEN LAYER NONLINEARITY	TANH
TARGET VALUE a	5
GRADIENT STEPS	300
LEARNING RATE	0.001
TOTAL BATCH SIZE	171
NEGATIVE STEPS	300
POSITIVE STEPS	40
NUMBER OF SAMPLING ENVIRONMENTS	8192
FRACTION OF NEGATIVE SAMPLES STORED	0.01

E ROBOTIC ENVIRONMENTS: ADDITIONAL $x - y$ VELOCITY PLOTS

In Figure 7 and 8 one scatter plot as in Figure 4 for every run is shown.

F MONTEZUMA'S REVENGE: ADAPTING EPISODE LENGTH

In order to save computation, we adapt the number of training steps according to several criteria. This lets us save a lot of compute on the skills which are easy to learn. We use the following measures:

- We train for $1M$ steps with guiding from the previous policy. The number of guiding steps is sampled uniformly at random between 0 and 450, each time the environment is reset. Then we train for another $0.65M$ steps without any guiding.
- If the average reward goes down after removing the guiding or is too low at any point, we restart the guiding phase for $0.65M$ steps.
- If the agent is reaching a terminal state in more than 10% of the episodes, we continue training.

On top of this, we ignore positive samples that receive almost no reward. All these tricks enable us to considerably reduce the training time. However, they are not a core change in our algorithm as they could all be replaced by just training all generations for a longer fixed number of steps, just as before.

G MONTEZUMA'S REVENGE: ENVIRONMENT DETAILS

Figure 9 shows the initial state of Montezuma's Revenge and the cropped version of it that is used for our agent.

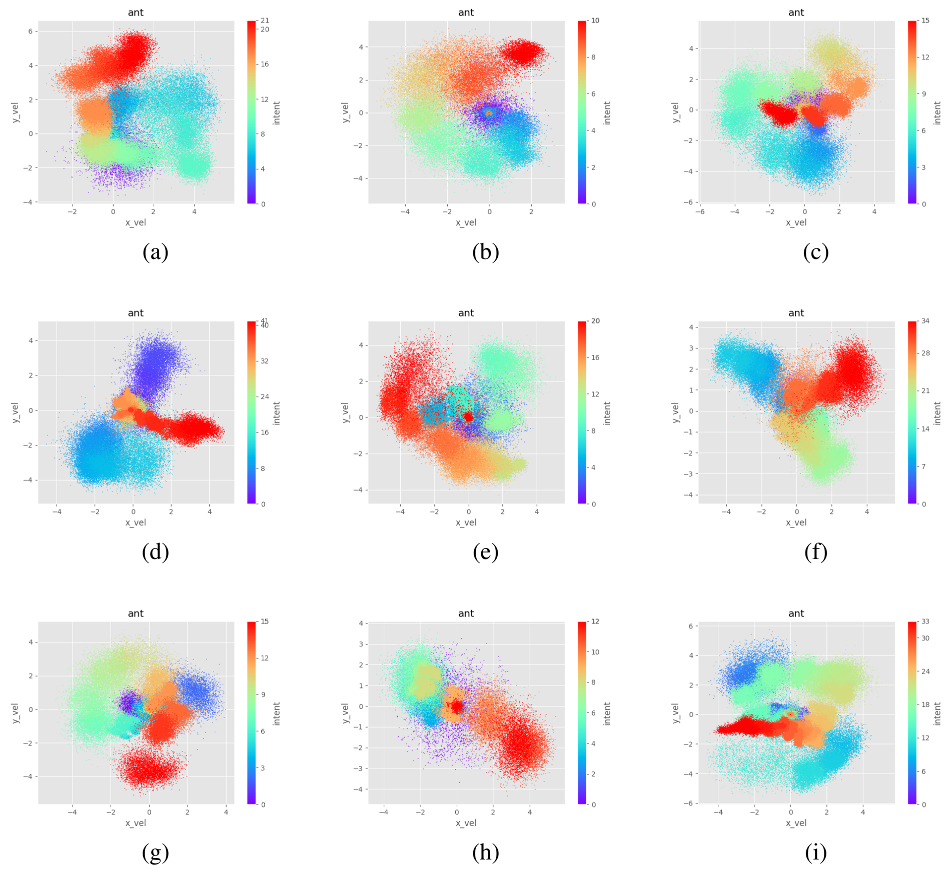


Figure 7: Scatter-plots of the x- and y-velocity dimension of the states visited by the first n skills of all additional 9 runs in ANT . n was picked by hand in each run. The second half of the trajectory is shown. Colors change from early skills in purple to late skills in red.

H MONTEZUMA'S REVENGE: EXPERIMENTAL DETAILS

In Montezuma's Revenge, we use the PPO implementation from the coax[7] library, with 1-step temporal differences. If the average score per 500 steps is below 5, we go back to the guiding phase. Due to the large memory requirements to store all negative samples, after the 15-th epoch we rewrite old negative samples to add the new ones. Table 7 summarizes our hyper-parameters.

[7] https://github.com/coax-dev/coax

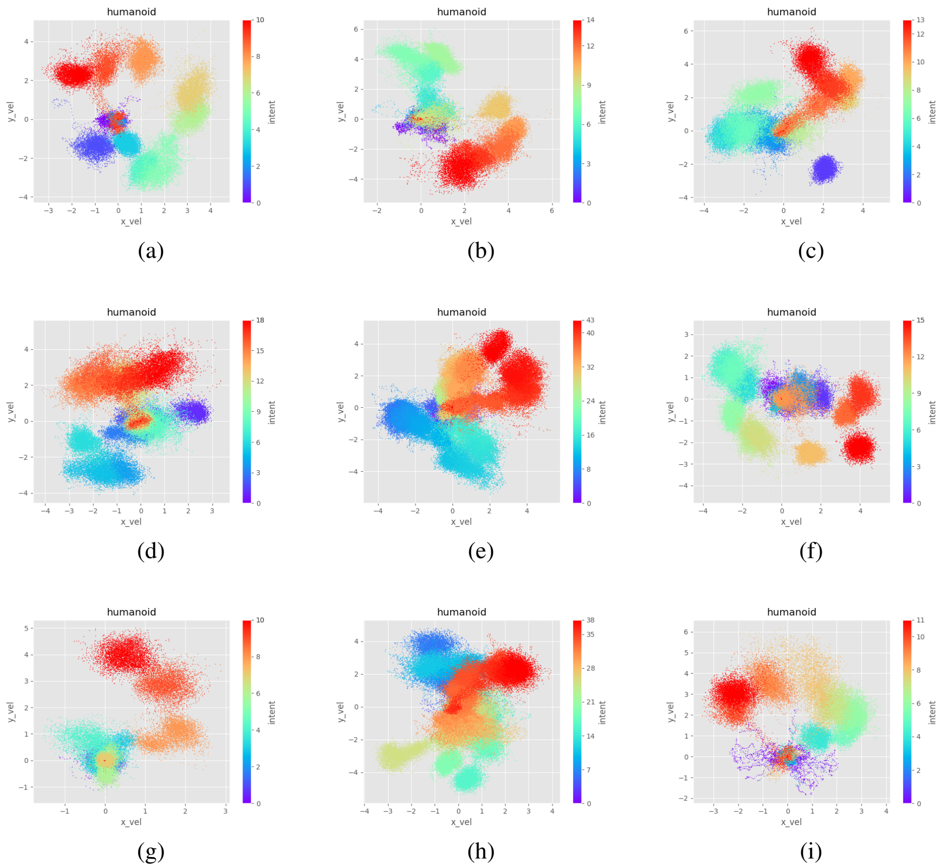
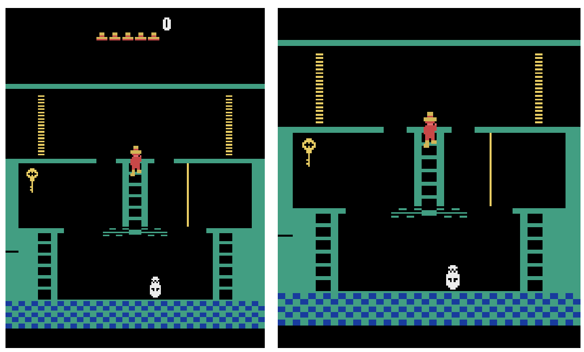


Figure 8: Scatter-plots of the x- and y-velocity dimension of the states visited by the first n skills of all additional 9 runs in HUMANOID . n was picked by hand in each run. The second half of the trajectory is shown. Colors change from early skills in purple to late skills in red.

Table 7: Hyperparameters for Montezuma's Revenge.

HYPER-PARAMETER	
BASE ENTROPY REGULARIZATION	0.003
EXTRA ENTROPY REGULARIZATION	0.03
EPISODE LENGTH	500
PPO LEARNING RATE	0.0003
PPO DISCOUNT FACTOR	0.99
PPO EPSILON	0.2
PPO BATCH SIZE	1024
PPO REPLAY BUFFER SIZE	4096
PPO REPLAY EPOCHS	4
PARALLEL ENVIRONMENTS	32
POSITIVE/NEGATIVE SAMPLE TARGET VALUE	$0.05/-0.05$
REWARD NETWORK UPDATES PER SKILL	1500
REWARD NETWORK LEARNING RATE	0.001
REWARD NETWORK TRAINING BATCH SIZE	$3 \cdot 243$



(a) (b)

Figure 9: (a) Initial state in the first room of Montezuma's Revenge. The agent controls the red/yellow character and its actions are moving up, down, right or left, and jumping. Touching the skull or jumping from a high height makes the agent lose a life. The agent must collect the key and open one of the two doors to access the next room. (b) Cropped version of the input that we use in our algorithm.