HIERARCHICAL RL USING AN ENSEMBLE OF PROPRIOCEPTIVE PERIODIC POLICIES

Kenneth Marino* **& Abhinav Gupta**
Carnegie Mellon University, Facebook AI Research
{kdmarino,abhinavg}@cs.cmu.edu

Rob Fergus
New York University, Facebook AI Research
fergus@cs.nyu.edu

Arthur Szlam
Facebook AI Research
aszlam@fb.com

ABSTRACT

In this work we introduce a simple, robust approach to hierarchically training an agent in the setting of sparse reward tasks. The agent is split into a low-level and a high-level policy. The low-level policy only accesses internal, proprioceptive dimensions of the state observation. The low-level policies are trained with a simple reward that encourages changing the values of the non-proprioceptive dimensions. Furthermore, it is induced to be periodic with the use a "phase function." The high-level policy is trained using a sparse, task-dependent reward, and operates by choosing which of the low-level policies to run at any given time. Using this approach, we solve difficult maze and navigation tasks with sparse rewards using the Mujoco Ant and Humanoid agents and show improvement over recent hierarchical methods.

1 INTRODUCTION

The notion of hierarchy is fundamental to AI systems. Effective approaches to perceptual tasks such as vision or audio rely on hierarchical neural network architectures (e.g. convnets and wavenets). Hierarchical methods are also a natural approach to solving reinforcement learning problems where reward is sparse. Consider an environment where a legged robot has to navigate a maze through winding paths to one of several possible goals. The sequence of actions (at the level of joint controls) required for this agent to ever see a reward is too long and complex for naïve reinforcement learning methods to find. Hierarchical controllers provide an intuitive solution to this problem: first learn a low-level controller that is able to handle low-level tasks such as locomotion and balance, and then learn a high-level controller that controls the inputs to that low-level controller and handles the high-level planning tasks (e.g. which path to take in the maze).

While conceptually straightforward, hierarchical approaches have been difficult to realize in practice for several reasons: (i) the lack of constraints on intermediate representations; (ii) the difficulty of learning a high-level controller when the low-level policies shifts, and (iii) the continued problem of sample-inefficiency when training both low-level and high-level controller jointly with sparse rewards. While there has been some progress recently, there is still no general hierarchical reinforcement learning method that is robust across domains, easy to implement, and effective.

The approach in this work avoids these problems by simplifying the high-level optimization problem to choosing from a set of independently trained low-level controllers. The problem then becomes how to train a variety of controllers which are generally useful for an agent acting in the world and are sufficiently diverse so that the high-level controller will always have access to the building blocks it needs to solve the high-level task.

*Work done as an intern at Facebook AI Research.
See videos and more at https://sites.google.com/view/hrl-ep3

The basic method of learning our low-level skills is as follows. First, as with Heess et al. (2016), we factorize the state space into components that describe the internal and external properties of the agent. The former carries information needed for proprioception (e.g. internal joint angles) and is exclusively used by the low-level policy and the latter is used exclusively by the high-level policy. Second, we provide a weak supervisory signal to the low-level policies that rewards them for changes in the external part of the space. We do not specify how; the low-level agent is rewarded for any change in the perceptual features. The intuition is that a policy is useful to an agent if following it changes the external world or the agent's place in it, (e.g. by changing position or moving objects). Finally, we build into the agent a notion of repeated cycles of action. Biological systems are known to possess central pattern generators (CPGs) that produce rhythmic outputs, assisting with perception, locomotion, and self-regulation (Marder & Bucher, 2001). Taking inspiration from this, we give our agent access to a "phase function" and reward it for acting in consistent, cyclic ways.

Using this method of training, we train a number of low-level controllers on Mujoco locomotion tasks and are able to learn a diverse and effective set of low-level policies. We then show that these policies are easily controllable by our high-level controller and sufficient for solving the final task. We show impressive results on a variety of difficult Mujoco maze and navigation tasks with sparse rewards and demonstrate that our method compares favorably to other recent hierarchical RL approaches. Finally, we demonstrate that our method is robust to choice of RL algorithm.

2 Related Work

There is an extensive literature on hierarchical approaches to reinforcement learning (Sutton et al., 1999; Dayan & Hinton, 1992; Dietterich, 1999). Options (Sutton et al., 1999) are perhaps the most standard framing for hierarchical RL, formalizing the notion of a subroutine in an MDP. Our low-level networks are options with a deterministic return condition. While earlier works used pre-specified subroutines, and focused on learning when to call them, recent works, for example Bacon et al. (2017); Vezhnevets et al. (2017); Nachum et al. (2018), have demonstrated success in learning multiple levels of hierarchy end-to-end from only task reward.

Still, in more complex settings such as the ones discussed in this work, with high dimensional action spaces, and especially with sparse rewards, performing unsupervised pre-training, or using auxiliary tasks during learning or pre-training have led to better results (Heess et al., 2016; Frans et al., 2017; Florensa et al., 2017; Haarnoja et al., 2018b; Eysenbach et al., 2018; Hausman et al., 2018).

One line of work in this direction starts from a variational inference approach to intrinsic motivation and exploration (Mohamed & Rezende, 2015; Gregor et al., 2016; Florensa et al., 2017; Eysenbach et al., 2018). They use an actor parameterized by state and a latent vector in such a way that the latent vector is predictable from a final state or a sequence of states the actor visits, but otherwise, the actions have high entropy. The latter of these works use this approach for designing hierarchical policies.

In Florensa et al. (2017), they use a pre-training task, and the prediction cost is an auxiliary reward. The main differences from our work is that we do not try to use stochastic neural networks, and we do not try to compress the one-hot representation of the low level networks into a dense vector during low level training. Instead we keep them fully separate. We also do not attempt to impose any regularization to encourage the low level networks to be diverse. These can be considered simplifications; what we show is that the simple thing works quite well.

In Eysenbach et al. (2018) the low-level policy is trained without using any environmental reward by pre-training using the unsupervised variational objective; they then training a high-level actor to issue commands via the latent vector. This latter work is in some senses more general than ours, in that a user is not required to separate out the proprioceptive variables from the external sensory variables. In addition, the formulation directly encourages diverse and controllable low-level policies. Nevertheless, both these works were validated mostly in the same settings we use here, where the separation between variables is natural, and the reward in equation 3.2 is well motivated. In these settings our method is simpler, and performs well in comparison (see Figure 7a).

Our work is most closely related to Heess et al. (2016). We operate under the same basic philosophy, separating the agent into a low-level component that only sees "proprioceptive" information, and works at a fine time scale, and a coarse agent that has access to "sensory" information. Like

that work, we evaluate or method in the setting of locomotive agents. However, there are several differences between Heess et al. (2016) and this work. There, the low-level component is trained along with a provisional high-level component to solve a set of high-level tasks designed for the agent and transfer tasks, whereas in this work, the low-level controller is trained with only the objective "change the sensory responses" in each setting. Furthermore, our underlying architecture is different. Whereas in Heess et al. (2016) the high-level agent interacts with the low-level component via continuous control vectors, in this work, we sample one of N low-level policies, as in Frans et al. (2017). In addition, whereas they train their low-level actor with only the current state, our low-level actor is shown to benefit from using a pattern generator. Finally, because of the structure of our low-level task, we need to ensure that the low-level policies are diverse and controllable (which in Heess et al. (2016) is given by the design of the pre-training task), and we do so by using multiple random instantiations of the "change the sensory response" policies that maximize equation 3.2. Our work can thus be considered in some ways a simplification and generalization of Heess et al. (2016).

The use of phase information in the low-level actor recalls Holden et al. (2017); Sharma & Kitani (2018). In the former, a neural network is used to control the gait of a video game character. The weights of the neural network are a function of a phase variable whose cycle corresponds to a cycle of the gait, and are trained as regressors on motion-capture data. The latter work takes this approach to RL problems that have an explicit cyclic structure, in particular the walker and hopper environments from MuJoCo. Phase has also been used in similar way in Peng et al. (2018; 2017; 2016). While we share a similar motivation with these, our low-level actor processes the phases in a different way: instead of a phase variable directly modulating the weight of the network, we concatenate an embedded phase index to the input of the network.

3 METHODOLOGY

Section 3.1 describes the division of the state space into the proprioceptive or internal state and the external state. In section 3.2 we describe the reward function which we use to train the low-level policies. These policies take in the proprioceptive state and the reward maximizes displacement in the external state. Section 3.3 describes how we augment the low-level policy by adding an additional input that tells the agent where it is in a "phase cycle" and constraining the motion of the agent to match that phase cycle. Finally, in section 3.4 we describe how we learn a high-level controller by learning to choose when and which of the trained low-level policies should be used.

3.1 PROPRIOCEPTIVE AND EXTERNAL STATES

We divide the agent's sensory output into two sets of variables. The proprioceptive states, s^p, are the measurements of the agents own body that can be directly affected by control. The external states, s^e are measurements of the environment outside of the agents direct control.

The proprioceptive part of the state is fed into the low-level policy network. This is the part of the state that gives the agent the information about the configuration of its own body and actuators. The low-level policy is independent of the external state as the low-level policies are intended to be generally usefully in moving in the external state independently of the exact location.

The external part of the state is fed into the high-level policy network. The high-level policy is responsible for things like moving to particular coordinates in the world, and so the external state is necessary. The high-level policy does not need the low-level states such as joint locations as that is handled by the low-level policy.

3.2 LOW-LEVEL POLICY TRAINING

We would like a set of low-level policies that are diverse, effective, and controllable. Here "effective" means that they can make significant changes to the environment. "Diverse" means they make different changes, hopefully covering all of the things the agent will need to be able to do in its test task. Finally "controllable" means that the low-level policies are easy for the high-level policy to use, for example because they are predictable and their effect is independent of the external state. Empirically, we have found that simply instantiating multiple randomly initialized neural networks trained to change the external perception suffices.

Formally, our basic low-level reward at each time step is

$$R_{disp}(s_t, a_t) = ||s^e_{t+1} - s^e_t||_2$$

Using this reward scheme, we train a number of agents (16 in the experiments below). We use the same reward for each agent, and across environments (except regularized with the standard environment-specific control and survival rewards). As we show in Section 4.1, changing only the random seed gives us a wide variety of policies that maximize the displacement of the agent.

3.3 PHASE-CONDITIONED POLICIES

One important aspect of our training procedure for our low-level policies is time index or *phase* inputs. We expect good movement policies to be at least roughly periodic, and phase inputs allow the model to more easily achieve periodicity. Formally, we denote our phase-conditioned policy as $\pi(s^p, \phi)$, where s_p is our proprioceptive observation and $\phi \in \{0, 1, \dots K-1\}$ is the phase index, where K is the length of the period (in all our experiments, we choose $K = 10$). At each time-step t, during training and evaluation, we receive our new observation s_p, and phase index $\phi = t \pmod K$.

In particular, in our phase-conditioned networks, we take as input s_p and a vector $b_\phi \in \mathbb{R}^d$, where b_ϕ is a learned parameter, like a bias term, for each possible phase indices ϕ. $d = 16$ in our experiments.

For our phase-conditioned policies, we also want to encourage the agents actions and internal states to be cyclic during training; at any given time step, the current action and proprioceptive state should match the one from the last cycle. Just as after one phase period, a pendulum is in the same state as it was before, during any given cycle, the behavior of the agent should be identical.

Formally, the reward maximization with a cyclic constraint can be written as:

$$\begin{aligned} \underset{a_t}{\text{argmax}} \quad & R(s_t, a_t) \\ \text{subject to} \quad & ||s^p_t - s^p_{t-K}||_2 \leq \sigma_s, ||a_t - a_{t-K}||_2 \leq \sigma_a \end{aligned}$$

This can be changed to be a cyclic loss as

$$\underset{a_t}{\text{argmax}} \quad R(s_t, a_t) - \lambda_s ||s^p_t - s^p_{t-K}||_2 - \lambda_a ||a_t - a_{t-K}||_2$$

For the phase conditioned policies, we include these losses in the reward function during training. In contrast to Holden et al. (2017); Sharma & Kitani (2018), the model need not be exactly periodic.

3.4 HIERARCHICAL TRAINING

Once we have learned a variety of low-level policies, we want to learn how and when to use these low-level policies in a way that lets us solve much more complex task with sparse rewards.

The way we compose our low-level policies is quite simple. Similar to Eysenbach et al. (2018) and many other works, at each time step, our high-level policy chooses one of our low-level policies to run. Given N trained low-level policies $\pi^{ll}(s^p_t, \phi)$, our high-level policy is

$$\theta = \pi^{hl}(s^e)$$

This is actually just a choice from among our low-level policies.

We go through the complete formulation for our high-level policy in Appendix A.

4 EXPERIMENTS

In our experiments, we test on a variety of difficult sparse reward problems simulated through Mujoco (Todorov et al., 2012). We use two popular and challenging agents: Ant and Humanoid.

In section 4.1 we show some results from our low-level training on Ant and Humanoid. In section 4.2 we show the results of our high-level policies on several different mazes and a navigation task.

4.1 LOW-LEVEL CONTROL

As described in Section 3.2 and Section 3.3, we train a number of low-level policies using our simple reward function. We train Ant and Humanoid in the default flat plane environments during this stage. Figure 1 shows the reward curves of our phase-conditioned low-level policies compared to standard networks. All networks are trained using PPO (Schulman et al., 2017). See Appendix D for more training and network details.

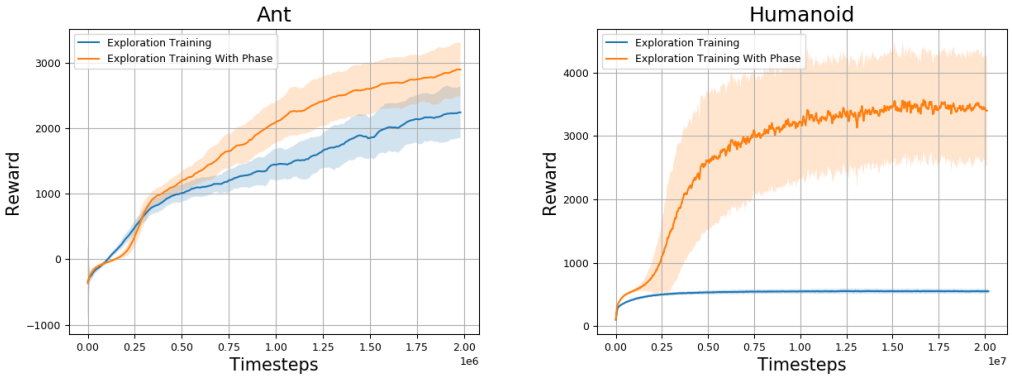


Figure 1: Environmental reward for our low-level policies, with and without phase conditioning For Ant, phase-conditioning gives a respectable boost, but both achieve reasonable rewards and easily move. For Humanoid, vanilla PPO (using a widely used implementation (Kostrikov, 2018)) using the same default hyperparameters performs poorly, even though it performs well for all other standard Open AI Gym Mujoco tasks. With the phase conditioning, however, using PPO, we are able to achieve good results with Humanoid surviving and moving forward.

In Figure 2 we show the traces of our low-level policies learned on Ant and Humanoid for 100 time steps. Each trajectory represents a different low-level policy. We can see that we learn a great variety of movements, more than sufficient for the high-level policy. We see that Humanoid moves more slowly, both because Humanoid is much more difficult to control, and because we use the standard OpenAI Gym (Brockman et al., 2016) simulator timesteps, which is shorter for Humanoid.

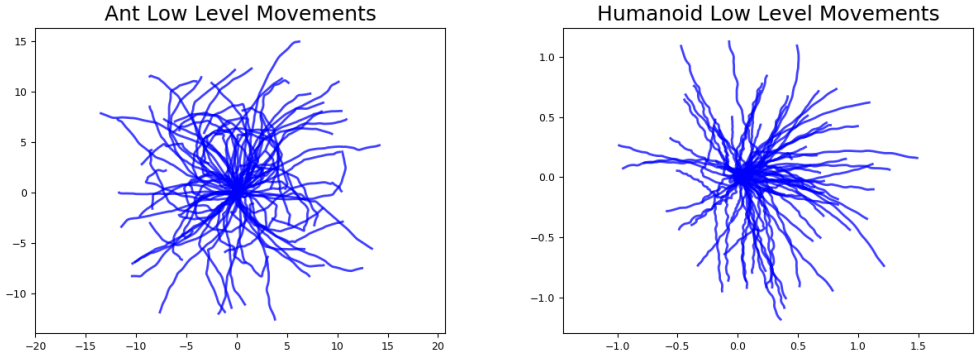


Figure 2: Traces of low-level policies after 100 timesteps of movement using different random seeds

4.2 MAZES AND NAVIGATION

The high-level policy network is trained according to Section 4.2, also using PPO. See Appendix E for more training and network details.

The first high-level environment that we evaluate on is the Cross Maze environment with fixed goals from Haarnoja et al. (2018a). As described in that work, the Cross Maze has three different goals along three different paths. In their version of this environment, three different environments are used for each of the three different goal locations and thus a different model is learned for each version of the maze. In Figure 3 we compare our high-level policy to the published numbers from their

method (SAC-LSP) and their baselines (see Haarnoja et al. (2018a) Figure 5). We used a different value for the length of the timestep for Ant in the simulator, so to make a fair comparison, we multiply our number of frames by 2.5 so that our x-axis compares the same amount of simulator time for high-level training. This comparison favors their method as we receive 2.5x fewer observations.

We can see that our method with a phase-input low-level policy converges faster than SAC-LSP and converges to a smaller final distance to goal, meaning that our method is both more efficient and more consistent on the task. Our method also has much smaller standard deviation across trials.

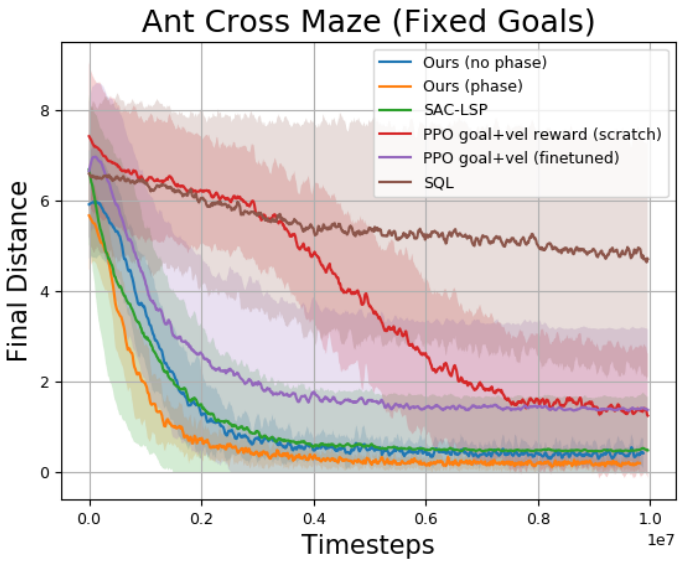


Figure 3: Results on Fixed Ant Cross Maze and comparison to Haarnoja et al. (2018a). We show the training curves for each method, showing the final distance of the agent to the goal.

One major limitation of this environment is that from the perspective of each model, there is just one fixed goal. So to solve it, the policy need only find the one fixed goal for that version of the environment and learn a policy to consistently move to it.

A much more difficult environment would be one where the goal is randomly chosen from a set of available options each episode. To solve this kind of maze, the policy would have to explore enough to find each goal during the fraction of episodes where it appears, and then consistent learn and keep separate paths of actions in its memory depending on the random goal location for the episode. We refer to these kinds of mazes as "random goal" mazes. All of the maze results except for Figure 3 use random goals.

Figure 4 shows the training curves for two random goal mazes: the cross maze from before and the "skull maze." See Figure 5 for the layout of these mazes. The skull maze has four possible goals instead of three and also forces the agent to make a decision immediately about which way to move, either to the top, left, or right corridor.

We compare our method to baselines similar to those used in Haarnoja et al. (2018a), all trained with PPO as is our method. The baseline models are either trained with or without the phase conditioning, and either from scratch, or finetuned (meaning that we initialize the network using a network trained on our low-level objective). We also give some of the baselines more information by also giving them a velocity reward during high-level training (meaning they are rewarded for movement of the agent).

Even with velocity rewards and pre-training on the low-level objective (both of which could be considered exploration bonuses), all of the baselines fail to get close to the goal locations. The problem of exploration and consistent navigation to different random goals is too difficult to learn from scratch.

Figure 5 shows traces of Ant navigating the cross and skull mazes. Most traces reach the goal, although some terminate early. All move fairly decisively towards the correct goal.

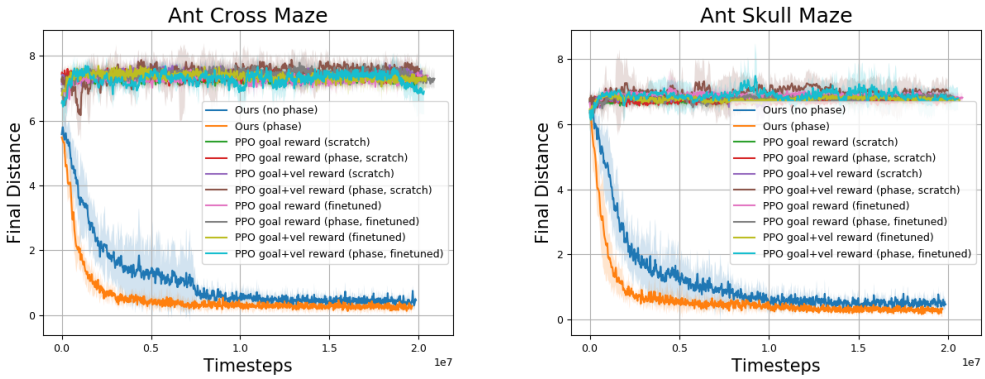


Figure 4: Training curves on the Random Ant Cross and Skull Maze environments. We again show the average final distance to the goal

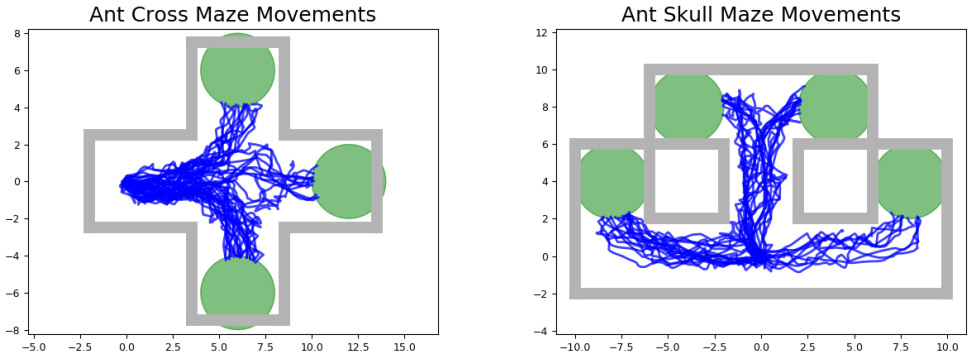


Figure 5: Traces of our method navigating in the ant Ant Cross and Skull Mazes

In Figure 6, we show the results using Humanoid on a smaller version of the cross maze. Because the PPO without phase fails at the low-level policy, we omit all non-phase results. The low-level Humanoid control problem is much harder, and moves less quickly (as we saw in Figure 2), so our method does not do as well as it did on Ant. But it is still able to successfully reach the goal much of the time, and again the baselines again fail to learn the maze. To our knowledge, ours is the only work that shows results on a Humanoid maze task.

In Appendix F Table 4, we also show the rates of success of these methods at reaching the random goals. Here it is even more obvious that naïve RL methods fail completely in this task.

In Figure 7a, we compare to Eysenbach et al. (2018) which is similar to our method in the way it composes low-level policies, but generates these using an entropy method. We compare to their results on the Ant Navigation task (see section 5.2 of their paper) in which Ant has to reach four different waypoints and then travel back to the original waypoint. The agent receives reward after reaching each waypoint, so it is another sparse reward task. Because the plot in their paper measures by walltime instead of timesteps, we show their maximum result (DIAYN) as a dotted horizontal line. While their method is only able to achieve an average of about 2.5 waypoints after 15 hours on their hardware, ours can consistently get to all 5 in less time than that, running serially on CPU.

In Figure 7b, we look at the effect of the choice of algorithm on our high-level training. We compare PPO with A2C (Mnih et al., 2016) using the implementation from Kostrikov (2018), and our own implementation of DQN (Mnih et al., 2013; Watkins, 1989). See Appendix C for more implimentation and hyperparameter details. We end DQN early as it took much longer in walltime, but we can see that the choice of algorithm is not critical for successfully training. This suggests that our method is fairly robust to the choice of RL algorithm.

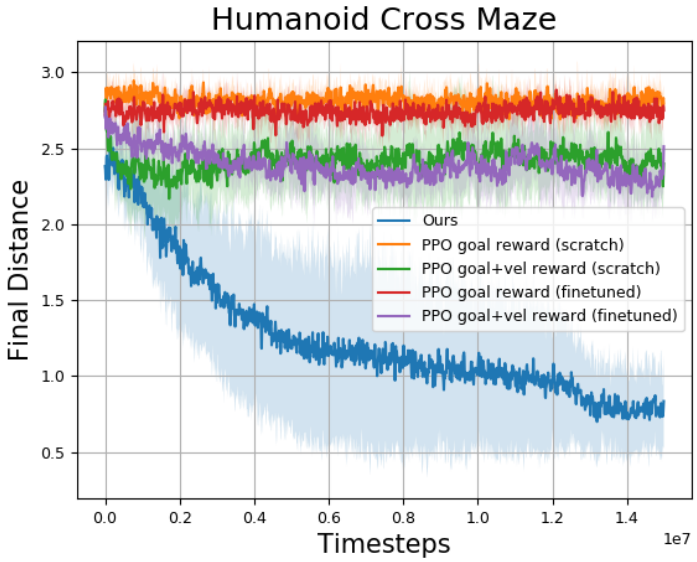
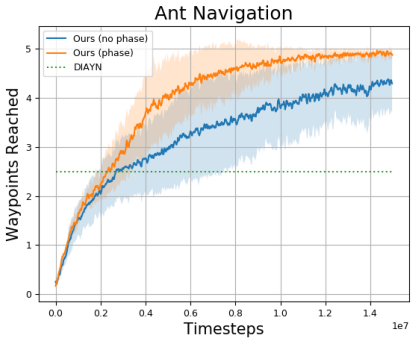
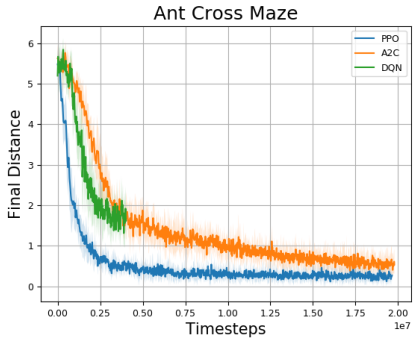


Figure 6: Training curves of our method and baseline on the Random Cross Maze Humanoid environment. This maze is smaller than the Ant version, so all curves start closer to the goal.



(a) Training curves for our method on the Ant Navigation task and comparison to Eysenbach et al. (2018). We show the average number of waypoints reached by each method.



(b) RL algorithm ablation study for our method. We compare the results of our method on Random Ant Cross Maze when training the high-level policy with PPO, A2C and DQN.

Figure 7: Additional experiments and comparisons on Ant

5 Discussion

In this work we present a simple, yet effective way of solving difficult sparse reward RL problems. We first train a set of proprioceptive low-level agents on an intuitive reward and then combine them by training a high-level policy to select the appropriate sequence of low-level policies. With this method we solve difficult Mujoco navigation tasks with sparse rewards that appear impossible to solve using standard RL algorithms.

We show that our method performs well in difficult sparse reward problems in Mujoco locomotion tasks. However, it can be applied to any environment where we have a strong notion of internal and external observation. Our reward function encourages behaviors of the agent that change the robot's state in the environment (i.e. "keep moving") or changes something else external to the agent. This notion exists in environments beyond our Mujoco locomotion problems. Some possible applications for our method are Atari and video game environments with sparse rewards (e.g. Montezuma's Revenge) where to receive rewards the agent must complete a long sequence of actions such as finding a key in one area and bringing it to a lock in another. It could also have potential applications

for robotics such as in navigation and manipulation where there are not dense rewards or there is a high labeling cost to those rewards.

In this work, we do not show any results where we fine-tune the low-level policies during high-level training. This is relatively straightforward, and we have implemented this. We omitted these results as they did not substantially improve the results. For other problems, however, there may be cases where this is useful. For example, if the friction of the ground or some other property of the environment changed between the high-level and low-level environment fine-tuning might be useful.

REFERENCES

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pp. 1726–1734, 2017.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Peter Dayan and Geoffrey E. Hinton. Feudal reinforcement learning. In *Advances in Neural Information Processing Systems 5, [NIPS Conference, Denver, Colorado, USA, November 30 - December 3, 1992]*, pp. 271–278, 1992.

Thomas G. Dietterich. State abstraction in MAXQ hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*, pp. 994–1000, 1999.

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*, 2017.

Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman. Meta learning shared hierarchies. *arXiv preprint arXiv:1710.09767*, 2017.

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

Tuomas Haarnoja, Kristian Hartikainen, Pieter Abbeel, and Sergey Levine. Latent space policies for hierarchical reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pp. 1846–1855, 2018a.

Tuomas Haarnoja, Kristian Hartikainen, Pieter Abbeel, and Sergey Levine. Latent space policies for hierarchical reinforcement learning. *arXiv preprint arXiv:1804.02808*, 2018b.

Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin Riedmiller. Learning an embedding space for transferable robot skills. 2018.

Nicolas Heess, Greg Wayne, Yuval Tassa, Timothy Lillicrap, Martin Riedmiller, and David Silver. Learning and transfer of modulated locomotor controllers. *arXiv preprint arXiv:1610.05182*, 2016.

Daniel Holden, Taku Komura, and Jun Saito. Phase-functioned neural networks for character control. *ACM Transactions on Graphics (TOG)*, 36(4):42, 2017.

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

Ilya Kostrikov. Pytorch implementations of reinforcement learning algorithms. `https://github.com/ikostrikov/pytorch-a2c-ppo-acktr`, 2018.

Eve Marder and Dirk Bucher. Central pattern generators and the control of rhythmic movements. *Current biology*, 11(23):R986–R996, 2001.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

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, 2016.

Shakir Mohamed and Danilo Jimenez Rezende. Variational information maximisation for intrinsically motivated reinforcement learning. In *Advances in neural information processing systems*, pp. 2125–2133, 2015.

Ofir Nachum, Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. *arXiv preprint arXiv:1805.08296*, 2018.

Xue Bin Peng, Glen Berseth, and Michiel Van de Panne. Terrain-adaptive locomotion skills using deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 35(4):81, 2016.

Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel Van De Panne. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 36(4):41, 2017.

Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *arXiv preprint arXiv:1804.02717*, 2018.

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

Arjun Sharma and Kris M Kitani. Phase-parametric policies for reinforcement learning in cyclic environments. In *AAAI Conference on Artificial Intelligence, Pittsburgh, PA*, 2018.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 5026–5033. IEEE, 2012.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pp. 3540–3549, 2017.

Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

A HIGH LEVEL RL FORMULATION DETAILS

Our high-level action space is

$$\Theta = \{\theta_1, \theta_2, \dots \theta_N\}$$
$$\Theta = \{\pi = \pi_1^{ll}(s_t^e, \phi), \pi = \pi_2^{ll}(s_t^e, \phi), \dots \pi = \pi_N^{ll}(s_t^e, \phi)\}$$

As is standard, we add a slowness parameter to the high-level network. That is, the high-level policy makes a choice of which low-level policy to run for the next T time-steps. In our experiments, we choose $T = 10$. So our optimization for our sparse reward task is

$$\underset{\theta}{\text{argmax}} \quad \sum_{i=0}^{T-1} R_{task}(s_{t+i}, \theta) \equiv \quad \underset{j}{\text{argmax}} \quad \sum_{i=0}^{T-1} R_{task}(s_{t+i}, \pi_j^{ll}(s_t^e, \phi))$$

B BASIC ALGORITHM

Algorithm 1 Our method

1: **procedure** TRAIN POLICIES
2: **Train low-level policies**:
3: *for i in N* (or in parallel):
4: initialize(π_i^{ll})
5: train(π_i^{ll}, env_{ll}, T_{ll}, $algo$, $objective_{ll}$)
6: **Train high-level policy**:
7: initialize(π^{hl})
8: train(π^{hl}, env_{hl}, T_{hl}, $algo$, $objective_{hl}$)

In Algorithm 1 above, we give the high-level algorithm we use. We first train each of our N low-level policies π_i^{ll} for T_{ll} steps, using our RL algorithm $algo$, on the low-level objective and environment. We then train our high-level policiy π^{hl} for T_{hl} using RL algorithm $algo$ on the high-level objective and using the high-level environment. Remember that our high-level policy takes our N low-level policies as input and trains to choose which one to run.

C IMPLEMENTATION DETAILS AND HYPERPARAMETERS

For all RL algorithms in the paper (except for DQN in Figure 7b and the values from other works in Figure 3 and Figure 7a), we used the implementation of Kostrikov (2018). For DQN, we wrote our own simple implementation. The hyperparameters for these three algorithms are shown in Tables 1, 2 and 3 We use the ADAM (Kingma & Ba, 2014) optimizer.

Table 1: PPO Hyperparameter values

Parameter	Value
Timesteps per batch	2048
Clip param	0.2
Entropy coeff	0.0
Number of parallel processes	1
Optimizer epochs per iteration	10
Optimizer step size	$3e^{-4}$
Optimizer batch size	32
Discount γ	0.99
GAE λ	0.95
learning rate schedule	constant

Table 2: A2C Hyperparameter values

Parameter	Value
Timesteps per batch	5
entropy coeff	0.01
number of parallel processes	16
Optimizer step size	$7e^{-4}$
Optimizer batch size	32
Discount γ	0.99
GAE	No
learning rate schedule	constant

Table 3: DQN Hyperparameter values

Parameter	Value
Timesteps per batch	2048
Target update frequency	10000
Replay memory size	10^7
Optimizer step size	$3e^{-4}$
Optimizer batch size	128
Discount γ	0.99
learning rate schedule	constant

D LOW-LEVEL TRAINING AND NETWORK DETAILS

During low-level training we train 80 policies using different random seeds. This is to give us 16 low-level policies for 5 variance runs of our high-level policy. The variance plots in Figure 1 use all 80 runs.

For our Ant models, we use a 3-layer MLP with tanh activation functions and a hidden size of 32. For Humanoid we add skip connections between layers and decrease the hidden size to 16.

For all models, we use a vector of size 16 for our phase learned parameters b_ϕ.

We choose the cyclic constraint multipliers for state (λ_s) and action (λ_a) to be 0.05 and 0.01 respectively.

E HIGH-LEVEL TRAINING

For all high-level result plots, we run 5 independent runs with different random seeds for each result. For any model using pre-trained low-level policies, we use different random low-level policies for each run. For our method, we use $N = 16$ low-level policies for our high-level training.

We give the methods the goal index as a one-hot vector to the high-level policies and baselines.

F SUCCESS RATES ON MAZE TASKS

To give another way of looking at the results on the maze tasks, we show below the average success rate on our three random goal maze environments averaged across runs across the last 100 episodes of training.

Table 4: Success Rates on Mazes

Parameter	Ant Cross Maze	Ant Skull Maze	Humanoid Cross Maze
Ours (no phase)	88.6%	86.8%	N/A
Ours (phase)	**92.8**%	**89.4**%	**38.8**%
PPO goal reward (no phase, scratch)	0%	0%	N/A
PPO goal reward (phase, scratch)	0%	0%	0%
PPO goal+vel Reward (no phase, scratch)	0%	0%	N/A
PPO goal+vel Reward (phase, scratch)	0%	0%	0%
PPO goal reward (no phase, finetuned)	0%	0%	N/A
PPO goal reward (phase, finetuned)	0%	0%	0%
PPO goal+vel Reward (no phase, finetuned)	0%	0%	N/A
PPO goal+vel Reward (phase, finetuned)	0%	0%	0%