LEARNING EFFECT-DEPENDENT EMBEDDINGS FOR TEMPORAL ABSTRACTION

William F. Whitney[1] **and Abhinav Gupta**[2]
[1]Department of Computer Science, New York University
[2]Robotics Institute, Carnegie Mellon University
wwhitney@cs.nyu.edu, abhinavg@cs.cmu.edu

ABSTRACT

We propose a technique for learning an embedding of k-step action sequences such that a point in the embedding space maps 1-to-1 with the effect of the corresponding action sequences; randomly sampling an action in this space at time t uniquely corresponds to selecting a state to visit at time $t + k$. We then leverage this embedding by introducing an extension for off-policy reinforcement learning algorithms which allows temporally-abstracted actions without reducing sample efficiency. We demonstrate that our learned action sequence embedding transfers within families of environments and substantially improves the performance of a state-of-the-art model-free reinforcement learning algorithm.

1 INTRODUCTION

In recent years there has been great progress in model-free reinforcement learning for control, both in simulation (Lillicrap et al., 2015; Andrychowicz et al., 2018; Haarnoja et al., 2018a; Fujimoto et al., 2018) and on real hardware (Pinto & Gupta, 2016; Kalashnikov et al., 2018; Haarnoja et al., 2018b). However such algorithms tend to suffer from high sample complexity, which reduces their applicability to general tasks in the real world. An important factor in the poor sample efficiency of reinforcement learning algorithms is their speed of exploration. Whether these algorithms explore via uniform actions (Mnih et al., 2015) or via a correlated random process (Lillicrap et al., 2015), stochastically selecting an action at each timestep corresponds to a (correlated) random walk through state space. While many methods have been proposed to improve exploration with intrinsic motivation (Mohamed & Rezende, 2015; Pathak et al., 2017; Burda et al., 2018), with visitation counts (Bellemare et al., 2016; Ostrovski et al., 2017; Tang et al., 2017), or with counterfactual rewards (Andrychowicz et al., 2017), these methods only indirectly influence the random walk dynamics of the environment by shaping the action distribution of the policy.

In this work, we seek to improve the behavior of reinforcement learning methods by incorporating information about the transition structure of the environment into the action space. We define a representation of action sequences called the Effect-Dependent Embedding (EDE). Instead of taking a random walk at each timestep, this structure allows a policy to explore by sampling a state to visit k timesteps in the future. These improvements are orthogonal to the supplementary rewards defined by e.g. Andrychowicz et al. (2017); Pathak et al. (2017) and can be combined with those methods without modification.

Our contribution has three components:

1. We propose a technique for learning an embedding space of action sequences based on the effect each sequence has on the environment.
2. We define a simple deterministic objective to approximately invert the many-to-one mapping of action sequences to embeddings.
3. We extend the Deterministic Policy Gradient (DPG) (Silver et al., 2014) family of algorithms to enable sample-efficient training in a temporally abstracted action space.

We demonstrate the effectiveness of our method by training the Twin Delayed Deep Deterministic policy gradient algorithm (TD3) (Fujimoto et al., 2018) in the EDE action space to form an algorithm

we call EDE-TD3. Furthermore, we show that the action representation learned by EDE transfers within families of environments as long as the basic environment dynamics (e.g. robot kinematics) remain the same. This allows us to train EDE once and use it to train policies for several different tasks, even when those tasks have different observation spaces, rewards, or objects to interact with. We find that EDE-TD3 delivers substantial gains to both sample efficiency and asymptotic performance compared to TD3 in the original action space, and this remains true when transferring the EDE space from another environment.

2 LEARNED ACTION SEQUENCE REPRESENTATION

In this section we describe a novel method for learning an abstract representation of action sequences. We first propose a new formalism for thinking about actions and sequences of actions in a Markov decision process (MDP). We then describe a method for learning an abstract space called Effect-Dependent Embedding (EDE) space, in which each action sequence is represented by the effects it has on the state. Finally, we propose a simple objective to train the inverse mapping from EDE space back into sequences of raw actions.

For space reasons we omit the usual definitions of the Markov decision process and its objectives; instead we will use the conventions described in Silver et al. (2014) except as otherwise noted.

2.1 REPRESENTING ACTIONS BY THEIR EFFECTS

Let $\mathcal{T}$ be the transition function of the environment such that $\mathcal{T}(s_t, a_t) = s_{t+1}$ is the next state after taking action a_t in state s_t. Likewise let $\mathcal{T}(s_t, a_{t:t+k-1}) = s_{t+k}$ be the result of starting in state s_t and taking a sequence of k actions $a_{t:t+k-1}$. We define two sequences of actions $a_{1:k} \equiv \bar{a}_{1:j}$ as equivalent if, $\forall s \in \mathcal{S}$, $\mathcal{T}(s, a_{1:k}) = \mathcal{T}(s, \bar{a}_{1:j})$. We then say that $a_{1:k}$ and $\bar{a}_{1:j}$ have the same "effect". Let $Eq(a_{1:k}) = \{\bar{a}_{1:j} : a_{1:k} \equiv \bar{a}_{1:j}\}$ be a set of equivalent action sequences. We denote by $\mathcal{A}$ the set of all such equivalence classes; $\mathcal{A}$ is thus the set of effects that action sequences can have on the environment.

We now consider the challenge of representing $\mathcal{A}$. For simplicity, let us consider a subset $\mathcal{A}_k \subseteq \mathcal{A}$ which is the set of effects realizeable in exactly k actions. For $k = 1$, the raw actions themselves form a reasonable representation of $\mathcal{A}_k$. Assuming no two actions a^i and a^j in the environment are exactly identical across all states, the raw actions are a minimal representation of $\mathcal{A}_1$ such that $a^i \equiv a^j$ if and only if $a^i = a^j$. However, for $k > 1$, representing some $\boldsymbol{A}_k \in \mathcal{A}_k$ by a sequence of actions $\{a_1, ..., a_k\}$ is redundant as many sequences may correspond to the same effect, e.g. moving back and forth several times is equivalent to not moving at all.

In the next section we propose a method for learning a minimal representation of action sequences for which each point corresponds to a unique effect. We term the space of these unique representations Effect-Dependent Embedding (EDE) space.

2.2 LEARNING THE EFFECT-DEPENDENT EMBEDDING

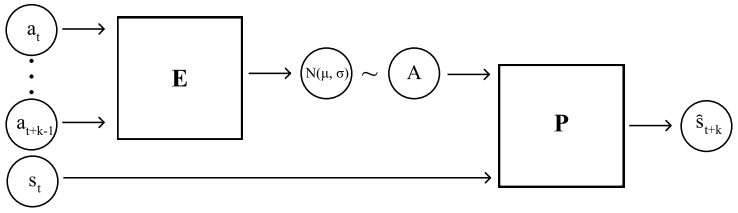


Figure 1: The training procedure for the EDE encoder **E**. The encoder is trained to minimize the information content of the learned embedding $\boldsymbol{A}$ while still allowing the predictor **P** to make accurate predictions.

We propose that one objective for learning a k-step EDE space $\mathcal{A}_k$ is to find the minimum description length (MDL) (Rissanen, 1978) encoding $\boldsymbol{A}_k$ for each action sequence of length k such that $p(s_{t+k}|s_t, \boldsymbol{A}_k) = p(s_{t+k}|s_t, a_t, ..., a_{t+k-1})$. That is, the encoding $\boldsymbol{A}_k$ of an action sequence

$\{a_1, ..., a_k\}$ should preserve *all* information about the final effects of this action sequence, and *no* information about what happens along the way.[1]

In practice, we approximate this objective by information-regularizing an action representation while learning a predictive model of the environment. This model, shown in Figure 1, takes a form similar to a conditional variational autoencoder (Kingma & Welling, 2013; Rezende et al., 2014) with an unconditional inference network. However, we find it more instructive to simply interpret it as a regularized forward model. The loss function used to train this forward model is a combination between the mean-squared error of the model's predictions and the KL divergence from a prior to the action encoding:

$$\mathcal{L}(\theta_{\mathbf{E}}, \theta_{\mathbf{P}}) = \mathbb{E}_{\boldsymbol{A} \sim \mathbf{E}} \Big[||\mathbf{P}\big(s_t, \boldsymbol{A}; \theta_{\mathbf{P}}\big) - s_{t+k}||_2^2 \Big] + \beta D_{KL}\big(\mathbf{E}(a_{t:t+k-1}; \theta_{\mathbf{E}}) \,||\, \mathcal{N}(0, \boldsymbol{I})\big)$$

Here $\mathbf{E}$ and $\mathbf{P}$ are the encoder and the predictor respectively, with corresponding parameters $\theta_{\mathbf{E}}$ and $\theta_{\mathbf{P}}$. In this work $\mathbf{E}$ and $\mathbf{P}$ are both realized as neural networks and the state and action observations are sampled from the environment according to a random policy. A visualization of the space learned by this objective is available in Appendix B.

2.3 Decoding to raw actions

In order to be useful for RL, the abstract action space produced by the encoder must be decodeable to raw actions in the environment. Since the mapping from action sequences to high-level actions is many-to-one, inverting it is nontrivial. We simplify this ill-posed problem by defining an objective with a single optimum.

Once the action encoder $\mathbf{E}$ is fully trained, we hold it fixed and train the action decoder to minimize

$$\mathcal{L}(\theta_{\mathbf{D}}) = \mathbb{E}_{\boldsymbol{A} \sim \mathcal{N}(0, \boldsymbol{I})} \Big[||\mathbf{E}(\mathbf{D}(\boldsymbol{A}; \theta_{\mathbf{D}})) - \boldsymbol{A}||_2^2 + \lambda ||\mathbf{D}(\boldsymbol{A}; \theta_{\mathbf{D}})||_2^2 \Big]$$

The first term of this objective ensures that the action decoder $\mathbf{D}$ is a one-sided inverse of $\mathbf{E}$; that is, $\mathbf{E}(\mathbf{D}(\boldsymbol{A})) = \boldsymbol{A}$ but $\mathbf{D}(\mathbf{E}(a_1, ..., a_k)) \neq a_1, ..., a_k$. The second term of the loss ensures that $\mathbf{D}$ is in particular the minimum-norm one-sided inverse of $\mathbf{E}$ and gives the objective for the output of $\mathbf{D}$ a single minimum. We choose λ to be small (e.g. 10^{-4}) to ensure that the reconstruction criterion dominates the optimization.

3 Efficient high-level RL

Once equipped with a decoder which maps from high-level actions to sequences of low-level actions, we wish to train a high-level policy that can solve a task by selecting high-level actions. In this section we extend the Deterministic Policy Gradient (Silver et al., 2014) family of algorithms to work with temporally-extended actions while maintaining their off-policy updates and learning from every environment step. This allows our method to achieve superior sample efficiency when working with high-level actions. In particular, we extend Twin Delayed Deterministic policy gradient (TD3) (Fujimoto et al., 2018) to work with the EDE representation of actions to form an algorithm we call EDE-TD3.

We first describe why DPG requires modification to accommodate temporally-abstracted actions. One simple approach to combining EDE with DPG would be to incorporate the k-step EDE action space into the environment to form a new MDP. This MDP allows the use of DPG without modification; however, it only emits observations once every k timesteps. As a result, after N steps in the original environment, the deterministic policy μ and critic function Q can only be trained on N/k observations. This has a substantial impact on sample efficiency when measured in the original environment.

[1] Using this objective as the action space for RL encodes an assumption about the environment: nothing that happens along a trajectory is relevant except for the final state. We say that such an environment satisfies the Machiavellian assumption ("the ends justify the means"). However, we evaluate our method on environments which violate this assumption to some degree and find EDE to be quite robust.

Instead we require an algorithm which can perform updates to μ and Q for every environment step. To do this, we train both μ and Q in the abstract action space with minor changes to their updates. We distinguish these policy and critic functions whose actions are in EDE space from their low-level equivalents by adding a superscript EDE, i.e. μ^{EDE} and Q^{EDE}. We augment the critic function with an additional input, i, which represents the number of steps $0 \leq i < k$ of the current embedded action e that have already been executed. This forms the EDE critic:

$$Q^{\text{EDE}}(s_t, e_t, i) = \sum_{j=0}^{k-i-1} \left(\gamma^j r_{t+j}\right) + \gamma^{k-i} Q^{\text{EDE}}\Big(s_{t+k-i}, \mu^{\text{EDE}}(s_{t+k-i}), 0\Big) \tag{1}$$

In plain language, the value of being on step i of abstract action e_t is the value of finishing the remaining $(k-i)$ steps of e_t and then continuing on following the policy. The EDE critic is trained by minimizing the Bellman error implied by Equation 1.

To update the policy, we follow the standard DPG technique of using the gradient of the critic. The only modification needed is to note that at the time of a new high-level action, i is always zero. The gradient of the return with respect to the policy parameters, given that data was collected according to a behavior policy β, is then

$$\nabla_\theta J_\beta(\mu_\theta^{\text{EDE}}) \approx \mathbb{E}_{s\sim\rho^\beta}\Big[\nabla_\theta \mu_\theta^{\text{EDE}}(s)\ \nabla_e Q^{\text{EDE}}(s, e, 0)|_{e=\mu_\theta^{\text{EDE}}(s)}\Big] \tag{2}$$

4 Related work

Similarly to this work, hierarchical reinforcement learning seeks to learn temporal abstractions. These abstractions are variously defined as skills (Florensa et al., 2017; Hausman et al., 2018), options (Sutton et al., 1999; Bacon et al., 2017), or goal-directed sub-policies (Kulkarni et al., 2016; Vezhnevets et al., 2017). Whereas these works train a low-level policy by maximizing the reward of the overall task or a heuristically-defined subtask, in this work we seek to learn a representation of the transition structure of the environment.

Most closely related are Co-Reyes et al. (2018) and Nachum et al. (2018). SeCTAR (Co-Reyes et al., 2018) simultaneously learns a generative model of future states and a low-level policy which can reach those states. Unlike this work, their latent space represents a particular trajectory through the environment rather than an effect, making it state-dependent. This limits its transferability between environments. HIRO (Nachum et al., 2018) addresses the aim of off-policy training of hierarchical policies. However, their off-policy performance depends on an approximate re-labeling of action sequences to train the high-level policy, and their low-level policy must be trained on an observation space which matches the target task.

Also related are methods which attempt to learn embeddings of single actions to enable efficient learning in very large action spaces (Dulac-Arnold et al., 2015; Chandak et al., 2019). These works similarly learn a latent space of actions, however their latent spaces are for a single action and their objectives are based on reconstruction of the action distribution, not the effects of an action on the environment.

5 Experiments

In the following experiments, we evaluate the effectiveness of the EDE representation as an action space for deep RL. We particularly wish to answer the following questions:

1. Can the EDE action space represent the optimal policy for continuous control tasks?
2. Does model-free RL in EDE space use fewer samples from the environment?
3. Does the EDE representation transfer between different tasks and environments in the same family?

To simultaneously evaluate all three questions, we use two different "families" of related tasks. Within each of these families, the robot which the agent controls maintains the same dynamics,

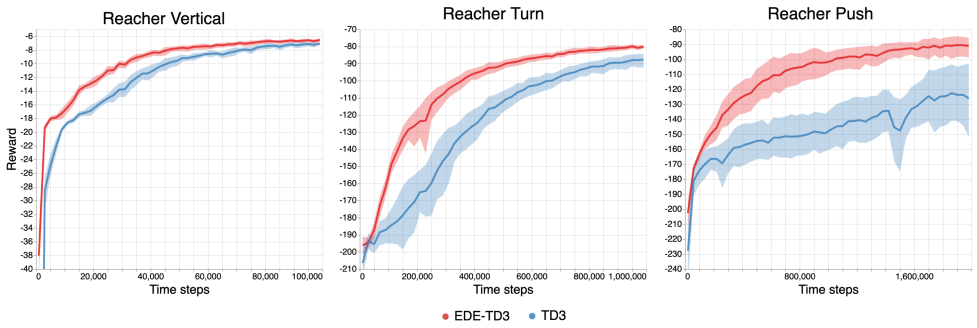


Figure 2: Performance of EDE-TD3 and TD3 on the Reacher family of environments. Dark lines are mean reward over 8 seeds and shaded areas are bootstrapped confidence intervals. Across all the environments, EDE-TD3 outperforms TD3 in sample efficiency, and matches or beats it in asymptotic performance. The EDE action space used across all three tasks was trained only on `ReacherVertical`, demonstrating that EDE representations are highly transferable.

implying that the transition information encoded in the EDE action representation is still approximately valid. However, each task in the family may have different objects to interact with and even a different observation space. Whereas directly transferring a policy to an environment with different observations would be impossible, EDE requires only that the action space remain the same.

The "Reacher" family of tasks involves controlling a 2D, 2DoF arm to interact with various objects. The "7DoF" family of tasks features a 3D, 7DoF arm which must use different end effectors to move various objects to their goal positions. All hyperparameters are the same for EDE-TD3 and for TD3, and are constant across all experiments. Images and detailed descriptions of both families of tasks, along with hyperparameter settings, are available in Appendix A.

5.1 Results

Figure 2 shows the results of EDE-TD3 and TD3 applied to the Reacher family of tasks, and Figure 3 shows results on the 7DoF family. These results demonstrate that (1) EDE action spaces can represent policies just as good as as the raw action space, at least up to the ability of RL to find them; (2) model-free RL shows substantial efficiency gains from using the EDE action space; and (3) results 1 and 2 continue to hold even when transferring the EDE space between environments. It is especially worth noting that the gains from EDE-TD3 increase as the tasks get harder, maintaining convergence, stability, and low variance in the face of high-dimensional control tasks with difficult exploration.

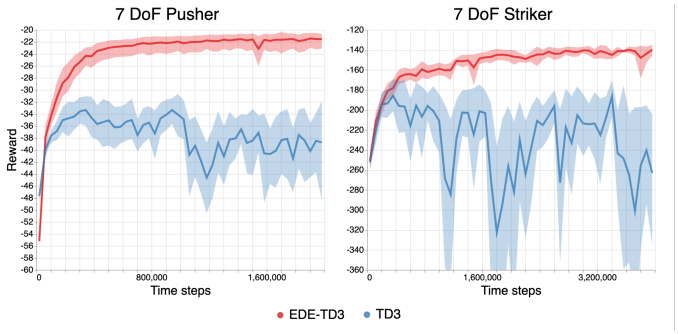


Figure 3: EDE-TD3 and TD3 on the 7DoF family of environments. Dark lines are mean reward over 8 seeds and shaded areas are bootstrapped confidence intervals. Whereas TD3 in the regular action space does not manage to solve either task and even diverges, EDE-TD3 demonstrates stable, low-variance convergence to a high-quality policy. The EDE action space used for these two tasks was trained only on the 7DoF Pusher environment.

REFERENCES

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pp. 5048–5058, 2017.

Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*, 2018.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pp. 1471–1479, 2016.

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

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

Yash Chandak, Georgios Theocharous, James Kostas, Scott Jordan, and Philip S Thomas. Learning action representations for reinforcement learning. *arXiv preprint arXiv:1902.00183*, 2019.

John D. Co-Reyes, Yuxuan Liu, Abhishek Gupta, Benjamin Eysenbach, Pieter Abbeel, and Sergey Levine. Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings. In *ICML*, 2018.

Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*, 2015.

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

Scott Fujimoto, Herke van Hoof, and Dave Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018a.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018b.

Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin Riedmiller. Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations*, 2018.

Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, pp. 3675–3683, 2016.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

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, 2015.

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, Shixiang Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *NeurIPS*, 2018.

Georg Ostrovski, Marc G Bellemare, Aäron van den Oord, and Rémi Munos. Count-based exploration with neural density models. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2721–2730. JMLR. org, 2017.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 16–17, 2017.

Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3406–3413. IEEE, 2016.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

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.

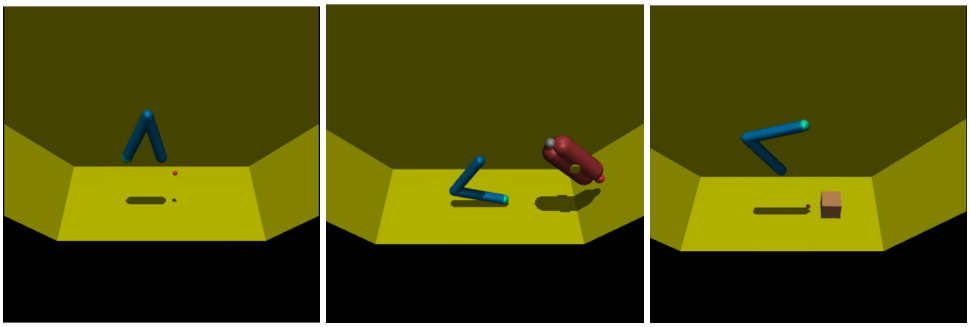
Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in neural information processing systems*, pp. 2753–2762, 2017.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 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-Volume 70*, pp. 3540–3549. JMLR. org, 2017.

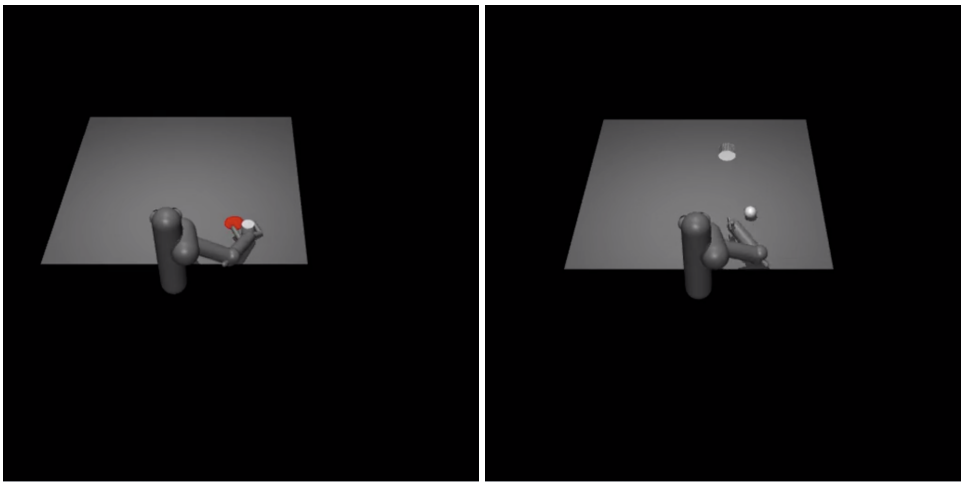
APPENDIX A EXPERIMENT DETAILS



(a) `ReacherVertical` (b) `ReacherTurn` (c) `ReacherPush`

Figure 4: The Reacher family of environments. `ReacherVertical` requires the agent to move the tip of the arm to the red dot. `ReacherTurn` requires the agent to turn a rotating spinner (dark red) so that the tip of the spinner (gray) is close to the target point (red). `ReacherPush` requires the agent to push the brown box onto the red target point. The initial state of the simulator and the target point are randomized for each episode. In each environment the rewards are dense and there is a penalty on the norm of the actions. The robot's kinematics are the same in each environment but the observation spaces are different.

The first task family, pictured in Figure 4, is the "Reacher family", based on the `Reacher-v2` MuJoCo (Todorov et al., 2012) task from OpenAI Gym (Brockman et al., 2016). These tasks form a simple new benchmark for multitask robot learning. The first task, which we use as the "source" task for training the EDE space, is `ReacherVertical`, a standard reach to a location task. The other two tasks are inspired by the DeepMind Control Suite's `Finger Turn` and `Stacker` environments, respectively (Tassa et al., 2018). In `ReacherTurn`, the same 2-link Reacher robot must turn a spinner to the specified random location. In `ReacherPush`, the Reacher must push a block to the correct random location.



(a) `Pusher-v2` (b) `Striker-v2`

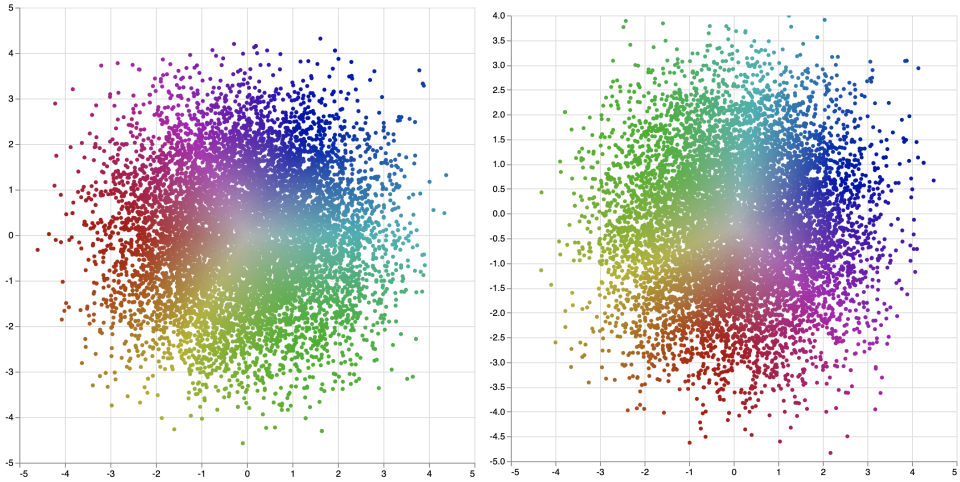
Figure 5: The 7DoF family of environments. `Pusher-v2` requires the agent to use a C-shaped end effector to push a puck across the table onto a red circle. `Striker-v2` requires the agent to use a flat end effector to hit a ball so that it rolls across the table and reaches the goal. As with the Reacher family, the dynamics of the robot are the same within the 7DoF family of tasks. However, the morphology of the robot, as well as the object it interacts with, is different.

The second task family is the "7DoF family", which comprises `Pusher-v2` (the source task) and `Striker-v2` (the target task) from OpenAI Gym (Brockman et al., 2016). These tasks use similar

(though not identical) robot models, making them a feasible family of tasks for transfer. They are shown in Figure 5.

We compare EDE-TD3 versus TD3 on these two families of tasks. In each family, the EDE representation is trained on a dataset of 100K steps of a random policy in the source task. As this EDE pretraining is unsupervised (i.e. does not use rewards) and only occurs once for each family of environments, the x axis on our training curves refers only to the samples used to train the policy. For both EDE-TD3 and regular TD3 we use all of the default hyperparameters from the authors' code[2] across all tasks. Throughout these experiments we set the loss scales $\beta = \lambda = 10^{-4}$, the number of actions in the EDE space $k = 4$, and we choose the dimension of the EDE space to be equal to the dimension of a single action in the environment.

APPENDIX B VISUALIZING THE EDE SPACE



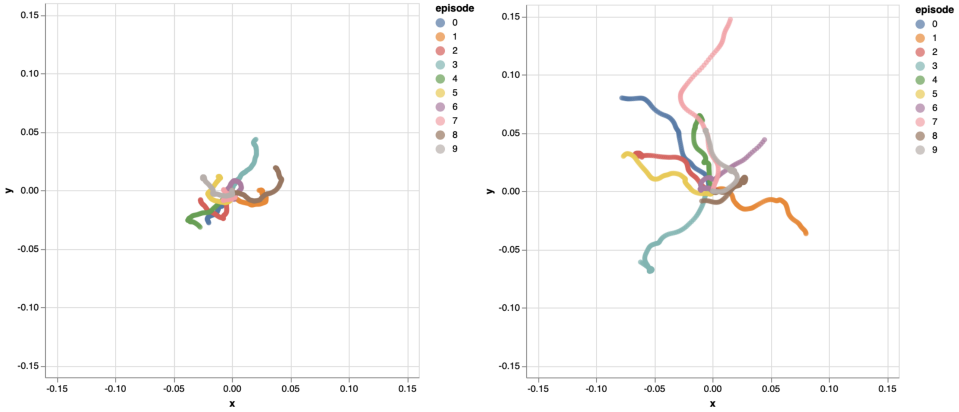
(a) Effect space (b) EDE space

Figure 6: These plots represent the mapping between the effects of action sequences and points in EDE space. In the left plot, each point represents the effect $s_{t+4} - s_t$ of a sequence of four randomly-selected actions $\{a_1, ..., a_4\}$, colored according to their location in the plot. That is, the e.g. x coordinate of a point represents the Δx between the initial state and the state reached at the end of the four actions. Each point in the right plot corresponds to the point in the left plot with the same color, and its coordinates in the right axes are given by $\mathbf{E}(a_1, ..., a_4)$. The 1:1 correspondence between the effects of a sequence of actions and the EDE space representation of that action sequence indicates that EDE is truly encoding the change in state induced by a sequence of actions.

To better understand whether the latent space of action sequences learned by EDE corresponds to the effects of those action sequences, we use a simple linear Point environment with a 2D state space and 2D action space. We render action sequences on a pair of plots in Figure 6. On the left plot we show the *effect* of the action sequence, measured as $s_{t+4} - s_t$. Each action sequence is assigned a color based on its location in this plot. On the right plot, we show each action sequence's location in EDE space, marking the point with that action sequence's color. If the spaces are isomorphic, we should expect to see smooth color transitions in the right plot, indicating that the EDE representations of any pair of action sequences are ordered according to their effects. Indeed, for this simple problem, we see that the EDE space is an affine transformation of the effect space. The correspondence between the two spaces appears to remain strong for high-dimensional and nonlinear environments, but is much harder to render in two dimensions.

[2] https://github.com/sfujim/TD3

APPENDIX C EXPLORATION WITH RAW AND EDE ACTION SPACES



(a) Random exploration without EDE

(b) Random exploration with EDE

Figure 7: These figures illustrate the way the EDE action space enables more efficient exploration. Each figure is generated by running a uniform random policy for ten episodes on a `PointMass` environment. Since the environment has only two position dimensions, we can plot the actual 2D position of the mass over the course of each episode. **Left:** A policy which selects actions at each environment timestep uniformly at random explores a very small region of the state space. **Right:** A policy which randomly selects EDE actions once every k timesteps explores much more widely.