Compositional Skill Learning Using Batch Meta RL

Jonathan Booher
Department of Computer Science
Stanford University
jaustinb@stanford.edu

Albert Tung
Department of Computer Science
Stanford University
atung3@stanford.edu

Abstract

Even the most state-of-the-art reinforcement learning algorithms struggle to effectively solve long horizon and dexterous manipulation tasks in the real world. The lack of sample efficiency in these algorithms, and inherent danger in real world policy rollouts has served as a barrier to success. We propose that by learning shorter sub-tasks as well as a method for sub-task composition, that more complex tasks can be efficiently solved. Furthermore with the recent successes in data driven robotics and learning from demonstrations, we show that our proposed method is capable of effectively learning from expert demonstrations and learn to solve various different sub-tasks, as well as their compositions without additional interactions with the environment. Additionally, we show that this method admits one-shot imitation of expert trajectories representing compositions of sub-tasks and provides useful exploration behaviors for further reinforcement learning showing that this method has potential as a pre-training mechanism for policy learning.

1 Introduction

Many useful tasks in robotics are long-horizon, staged, do not admit easy reward functions, and require dexterous manipulation [3, 21]. All of these hallmarks of useful robotics tasks make traditional reinforcement learning, even with the most state-of-the-art methods in the real world, extremely difficult. Furthermore, current deep reinforcement learning algorithms such as Deep Q-Networks (DQN) and Deep Deterministic Policy Gradients (DDPG) require large amounts of environment interactions to obtain good performance [9, 19, 11]. The data inefficiency of many of these algorithms makes it difficult for training new tasks and, as a result, learning new policies efficiently.

One possible way to address real-world robot data issues is to collect the entirety of the data in simulation. For this reason, many approaches that seek to solve difficult robotic manipulation tasks train policies in simulation and then address the sim-to-real gaps in perception and dynamics [7, 13, 1]. These approaches are undesirable in the sense that they require two stages, policy learning and then transfer. In addition, these methods also significantly more training time in simulation, often needing domain randomization approaches.

We seek to show that through learning many short time horizon tasks, and then composing the resulting policies that the problem of learning to solve the long-horizon tasks in the target domain is made simpler. But even these sub-tasks can exhibit sufficient variation and require dexterous manipulation which might be difficult to solve using traditional approaches from reinforcement learning due to the exploration burden. One way to address this is to use demonstrations from an expert, the most natural of which is human demonstrations. Such demonstrations can reduce the complexities of exploration and also provide methods for learning without interactions. We present an algorithm that can efficiently learn to accomplish many different sub-tasks simultaneously from expert demonstrations while simultaneously learning methods for sub-task composition. In short, our contributions are

Preprint. Under review.

1. Efficient learning from limited tasks instances without environment interactions
2. A simple method for composing sub-tasks using latent variables
3. Show the usefulness of latent variables for one-shot imitation learning
4. Illustrate the link between latent variables and efficient, learned exploration policies

2 Related Works

2.1 Reinforcement Learning

A promising approach in reinforcement learning that has direct application to learning from demonstrations is batch reinforcement learning [8] which separates data collection and optimizes over the fixed set of experiences. Batch algorithms are therefore more data-efficient and stable, but suffer from a lack of exploration-exploitation. Some other explorations have involved using batch-constrained Q-learning (BCQ) which utilizes a generative and perturbation model to improve results [6].

2.2 Knowledge Distillation

Knowledge distillation takes larger networks and performs model compression by teaching smaller networks to learn to mimic the larger, often being used to create compressed and more efficient models [10]. Distillation is also being utilized to compress policies for reinforcement learning. The method can thus be used to consolidate task-specific policies into a single policy, therefore outperforming the single-task teachers and is capable of generalizing to new tasks with no prior expert guidance [14, 15].

2.3 Meta Learning

Current means of meta learning involve learning an action-value neural network that learns to criticize any actor trying to solve a specific task, which helps with few-shot and semi-supervised learning tasks. In doing so, the meta-critic teaches task-specific actors, with the ultimate goal of being able to train on new unseen tasks with relatively fewer examples and gradient update steps. Current papers have shown that training algorithms that are model-agnostic can improve convergence on few-shot learning and policy gradient reinforcement learning [4, 16]. Other meta-learning algorithms have explored fusing hierarchical learning and meta-learning to no longer necessitate repeating meta-training over heterogeneous tasks by factorizing similar training tasks into heterogeneous tasks and then training them in a hierarchical manner to maximize adaptation and generalization [24].

2.4 Batch-Meta RL

Many different supervised learning algorithms benefit from pre-training [22, 2] which allows algorithms to use large-scale processes and effectively give the algorithm a better initialization to more quickly learn on smaller datasets. Batch meta reinforcement learning proposes using Batch RL algorithms to extract MDP-specific networks and then interpolating knowledge about seen MDPs to perform well on unseen MDPs by distilling value functions, all of which can support generalization. Such initialization has shown that this approach is competitive with state-of-the-art model-free RL methods trained with hundreds of thousands of interactions [19].

3 Tasks and Environments

We examine several different dexterous manipulation behaviors with a simulated sawyer robot arm and a single cube from robosuite [3]. These include: reaching, lifting, and pushing in all four directions. For examples of the environments, see Fig. 1. We treat each of these tasks as distinct resulting in 6 different tasks. All of these environments contain goals. For reaching, the goal is the location to reach to, for lifting and pushing, the goal is the location of the cube. Observations in each of these tasks are the same:

$$o_t = [\sin(joint\_angles), \cos(joint\_angles), gripper\_state, joint\_velocity, eef\_pose, goal]$$

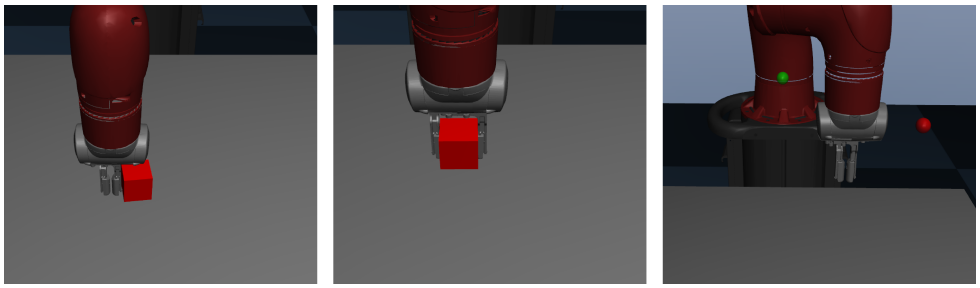


Figure 1: The environments used in this work. From left to right: the pushing task, the lifting task, the reaching task. In the pushing task, the robot begins above the cube and needs to move to a side of the cube depending on the direction to push, and then push the cube. In the lifting task, the robot again begins above the cube and needs to reach downward, grasp, and lift the cube. In the reaching task, the robot begins with its end effector at the green dot and needs to move to the randomly placed red dot.

which gives a good low dimensional state representation for policy learning. Note that the $\sin$ and $\cos$ help to address the discrepancy between 0 and 2π as they should be considered the same angle. With this observation space, all the policies will be goal conditioned. In the reaching task, the goals are the location for the end effector to reach; for the lifting and pushing tasks, the goal is the location of the cube. Additionally, the start state distribution for the reaching task is fixed around a set point, whereas the starting state distribution for the lifting and pushing tasks is above a randomly placed cube. This is to encourage the learning of distinct *skills* from each of these environments.
Each of the tasks has a separate dense reward function which are easily written down

$$L_{reach} = 1 - tanh(10 * ||pos - goal||_2^2), \quad L_{lift} = L_{reach} + 1\{grasped\} + 1\{height > 0.05\}$$

$$L_{push} = L_{reach} + 1 - tanh(||velocity - target\_direction||) + 1\{distance\_from\_start > 0.07\}$$

where the reward functions are a mix of dense and sparse signals that represent progress throughout the stages of each of the tasks. These reward functions help the learning process by providing a meaningful signal to the learning algorithm while still being possible to extend to the real world through instrumentation like motion capture systems. The environments all have a horizon of 200 time steps, corresponding to approximately 10 seconds of execution time. The action space for all the environments is delta transforms at the end effector and will be discussed further in the next section.

4 Data

We collect 500 expert demonstrations for the reach and lift tasks, and 100 demonstrations from the push tasks. The expert policies in this case are hard-coded controllers that leverage privileged information to accomplish the task in an optimal manner. We add noise to the actions at each timestep to ensure that the reinforcement learning algorithm can learn corrective behaviors addressing the issues of accumulating errors typically seen in methods like behavioral cloning. These noisy demonstrations also serve to emulate the quality of demonstrations expected from human teleoperation.
The action space is delta end effector poses combined with an operational space control scheme that allows for smooth control and inter-operability with human demonstrations. This action space allows for use with real robot teleoperation systems like RoboTurk for use on real robots. In particular, operational space control is used because it is a robust and precise controller that works on the end effector poses and produces actions at the low level based on joint velocities.

5 Preliminaries

Here we cover some of the necessary background for the motivation and implementation of our proposed method.

5.1 The Importance of Demonstrations

Prior works have shown that demonstrations can significantly reduce the burden of exploration in training reinforcement learning agents. Methods for leveraging these demonstrations have ranged from behavioral cloning [17], to demonstration state resets [12] and inverse reinforcement learning [20, 23, 5]. These methods are able to ingest demonstrations by either directly learning a policy on the demo transitions, using demonstrations to explicitly narrow the exploration problem, or learning a reward function from the data. While these methods have shown promise, BC is brittle and cannot handle large task variations, demo state resets can only ingest small amounts of data, and inverse RL is computationally expensive.

Learning from demonstrations in a robotics setting provides multiple different signals that afford it advantages over pure reinforcement learning. First, human demonstrations provide diverse and dexterous behaviors and reasoning that are unlikely to be discovered through random exploration; second, they provide high quality information over random exploration that allows agents to learn significantly quicker. With recent works like RoboTurk, human demonstrations are increasingly available and thus it is useful to examine approaches that can effectively use large amounts of demonstration information to learn robust policies.

5.2 Batch Reinforcement Learning

In real-world robotics, with access to high-quality demonstrations, it is useful to approach the reinforcement learning problem in a batch setting. This setting is one in which the agent is not allowed interactions with the environment and must instead learn to accomplish the task based solely on the demonstrations provided. In a real world setting this is useful as the issue of robot and human safety as well as execution time (learning bottlenecked by human environment resets) common in real-robot RL are both limited. In particular we take the approach of Batch Constrained Q-Learning (BCQ).

In Batch Constrained Q-Learning, it was observed that training a DDPG agent on the replay buffer of another DDPG agent (a batch of data) was unable to recover the performance of the original agent due to extrapolation error in actions that were not present in the batch. The Batch Constrained Q-Learning algorithm does not run into this failure mode as it learns an action proposal network G that is a variational autoencoder that encodes states and reconstructs actions. These produced actions are trained to reproduce the actions from the batch. Similarly, the Q function is trained based directly on the s and a in the batch rather than from a policy to regress the value from the batch.

$$Q(s,a) \leftarrow (1-\alpha_t)Q(s,a) + \alpha_t(r + \gamma Q(s', \pi(s'))) \tag{1}$$

$$\pi(s') = \arg\max_{a_m + \epsilon(s', a_m)} Q(s, a_m + \epsilon(s', a_m)) \quad \{a_m = G(s', z_m)\}_m \quad z_m \; N(0,1) \tag{2}$$

The updates for the learned Q-function is BCQ follow the same soft update pattern as in DDPG. Because it uses only actions from the batch to train the Q and G functions, the BCQ algorithm does not need to rely on generalization performance during training in order to produce accurate predictions of the Q values and actions, eliminating the possibility for extrapolation error. A policy can then be rolled out by sampling N actions from G and then choosing the action that produces the highest Q value.

5.3 Knowledge Distillation

Given a network F that takes as input o and outputs a, we can train another network G to produce the same output with the loss

$$L_{distill} = d(G(o) - F(o))$$

where d is a valid distance metric that can be task specific, in our case, euclidean distance. In this way, the network G, typically of lower capacity, can learn to output the same values as F. Note that the gradients only pass through G.

5.4 Meta Reinforcement Learning

Meta Reinforcement Learning can be posed as learning over distributions of MDPs. An goal based MDP can be defined as $(S, A, T, \gamma, r, G, I)$ where S is a set of the states, A is a set of actions possible in each state, T is the set of transition probabilities for each state action pair, γ is a discount

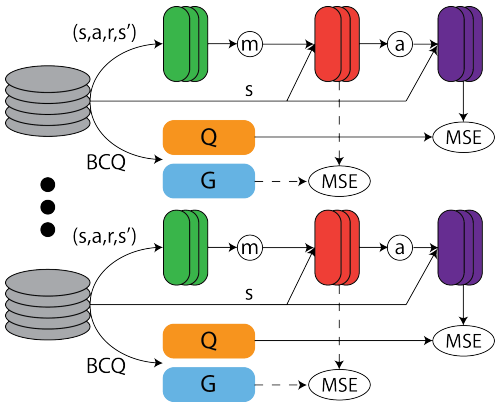


Figure 2: The proposed distillation step (stage 2). Given batches, and pretrained G and Q for each of the tasks, we can learn a meta-Q and meta-G function such that it can accomplish all the tasks based on a single latent code m predictive of the task to be executed. Red depicts the shared weights of the meta-G function and purple is the shared weights of the meta-Q function. We train the distillation on the same batches as the original BCQ policies were trained on, and sample batches from random tasks at each step.

factor, r is a reward function, G is a set of goal states, and I is the set of initial conditions. We can define a distribution over these tuples P from which we can draw meta-train MDPs, as well as meta-test MDPs. In our case, the meta-train MDPs are the different sub-tasks and the meta-test MDPs are compositions of the sub tasks. Some approaches in meta-RL focus on learning a good policy initialization for efficiently learning to solve each of the MDPs. Other approaches, like ours, use additional task specific information to learn a meta policy that can, given the correct task specific information, accomplish each of the tasks. In this way, this approach is similar to that of contextual policies, but is still in the realm of meta-learning as we learn the context based on one or potentially many transitions or experiences, thus linking to RNN based meta-RL approaches like RL^2.

6 Method

Our proposed method is similar to that of [18], but there are several key distinctions. First, the prior work introduced many inductive biases including a state encoder, reward predictor, and a dynamics model which pose challenges for extensions to the real world with no reward functio as well as image observations as image based dynamics models are often difficult to train in practice. Secondly, prior work utilizes a discrete representation for the MDPs limiting the expressivity of the MDP identification. We seek to relax these assumptions to admit easier extension to image state as well as allow the possibility of MDP interpolation based on the learned embeddings leading to novel exploration strategies. Importantly, our method is related to hierarchical RL in the sense that the continuous MDP embedding we propose could be the output of a higher level reasoning module, but that is left for future work. The algorithm is summarized in Fig. 2 Our method is composed of two distinct stages

6.1 Stage 1

Given n batches of data $B_1, B_2, ...B_n$ corresponding to batches for each of n tasks, we learn two functions G_i and Q_i for each of the tasks through Batch Constrained Q-Learning. We train each of these networks to convergence on the batches provided and freeze their weights. This stage results in n different policies and Q functions that together can solve each of the short time-horizon tasks.

6.2 Stage 2

We train a network M which takes as input $o = (s_i, a_i, r_i, s'_i)$ from the ith batch and produces a continuous value m which is predictive of the MDP. Specifically, the embedding network M is

trained with the following loss function

$$L_{mdp\ embedding} = \texttt{CROSSENTROPY(P(M(o)), y}$$

where P is a small capacity network (in our case a single layer) that takes the MDP embedding and predicts the class corresponding to the batch which the transition came from. In this way, the output of the network M is predictive of the task, but can also be predictive of the stage of the task (ie differentiating large magnitude actions when the goal is far away and small when the goal is near, or differentiating sub-tasks of the task as we will see later).
We also train, using the same transition, a meta-Q ($\hat{Q}$) function and meta-G ($\hat{G}$) function to clone the output of the original Q and G, specifically,

$$L_{meta\ G} = ||\hat{G}(s_i, M(o)) - G_i(s_i)||_2^2 + L_{aux}$$

$$L_{aux} = 1. - \frac{\hat{G}(s_i, M(o))G_i(s_i)}{||\hat{G}(s_i, M(o))|| \cdot ||G_i(s_i)||}$$

$$L_{meta\ Q} = ||\hat{Q}(s_i, \hat{G}(s_i, M(o)), M(o)) - Q_i(s_i, G(s_i))||_2^2$$

where we have the distillation losses for G and Q as well as an additional loss for encouraging the action proposals to move in the correct direction. It is important to note that since G is a variational autoencoder, it is important to ensure that the same latent vector is used to train $\hat{G}$ as was sampled in G to ensure that there exists a one-to-one distillation of the knowledge.

7 Experiments

Here we outline the training experiments along with necessary hyperparameters and modifications for replication.

7.1 Stage 1: BCQ Policy Training

As noted before, we examine the case with 500 demonstrations for reaching and lifting because of the diversity of reaching and the data imbalance for grasping; we also examine the case with 100 demonstrations for pushing. The results of phase 1 training can be seen in Table 1 and Fig. **??**.

Task	Expert Policy Return	BCQ Policy Return	Epochs
Reach	119.07	151.16	100
Lift	136.45	206.4	50
Push Right	160.20	296.5	30
Push Left	189.42	234.4	30
Push Up	192.87	298.6	30
Push Down	188.58	304.2	30

Table 1: Results from the first stage training of the BCQ agents to accomplish each of the short horizon sub-tasks. Note horizon for these rollouts was 200. It is clear that the learning in these restricted task instances is efficient.

It is clear from the results that the training of the agents is successful and furthermore that training on these short time horizon tasks is extremely efficient. We were able to learn successful policies in just a few epochs and show that sometimes these polices can perform better than their expert counterparts; this is not unusual as the original BCQ work observed this behavior as well. Notably, we eliminate the perturbation network that is proposed for BCQ as we did not see any improvement with the inclusion of such a network and it would have required an additional network to distill. We train using Adam optimization with learning rate $1e-4$ with 100 action proposals for the policy during evaluation.

7.2 Stage 2: Distillation

We train the distillation phase of the process on all the tasks resulting in a 6 way classification problem for the MDP identification step. We observe that the MDP identification problem is highly feasible and straightforward, being able to correctly classify the MDP 95% of the time; however, the learned

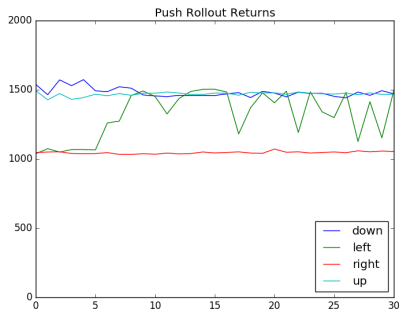


Figure 3: Stage 1 training on the push tasks. The graph shows the rollout return over a horizon of 1000 timesteps at the end of each training epoch. It is clear that the agent was able to effectively learn in relatively few number of epochs. We note that as training progressed that even though the means did not change significantly, the variance of the returns decreased.

embedding is highly useful in the distillation phase. The learned embedding space allow the multi task knowledge distillation to be effective. The results are summarized in Table 2. We train using Adam with learning rate $1e-4$ with the MDP identification task using a one-layer network predicting the task from the MDP embedding. We find that using a deeper MDP identification network is unnecessary and makes the disentanglement of MDP embeddings more difficult.

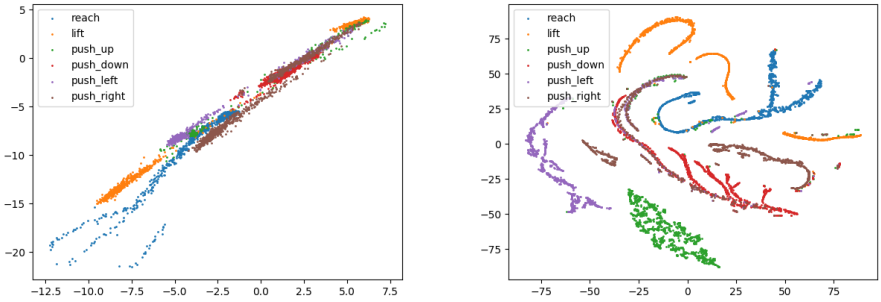
Task	**Performance of Distilled Policy on Task**	**% Change**
Reach	105.13	70%*
Lift	241.15	116%
Push Right	281.09	95%
Push Left	270.57	115%
Push Up	288.21	96%
Push Down	283.27	93%

Table 2: Results from the distillation step, we can see that the distillation process was able to learn a combined policy that is able to complete all the tasks individually at a performance on par with that of the original policies. Asterisks denote tasks where the return is lower than that of the BCQ policy, but the task was still solved as a result of the magnitude of the actions in the distilled policy being less than the original policy. Furthermore, we note that the instances with higher return than the base BCQ policy tend to move with large action magnitudes than the base BCQ policies as the direction loss was weighted higher during training than the L2 loss on actions.

7.3 Use of MDP Embeddings

We see that the embeddings of the transitions are clustered allowing for the successful classification of the MDPs. Furthermore, we observe the emergent property of discovered clusters representing task variation within the sub-tasks. This is clearly visible in the t-SNE embeddings in Fig. 4b where there are three distinct clusters for lifting, one refers to moving down to the cube, another grasping, and the third lifting the cube upward. These distinct per-task and intra-task groupings suggest that this approach could actually be used for unsupervised sub-task discovery. See Fig. 4 for visualizations of the embedding space. We note that the embedding space can be extended to arbitrarily many dimensions; however, experimentally we found 2 to perform well.

We also observe that the MDP embeddings for a given task changes over time. Furthermore, the embeddings change in task space, which suggests that the embeddings are actually meaningful in terms of task completion. This change is an interesting property as it shows almost TCN like properties even when not trained on the contrastive objective. The ends of different trajectories (at success) are located in the same location in embedding space, and furthermore, level sets of semicircles with center at the confluence of the different trajectories is predictive of distance to the goal. This suggests that the MDP embeddings themselves are useful for exploration and producing



(a) The raw MDP embeddings for trajectories for each of the tasks.

(b) A t-SNE visualization of the learned embedding space.

Figure 4: Visualizations of the 2-D learned MDP embedding space through the distillation phase, transitions taken from expert trajectories. It is clear from (a) that the learned space shows distinct clusters for each of the tasks, as well as a space of shared embeddings in the area just to the bottom right of the legend in (b) showing an area that is common to all of the pushing and lifting tasks: reaching down to the cube. These visualizations show that the learned space is meaningful and potentially useful as an exploration space.

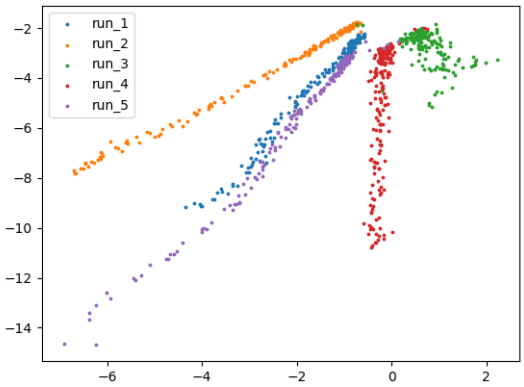


Figure 5: Embeddings for the reach MDP across different episodes. Colors indicate different episodes. The convergence of the trajectories occurs at the end of the trajectories. We also observe that circles centered at the place of convergence (approximately $(0, -2)$) intersect with the trajectories in locations where the robot was at the same distance from the goal. This is an interesting emergent property of this approach.

different behaviors like varying speeds (further away from the end, the arm moves faster whereas closer the arm moves slower).

7.4 One-shot Imitation Learning

We evaluate the possibility for using this method as a way to imitate a trajectory. We can take the trajectory and embed each of the transitions into their values m and use those as the embeddings passed to the distilled policy functions. We observe that this approach allows for robust imitation of the behavior under perturbations to the magnitude and direction of the actions. This approach is succcessful for compositions of the sub-tasks like reaching followed by lifting followed by reaching again.

7.5 MDP Embedding Interpolation as Exploration

We observe that interpolation in the MDP embedding space results in distinct behaviors suggesting that such an approach could be a useful space of exploration behaviors to build reinforcement learning algorithms on top of (for instance hierarchical approaches or further distillation). For example, in the reaching trajectories from Fig. 5, we can begin on one of the trajectories, and interpolate in the latent space along a curve to a different trajectory and then continue on the new trajectory to solve the task. This suggests that the latent space could be useful for interpolating between different solution strategies as well.

8 Conclusion and Future work

We presented a method for learning and composing short time horizon tasks to create longer and more complicated tasks. The proposed method takes batches of data on many sub-tasks and leverages them to learn a meta policy that is capable of solving all of the different sub-tasks. We show a straightforward method to leverage demonstrations to learn a latent space that is predictive of the task and usable to improve policy learning. This latent space is rich and shows potential for use in hierarchical settings, few-shot imitation, and use as a higher-level exploration policy. The presented method operates completely in a batch off-policy setting making it ideal for use with robots in the real world and does not require expensive assumptions or inductive biases as in previous works. We also show the interesting property that the learned embeddings encode task relevant information like distance to goal which is an emergent property of this approach. This result is important because if the MDP embeddings encode task relevant information, they could be used in a learned reward function similarly to the metric spaces learned in contrastive methods like Time Contrastive Networks. However such explorations of distances in the learned space is left to future work.

The results in this work are encouraging and suggest that using this approach might enable in batch off-policy hierarchical reinforcement learning from demonstrations. This is important as it could enable learning of complex and highly staged tasks like those in industrial settings. Furthermore, explorations of image based state spaces would be beneficial as well considering that image based policies can solve tasks in which it might not be straightforward to design states (ie non-rigid objects). Since the proposed distillation step is lightweight, it should be possible to efficiently learn in pixel space.

References

[1] P. Christiano, Z. Shah, I. Mordatch, J. Schneider, T. Blackwell, J. Tobin, P. Abbeel, and W. Zaremba. Transfer from simulation to real world through learning deep inverse dynamics model. *arXiv preprint arXiv:1610.03518*, 2016.

[2] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[3] L. Fan, Y. Zhu, J. Zhu, Z. Liu, O. Zeng, A. Gupta, J. Creus-Costa, S. Savarese, and L. Fei-Fei. Surreal: Open-source reinforcement learning framework and robot manipulation benchmark. In *Conference on Robot Learning*, pages 767–782, 2018.

[4] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *CoRR*, abs/1703.03400, 2017.

[5] C. Finn, S. Levine, and P. Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning*, pages 49–58, 2016.

[6] S. Fujimoto, D. Meger, and D. Precup. Off-policy deep reinforcement learning without exploration. *CoRR*, abs/1812.02900, 2018.

[7] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. *CoRR*, abs/1812.07252, 2018.

[8] S. Lange, T. Gabel, and M. Riedmiller. Batch reinforcement learning. In *Reinforcement learning*, pages 45–73. Springer, 2012.

[9] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[10] S. Mirzadeh, M. Farajtabar, A. Li, and H. Ghasemzadeh. Improved knowledge distillation via teacher assistant: Bridging the gap between student and teacher. *CoRR*, abs/1902.03393, 2019.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[12] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6292–6299. IEEE, 2018.

[13] X. Pan, Y. You, Z. Wang, and C. Lu. Virtual to real reinforcement learning for autonomous driving. *arXiv preprint arXiv:1704.03952*, 2017.

[14] E. Parisotto, J. L. Ba, and R. Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2015.

[15] A. A. Rusu, S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

[16] F. Sung, L. Zhang, T. Xiang, T. M. Hospedales, and Y. Yang. Learning to learn: Meta-critic networks for sample efficient learning. *CoRR*, abs/1706.09529, 2017.

[17] F. Torabi, G. Warnell, and P. Stone. Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954*, 2018.

[18] Q. Vuong, S. Liu, M. Liu, K. Ciosek, H. Su, and H. I. Christensen. Pre-training as batch meta reinforcement learning with time, 2019.

[19] Q. H. Vuong, Y. Zhang, and K. W. Ross. Supervised policy update. *CoRR*, abs/1805.11706, 2018.

[20] K. Xu, E. Ratner, A. Dragan, S. Levine, and C. Finn. Learning a prior over intent via meta-inverse reinforcement learning. *arXiv preprint arXiv:1805.12573*, 2018.

[21] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. *arXiv preprint arXiv:1910.10897*, 2019.

[22] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.

[23] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. 2008.

[24] Y. Zou and J. Feng. Hierarchical meta learning. *CoRR*, abs/1904.09081, 2019.