ONE DEMONSTRATION IMITATION LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

We develop a new algorithm for imitation learning from a single expert demonstration. In contrast to many previous one-shot imitation learning approaches, our algorithm does not assume access to more than one expert demonstration during the training phase. Instead, we leverage an exploration policy to acquire unsupervised trajectories, which are then used to train both an encoder and a context-aware imitation policy. The optimization procedures for the encoder, imitation learner, and exploration policy are all tightly linked. This linking creates a feedback loop wherein the exploration policy collects new demonstrations that challenge the imitation learner, while the encoder attempts to help the imitation policy to the best of its abilities. We evaluate our algorithm on 6 MujoCo robotics tasks.

1 OPENING REMARKS

The sample efficiency of imitation learning is really and truly incredible. After training on only a few dozen examples, imitation learning agents can acquire new skills that mimic the behaviors of experts (Abbeel & Ng, 2004; Ziebart et al., 2008; Ho & Ermon, 2016; Zhang et al., 2018). This is in stark contrast to reinforcement learning (RL), wherein agents routinely require millions or even billions of data points before they can acquire a new skills (Haarnoja et al., 2018; Mnih et al., 2015; Andrychowicz et al., 2017). What's more, imitation learning does not require access to reward functions, which must be crafted by skilled researchers and are often fiendishly difficult to design. Still, for all its advantages, imitation learning is not particularly convenient. Acquiring expert demonstrations is often onerous, requiring some combination of motion controls, virtual reality, or teleoperation (Zhang et al., 2018). Further, the policies acquired via imitation are often brittle and limited. They will break when the desired behavior changes even slightly.

Recently, there has been a series of monographs that try to address the shortcomings of imitation learning by considering the problem of one-shot imitation learning (Duan et al., 2017; Finn et al., 2017; Wang et al., 2017). In the one-shot setting, agents must learn to acquire new skills from only a single demonstration. Broadly, this is usually done by meta-training an imitation policy over a distribution of tasks, each of which requires its own expert trajectories. Using this task distribution, agents meta-learn some model that allows them to do quick inference on a new expert demonstration at test time. While this is a great idea in theory, requiring a distribution of related tasks is often an onerous assumption. Indeed, collecting expert demonstrations for a single task is often quite difficult, let alone an entire distribution. In practice, these task distributions are often rather limited and most of the tasks an agent can accomplish at inference time are quite similar to the training tasks, i.e. training and test tasks both involve block stacking. In many ways, it seems one shot imitation learning is simply kicking the can down the road.

Given the burden of providing training data for one-shot imitation learners, one might ponder what kind of algorithm we could develop if we simply removed this requirement. This leads us to ask the (perhaps foolish)[1] question: *Can we develop an effective one shot imitation learning algorithm that assumes access to only a single expert demonstration?* In this paper, we construct such an algorithm. Presented with the constraints of this problem, we proceed in perhaps the only reasonable way: by attempting to collect diverse unsupervised trajectories, and then using those trajectories to train an imitation policy. This type of approach to one demonstration imitation has been considered multiple times before, most notably in (Pathak et al., 2018; Nair et al., 2017), which enjoyed a great deal of

[1] See Section 5.3.5 question 7 for discussion on why this question might be inherently foolish, and how this foolishness can be largely eased by instead considering the problem of *two-demonstration* imitation learning.

success in knot tying and navigation domains, and in (Wang et al., 2017), which likewise enjoyed a great deal of success in locomotion domains.

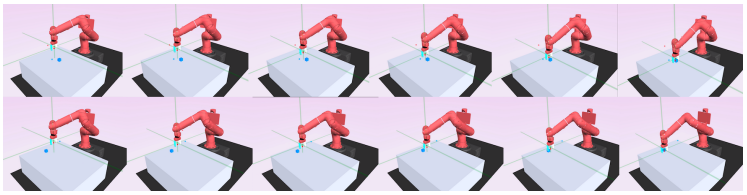


Figure 1: Top: A single expert demonstration of the Sawyer Pick Task. Bottom: Our method, having been trained to imitate trajectories from an exploration policy, uses the top demonstration in an attempt to mimic the expert.

In contrast to prior approaches, we wanted to develop an algorithm that does not rely on pre-training a powerful encoder (as in Wang et al. (2017)), nor on a forward predictive model exploration heuristic (Pathak et al., 2018). Instead, we wanted to build an algorithm where exploration for new unsupervised trajectories and the capabilities of the imitation learner are tightly linked, and grow continuously together. Broadly, our algorithm proceeds by leveraging an exploration policy to acquire unsupervised trajectories. An encoder module then consumes an entire unsupervised trajectory and produces a context z. This context, along with the present state, are used in a behavioral cloning algorithm that trains both the encoder and the imitation network end-to-end. Finally, a diversity reward based on the KL divergence between these encodings' underlying distributions is calculated. The exploration policy is subsequently trained on this reward to visit trajectories that produce more diverse encodings, completing the loop. At inference time, the single expert demonstration is encoded and the imitation policy is run forward. Full algorithmic details and experimental results are presented below.

2 Related Work

Our work is closely related to the field of meta-learning (Thrun, 1996; Schmidhuber et al., 1997; Snell et al., 2017), specifically the idea of meta-learning an inference network (Gordon et al., 2018; Chen et al., 2019). Meta-learning has itself been applied to the problem of imitation (Ghasemipour et al., 2020; Xu et al., 2018a), sometimes specifically in the context of one-shot learning (Duan et al., 2017; Finn et al., 2017). Our approach is related to the idea of meta imitation learning with automatic task generation. Automatic task proposal methods have themselves been investigated extensively, for example in (Gupta et al., 2018; Hsu et al., 2019; Storck et al., 1995; Schmidhuber, 2015). The two closest automatic task generation papers, in terms of our considered problem setting, are probably (Wang et al., 2017; Pathak et al., 2017), which are discussed above.

We make extensive use of a trajectory embedding module. The idea of embedding a trajectory into a useful context is an evergreen one. Recent interesting work on learning embeddings that are good for planning includes (Watter et al., 2015; Srinivas et al., 2018). Kurutach et al. (2018) and Thomas et al. (2017) try to learn representations with causal interpretations, in the sense that the agent should learn how to independently manipulate factors of variation in the environment. The specific type of encoder we consider in this paper also features in Garnelo et al. (2018) for neural processes and Rakelly et al. (2019) for RL. In Vezzani et al. (2019), the core idea was to learn encodings that are useful for exploration, in the sense that visiting encoded states with low density is an exploration metric that approximates something like Kolter & Ng (2009).

Finally, one of the underlying ideas in this paper is learning an exploration policy to fill a buffer with data that is useful for an imitation learner. Filling a replay buffer with exploration policy data was previously considered in (Xu et al., 2018b). Other interesting candidates for curiosity signals are Intrinsic Curiosity Modules (Pathak et al., 2017), DIAYN (Eysenbach et al., 2018), and VIME (Houthooft et al., 2016). See (Ngo et al., 2012; Schmidhuber, 1991) for general references on curiosity.

3 PROBLEM STATEMENT

3.1 PRELIMINARIES

We consider infinite-horizon Markov decision process (MDPs), defined by $(\mathcal{S}, \mathcal{A}, P, R, T)$, where $\mathcal{S}$ denotes the state space and $\mathcal{A}$ the action space. P is the transition probability, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$. R is the reward function $R : \mathcal{S} \to \mathbb{R}$. T is the horizon. A policy $\rho_\phi(a_t|s_t)$ gives a distribution over the action space given the current state. Often, but not always, policies are trained to maximize the expected sum of discounted reward $\sum_t \mathbb{E}_{(s_0,s_2,\ldots)}[\gamma^t R(s_t)]$, for some $\gamma < 1$, a discount factor. In this paper, we will make use of the Soft Actor Critic algorithm, which instead solves the maximum entropy objective $\sum_t \mathbb{E}_{(s_0,s_1,\ldots)}[\gamma^t R(s_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))]$, using an actor-critic model. For more details, refer to Haarnoja et al. (2018).

In imitation learning, an agent is supplied with expert demonstrations $\{\tau\}_{i=1}^N$, where $\tau_i = [(s_0, a_0), (s_1, a_1), (s_2, a_2), ...]_i$. Behavioral cloning (also called supervised learning in other contexts) fits a model $\pi_\theta(s_j) = \hat{a_j}$, which is trained with the loss $\mathcal{L}_\theta = d(\pi_\theta(s_j), a_j)$. Here, d is a distance metric – often Euclidean distance or cross entropy. However, this distance metric can also be a learned density metric, in which case an inverse reinforcement learning method is required (Abbeel & Ng, 2004; Ziebart et al., 2008; Ho & Ermon, 2016). For this paper, we will assume WLOG that Euclidean loss is used unless otherwise stated.

3.2 THE ONE DEMONSTRATION IMITATION LEARNING OBJECTIVE

We consider a scenario where we are given a single expert demonstration τ. Assume that the expert acted according to some reward function R_E. Then, our goal is to train an imitation learner π_θ that takes as input τ, along with the current state s_t and maximizes the discounted sum of future rewards under R_E.

$$\max_\theta \mathbb{E}_{a_t \sim \pi_\theta(s_t;\tau)} \left[\sum_{t=0}^{T} \gamma^t \cdot R_E(s_t) \right] \tag{1}$$

Note the dependence of π_θ on τ. Of course, we usually do not have access to R_E. We will, however, assume that R_E exists and use it for evaluation purposes only, similarly to (Ho & Ermon, 2016).

If a second expert demonstration exists, then we can write down an alternative form of this objective. Suppose τ_1, τ_2 are both expert trajectories. Let s_i, a_i be a state action pair from τ_2. Then the goal is to minimize

$$\sum_i \|\pi_\theta(s_i; \tau_1) - a_i\|_2^2 = \sum_i \|\hat{a}_i - a_i\|_2^2 \tag{2}$$

In words, we want to train a policy π_θ that can use τ_1 to infer what actions τ_2 would have taken at states $s_i \in \tau_2$.

4 AN ALGORITHM FOR ONE DEMONSTRATION IMITATION LEARNING

We will proceed by essentially making an approximation to Equation 2, in the hope that the learned policy resulting from this approximation is still performant on the original objective Equation 1. This approximation, essentially, stems from a lack of access to expert trajectories τ_E at training time. Instead, we want to collect diverse unsupervised trajectories τ_i which can be used to train the imitation learning objective. For just a moment, let's forget about how we will acquire these unsupervised trajectories and assume they are collected randomly. Initialize a buffer $\mathcal{B}$ and fill it with random trajectories τ_i.

Let us turn our attention towards what kind of architecture we should use for the imitation learner $\pi_\theta(s; \tau)$, where $\tau \sim \mathcal{B}$. This model will need to consume all of τ, which is a (potentially long) sequence of state-action pairs. A natural choice here would be to encode τ with of recurrent network, and then combine the encoded context z with the agent's current state to form the input for the imitation learner.

$$z = RNN(\tau)$$
$$\pi_\theta(s_t, z) = \hat{a}_t$$

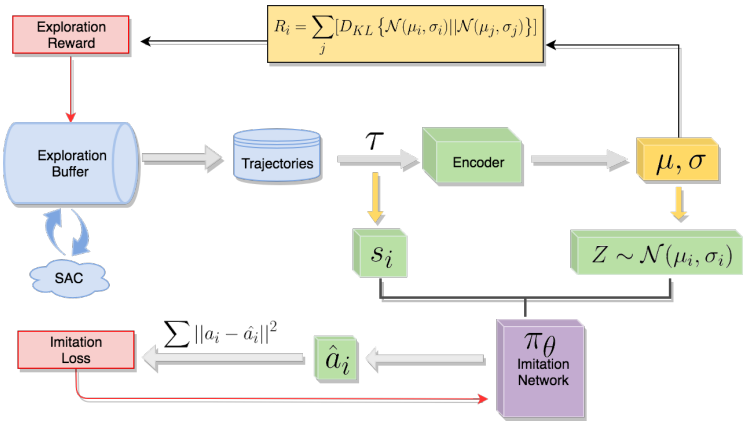


Figure 2: Training starts with initializing the exploration buffer $\mathcal{B}$, with trajectories generated by the SAC policy (blue area). A batch of trajectories $\{\mathcal{T}_i\}$ are then sampled from the buffer and are passed into the encoder $\mathcal{E}$ to produce contexts z_i for each $\mathcal{T}_i$. The imitation network (purple box) π_θ takes in both current state s_i and its corresponding trajectory context z to predict next action a_i. It is trained with imitation loss (bottom red box). To encourage diversity of trajectories, diversity reward R is computed as in Equation 6 (top yellow box). SAC (blue cloud) is then trained with R to update the exploration buffer $\mathcal{B}$, completing one iteration of the training pipeline.

This type of pipeline can be trained end-to-end, as in Duan et al. (2017). We considered this (see Section 5). However, we ultimately decided on using a Gaussian factor model as a trajectory encoder. We first read about this type of encoder in the context of neural networks in (Garnelo et al., 2018) (Equation 6), although prior development can be attributed to (Eslami et al., 2018). A similar model was also later featured in (Rakelly et al., 2019). Overall, the encoder's graphical model interpretation of finding a global task prior convinced us that it would be a good fit for our problem. See those references for further discussion on the benefits of this type of encoder model.

The Gaussian factor encoder proceeds by selecting state and action pairs $(s_k, a_k) \subset \tau$. These pairs are passed through a parametric model to parameters that are treated as the mean and variance of a normal distribution

$$\mu_k, \sigma_k = f_\phi(s_k, a_k) \tag{3}$$

Finally, a context can be sampled from the product of Gaussians over the entire trajectory.

$$z_\tau \sim \prod_{k=1}^{T} \mathcal{N}(\mu_k, \sigma_k) \tag{4}$$

By using the reparameterization trick (Kingma & Welling, 2013) to backpropogate through the sampling process, this model can be trained end to end simultaneously with the imitation learner. We call the process from Equations 3 and 4 the encoder. We write

$$\mathcal{E}(\tau) = z_\tau \tag{5}$$

to denote taking an entire trajectory τ and encoding it with equations 3 and 4.

We are finally ready to return to the question of how we should fill our buffer $\mathcal{B}$ with diverse unsupervised trajectories. One tantalizing possibility is to collect trajectories that produce maximally different encodings. This serves the dual purpose of finding unique trajectories (trajectories with different encodings likely have different properties) and finding trajectories that force the encoder and imitation learner to grow and capture a wider variety of interesting behaviors. Suppose τ_i has encoded mean and variance[2] μ_i, σ_i. We define a diversity reward over trajectories as

$$R_i = \sum_j D_{KL}\left(\mathcal{N}(\mu_i, \sigma_i) || \mathcal{N}(\mu_j, \sigma_j)\right) \tag{6}$$

[2]This is the mean and variance of the product of Gaussian PDFs from equation 4, which is itself a Gaussian function. See (Bromiley, 2003) for calculation details.

Algorithm 1 $One-Demo$ Imitation Learning Algorithm

Require: Learning rates: α, β
Initialize $\theta_{pol}, \theta_{im}, \theta_{en}$ for exploration policy ρ, imitation network π and encoder $\mathcal{E}$, respectively.
while not done **do**
  Initialize buffer $\mathcal{B}$ with trajectories rolled out from exploration policy ρ
  for number of imitation training iterations **do**
    Sample batch of trajectories $\{\mathcal{T}_i\} \sim \mathcal{B}$
    Encode each trajectory $\mathcal{T}_i \in \{\mathcal{T}_i\}$ to its context z_i using encoder $\mathcal{E}$
    Sample batch of transitions $\tau = \{s_1, a_1, ...s_j, a_j\}$ from $\{\mathcal{T}_i\}$
    Compute $\hat{a_j} = \pi_(s_j, z_i)$ for each s_j
    Update $\theta_{im} \leftarrow \theta_{im} - \alpha \nabla_{\theta im} \mathcal{L}(\sum_i \|\hat{a}_j - a_j\|_2^2)$
    Update $\theta_{en} \leftarrow \theta_{en} - \beta \nabla_{\theta en} \mathcal{L}(\sum_i \|\hat{a}_j - a_j\|_2^2)$
  end for
  for all $\mathcal{T}_i \in \{\mathcal{T}_i\}$ **do**
    Compute diversity reward $R_i = \sum_j D_{KL}\left(\mathcal{N}(\mu_i, \sigma_i) || \mathcal{N}(\mu_j, \sigma_j)\right)$
    Add R_i to $\mathcal{B}$.
  end for
  for number of exploration training iterations **do**
    Train SAC to update θ_{pol} with $\{\tau_i, R_i\} \sim \mathcal{B}$.
  end for
end while

Which is given explicitly as

$$\sum_j \log\left(\frac{\sigma_i}{\sigma_j}\right) + \frac{\sigma_i^2 + (\mu_i - \mu_j)^2}{2\sigma^2} - \frac{1}{2}$$

This reward, R_i, is given to each trajectory i in the buffer at its terminal state. SAC (Haarnoja et al., 2018) is then trained to maximize this reward and subsequently used to collect more trajectories. Although we found giving each trajectory a reward at only its terminal state to be sufficient, in principle one could compute this reward at every timestep j by using only the first j timesteps of every trajectory. We empirically compared these two reward choices in 7.2

$$\mu_{i,j}, \sigma_{i,j} = \mathcal{E}\left[(s_0, a_0, \ldots, s_j, a_j)_i\right] \tag{7}$$

$$R_{i,j} = \sum_k D_{KL}\left(\mathcal{N}(\mu_{i,j}, \sigma_{i,j}) || \mathcal{N}(\mu_{k,j}, \sigma_{k,j})\right) \tag{8}$$

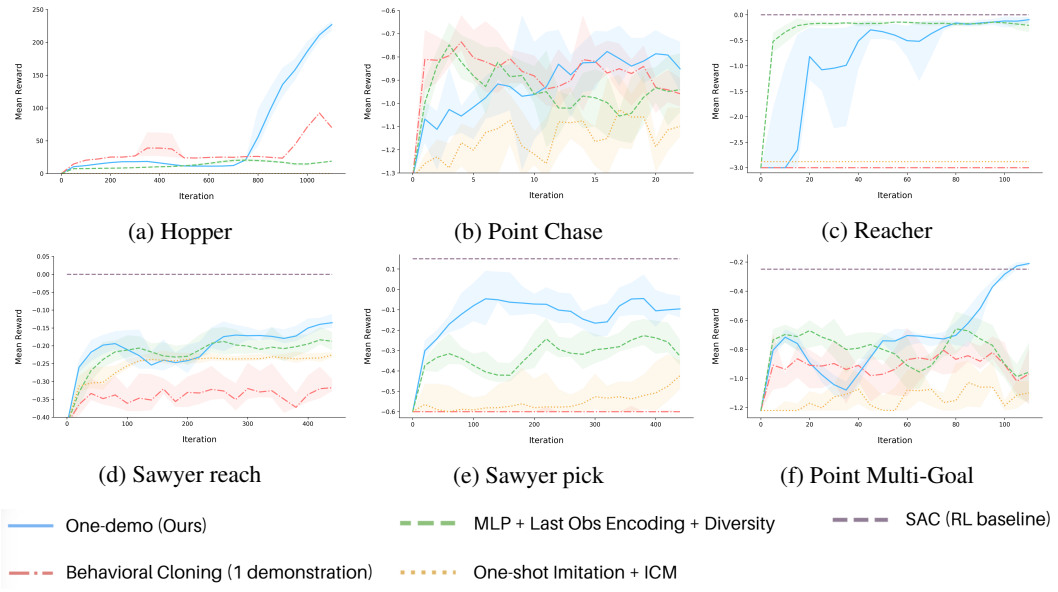
This alternative diversity reward is considered in Section 7.2.

We can now approximate Equation 2 during training by using $\tau \sim \mathcal{B}$ that were obtained by an exploration policy trained under Equation 6. Only one small snag remains. Equation 2 assumes access to two paired trajectories τ_1, τ_2. Our trajectories, however, are unpaired. In principle, one could pair trajectories with the closest encodings. However, in practice, we found it sufficient to simply sample input states s_i from τ_i during training. Although this has been known to cause issues (Ross et al., 2011), it did not deter us. We leave the pairing of unsupervised exploration trajectories during training as interesting future work.

5 Experiments

5.1 Environments

Experiments are run on 6 MuJoCo environments. One of the environments, Hopper-V3, is taken from OpenAI Gym. Another, Reacher, is a modification of Gym's Reacher-V2 that augments the observation space. In Point Chase, a point mass has to chase another point mass around a bounded arena. In Point Multi-Goal, a point mass must go to one of several potential stationary goals. In Sawyer Reach, a Sawyer robot arm must reach a specified goal location. In Sawyer Pick, a Sawyer robot arm must move to a block on a table and close its gripper around the block.



(a) Hopper (b) Point Chase (c) Reacher

(d) Sawyer reach (e) Sawyer pick (f) Point Multi-Goal

Figure 3: Learning Curves for 6 environments. Averaged over 10 seeds. The RL baseline for hopper, which is around 3000, is omitted because of the graph's scale. Similarly, the RL baseline for Point-Chase, which is 0, is omitted due to scale.

5.2 Primary Results

The primary learning curves are presented in Figure 3. The evaluation step is completed by taking a single expert trajectory and running it through the encoder to produce $\mathcal{E}(\tau_E) = z_E$. Subsequently, z_E is given to the imitation policy π_θ, along with the agent's current state s to produce actions $\hat{a}$. This process continues until the trajectory's horizon is reached. At each time step, the environment produces a ground truth RL reward, which is what we plot in Figure 3 on the y-axis.[3] This reward is only used for evaluation purposes.

In addition to our algorithm, we consider the following baselines:

Behavioral Cloning (1 demo): Standard supervised learning with a single expert demonstration.

One Shot Imitation + ICM: The entire demonstration is consumed via the GRU before its output is given concatenated to the current state. This resulting concatenated vector is put through an MLP to produce an action prediction. Diverse training trajectories for self-imitation are obtained via an ICM Pathak et al. (2017). We also attempted to obtain these trajectories with a diversity metric on the output encoded states h from the GRU. However, this method totally failed to learn.

MLP + Last Obs Encoding + Diversity: A baseline of this form appears in both Finn et al. (2017) and Duan et al. (2017). The last observation in the expert trajectory is processed through an MLP, producing an output h. This h is then concatenated with the current state, and the result is sent through another MLP before guessing the expert action. To collect diverse trajectories for self-imitation, we use a diversity loss on the output encodings h across trajectories, similar to equation X, only replacing KL divergence with Euclidean Loss. We also considered an ICM loss to collect diverse trajectories, but for this baseline, a diversity loss worked much better.

5.3 Further Analysis and Discussion

During the course of developing our algorithm, several design decisions required careful consideration. In this section, we do our best to recount those key design decisions, presenting the options we

[3]To be clear, for Hopper we plot the returns, as is standard. For every other environment, we plot the reward at the final state.

considered and the evidence we gathered to help inform our final decisions. Additional analysis is presented in Appendix A.

5.3.1 ON THE ENCODER'S EFFECTIVENESS

1. Does the imitation learner benefit from the presence of the encoder?

Yes. To verify this, we plot two curves. First, the validation loss curve of an imitation learning policy trained with available context encoding z. Second, the loss curve for an imitation policy without access to this context. Both imitation learners are trained simultaneously on the same trajectories. To be clear, this validation loss is the aggregate euclidean distance between the imitation policy and the ground truth actions, $\sum_t \|\pi(s_t, z) - a_t\|^2$ for the policy with encoder and $\sum_t \|\pi(s_t) - a_t\|^2$ for the policy without.

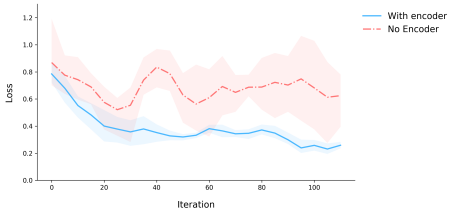


Figure 4: Reacher imitation policy evaluation loss, with encoder (blue) and no encoder (red).

5.3.2 ALTERNATIVES TO THE DIVERSITY REWARD IN EQUATION 6

2. In our algorithm, diverse unsupervised trajectories are encountered by training a SAC policy to maximize the reward from Equation 6. How does this compare to alternative methods of unsupervised trajectory collection? The simplest baseline here is to collect random trajectories. Alternatively, if more sophistication is desired, one could use a curiosity signal such as an ICM to encourage visitation of unique states. Moreover, there is the possibility of combining the diversity reward in Eq 6 with a curiosity module such as an ICM. Results are presented in Figure 11. In principle, one can use any method they desire to collect unsupervised trajectories. For example, (Thomas et al., 2017) or (Vezzani et al., 2019) might be an interesting choice. However, a major benefit of our algorithm is the tightly linked optimization between exploration, the encoder, and imitation. This is lost with a generic method of unsupervised trajectory collection.

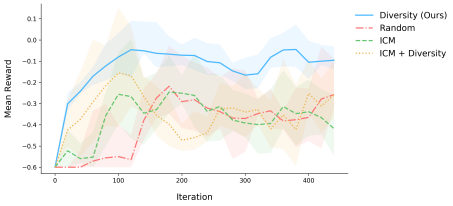


Figure 5: Sawyer Pick, alternative methods for unsupervised traj collection.

5.3.3 ALTERNATIVE ARCHITECTURE CHOICES: GAIL FOR IMITATION OR VAES FOR ENCODING

3. Can you use GAIL (Ho & Ermon, 2016) or other inverse RL techniques for imitation learning, rather than behavioral cloning?

Yes. One could collect the unsupervised trajectories using the diversity metric from One-Demo, and then use the encoded trajectories as an input for the GAIL policy. The GAIL algorithm also requires learning a density metric (inverse reward function), which will also have to be conditioned on the encoded state z. Recent work Ghasemipour et al. (2020) suggests that the best results can be obtained by only backpropagating through z via the density metric discriminator loss, while stopping gradients through z during policy training. We used this trick and substituted GAIL for behavioral cloning in our algorithm pipeline. This typically performed worse than behavioral cloning, probably because of the added instability of training both the GAIL policy (with policy gradients) and the GAIL reward discriminator. See Figure 6.

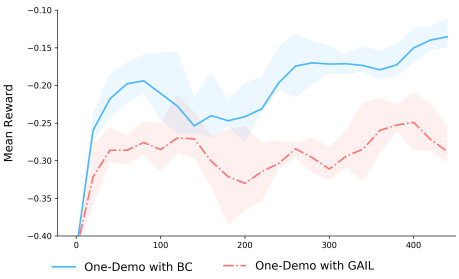


Figure 6: Sawyer Reach with BC (blue) and GAIL (red)

4. Did you consider another architecture for the encoder, such as a VAE? In Figure 3 we considered an RNN encoder and an MLP encoder. We did also consider a variational autoencoder, but found it to be ineffective. See Figure 7. Similar methods that have had success with this type

of encoder have typically pre-trained the VAE on a large number of sensible demonstrations (Wang et al., 2017). We assumed no access to such demonstrations, so getting such an encoder to work reliably was difficult.

In any case, our algorithm's novelty is not reliant on which encoder one chooses, so this design decision is more of a secondary concern. See Pathak et al. (2018) for more discussion on the drawbacks of VAEs in this problem setting.

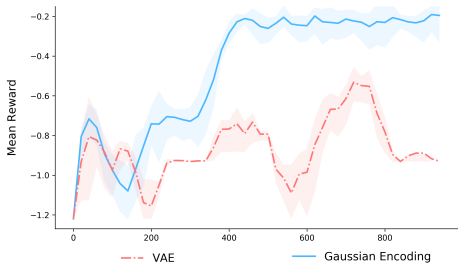


Figure 7: VAE (red) vs Gaussian Encoder (blue), Point Multi-Goal

5.3.4 Final considerations: multiple demos and failure cases

5. What happens if you have more than one expert trajectory? Does performance improve? There are two simple methods of adapting our algorithm to multiple trajectories. First, one could compute the encoding z_i from Equation 4 for each of the demonstrations i and then average together the resulting z_i to get $z = \frac{1}{n}\sum z_i$. This averaged z can then be fed into the imitation network. Alternatively, each of the individual z_i can be fed into the imitation network, producing n actions a_i. These actions can then be averaged and the mean action taken. We consider developing more principled techniques for two-demonstration imitation learning an interesting avenue for future work. Results using context averaging are presented in Figure 8. One, two,five, and ten demo cases are considered.

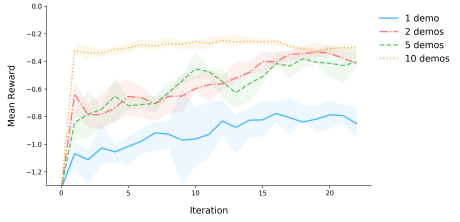


Figure 8: Point Chase, varying number of demos.

6. This method has an identifiability problem: For example, if an expert demonstration shows a robot arm moving right to pick a blue block, how should the imitation agent know whether the true intention of the expert was to move the hand to the right or move the hand towards the block? If, at test time, the blue block is to the agent's left, should the imitation agent move to the right (as the expert did), or towards the block? It is true that this problem exists, but it's fundamentally inherent to the one-shot imitation setting, and not unique to our method. This lack of identifiability cannot be addressed without introducing a prior, feature space engineering, fixed reset positions, including sparse reward functions, or using at least two demonstrations. In practice, we've found that learning still occurs in spite of this potential issue, possibly because we considered domains where the initial state distribution had sufficiently low variance. In problem domains where this identifiability does become an issue, using some small number of demonstrations is possible, as outlined above. As we saw, performance on Point Chase (where identifiability matters) improves dramatically by adding just a single additional expert demonstration.

6 Closing Remarks

Before this research project began, we developed an environment called *Cylinder*. In this environment, a point mass must push a small cylinder onto a goal location in the environment. To our surprise, *Cylinder* was quite challenging, and could not be solved even by state of the art RL algorithms after millions of iterations. Now curious about the problem's overall level of difficulty, we started collecting expert trajectories, to see how many it would take to solve the environment via imitation learning.

Eventually, GAIL did solve this environment, but only after 2500 trajectories were manually collected with a mouse and keyboard. This lead us to wonder, could we develop vastly more efficient algorithm? This paper was crafted with this purpose in mind. Unfortunately, all algorithms considered in this paper failed to make any progress on the task whatsoever, even when we allowed for five expert demonstrations rather than one. Thus, results for *Cylinder* were omitted from our results. Along with our code for this paper, we will also release this environment, in the hopes that some day somebody can figure out how to solve it with only one (or two) expert demonstrations.

REFERENCES

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

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

Paul Bromiley. Products and convolutions of gaussian probability density functions. *Tina-Vision Memo*, 3(4):1, 2003.

Yutian Chen, Abram L Friesen, Feryal Behbahani, David Budden, Matthew W Hoffman, Arnaud Doucet, and Nando de Freitas. Modular meta-learning with shrinkage. *arXiv preprint arXiv:1909.05557*, 2019.

Yan Duan, Marcin Andrychowicz, Bradly Stadie, Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. In *Advances in neural information processing systems*, pp. 1087–1098, 2017.

SM Ali Eslami, Danilo Jimenez Rezende, Frederic Besse, Fabio Viola, Ari S Morcos, Marta Garnelo, Avraham Ruderman, Andrei A Rusu, Ivo Danihelka, Karol Gregor, et al. Neural scene representation and rendering. *Science*, 360(6394):1204–1210, 2018.

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.

Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot visual imitation learning via meta-learning. *arXiv preprint arXiv:1709.04905*, 2017.

Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, SM Eslami, and Yee Whye Teh. Neural processes. *arXiv preprint arXiv:1807.01622*, 2018.

Seyed Kamyar Seyed Ghasemipour, Shixiang Gu, and Richard Zemel. Smile: Scalable meta inverse reinforcement learning through context-conditional policies. In *CORAL*, 2020.

Jonathan Gordon, John Bronskill, Matthias Bauer, Sebastian Nowozin, and Richard E Turner. Meta-learning probabilistic inference for prediction. *arXiv preprint arXiv:1805.09921*, 2018.

Abhishek Gupta, Benjamin Eysenbach, Chelsea Finn, and Sergey Levine. Unsupervised meta-learning for reinforcement learning. *arXiv preprint arXiv:1806.04640*, 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*, 2018.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in neural information processing systems*, pp. 4565–4573, 2016.

Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems*, pp. 1109–1117, 2016.

Kyle Hsu, Sergey Levine, and Chelsea Finn. Unsupervised learning via meta-learning. *ICLR*, 2019.

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

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

J Zico Kolter and Andrew Y Ng. Near-bayesian exploration in polynomial time. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 513–520. ACM, 2009.

Thanard Kurutach, Aviv Tamar, Ge Yang, Stuart J Russell, and Pieter Abbeel. Learning plannable representations with causal infogan. In *Advances in Neural Information Processing Systems*, pp. 8733–8744, 2018.

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.

Ashvin Nair, Dian Chen, Pulkit Agrawal, Phillip Isola, Pieter Abbeel, Jitendra Malik, and Sergey Levine. Combining self-supervised learning and imitation for vision-based rope manipulation. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2146–2153. IEEE, 2017.

Hung Ngo, Matthew Luciw, Alexander Forster, and Juergen Schmidhuber. Learning skills from play: Artificial curiosity on a Katana robot arm. *Proceedings of the International Joint Conference on Neural Networks*, pp. 10–15, 2012. ISSN 2161-4393. doi: 10.1109/IJCNN.2012.6252824.

Boris Oreshkin, Pau Rodríguez López, and Alexandre Lacoste. Tadam: Task dependent adaptive metric for improved few-shot learning. In *Advances in Neural Information Processing Systems*, pp. 721–731, 2018.

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.

Deepak Pathak, Parsa Mahmoudieh, Guanghao Luo, Pulkit Agrawal, Dian Chen, Yide Shentu, Evan Shelhamer, Jitendra Malik, Alexei A Efros, and Trevor Darrell. Zero-shot visual imitation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 2050–2053, 2018.

Kate Rakelly, Aurick Zhou, Deirdre Quillen, Chelsea Finn, and Sergey Levine. Efficient off-policy meta-reinforcement learning via probabilistic context variables. *arXiv preprint arXiv:1903.08254*, 2019.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 627–635, 2011.

J. Schmidhuber. Continual curiosity-driven skill acquisition from high-dimensional video inputs for humanoid robots. *Artificial Intelligence*, 2015.

J. Schmidhuber, J. Zhao, and N. Schraudolph. Reinforcement learning with self-modifying policies. *Learning to learn, Kluwer*, 1997.

Jürgen Schmidhuber. A Possibility for Implementing Curiosity and Boredom in Model-Building Neural Controllers. *Meyer, J.A. and Wilson, S.W. (eds) : From Animals to animats*, pp. 222–227, 1991.

Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pp. 4077–4087, 2017.

Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks. *arXiv preprint arXiv:1804.00645*, 2018.

Jan Storck, Sepp Hochreiter, and Jürgen Schmidhuber. Reinforcement driven information acquisition in non-deterministic environments. *Proceedings of the International . . .*, 2:159–164, 1995.

Valentin Thomas, Jules Pondard, Emmanuel Bengio, Marc Sarfati, Philippe Beaudoin, Marie-Jean Meurs, Joelle Pineau, Doina Precup, and Yoshua Bengio. Independently controllable factors. *arXiv preprint arXiv:1708.01289*, 2017.

Sebastian Thrun. Is learning the n-th thing any easier than learning the first? In *Advances in neural information processing systems*, pp. 640–646, 1996.

Giulia Vezzani, Abhishek Gupta, Lorenzo Natale, and Pieter Abbeel. Learning latent state representation for speeding up exploration. *arXiv preprint arXiv:1905.12621*, 2019.

Ziyu Wang, Josh S Merel, Scott E Reed, Nando de Freitas, Gregory Wayne, and Nicolas Heess. Robust imitation of diverse behaviors. In *Advances in Neural Information Processing Systems*, pp. 5320–5329, 2017.

Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in neural information processing systems*, pp. 2746–2754, 2015.

Kelvin Xu, Ellis Ratner, Anca Dragan, Sergey Levine, and Chelsea Finn. Learning a prior over intent via meta-inverse reinforcement learning. *arXiv preprint arXiv:1805.12573*, 2018a.

Tianbing Xu, Qiang Liu, Liang Zhao, and Jian Peng. Learning to explore via meta-policy gradient. In *International Conference on Machine Learning*, pp. 5459–5468, 2018b.

Tianhao Zhang, Zoe McCarthy, Owen Jow, Dennis Lee, Xi Chen, Ken Goldberg, and Pieter Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–8. IEEE, 2018.

Brian D Ziebart, Andrew Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. 2008.

7 APPENDIX A: ADDITIONAL EXPERIMENTS AND DISCUSSIONS

This section is a continuation of the analysis and discussion in Section 5. It includes discussions that are elucidating, but were nevertheless not crucial enough to appear in the paper's main body.

7.1 WHAT IS THE BEST WAY TO USE THE ENCODER?

Having previously established that the encoder is useful, we now want to study its specific implementation details and the interplay between the encoder and the imitation learning network.

7. Optimally, how should the behavioral cloning network best consume the encoded trajectories? We considered three techniques for incorporating the encoder context z into the behavioral cloning network. First, we simply concatenate it with the state. Second, we run the state through its own MLP and then concatenate this output h with the context z. Finally, we use conditional batch normalization as in Oreshkin et al. (2018) to modulate the behavioral cloning network's weights as a function of the encoder context z. We found that this decision made little appreciable difference in final performance.

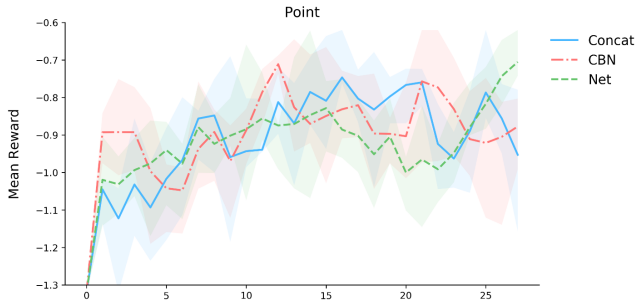


Figure 9: Additional methods of processing the encoder context z into the imitation learning network. CBN is conditional batch normalization, which modulates the imitation network's weights as a function of z. Concat simply concatenates the state with z. Net puts z and s through 2 independent MLP layers, producing outputs h_1 and h_2, which are then concatenated.

8. Do you have to encode the entire trajectory? Can you just encode the last k-states? Yes, you can encode the last k-states of the demonstration trajectory.

The impact of this decision on the final returns depends on the environment. Overall, we found encoding the entire trajectory to be best. Figure 10 looks at the number of encoded frames vs. the final rewards for the Point Chase environment. One interesting possibility we did not consider was sub-sampling some percentage of frames at random, as in temporal dropout (Duan et al., 2017).

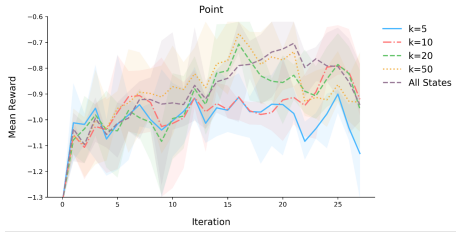


Figure 10: Number of encoded states in each trajectory vs. return.

7.2 KL DIVERGENCE VS EUCLIDEAN DISTANCE

9. Is the use of KL divergnece for the diversity reward important? We found that performance was only slightly worse if, rather than computing $D_{KL}\left(\mathcal{N}(\mu_i, \sigma_i)||\mathcal{N}(\mu_j, \sigma_j)\right)$ in Equation 6, one instead samples $z \sim \mathcal{N}(\mu_j, \sigma_j)$ and computes the loss with euclidean distance as

$$R_i = \sum_j \|z_i - z_j\|_2^2 \tag{9}$$

10. What if you compute the diversity reward in Equation 6 at every time step, rather than the last time step? As discussed in Equation 7, you can compute rewards at every time step, rather than a single reward at the end of each trajectory.

Results for this alternative reward choice are presented in Figure 11. Giving one reward at the end of each trajectory seemed to work best, possibly because the encoder always receives entire trajectories at training time.

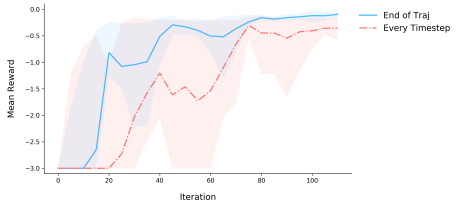


Figure 11: Reacher, rewards every timestep vs end of trajectory

7.3 Failure and Limitations

11. Where does our method fail? Like many imitation learning methods, our algorithm fails at long tasks. Specifically, it fails at tasks that can be segmented into distinct components. It also fails at tasks that require a high amount of precision. For example, our algorithm achieves a 0 % success rate on the task of stacking two blocks.

In contrast, One Shot Imitation Learning succeeded at the difficult task of stacking five blocks, a task with a long horizon that requires a high amount of precision. Part of this has to do with the way One Shot Imitation Learning defined its action space (by discretizing each of the robot's actuator controls into bins and computing a cross entropy loss on each bin), making the problem more tractable. However, part of this also has to do with One Shot Imitation training on actual expert actions, which our algorithm only receives at inference time. Still, it's important to note these deficiencies in our algorithm which are not present in One Shot Imitation.

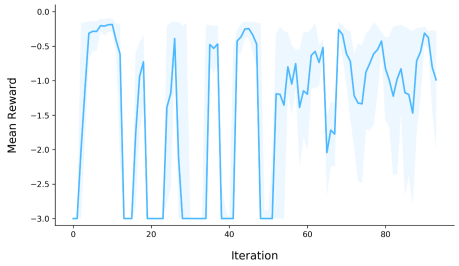


Figure 12: Learning curve for one run of Reacher, with no averaging of seeds. Note the instability.

As for further shortcomings, our method was generally rather unstable. The learning curves in Figure 3 were averaged over 10 seeds. However, if one looks at the individual runs, frequent oscillations are common (this problem is not exclusive to our algorithm, and somewhat symptomatic of RL as a whole). See Figure 12. This deficiency is more or less expected, because the context encoder is being updated with each algorithm iteration without regard to the baseline RL reward.

8 APPENDIX B: HYPER-PARAMETERS

Table 1 lists the common SAC parameters and the architecture parameters for the imitation network and the encoder. Table 2 and Table 3 list the environment specifications and the reward scale parameters that was tuned for each environment.

Table 1: Algorithm Hyper-parameters and Architectures

Parameter	Value
SAC	
optimizer	Adam (Kingma & Ba, 2014)
learning rate	$3 \cdot 10^{-4}$
discount (γ)	0.995
replay buffer size	50000
random initial steps	200
number of hidden layers (all networks)	3
number of hidden units per layer	64
number of samples per minibatch	64
nonlinearity	ReLU
target smoothing coefficient (τ)	0.005
target update interval	1
gradient steps	1
Imitation Network π_θ	
net size	32
number of samples per minibatch	20
number of MLP layers	3
gradient steps	10
nonlinearity	ReLU
Encoder $\mathcal{E}$	
net size	32
number of samples per minibatch	20
number of MLP layers	3
output encoding dimension	16
gradient steps	10
nonlinearity	ReLU

Table 2: Environment Specifications

Environment	Action Dimension	Observation Dimension
Hopper-v3	3	11
Reacher-v2	2	11
Sawyer Reach	4	24
Sawyer Pick	4	24
Point Chase	2	6
Point Multi-Goal	2	12

Table 3: Environment Reward Information

Environment	Maximum Reward	Random Policy Average Reward
Hopper-v3	∞	2.3
Reacher-v2	0.0	-1012.0
Sawyer Reach	1.2	-312.0
Sawyer Pick	3.5	-218.1
Point Chase	0.0	-1.45
Point Multi-Goal	0.0	-1.6