Abstraction-Guided Policy Recovery from Expert Demonstrations

Canmanie T. Ponnambalam
Department of Software Technology
Delft University of Technology
c.t.ponnambalam@tudelft.nl

Frans A. Oliehoek
Department of Intelligent Systems
Delft University of Technology
f.a.oliehoek@tudelft.nl

Matthijs T. J. Spaan
Department of Software Technology
Delft University of Technology
m.t.j.spaan@tudelft.nl

Abstract

The goal in behavior cloning is to extract meaningful information from expert demonstrations and reproduce the same behavior autonomously. However, the available data is unlikely to exhaustively cover the potential problem space. As a result, the quality of automated decision making is compromised without elegant ways to handle the encountering of out-of-distribution states that might occur due to unforeseen events in the environment. Our novel approach RECO uses only the offline data available to recover a behavioral cloning agent from unknown states. Given expert trajectories, RECO learns both an imitation policy and recovery policy. Our contribution is a method for learning this recovery policy that steers the agent back to the trajectories in the data set from unknown states. While there is, per definition, no data available to learn the recovery policy, we exploit abstractions to generalize beyond the available data thus overcoming this problem. In a tabular domain, we show how our method results in drastically fewer calls to a human supervisor without compromising solution quality and with few trajectories provided by an expert. We further introduce a continuous adaptation of RECO and evaluate its potential in an experiment.

1 Introduction

In seeking to automate increasingly complex processes, algorithms that learn from data are an attractive approach to autonomous decision making. Realistically, learning methods often fall short of meeting the requirements of real-world problems. In several domains, particularly safety-critical applications, but also any where decisions require explicit justification, it is not practical to take random actions. This ability to explore is, however, where many online learning algorithms get their power. One way to assuage the limitations of such algorithms is to incorporate demonstrations by an expert. Agents are then trained to copy and deploy the exemplar behavior online with as little supervision required as possible. Particularly in the field of behavioral cloning [2], the agent is expected to directly replicate the demonstrated behavior rather than make insights about the underlying reasoning. We focus on this case, where an expert has provided examples of correct behavior that we seek to automate in the real environment without any exploration or further learning. While the given trajectories are considered optimal, in large state spaces with several state variables it is infeasible for a human to define the correct behavior for every state. Presented with a state far from its expert data set, an agent can either follow its potentially very incorrect model or take random actions, both of which are undesirable in safety-critical applications. An option is to facilitate human

Offline Reinforcement Learning Workshop at Neural Information Processing Systems, 2020.

intervention whenever the agent is lost, but this severely limits the quality of automating the system. In a typical behavior cloning scenario, a supervised learning approach can be used to predict the expert actions in given states [15]. As with any supervised learning method, the ability to generalize over states never seen before is not guaranteed. Further, any model error accumulates over every planning step. These issues severely limit the applicability of behavior cloning in more than the simplest environments. Our method produces a more robust behavior clone that can recover from parts of the state space not covered by its imitation policy. We do this by applying state abstraction to offline data in order to plan a way back to a given behavior policy when in an unknown state.

Consider being in a situation where you are looking for a particular shop that you've been to before but, due to unexpected road work, you've taken a detour from the known route. A reasonable solution would be to first find your way back to a place you know, and then follow your memory to the shop from there. When lost, you substituted the goal of locating the shop with getting back to a familiar path. This key principle can be applied to a planning agent that encounters a state far from the expert trajectories it was given. This is done in our method by applying state abstraction to the given offline data. We compute a coarse model of the problem of planning a path back to a state where the agent can confidently follow its behavior cloning policy. Instead of trying to generalize over the entire decision making task, our agent focuses on the simpler problem of using what it knows to get back on track, thus minimizing the model error. The result is a reduction in calls for human supervision and robustness to unexpected changes that might occur in the environment. The key point here is that, even with a biased data set (containing only successful trajectories) and without making potentially dangerous conclusions about never-before seen states, we are able to recover a lost agent back to a known policy.

This paper introduces RECO, an abstraction-guided policy recovery approach to behavior cloning. We first discuss related methods in the literature. Then, we formally define the policy recovery problem and show how we use data from expert trajectories to specify and solve it. We assess the performance of our method against several baselines in tabular and continuous problems and conclude with suggestions for expanding this work.

2 Related Work

All offline (or *batch*) learning techniques must deal with the issue of what to do when the agent encounters an out-of-distribution state during execution. This issue of *distributional shift* is a well-studied problem in the literature [6, 9, 10]. In behavior cloning or imitation learning, the offline data is produced by an expert and contains only good examples of actions taken in the environment. In our method, we focus on agents that, once deployed, cannot execute random (explorative) actions in the environment nor learn from their online interactions. This distinguishes our problem set up from inverse reinforcement learning, though the rewards are also unknown, as the aim there is to learn a reward structure that produces the observed behavior.

Several authors have explicitly approached the subject of recovering an agent from an out-of-distribution or anomalous state. The question of what to do when in such a situation has had a wide-range of proposed answers, from reverting to a safe policy [14], forcefully resetting the agent [1], or requesting human intervention [11, 7]. In our problem set-up, we assume that a safe policy is not known outside of the expert trajectories provided, that resetting the agent is not possible and that human supervision in the true environment is very costly and therefore undesirable. Though not an imitation learning approach, the work of [5] involves the agent simultaneously learning to perform the task and learning to undo the actions it has done. It this way it learns policies that avoid irreversible states and aims to minimize the need for human intervention.

Others have considered methods that steer the agent towards the expert behavior [8, 16]. In *soft Q imitation learning*, they incentivize a learning agent to take actions that lead back to states known in the expert demonstrations [13]. Similarly to our method, they define a sparse reward function that provides reward for taking demonstrated actions from demonstrated states, training the agent to favor behavior close to the expert. However, they do not isolate the planning problem of returning the agent to known states. In addition to doing this, our application of abstraction to learn this coarse problem representation is a novel contribution to existing work.

3 Background

This section briefly introduces preliminary material that serves as a foundation for our method.

3.1 Markov Decision Processes

A Markov decision process (MDP) [12] is a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$ that describes a sequential decision making problem. The variables $\mathcal{S}$ and $\mathcal{A}$ denote the state and action space, $\mathcal{T}$ and $\mathcal{R}$ the transition and reward functions, and γ is a discounting factor. At each time step t, an agent observes the state of the environment $s_t \in \mathcal{S}$ and chooses an action $a_t \in \mathcal{A}$. Upon executing action a_t, the environment transitions to a new state $s_{t+1} \sim (\cdot|s_t, a_t)$, sampled according to a transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ which maps (s, a) pairs to a distribution over next states. The agent receives a reward r_t according to the reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. The value of a state $V(s)$ is the total discounted reward of the state, given by

$$V(s) = \sum_{t=0}^{\infty} \gamma^t r_t$$

The solution to an MDP is a policy π that maps states to actions. The value of a policy π is the sum of expected discounted rewards over all states. An optimal policy π^* is one that maximizes this value. An MDP can be solved using Value Iteration or Policy Iteration; these are standard approaches that, for reasonably sized problems, produce an optimal policy π^*.

Reinforcement learning concerns solving techniques for MDPs with unknown transition and reward functions. This typically involves executing actions in the environment and learning to both explore randomly and exploit the knowledge already learned regarding rewarding actions. For the uninitiated and interested reader, we refer to [17].

4 RECO: Abstraction-Guided Policy Recovery

Our overall aim is to automate the process of making decisions in an environment given examples of good behaviour provided by an expert. These trajectories of decisions consist of environment observations and corresponding expert actions, but do not cover the entire space of possible observations and actions. Our behavior cloning agent must use this data in order to compute a policy that is then deployed online without any additional data collection or computation. We make no conclusions about the abilities of our behaviour cloning agent in states outside the data set. Thus, we assume that a human supervisor is available to provide an action when the agent finds itself in a state sufficiently far from those included in the expert trajectories. The goal of our method is to reduce the number of calls to the human supervisor using only the data provided by the expert without a substantial loss in performance.

4.1 Problem Formalization

The problem considers making decisions in a system represented as Markov decision process, where the state and action space are known but the transition and reward function are unknown. We assume a factored state space is available, thus the space spanned by the domains of k state variables $\mathcal{S} = \{\mathcal{S}_0 \times ... \times \mathcal{S}_k\}$. Our data set of several trajectories provided by an expert is called $\mathcal{D}$, and is collected according to an unknown policy; we want to copy the behavior of the expert given only these examples. The data set is of the form $\mathcal{D} = (s_1, a_1, s_2, a_2, ..., s_T, a_T)$ containing a finite number T of (state, action) pairs. The set of states found in $\mathcal{D}$ is called the state space $\mathcal{S}_\mathcal{D}$, where $\mathcal{S}_\mathcal{D} \subset \mathcal{S}$. The imitation policy $\pi_\mathcal{D}(s)$ is defined for all $s \in \mathcal{S}_\mathcal{D}$. Our method RECO learns both an imitation policy and a recovery policy from expert trajectories. This latter policy guides the agent back to states covered by its imitation policy from states never visited in the expert demonstrations. The recovery policy uses abstraction to focus on the parts of the state space that are relevant for recovery, thus being able to generalize over unseen states.

We explain this further with an example (pictured in Figure 1). In the classical taxi problem [3], a taxi agent (T) in a grid world must pick up a passenger from a given location (P) and drop them off at their given destination (D). The variables in the factored state description indicate the x and y coordinates of the taxi and locations of the passenger and destination, i.e. $s_0 = (x_0, y_0, p_0, d_0)$. The

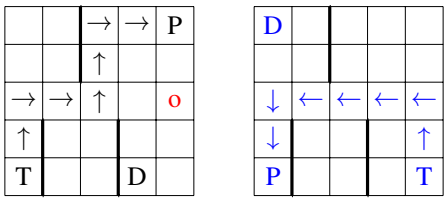

Figure 1: Example of expert trajectories for the classical taxi problem. T indicates the taxi position, P the passenger location and D the destination of the passenger.

arrows in Figure 1 indicate expert behavior for the locations of the taxi, passenger and destination shown. Suppose our agent finds itself at location o, pictured in red, with the passenger and destination located in the black. For this particular state of the passenger and destination, the expert has not provided an example, only the path given by the black arrows. However, another trajectory (blue arrows) for a different configuration of passenger and destination does contain this taxi location. By ignoring irrelevant information, our agent can reason about actions that return it to a known state. A RECO agent will leverage the actions in the blue path, taking them from its position at o and getting back to the known policy (black path) which it then executes. It does this by applying what it knows in the abstract state space (x and y position) to get back to a known state.

4.2 Tabular RECO

In tabular domains, we define our imitation policy $\pi_{\mathcal{D}}$ as simply sampling an action according to a weighted distribution, weighted by the number of times an action was executed by the expert from state s. The main contribution of our method is the definition and use of a recovery policy. We model the recovery problem of getting back to a state in the data set from a state outside of $\mathcal{S}_{\mathcal{D}}$ as an MDP $\mathcal{M}_O$, where:

$$\mathcal{M}_O = (\mathcal{S}_O, \mathcal{A}_O, \mathcal{T}_O, \mathcal{R}_O, \gamma_O)$$

We build the recovery MDP using transitions that occurred in the expert data set, projecting this data onto only the relevant parts of the state space $\mathcal{S}$. We assume there exists an abstraction mapping function μ which, given the state s, returns only the state variables relevant to the orientation task, $\mu(s) = \bar{s}$. The inverse function $\mu'(\bar{s})$ returns the set of ground states that map to the abstract state $\bar{s}$ under the mapping function μ. We keep the same action space as in the original MDP. By applying the function μ to our data set, we get an abstracted data set $\bar{\mathcal{D}} = (\bar{s}_1, a_1, \bar{s}_2, a_2, ..., \bar{s}_T, a_T)$. We call $\mathcal{S}_{\bar{D}}$ the set of ground states that map to the abstract states in $\bar{D}$, retrieved by applying μ' to the states in the expert trajectory. Both $\mathcal{D}$ and $\bar{\mathcal{D}}$ are used to build the MDP $\mathcal{M}_O$. This MDP is defined over the entire state space $\mathcal{S}$ augmented with an additional sink state z. Transitions that are not represented in the expert-provided data set transition to the sink state. In the sink state, all actions transitions back to the sink state with 0 reward. The reward for taking action a from state s is 1 if s is present in $\mathcal{S}_{\bar{D}}$, 0 if the abstract state pair $(\bar{s}, a)$ is present in $\bar{\mathcal{D}}$, and otherwise -1. Thus, for all transitions for which the abstract state has never been visited, there is a negative reward. The transition function contains the transitions observed in the data set $\mathcal{D}$. Below, we define this formally for deterministic transitions. In the case of stochastic transitions, the transition function can be defined by first looking at the frequency of visits to each s' given (s, a), and taking the softmax to produce probabilities that sum to 1.

Our MDP $\mathcal{M}_O$ is then fully specified as:

$$\mathcal{S}_O = \mathcal{S} \cup \{z\}$$
$$\mathcal{A}_O = \mathcal{A}$$

$$\mathcal{R}_O(s,a) = \begin{cases} 1, \text{if } s \in \mathcal{S}_D, \\ 0, \text{else if } (\bar{s}, a) \in \mathcal{S}_{\bar{D}} \vee s = z, \\ -1, \text{otherwise.} \end{cases}$$

$$\mathcal{T}_O(s, a, s') = \begin{cases} 1, \text{if } (\bar{s}, a, \bar{s}') \in \mathcal{S}_{\bar{D}}, \\ 1, \text{if } (\bar{s}, a) \notin \mathcal{S}_{\bar{D}} \wedge s' = z, \\ 1, \text{if } s = s' = z, \\ 0, \text{otherwise.} \end{cases}$$

$$\gamma = [0, 1)$$

The recovery MDP can be solved using an off-the-shelf method such as Value Iteration, producing the recovery policy $\pi_{\mathcal{O}}$. We assume our agent has access to an emergency action *request* that requests an action from an expert. A tabular RECO agent then follows the aggregated policy:

$$\pi_{RECO}(s, \mathcal{D}, \mu) = \begin{cases} \pi_{\mathcal{D}}(s), \text{if } s \in \mathcal{S}_{\mathcal{D}}, \\ \pi_{\mathcal{O}}(s), \text{else if } \bar{s} \in \mathcal{S}_{\bar{\mathcal{D}}}, \\ request, \text{otherwise.} \end{cases}$$

4.3 Continuous RECO

There are several adaptations to be made in order to apply our method to a continuous problem. We first consider the imitation policy. A supervised learning approach can be used to learn this imitation policy, predicting the expert's actions from a given state, trained on the data in $\mathcal{D}$. Again, we call this policy $\pi_{\mathcal{D}}$. As in the tabular version, we assume the existence of an abstraction function $\mu(s)$ that, when applied to the states in our data set, gives us the abstract data set $\bar{\mathcal{D}}$. We gauge whether a state falls within the space defined by our expert trajectory according to a distance measure $d(s_a, s_b)$. Many distance measures are possible, but in this work we use a weighted L2 norm calculation between the states s_a and s_b, summed and weighted over each state dimension.

The transition and reward functions of our continuous recovery problem can no longer be represented in tabular form. We want the transition function to capture the transition in the abstract space. Thus, we train a supervised learning model on $\bar{\mathcal{D}}$ to predict abstract next states from abstract states and actions. We call this predictor $\phi_{\mathcal{T}}$, where $\phi_{\mathcal{T}}(\bar{s}, a) = \bar{s}'$. Our transition function $\mathcal{T}_O$ takes a full state s and transitions only the abstract variables according to $\phi_{\mathcal{T}}$, leaving the other variables unchanged. Thus,

$$\mathcal{T}_O(\phi_{\mathcal{T}}, s, a) = s'.$$

The reward function takes a state and the expert trajectories, and returns a value of 0 or 1 determined by the distance measure and threshold ζ_D. The reward function is defined:

$$\mathcal{R}_O(s, \mathcal{D}) = \begin{cases} 1, \text{if } \exists s^* \in \mathcal{S}_D : d(s, s^*) < \zeta_D, \\ -1, \text{if } s \notin \mathcal{S}, \\ 0, \text{otherwise.} \end{cases}$$

This gives a reward of 1 for entering a state only if the distance between it and the closest state in the expert trajectories is lower than the threshold. A negative reward is given if the agent transitions to an out-of-bounds state. With the transition and reward function defined, we can now simulate the recovery environment. We initialize a random state by sampling a state from the expert trajectories and replacing the irrelevant variables (those ignored by abstraction mapping μ) with a uniformly random value within their bounds. Given an action, the environment transitions according to the transition function, receiving a next state and reward from the reward function. An episode ends if: a maximum number of steps is reached, the environment transitions to an out-of-bounds state, or if the reward function returns a reward of 1. We can solve the recovery problem using a standard continuous state-space reinforcement learning method, obtaining π_O. For the experiments presented in section 5.2, we used a Proximal Policy Optimization variant.

The threshold for distance between the abstract state $\bar{s}$ and the abstracted data set is another parameter ζ_A, different from that regarding the ground data set. A continuous RECO agent acts according to the following policy:

$$\pi_{RECO}(s, \mu, \mathcal{D}, \zeta_D, \zeta_A) = \begin{cases} \pi_{\mathcal{D}}(s), \text{if } \exists s^* \in \mathcal{S}_D : d(s, s^*) < \zeta_D, \\ \pi_{\mathcal{O}}(s), \text{if } \exists s^* \in \mathcal{S}_{\bar{D}} : d(\bar{s}, s^*) < \zeta_A, \\ request, \text{otherwise.} \end{cases}$$

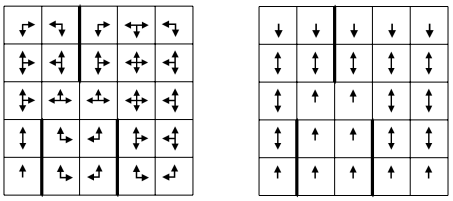

Figure 2: **Recovery potential**: Expert actions in the {row, column} (left) and {row} (right) abstractions (for all possible locations of passenger and destination). The actions are filtered to include only those that result in a change in abstract state.

4.4 Choosing Abstractions

Given a data set $\mathcal{D} = (s_t, a_t, s_{t+1}, a_{t+1}, ..., s_{t+T}, a_{t+T})$, where the state space S is factored into k components, the aim is to define a mapping function $\mu(s)$ that maps a state s to an abstract state $\bar{s}$ made of a subset of its k variables. A natural question arises of how to choose or learn this subset. There might be several abstraction mappings that have the potential to facilitate recovery and we rely here on expert knowledge to choose a suitable subset. There are some general guidelines to consider when doing so. For example, in the taxi domain described above, the destination location is not affected by the actions of the agent; such variables cannot be included in the abstracted subset as the agent has no control over recovering them. If a very small number of expert traces are available, the designer should favour a small abstraction (smaller state space, higher compression) which requires fewer samples to facilitate recovery but in turn has a limited potential to recover. A larger abstraction requires more expert data but also means the recovery agent can recover more states. Figure 2 demonstrates this, showing how an abstraction of {row, col} can (with enough data) traverse the whole map to recover a state whereas using just {row} limits the agent to recovery of states within the same column.

5 Empirical Evaluation

We conducted experiments on a 10x10 extension [4] of the classical taxi problem [3] and a continuous problem where the agent is tasked with picking up an object from a given location in the map. In each trial, all learning was done offline on the same set of given expert trajectories (a trajectory is 1 episode). They were then deployed and evaluated on their performance online in the test environment, initialized to the same random state in each episode.

5.1 Tabular Experiments

The large taxi problem consists of a 10x10 grid and has 8 possible passenger locations (plus 1 for when the passenger is in the taxi) and 4 destination locations (3600 total states). It serves as an example of a factored problem where the agent must pick up and drop off objects according to positions indicated in the state description. Generalizing a policy across unseen states is made more difficult by walls in the environment. Expert trajectories were generated by initializing the agent in a random state and following a trained Q-learning agent greedy policy until episode termination. We compared our RECO agent test performance on 100 episodes against three baselines. The Imitation baseline follows the imitation policy and calls for help in any states outside the expert trajectories given. The Nearest Neighbor baseline uses a crude nearest neighbor approach to choose actions in undefined states, sampling an action from its Cartesian neighbors in the expert trajectories, considering only the relevant subset of state variables. In this way, it uses the same information regarding abstraction as provided to RECO. The Factored Batch RL approach is an offline reinforcement learning method. It uses complete knowledge of the transition dynamics in the form of a dynamic Bayesian network and calculates the parameters of the model with the offline data. This acts as an upper bound of offline learning performance in a factored MDP.

Figure 3 displays the results from the tabular experiment. We ran 25 trials (25 different sets of traces provided by the expert), testing the agent on 100 episodes initialized in a random state. Even with a small number of expert trajectories given, RECO requests help from a human supervisor in less than 10% of episodes, while the Imitation agent calls for help in over 60% of test episodes. As expected,

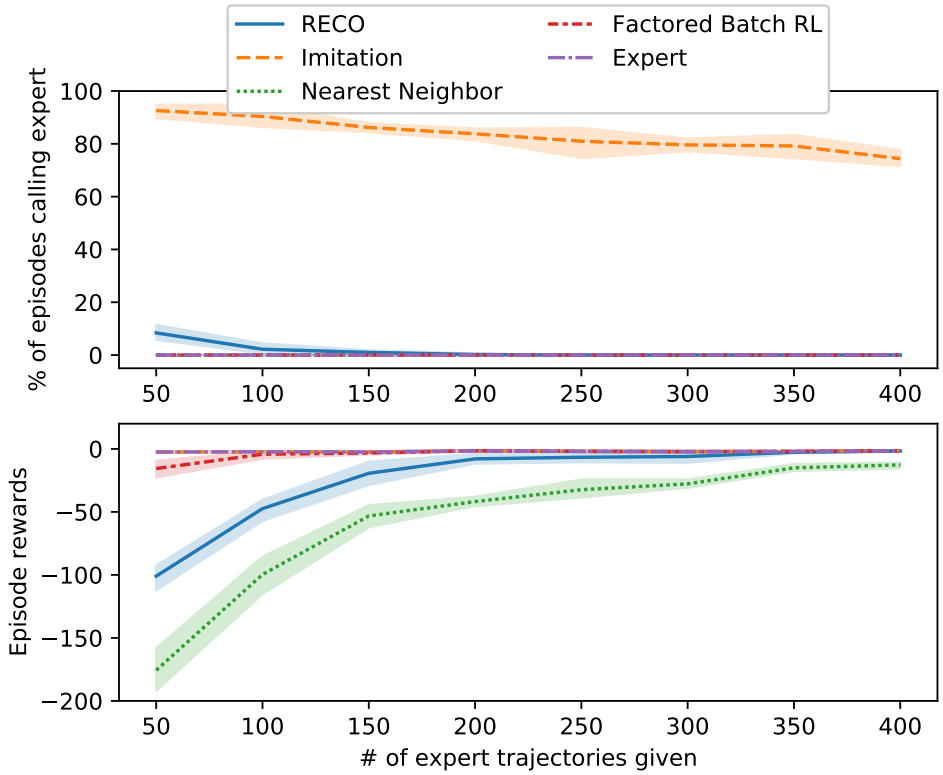

Figure 3: Performance on 100 test episodes of the 10x10 taxi problem averaged over 25 trials of random initial states. 98% confidence interval indicated by shaded regions.

the upper baseline Factored RL, which uses extensive knowledge of the task, has better performance. In the tabular case the impact of RECO is very clear, as without function approximation an agent is stuck when faced with a state outside its expert demonstrations. Our experiments show that the potential for an agent to get lost can be very high without an exhaustive number of traces provided by the expert. Instead, with about 200 traces, RECO performs relatively well and rarely requires the human expert. In many real-life applications, a small performance loss would be acceptable given the substantial savings in required human supervision.

5.2 Continuous Experiment

The continuous world is a simple problem meant to illustrate the potentially poor results when following a naïve imitation policy. An agent is tasked with picking up an object from one of 4 possible locations, again in an environment that includes walls. If the agent performs a pickup action at the correct location, it receives a +100 reward and the episode ends. The agent gets a -1 reward every time step to encourage it to solve the task in as few time steps as possible and incurs a penalty of -25 for illegal pickup actions. The agent gets a large penalty for hitting a wall and a very large penalty (-1000) if the episode ends because a maximum number of time steps was reached. The state description is (agent x coordinate, agent y coordinate, agent x displacement, agent y displacement, goal x coordinate, goal y coordinate). The actions available are (propel north, propel south, propel west, propel east, pickup). A propel action results in a discrete magnitude of acceleration added in the corresponding direction, with acceleration decaying each time step. The “expert" is a reinforcement learning agent trained by a variant of Proximal Policy Optimization on 1e6 time steps of this task. We compare the performance of RECO against two baselines. The Imitation agent follows the imitation policy if it is close enough to expert trajectories, otherwise it calls for help. The Imitation (no expert) follows the imitation policy for any state it encounters regardless of its distance from the expert data set.

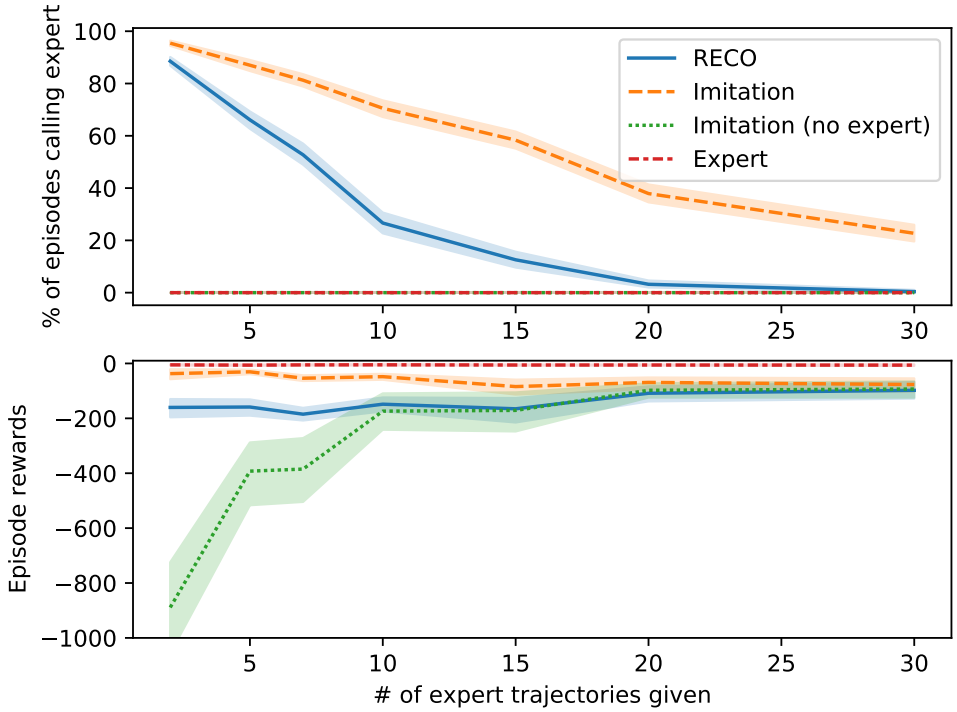

Figure 4: Performance on 100 test episodes of the continuous problem averaged over 30 trials of randomly initialized episodes. 98% confidence interval indicated by shaded regions.

With very little data in this simple problem, the naïve imitation policy fails to grasp the correct behavior of the expert, demonstrating the need for a human supervisor. However, at this point RECO also relies highly on the supervisor. When the Imitation (no expert) agent has enough data to perform adequately, RECO provides little advantage. We do see a reduction in the percent of episodes resulting in a call to the expert, showing that with a better continuous recovery policy there is potential for such a method to leverage information in its data set.

6 Conclusion and Future Work

Our work introduces the notion of applying state abstraction to expert trajectories in order to learn a recovery policy which guides a lost agent back to known states. The aim is to make naïve behavior cloning algorithms more suitable in environments where they would otherwise fail with too little data provided by the expert. We show in experiments that RECO can drastically improve the performance of a behavior cloning agent in tabular domains, requiring few calls to a human supervisor with a relatively small number of expert trajectories given. In continuous domains, we see potential for the RECO agent to make use of abstraction in the data set, however the recovery policy fails to make gains over the baseline. Future work will focus on improving the continuous implementation to better match the performance we see in tabular domains. We would also like to further explore in which particular continuous problems RECO can provide a significant benefit.

Acknowledgments and Disclosure of Funding

This work is part of the research programme Physical Sciences TOP-2 with project number 612.001.602, which is financed by the Dutch Research Council (NWO).

References

[1] Samuel Ainsworth, Matt Barnes, and Siddhartha Srinivasa. 2019. Mo'States Mo'Problems: Emergency Stop Mechanisms from Observation. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc., 15182–15192. https://proceedings.neurips.cc/paper/2019/file/966eaa9527eb956f0dc8788132986707-Paper.pdf

[2] Michael Bain and Claude Sammut. 1996. A Framework for Behavioural Cloning. In *Machine Intelligence 15*. Oxford University Press, 103–129.

[3] Thomas G. Dietterich. 2000. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research* 13, 1 (Nov. 2000), 227–303.

[4] Carlos Diuk, Andre Cohen, and Michael L. Littman. 2008. An Object-Oriented Representation for Efficient Reinforcement Learning. In *Proceedings of the 25th International Conference on Machine Learning* (Helsinki, Finland). Association for Computing Machinery, New York, NY, USA, 240–247. https://doi.org/10.1145/1390156.1390187

[5] Benjamin Eysenbach, Shixiang Gu, Julian Ibarz, and Sergey Levine. 2018. Leave no Trace: Learning to Reset for Safe and Autonomous Reinforcement Learning. In *International Conference on Learning Representations*. https://openreview.net/forum?id=S1vuO-bCW

[6] Scott Fujimoto, David Meger, and Doina Precup. 2019. Off-Policy Deep Reinforcement Learning without Exploration. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, Long Beach, California, USA, 2052–2062. http://proceedings.mlr.press/v97/fujimoto19a.html

[7] Javier García and Fernando Fernández. 2019. Probabilistic Policy Reuse for Safe Reinforcement Learning. *ACM Trans. Auton. Adapt. Syst.* 13, 3, Article 14 (March 2019), 24 pages. https://doi.org/10.1145/3310090

[8] Todd Hester, Matej Vecerík, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Andrew Sendonaris, Gabriel Dulac-Arnold, Ian Osband, John P. Agapiou, Joel Z. Leibo, and Audrunas Gruslys. 2018. Deep Q-learning from Demonstrations. *In Proceedings of the 32nd AAAI Conference on Artificial Intelligence*.

[9] Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. 2019. Stabilizing Off-Policy Q-Learning via Bootstrapping Error Reduction. In *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Vol. 32. Curran Associates, Inc., 11784–11794. https://proceedings.neurips.cc/paper/2019/file/c2073ffa77b5357a498057413bb09d3a-Paper.pdf

[10] Aviral Kumar, Aurick Zhou, G. Tucker, and Sergey Levine. 2020. Conservative Q-Learning for Offline Reinforcement Learning. In *Advances in Neural Information Processing Systems 33*. https://arxiv.org/abs/2006.04779

[11] M. Laskey, S. Staszak, W. Y. Hsieh, J. Mahler, F. T. Pokorny, A. D. Dragan, and K. Goldberg. 2016. SHIV: Reducing supervisor burden in DAgger using support vectors for efficient learning from demonstrations in high dimensional state spaces. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 462–469. https://doi.org/10.1109/ICRA.2016.7487167

[12] Martin L. Puterman. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., Hoboken, NJ, USA.

[13] Siddharth Reddy, Anca D. Dragan, and Sergey Levine. 2020. {SQIL}: Imitation Learning via Reinforcement Learning with Sparse Rewards. In *International Conference on Learning Representations*. https://openreview.net/forum?id=S1xKd24twB

[14] Charles Richter and Nicholas Roy. 2017. Safe Visual Navigation via Deep Learning and Novelty Detection. *Robotics: Science and Systems*. https://doi.org/10.15607/RSS.2017.XIII.064

[15] Stephane Ross and Drew Bagnell. 2010. Efficient Reductions for Imitation Learning. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 9)*, Yee Whye Teh and Mike Titterington (Eds.). JMLR Workshop and Conference Proceedings, Chia Laguna Resort, Sardinia, Italy, 661–668. http://proceedings.mlr.press/v9/ross10a.html

[16] Noah Siegel, Jost Tobias Springenberg, Felix Berkenkamp, Abbas Abdolmaleki, Michael Neunert, Thomas Lampe, Roland Hafner, Nicolas Heess, and Martin Riedmiller. 2020. Keep Doing What Worked: Behavior Modelling Priors for Offline Reinforcement Learning. In *International Conference on Learning Representations*. https://openreview.net/forum?id=rke7geHtwH

[17] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction* (2nd ed.). The MIT Press, Cambridge, MA, USA.