# Reproducibility Assignment: Behavioral Cloning from Observation

Jeroen Zwanepol (4472764), Anish Sridharan (5311209),
Iva Surana (5282896), Srimannarayana (5248566)

***Abstract*— Imitation learning technique's aim is for robots to mimic expert techniques to perform a certain task. One of the main obstacles in imitation learning, is that it is expensive/not always possible to infer the exact actions taken by the expert. An approach to circumvent this bottleneck was proposed by Faraz Torabi, Garrett Warnell and Peter Stone in their paper [1] "Behavioural Cloning from Observation (2018)". This method uses an additional step where the robot explores the environment and trains to infer the action which causes change from one state to another. This is then utilized by the robot to predict the missing actions of the expert demonstrations, and thus learn the expert trajectories. In this work, an attempt is made to reproduce the results of this paper and discuss the algorithm while considering possible alternatives to improve the method. The code for this project is available here.**

## I. INTRODUCTION

Sequential prediction problems are common place in the field of Robotics. The motion of an arm, or the integration of multiple components of a robotic agent are examples of sequential prediction problems. It is not so straight forward to build a controller that solves such complex tasks. In the case of complex sequential prediction problems, imitation learning techniques have proven a useful tool to learn such a controller. One of the main approaches to imitation learning is behaviour cloning (BC) [2], where an agent receives training data on both the states and actions of an expert and tries to replicate this policy. We typically need near-optimal expert demonstrations for different algorithms of imitation learning. However, collecting demonstrations that match the required standard can be time-consuming and expensive. Hence creating a large data set can be extremely difficult, if at all possible. In these cases, behavioral cloning directly might not be the most optimal solution [3]. Further, BC is not robust to variation in the environment [4] and tasks are at times learnt independently; learning one task doesn't speed up the learning process of another task [5]. It is much easier to obtain only the observations (of the states), since these are evident from different sensors. Finding the action taken to transition from one state to the next might require more complex sensors, making this less feasible. In such cases the agent must infer this action. There are two approaches to such Learning from Observations (LfO) tasks; model-based learning and model-free learning. Model-based approaches learns a policy model and requires scalable algorithms while model free learners infer behaviour by learning the cost function through the expert data [6].

Behaviour cloning from observation (BCO) [1] is a model-based approach which trains using only state transitions available in expert demonstrations. BCO consists of several steps in the training process. First, an inverse dynamics model is trained based off of random exploration steps. Here the model learns to predict the action taken to transition from one state to the next. Second, a conventional BC algorithm is used to train the policy of the task at hand. Here the model learns to predict which action is needed given the current state of the system to achieve the given task. The actions of the expert demonstrations are inferred using the previously trained inverse dynamics model. An extension to BCO, called BCO($\alpha$), continues training after these steps to further develop the inverse dynamics model and the policy model.

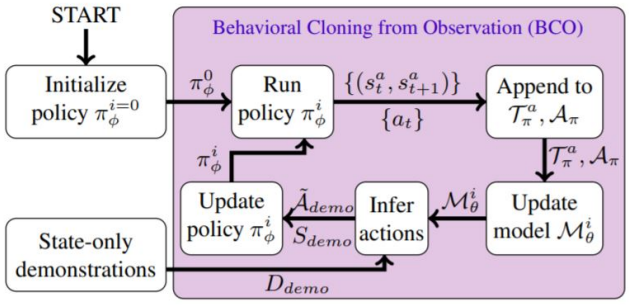



Fig. 1: Behavioral Cloning from Observation (BCO($\alpha$)) framework proposed in this paper

## II. RELATED WORKS

Behaviour Cloning from Observation is a model-based approach to learning from observation, where the agent learns from the inverse dynamics model, i.e., getting the actions for given state-transitions. There are related works in inverse-dynamics model-based approaches similiar to similar to behavioral cloning from observation. Niekum et al looked at an approach where kinematic equations are already known [7]. Nair et al, proposes an algorithm to find the inverse dynamics model and then this is used to infer missing actions of a single demonstration [8]. Behaviour Cloning from Observation, on the other hand does not need any initial knowledge of the system and can also generalize given multiple demonstrations. Model-based approaches, can also be based on solving the forward-dynamics of the system, that is finding the next state of a system, given a state-action pair. as in the case of [9].

Model-free approaches to learn from observation, attempt to

learn the imitation policy without a model learning step [10]. There are two distinct methods to solve this Adversarial methods [11] and Reward Engineering [12]. One of the main advantages of model-based approaches, such as BCO, is that they are sample efficient [13], that is they require less environment interactions when compared to model-free learners. The learned models of a model-based learner can be generalised across tasks unlike model-free techniques [14].

## III. METHOD

BCO consists of two components. First an inverse dynamics model to infer the probability of an action given that an agent transitioned between two states, and second learning an imitation policy from a set of demonstration trajectories. Further, an improvement over this method is proposed as a modified version referred to as BCO($\alpha$).

**Algorithm 1** BCO($\alpha$)

Initialize the model $\mathcal{M}_\theta$ as random approximator
Set $\pi_\phi$ to be a random policy
Set $I = |\mathcal{I}^{pre}|$
**while** policy improvement **do**
  **for** time-step t=1 to $I$ **do**
    Generate samples $(s_t^a, s_{t+1}^a)$ and $a_t$ using $\pi_\phi$
    Append samples $\mathcal{T}_{\pi_\phi}^a \leftarrow (s_t^a, s_{t+1}^a)$, $\mathcal{A}_{\pi_\phi} \leftarrow a_t$
  **end for**
  Improve $\mathcal{M}_\theta$ by modelLearning($\mathcal{T}_{\pi_\phi}^a$, $\mathcal{A}_{\pi_\phi}$)
  Generate set of agent-specific state transitions $\mathcal{T}_{demo}^a$ from the demonstrated state trajectories $D_{demo}$
  Use $\mathcal{M}_\theta$ with $\mathcal{T}_{demo}^a$ to approximate $\tilde{\mathcal{A}}_{demo}$
  Improve $\pi_\phi$ by behavioralCloning($S_{demo}, \tilde{\mathcal{A}}_{demo}$)
  Set $I = \alpha|\mathcal{I}^{pre}|$
**end while**

Fig. 2: Algorithm

Agents act in a framework of Markov decision processes (M) consisting of $M = \{S, A, T, r, \gamma\}$, where $S$ is the state-space of agent, $A$ is the action space, $T_{s_{i+1}}^{s_i a} = P(s_{i+1} \mid s_i, a)$ is the probability of transitioning from $s_i$ to $s_{i+1}$. $r$ is the reward an agent receives after taking a specific action and finally discount factor is represented by $\gamma$. Inverse Dynamics Model (IDM) is modelled as a neural network where the agent performs an exploration policy ($\pi$), to learn actions based on state transitions and these are stored as $\mathcal{I}^{pre}$. These actions and the states of the random exploration policy are used to train the neural network, the states being the input and the actions the output. The initial exploration stage is required by the robot to learn the inverse dynamics model. Since the robot cannot directly learn the actions taken by the expert trajectory, this phase is crucial to the behaviour cloning from observation technique. Based on this information, model learns the inverse dynamics model $\mathcal{M}_\theta$ for the agent. To determine the best parameter $\theta^*$, we use maximum-likelihood estimation, given by:

$$\theta^* = \arg\max_\theta \Pi_{i=0}^{|\mathcal{I}^{pre}|} p_\theta \left(a_i \mid s_i^a, s_{i+1}^a\right) \tag{1}$$

Here $p_\theta$ is the conditional distribution over actions. This is followed by the second step which is behavioral cloning. The robot uses this learnt inverse dynamic model, in the behavior cloning stage where it has to learn from the demonstrations of the expert. In order to predict the actions, the robot uses the inverse dynamic model learnt from step one to move from one step of the expert demonstration to another. We have a set of state-action pairs $\{(s_i, \tilde{a}_i)\}$. Now, we need to determine $\phi$, for which $\pi_\phi$ matches the set of state-action pairs. This is also determined through maximum-likelihood estimation.

$$\phi^* = \arg\max_\phi \Pi_{i=0}^{N} \pi_\phi \left(\tilde{a}_i \mid s_i\right) \tag{2}$$

The above steps form the behaviour cloning from observation algorithm. In the paper, the authors have extended this algorithm further to allow the robot to better learn its inverse dynamics model and the behaviour cloning. In order to implement this, the robot is expected to be allowed some post-demonstration interaction with the environment, where the robot executes imitation policy for a short period of time, then these state-action sequences are used to further update the policy itself. This is repeated until the imitation policy cannot be improved further. This version of the algorithm is called BCO ($\alpha$) in the paper (Ref: 1) , where $\alpha$ is a parameter used to control the post-demonstration interactions. The pseudo code is given in (Ref: 2).

## IV. EXPERIMENTS FOR PAPER REPRODUCTION

### *Selection of Code*

In order to reproduce the results from the paper, ready-to-use implementations were looked at on Github. The author of the paper did not make the source code or the data publicly available, therefore unofficial implementations were considered. Two repositories which are unofficial implementation of the BCO algorithm proposed in the paper were considered to build on. [15] repository had script written for discrete environment using TensorFlow libraries; however, issues raised on the platform about the algorithm implementation have not been addressed so far. [16] repository uses PyTorch library, having implementations of both discrete and continous environments, with no active issues which appeared favourable as a base code for implementing the algorithm.

### *Code Modifications*

However, the script from [16] deviates significantly from the proposed method in [1]. The modifications introduced below attempts towards accurate reproduction of the paper.

*1) Neural Network Architecture:* [1] proposes a linear model for Cartpole while [16] implemented non-linear activation - ReLU. Modified code addressed this issue by considering a linear neural network with a single hidden layer of 32 nodes for both inverse dynamics and control policy networks. Similar fix is carried out for Reacher environment.

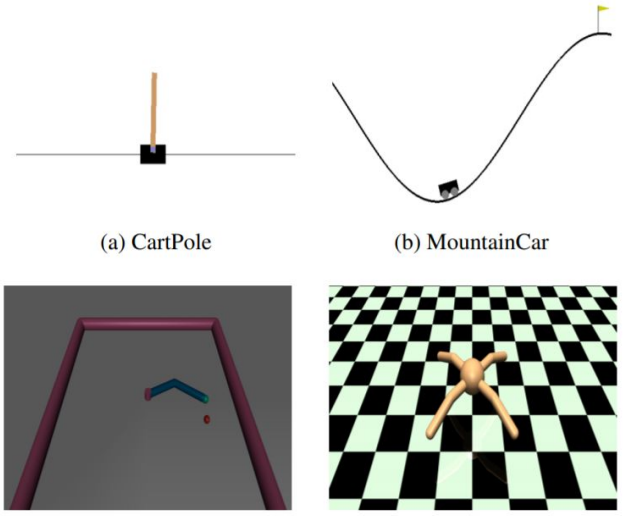
*2) Training of neural networks:* - In [16], the author did not split the data into training and validation sets to run epochs till the policy no longer improved. Rather, batch training was implemented with a single epoch. In the modified code, mini batch training with early stopping is implemented with training and validation split of 70-30 as proposed in [1].

*3) BCO($\alpha$):* - The algorithm from [1] requires that the policy updated in the previous iteration to be used for sample generation subsequently. However, [16] uses epsilon-greedy approach to choose actions for each interaction. In the modified code, random sampling is carried from action space for BCO(0) and the updated policy is used for all iterations that follow. The [16] also did not utilize $\alpha$, to limit the number of post-demonstration interactions.

*Expert Demonstration Data Generation*

Expert data was generated through a pre-trained algorithm from Marcus Lee [17] based on a deep deterministic policy generation (DDPG) for the Ant environment and Aditya Singh [18] based on a deep Q-network (DQN) for the Mountain Car environment. The Reacher data provided with the code from Fu [16] seems inadequate to consider as expert data. Therefore the expert demonstrations available from Jacob [19] were used, this Github page already contained expert trajectories, but it is not specified how this data is generated. 25 expert demonstrations were generated with these models for training the BCO algorithm.

*Extending to Other Environments*



(a) CartPole (b) MountainCar

(c) Reacher (d) Ant

Fig. 3: Representative screenshots of the domains considered in this paper

The script from Fu [16], only attempted implementing the BCO algorithm in the CartPole and Reacher environments. As a part of this reproducibility project, scripts are extended to Mountain Car and Ant environments as well (Fig: 3).

*Parameters and Hyper-Parameters*

Neural Network Architectures and training hyper-parameters used in our attempt to reproduce the paper are documented and provided in tabular form in Appendix : Table 1 and 2. Publishing and disclosing these parameters are critical for independent reproducibility of any machine learning research

## V. RESULTS

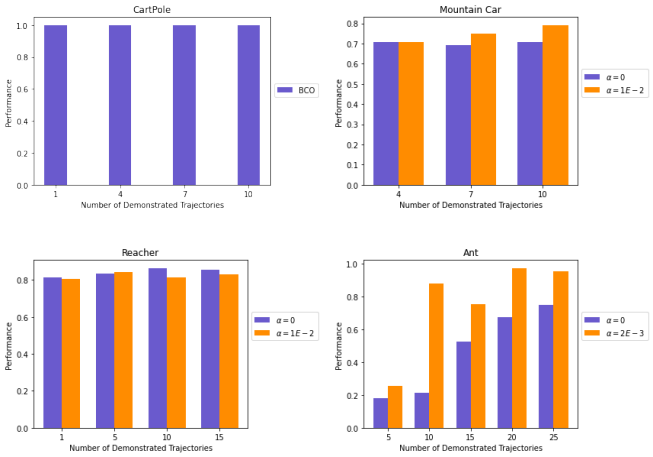



Fig. 4: Performance vs Number of Demonstrated Trajectories

The authors of this paper, attempted a reproduction to validate the claims of [1]. The results can be visualized (Ref Fig: 4) and have been tabulated in Appendix (Tables 3 to 6). It can be observed that the performance claims of BCO and BCO($\alpha$) are reproduced and thereby validated with comparable trends and magnitudes. The main challenge in this reproducibility attempt was that the training hyper-parameters of the neural network were not disclosed by authors of [1]. The cart pole model was the easiest to train and verify the claims, due to this environment being relatively stable to a wide range of hyper-parameters. The Mountain Car, Ant and Reacher models were relatively difficult to train and accurately reproduce the results. The one trajectory results claimed in [1], could not be validated, as the number of expert demonstration transitions was too less for the policy model to converge. The reproduced results show similar performance to those presented in the original paper.

## VI. DISCUSSION AND FUTURE WORK

*Limitations*

Although BCO looks promising as a LfO technique, there are few limitations that should be considered while using the method.

While BCO requires significantly less environment interactions to learn the expert policy compared to other imitation learning methods, it is still unsafe and costly

to carry out random policies in the environment. Model based LfO techniques which use forward dynamics in the algorithm are efficient with regards to the number of interactions needed [9].

The inverse dynamics model does not produce the same state-action pairs as the expert demonstration. The policy is therefore training on imperfect state-action pairs and under-performing when compared to BC in most cases. From Figure 5 in the paper, it can be seen that BC outperforms BCO($\alpha$) in all cases except for the Reacher environment. BCO does not guarantee recovery of the exact expert actions through the inverse dynamics model. A deterministic and injective MDP is needed for this [20]. To verify if this is indeed due to an insufficient inverse dynamics, further research in this area is needed.

When a BCO policy makes a mistake, the observations it receives are different observations than those of the expert trajectories. This will lead to compounding errors [2], [21].

Further, the rewards obtained when completing a task are not taken into consideration for updating the policy. The loss function is based on the difference between expected and obtained values for the actions. The reward obtained when reaching the goal is not used in the training of the algorithm, it learns only on the difference between the output and the expert demonstrations. The use of the rewards would especially be beneficial for the BCO($\alpha$) version, since a lot of post-demonstration interactions take place.

A general drawback of an iterative imitation learning from observations includes the policy being over-fitted in the first iterations. This leads to some actions being ignored due to error in Inverse Dynamics Model during the learning process of the policy model. To solve this problem, Juarez has described a new approach, which uses attention models and a sampling mechanism to regulate the observations that are fed into the inverse dynamics model [22]. This prevents the model from reaching undesirable local minima.

*Behavioral Cloning from Observations with Reinforcement Learning*

Learning any task from scratch without prior knowledge is very tough and with the increase in state space dimensions, the time taken for Reinforcement Learning techniques increases drastically to converge towards the learning goal. On the other hand, imitation learning methods benefit from expert demonstrations available to quickly update a control policy to attain a satisfactory performance, however it cannot attain a policy that can carry out the task better than the demonstration used to train. So, combining both the techniques would be highly effective in optimizing and updating the control policy. However, the challenge here is that both the techniques optimize different loss functions. Multiple works have been carried out to enable this transition such as DQfD, DDPG, DDPGfD [8], [23]–[28] where the approach has been to optimize a new loss which is a combination of both behavior cloning losses and reinforcement losses. A recent work by Vinicius G. Goecks et al. [29] proposes a Cycle-of-Learning (CoL) framework. It uses an actor-critic architecture with a loss function that combines behavior cloning and 1-step Q-learning losses with an off-policy pre-training step from human demonstrations. Considering that BCO can predict corresponding actions of expert demo observations, we can use such a framework to integrate Behavioral Cloning from observation and reinforcement learning.
Booher [30] attempted this combination of behavioral Cloning and reinforcement learning. Combining BC and RL addresses the downsides of each. BC cannot outperform the expert demonstrations given, and RL struggles with sparse loss functions for long horizon tasks. Therefore Booher proposes a combined loss function where BC is favored to initialize a good policy, and later RL is favored to exceed the performance of the expert:

$$\theta_t L_{BC} + (1 - \theta_t) L_{RL} \tag{3}$$

$$\theta_t = \begin{cases} 1, & t < \alpha \\ exp(-\frac{t}{\beta}), & \alpha \leq t \leq \phi \\ 0, & \phi < t \end{cases}$$

The $L_{BC}$ used in this function is essentially the same as the loss function of BCO, since this uses a BC model to train its policy.

## VII. CONCLUSION

From the results obtained it can be seen that the claims made by the paper hold in this independent reproducibility attempt. The plots obtained by the reproduction of the paper correspond to a satisfying degree with the plots of the paper itself. There are small differences, especially in the plots for the continuous environments, but this is most likely due to the lack of information provided on the training of the neural networks.

There are some limitations which come with BCO and BCO($\alpha$) as discussed in the previous section. Nevertheless, it can be seen that BCO provides an effective controller for complex sequential prediction problems. BCO($\alpha$) provides a controller that obtains rewards close to those of the expert for all four environments tested. It does so with less environmental interactions when compared with other state-of-the-art algorithms such as GAIL, FEM and BC.

## REFERENCES


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

[2] S. Ross, G. Gordon, and D. 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*. JMLR Workshop and Conference Proceedings, 2011, pp. 627–635.

[3] X. Guo, S. Chang, M. Yu, G. Tesauro, and M. Campbell, "Hybrid reinforcement learning with expert state sequences," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 3739–3746.

[4] E. F. Morales and C. Sammut, "Learning to fly by combining reinforcement learning with behavioural cloning," in *Proceedings of the twenty-first international conference on Machine learning*, 2004, p. 76.

[5] S. James, A. J. Davison, and E. Johns, "Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task," in *Conference on Robot Learning*. PMLR, 2017, pp. 334–343.

[6] H. Geffner, "Model-free, Model-based, and General Intelligence," Tech. Rep., 2017.

[7] S. Niekum, S. Osentoski, G. Konidaris, S. Chitta, B. Marthi, and A. G. Barto, "Learning Grounded Finite-State Representations from Unstructured Demonstrations," Tech. Rep.

[8] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Overcoming Exploration in Reinforcement Learning with Demonstrations," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 6292–6299, 9 2017. [Online]. Available: http://arxiv.org/abs/1709.10089

[9] A. Edwards, H. Sahni, Y. Schroecker, and C. Isbell, "Imitating latent policies from observation," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 1755–1763. [Online]. Available: http://proceedings.mlr.press/v97/edwards19a.html

[10] F. Torabi, G. Warnell, and P. Stone, "Recent advances in imitation learning from observation," *arXiv preprint arXiv:1905.13566*, 2019.

[11] J. Merel, Y. Tassa, D. TB, S. Srinivasan, J. Lemmon, Z. Wang, G. Wayne, and N. Heess, "Learning human behaviors from motion capture by adversarial imitation," *CoRR*, vol. abs/1707.02201, 2017. [Online]. Available: http://arxiv.org/abs/1707.02201

[12] P. Sermanet, C. Lynch, J. Hsu, and S. Levine, "Time-contrastive networks: Self-supervised learning from multi-view observation," *CoRR*, vol. abs/1704.06888, 2017. [Online]. Available: http://arxiv.org/abs/1704.06888

[13] Y. Chebotar, K. Hausman, M. Zhang, G. Sukhatme, S. Schaal, and S. Levine, "Combining model-based and model-free updates for trajectory-centric reinforcement learning," in *International conference on machine learning*. PMLR, 2017, pp. 703–711.

[14] M. E. Taylor, N. K. Jong, and P. Stone, "Transferring instances for model-based reinforcement learning," in *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 2008, pp. 488–505.

[15] "GitHub - jerrylin1121/BCO: Implementation of Behavioral Cloning from Observationmentation." [Online]. Available: https://github.com/jerrylin1121/BCO

[16] "GitHub - tsujuifu/pytorch_bco: A PyTorch implementation of BCO." [Online]. Available: https://github.com/tsujuifu/pytorch_bco

[17] "GitHub - kangtinglee/reinforcement-learning: reinforcement learning algorithms implemented in pytorch." [Online]. Available: https://github.com/kangtinglee/reinforcement-learning

[18] "GitHub - adibyte95/Mountain_car-OpenAI-GYM: solution to mountain car problem of OpenAI Gym." [Online]. Available: https://github.com/adibyte95/Mountain_car-OpenAI-GYM

[19] "GitHub - JacobXPX/Imitation_Learning : Imitation learning for Reacher." [Online]. Available: https://github.com/JacobXPX/Imitation_Learning

[20] Z. Zhu, K. Lin, B. Dai, G. Research, and J. Zhou, "Off-Policy Imitation Learning from Observations," Tech. Rep.

[21] V. G. Goecks, G. M. Gremillion, V. J. Lawhern, J. Valasek, and N. R. Waytowich, "Integrating behavior cloning and reinforcement learning for improved performance in dense and sparse reward environments," *arXiv preprint arXiv:1910.04281*, 2019.

[22] J. Monteiro, N. Gavenski, R. Granada, F. Meneguzzi, and R. Barros, "Augmented behavioral cloning from observation," *arXiv preprint arXiv:2004.13529*, 2020.

[23] 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," in *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, 9 2016. [Online]. Available: https://goo.gl/J4PIAz

[24] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, G. Dulac-Arnold, I. Osband, J. Agapiou, J. Z. Leibo, and A. Gruslys, "Deep Q-learning from Demonstrations," *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pp. 3223–3230, 4 2017. [Online]. Available: http://arxiv.org/abs/1704.03732

[25] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2 2015. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/25719670/

[26] B. Kim, A.-M. Farahmand, J. Pineau, and D. Precup, "Learning from Limited Demonstrations," Tech. Rep., 2013.

[27] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller, "Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards," *arXiv*, 7 2017. [Online]. Available: http://arxiv.org/abs/1707.08817

[28] C. G. Atkeson, C. G. Atkeson, and S. Schaal, "Robot Learning From Demonstration," 1997. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.54.8531

[29] V. G. Goecks, G. M. Gremillion, V. J. Lawhern, J. Valasek, and N. R. Waytowich, "Integrating behavior cloning and rein-forcement learning for improved performance in dense and sparse reward environments." [Online]. Available: www.ifaamas.org

[30] J. Booher, "Bc+ rl: Imitation learning from non-optimal demonstrations."

# APPENDIX

## Parameters and Hyperparameters

| Discrete Models | | |
|---|---|---|
| **Parameters and Hyperparameters** | **CartPole** | **Mountain Car** |
| **NN: ID** | **Linear Model<br>1 Hidden Layer (32 Nodes)<br>No Activation Function** | **2 Hidden Layers (8 Nodes)<br>Activation Function-ReLU** |
| **NN: Policy** | **Linear Model<br>1 Hidden Layer (32 Nodes)<br>No Activation Function** | **2 Hidden Layers (8 Nodes)<br>Activation Function-ReLU** |
| **Transitions per Demo** | **100** | **5** |
| **Loss Function** | **Cross Entropy** | **Cross Entropy** |
| **Learning Rate (BCO) ID Model** | **5e-3** | **5e-4** |
| **Learning Rate (BCO) Policy Model** | **5e-3** | **4 Trajectories: 8e-3<br>7 and 10 Trajectories: 8e-4** |
| **Learning Rate (BCO-$\alpha$) ID Model** | – | **First Iteration: 5e-4<br>Alpha Iteration: 5e-5** |
| **Learning Rate (BCO-$\alpha$) Policy Model** | – | **First Iteration: 8e-4<br>Alpha Iteration: 8e-5** |
| **Batch Size for Training ID and Policy** | **32** | **Training ID: 8<br>Policy: 5** |
| **Early Stopping Patience Counter for NN Training** | **5** | **250** |
| **BCO ($\alpha$) Patience Counter** | – | **10** |
| **Optimizer** | **Adam** | **Adam** |

Table 1: Parameters and Hyperparameters for Discrete Models

| Continuous Models | | |
|---|---|---|
| **Parameters and Hyperparameters** | **Reacher** | **Ant** |
| **NN: ID** | **MLP<br>2 Hidden Layers<br>(100 Nodes each)<br>Activation Fucntion-LReLU** | **MLP<br>2 Hidden Layers<br>(100 Nodes each)<br>Activation Function-LReLU** |
| **NN: Policy** | **MLP<br>2 Hidden Layers<br>(32 Nodes each)<br>Activation Function-LReLU** | **MLP<br>2 Hidden Layers<br>(32 Nodes each)<br>Activation Function-LReLU** |
| **Transitions per Demo** | **50** | **100** |
| **Loss Function** | **Cross Entropy** | **MSE** |
| **Learning Rate (BCO)** | **5e-3** | **5e-4** |
| **Learning Rate (BCO-$\alpha$)** | **First Iteration: 5e-3<br>Alpha Iteration: 5e-5** | **First Iteration: 5e-4<br>Alpha Iteration: 5e-5** |
| **Batch Size for Training ID and Policy** | **5** | **32** |
| **Early Stopping Patience Counter for NN Training** | **100** | **100** |
| **BCO $\alpha$ Patience Counter** | **50** | **50** |
| **Optimzer** | **Adam** | **Adam** |

Table 2: Parameters and Hyperparameters for Continuous Models

## Results

| Rewards (10 Demo Traj) | | |
|---|---|---|
| **Env** | **Random** | **Expert** |
| **CartPole** | **200** | **20.5** |
| **MountainCar** | **-128** | **-200** |
| **Reacher** | **-4.134** | **-43** |
| **Ant** | **4508** | **-54** |

Table 3: Expert and Random AWR

| Mountain Car - Rewards (Mean over 5000 Episodes)<br>Demos - 5 Transitions per Expert Trajectory | | | | |
|---|---|---|---|---|
| **Env** | **1 Traj** | **4 Traj** | **7 Traj** | **10 Traj** |
| **BCO(0)**<br>**(LR=8E-4)** | – | **-149 (71%)** | **-150 (69%)** | **-149 (71%)** |
| **BCO(1E − 2)**<br>**(LR=8E-5)** | – | **-149 (71%)** | **-146 (75%)** | **-143 (79%)** |

Table 4: Mountain Car

| Reacher - Rewards (Mean over 5000 Episodes)<br>Demos - 50 Transitions per Expert Trajectory | | | | |
|---|---|---|---|---|
| **Env** | **1 Traj** | **5 Traj** | **10 Traj** | **15 Traj** |
| **BCO(0)**<br>**(LR = 5e-3)** | **-11.43 (81.23%)** | **-10.62 (83.3%)** | **-9.41 (86.4%)** | **-9.74 (85.5%)** |
| **BCO(1E − 2)**<br>**(LR = 5e-5)** | **-11.72 (80.5%)** | **-10.16 (84.5%)** | **-11.34 (81.5%)** | **-10.69 (83%)** |

Table 5: Reacher

| Ant - Rewards (Mean over 1000 Episodes)<br>Demos - 50 Transitions per Expert Trajectory | | | | | |
|---|---|---|---|---|---|
| **Env** | **5 Traj** | **10 Traj** | **15 Traj** | **20 Traj** | **25 Traj** |
| **BCO(0)**<br>**(LR = 5e-4)** | **775 (17%)** | **921 (20%)** | **2351 (52%)** | **3020 (67%)** | **3367 (75%)** |
| **BCO(2E − 3)**<br>**(LR = 5e-5)** | **1114 (20%)** | **3956 (87%)** | **3376 (75%)** | **4385 (97%)** | **4290 (95%)** |

Table 6: Ant