Appendix : Batch IRL to Extract Goals in ICU Hypotension Management

Srivatsan Srinivasan, M.E. [1], **Finale Doshi-Velez, PhD**[1]
[1] **Harvard University, Paulson School of Engineering and Applied Sciences, Cambridge, MA**

A1. Hypotension Management in the ICU : MDP Formulation and Supervised Learning Warm-Start

Construction of MDP

As decisions in ICU have long-term temporal dependencies and we have an ability to obtain feedback from the environment for our actions in terms of change in patient vitals and lab values, we can construct an MDP out of this time-series data. Any MDP requires specifying a state space and an action space. We define the state space to be the concatenation of all the static and time-varying features that we described in our ICU patient cohort section in the main text. That is, the state s_t^k for episode k at time t is $[z^k, x_t^k, o_t^k]$. We fully acknowledge that this representation—which does not consider previous history—is an imperfect statistic (and future work should consider various history encodings), but what it does allow is for us to learn an interpretable state-only reward $g(s)$ that is a function of only where the patient features currently are; in this sense, the state definition is consistent with clinicians seeing a current set of measures and determining whether that setting is more or less desirable than another.

Remember that we are focusing only on vasopressor and fluids as interventions in this work. We discretize the action space similar to[1,2]. We create 5 bins each for the dosage values of vasopressors and fluids, with bin 0 representing no dosage and the remaining 4 bins created out of quartile values of each dosage learned from the data. For the purposes of this MDP, the set of actions $a \in [\{0, 1\}]^{25}$, accounts for every possible combination of 5 bins of both interventions. Finally, we choose high $\gamma = 0.99$ since we expect to have long-term temporal dependencies. Since long-horizon planning is hard, We also truncated the trajectory length of patients to 200 i.e. for our cohort of patients, we consider planning over only the first 100 hours of their stay in the ICU.

Initial warm-start Policy via Supervised Learning

Recall from the methods section in the main text that in the fully-batch setting, it is essential to begin the learning process in the support of the already-collected data. To do so, we first train a supervised classifier to predict the expert action (discretized as above) given the state features. This expert-imitation task is in itself non-trivial because the expert actions are highly imbalanced, i.e. $\sim 82\%$ of the times, no action (action 0) is recommended in the dataset while the extreme actions are very rare (for instance, action 24—bin 5 of vasopressor and bin 5 of fluids occur less than 0.2% of the time; see Figure 1 to see the distribution and the imbalance of actions in our data).

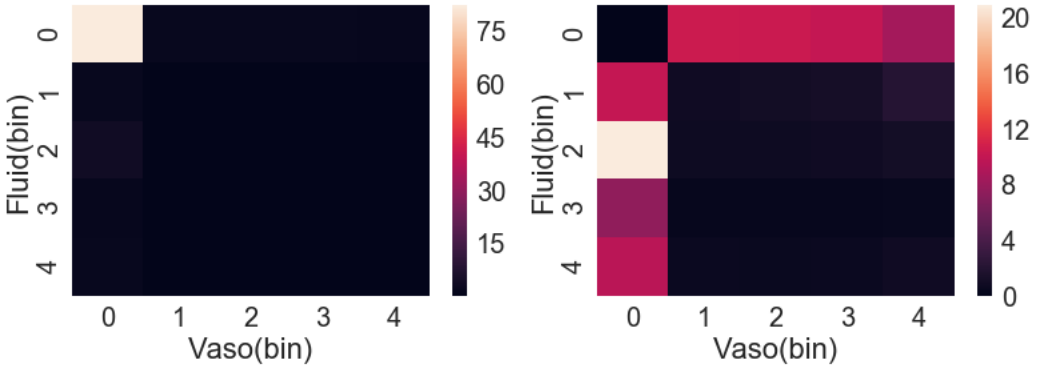

Figure 1: Action Distribution in our dataset expressed as percentage. (Left) Actual action distribution. (Right) Action Distribution removing all the points where no fluids or vasopressors were provided to the patient (action 0)

To account for this imbalance, we first learn a binary classifier that learns to predict whether action 0 would take place from a given state. Conditional on this model predicting a non-zero action, we learn each intervention's action

bin sequentially using a supervised classifier—fluids followed by vasopressors—using the former's prediction as an additional feature for the latter. In total the three models could be mathematically represented as

$$\begin{aligned}
&\text{M1} : \rho(s) :\rightarrow P(a = 0|s) \\
&\text{M2} : \phi_{fluid}(s) :\rightarrow P(a_{fluid} = \{0, 1, 2, 3, 4\}|s) \\
&\text{M3} : \phi_{vaso}(s, a_{fluid}) :\rightarrow P(a_{vaso} = \{0, 1, 2, 3, 4\}|s, a_{vaso} \sim \phi_{fluid}(s)) \\
&\text{Overall Model} : \pi(a|s) :\rightarrow P(a = \{0, 1..24\}|s) \\
&\qquad = \rho(s) * \delta(a = 0) + (1 - \rho(s)) * (5 * a_{fluid} \sim \phi_{fluid} + a_{vaso} \sim \phi_{vaso})
\end{aligned}$$

Note that a_{fluid} is multiplied by 5 to convert two sets of 5 bins into one single set of 25 bins and is in no way connected to the parametric form of the model. An initial policy learned from this step-wise supervised classifier is used to warm-start both the IRL variants in our ICU experiments.

A2. Results on Synthetic Experiments

GridWorld

We first consider a 9×9 grid world in which experts demonstrate trajectories that travel from the bottom left to the top right of the grid via the center. Figure 2 summarizes our interpretable AIRL model iAIRL's results on this experiment. The true reward structure can be seen in Figure 2 (left). We chose this form of the rewards because the axis-aligned reward boundaries are similar to the kinds of rewards we expect to see in the clinical settings (for e.g. BP > 70, urine output > 50 - good rewards, BP < 60, urine output < 50 - bad rewards). Figure 2 (right) show the rewards and the feature bins (that support reward decisions) discovered by our iAIRL algorithm. We see that we recover intuitively appropriate boundaries and the relative rewards across the bins are consistent with the ground truth. At the same time, the model learns a spurious reward structure as it places very small weights (rewards) on the "unnecessary" regions within the environment.

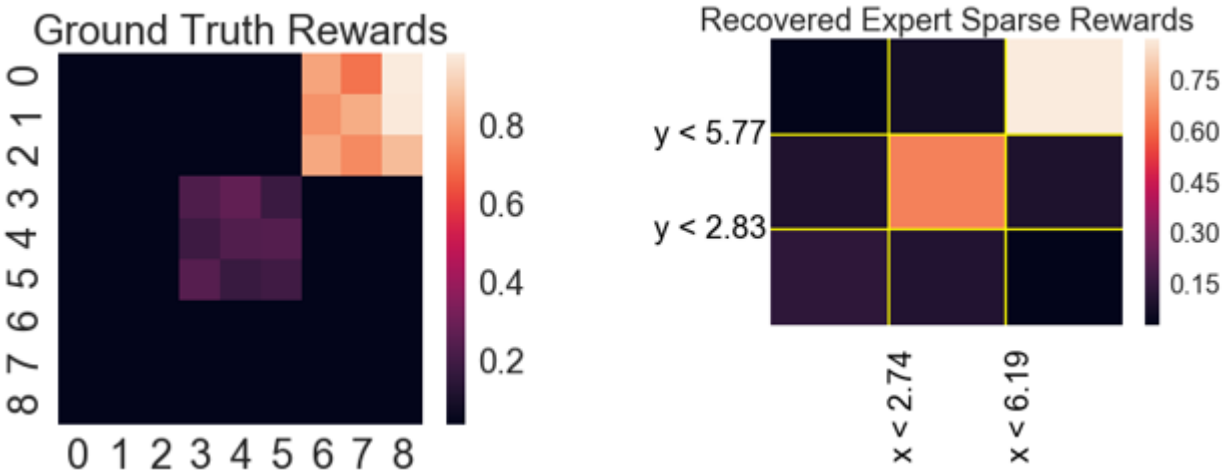

Figure 2: (Left) : Chosen ground truth reward of the 9×9 GridWorld setting with a goal o reach from the bottom left to the top right.(Right) : Rewards recovered by our batch-iAIRL model (3 bins along each dimension, 9 discrete rewards of which 7 are less significant) - The agent learns to value the center and top right region more as intended. Note that the boundaries are learned on a 1-9 row/column numbering for iAIRL. For e.g. $x < 2.74$ means that it learns a boundary between the second and third columns (the ground truth reward had a significant jump after the third column.)

Model	Accumulated Rewards
Expert (Data-Generating)	-100
TRIL+DSFN	-103 $\pm$ 4
LSTD-mu	-121 $\pm$ 6
batch-AIRL (**Ours**)	-105 $\pm$ 3
iAIRL (**Ours**)	-112 $\pm$ 4

Table 1: MountainCar-v0 Results : Rewards accumulated with 1000 episodes of batch data. Higher the actual value, the better e.g. -100 is better than -120. Remember in iAIRL, we are losing some rewards (performance) for the sake of learning more interpretable sparse and discrete rewards.

MountainCar : Description and Results

Since the GridWorld experiment outlined earlier had a simple state space and environment dynamics, we also experiment with Mountaincar-v0 [1] environment as part of OpenAI-Gym to test the performance of our model on a continuous state space and harder environment dynamics. In this environment, the car agent starts from a valley and is expected to learn to navigate on a one-dimensional track between two "mountains" (Check figure in the Synthetic Examples section from the main text). The goal is to drive up the mountain on the right; However this cannot be accomplished in a single pass driving towards right because the car engine is not powerful enough to overcome the mountain's gravity in a single pass and hence the only way to succeed is to drive back and forth to build up momentum. Here, the state is represented by the position and velocity of the agent and the discrete actions include going right, left or do nothing. Table 1 positions the results of our models compared to other standard batch setting IRL algorithms. We see that our models do comparatively well while iAIRL loses little in terms of performance while producing highly interpretable sparse and discrete rewards as seen in the main text.

A3. Technical Background

Technical Background

Notations and Definitions We begin by defining core notation and terminology. We assume access to patient histories τ of states and actions: $\tau = \{s_0, a_0, ...s_T, a_T\}$. In the clinical context, the state s may be some statistics (vitals, labs, attributes etc.) of the patient while the action a would be the intervention. The collection of such trajectories constitute our batch data which we denote by $\mathcal{D}$. Let $T(s'|s,a)$ be the (usually unknown) transition function that gives a distribution over next states given a current state and action, and let $R(s,a,s')$ be the reward for taking a certain action in a certain state and ending up in a certain next-state. Let the policy $\pi(s,a)$ be a decision-making strategy (gives the probability of doing action a in state s). The *value* V, the *action-value* Q and *advantage* A under a policy π are defined as:

$$V(s_0;\pi) = \sum_{t=0}^{\infty} E_{T(s_t,a_t\sim\pi,s_{t+1})}\gamma^t R(s_t,a_t,s_{t+1})$$

$$Q(s,a;\pi) = E_{T(s,a,s')}(R(s,a,s') + \gamma * V(s';\pi))$$

$$A(s,a;\pi) = Q(s,a;\pi) - V(s;\pi)$$

The optimal policy π^* (correspondingly optimal value V^*, optimal advantage A^* can be derived) is defined as the one that maximizes V, that is $\arg\max_\pi E_{s_0}[V(s_0,\pi)]$. Policy optimization algorithms such as Q-Learning (We use deep Q-Learning ([3]) in our work) aim to find this π^* based on the transitions (s,a,r,s') that an agent observes either from interacting in the environment or from the batch data that has been already collected from the expert demonstrations of the task.

[1] Consult these links for the exact details of the environment - https://gym.openai.com/envs/MountainCar-v0/, https://github.com/openai/gym/wiki/MountainCar-v0

Inverse Reinforcement Learning (IRL) and Adversarial IRL Rather than optimizing actions given a reward function, the problem of inverse reinforcement learning seeks to infer the reward function $R(s, a, s')$ given a set of demonstrations $\mathcal{D} = \tau_1, ..., \tau_N$. We assume that the demonstrations are drawn from an optimal expert policy $\pi^e(a|s)$, that is, the expert is acting optimally to *optimize* some (hidden to us) notion of rewards $R(s, a, s')$.

Adversarial Training and its relevance to IRL Recently, Generative Adversarial Networks have been a very popular generative model (GAN,[4]) that learns an implicit distribution of the data samples. GANs typically consist of a generator and discriminator, two neural networks which are trained in a min-max framework - the discriminator D aims to distinguish true data samples from the ones generated by the generator while the generator G learns to generate synthetic samples that are good enough to maximize the uncertainty of the discriminator in distinguishing between the true and the synthetic samples. As a parallel in IRL, one can think of an MDP characterized by the learned reward function $R(s, a, s')$ as a generator which generates trajectories (s, a, s') when the policy is rolled out. A discriminator should learn to differentiate the samples seen in the true batch data from the ones generated by the MDP policy (that maximizes the learned IRL rewards) while the goal of the reward function training should be to allow the MDP to generate those policies that produce highly similar transition samples as the ones seen from the expert batch data. Ideally, if IRL recovers the true expert rewards, then the MDP optimal policy based samples (induced by the learned rewards) and the samples from the true batch data will be indistinguishable to the discriminator.

[5] setup the IRL optimization problem in a GAN type framework (Refer[6] and[5] for theoretical justifications) where the discriminator takes on a very specific form.

$$
\begin{aligned}
D_{\theta,\phi}(s, a, s') &= \frac{\exp f_{\theta,\phi}(s, a, s')}{\exp f_{\theta,\phi}(s, a, s') + \pi(a|s)} \text{ (Binary Classification of MDP vs. IRL samples)} \\
f_{\theta,\phi(s,a,s')} &= \underbrace{g_\theta(s, a)}_{\substack{\text{True Reward} \\ \text{approximator}}} + \underbrace{\gamma h_\phi(s') - h_\phi(s)}_{\text{Shaping term}} \\
R^{shaped}_{\theta,\phi}(s, a, s') &= \log(D_{\theta,\phi}(s, a, s')) - \log(1 - D_{\theta,\phi}(s, a, s'))
\end{aligned}
\tag{1}
$$

This specific kind of formulation of $f_{\theta,\phi}$ provides the freedom to parametrize the reward approximator $g_\theta(s, a) := g_\theta(s)$ solely as a function of state (state-only rewards) i.e. $R(s, a) = R(s)$ while letting the additional shaping term help mitigate the effects of unwanted shaping on the reward approximator due to the dynamics of the environment. In cases where the true underlying rewards are assumed to be a function of state, i.e. $R^*(s, a, s') = R^*(s)$,[5] also prove that g_θ is able to recover the state-only rewards of the original MDP that generated the demonstrations (R^*) if the environment has deterministic dynamics satisfying the decomposability conditions (assuming that all models are trained to optimality)

$$
\begin{aligned}
g^*(s) &= R^*(s) + c \text{ (Expert Rewards)}, \quad h^*(s) = V^*(s) + c \\
f^*(s, a, s') &= R^*(s) + \gamma V^*(s') - V^*(s) = Q^*(s, a) - V^*(s) = A^*(s)
\end{aligned}
\tag{2}
$$

A4. Discussion on the technical aspects of implementing iAIRL

Our success in recovering interpretable reward structures in a fully-batch settings relied on many extensions and innovations: AIRL, DNDT, nonparametric transition model, choices of MDP solver, and warm-starting near the data. For practitioners looking to apply these components to other batch settings, below we list key engineering efforts, constraints, and suggestions for future work.

For stable training of DQNs, we found that clipping/rescaling the rewards provided by the IRL to [-1,1] for all observations model prevented explosion of Q-values and consequent instability in optimization. Since we are operating in batch settings and the transition model is expected to be performant only in regions significant representation in the batch data, we need to ensure that the update to a policy does not change it significantly from the current policy. We

managed to achieve this by a simple trick of setting $\pi^{new} = \alpha * \pi^{old} + (1 - \alpha) * \hat{\pi}$ where $\hat{\pi}$ is the optimal policy provided by DQN and α, a hyperparameter controls the change in policy. Future work could also focus on more principled policy gradient based MDP optimization methods that ensure gradual policy updates such as Trust-Region Policy Optimization ([7]).

We found our DNDT to be highly sensitive towards the initialization of decision bins. If we initialize decision bins far off from the ground truth, we found that the model focuses predominantly on adjusting the only the decision layer weights while the learning of decision bins gets stuck in some undesirable local optima. In all our experiments, we had some intuition about reasonable decision bins and we were able to initialize the model with those values effectively. Future work should focus on addressing this sensitivity by engineering better decision architectures as domain expertise to choose these boundaries might not always be available along with batch data. Future work in DNDT should also address the scalability of DNDT to a growing number of features. This would involve exploring ways to encourage ensemble decision models using subsets of features or modifying the current architecture/training to incentivize smart pruning of decision boundaries before reaching the last layer. We also found that the choice of the number of decision bins for each feature was not a major concern during training. As noted by[8], choosing more bins than necessary often ended with the model learning some trivial decision boundaries outside the support of the training data.

Initialization of the IRL policy is another pivotal step that is not as significant in non-batch settings. While warm-starting with a near-expert initial policy leads to faster convergence in both batch and non-batch settings, the lack of ability to collect additional data means that our IRL is allowed to explore only those policies in the vicinity of our batch data's support since our transition models are expected to be good only in that regime. For example in the ICU data, an initial policy which had an overall action matching of about 53% led to very unstable IRL training whose final action matching performance was bad - close to 11% whereas when the initial policy had an action matching of 74%, the IRL training was reasonably stable and resulted in an overall action matching of about 66%. Also, the transition models that we implement are trustworthy only in the batch data's support and hence our model's results cannot be trusted for the treatment of patients whose features and transitions are non-trivially different compared to our chosen patient cohort.

References

1. Aniruddh Raghu, Matthieu Komorowski, Leo Celi Ahmed, Imran, Peter Szolovits, and Marzyeh Ghassemi. Deep reinforcement learning for sepsis treatment. 2017.

2. Donghun Lee, Srivatsan Srinivasan, and Finale Doshi-Velez. Truly Batch Apprenticeship Learning with Deep Successor Features. *arXiv e-prints*, page arXiv:1903.10077, Mar 2019.

3. 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. volume 518, page 529. Nature Publishing Group, 2015.

4. Ian GoodFellow, Mehdi Pouget-Abadie, Jean andMirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Neural Information Processing Systems*, 2014.

5. Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. In *arXiv preprint arXiv:1710.11248*, 2017.

6. Chelsea Finn, Paul Christiano, Pieter Abbeel, and Sergey Levine. A Connection between Generative Adversarial Networks, Inverse Reinforcement Learning, and Energy-Based Models. *arXiv e-prints*, page arXiv:1611.03852, 2016.

7. John Schulman, Sergei Levine, Phillip Moritz, and Michael and Jordan. Trust region policy optimization. 2015.

8. Yonxin Yang, Irene Garcia Morillo, and Timothy M Hospedales. Deep neural decision trees. In *2018 ICML Workshop on Human Interpretability in Machine Learning*, 2018.