# Symbolic Regression for Interpretable Reinforcement Learning

***Abstract*—While recent years have been company to much progress in the reinforcement learning community, many tasks in use today still rely on carefully designed *reward functions*, many of which are products of constant tweaking and tuning by engineers and scientists. These reward functions, often *dense, symbolic functions of state*, don't exist in real world datasets, many of which are labeled by human experimenters — each with their own biases about desired behavior. In this work, we describe a new paradigm of *extracting symbolic reward functions from noisy data* called Interpretable Symbolic Reinforcement Learning (ISRL). ISRL allows for human experimenters to extract interpretable reward functions solely from data via symbolic regression. Our methods are able to find many reward functions commonly used in reinforcement learning benchmarks, using only tabular state-reward pairs as input. When ISRL finds alternative functions, we find that (a) the new reward functions generate better performing policies or (b) uncover a bias from human labelers — both of which may be crucial for the future of reward and task design. Our code and data labelling infrastructure is completely open-sourced for further development, along with an extensive "reward dataset" of hand-labeled trajectories.**

## I. Introduction

Reinforcement learning, especially robotic reinforcement learning, currently stands at an crossroads. Many of today's reinforcement learning algorithms are powerful, competent, and can solve many of the tasks in popular benchmarks. The onus of reinforcement learning progress now rests on the *design* of these tasks.

What makes task design - more specifically, *reward function design* - difficult? It turns out that many tasks of interest have easy-to-classify end states (as *achieved* or *incomplete*, as *good* or *bad* solutions, etc.), but ambiguous intermediate states. While we can discern whether a robot has correctly shelved a book, it becomes harder to define a reward for such an agent within the middle of its trajectory to that state. This notion of a *dense* reward is an assumption on which many of today's reinforcement learning algorithms rely, and stands in the way of wide adoption of robotic reinforcement learning.

Imitation learning assumes that agents are given expert demonstrations of a task and attempts to minimize divergences between the state-action visitation distributions of the agent and the expert. The expert's distribution needs to be inferred, and becomes a formidable problem when dealing with high-dimensional state and action distributions. Imitation learning also suffers from effective generalization due to overfitting and the *finite-data* assumption (i.e only a limited number of trajectories are given, and thereby cannot truly represent the entire space of possibilities an agent might encounter), but represents the most popular approach to encode human preferences into artificial agents. Offline reinforcement learning extends the imitation learning paradigm to a more challenging setting, where agents must learn purely from a dataset comprised of interactions from the environment and an *offline* policy.

Recent work (1) has shown that with the right interface, humans can quickly label *approximate* dense rewards for robotic agents. Large datasets now exist for this purpose, often developed for the field of batch (offline) reinforcement learning. As reward functions in reinforcement learning are often simple symbolic functions of state and action, we propose using human reward sketches to symbolically regress reward functions. Current work fits (and predicts) these *reward networks* with large MLPs, leading to black-box predictions of reward and a lack of intepretability.

This paper introduces the extraction of explicit reward functions using recent advances in *symbolic regression* (2). Our regressed reward functions generate interpretable, analytic reward functions which can be edited by human experimenters before use. In addition, we find that our recovered functions are less likely to overfit to the collected data, especially when using small amounts of transitions. Our proposed approach, titled *Interpretable Symbolic Reinforcement Learning* and illustrated in Figure 1, introduces a new paradigm to automatically generate interpretable reward functions. We test the efficacy of Interpretable Symbolic Reinforcement Learning (ISRL) and is tested on a variety of simulated batch Reinforcement Learning (RL) and traditional reinforcement learning settings. We also release a dataset of over two million hand-labelled reward sketches for a variety of robotic tasks, alongside the core Python interface to expand the data collection process.

## II. Preliminaries

### A. Reinforcement Learning

We consider a RL framework (3) where some task $T$ is defined by a Markov Decision Process (MDP) consisting of a state space $S$, action space $A$, state transition function $P : S \times A \mapsto S$, reward function $R : S \times A \mapsto \mathbb{R}$, and discount factor $\gamma \in (0, 1)$. The goal for an agent trying to solve $T$ is to learn a policy $\pi$ with parameters $\theta$ that maximizes the expected total discounted reward. We define a rollout $\tau = (s_0, a_0..., s_T, a_T)$ to be the sequence of states $s_t$ and actions $a_t \sim \pi(a_t|s_t)$ executed by a policy $\pi$ in the environment.

### B. Offline Reinforcement Learning

In standard RL, policies are constantly allowed to interact with the environment during policy improvement. This allows for exploratory behavior, especially in early stages of learning, which can often be crucial to an agent's success. In offline

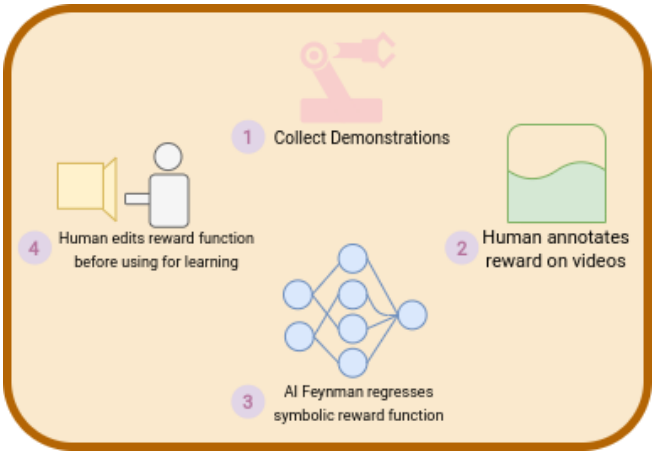


Fig. 1: Symbolic regression of rewards can provide human experimenters faster ways to debug imitation learning, by more easily casting it as reinforcement learning. Our approach, *Interpretable Symbolic RL* (ISRL), provides an interactive and safe way for humans to extract reward functions from coarsely-labelled data.

RL, an agent must learn entirely from a dataset collected offline, attempting to learn an optimal policy on some MDP without getting to interact with the environment. In addition, unlike imitation learning, the dataset is often not considered to be optimal, leading to further difficulty during policy optimization.

Offline reinforcement learning has made extensive progress in recent years. In this paper, we abstract the policy learning portion, focusing solely on reward function inference. For all offline RL experiments, we use an open-source implementation of Batch-Constrained Q-Learning (BCQ) (4), which is expanded on in detail in Appendix VI-B.

### C. Symbolic Regression

Symbolic regression concerns the discovery of an analytical expression that accurately matches a given dataset. Symbolic regression is an important tool in both engineering and natural sciences, and allows for fast recovery of even compositional or higher-order expressions from noisy data. While software tools in symbolic regression are well known (5), recent approaches have turned to the function approximation abilities of neural networks to amplify the power of symbolic regressors. In many cases (2; 6), neural networks are regressed to datasets before the symbolic expressions are extracted using heuristics such as symmetry, compositionality, and other simplifying properties.

In this work, we heavily utilize AI Feynman (2), a recent approach for symbolic regression. AI Feynman fits neural networks to datasets, and then uses heuristics based on input-output pairs to reduce and combine input variables. What remains is often a simple function of input, recovered analytically purely from tabular data. AI Feynman is expanded on in Appendix VI-C.

### D. Reward Sketching

Many tasks, especially in reinforcement learning scenarios, have complex, engineered reward functions. While many tasks have intuitive termination states, leading to a large number of tasks that are defined as *sparse-reward* problems, many algorithms today require the notion of a dense reward (7). As human labelers are often used in supervised learning settings (8), a simple way to generate dense reward datasets is to label trajectories approximately using web interfaces (1). Given the context of a whole trajectory, humans can quickly *sketch* reward accumulation for agents, generating approximate yet dense rewards for even complex problems.

## III. Method

In this section, we introduce an approach to symbolically regress reward functions from data. Recovering analytic reward functions is crucial for interpretable reinforcement learning agents, especially as the research community moves towards batch and offline reinforcement learning settings. As described in Algorithm 1, we introduce **Interpretable Symbolic RL**, a new paradigm for extracting symbolic rewards from arbitrary datasets.

**Algorithm 1:** Interpretable Symbolic RL

**Returns**: A symbolic reward expression.
**Input**: Dataset $X$, consisting of $(s, a, r)$ pairs, $N$ transitions to use.
$X' \sim$ `random.sample(`$X$`,` $N$`)`
Pareto Frontier of $f_{reward}$ = `AIFeynman(`$X'$`)`
*Human Experimenter*: Choose, edit final $f_{reward}$
**if** *offline* **then**
  **for** *transition in* $X$ **do**
    $s, a$ = `transition['s'],`
     `transition['a']`
    `transition['r']` = $f_{reward}(s, a)$
  **end**
**else**
  `environment.reward` = $f_{reward}$
**end**

### A. Batch RL

To test AI Feynman's ability to regress reward functions from datasets, we begin in the batch RL setting, where datasets are collected and stored offline. We take a randomized subset of the dataset, and then tabularize the observation and action and generate a dataset of $(x, y)$ regression pairs, where $x$ is the concatenated observation and action and $y$ is the reward[1]. We then run the AI Feynman subroutine, as described in Section II-C, which returns a Pareto frontier of reward functions. As the Pareto frontier is a two-dimensional "ranking" based on accuracy and complexity, we return the top-three choices (by accuracy) to a human experimenter, who then picks the reward function an agent is to use.

[1]The size of the dataset is ablated on in Section IV-E1.

In the batch RL setting, we then relabel the entirety of the dataset using the regressed reward function, and run the batch RL algorithm on the annotated dataset.

### B. Human-Annotated RL

Human-annotated rewards are, by nature, *noisy* functions of state and goal. To generate a dataset, we run a reinforcement learning algorithm (here, we use Deep Deterministic Policy Gradient (DDPG) and Hindsight Experience Replay (9; 10), which are described in detail in Appendix VI-A) and learn a particular task. We store every transition executed by the training policy, and use a designed web interface to play back each transition to be labelled by a human experimenter, as visualized in Figure 1. In this context, *label* refers to an approximate reward a human user attributes to a particular state. As human annotations are time-intensive, we linearly-interpolate between labelled points. In our experiments, human users, on average, labelled $< 10\%$ of states in a particular trajectory, trading off efficiency for quality of labelling. However, as we shall see in the following section, our approach is able to use even these noisy annotations to recover interesting reward functions. More interestingly, the reward functions often highlight hidden biases that human labelers seem to have about task performance in reinforcement learning, leading to insights regarding how reward functions might be designed in future benchmarks.

The dataset released with this paper includes two million unique transitions labelled by four human experimenters across three different robotic tasks. For all human-annotation experiments, we use this dataset, ablating mainly on the size of the data subset that AI Feynman uses to regress a reward function.

## IV. Experimental Setup

### A. Baselines

The main comparisons we use against our approach are against the method described in (1) as well as against an oracle, ground-truth reward[2]. (1) fits a neural network to a batch of human-annotated data, continuously refreshing it as the agent collects more experience. In this work, we implement their approach by fitting a neural network to the subset of data our method would see, and use train it with cross-validated hyperparameters described in Appendix VI-D; in the offline setting, we fit the network to the static dataset (with ground-truth rewards), and in the annotation setting, we fit the network to the human annotations our approach sees. Most importantly, we use the same hyperparameters for the neural network training in both our approach and the neural network baseline from (1). All experiments are mean-averaged over five seeds.

### B. Environments

We test our approach on a variety of goal-directed environments, described below and in Tables I-II.

[2]The ground-truth reward function is the one that "ships" with the environment, often implemented inside of a programming language using state information.

Fig. 2: A simple agent - a *point mass* - travels in two dimensions to the goal. The simpler task rewards the agent with the standard Euclidean distance between itself (*green* circle) and the goal (*red* circle). The more difficult environment rewards the agent with the exponential of the negative Euclidean distance to the goal.

*1) Batch RL:* In the batch setting, we test our approach on **2D-Pointmass** and **AntNavigation**, both from (11). The point mass simply moves in the $xy$ plane, but the Ant requires an agent to control the eight joints to propel itself forward (Figure 3). Both environments require navigation of agent to randomly selected goals in a maze; the environment variations increase the size (and thereby the difficulty) of these mazes (Figure 2). Lastly, the ground truth reward functions used in each environment are either the negative of the euclidean distance between the agent and goal (*Simple* variants) or the exponential of that same quantity (*Exponential* variants). During evaluations, we report either the accumulated (ground-truth) reward over a trajectory (in *2D-Pointmass*) or number of successful navigations (in *AntNavigation*), where higher values are better.

*2) Human Annotations:* When testing our approach on human annotations, we test using the Fetch Robotics environments from (12). We test, in order of difficulty, **FetchReach** (move end effector to goal location), **FetchPush** (move block to goal location), and **FetchSlide** (knock puck on slippery surface to goal location), as seen in Figure 4. During evaluations, we report final distance of the end effector (Reach) or object (Push, Slide) to the goal, where smaller values are better.

### C. Batch RL Results

We start by verifying that AI-Feynman can learn and recover symbolic reward functions from simple navigation tasks, illustrated in Figure 2 and listed in Table I. In the *SimpleNavigation* tasks, we train a pointmass agent to move itself in two-dimensions towards a randomly selected goal in the maze. The reward has two variants — standard (negative) Euclidean distance and the exponential of the same value — which we denote as two different environments *Simple-** and *Exponential-** respectively.

With the results in Table I, we see that for the simpler variants of the task, the reward recovery system has no issue, whereas the *true* reward for the exponential variants is unable to be recovered. However, as we see in Figure 5 (which plots the error distribution between the recovered and ground truth on 1M randomly sampled vectors), the recovered function shows remarkable similarity to the true function. While seeming a failure for ISRL, when relabelling

Fig. 3: A more complex agent - the *Ant* - is a high-dimensional locomoter which navigates to the goal by controlling its eight joints. The simpler task rewards the agent with the standard Euclidean distance between itself (*green* circle) and the goal (*red* circle). The more difficult environment rewards the agent with the exponential of the negative Euclidean distance to the goal.

the dataset with the recovered reward function, we see (Figure 9) that these recovered functions actually are more efficient in learning agent navigation policies than the original negative exponential of Euclidean distance between goal and agent.

AI Feynman, comparing Figures 6 and 7, is able to find a reward that generates a natural *curriculum* (13) for the agent; the standard exponential of Euclidean distance drops off quite sharply, making it hard to pick up reward signals far away from the goal (where an agent mind find itself often in early stages of training). ISRL's recovered reward function is smoother towards the optima reached at the goal, likely allowing for an agent to have a smoother policy optimization process. The recovered function is reminiscient of reward shaping literature, where reward functions components are designed by a human experimenter to progressively nudge the agent towards the desired behavior. We see in Figure 8 that the neural network completely fails to capture the true reward function, despite fitting the given data subset well.

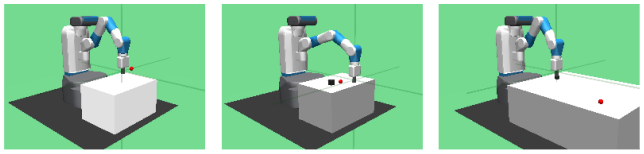

Fig. 4: The robotics tasks we focus on, *FetchReach*, *FetchPush*, *FetchSlide*, all require an agent to accomplish a randomized goal in the environment. The reward functions here are learned from human annotations, making the task completely subjective on the quality of the labeling.

### D. Human Annotation Results

In this section, we use a subset of data labelled as *Expert* data, which includes only the final $10\%$ of transitions from the training policy. These transitions are gathered toward the end of learning, where task completion rates are generally higher. We ablate on the effects size and quality of dataset in Section IV-E1 and Section IV-E2 respectively.

Using the *Expert* subset from the human-annotated dataset described in Section III-B, we report our results in Table II and Figures 12-17. On the *FetchReach* task, our approach is unable to find the true reward, leading to a drop in performance as compared to the ground-truth baseline.

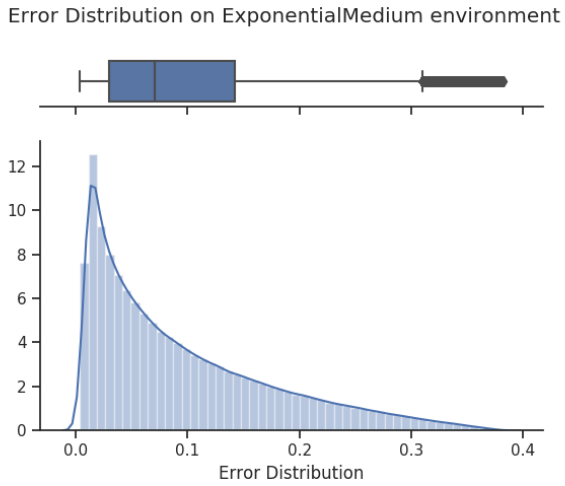


Fig. 5: Despite recovering a "different" reward function, the error distribution between the two functions on randomly sampled inputs is skewed heavily towards zero.

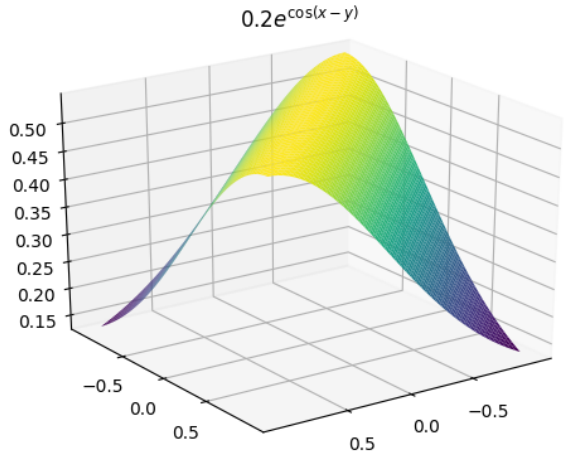


Fig. 6: $0.2e^{\cos(x-y)}$, plotted. AI Feynman is able to find a naturally smooth reward function that fits the data, likely leading to a natural curriculum to help the locomoter slowly learn to reach the goal.

However, in both *FetchPush* and *FetchSlide*, AI Feynman we see performance that matches the ground-truth baseline. Most important is when compared to the pure neural network approach from (1), we see that the symbolic regression step generates sizeable improvements in task performance, especially in these low data regimes.

### E. Ablation Studies

*1) Dataset Sizes Affects Reward Inference:* In this section, we increase the size of the dataset (labelled as *Medium*) to show each method the final $25\%$ of transitions from the training policy to do reward inference. While still generally high-quality transitions, this dataset includes much noisier data. From our experiments, we found that human experimenters had trouble labelling *extremely* bad policies (i.e policy does not move the agent for the entirety of the

| Environment Name | Description | True Reward Function | Recovered? | Recovered Function |
|---|---|---|---|---|
| SimpleSmall | 2D Navigation with Pointmass | $-\|\|x_{agent} - x_{goal}\|\|_2$ | Yes | |
| SimpleLarge | 2D Navigation with Pointmass | $-\|\|x_{agent} - x_{goal}\|\|_2$ | Yes | |
| ExponentialLarge | 2D Navigation with Pointmass, Large Maze | $e^{-\|\|x_{agent} - x_{goal}\|\|_2}$ | No | $0.1e^{\cos(x_0 - x_2) + \cos(x_1 - x_3)}$ |
| AntUMaze | High dimensional locomoter | $e^{-\|\|x_{agent} - x_{goal}\|\|_2}$ | No | $\sqrt{0.217e^{(cos(x1-x3)-(x0-x2)^2)}}$ |

TABLE I: The environments and their respective AI Feynman-recovered reward functions. We see that the standard Euclidean distance rewards are recovered easily, whereas the exponential reward function has difficulty being recovered. The agent's state — $x_{agent}$ — consists of $[x_0, x_1]$ whereas the goal state — $x_{goal}$ — consists of $[x2, x3]$.

| Environment Name | Description | Number of transitions | Recovered? | Recovered Function |
|---|---|---|---|---|
| FetchReach | Robot reaching fictional goal | 2500 | No | $\arccos(0.5 + c(\frac{x0}{x2 - x5 + x3} + x1 - x4))$ |
| FetchPush | Robot pushing block to goal | 1000 | No | $-\|x_1 - x_4\| + \sin(1/\cos(x_0 + x_2 - x_3 + x_5))$ |
| FetchSlide | Pushing slippery block to goal | 1000 | Yes | Euclidean distance b.w object and goal |

TABLE II: On a small dataset (i.e. 3000 transitions or less) **of human-labelled** rewards, AI Feynman is able to recover functions that are remarkably close to the original. All environments use the negative Euclidean distance between the object and goal (or, Cartesian coordinates of robot hand and goal) as the reward for the agent.

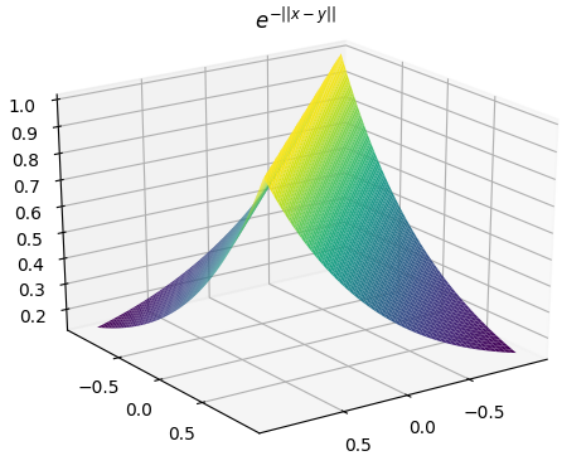


Fig. 7: $e^{-||x-y||_2}$, plotted. Unlike the function found by AI Feynman, the exponentiated euclidean distance, traditionally used to train navigation and locomotion, provides a sharp increase in value. At distances further from the goal (where an agent may normally find itself early during training), such a sharp dropoff may introduce difficulties into the policy optimization process. We especially highlight the difference in y-axis between this function and the function found in Figure 6.

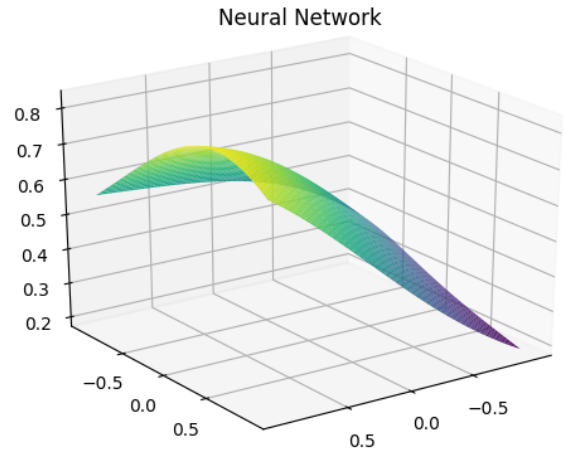


Fig. 8: Fitted neural network function, plotted. Despite fitting the small dataset well, we see that the neural network fails to learn the true reward function.

episode), leading to constant labelled rewards. We see, in almost all environments, that the policies trained with our approach do worse here than with the *Expert* counterparts, despite still maintaining better performance than the neural network approach from (1).

With this larger dataset, we find interesting cases when our approach is *unable* to recover the true reward from human annotations; for example, in Table III, we can see that the **FetchSlide** experiment has a multiplicative term for the difference between $x_1$ and $x_4$, which is the $y$-coordinate of the object and goal location respectively. Due to the design of the environment and physics engine, the $y$-coordinate is the more "important" coordinate, as the goal is often located at a large distance (with respect to $y$) from the start, while the $x$-coordinate varies much less heavily. We can infer the the human annotators "valued" the closeness with respect to $y$ more than the other two coordinates, which emerges as the multiplicative constant seen in the symbolic equation.

*2) Dataset Quality Affects Reward Inference:* While we do not include plots, we find that when we show our approach and the neural network approach the entirety of the annotated datasets, neither approach is able to learn any useful information. We hypothesize that this is an exacerbation of the issue described in the previous section: when the policy performs incredibly poorly (i.e when the policy moves randomly, or does not move at all), human labelers have no information on how to label such episodes. We found that over 20% of transitions in the first quarter of training epochs were labelled as constant. When replaying those episodes, we see that the policy is often in one of these two failure states. We acknowledge our method's need for high quality transition data, and aim to improve on this issue in future work.

| Environment Name | Description | Number of transitions | Recovered? | Recovered Function |
|---|---|---|---|---|
| FetchReach | Robot reaching fictional goal | 10000 | Yes | Euclidean distance b.w arm and goal |
| FetchPush | Robot pushing block to goal | 5000 | Yes | Euclidean distance b.w object and goal |
| FetchSlide | Pushing slippery block to goal | 3000 | No | $-23 * ((x0 - x5 + x2 - x3) * (x1 - x4))^2$ |

TABLE III: On a larger dataset **of human-labelled** rewards, AI Feynman is able to recover functions that are remarkably close to the original. All environments use the negative Euclidean distance between the object and goal (or, Cartesian coordinates of robot hand and goal) as the reward for the agent.

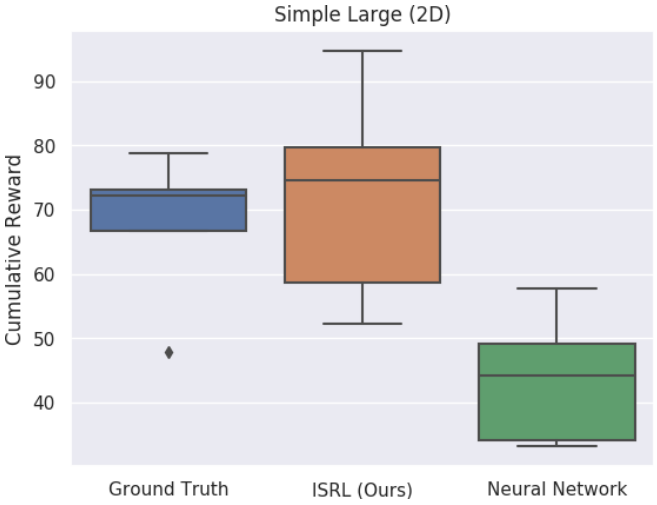


Fig. 9: When AI Feynman is able to recover the correct reward function, we see that performance is similar to performance when training with the ground truth reward function. In this simpler regression task, we see that the neural network is reasonably close to the performance of both the ground truth baseline and our approach. **Higher is better.**

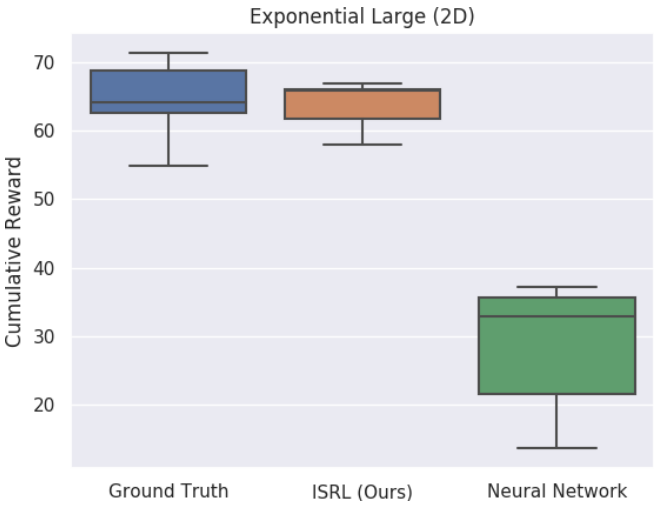


Fig. 10: When using the recovered reward functions seen in Table I, we see similar final performance but smaller spread of agents than when using the groundtruth, "intuitive" reward. In this harder task, we start to see the benefits of symbolic regression over the neural network approach proposed in (1). **Higher is better.**

## V. Related Work

In this section, we place our work in the context of other methods, centered around *human-in-the-loop* learning systems. Recently, (14) proposed an evolutionary approach to building symbolic, interpretable reward functions. The method described requires learning of a multitude of agents (where an agent, in a reinforcement learning setting, consists of a *(policy, reward function)* pair), which is much less computationally-efficient than the approach described in this work. In addition, as discussed throughout the text, (1)

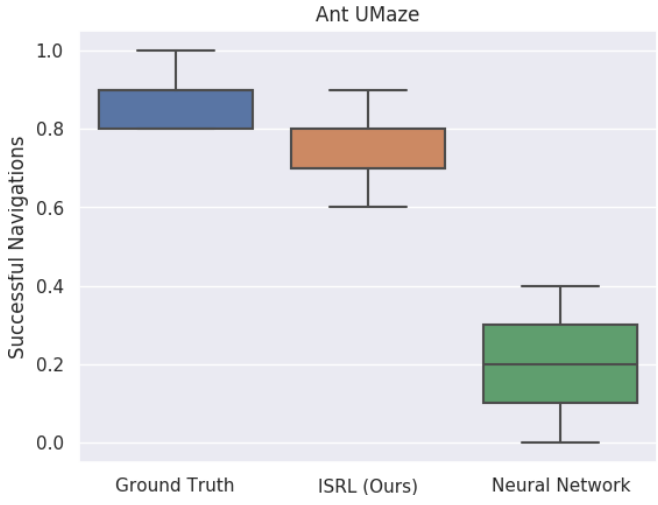


Fig. 11: Even on the more difficult *Ant* variant of the navigation task, using the recovered reward function leads to comparable performance when comparing to the groundtruth, "intuitive" reward. **Higher is better.**

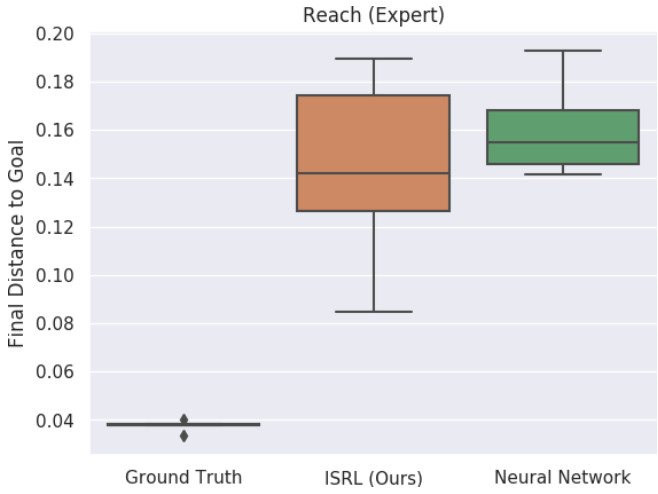


Fig. 12: When using the high-quality, small annotated dataset on a reaching task, we see comparable performance between ours and neural network driven approaches. **Lower is better.**

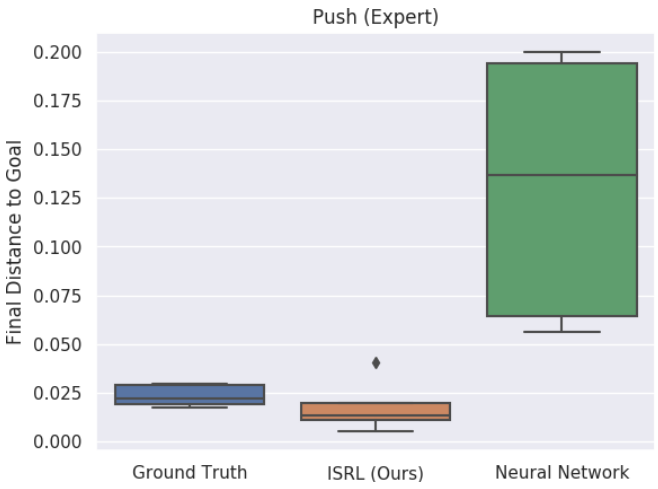


Fig. 13: When using the high-quality, small annotated dataset on a pushing task, we start to see the benefits of symbolic regression. Our regressed function, seen in Row 2 of Table II, generates better policies than even the ground truth function, which both outperform the neural network approach from (1). **Lower is better.**

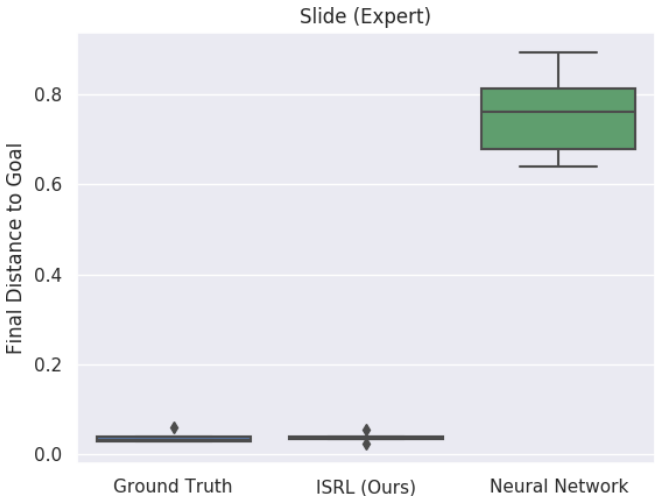


Fig. 14: When using the high-quality, small annotated dataset on the sliding task, our method vastly outperforms the black-box neural network reward function. **Lower is better.**

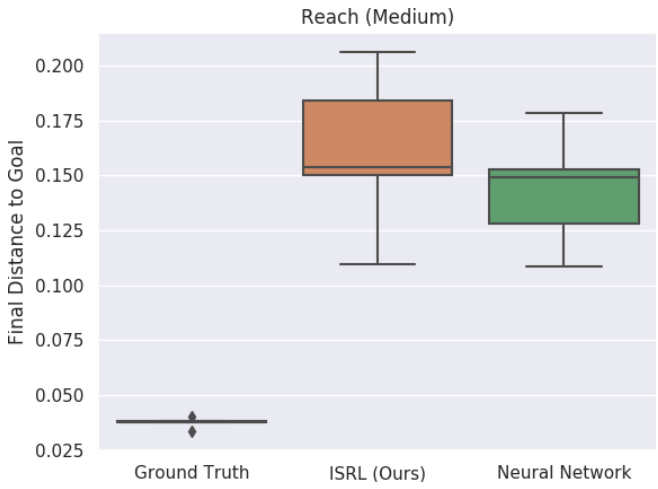


Fig. 15: On the larger, noisier dataset, our approach degrades in the simpler task. **Lower is better.**

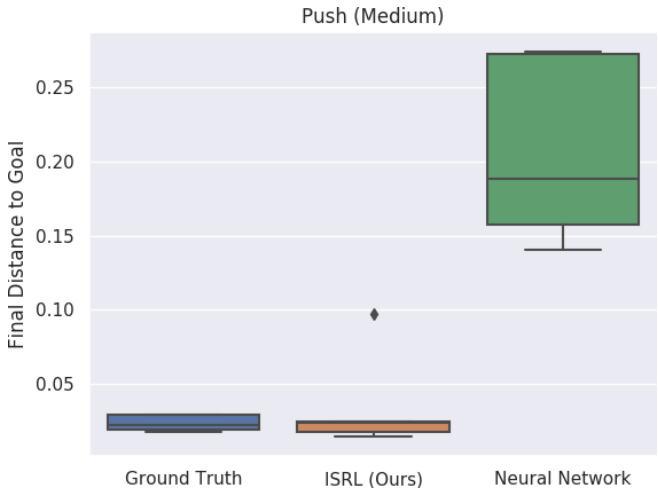


Fig. 16: On the harder, pushing task with a noisy dataset, we find significant improvements compared to the neural network approach, further solidifying the hypothesis that symbolic regression improves reward learning baselines. **Lower is better.**

learns a black box reward function via human annotations; our results show that by regression of symbolic reward functions, there are improvements in all important dimensions of reinforcement learning agents: performance, robustness, and interpretability.

### A. Reward Inference and Learning

The field of *inverse reinforcement learning* — the learning of reward functions from trajectories — is a well-known, computationally-difficult problem in the reinforcement learning community (15). Recently, work using deep networks has to do reward inference has proliferated the space for the last decade (16). Most applicable to our paper is the field of *AI (Value) Alignment* (17). AI alignment often uses inverse RL approaches to help agents specify goals to human experimenters before optimizing for them, hopefully leading to safe behavior upon deployment (18; 19; 20; 21). Our work also requires a human to pick the reward function from the Pareto-frontier, enabling a safer optimization loop than black box inference of reward functions, such as the approach studied in (1).

### B. Human-in-the-Loop Reinforcement Learning

Recently, there has been a plethora of interest in learning reward functions from human preferences. As discussed in Section I, humans show poor performance in defining dense rewards to agent trajectories (even reward design is difficult, and can lead to interesting failure cases (22)). As a result, recent approaches have learned *classification-based* rewards from human labels. In these works, classifiers are learned from a dataset of end states; these end states are labelled quickly by humans as *good* and *bad*, allowing the

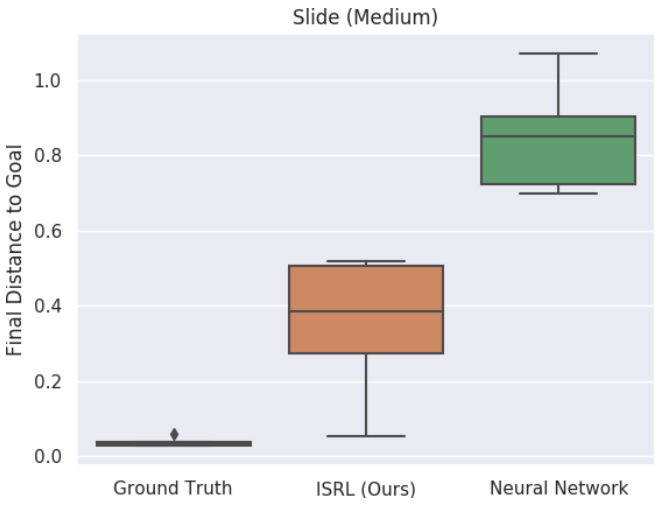


Fig. 17: In the hardest sliding task with a noisy dataset, our method improves upon the neural network baseline significantly, despite not matching the oracle in this task. Our method provides greater interpretability into biases from the human labelers, as described in Section IV-E1. **Lower is better.**

agent to classify novel end states with sparse reward signals (23). Another line of work attacks these issues by using meta-learning approaches for fast adaptation (24), but often requires ground-truth dense rewards during training.

## References


[1] S. Cabi, S. Gómez Colmenarejo, A. Novikov, K. Konyushova, S. Reed, R. Jeong, K. Zolna, Y. Aytar, D. Budden, M. Vecerik, and et al., "Scaling data-driven robotics with reward sketching and batch reinforcement learning," *Robotics: Science and Systems XVI*, Jul 2020. [Online]. Available: http://dx.doi.org/10.15607/rss.2020.xvi.076

[2] S.-M. Udrescu, A. Tan, J. Feng, O. Neto, T. Wu, and M. Tegmark, "Ai feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity," 2020.

[3] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[4] S. Fujimoto, D. Meger, and D. Precup, "Off-policy deep reinforcement learning without exploration," 2019.

[5] M. Schmidt and H. Lipson, "Distilling free-form natural laws from experimental data," *science*, vol. 324, no. 5923, pp. 81–85, 2009.

[6] M. Cranmer, A. Sanchez-Gonzalez, P. Battaglia, R. Xu, K. Cranmer, D. Spergel, and S. Ho, "Discovering symbolic models from deep learning with inductive biases," 2020.

[7] A. Irpan, "Reinforcement learning doesn't work, yet." [Online]. Available: https://www.alexirpan.com/2018/02/14/rl-hard.html

[8] Amazon, "Amazon mechanical turk." [Online]. Available: https://mturk.com

[9] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2019.

[10] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, "Hindsight experience replay," 2018.

[11] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," 2016.

[12] M. Plappert, M. Andrychowicz, A. Ray, B. McGrew, B. Baker, G. Powell, J. Schneider, J. Tobin, M. Chociej, P. Welinder *et al.*, "Multi-goal reinforcement learning: Challenging robotics environments and request for research," *arXiv preprint arXiv:1802.09464*, 2018.

[13] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*, 2009, pp. 41–48.

[14] H. Sheikh, S. Khadka, S. Miret, and S. Majumdar, "Learning intrinsic symbolic rewards in reinforcement learning," 2020.

[15] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning," in *Proceedings of the Seventeenth International Conference on Machine Learning*, ser. ICML '00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, p. 663–670.

[16] R. Akrour, M. Schoenauer, and M. Sebag, "Preference-based policy learning," in *Proceedings of the 2011 European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part I*, ser. ECML PKDD'11. Berlin, Heidelberg: Springer-Verlag, 2011, p. 12–27.

[17] I. Gabriel, "Artificial intelligence, values and alignment," *ArXiv*, vol. abs/2001.09768, 2020.

[18] P. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," 2017.

[19] D. Amodei, C. Olah, J. Steinhardt, P. F. Christiano, J. Schulman, and D. Mané, "Concrete problems in AI safety," *CoRR*, vol. abs/1606.06565, 2016. [Online]. Available: http://arxiv.org/abs/1606.06565

[20] D. Hadfield-Menell, A. D. Dragan, P. Abbeel, and S. J. Russell, "The off-switch game," *CoRR*, vol. abs/1611.08219, 2016. [Online]. Available: http://arxiv.org/abs/1611.08219

[21] T. Everitt, V. Krakovna, L. Orseau, M. Hutter, and S. Legg, "Reinforcement learning with a corrupted reward channel," *CoRR*, vol. abs/1705.08417, 2017. [Online]. Available: http://arxiv.org/abs/1705.08417

[22] OpenAI, "Faulty reward functions in the wild." [Online]. Available: https://openai.com/blog/faulty-reward-functions/

[23] M. Palan, N. C. Landolfi, G. Shevchuk, and D. Sadigh, "Learning reward functions by integrating human demonstrations and preferences," *CoRR*, vol. abs/1906.08928, 2019. [Online]. Available:

http://arxiv.org/abs/1906.08928
[24] A. Gupta, R. Mendonca, Y. Liu, P. Abbeel, and S. Levine, "Meta-reinforcement learning of structured exploration strategies," *CoRR*, vol. abs/1802.07245, 2018. [Online]. Available: http://arxiv.org/abs/1802.07245
[25] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," *CoRR*, vol. abs/1802.09477, 2018. [Online]. Available: http://arxiv.org/abs/1802.09477

## VI. APPENDIX: DETAILED ALGORITHM EXPLANATIONS

### A. Deep Deterministic Policy Gradients

Across all experiments, all networks share the same network architecture and hyperparameters. For each policy, we use Deep Deterministic Policy Gradients (9), using the `OurDDPG.py` implementation from the open source repository of (25). Each actor and the critic have two hidden layers with 400 and 300 neurons, respectively, and use ReLU activation.

DDPG concurrently learns a Q-function and policy, using off-policy data to learn the Q-function, and then optimizing the policy with respect to the learned Q-function. In our work, the policy learning is abstracted away, and DDPG can be replaced with any on- or off-policy reinforcement learning method (or, any other control method, such as Model Predictive Control).

### B. Batch Constrained Q-Learning

Despite success of off-policy reinforcement learning algorithms such as DDPG, many algorithms only work when the replay buffer used for Q-function updates is *correlated* to the current policy. Batch Reinforcement Learning extends the difficulty of this setting, by using data collected from an *entirely* different policy, where algorithms like DDPG have been shown to break down quickly.

Batch Constrained Q-Learning (4) claims that using algorithms like DDPG on such buffers (i.e from policies entirely unrelated to the current) leads to the learning of the Q-function of a *different* MDP, which then generates large *extrapolation errors* and poor performance. BCQ fixes these issues by restricting the policy only to possible actions to be in the batch. Their work introduces several other points that are crucial for correct implementation, all of which can be abstracted away as in this work, what matters is that ISRL operates on the same batch of data that BCQ uses for policy learning.

### C. AI-Feynman

AI Feynman (2) is a recent advance in symbolic regression, which aims to find symbolic expressions from data by fitting neural networks and finding symmetries from neural network gradients.

AI Feynman receives as input *tabular, input-output regression pairs*, and outputs a Pareto frontier of symbolic expressions, ordered by both complexity (in bits) and accuracy (of fit with respect to dataset). After fitting a neural network to the dataset, AI Feynman looks for symmetries in gradients under translation, scaling, and other mathematical operations; after finding such symmetries in data, it *replaces* the affected input variables with another, "meta"-variable, builds a new dataset (i.e by replacing an expression such as $(x - y)$ with a new variable that represents it: $z$), and continues the recursive operation. This allows AI Feynman to reduce high dimensional datasets to compact symbolic representations of data.

### D. Regression Network Hyperparameters

For all neural networks used in regression, we train until *convergence*: a $< 10^{-5}$ MSE. We use a three layer MLP with layer sizes of $128 - 128 - 128$, all except the final layer activated with tanh activations. We use the ADAM optimizer with an initial learning rate of $0.001$, and use an adaptive learning rate schedule (dropping the learning rate by a factor of 10 with a tolerance of 10 epochs) until convergence is reached.