

Policy Transfer Algorithms for Meta Inverse Reinforcement Learning

Benjamin Kha

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2019-54

<http://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-54.html>

May 17, 2019



Copyright © 2019, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Policy Transfer Algorithms for Meta Inverse Reinforcement Learning

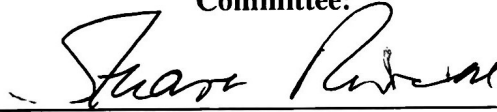
by Benjamin Kha

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences,
University of California at Berkeley, in partial satisfaction of the requirements for the
degree of **Master of Science, Plan II.**

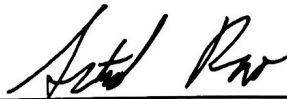
Approval for the Report and Comprehensive Examination:

Committee:



Professor Stuart Russell
Research Advisor

(05/15/19)



Professor Satish Rao
Second Reader

(05/15/19)

Policy Transfer Algorithms for Meta Inverse Reinforcement Learning

Benjamin Kha

Abstract

Inverse reinforcement learning (Ng & Russell, 2000) is the setting where an agent is trying to infer a reward function based on expert demonstrations. Meta-learning is the problem where an agent is trained on some collection of different, but related environments or tasks, and is trying to learn a way to quickly adapt to new tasks. Thus, meta inverse reinforcement learning is the setting where an agent is trying to infer reward functions that generalize to multiple tasks. It appears, however, that the rewards learned by current meta IRL algorithms are highly susceptible to overfitting on the training tasks, and during finetuning are sometimes unable to quickly adapt to the test environment.

In this paper, we contribute a general framework of approaching the problem of meta IRL by jointly meta-learning both policies and reward networks. We first show that by applying this modification using a gradient-based approach, we are able to improve upon an existing meta IRL algorithm called Meta-AIRL (Gleave & Habryka, 2018). We also propose an alternative method based on the idea of contextual RNN meta-learners. We evaluate our algorithms against a single-task baseline and the original Meta-AIRL algorithm on a collection of continuous control tasks, and we conclude with suggestions for future research.

1. Introduction

Inverse reinforcement learning (IRL) attempts to model the preferences of agents by observing their behavior. This goal is typically realized via attempting to approximate an agent’s reward function, rather than being provided them explicitly. Inverse reinforcement learning is particularly attractive because it allows leveraging machine learning to model the preferences of humans in complex tasks, where explicitly encoding reward functions has performed poorly and has been subject to issues such as negative side effects and reward hacking (Amodei et al., 2016).

Standard reinforcement learning traditionally models an agent’s interaction with its environment as a *Markov deci-*

sion process (MDP), wherein the solution is a policy mapping states to actions, and an optimal policy is derived by receiving rewards as feedback and modifying the policy accordingly. Inverse reinforcement learning, on the other hand, assumes an agent that acts according to an optimal (or almost optimal) policy and uses data collected about the optimal agent’s actions to infer the reward function.

Inverse reinforcement learning has incredible ramifications for the future of artificial intelligence, and has generated increasing interest for a couple of important reasons:

1. *Demonstration vs. Manual Rewards* - Currently, the requirement in standard reinforcement learning of pre-specifying a reward function severely limits its applicability to problems where such a function can be specified. The types of problems that satisfy these constraints tend to be considerably simpler than ones the research community hopes to solve, such as building autonomous vehicles. As IRL improves, this paradigm will shift towards learning from demonstration.
2. *Increasing Generalizability* - Reward functions as they stand offer a very rigid way of establishing rewards in specific environments; they typically fail to generalize. However, learning from demonstration, as is done in IRL, lends itself to transfer learning when an agent is placed in new environments where the rewards are correlated with, but not the same as those observed during training.

Meta-learning is another exciting subfield of machine learning that has recently gained significant following, and tackles the problem of *learning how to learn*. Standard machine learning algorithms use large datasets to generate outputs based on seen training examples; unlike their human counterparts, these algorithms are typically unable to leverage information from previously learned tasks. Meta-learning is useful largely because it allows for rapid generalization to new tasks; we hope to apply meta-learning techniques to achieve this rapid generalizability for approximating reward functions in inverse reinforcement learning.

In this paper, we propose a general framework for meta-learning reward functions (learning how to learn reward functions) that should improve as the performance of single-

task IRL algorithms improve. Specifically, our contributions are as follows:

- We propose a framework for meta inverse reinforcement learning based on jointly meta-learning a policy along with a reward network, and develop two prototype algorithms that follow this framework.
- We provide an evaluation of our algorithms on a collection of continuous control environments, and evaluate them against both single-task and multi-task baselines.

2. Related Work

Older work in IRL (Dimitrakakis & Rothkopf, 2011) (Babes et al., 2011) is based on a Bayesian Inverse Reinforcement Learning model. The drawback behind this approach is that no methods based on Bayesian IRL have been able to scale to more complex environments such as continuous control robotics tasks.

A more promising direction is offered by the maximum causal entropy model (MCE), which as originally stated is still limited to finite state spaces. However, recent methods such as Guided Cost Learning (Finn et al., 2016), and Adversarial IRL (Fu et al., 2017) have been able to extend IRL methods to continuous tasks.

In traditional meta-learning, there has been a broad range of approaches in recent years. Some of these methods include algorithms like Model-Agnostic Meta-Learning (Finn et al., 2017) which tries to learn a good initialization of a model’s parameters that can quickly adapt to a new task with a small number of gradient updates, while also attempting to prevent overfitting. Reptile (Nichol et al., 2018) is a similar algorithm to MAML, except it does not unroll a computation graph or calculate any second derivatives, thereby saving computation and memory. Finally, RNN meta-learners (Chen et al., 2016) (Duan et al., 2016) try to adapt to new tasks by training on *contexts*, which are the past experience of an agent during a particular trial, and can encode some structure of the task.

There has been very recent work on applying meta-learning algorithms to the IRL setting. Specifically, in a recent paper by Xu et al. (2018) the authors explore applying MAML on a discrete grid-based environment. Similarly, in a paper by Gleave & Habryka (2018), the authors explore applying the Reptile and Adversarial IRL (AIRL) algorithms on continuous control tasks. In this work, we explore the use of both gradient-based and contextual RNN meta-learners in continuous IRL settings.

3. Background and Preliminaries

In this section, we describe some mathematical background on inverse reinforcement learning and meta-learning problems.

3.1. Inverse Reinforcement Learning

The standard Markov decision process (MDP) is defined by a tuple $(\mathcal{S}, \mathcal{A}, p_s, r, \gamma)$, where \mathcal{S} and \mathcal{A} denote the set of possible states and actions, $p_s : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ denotes the transition function to the next state s_{t+1} given both the current state s_t and action a_t , $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbf{R}$ denotes the reward function, and $\gamma \in [0, 1]$ is the discount factor. Traditionally, the goal of standard reinforcement learning is to learn a policy that maximizes the expected discounted return after experiencing an episode of T timesteps:

$$R(\tau) = \sum_{t=1}^T \gamma^{t-1} r(s_t, a_t)$$

Inverse reinforcement learning assumes that we don’t know r , but rather we have a sequence of expert trajectories $\mathcal{D} = \{\tau_1, \dots, \tau_K\}$ where each trajectory $\tau_k = \{s_1, a_1, \dots, s_T, a_T\}$, is a sequence of states and actions.

3.2. Meta-Learning

Meta-learning tries to *learn how to learn* by optimizing for the ability to generalize well and learn new tasks quickly. In meta-learning, the agent interacts with tasks from a meta-training set $\{\mathcal{T}_i; i = 1, \dots, M\}$ and meta-test set $\{\mathcal{T}_j; j = 1, \dots, N\}$, both of which are drawn from a task distribution $p(\mathcal{T})$. During the meta-training process, the meta-learner learns to better generalize across the tasks it trains on, such that it is able to leverage this information to efficiently learn new tasks in the meta-test set with fewer required training examples to achieve comparative performance.

In reinforcement learning this amounts to acquiring a policy for a new task with limited experience, for which there are two main approaches:

1. *Gradient-based Methods* - Gradient-based meta-learning methods maintain a meta-parameter θ , which is used as the initialization parameter to standard machine learning and reinforcement learning algorithms, which then compute local losses for and update parameters for sampled batches of individual tasks $\mathcal{T}_i \sim p(\mathcal{T})$. Localized training follows the gradient update rule below:

$$\theta'_i \leftarrow \theta - \alpha \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta})$$

These updated parameters after gradient steps on sampled individual tasks are then used to update the meta-

parameter with the following update rule:

$$\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{\tau_i \sim p(\tau)} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i})$$

With sufficient iterations, a meta-learner is able to use the learned meta-parameter θ to quickly adapt to new, unseen tasks.

2. *Recurrence-based Methods* - Recurrence-based methods take an entirely different approach. Rather than explicitly compute gradients and update parameters, they use a recurrent neural network to condition on past experience via a hidden state. By leveraging this past experience or *context*, these policies can encode the structure of the training environments, which can enable them to quickly adapt to similar test tasks.

4. Policy Transfer for Meta Inverse Reinforcement Learning

4.1. Formulation

We assume that we have a set of tasks \mathcal{T} over which we want our agent to meta-learn, and a task distribution $p(\mathcal{T})$ over these tasks from which we sample them. We define a *trial* as a series of episodes of interaction with a given MDP.

Within each trial, a new MDP environment is drawn from our task distribution, and for each episode within a trial a new initial state s_0 is drawn from the corresponding MDP’s underlying state distribution.

At each timestep t , the policy takes an action a_t , which produces the a reward r_t , a termination flag d_t (which indicates whether the episode has ended or not), and the next state s_{t+1} .

Under this framework, our goal is to minimize the loss across entire trials, rather than individual episodes. In the recurrence-based setting, the hidden state h_t is used as additional input to produce the next action, and stores contextual information that is aggregated across the many episodes in a trial. Because the environment stays the same within a single trial, an agent can leverage information from past episodes and the current one to output a policy that adapts to the environment of the current trial. With sufficient training this leads to a more efficiently adaptable meta-learner that is able to quickly infer reward functions. A visualization of this process can be seen in Figure 1.

4.2. Gradient-Based Policy Transfer

We first implement the idea of jointly meta-learning both a policy and reward network by applying a gradient-based approach to meta-learning the policy. As the basis for our initial experiments, we selected Reptile (Nichol et al., 2018)

Algorithm 1 PolicyMetaAIRL

Randomly initialize policy π_{θ} and reward network r_{ϕ} , with global weights θ_G, ϕ_G
 Obtain expert trajectories \mathcal{D}_i for each task \mathcal{T}_i
for $i = 1$ **to** N **do**
 Sample task \mathcal{T}_j with expert demonstrations \mathcal{D}_j
 Set weights of r_{ϕ} to be ϕ_G
 Set weights of π_{θ} to be θ_G
 for $n = 1$ **to** M **do**
 Train r_{ϕ}, π_{θ} using AIRL on $\mathcal{T}_j, \mathcal{D}_j$, saving weights in ϕ_n, θ_n
 end for
 $\phi_G \leftarrow \phi_G + \alpha(\phi_M - \phi_G)$
 $\theta_G \leftarrow \theta_G + \beta(\theta_M - \theta_G)$
end for
 return ϕ_G, θ_G

for its computational efficiency, and ability to extend to more complex continuous tasks, where other gradient-based methods such as MAML are not yet applicable.

In the original Meta-AIRL algorithm, the authors provide two implementation choices for the policy

- *Random*: At the beginning of each task, the policy is randomly initialized. The authors point out that this can work in relatively simple environments, but can fail in more complex tasks where the policy is unable to cover most of the state space.
- *Task-specific*: Separate policy parameters are maintained for each task. The drawback to this approach is if a task is rarely sampled, the policy is optimized for stale reward network weights and is very suboptimal for the current weights.

Our algorithm, which we call PolicyMetaAIRL, we propose an alternative to these two choices. Instead, we recommend that there be a global policy used for all tasks which is meta-learned along with the reward. Thus, we seek an initialization for the policy that can be quickly adapted to new tasks, which we transfer along with the reward network when finetuning on the test environment. The pseudocode for this procedure can be seen in Algorithm 1.

4.3. Recurrence-based Policy Transfer

We aimed to show that our idea of meta-learning the policy could be apply generally, and not to just gradient-based meta-learning methods, so as an alternative, we implemented this concept using a recurrence-based meta-learning procedure. Our algorithm is based off of RL² (Duan et al., 2016). In the original RL² procedure, at each timestep, the tuple (s, a, r, d) containing the current state s , and the previ-

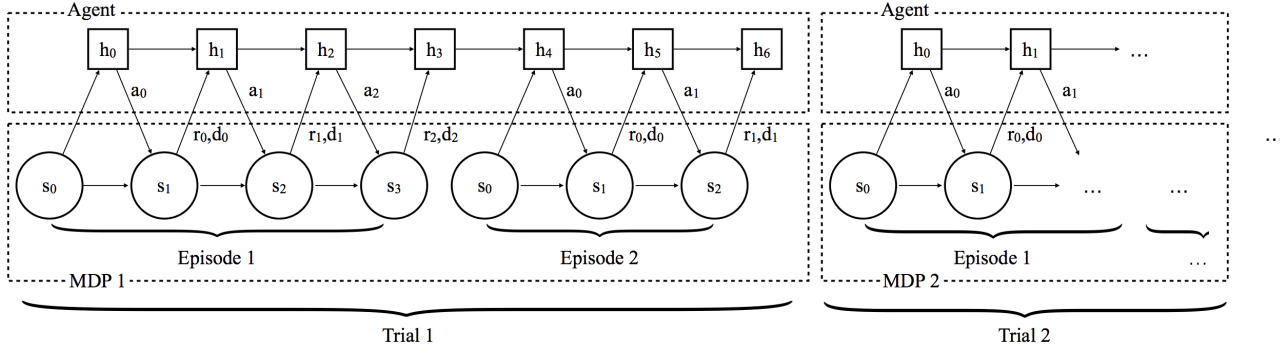


Figure 1. Agent-environment interaction in the multi-task setting for contextual policies (figure from Duan et al. (2016)).

Algorithm 2 Meta-RNN

Randomly initialize RNN policy π_θ and reward network r_ϕ , with global weights ϕ_G
 Obtain expert trajectories \mathcal{D}_i for each task \mathcal{T}_i , along with contexts
for $i = 1$ **to** N **do**
 Sample task \mathcal{T}_j with expert demonstrations \mathcal{D}_j , along with contexts
 Set weights of r_ϕ to be ϕ_G
 for $n = 1$ **to** M **do**
 Train r_ϕ, π_θ using AIRL on $\mathcal{T}_j, \mathcal{D}_j$, saving reward weights in ϕ_n
 end for
 $\phi_G \leftarrow \phi_G + \alpha(\phi_n - \phi_G)$
end for
 return ϕ_G, θ

ous action, reward and termination flag a, r, d are provided as input (along with the hidden state) to the agent to produce the next action. In principle, this black-box learning method should be able to learn a similar learning rule as the gradient-based approach. Since we are in the IRL setting, we do not have access to the true rewards r , so instead we propose just conditioning on the tuple (s, a, d) . The pseudocode for this procedure, which we call Meta-RNN, can be seen in Algorithm 2.

5. Experiments and Evaluation

5.1. Environments

5.1.1. GOAL VELOCITY ENVIRONMENTS

We experimented with an environment called PointMass, where an agent has to navigate in a continuous 2D environment. The observation space consists of the current $x \in \mathbf{R}^2$ coordinates of the agent, and the action space consists of actions $a \in \mathbf{R}^2$, where the next state is deterministic based on adding the action to the current state. The goal of the

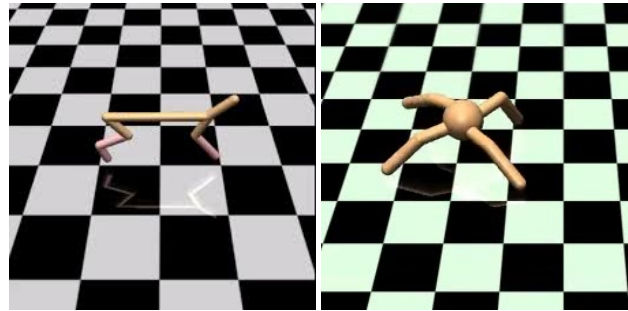


Figure 2. (a) HalfCheetah (b) Ant

task is for the agent to reach a goal velocity, and the agent receives rewards based on the difference between its current velocity and the goal velocity (and some control cost).

We also experimented with the Mujoco environments HalfCheetah and Ant (Figure 2), where similarly, the goal is to reach some target velocity.

5.1.2. FRICTION ENVIRONMENT

The Meta-RNN algorithm aims to encode some structure of the task by conditioning on past experience, but since we are in the IRL setting, it does not have access to the rewards. Therefore, it doesn't make sense to apply Meta-RNN in environments where the structure of the task cannot be inferred by the states and actions only. This is the case for the previously mentioned environments since the reward is determined by some varying goal velocity but the dynamics remain the same. Thus, to evaluate the Meta-RNN algorithm, we created an additional environment called PointMassFriction where the goal velocity is fixed, and what is varied between tasks is a friction parameter $\kappa \in (0, 1]$. In this environment, each action a_t is multiplied by κ before being added to the current state s_t to generate the next state. The reward is still based on the difference between the current velocity and the goal velocity, but in this environment, an agent theoretically should be able to learn some structure

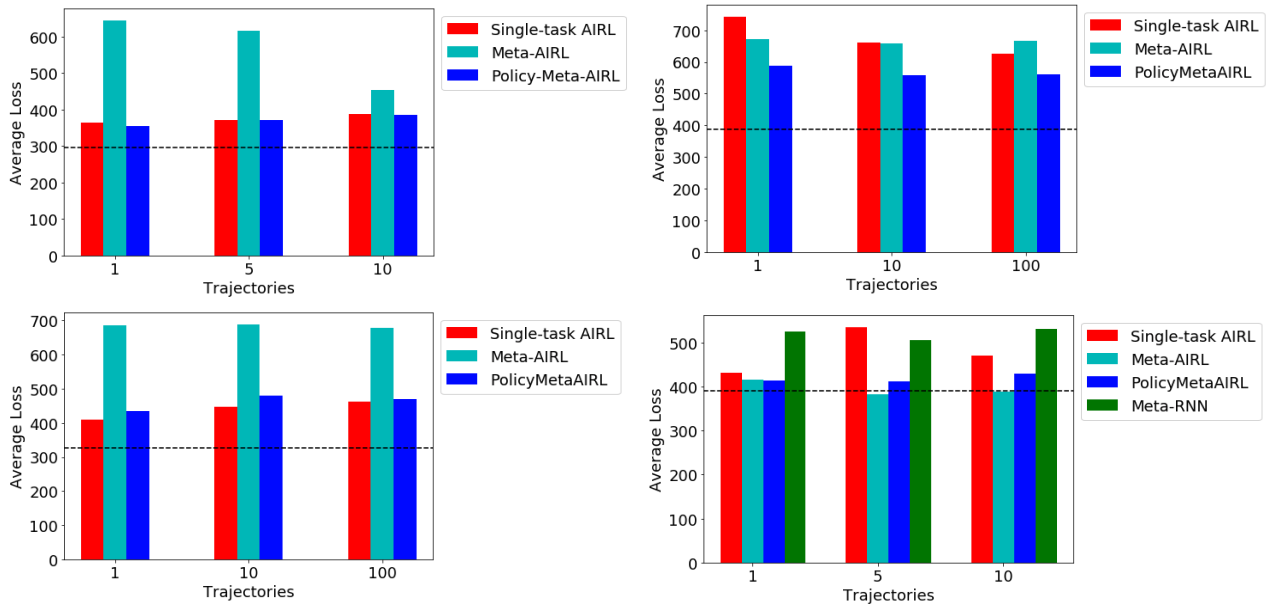


Figure 3. Average loss (lower is better) across 3 runs of: (top left) PointMass (top right) PointMassFriction. The dashed line represents the average loss of an optimal PPO planner. The best performance of 5 random seeds is shown here.

of the task just by conditioning on the past states and actions because what is changing between tasks is the dynamics.

5.2. Experiment Details

We generated expert trajectories using PPO (Schulman et al., 2017) policies trained on the ground truth reward. For the training environments, we generated 10 expert trajectories for the PointMass environment (since the task is pretty simple), and 100 expert trajectories for the Mujoco environments. For the test environments, we varied k , the number of expert trajectories available for training.

In the test environments, we reported the best average reward out of 5 random seeds. Additional details can be found in the appendix (Section 8).

5.3. Comparison to Baselines

The baselines we compared to were a single-task AIRL baseline, and also a Meta-AIRL baseline implemented in Gleave & Habryka (2018). As you can see in Fig. 3, the single-task AIRL policy is nearly optimal even after seeing only 1 expert demonstration for the PointMass and Ant environments. The PolicyMetaAIRL agent is nearly optimal as well. However, the Meta-AIRL finetune policy demonstrates suboptimal performance on the test task.

For the HalfCheetah task, single-task AIRL produces a sub-optimal policy, while PolicyMetaAIRL produces nearly optimal results. Presumably, the inductive bias from training

on the training environments enabled the PolicyMetaAIRL agent to learn an optimal policy on the test task even after only seeing one expert demonstration.

For the PointMetaFriction environment, single-task AIRL performs suboptimally, while all Meta-AIRL and PolicyMetaAIRL produce nearly optimal results. Meta-RNN seems to do the worst out of all the methods.

5.4. Analysis

When running the experiments, we tried different settings for the Meta-AIRL baseline, but could not seem to achieve similar levels of performance as the PolicyMetaAIRL and single-task AIRL algorithms in some of the environments. It appeared during training that the Meta-AIRL planner was able to achieve good performance on the training sets, but when it came to finetuning on the test set, it immediately began to perform poorly and was unable to recover the performance. The conclusion that we draw from this result is that transferring both the policy and reward network helps in preventing the reward network from overfitting on the training environments, and improves its ability to be finetuned quickly on the testing environment.

As currently implemented, it is clear that Meta-RNN is probably inferior to PolicyMetaAIRL. Our hypothesis is that this difference in performance stems from previously reported disadvantages of RL^2 relative to gradient-based methods like MAML. RNN meta-learners seem to be harder to tune, probably because they are black-box learning al-

gorithms, whereas gradient-based methods explicitly try to find initializations that can quickly adapt to new test tasks via gradient descent. This is probably exacerbated by the fact that Meta-RNN only has access to the previous states and actions, and not the rewards, which means the signal provided as input is even noisier. There are many ways in which the Meta-RNN algorithm could be improved upon however, and we mention some of these in the next section.

In addition to evaluating the performance of the policies learned with the IRL algorithms, we also were interested in whether a new policy could be learnt from scratch by training on the learned rewards. We found, however, that PPO planners reoptimized even on the single-task AIRL baseline tended to perform much worse than those trained on the ground-truth reward (see the Appendix in Section 8 for some results). This points to a limitation of current IRL algorithms: the learned rewards are often overfit to the policy generator. We found this to be the case even in the case of PolicyMetaAIRL which also meta-learns the policy along with the reward. Therefore, we agree with the conclusion in Gleave & Habryka (2018) that major improvements need to be made to current IRL algorithms in order for the performance of meta IRL algorithms to significantly increase.

6. Conclusion and Future Work

Current inverse reinforcement algorithms typically struggle to generalize beyond the specific environments they were trained on. In this paper, we introduce a meta IRL framework for jointly learning policies and rewards, and apply this framework to the two major meta-learning approaches in existence today: gradient-based and recurrence-based methods. By combining both meta-learning and inverse reinforcement learning methods, this framework should improve as both meta RL and IRL algorithms improve.

Meta inverse reinforcement learning is quite a promising area of active research, and we believe it holds great potential for the future. We hope to extend the results of this paper and improve them in the future, with two specific ideas in mind:

1. *Utilizing Past Rewards* - In the algorithms we proposed, neither the policies nor the reward networks took past rewards as input, even though this has shown to be helpful in meta RL. Although in the IRL setting we do not have access to the true rewards, it is still possible to condition on approximate rewards, such as those learned using single-task IRL, and this would be an interesting direction to explore.
2. *Meta IRL with Attention* - We hope to incorporate soft attention into our meta IRL models, similar to

the SNAIL algorithm (Mishra et al., 2017). We believe that this will enable our meta-learners to pinpoint and extract the most relevant information from each trial for aggregation, leading to additional gains in terms of performance and training efficiency.

The environments we tested our meta IRL agent on were simple, but showed promising results. Hopefully these results will influence future research on more complex tasks, in spaces such as robotics, autonomous driving, and natural language processing.

7. Acknowledgements

I would like to thank Professor Stuart Russell and Adam Gleave for productive discussions regarding this project.

I would also like to thank Yi Wu for his guidance on other research projects.

Finally, I would like to thank my parents for their unwavering love and encouragement, and also my other family and friends for all their support during my time at UC Berkeley.

References

- Amodei, D., Olah, C., Steinhardt, J., Christiano, P. F., Schulman, J., and Mané, D. Concrete problems in AI safety. *CoRR*, abs/1606.06565, 2016. URL <http://arxiv.org/abs/1606.06565>.
- Babes, M., Marivate, V. N., Subramanian, K., and Littman, M. L. Apprenticeship learning about multiple intentions. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pp. 897–904, 2011.
- Chen, Y., Hoffman, M. W., Colmenarejo, S. G., Denil, M., Lillicrap, T. P., and de Freitas, N. Learning to learn for global optimization of black box functions. *CoRR*, abs/1611.03824, 2016. URL <http://arxiv.org/abs/1611.03824>.
- Dimitrakakis, C. and Rothkopf, C. A. Bayesian multi-task inverse reinforcement learning. In *Recent Advances in Reinforcement Learning - 9th European Workshop, EWRL 2011, Athens, Greece, September 9-11, 2011, Revised Selected Papers*, pp. 273–284, 2011. doi: 10.1007/978-3-642-29946-9_27. URL https://doi.org/10.1007/978-3-642-29946-9_27.
- Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. RI²: Fast reinforcement learning via slow reinforcement learning. *CoRR*, abs/1611.02779, 2016. URL <http://arxiv.org/abs/1611.02779>.

Finn, C., Levine, S., and Abbeel, P. Guided cost learning: Deep inverse optimal control via policy optimization. In *Proceedings of the 33rd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pp. 49–58, 2016. URL <http://jmlr.org/proceedings/papers/v48/finn16.html>.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. *CoRR*, abs/1703.03400, 2017. URL <http://arxiv.org/abs/1703.03400>.

Fu, J., Luo, K., and Levine, S. Learning robust rewards with adversarial inverse reinforcement learning. *CoRR*, abs/1710.11248, 2017. URL <http://arxiv.org/abs/1710.11248>.

Gleave, A. and Habryka, O. Multi-task maximum entropy inverse reinforcement learning. *CoRR*, abs/1805.08882, 2018. URL <http://arxiv.org/abs/1805.08882>.

Mishra, N., Rohaninejad, M., Chen, X., and Abbeel, P. Meta-learning with temporal convolutions. *CoRR*, abs/1707.03141, 2017. URL <http://arxiv.org/abs/1707.03141>.

Ng, A. Y. and Russell, S. J. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*, pp. 663–670, 2000.

Nichol, A., Achiam, J., and Schulman, J. On first-order meta-learning algorithms. *CoRR*, abs/1803.02999, 2018. URL <http://arxiv.org/abs/1803.02999>.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL <http://arxiv.org/abs/1707.06347>.

Xu, K., Ratner, E., Dragan, A. D., Levine, S., and Finn, C. Learning a prior over intent via meta-inverse reinforcement learning. *CoRR*, abs/1805.12573, 2018. URL <http://arxiv.org/abs/1805.12573>.

Ziebart, B. D., Bagnell, D., and Dey, A. K. Maximum causal entropy correlated equilibria for markov games. In *Interactive Decision Theory and Game Theory, Papers from the 2010 AAAI Workshop, Atlanta, Georgia, USA, July 12, 2010*, 2010. URL <http://aaai.org/ocs/index.php/WS/AAAIW10/paper/view/1977>.

8. Appendix

8.1. Experiment Hyperparameters

To generate expert demonstrations and reoptimize a policy based on the learned rewards, we used a PPO planner with an entropy coefficient of 0.01 and a clip range of 0.1.

For the IRL agents, we used TRPO to optimize the policies using conjugate gradient descent. All feed-forward policies had 2 hidden layers with dimension 32. For the Meta-RNN planner, we first embedded the tuple using a 2-layer MLP with sizes 32 and 16, and used a GRU cell with hidden dimension 16. For the test task, we limited the interaction to 1×10^6 timesteps, which was enough in most cases for single-task AIRL to converge to a roughly optimal policy. The reward networks also had 2 layers of size 32. We used a batch size of 10,000 timesteps for all the environments.

Due to the complexity of our environments, we used the *task-specific* policy for the Meta-AIRL baseline.

8.2. Environment Details

For the PointMass and PointMassFriction environments, we used an episode length of 100. For the HalfCheetah and Ant environments, we used episode lengths of 150 and 200 respectively.

We used the custom Ant environment from the AIRL paper (Fu et al., 2017) that has modified joints for easier training.

8.3. Reoptimized Policies

We found that a PPO policy reoptimized on AIRL (single and multi-task) rewards tended to perform suboptimally when compared to an agent trained on the ground-truth rewards. Note however that the experiments for reoptimized policies in Fu et al. (2017) use a modified Ant that has some legs disabled, which is different than the environment we experimented on.

Table 1. Loss (lower is better) for reoptimized PPO policy on Ant (100-shot)

Reward	Loss
Ground-truth	326.87
Single-task AIRL	952.40
Meta-AIRL	615.07
PolicyMetaAIRL	614.07