Meta Inverse Reinforcement Learning with Contextual Recurrent Neural Networks

A report for CS 294-112: Deep Reinforcement Learning

Benjamin Kha
UC Berkeley
ben.kha@berkeley.edu

Nikhil Sharma
UC Berkeley
ennsharma@berkeley.edu

Vincent Ninh Do
UC Berkeley
ninhdo@berkeley.edu

1 Extended Abstract

Inverse Reinforcement Learning (IRL) [1] solves the problem of inferring 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 tries to learn a way to quickly adapt to new tasks. Incorporating both into one, meta inverse reinforcement learning attempts to leverage contextual information from multiple tasks with different reward functions to approximate reward functions for new tasks quickly. In this paper, we contribute a new algorithm for approaching the problem of meta IRL that is based on both the Maximum Causal Entropy IRL model [2], and the idea of of contextual RNN meta-learners.

A high-level description of our algorithm follows. Given a distribution of tasks and corresponding expert demonstrations for them, the agent interacts with a randomly selected task in each trial to meta-learn the rewards. This can be done by utilizing two recurrent neural networks: the policy network used to generate demonstrations for each task, and a reward network used to approximate the reward for every timestep in each demonstration. Both are trained concurrently on the policy-generated demonstrations labelled with the expert rewards, and the training datasets accumulate new demonstrations in every iteration. The objective function is the reward loss across entire trials, which is minimized by applying gradient descent separately for each network.

Our algorithm explicitly leverages single-task IRL algorithms, and therefore should improve as the quality of single-task IRL algorithms improve. We evaluate the results of our Meta-RNN algorithm by comparing them to prior work from the Meta-AIRL [3] algorithm and single-task AIRL in two scenarios: the performance after learning just from the training set, and that after being finetuned on the test demonstrations. Experiments on a PointMass continuous control task show that the Meta-RNN algorithm consistently outperforms the Meta-AIRL baseline in both the finetuned and non-finetuned scenarios by considerable margins over k-shot settings (for all attempted values of k), and beats the single-task IRL in at least one of the cases for each of the k-shot settings. All of this can be viewed in the graphs in Section 6. Thus, we conclude that our method can learn to quickly adapt to new environments given limited expert demonstrations.

Overall, we conclude that Meta-RNN yields significant performance gains over Meta-AIRL. We attribute our performance gain to our idea of not only transferring the reward network *but also the policy network* during test time, rather than training a policy from scratch as done by Gleave et al. [3] We believe that these results are quite promising, and hope to try additional extensions in the near future, such as utilizing soft attention in a way similar to the SNAIL algorithm. [4]

2 Introduction

Inverse reinforcement learning (IRL) attempts to model the preferences of other agents via the other agents' observed behaviors. This goal is typically realized via attempting to approximate other agents' reward functions, rather than being provided them explicitly. Over the course of the past decade, inverse reinforcement learning has gained traction in the research community in several domains ranging from artificial intelligence and machine learning to control theory and even psychology. Inverse reinforcement learning is particularly attractive because it allows leveraging machine learning to model preferences of rational agents, where using complex mathematical functions to explicitly encode reward functions has performed poorly and/or failed to yield promising results.

Standard reinforcement learning traditionally models an agent's interaction with its environment as a *Markov decision 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. As an example illustrating the utility of inverse reinforcement learning, consider an autonomous vehicle that must react to other autonomous / nonautonomous vehicles on the road around it. By collecting data about the specific driving behavior of each surrounding car, our autonomous vehicle can infer their reward functions and react more robustly.

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 versus 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. *Increase Generalizability* - Reward functions as they stand offer a very rigid way of establishing rewards in specific environments; they typically fail to generalize. Learning from demonstration however, as is done in IRL, lends itself to transfer learning when an agent is placed in new environments that differ only slightly from the original.

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 learn to estimate an output given an input; unlike their human counterparts these algorithms are unable to leverage information from previously learned tasks. Meta-learning is useful largely because it allows 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. Returning to our autonomous vehicle example, an average trip involves driving in the vicinity of anywhere from several dozen to several hundred cars. Applying meta-learning to infer these cars' reward functions would vastly improve the speed with which we can adapt to new cars we encounter on the road.

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 to support meta-inverse reinforcement learning, to quickly adapt to inferring the reward functions for new tasks and agents in environments with infinite state spaces.
- We provide an evaluation of our algorithm on a continuous control environment, and evaluate it against both single-task and multi-task baselines.

3 Related Work

Older work in IRL [5] [6] 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 [7], and Adversarial IRL [8] 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 [9] 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 [10] 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 [11] [12] 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. [13] the authors explore applying MAML on a discrete grid-based environment, while in a paper by Gleave et al. [3] the authors explore applying the Reptile and Adversarial IRL (AIRL) algorithms on continuous control tasks. In this work, we explore the use of contextual RNN meta-learners in a continuous IRL setting.

4 Background and Preliminaries

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

4.1 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 $\mathbf{s}_{t+1}$ given both the current state $\mathbf{s}_t$ and action $\mathbf{a}_t$, $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ 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:

$$R(\tau) = \sum_{t=1}^{T} \gamma^{t-1} r(\mathbf{s}_t, \mathbf{a}_t)$$

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

4.2 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, we treat entire tasks as datapoints, with a meta-training set $\{\mathcal{T}_i \,;\, i = 1, \ldots, M\}$ and meta-test set $\{\mathcal{T}_j \,;\, j = 1, ..., N\}$ both 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 $\tau_i \sim p(\tau)$. Localized training follows the gradient update rule below:

   $$\theta_i' \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}_{\tau_i}(f_\theta)$$

   These updated parameters after gradient steps on sampled individual tasks are then used to update the meta-parameter with the following update rule:

   $$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\tau_i \sim p(\tau)} \mathcal{L}_{\tau_i}(f_{\theta_i'})$$

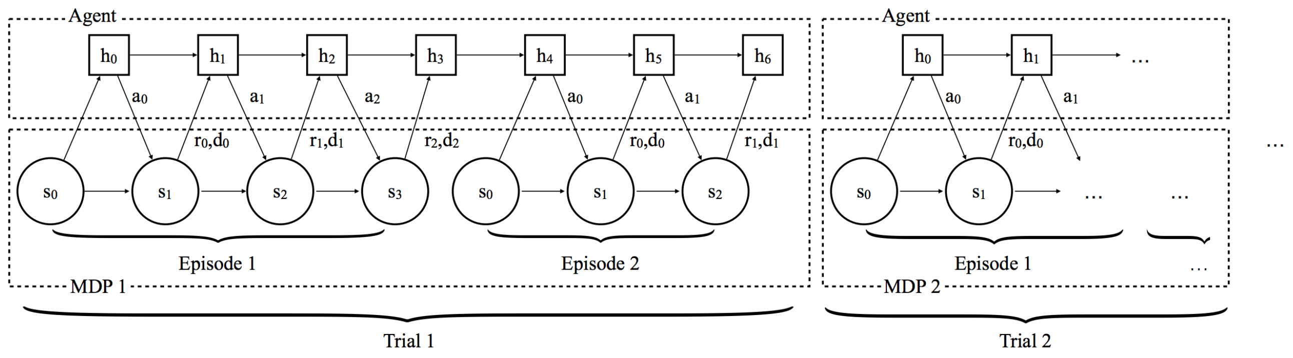
   With sufficient iterations, a meta-learner is able to use the learned meta-parameter θ to quickly adapt to new, unseen tasks.

2. *Recurrence-based Methods* - Recurrence-based methods take an entirely different approach. Rather than explicitly compute gradients and update parameters, they simple use a recurrent neural network to ingest information and "remember" attributes about the past via a hidden state. Policies trained by such learners are entirely contextual, but can be leveraged to have memory for multiple different tasks and hence quickly adaptable to new tasks similar to the ones that were trained on.

5 Meta Inverse Reinforcement Learning

5.1 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, and denote the number of episodes per trial with n.



The process of interaction between our agent and the task distribution over which it meta-learns is illustrated above. In the above example, we set the number of episodes in each trial to $n = 2$. 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. The agent then produces rewards r_t given the actions taken a_t and successor states s_{t+1}. A termination flag d_t is maintained as well, and is set to 1 if the current episode has terminated and 0 otherwise. This tuple is passed as input to the policy which is conditioned on a hidden state h_{t+1} and outputs both the next reward and hidden state, r_{t+1} and h_{t+2} respectively.

Under this framework, our goal is to minimize the loss across entire trials, rather than individual episodes; the hidden states h_t within our recurrent neural network 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.

5.2 Procedure

Our procedure works by maintaining two recurrent neural networks: a policy network π and reward network ϕ. We assume that we have some collection of tasks $\{\mathcal{T}_i\}$ with expert demonstrations for each task $\mathcal{D}_i = \{\tau_1^i, \ldots\}$. We first train single-task IRL approximators $\hat{r}_i$ on each task τ_i in our training set using the expert demonstrations $\mathcal{D}_i$. Then, for each iteration of the algorithm, we first sample a task i and collect demonstrations on task i using π, labeling each timestep in the trajectories using $\hat{r}_i$. We add these demonstrations to datasets $\mathcal{B}_\pi$ and $\mathcal{B}_\phi$, which will be used to train the two networks. One important point in this procedure is that for each timestep, we add the context of that timestep to the two datasets. A context, as mentioned before, consists of (s, a, r, d) tuples (up to some trial length) where r is not the true reward (which we do not have access to), but is instead the predicted reward $\hat{r}_i(s, a)$. The reason why have two datasets is that the contexts used for each network are "off-by-one" in the following sense: for the policy the last (s, a, r, d) tuple in the context is $(s_{t+1}, a_t, \hat{r}_i(s_t, a_t), d_t)$, which is used to generate the next action a_{t+1}. Whereas for the reward function, the last tuple in the context consists of $(s_{t+1}, a_{t+1}, \hat{r}_i(s_t, a_t), d_t)$, which is used to predict $\hat{r}_i(s_{t+1}, a_{t+1})$. This is because ϕ is trying to predict the reward given some state and action (and context), whereas π is trying to generate the next action based on a state (and context).

After collecting trajectories, we update the policy π by sampling batches from $\mathcal{B}_\pi$ and doing a policy gradient update. Likewise, we update ϕ using L gradient steps, which is just supervised learning. More explicitly, ϕ : contexts $\to \mathbb{R}$ is trying to minimize the loss

$$\mathcal{L} = \frac{1}{D}\sum_{j=1}^{D}\left(\hat{r}_j - \phi(\text{context}_j)\right)^2 \tag{1}$$

where D is the size of the batch that we sample from $\mathcal{B}_\phi$. At test time, our algorithm uses a single-task IRL reward approximator to incorporate to the context, just like during training. The pseudocode can seen in Algorithm 1 below.

Algorithm 1 Meta-RNN: RNNs for Inverse Reinforcement Learning

```
1: Randomly initialize RNN reward network $\phi$ and RNN policy $\pi$
2: For task $i = 1, \dots, N$, train $\hat{r}_i$ using single-task IRL using expert demonstrations $\mathcal{D}_i$
3: for $t = 1$ to $T$ do
4:     Sample task $i$ and collect demonstrations on task $i$ using $\pi$, labeling using rewards $\hat{r}_i$, adding to datasets $\mathcal{B}_\pi$ and $\mathcal{B}_\phi$
5:     for $n = 1$ to $K$ do
6:         Take one policy gradient step for $\pi$ by sampling a batch from $\mathcal{B}_\pi$
7:     for $n = 1$ to $L$ do
8:         Take supervised learning gradient step for $\phi$ by sampling a batch from $\mathcal{B}_\phi$
```

6 Experiments and Evaluation

We experimented with an environment called `PointMass`, where an agent has to navigate to a goal in a continuous 2D environment. The observations consists of the current $x \in \mathbb{R}^2$ coordinates of the agent, and the action space consists of actions $a \in \mathbb{R}^2$, where the next state is deterministic based on adding the action to the current state. At the start of each episode, the agent starts at the origin. The agent does not observe where the goal is, but receives rewards according to the l_2 distance from the goal. For our training environment, we generated two goals in a 2x2 square, and for the test environment, we generated a goal in a 2x2 square that was distinct from the training square. We generated expert demonstrations using a PPO [14] planner, and used single-task AIRL to generate the approximated rewards $\hat{r}_i$.

6.1 Experiment Details

For all policies, we used 3 hidden layers with a hidden layer size of 32. The PPO planners used a Gaussian MLP policy, whereas our policy π was a Gaussian RNN (GRU) policy that was optimized using PPO as well. The architecture of the reward network ϕ was identical. We used a discount rate of $\gamma = 0.90$, and a learning rate $\lambda = 0.0005$. We trained the PPO agents to generate expert demonstrations and for evaluation using 1×10^6 timesteps, and reported the best average loss out of 2 random seeds. We used a history length of 20 for our RNN networks. We generated 100 demonstrations for the two training environments, and then evaluated the performance on our algorithm on the test environment using $k = 1, 5, 100$ test demonstrations.

6.2 Comparison to Baselines

The baselines we compared to were a single-task AIRL baseline, and also a Meta-AIRL baseline implemented in [3]. For our Meta-RNN algorithm, we evaluated two options: the first is the policy/reward approximator learned just from the training set, the second is the policy/reward approximator that is *finetuned* on the test demonstrations. First we evaluated the policies that were learned during IRL. As you can see in Fig. 1, the single-task AIRL policy is nearly optimal even after seeing only 1 expert demonstration, which makes sense because the task is relatively simple (just following the path to a goal). Both the Meta-RNN and Meta-RNN finetune policies are nearly optimal as well. However, the Meta-AIRL finetune policy demonstrates suboptimal perfomance on the test task.

We also evaluated the performance of a PPO planner reoptimized based on the learned rewards. Here, for each of the k-shot settings, at least one of the Meta-RNN options beats the performance of the single-task AIRL, and in the case of 1-shot, both of them do.

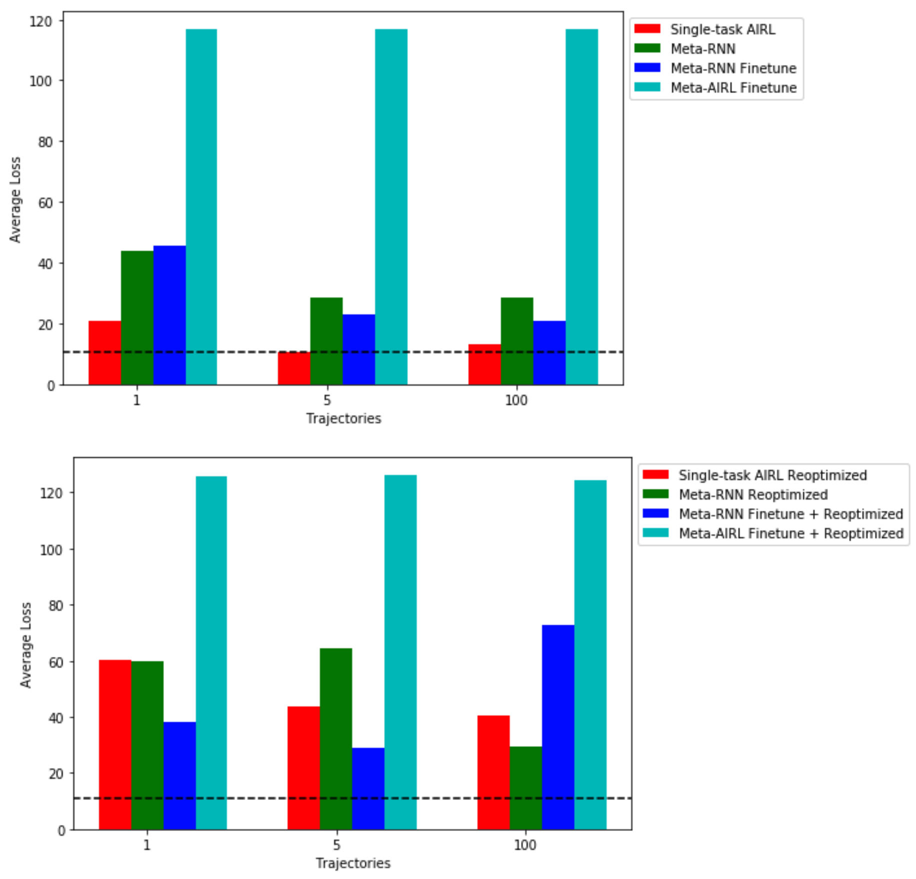


Figure 1: (a) Average loss of the policies learned along with IRL (b) Average loss of a reoptimized PPO planner based on the learned rewards. Dashed line represents the average loss of an optimal PPO planner.

6.3 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 Meta-RNN algorithm. 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. We hypothesize this is because in the Meta-AIRL implementation, when finetuning on a testing environment, only the reward network parameters are kept, while the policy is inititalized from scratch. We feel that because the policy is not transferred to the test environment, the new policy is not able to adequately bring the reward network out of the local optimum of the training environments that performs poorly on the test environment. Therefore, we recommend that those who use the Meta-AIRL algorithm try transferring both the reward network and the policy network.

One interesting result is that even the Meta-RNN that is not finetuned on the test set is still able to perform relatively well. This shows that the RNN policy and reward network are able to use the context to produce a good policy/reward approximator that transfers well to the new environment.

7 Conclusion and Future Work

Current inverse reinforcement algorithms achieve too low performance for them in practice and have high training costs. In this paper, we introduce a framework for "remembering" information through recurrent neural networks for meta-inverse reinforcement learning, attempting to mitigate these problems by improving both learning efficiency and, eventually, performance as 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. *IRL Reward Recombination* - Currently, our procedure uses single-task IRL predicted rewards during test time; however, these tend to be considerably noisy, especially when the number of expert demonstrations is low. We hope to achieve better results by computing our final reward as a trainable function of both the meta-learned IRL rewards and the single-task IRL rewards.
2. *IRL Rewards with Attention* - We hope to incorporate soft attention into our meta-IRL models, similar to the SNAIL algorithm [4]. 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.

Acknowledgements

We would like to personally thank John Schulman, Abhishek Gupta, Joshua Achiam, Michael Chang, Soroush Nasiriany, Kate Rakelly and Michael Chang, whose RNN meta-learner code from the CS 294-112 course we adapted for our experiments. Additionally, we would like to thank Sergey Levine for his lectures on reinforcement learning, which inspired the fundamental ideas underlying this project. Finally, we would like to thank Adam Gleave, whose paper [3] and implementation were very useful, and also for his suggestions for this final project.

Notes and Team Contributions

This final project was used for both CS 294-149 and CS 294-112.

1. **Benjamin Kha**: Benjamin came up with the idea for this paper and wrote the code to collect expert demonstrations, generate single-task IRL policies, and the baselines. He also wrote a lot of the code for running the different experiments, and contributed to the implementation of the RNN meta-learner.
2. **Nikhil Sharma**: Nikhil contributed to the implementation of the RNN meta-learner, and also proposed ideas on improving the methods described in the paper, and helped write and run experiments.
3. **Vincent Ninh Do**: Vincent helped run experiments and explore other ideas for the algorithm.

The source code for our algorithms and experiments can be found at https://github.com/benkha/meta_irl/tree/master.

References

[1] Andrew Y. Ng and Stuart J. Russell. 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*, pages 663–670, 2000.

[2] Brian D. Ziebart, Drew Bagnell, and Anind K. Dey. 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.

[3] Adam Gleave and Oliver Habryka. Multi-task maximum entropy inverse reinforcement learning. *CoRR*, abs/1805.08882, 2018.

[4] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. Meta-learning with temporal convolutions. *CoRR*, abs/1707.03141, 2017.

[5] Christos Dimitrakakis and Constantin A. Rothkopf. Bayesian multitask inverse reinforcement learning. In *Recent Advances in Reinforcement Learning - 9th European Workshop, EWRL 2011, Athens, Greece, September 9-11, 2011, Revised Selected Papers*, pages 273–284, 2011.

[6] Monica Babes, Vukosi N. Marivate, Kaushik Subramanian, and Michael L. Littman. 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*, pages 897–904, 2011.

[7] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 49–58, 2016.

[8] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. *CoRR*, abs/1710.11248, 2017.

[9] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *CoRR*, abs/1703.03400, 2017.

[10] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *CoRR*, abs/1803.02999, 2018.

[11] Yutian Chen, Matthew W. Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Timothy P. Lillicrap, and Nando de Freitas. Learning to learn for global optimization of black box functions. *CoRR*, abs/1611.03824, 2016.

[12] Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl\$ˆ2\$: Fast reinforcement learning via slow reinforcement learning. *CoRR*, abs/1611.02779, 2016.

[13] Kelvin Xu, Ellis Ratner, Anca D. Dragan, Sergey Levine, and Chelsea Finn. Learning a prior over intent via meta-inverse reinforcement learning. *CoRR*, abs/1805.12573, 2018.

[14] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.