Learning as a Sampling Problem

By

Bradly C Stadie

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Statistics

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Peter Bartlett, Chair
Professor Pieter Abbeel
Professor Jasjeet Sekhon

Summer 2018

Abstract

Learning as a Sampling Problem

by

Bradly C Stadie

Doctor of Philosophy in Statistics

University of California, Berkeley

Professor Peter Bartlett, Chair

The past five years have seen rapid proliferation of work on *deep learning*: learning algorithms that utilize deep neural networks for nonlinear function approximation. Although this proliferation had its roots in supervised learning, it subsequently spread to numerous other learning problems including reinforcement learning, imitation learning, meta learning, and unsupervised learning. Today, deep learning enables a variety of previously unobtainable capabilities:

1. Computers can play complex video games from raw images
2. Unsupervised learning algorithms can generate photo-realistic bedroom images from scratch without a reference
3. Robots can learn by copying other robot behavior. This imitation is quite robust and does not falter even when the demonstrated behavior is complex, abstract, or demonstrated sub-optimally.
4. The world's best translation and text to speech engines
5. One-shot image classification

Yet, in spite of myriad successes, any deep learning practitioner will quickly run into difficulties when applying many of these learning algorithms to a novel problem. Everything is hard and nothing works easily. This thesis was born out of the difficulties I experienced while working through many

problems in the fields of meta learning, reinforcement learning, and imitation learning. It is an attempt to fix many frustrating gaps in the prior art.

The first problem we consider is the exploration vs. exploitation dilemma in high-dimensional control problems with an image input space. We provide a practical algorithm to overcome the exploration vs. exploitation dilemma in this setting. This algorithm shrewdly makes use of a learned dynamics model to asses a transition's novelty. This dynamics model has the benefit of being fast to train and generalizable. We show that using this learned dynamics model to incentivise exploration leads to massive gains on several difficult Atari games.

The second problem we consider is a good deal more technical, and deals largely with fixing certain mathematical dependencies in the computational graph of meta reinforcement learning algorithms. In particular, we show that policy-gradient-like algorithms for meta learning must take care to correctly compute the gradient of the meta learner with respect to the task-specific learners. We argue that fixing this dependency issue leads to better exploratory behavior in meta learned agents.

The third problem comes from the field of imitation learning. In imitation learning, agents typically imitate other identical agents. Moreover, it is always assumed that the agent's perspective while learning is identical to the perspective of the agent it is trying to imitate. In other words, agents do not learn by watching other agents. Instead, they learn by watching an exact replica of themselves completing a task. This is a strong and impractical assumption. We remove it by introducing algorithms for third-person imitation. These algorithms allow agents to learn by watching different agents, and not just copies of themselves.

The final problem we consider comes from the field of causal inference. In causal inference, each experiment is typically treated as independent. All treatment effect estimation is thus done in a vacuum without consideration to other relevant experiments. To rectify this shortcoming, we develop the idea of deep causal transfer learning. By modifying some ideas from transfer reinforcement learning, we are able to train neural networks that can rapidly learn new treatment effects and causal relationships.

All of these problems can be derived when one takes the perspective that learning is a sampling problem. That is, many learning problems amount to analyzing a sampling distribution over a state space. For reinforcement learning, we will see that the underlying data distribution we wish to optimize over is not stationary as in supervised learning. Instead, it is sampled directly from the policy we are optimizing over. Furthermore, many common methods of optimizing this policy rely heavily on our ability to sample from the policy and do not require, for instance, derivatives with respect to the true reward. For the meta learning algorithms we consider, we will see the fact that we are optimizing over the data-generating process is an important consideration. Taking this consideration into account, we derive new meta learning gradients that account for the impact of task-specific sampling distributions on the meta sampling distribution. For imitation learning, we see that the problem is to sample data that matches some unknown expert distribution. Although we do not know the expert's true sampling distribution, we do have access to samples. We can use these samples to guide the imitator towards sampling from the correct expert distribution. Finally, we would like to one day allow agents to sample over causal relationships in their environment. This is in contrast to the present-day reality that sampling is almost always considered over low-level states or hierarchical constructs like options. The above arguments relating learning and sampling can be used to derive all of the problems we consider in this thesis. In this thesis, we will make these derivations explicit. We have thus chosen to title the thesis, 'Learning as a sampling problem.'

To Jordan Maday, David Skuban, and Jay Page

Contents

1 Opening Remarks

In this thesis, we will consider three learning problems: reinforcement learning, imitation learning, and meta learning. In all three problems, there will exist an agent in some environment E. The agent can interact with the environment by performing some action a from an underlying action space $\mathcal{A}$. This action will cause a change in E.

We typically assume the agent has access to some observation $\omega \in \Omega$ which it can analyze before acting. For example, if our agent is a robot, then its environment is the world, its observations are the measurements taken from its force and position sensors and the videos taken from its cameras, and its actions are its joint movements (e.g. moving its arm and opening or closing its gripper).

Usually, we train the agent to ingest ω and take actions that accomplish some goal within its environment. For example, we might want to train a robot to stack blocks on a table. From this desire to train our agents, the three learning problems we wish to consider arise naturally.

- **Reinforcement Learning:** There exists some reward function R which takes the current state of the environment and produces some reward. The agent learns how to take actions that maximize this reward. For example, if we want a robot to learn how to place one block on top of another block, we can give the agent a reward function which is smaller when the two blocks are far apart and larger when the two blocks come closer together. Another possible reward function is simply defined to be 1 when the two blocks are stacked and 0 otherwise. In both cases, the reward is used to reinforce a specific behavior in the agent's actions.
- **Imitation Learning:** There exists another agent in the environment that is capable of some desirable behavior. Our agent wishes to copy this behavior. For example, a robot may want to learn how to complete a new task by watching another robot or a human completing the task.
- **Meta Learning:** In their lifetime, some agents learn how to complete more than one task. As more tasks are learned, one might hope that the learning process itself accelerates. If a robot learns how to do 10 household tasks, then we would expect it to learn the 11th task more

quickly. Meta Learning seeks to make this intuition explicit by considering the problem of optimizing over the learning process. Alternative names for meta learning include: learning to learn, transfer learning, and lifelong learning.

At first glance, these three learning problems appear quite distinct. RL requires an explicit reward for learning. Imitation requires no reward but instead requires example demonstrations. Meta learning requires a multi-task setting and an outer meta-optimization signal that supervises the underlying learning problem.

However, the three problems are related. For all three problems, the agent usually has a similar observation space consisting of sensor readings. In all three cases, the agent wants to use these observations to help it learn how to visit parts of the environment that are more desirable according to some fitness function. The exact nature of this fitness function and the auxiliary information available to the agent change during each learning problem. However, the broad underlying goal always remains the same.

For all three learning problems, we proceed by biasing some sampling process $\pi(\omega) = \mathcal{D}(\mathcal{A})$ towards areas that have more density with respect to the given fitness function $R(z)$. We have called $\pi(\omega)$ a sampling process because it produces a distribution $\mathcal{D}$ over the agent's action space $\mathcal{A}$. Sampling from this distribution produces an action $a \sim \mathcal{D}(\mathcal{A})$ that affects changes in the distribution of the next state $s' \sim T(s, a)$. As a result, it is natural to identify π with a sampling distribution over the state space. Furthermore, when we say that we would like $\pi(\omega)$ to be biased towards areas with more density with respect to R, we roughly mean that we would like to train π to maximize $\int_\Omega \pi(\omega)R(\omega)d\omega$. This integral is immediately recognized as a density or expectation of R with respect to an underlying distribution π. In RL, R is an extrinsically defined reward function. In imitation learning, R is the likelihood of the state with respect to the expert's density function. In meta learning, R is a meta density that encourages π to adapt to new reward functions quickly. As we analyze each type of learning, we will highlight this viewpoint in further detail.

1.1 Contributions

In this thesis, we will be able to make contributions towards three learning problems: reinforcement learning, meta learning, and imitation learning.

In chapter 2, we will provide an interpretation of RL as a sampling problem, as motivated by REINFORCE. This interpretation naturally raises many questions about how to best ensure RL places sufficient mass across the agent's state space as the sampling distribution is being learned. These questions can subsequently be reinterpreted as the exploration vs exploitation dilemma of RL.

In chapter 3, we consider the problem of exploration in RL in earnest [119]. We propose a new method for exploration that makes use of a learned dynamics model to assess novelty. Most previous methods for exploration in RL focused on count-based methods [13, 86]. These methods relied on the ability to count state visitation frequencies. Measuring this quantity is often not tractable in complex environments. Our method replaces these visitation frequencies with evaluations of a learned dynamics model, which is tractable in high dimensional problems. Further, predicting model dynamics has better generalization properties than counting visitation frequencies (which has no generalization properties). This work was followed up by many papers from other authors, including [52, 85].

In chapter 4, we cast meta learning as a sampling problem. In particular, we will consider meta reinforcement learning from the perspective of trying to bias a sampling distribution [121]. This perspective leads us to see that meta learning gradients require an extra term to account for the impact of the original sampling process on the meta gradient. From these insights, we develop two new meta-learning algorithms: E-MAML and E-RL^2. We show that these algorithms deliver better performance on tasks where exploration is important.

In chapter 5, we will shift our focus to imitation learning. We will show that imitation learning can be viewed as a problem wherein one attempts to define a sampling process that generates data which matches the data generated by another sampling process called an expert. In this way, imitation learning is seen to be both a density matching problem and a sampling problem. This interpretation will naturally lead to a discussion on several deficiencies of imitation learning. In particular, we will see that the density matching framework fails in situations wherein there are irreconcilable differences between the agent's state space and the expert's state space. We will spend the next chapter addressing and repairing these deficiencies.

In chapter 6, we will consider a new imitation learning problem: Third-Person Imitation Learning [118]. In classical imitation learning, it is assumed that the teacher and the student are physical identical and trying to solve identical problems within identical environments. However, this is often not a reasonable assumption. Humans, for example, can infer the essence of a task and imitate this essence even when the expert's underlying environment is quite different. Third-Person Imitation Learning formalizes the problem of trying to extract the essence of a demonstrated task and apply it in a new environment. We provide a formal definition of Third-Person Imitation Learning. We also derive an algorithm for Third-Person Imitation and show that it succeeds in several environments.

In chapter 7, we make our final novel contribution when we apply transfer learning to the problem of causal inference [120]. We see that one of our proposed methods, MLRW, achieves excellent transfer on difficult causal inference problems in the field of voter encouragement. Better transfer in causal inference should lead to a better integration of reasoning in learning problems. This type of causal reasoning will allow learning agents to short-circuit many of the problems associated with learning as a sampling problem. In particular, it allows learning algorithms to achieve better generalization over both space and time, which reduces the necessary complexity of their search paths as they try and define sampling distributions that lead to desirable behavior.

In chapter 8, we end this thesis with a discussion of the difficulties I repeatedly faced when writing this thesis. We suggest several potential solutions.

2 Deep Reinforcement Learning as a Sampling Problem

2.1 The RL objective

Consider an agent living in some environment E. We can train an agent to complete a task within its environment by making it solve a Markov decision process (MDP). A Markov decision process can be thought of as a puzzle which provides the necessary information for an agent to acquire a new behavior in its environment. It is usually represented as a tuple $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \rho_0, \gamma, T)$ where $\mathcal{S}$ is a set containing all possible world states the agent can encounter, $\mathcal{A}$ is an set containing all possible actions an agent can use, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_+$ is a probability distribution which provides transition probabilities between states when sampled, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function, $\rho_0 : \mathcal{S} \to \mathbb{R}_+$ is an initial state distribution, $\gamma \in [0, 1]$ a discount factor, and T is the problem horizon.

In RL, the objective is to solve a given MDP. This is done by defining an object called a policy (usually denoted by π) that tells the agent which action it should optimally take in any given state. We can express the policy in mathematical notation as $\pi_\theta : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_+$. The subscript θ indicates π is parametrized by θ. We want to train these parameters so that π prescribes the optimal action in any given state s.

In practice, the policy π is often not deterministic. Instead, π usually defines a distribution over the action space conditioned on the present state. Recalling that the actions prescribed by the policy cause a transition to a new state, we see that the policy implicitly defines a sampling distribution over the state space. This induced distribution can be written as $p_\theta(\tau) = \Pi_{t=0}^{T-1} p(s_{t+1}|s_t, a_t)\pi_\theta(a_t|\tau_t)$, where $\tau_i = (s_0, a_0, \ldots, s_i, a_i)$. Our goal is to train this sampling distribution to favor areas with higher reward. Put another way, we want π to produce trajectories with a high reward density. Mathematically, we want to maximize the expected reward under the induced distribution p.

Putting everything together, the RL objective is for π to maximizes the expected discounted sum of future rewards incurred. Let us write the expected discounted sum of rewards induced by π in a mathematically rigorous way as

$$\begin{aligned}\eta(\pi_\theta) &= \mathbb{E}_{p_\theta(\tau)}\left[\sum_{t=0}^{T} \gamma^t r(s_t)\right] \\ &= \mathbb{E}_{p_\theta(\tau)}[R(\tau)]\end{aligned}$$

where $s_0 \sim \rho_0(s_0)$, $a_t \sim \pi_\theta(a_t|s_t)$, $s_{t+1} \sim \mathcal{P}(s_{t+1}|s_t, a_t)$, and $R(\tau) = \sum_{t=0}^{T} \gamma^t r(s_t)$. Thus, maximizing $\eta(\pi_\theta)$ requires we maximize an expectation. The REINFORCE algorithm provides one such algorithm for computing a tractable form of this gradient [139].

2.2 REINFORCE

We want to maximize the expression

$$\eta(\pi_\theta) = \mathbb{E}_{p_\theta(\tau)}[R(\tau)]$$

The most straightforward way to achieve this is with stochastic gradient ascent. This algorithm requires that we compute the gradient of $\eta(\pi_\theta)$ with respect to θ. To accomplish this, we can expand the expectation into an integral and then pull the gradient under the integral.

$$\begin{aligned}\nabla_\theta \eta(\pi_\theta) &= \nabla_\theta \mathbb{E}_{p_\theta(\tau)}[R(\tau)] \\ &= \nabla_\theta \int R(\tau) p_\theta(\tau) d\tau \\ &= \int R(\tau) \nabla_\theta p_\theta(\tau) d\tau\end{aligned}$$

Recalling the derivative of $\log(x)$ is $\frac{dx}{x}$, we can use the classical trick of multiplying the expression under the integral by $1 = \frac{p_\theta(\tau)}{p_\theta(\tau)}$.

$$\begin{aligned}
\int R(\tau)\nabla_\theta p_\theta(\tau)d\tau &= \int R(\tau)\nabla_\theta p_\theta(\tau) \cdot \frac{p_\theta(\tau)}{p_\theta(\tau)} d\tau \\
&= \int R(\tau)\left(\frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)}\right) p_\theta(\tau)d\tau \\
&= \int R(\tau)\nabla_\theta \log p_\theta(\tau) p_\theta(\tau)d\tau \\
&= \mathbb{E}_{p_\theta(\tau)}\left[R(\tau)\nabla_\theta \log p_\theta(\tau)\right] \\
&= \mathbb{E}_{p_\theta(\tau)}\left[R(\tau)\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|\tau_t)\right]
\end{aligned}$$

The above expression requires only that we compute the gradient of $\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|\tau_t)$, which is tractable though quite inefficient. Critically, this algorithm only requires that we have the ability to sample from π_θ. It does not require that we directly compute gradient with respect to the reward R, which would be quite difficult. Instead, it computes gradients of the distribution induced by the policy with respect to θ and then weights those gradients by R. Once again, we see that we are using this algorithm to essentially randomly search for areas with high reward density.

2.3 The Exploration Problem

Policy gradient methods such as REINFORCE are slow. This is because they do not directly compute gradients with respect to the true underlying reward function, but instead rely on treating learning as a sampling problem and sampling from π_θ to compute gradients.

When some part of the state space is difficult to repeatedly sample from, this area will often be overlooked even if it presents a higher reward density. Furthermore, once the algorithm finds an area of the state space with better than random reward density, it will tend to adjust its sampling weights too much towards this area in response. This results in the policy visiting one area repeatedly. Consequently, it exploits only a fractional part of the state space before it had sufficient time to inspect or randomly search other areas of the state space. This is referred to as the exploration vs exploitation problem. We would like our algorithms to be more exploratory when defining a sampling distribution over the state space. In the next chapter, we provide a novel method for achieving efficient exploration. This method learns a dynamics model to help the sampling distribution determine which areas of state space have been sufficiently accounted for. Thus, it helps the policy ensure it samples a sufficiently eclectic portion of the environment's states.

3 Incentivizing Exploration with Deep Predictive Models

Achieving efficient and scalable exploration in complex domains poses a major challenge in reinforcement learning. While Bayesian and PAC-MDP approaches to the exploration problem offer strong formal guarantees, they are often impractical in higher dimensions due to their reliance on enumerating the state-action space. Hence, exploration in complex domains is often performed with simple epsilon-greedy methods. In this work, we consider the challenging Atari games domain, which requires processing raw pixel inputs and delayed rewards. We evaluate several more sophisticated exploration strategies, including Thompson sampling and Boltzman exploration, and propose a new exploration method based on assigning exploration bonuses from a concurrently learned model of the system dynamics. By parameterizing our learned model with a neural network, we are able to develop a scalable and efficient approach to exploration bonuses that can be applied to tasks with complex, high-dimensional state spaces. In the Atari domain, our method provides the most consistent improvement across a range of games that pose a major challenge for prior methods. In addition to raw game-scores, we also develop an AUC-100 metric for the Atari Learning domain to evaluate the impact of exploration on this benchmark. This work was published as [119].

3.1 Introduction

In reinforcement learning (RL), agents acting in unknown environments face the exploration versus exploitation tradeoff. Without adequate exploration, the agent might fail to discover effective control strategies, particularly in complex domains. Both PAC-MDP algorithms, such as MBIE-EB [123], and Bayesian algorithms such as Bayesian Exploration Bonuses (BEB) [60] have managed this tradeoff by assigning exploration bonuses to novel states. In these methods, the novelty of a state-action pair is derived from the number of times an agent has visited that pair. While these approaches offer strong formal guarantees, their requirement of an enumerable representation of the agent's environment renders them impractical for large-scale tasks. As such, exploration in large RL tasks is still most often performed using simple heuristics, such as the epsilon-greedy strategy [77], which can be inadequate in more complex settings.

In this work, we evaluate several exploration strategies that can be scaled up to complex tasks with high-dimensional inputs. Our results show that Boltzman exploration and Thompson sampling significantly improve on the naïve epsilon-greedy strategy. However, we show that the biggest and most consistent improvement can be achieved by assigning exploration bonuses based on a learned model of the system dynamics with learned representations. To that end, we describe a method that learns a state representation from observations, trains a dynamics model using this representation concurrently with the policy, and uses the misprediction error in this model to asses the novelty of each state. Novel states are expected to disagree more strongly with the model than those states that have been visited frequently in the past, and assigning exploration bonuses based on this disagreement can produce rapid and effective exploration.

Using learned model dynamics to assess a state's novelty presents several challenges. Capturing an adequate representation of the agent's environment for use in dynamics predictions can be accomplished by training a model to predict the next state from the previous ground-truth state-action pair. However, one would not expect pixel intensity values to adequately capture the salient features of a given state-space. To provide a more suitable representation of the system's state space, we propose a method for encoding the state space into lower dimensional domains. To achieve sufficient generality and scalability, we modeled the system's dynamics with a deep neural network. This allows for on-the-fly learning of a model representation that can easily be trained in parallel to learning a policy.

Our main contribution is a scalable and efficient method for assigning exploration bonuses in large RL problems with complex observations, as well as an extensive empirical evaluation of this approach and other simple alternative strategies, such as Boltzman exploration and Thompson sampling. Our approach assigns model-based exploration bonuses from learned representations and dynamics, using only the observations and actions. It can scale to large problems where Bayesian approaches to exploration become impractical, and we show that it achieves significant improvement in learning speed on the task of learning to play Atari games from raw images [14]. Our approach achieves state-of-the-art results on a number of games, and achieves particularly large improvements for games on which human players strongly outperform prior methods. Aside from achieving a high final score, our method also achieves substantially faster learning. To evaluate the speed of the learning process, we propose the AUC-100 benchmark to evaluate learning progress on the Atari domain.

3.2 Preliminaries

We consider an infinite-horizon discounted Markov decision process (MDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \rho_0, \gamma)$, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ a finite set of actions, $\mathcal{P} : \mathcal{S} \times A \times \mathcal{S} \to \mathbb{R}$ the transition probability distribution, $\mathcal{R} : \mathcal{S} \to \mathbb{R}$ the reward function, ρ_0 an initial state distribution, and $\gamma \in (0, 1)$ the discount factor. We are interested in finding a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ that maximizes the expected reward over all time. This maximization can be accomplished using a variety of reinforcement learning algorithms.

In this work, we are concerned with online reinforcement learning wherein the algorithm receives a tuple (s_t, a_t, s_{t+1}, r_t) at each step. Here, $s_t \in \mathcal{S}$ is the previous state, $a_t \in \mathcal{A}$ is the previous action, $s_{t+1} \in \mathcal{S}$ is the new state, and r_t is the reward collected as a result of this transition. The reinforcement learning algorithm must use this tuple to update its policy and maximize long-term reward and then choose the new action a_{t+1}. It is often insufficient to simply choose the best action based on previous experience, since this strategy can quickly fall into a local optimum. Instead, the learning algorithm must perform exploration. Prior work has suggested methods that address the exploration problem by acting with “optimism under uncertainty.” If one assumes that the reinforcement learning algorithm will tend to choose the best action, it can be encouraged to visit state-action pairs that it has not frequently seen by augmenting the reward function to deliver a bonus for visiting novel states. This is accomplished with the augmented reward function

$$\mathcal{R}_{\text{Bonus}}(s, a) = \mathcal{R}(s, a) + \beta \mathcal{N}(s, a), \tag{1}$$

where $\mathcal{N}(s, a) : \mathcal{S} \times \mathcal{A} \to [0, 1]$ is a novelty function designed to capture the novelty of a given state-action pair. Prior work has suggested a variety of different novelty functions e.g., [123, 60] based on state visitation frequency.

While such methods offer a number of appealing guarantees, such as near-Bayesian exploration in polynomial time [60], they require a concise, often discrete representation of the agent's state-action space to measure state visitation frequencies. In our approach, we will employ function approximation and representation learning to devise an alternative to these requirements.

3.3 Model Learning For Exploration Bonuses

We would like to encourage agent exploration by giving the agent exploration bonuses for visiting novel states. Identifying states as novel requires we supply some representation of the agent's state space, as well as a mechanism to use this representation to assess novelty. Unsupervised learning methods offer one promising avenue for acquiring a concise representation of the state with a good similarity metric. This can be accomplished using dimensionality reduction, clustering, or graph-based techniques [46, 12]. In our work, we draw on recent developments in representation learning with neural networks, as discussed in the following section. However, even with a good learned state representation, maintaining a table of visitation frequencies becomes impractical for complex tasks. Instead, we learn a model of the task dynamics that can be used to assess the novelty of a new state.

Formally, let $\sigma(s)$ denote the encoding of the state s, and let $\mathcal{M}_\phi : \sigma(\mathcal{S}) \times \mathcal{A} \to \sigma(\mathcal{S})$ be a dynamics predictor parameterized by ϕ. $\mathcal{M}_\phi$ takes an encoded version of a state s at time t and the agent's action at time t and attempts to predict an encoded version of the agent's state at time $t+1$. The parameterization of $\mathcal{M}$ is discussed further in the next section.

For each state transition (s_t, a_t, s_{t+1}), we can attempt to predict $\sigma(s_{t+1})$ from $(\sigma(s_t), a_t)$ using our predictive model $\mathcal{M}_\phi$. This prediction will have some error

$$e(s_t, a_t) = \|\sigma(s_{t+1}) - \mathcal{M}_\phi(\sigma(s_t), a_t)\|_2^2. \tag{2}$$

Let $\overline{e_T}$, the normalized prediction error at time T, be given by $\overline{e_T} := \frac{e_T}{\max_{t \le T}\{e_t\}}$. We can assign a novelty function to (s_t, a_t) via

$$\mathcal{N}(s_t, a_t) = \frac{\bar{e}_t(s_t, a_t)}{t * C} \tag{3}$$

where $C > 0$ is a decay constant. We can now realize our augmented reward function as

$$\mathcal{R}_{Bonus}(s, a) = \mathcal{R}(s, a) + \beta \left(\frac{\bar{e}_t(s_t, a_t)}{t * C} \right) \tag{4}$$

This approach is motivated by the idea that, as our ability to model the dynamics of a particular state-action pair improves, we have come to understand the state better and hence its novelty is lower. When we don't understand the state-action pair well enough to make accurate predictions, we assume that more knowledge about that particular area of the model dynamics is needed and hence a higher novelty measure is assigned.

Using learned model dynamics to assign novelty functions allows us to address the exploration versus exploitation problem in a non-greedy way. With an appropriate representation $\sigma(s_t)$, even when we encounter a new state-action pair (s_t, a_t), we expect $\mathcal{M}_\phi(\sigma(s_t), a_t)$ to be accurate so long as enough similar state-action pairs have been encountered.

Our model-based exploration bonuses can be incorporated into any online reinforcement learning algorithm that updates the policy based on state, action, reward tuples of the form (s_t, a_t, s_{t+1}, r_t), such as Q-learning or actor-critic algorithms. Our method is summarized in Algorithm 1. At each step, we receive a tuple $(s_t, a_t, s_{t+1}, \mathcal{R}(s_t, a_t))$ and compute the Euclidean distance between the encoded state $\sigma(s_{t+1})$ to the prediction made by our model $\mathcal{M}_\phi(\sigma(s_t), a_t)$. This is used to compute the exploration-augmented reward $\mathcal{R}_{Bonus}$ using Equation (4). The tuples $(s_t, a_t, s_{t+1}, \mathcal{R}_{\text{Bonus}})$ are stored in a memory bank Ω at the end of every step. Every step, the policy is updated. [1] Once per epoch, corresponding to 50000 observations in our implementation, the dynamics model $\mathcal{M}_\phi$ is updated to improve its accuracy. If desired, the representation encoder σ can also be updated at this time. We found that retraining σ once every 5 epochs to be sufficient.

This approach is modular and compatible with any representation of σ and $\mathcal{M}$, as well as any reinforcement learning method that updates its policy based on a continuous stream of observation, action, reward tuples. Incorporating exploration bonuses does make the reinforcement learning task

[1] In our implementation, the memory bank Ω is used to retrain the RL algorithm via experience replay once per epoch (50000 steps). Hence, 49999 of these policy updates will simply do nothing.

Algorithm 1 Reinforcement learning with model prediction exploration bonuses

```
1: Initialize $\max_e = 1$, EpochLength, $\beta$, $C$
2: for iteration $t$ in $T$ do
3:     Observe $(s_t, a_t, s_{t+1}, \mathcal{R}(s_t, a_t))$
4:     Encode the observations to obtain $\sigma(s_t)$ and $\sigma(s_{t+1})$
5:     Compute $e(s_t, a_t) = \|\sigma(s_{t+1}) - \mathcal{M}_\phi(\sigma(s_t), a_t)\|_2^2$ and $\bar{e}(s_t, a_t) = \frac{e(s_t,a_t)}{\max_e}$.
6:     Compute $\mathcal{R}_{Bonus}(s_t, a_t) = \mathcal{R}(s, a) + \beta \left(\frac{\bar{e}_t(s_t,a_t)}{t*C}\right)$
7:     if $e(s_t, a_t) > \max_e$ then
8:         $\max_e = e(s_t, a_t)$
9:     end if
10:    Store $(s_t, a_t, \mathcal{R}_{bonus})$ in a memory bank $\Omega$.
11:    Pass $\Omega$ to the reinforcement learning algorithm to update $\pi$.
12:    if $t$ mod EpochLength == 0 then
13:        Use $\Omega$ to update $\mathcal{M}$.
14:        Optionally, update $\sigma$.
15:    end if
16: end for
17: return optimized policy $\pi$
```

nonstationary, though we did not find this to be a major issue in practice, as shown in our experimental evaluation. In the following section, we discuss the particular choice for σ and $\mathcal{M}$ that we use for learning policies for playing Atari games from raw images.

3.4 Deep Learning Architectures

Though the dynamics model $\mathcal{M}_\phi$ and the encoder σ from the previous section can be parametrized by any appropriate method, we found that using deep neural networks for both achieved good empirical results on the Atari games benchmark. In this section, we discuss the particular networks used in our implementation.

3.4.1 Autoencoders

The most direct way of learning a dynamics model is to directly predict the state at the next time step, which in the Atari games benchmark corresponds to the next frame's pixel intensity values. However, directly predicting these pixel intensity values is unsatisfactory, since we do not expect pixel intensity to capture the salient features of the environment in a robust way. In our experiments, a dynamics model trained to predict raw frames exhibited extremely poor behavior, assigning exploration bonuses in near equality at most time steps, as discussed in our experimental results section.

To overcome these difficulties, we seek a function σ which encodes a lower dimensional representation of the state s. For the task of representing Atari frames, we found that an autoencoder could be used to successfully obtain an encoding function σ and achieve dimensionality reduction and feature extraction [49]. Our autoencoder has 8 hidden layers, followed by a Euclidean

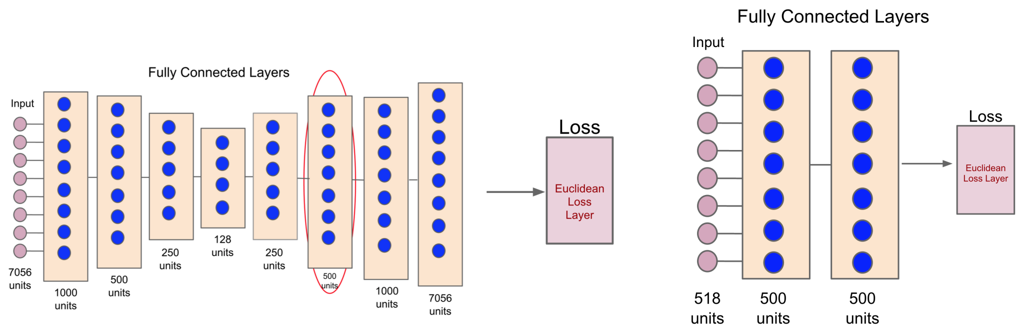


Figure 1: Left: Autoencoder used on input space. The circle denotes the hidden layer that was extracted and utilized as input for dynamics learning. Right: Model learning architecture.

loss layer, which computes the distance between the output features and the original input image. The hidden layers are reduced in dimension until maximal compression occurs with 128 units. After this, the activations are decoded by passing through hidden layers with increasingly large size. We train the network on a set of 250,000 images and test on a further set of 25,000 images. We compared two separate methodologies for capturing these images.

1. **Static AE:** A random agent plays for enough time to collect the re-

quired images. The auto-encoder σ is trained offline before the policy learning algorithm begins.

2. **Dynamic AE:** Initialize with an epsilon-greedy strategy and collect images and actions while the agent acts under the policy learning algorithm. After 5 epochs, train the auto encoder from this data. Continue to collect data and periodically retrain the auto encoder in parallel with the policy training algorithm.

We found that the reconstructed input achieves a small but non-trivial residual on the test set regardless of which auto encoder training technique is utilized, suggesting that in both cases it learns underlying features of the state space while avoiding overfitting.

To obtain a lower dimensional representation of the agent's state space, a snapshot of the network's first six layers is saved. The sixth layer's output (circled in figure one) is then utilized as an encoding for the original state space. That is, we construct an encoding $\sigma(s_t)$ by running s_t through the first six hidden layers of our autoencoder and then taking the sixth layers output to be $\sigma(s_t)$. In practice, we found that using the sixth layer's output (rather than the bottleneck at the fifth layer) obtained the best model learning results. We discuss this result further below.

3.4.2 On Auto Encoder Layer Selection

Recall that we trained an auto-encoder to encode the game's state space. We then trained a predictive model on the next auto-encoded frame rather than directly training on the pixel intensity values of the next frame. To obtain the encoded space, we ran each state through an eight layer auto-encoder for training and then utilized the auto-encoder's sixth layer as an encoded state space. We chose to use the sixth layer rather than the bottleneck fourth layer because we found that, over 20 iterations of Seaquest at 100 epochs per iteration, using this layer for encoding delivered measurably better performance than using the bottleneck layer. The results of that experiment are presented below.

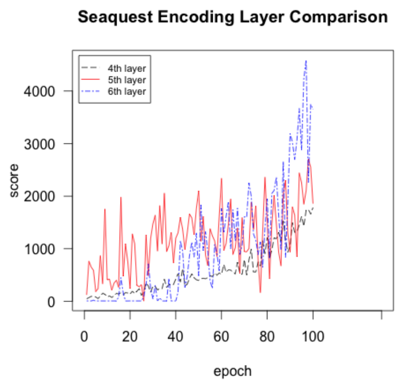


Figure 2: Game score averaged over 20 Seaquest iterations with various choices for the state-space encoding layer. Notice that choosing the sixth layer to encode the state space significantly outperformed the bottleneck layer.

3.4.3 Model Learning Architecture

Equipped with an encoding σ, we can now consider the task of predicting model dynamics. For this task, a much simpler two layer neural network $\mathcal{M}_\phi$ suffices. $\mathcal{M}_\phi$ takes as input the encoded version of a state s_t at time t along with the agent's action a_t and seeks to predict the encoded next frame $\sigma(s_{t+1})$. Loss is computed via a Euclidean loss layer regressing on the ground truth $\sigma(s_{t+1})$. We find that this model initially learns a representation close to the identity function and consequently the loss residual is similar for most state-action pairs. However, after approximately 5 epochs, it begins to learn more complex dynamics and consequently better identify novel states. We evaluate the quality of the learned model below.

3.5 Related Work

Exploration is an intensely studied area of reinforcement learning. Many of the pioneering algorithms in this area, such as R-Max [18] and E^3 [55], achieve efficient exploration that scales polynomially with the number of parameters in the agent's state space (see also [56, 115]). However, as the size of state spaces increases, these methods quickly become intractable. A number of prior methods also examine various techniques for using models and prediction to incentivize exploration [117, 72, 37, 4]. However, such methods typically operate directly on the transition matrix of a discrete MDP, and do not provide for a straightforward extension to very large or continuous spaces, where function approximation is required. A number of prior methods have also been proposed to incorporate domain-specific factors to improve exploration. Doshi-Velez et al. [30] proposed incorporating priors into policy optimization, while Lang et al. [66] developed a method specific to relational domains. Finally, Schmidhuber et al. have developed a curiosity driven approach to exploration which uses model predictors to aid in control [106].

Several exploration techniques have been proposed that can extend more readily to large state spaces. Among these, methods such as C-PACE [88] and metric-E^3 [53] require a good metric on the state space that satisfies the assumptions of the algorithm. The corresponding representation learning issue has some parallels to the representation problem that we address by using an autoencoder, but it is unclear how the appropriate metric for the prior methods can be acquired automatically on tasks with raw sensory input, such as the Atari games in our experimental evaluation. Methods based on Monte-Carlo tree search can also scale gracefully to complex domains [43], and indeed previous work has applied such techniques to the task of playing Atari games from screen images [44]. However, this approach is computationally very intensive, and requires access to a generative model of the system in order to perform the tree search, which is not always available in online reinforcement learning. On the other hand, our method readily integrates into any online reinforcement learning algorithm.

Finally, several recent papers have focused on driving the Q value higher. In [35], the authors use network dropout to perform Thompson sampling. In Boltzman exploration, a positive probability is assigned to any possible action according to its expected utility and according to a temperature parameter [23]. Both of these methods focus on controlling Q values rather than model-based exploration. A comparison to both is provided in the next section.

3.6 Experimental Results

3.6.1 Learning Curves, AUC 100 Scores, and Discussion

We evaluate our approach on 14 games from the Arcade Learning Environment [14]. The task consists of choosing actions in an Atari emulator based on raw images of the screen. Previous work has tackled this task using Q-learning with epsilon-greedy exploration [77], as well as Monte Carlo tree search [44] and policy gradient methods [110]. We use Deep Q Networks (DQN) [77] as the reinforcement learning algorithm within our method, and compare its performance to the same DQN method using only epsilon-greedy exploration, Boltzman exploration, and a Thompson sampling approach.

The results for 14 games in the Arcade Learning Environment are presented in Table 1. We chose those games that were particularly challenging for prior methods and ones where human experts outperform prior learning methods. We evaluated two versions of our approach; using either an autoencoder trained in advance by running epsilon-greedy Q-learning to collect data (denoted as "Static AE"), or using an autoencoder trained concurrently with the model and policy on the same image data (denoted as "Dynamic AE"). Table 1 also shows results from the DQN implementation reported in previous work, along with human expert performance on each game [77]. Note that our DQN implementation did not attain the same score on all of the games as prior work due to a shorter running time. Since we are primarily concerned with the rate of learning and not the final results, we do not consider this a deficiency. To directly evaluate the benefit of including exploration bonuses, we compare the performance of our approach primarily to our own DQN implementation, with the prior scores provided for reference.

In addition to raw-game scores, and learning curves, we also analyze our results on a new benchmark we have named Area Under Curve 100 (AUC-100). For each game, this benchmark computes the area under the game-score learning curve (using the trapezoid rule to approximate the integral). This area is then normalized by 100 times the score maximum game score achieved in [77], which represents 100 epochs of play at the best-known levels. This metric more effectively captures improvements to the game's learning rate and does not require running the games for 1000 epochs as in [77]. For this reason, we suggest it as an alternative metric to raw game-score.

Bowling The policy without exploration tended to fixate on a set pattern of knocking down six pins per frame. When bonuses were added, the dynamics learner quickly became adept at predicting this outcome and was thus encouraged to explore other release points.

Frostbite This game's dynamics changed substantially via the addition of extra platforms as the player progressed. As the dynamics of these more complex systems was not well understood, the system was encouraged to visit them often (which required making further progress in the game).

Seaquest A submarine must surface for air between bouts of fighting sharks. However, if the player resurfaces too soon they will suffer a penalty with effects on the game's dynamics. Since these effects are poorly understood by the model learning algorithm, resurfacing receives a high exploration bonus and hence the agent eventually learns to successfully resurface at the correct time.

Q*bert Exploration bonuses resulted in a lower score. In Q*bert, the background changes color after level one. The dynamics predictor is unable to quickly adapt to such a dramatic change in the environment and consequently, exploration bonuses are assigned in near equality to almost every state that is visited. This negatively impacts the final policy.

Learning curves for each of the games are shown in Figure (3). Note that both of the exploration bonus algorithms learn significantly faster than epsilon-greedy Q-learning, and often continue learning even after the epsilon-greedy strategy converges. All games had the inputs normalized according to [77] and were run for 100 epochs (where one epoch is 50,000 time steps). Between each epoch, the policy was updated and then the new policy underwent 10,000 time steps of testing. The results represent the average testing score across three trials after 100 epoch each.

The results show that more nuanced exploration strategies generally improve on the naive epsilon greedy approach, with the Boltzman and Thompson

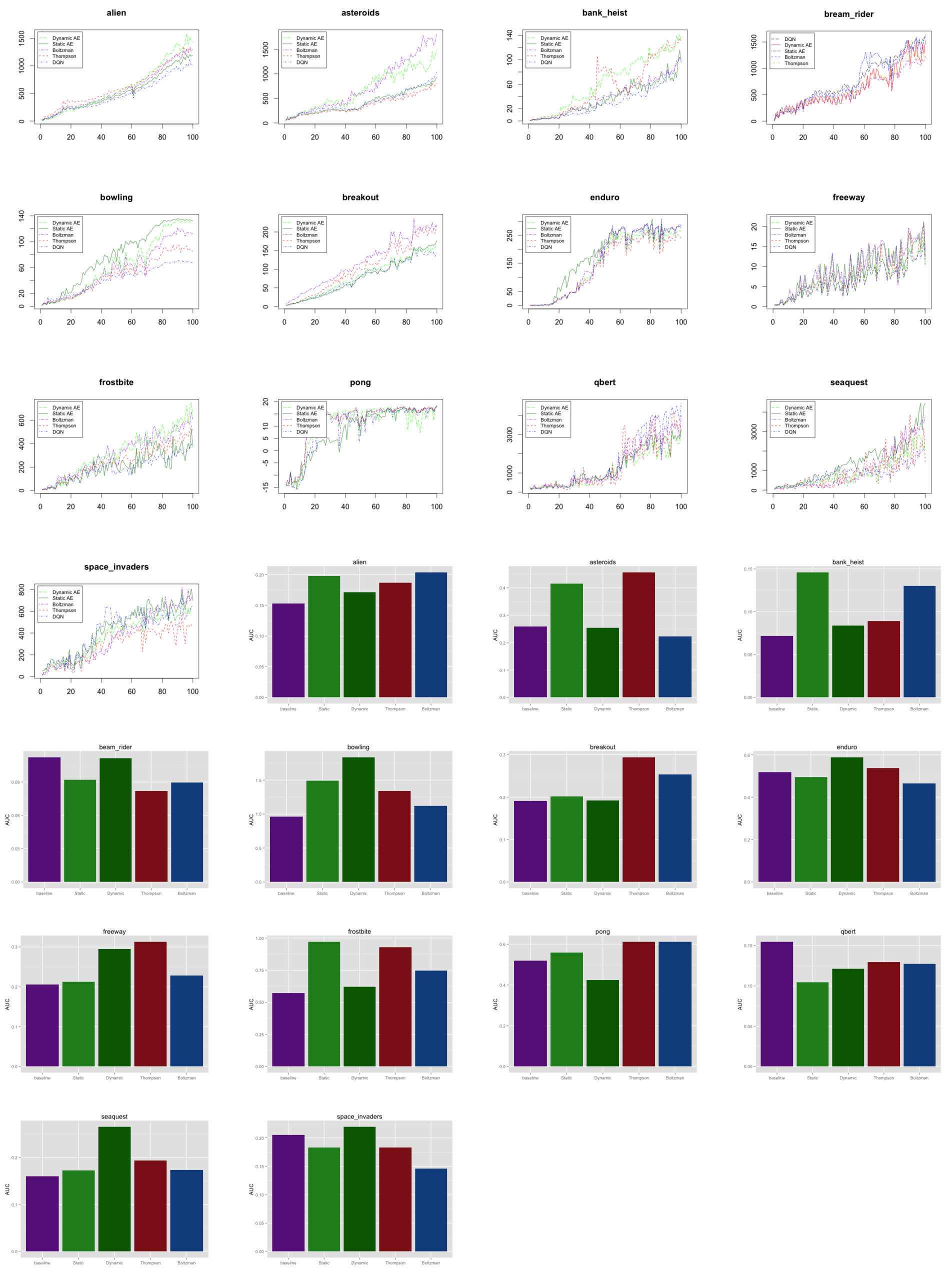


Figure 3: Full learning curves and AUC-100 scores for all Atari games. We present the raw AUC-100 scores below in tabular format.

Game	DQN 100 epochs	Exploration Static AE 100 epochs	Exploration Dynamic AE 100 epochs	Boltzman Exploration 100 epochs	Thompson Sampling 100 epochs	DQN [77] 1000 epochs	Human Expert [77]
Alien	1018	**1436**	1190	1301	1322	3069	6875
Asteroids	1043	**1486**	939	1287	812	1629	13157
Bank Heist	102	**131**	95	101	129	429.7	734.4
Beam Rider	1604	1520	**1640**	1228	1361	6846	5775
Bowling	68.1	130	**133**	113	85.2	42.4	154.8
Breakout	146	162	178	219	**222**	401.2	31.8
Enduro	281	264	277	**284**	236	301.8	309.6
Freeway	10.5	10.5	12.5	**13.9**	12.0	30.3	29.6
Frostbite	369	**649**	380	605	494	328.3	4335
Montezuma	0.0	0.0	0.0	0	0	0.0	4367
Pong	17.6	**18.5**	18.2	18.2	18.2	18.9	9.3
Q*bert	**4649**	3291	3263	4014	3251	10596	13455
Seaquest	2106	2636	**4472**	3808	1337	5286	20182
Space Invaders	634	649	**716**	697	459	1976	1652

Table 1: A comparison of maximum scores achieved by different methods. Static AE trains the state-space auto encoder on 250000 raw game frames prior to policy optimization (raw frames are taken from random agent play). Dynamic AE retrains the auto encoder after each epoch, using the last 250000 images as a training set. Note that exploration bonuses help us to achieve state of the art results on Bowling and Frostbite. Each of these games provides a significant exploration challenge. Bolded numbers indicate the best-performing score among our experiments. Note that this score is sometimes lower than the score reported for DQN in prior work as our implementation only one-tenth as long as in [77].

sampling methods achieving the best results on three of the games. However, exploration bonuses achieve the fastest learning and the best results most consistently, outperforming the other three methods on 7 of the 14 games in terms of AUC-100.

3.6.2 On the Quality of the Learned Model Dynamics

Evaluating the quality of the learned dynamics model is somewhat difficult because the system is rewarded achieving higher error rates. A dynamics model that converges quickly is not useful for exploration bonuses. Nevertheless, when we plot the mean of the normalized residuals across all games and all trials used in our experiments, we see that the errors of the learned dynamics models continually decrease over time. The mean normalized residual after 100 epochs is approximately half of the maximal mean achieved. This suggests that each dynamics model was able to correctly learn properties of underlying dynamics for its given game.

Game	DQN	Exploration Static AE	Exploration Dynamic AE	Boltzman Exploration	Thompson Sampling
Alien	0.153	0.198	0.171	0.187	0.204
Asteroids	0.259	0.415	0.254	0.456	0.223
Bank Heist	0.0715	0.1459	0.089	0.089	0.1303
Beam Rider	0.1122	0.0919	0.1112	0.0817	0.0897
Bowling	0.964	1.493	1.836	1.338	1.122
Breakout	0.191	0.202	0.192	0.294	0.254
Enduro	0.518	0.495	0.589	0.538	0.466
Freeway	0.206	0.213	0.295	0.313	0.228
Frostbite	0.573	0.971	0.622	0.928	0.746
Montezuma	0.0	0.0	0.0	0	0
Pong	0.52	0.56	0.424	0.612	0.612
Q*bert	0.155	0.104	0.121	0.13	0.127
Seaquest	0.16	0.172	0.265	0.194	0.174
Space Invaders	0.205	0.183	0.219	0.183	0.146

Table 2: AUC-100 is computed by comparing the area under the game-score learning curve for 100 epochs of play to the area under of the rectangle with dimensions 100 by the maximum DQN score the game achieved in [77]. The integral is approximated with the trapezoid rule. This more holistically captures the games learning rate and does not require running the games for 1000 epochs as in [77]. For this reason, we suggest it as an alternative metric to raw game-score.

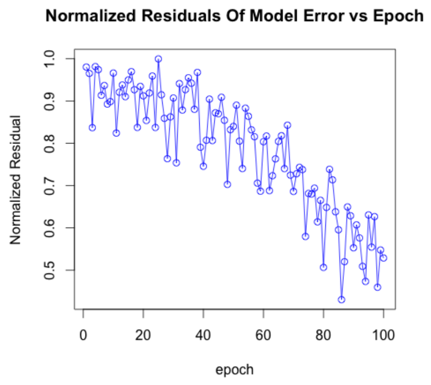


Figure 4: Normalized dynamics model prediction residual across all trials of all games. Note that the dynamics model is retrained from scratch for each trial.

3.7 Conclusion

In this work, we evaluated several scalable and efficient exploration algorithms for reinforcement learning in tasks with complex, high-dimensional observations. Our results show that a new method based on assigning exploration bonuses most consistently achieves the largest improvement on a range of challenging Atari games, particularly those on which human players outperform prior learning methods. Our exploration method learns a model of the dynamics concurrently with the policy. This model predicts a learned representation of the state, and a function of this prediction error is added to the reward as an exploration bonus to encourage the policy to visit states with high novelty.

One of the limitations of our approach is that the misprediction error metric assumes that any misprediction in the state is caused by inaccuracies in the model. While this is true in deterministic environments, stochastic dynamics violate this assumption. An extension of our approach to stochastic systems requires a more nuanced treatment of the distinction between *stochastic* dynamics and *uncertain* dynamics, which we hope to explore in future work. Another intriguing direction for future work is to examine how the learned dynamics model can be incorporated into the policy learning process, beyond just providing exploration bonuses. This could in principle enable substantially faster learning than purely model-free approaches.

Since this chapter's initial publication in 2015, there has been a flurry of follow-up work. These results include but are not limited to [52, 87, 85].

4 The Importance of Sampling in Meta-Reinforcement Learning

We interpret meta-reinforcement learning as the problem of learning how to quickly find a good sampling distribution in a new environment. This interpretation leads to the development of two new meta-reinforcement learning algorithms: E-MAML and E-RL^2. Results are presented on a new environment we call 'Krazy World': a difficult high-dimensional gridworld which is designed to highlight the importance of correctly differentiating through sampling distributions in meta-reinforcement learning. Further results are presented on a set of maze environments. We show E-MAML and E-RL^2 deliver better performance than baseline algorithms on both tasks. This work was published as [121].

4.1 Introduction

Reinforcement learning can be thought of as a procedure wherein an agent bias its sampling process towards areas with higher rewards. This sampling process is embodied as the policy P, which is responsible for outputting an action (a) conditioned on past environmental states ($\{s\}$). Such action affects changes in the distribution of the next state $s' \sim T(s, a)$. As a result, it is natural to identify the policy P with a sampling distribution over the state space.

This perspective highlights a key difference between reinforcement learning and supervised learning: In supervised learning, the data is sampled from a fixed set. The i.i.d. assumption assumes that the model does not affect the underlying distribution. In reinforcement learning however, the very goal in the problem setup is to learn a policy that could manipulate the state samples to the agent's advantage.

This property of RL to affect the data distribution during its own learning process is particularly salient in the field of meta-reinforcement learning (meta RL) [107, 108, 102, 101, 104, 61, 105, 138, 129]. Meta RL goes by many different names: learning to learn, multi-task learning, lifelong learning, transfer learning, etc. The goal however, is usually the same– we wish to train agents that in some way optimize over, and consequently improve upon, their own learning process by training over a multitude of episodes, tasks, and environments. As a consequence of these improvements to the learning process, we expect the agent to solve new tasks more quickly than a regular RL agent that starts from scratch.

This problem definition induces an interesting consequence: during meta-learning, we are no longer under the obligation to optimize for maximal reward during training. Instead, we are optimizing to obtain a sampling process that maximally informs the meta-learner how it should adapt to new environments. In the context of gradient based algorithms, this means that a principled approach in learning an optimal sampling strategy is to differentiate the meta RL agent's per-task sampling process with respect to the goal of maximizing the meta-learners reward. To the best of our knowledge, such a scheme is hitherto unexplored.

In this work, we derive an algorithm for gradient-based meta-learning that explicitly considers the deviates of the per-task sampling distributions with respect to the expected future returns produced by its parent meta-learner. We show that this algorithm is closely related to the recently proposed MAML algorithm [33]. For reasons that will become clear later, we call this algorithm E-MAML. Inspired by this algorithm, we develop less principled extension of RL^2 that we call E-RL^2. We show that our algorithms outperform baselines on a high-dimensional dynamically-changing set of grid-world environments that are challenging even for state-of-the-art RL algorithms. To verify we are not over-fitting to this environment, we also present results on a set of maze environments.

4.2 Problem Formulation and Algorithms

4.2.1 Reinforcement Learning Notation

Let $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \rho_0, \gamma, T)$ represent a discrete-time finite-horizon discounted Markov decision process (MDP). The elements of M have the following definitions: $\mathcal{S}$ is a state set, $\mathcal{A}$ an action set, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_+$ a transition probability distribution, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ a reward function, $\rho_0 : \mathcal{S} \to \mathbb{R}_+$ an initial state distribution, $\gamma \in [0, 1]$ a discount factor, and T the horizon. We will sometimes speak of M having a loss function $\mathcal{L}$ rather than reward function r. All we mean here is that $\mathcal{L}(s) = -r(s)$ In a classical reinforcement learning setting, we optimize to obtain a set of parameters θ which maximize the expected discounted return under the policy $\pi_\theta : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}_+$. That is, we optimize to obtain θ that maximizes $\eta(\pi_\theta) = \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{T} \gamma^t r(s_t)]$, where $s_0 \sim \rho_0(s_0)$, $a_t \sim \pi_\theta(a_t|s_t)$, and $s_{t+1} \sim \mathcal{P}(s_{t+1}|s_t, a_t)$.

4.2.2 The Meta Reinforcement Learning Objective

In meta reinforcement learning, we consider a family of MDPs $\mathcal{M} = \{M_i\}_{i=1}^N$ which comprise a distribution of tasks. The goal of meta RL is to find a policy π_θ and paired update method U such that, given $M_i \sim \mathcal{M}$, we have $\pi_{U(\theta)}$ solves M_i quickly. The word *quickly* is key: By quickly, we mean orders of magnitude more sample efficient than simply solving M_i with policy gradient or value iteration methods from scratch. For example in an environment where policy gradients require over 100,000 samples to produce good returns, an ideal meta RL algorithm should solve these tasks by collecting less than 10 trajectories. The assumption is that if an algorithm can solve a problem with so few samples, then it might be 'learning to learn.' That is, the agent is not learning how to master a particular task but rather how to quickly adapt to new ones. This objective can be written cleanly as

$$\min_\theta \sum_{M_i} \mathbb{E}_{\pi_{U(\theta)})} \left[\mathcal{L}_{M_i}\right] \tag{5}$$

This objective is similar to the one that appears in MAML [33], which we will discuss further below. In MAML, U is chosen to be the stochastic gradient descent operator parameterized by the task.

4.2.3 Fixing the Sampling Problem with E-MAML

We can expand the expectation from (5) into the integral

$$\int R(\tau)\pi_{U(\theta)}(\tau)\mathrm{d}\tau \tag{6}$$

It is true that the objective (5) can be optimized by taking a derivative of this integral with respect to θ and carrying out a standard REINFORCE style analysis to obtain a tractable expression for the gradient [139]. However, this decision is sub-optimal.

Our key insight is to recall the sampling process interpretation of RL. In this interpretation, the policy π_θ implicitly defines a sampling process over the state space. Under this interpretation, meta RL tries to learn a strategy for quickly generating good per-task sampling distributions. For this learning process to work, it needs to receive a signal from each per-task sampling distribution which measures its propensity to positively impact the meta-learning process. Such a term does not make an appearance when directly optimizing (5). Put more succinctly, **directly optimizing (5) will not account for the impact of the original sampling distribution π_θ on the future rewards** $R(\tau), \tau \sim \pi_{U(\theta)}$. Concretely, we would like to account for the fact that the samples $\bar{\tau}$ drawn under π_θ will impact the final returns $R(\tau)$ by influencing the initial update $U(\theta, \bar{\tau})$. Making this change will allow initial samples $\bar{\tau} \sim \pi_\theta$ to be reinforced by the expected future returns after the sampling update $R(\tau)$. Under this scheme, the initial samples $\bar{\tau}$ are encouraged to cover the state space enough to ensure that the update $U(\theta)$ is maximally effective.

Including this dependency can be done by writing the modified expectation as

$$\iint R(\tau)\pi_{U(\theta)}(\tau)\pi_\theta(\bar{\tau})\mathrm{d}\bar{\tau}\mathrm{d}\tau$$

This provides an unbiased expression for computing the gradient which correctly accounts for the dependence on the initial sampling distribution.

We now find ourselves wishing to find a tractable expression for the gradient of (3). This can be done quite smoothly by applying the product rule under the integral sign and going through the REINFORCE style derivation twice to arrive at a two term expression

$$\begin{aligned}
&\frac{\partial}{\partial\theta}\iint R(\tau)\pi_{U(\theta)}(\tau)\pi_\theta(\bar{\tau})\mathrm{d}\bar{\tau}\mathrm{d}\tau \\
=&\iint R(\tau)\left[\pi_\theta(\bar{\tau})\frac{\partial}{\partial\theta}\pi_{U(\theta)}(\tau) + \pi_{U(\theta)}(\tau)\frac{\partial}{\partial\theta}\pi_\theta(\bar{\tau})\right]\mathrm{d}\bar{\tau}\mathrm{d}\tau \\
\approx&\frac{1}{T}\sum_{i=1}^{T} R(\tau^i)\frac{\partial}{\partial\theta}\log\pi_{U(\theta)}(\tau^i) + \frac{1}{T}\sum_{i=1}^{T} R(\tau^i)\frac{\partial}{\partial\theta}\log\pi_\theta(\bar{\tau}^i) \quad \Bigg|_{\substack{\bar{\tau}^i \sim \pi_\theta \\ \tau^i \sim \pi_{U(\theta)}}}
\end{aligned}$$

The term on the left is precisely the original MAML algorithm [33]. This term encourages the agent to take update steps U that achieve good final rewards. The second term encourages the agent to take actions such that the eventual meta-update yields good rewards (crucially, it does not try and exploit the reward signal under its own trajectory $\bar{\tau}$). In our original derivation of this algorithm, we felt this term would afford the the policy the opportunity to be more exploratory, as it will attempt to deliver the maximal amount of information useful for the future rewards $R(\tau)$ without worrying about its own rewards $R(\bar{\tau})$. Since we felt this algorithm augments MAML by adding in an exploratory term, we called it E-MAML. At present, the validity of this interpretation remains an open question.

For the experiments presented in this work, we will assume that the operator U that is utilized in MAML and E-MAML is stochastic gradient descent. However, many other interesting choices exist.

4.2.4 Choices for the Operator U

During our derivation of E-MAML, we made an implicit choice in that the inner update operator U takes the form of SGD. However, this is but one of many eligible operators where the exploration is directed by the task reward. Consider the path-integral formulation of the expected reward in E-MAML

$$\mathcal{L}(\theta_0) = -\iint R(\tau) \cdot \pi(a_i|s_i, U(\theta_0, R(\bar{\tau}), \bar{\tau} \sim \pi(a|\theta_0))) \cdot \pi(\bar{a}_i\,|\bar{s}_i, \theta_0)\, \mathrm{d}\tau \mathrm{d}\bar{\tau}. \tag{7}$$

We can make it easier to tell the role of U by re-writing the action probability π and U in type notation where

$$\hat{\pi}(\theta) : \Rightarrow \{(\tau) : \mathbb{R}^+ \Rightarrow \pi(a_t|s_t, \theta)\} \tag{8}$$

$$\text{and}\;\; \hat{U}(\theta_0) : \Theta \Rightarrow U\Big(\theta_0, R(\tau), \tau \sim \pi(a|\theta_0)\Big). \tag{9}$$

$\hat{\pi}$ returns a function on trajectories $\tau \in \mathcal{T}$, whereas $\hat{U}$ takes in two more variables from its closure, and returns an updated $\theta \in \Theta$. Using the Polish notation @ applied from-left-to-right, the surrogate loss could be written as

$$\mathcal{L}(\theta_0) = -\iint R(\tau) \cdot \left(\hat{\pi} \circ \hat{U}\right) @\theta_0 @\bar{\tau} \\ \cdot \hat{\pi} @\theta_0 @\bar{\tau} \, \mathrm{d}\tau \mathrm{d}\bar{\tau}. \tag{10}$$

In this notation, it becomes clear that you could pick any arbitrary function for the inner-update operator $\hat{U}$. Under this context, the update operator of choice in E-MAML (and similarly MAML) is an instance from a more general class of exploratory operators $\mathcal{O}$:

$$\hat{U}_{\mathrm{MAML}}(\theta_0) : \Theta \Rightarrow \mathrm{SGD}\left(\mathcal{L}_{\mathrm{PPO}}, \tau(\pi(\theta_0)), \theta_0\right) \tag{11}$$

where the class of operators it belongs to is $\hat{U}_{\mathrm{MAML}} \in \mathcal{O} : \{(\Theta) :\Rightarrow \Theta\}$

So what is this policy gradient update operator? One could argue that U_{MAML} is a "reward-maximizing" exploration strategy. The reward collected during the exploration phase is used to compute the gradient direction for θ_0, which is in-turn used to update $\theta_0 \to \theta$.

Then it becomes immediately obvious there are a few other operators that could also be plugged in:

- *random exploration* on a single task, where θ is perturbed randomly.
- *natural gradient* on a single task, where such perturbation is scaled with the inverse fisher information.
- *perpendicular to gradient*: Another extreme is to update the initial parameter θ_0 perpendicular to the gradient direction.
- *ε-greedy*: A middle ground between these extremes is a ε-greedy exploration strategy, where for ε of the time the per-task exploration is done randomly, whereas for the rest of the time $(1 - \varepsilon)$ the gradient direction is picked.
- *Meta-learned Exploratory Operator* where the inner-update operator U is learned to optimality to a specific class of tasks. See [102, 101].
- *Distraction-free Learning* where the sign of the explorative log probability is flipped. This should discourage exploratory behavior. This could be useful in a life-long learning agent during test time, where exploration is extremely costly.

4.2.5 E-RL^2

RL^2 optimizes (5) by feeding multiple rollouts from multiple different MDPs into an RNN. The hope is that the RNN hidden state update $h_t = C(x_t, h_{t-1})$, will learn to act as the update function U. Then, performing a policy gradient update on the RNN will correspond to optimizing the meta objective (5).

We can write the RL^2 update rule more explicitly in the following way. Suppose L represents an RNN. Let $\text{Env}_k(a)$ be a function that takes an action, uses it to interact with the MDP representing task k, and returns the next observation o [2] , reward r, and a termination flag d. Then we have

$$x_t = [o_{t-1}, a_{t-1}, r_{t-1}, d_{t-1}] \tag{12}$$

$$[a_t, h_{t+1}] = L(h_t, x_t) \tag{13}$$

$$[o_t, r_t, d_t] = \text{Env}_k(a_t) \tag{14}$$

To train this RNN, we sample N MDPs from $\mathcal{M}$ and obtain k rollouts for each MDP by running the MDP through the RNN as above. We then compute a policy gradient update to move the RNN parameters in a direction which maximizes the returns over the k trials performed for each MDP.

Inspired by our derivation of E-MAML, we attempt to make RL^2 better account for the impact of its initial sampling distribution on its final returns. However, we will take a slightly different approach. Call the rollouts that help account for the impact of this initial sampling distribution S-rollouts. Call rollouts that do not account for this dependence G-rollouts. For each MDP M_i, we will sample p S-rollouts and $k - p$ G-rollouts. During an S-rollout, the forward pass through the RNN will receive all information. However, during the backwards pass, the rewards contributed during S-rollouts will be set to zero. For example, if there is one S-rollout and one G-rollout, then we would proceed as follows. During the forward pass, the RNN will receive all information regarding rewards for both episodes. During the backwards

[2]RL^2 works well with POMDP's because the RNN is good at system-identification. This is the reason why we chose to use o as in “observation” instead of s for “state” in this formulation.

pass, the returns will be computed as

$$R(x_i) = \sum_{j=i}^{T} \gamma^j r_j \cdot \chi_E(x_j) \tag{15}$$

Where χ_E is an indicator that returns 0 if the episode is exploratory and 1 otherwise. This return, and not the standard sum of discounted returns, is what is used to compute the policy gradient. The hope is that zeroing the return contributions from S-rollouts will encourage the RNN to account for the impact of casting a wider sampling distribution on the final meta-reward. That is, during S-rollouts the policy will take actions which may not lead to immediate rewards but rather to the RNN hidden weights that perform better system identification. This system identification will in turn lead to higher rewards in later episodes.

4.3 Experiments

4.3.1 Krazy World Environment

To test the importance of correctly differentiating through the sampling process in meta reinforcement learning, we engineer a new environment known as Krazy World. A successful meta learning agent will first need to identify the different tile types, color palette, and dynamics. This environment is challening even for state-of-the-art RL algorithms. In this environment, it is essential that meta-updates account for the impact of the original sampling distribution on the final meta-updated reward. Without accounting for this impact, the agent will not receive the gradient of the per-task episodes with respect to the meta-update. But this is precisely the gradient that encourages the agent to quickly learn how to correctly identify parts of the environment. See below for a full description of the environment.

4.3.2 Further Krazy World Details

We find this environment challenging for even state-of-the-art RL algorithms. For each environment in the test set, we optimize for 5 million steps using the Q-Learning algorithm from Open-AI baselines. This exercise delivered a

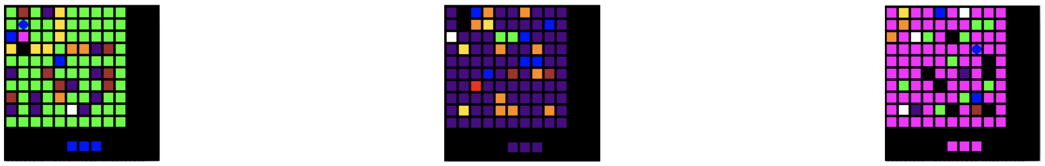


Figure 5: Three example worlds drawn from the task distribution. A good agent should first complete a successful system identification before exploiting. For example, in the leftmost grid the agent should identify the following: 1) the orange squares give +1 reward, 2) the blue squares replenish energy, 3) the gold squares block progress, 4) the black square can only be passed by picking up the pink key, 5) the brown squares will kill it, 6) it will slide over the purple squares. The center and right worlds show how these dynamics will change and need to be re-identified every time a new task is sampled.

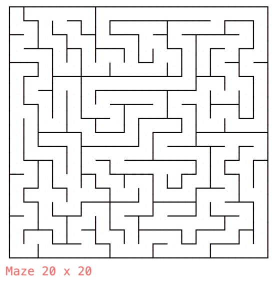


Figure 6: One example maze environment rendered in human readable format. The agent attempts to find a goal within the maze.

mean score of 1.2 per environment, well below the human baselines score of 2.7. The environment has the following challenging features:

1. **8 different tile types**: ***Goal squares*** provide +1 reward when retrieved. The agent reaching the goal does not cause the episode to terminate, and there can be multiple goals. ***Ice squares*** will be skipped over in the direction the agent is transversing. ***Death squares*** will kill the agent and end the episode. ***Wall squares*** act as a wall, impeding the agent's movement. ***Lock squares*** can only be passed once the agent has collected the corresponding key from a key square. ***Teleporter squares*** transport the agent to a different teleporter square on the map. ***Energy squares*** provide the agent with additional energy. If the agent runs out of energy, it can no longer move. The agent

proceeds normally across ***normal squares***.

2. **Ability to randomly swap color palette**: The color palette for the grid can be randomly permuted, changing the color that corresponds to each of the different tile types. The agent will thus have to identify the new system to achieve a high score. Note that in representations of the gird wherein basis vectors are used rather than images to describe the state space, each basis vector will correspond to a tile type, and permuting the colors will correspond to permuting the types of tiles these basis vectors represent. We prefer to use the basis vector representation in our experiments, as it is more sample efficient.

3. **Ability to randomly swap dynamics**: The game's dynamics can be altered. The most naive alteration simply permutes the player's inputs and corresponding actions (issuing the command for down moves the player up etc). More complex dynamics alterations allow the agent to move multiple steps at a time in arbitrary directions, making the movement more akin to that of chess pieces.

4. **Local or Global Observations**: The agent's observation space can be set to some fixed number of squares around the agent, the squares in front of the agent, or the entire grid. Observations can be given as images or as a grid of basis vectors. For the case of basis vectors, each element of the grid is embedded as a basis vector that corresponds to that tile type. These embeddings are then concatenated together to form the observation proper. We will use local observations.

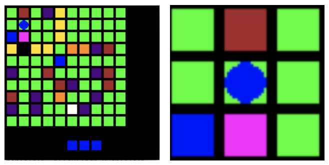


Figure 7: A comparison of local and global observations for the Krazy World environment. In the local mode, the agent only views a 3×3 grid centered about itself. In global mode, the agent views the entire environment.

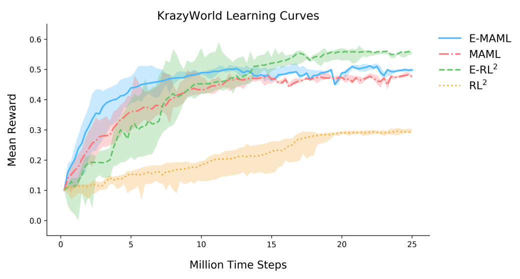


Figure 8: Meta learning curves on Krazy World. We see that E-RL^2 is at achieves the best final results, but has the highest initial variance. Crucially, E-MAML converges faster than MAML, although both algorithms do manage to converge. RL^2 has relatively poor performance and high variance. A random agent achieves a score of around 0.05 on this task.

4.3.3 Mazes

A collection of maze environments. The agent is placed at a random square within the maze and must learn to navigate the twists and turns to reach the goal square. A good exploratory agent will spend some time learning the maze's layout in a way that minimizes repetition of future mistakes. The mazes are not rendered, and consequently this task is done with state space only. The mazes are 20×20 squares large.

4.3.4 Results

In this section we present the following experimental results.

1. Learning curves on Krazy World and mazes.
2. The gap between the agent's initial performance on new environments and its performance after updating. A good meta learning algorithm will have a large gap after updating. A standard RL algorithm will have virtually no gap after only one update.
3. Three heuristic metrics for how much of the state space an algorithm manages to correctly identify.

When plotting learning curves in Figure 8 and Figure 9, the Y axis is the reward obtained after training at test time on a set of 64 held-out test environments. The X axis is the total number of environmental time-steps the algorithm has used for training. Every time the environment advances forward by one step, this count increments by one. This is done to keep the time-scale consistent across meta-learning curves.

For Krazy World, learning curves are presented in Figure 8. E-MAML and E-RL^2 have the best final performance. E-MAML has the steepest initial gains for the first 10 million time-steps. Since meta-learning algorithms are often very expensive, having a steep initial ascent is quite valuable. Around 14 million training steps, E-RL^2 passes E-MAML for the best performance. By 25 million time-steps, E-RL^2 has converged. MAML delivers comparable final performance to E-MAML. However, it takes it much longer to obtain this level of performance. Finally, RL^2 has comparatively poor performance on this task and very high variance. When we manually examined the RL^2 trajectories to figure out why, we saw the agent consistently finding a single goal square and then refusing to explore any further. The additional metrics presented below seem consistent with this finding.

Learning curves for mazes are presented in Figure 9. Here, the story is different than Krazy World. RL^2 and E-RL^2 both perform better than MAML and E-MAML. We suspect the reason for this is that RNNs are able to leverage memory, which is more important in mazes than in Krazy World. This environment carries a penalty for hitting the wall, which MAML and E-MAML discover quickly. However, it takes E-RL^2 and RL^2 much longer to discover this penalty, resulting in worse performance at the beginning of training. MAML delivers worse final performance and typically only learns how to avoid hitting the wall. RL^2 and E-MAML sporadically solve mazes. E-RL^2 manages to solve many of the mazes.

When examining meta learning algorithms, one important metric is the update size after one learning episode. Our meta learning algorithms should have a large gap between their initial policy, which is largely exploratory, and their updated policy, which should often solve the problem entirely. For MAML, we look at the gap between the initial policy and the policy after one policy gradient step (see figure 11 for information on further gradient steps). For RL^2, we look at the results after three exploratory episodes, which give the RNN hidden state h sufficient time to update. Note that three is the number of exploratory episodes we used during training as well. This metric shares similarities with the Jump Start metric considered in prior literature

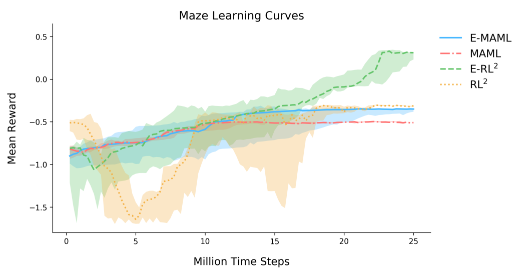


Figure 9: Meta learning curves on mazes. Figure 10 shows each curve in isolation, making it easier to discern their individual characteristics. E-MAML and E-RL^2 perform better than their counterparts.

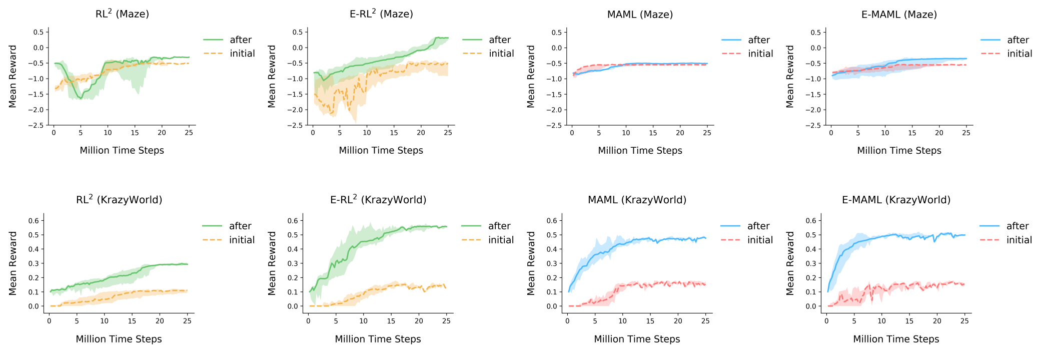


Figure 10: Gap between initial performance and performance after one update. All algorithms show some level of improvement after one update. This suggests meta learning is working, because normal policy gradient methods learn nothing after one update.

[127]. These gaps are presented in figure 10.

Finally, in Figure 12 we see three heuristic metrics desigend to measure a meta-learners capacity for system identification. First, we consider the fraction of tile types visited by the agent at test time. A good agent should visit and identify many different tile types. Second, we consider the number of times an agent visits a death tile at test time. Agents that are efficient at identification should visit this tile type exactly once and then avoid it. More naive agents will run into these tiles repeatedly, dying repetedly and instilling a sense of pity in onlookers. Finally, we look at how many goals the agent reaches. RL^2 tends to visit fewer goals. Instead, it finds one goal and exploits it. Overall, our suggested algorithms achieve better performance under these metrics.

4.3.5 Number of Gradient Steps vs. Return MAML and E-MAML

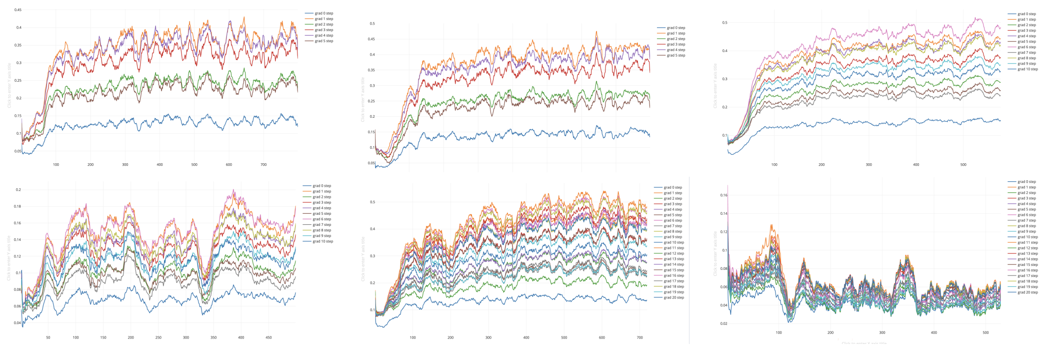


Figure 11: MAML on the right and E-MAML on the left. A look at the number of gradient steps at test time vs reward on the Krazy World environment. Both MAML and E-MAML do not typically benefit from seeing more than one gradient step at test time. Hence, we only perform one gradient step at test time for the experiments in this work.

4.3.6 Further Details on Procedure for Running Experiments

For both Krazy World and mazes, training proceeds in the following way. First, we initialize 32 training environments and 64 test environments. Every initialized environment has a different seed. Next, we initialize our chosen meta-RL algorithm by uniformly sampling hyper-parameters from predefined ranges. Data is then collected from all 32 training environments. The meta-RL algorithm then uses this data to make a meta-update, as detailed in the algorithm section of this section. The meta-RL algorithm is then allowed to do one training step on the 64 test environments to see how fast it can train at test time. These test environment results are recorded, 32 new tasks are sampled from the training environments, and data collection begins again. For MAML and E-MAML, training at test time means performing one VPG update at test time (see figure 11 for evidence that taking only one gradient step is sufficient). For RL^2 and E-RL^2, this means running three exploratory episodes to allow the RNN memory weights time to update and then reporting the loss on the fourth and fifth episodes. For both algorithms, meta-updates are calculated with PPO [109]. The entire process from the above paragraph is repeated from the beginning 64 times and the results are averaged to provide the final learning curves featured in this work.

4.4 Related Work

This work builds depends upon recent advances in deep reinforcement learning. [77, 76, 62] allow for discrete control in complex environments directly from raw images. [110, 76, 109, 70], have allowed for high-dimensional continuous control in complex environments from raw state information.

It has been suggested that our algorithm is related to the exploration vs. exploitation dilemma. There exists a large body of RL work addressing this problem [57, 18, 60]. In practice, these methods are often not used due to difficulties with high-dimensional observations, difficulty in implementation on arbitrary domains, and lack of promising results. This resulted in most deep RL work utilizing epsilon greedy exploration [77], or perhaps a simple scheme like Boltzmann exploration [23]. As a result of these shortcomings, a variety of new approaches to exploration in deep RL have recently been suggested [126, 52, 119, 85, 13, 86, 82, 41, 104, 82, 41, 108, 107, 122, 124, 61, 105]. In spite of these numerous efforts, the problem of exploration in RL remains difficult.

Many of the problems in meta RL can alternatively be addressed with the field of *hierarchical reinforcement learning*. In hierarchical RL, a major focus is on learning primitives that can be reused and strung together. Frequently, these primitives will relate to better coverage over state visitation frequencies. Recent work in this direction includes [134, 9, 128, 98, 11, 138]. The primary reason we consider meta-learning over hierarchical RL is that we find hierarchical RL tends to focus more on defining specific architectures that should lead to hierarchical behavior, whereas meta learning instead attempts to directly optimize for these behaviors.

As for meta RL itself, the literature is spread out and goes by many different names. There exist relevant literature on life-long learning, learning to learn, continual learning, and multi-task learning [102, 101]. We encourage the reviewer to look at the review articles [113, 127, 129] and their citations. The work most similar to ours has focused on adding curiosity or on a free learning phrase during training. However, these approaches are still quite different because they focus on defining an intrinsic motivation signals. We only consider better utilization of the existing reward signal. Our algorithm makes an explicit connection between free learning phases and the its affect on meta-updates.

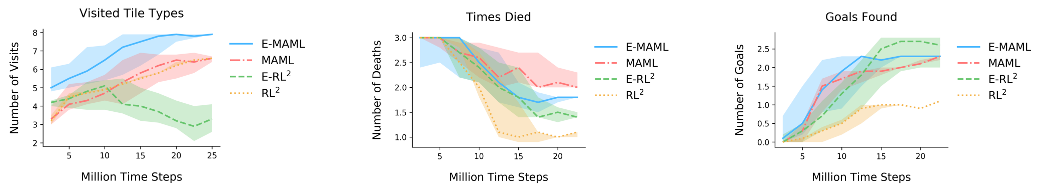


Figure 12: Three heuristic metrics designed to measure an algorithm's system identification ability on Krazy World: Fraction of tile types visited during test time, number of times killed at a death square during test time, and number of goal squares visited. We see that E-MAML is consistently the most diligent algorithm at checking every tile type during test time. Improving performance on these metrics indicates the meta learners are learning how to do at least some system identification.

4.5 Closing Remarks

In this work, we considered the importance of sampling in meta reinforcement learning. Two new algorithms were derived and their properties were analyzed. We showed that these algorithms tend to learn more quickly and cover more of their environment's states during learning than existing algorithms. It is likely that future work in this area will focus on meta-learning a curiosity signal which is robust and transfers across tasks, or learning an explicit exploration policy. Another interesting avenue for future work is learning intrinsic rewards that communicate long-horizon goals, thus better justifying exploratory behavior. Perhaps this will enable meta agents which truly *want* to explore rather than being forced to explore by mathematical trickery in their objectives.

5 Imitation Learning as a Sampling Problem

This chapter interprets imitation learning as the problem of learning a sampling distribution that matches an underlying expert distribution. It serves as a bridge, connecting the third-person imitation learning problem we explore in the next chapter to the broader themes of this thesis.

5.1 The Imitation Learning Objective

The central goal of imitation learning is for one agent to acquire a skill by watching expert demonstrations of that skill. For example, a robot might want to watch a video of another robot poring a glass of water and use these expert demonstrations to acquire the skill of pouring.

We can make this objective more rigorous in the following way. Suppose there is some expert policy π_E. This policy produces expert rollouts $\tau_E^i = (s_E^0, a_E^0, \ldots, s_E^i, a_E^i)$ which represent a successful completion of the given task. Meanwhile, the novice policy π_N similarly produces novice rollouts $\tau_N^i = (s_N^0, a_N^0, \ldots, s_N^i, a_N^i)$. We want to train π_N to produce trajectories that look like they came from π_E. Let $\hat{a}^i = \pi_N(s_N^{i-1})$. Then the imitation learning objective is that

$$d(\hat{a}^i, a_E^i) \tag{16}$$

should be small. Of course, if we assume d is the euclidean distance metric, we can optimize for this objective directly! It's a classical supervised learning problem that offers no complications.

In practice, direct optimization of the imitation learning objective via supervised learning often leads to over-fitting and poor generalization. An alternative route would be to replace the distance metric d with some sort of distributional distance, which will turn imitation learning into a more robust sampling problem than exact matching[3]. This is exactly what is done in Generative Adversarial Imitation Learning (GAIL).

[3]Technically, the supervised learning approach also turns imitation learning into a sampling problem, since we are hoping π_N can be used to sample expert trajectories. However, it's an exact matching problem that tries to match actions directly rather than explicitly trying to model the expert distribution and explicitly sample from it as in GAIL

5.2 Generative Adversarial Imitation Learning

One interesting scheme is to replace the distance metric d in (16). Rather than using euclidean distance, one might instead use a distributional distance that assesses how far away the distribution generated by sampling from π_N is from the distribution generated by sampling from π_E. One would then train π_N to minimize this distance, generating samples that look like they come from the distribution induced by π_E. This idea is quite similar to Generative Adversarial Networks (GANs) [40], as first noted by Ho in GAIL [50].

In [50], we assume that π_E induces a distribution over the state space Ω_E. Likewise, we assume π_N induces a distribution Ω_N. Let D be a neural network that returns the log probability that a state space belongs to Ω_E. In other words, D models the KL divergence[4] between Ω_E and Ω_N. GAIL proceeds by alternatively training D to better model this distance between the induced distributions and training π_N to further minimize the distance by enabling π_N to generate samples that look like they come from Ω_E. In GAIL, π_N is trained with policy gradient methods (See chapter 2) where the cost signal is equal to the distributional loss. Hence, we see that imitation learning is twice a sampling problem. First, you are trying to train a novice sampling distribution to match an expert sampling distribution. Second, the primary method of training this novice is via RL as a sampling problem as discussed in chapter 2.

5.3 The Mismatched Distribution Problem

It is worth noting that in practical imitation learning, a robot is almost never trying to imitate another robot. Instead, it tries to imitate expert trajectories provided by itself. These trajectories are usually acquired by using a human to manually move the robot, moving the robot with use of a video game controller, or using motion capture and a virtual reality headset to move the robot. Robots never actually imitate other robots. They certainly never imitate humans. This limitation of imitation learning is one instance of what I call the mismatched distribution problem: the problem of performing imitation learning in a setting where the distribution induced by the novice

[4]Technically the cross entropy loss is used but they are equivalent in this setting.

policy can never fully match the underlying expert distribution. Let us briefly review three instances of the problem.

1. **One-Shot Imitation Learning:** The novice is only provided a single expert demonstration to mimic. This means that the expert sampling distribution is a single point, correctly modeled by a Dirac delta function. This poses numerous issues. Neural networks are poor at modeling distributions with infinite derivatives. The generalization capabilities of GAIL in this context would be nonexistent. Sampling from this distribution is practically meaningless since all the mass is concentrated around a single point. In [32], we provide a practical algorithm for one-shot imitation learning that fixes many of these issues.

2. **Super-Expert Imitation Learning:** The expert that the agent is attempting to imitate may in fact be sub-optimal in some ways. For instance, the expert robot may shake too frequently, take inefficient paths to its goal, clumsily knock over objects, or sometimes fail to complete its goal. However, even though its demonstrations are sub-optimal, the expert agent is still demonstration the correct intent and providing good baseline performance. In Super-Expert imitation learning, we use the expert to bootstrap a policy with a performance that far exceeds that of the expert. Two primary algorithms are considered: meta-data imputation (MDI) and meta learning from sub-optimal demonstrations.

3. **Third-Person Imitation Learning:** A robot tries to imitate an agent that is different from itself. For instance, a robot might imitate a different model of robot or a human. In GAIL, any attempts at such imitation are doomed to failure because the robot and the expert have an inherent difference in their state spaces. Hence, the distributions induced by sampling from their policies will also be inherently different. Any reasonable attempt to naively measure the KL divergence between the expert and novice distributions will immediately find the obvious differences between the novice and the expert. In [118], we rectify this issue by introducing a third-person imitation learning algorithm that allows a novice to learn from expert demonstrations even when there is an inherent difference between the novice and expert's underlying state spaces. In the next chapter, we fully explore this algorithm.

6 Third-Person Imitation Learning

Reinforcement learning (RL) makes it possible to train agents capable of achieving sophisticated goals in complex and uncertain environments. A key difficulty in reinforcement learning is specifying a reward function for the agent to optimize. Traditionally, imitation learning in RL has been used to overcome this problem. Unfortunately, hitherto imitation learning methods tend to require that demonstrations are supplied in the *first-person*: the agent is provided with a sequence of states and a specification of the actions that it should have taken. While powerful, this kind of imitation learning is limited by the relatively hard problem of collecting first-person demonstrations. Humans address this problem by learning from *third-person* demonstrations: they observe *other humans* perform tasks, infer the task, and accomplish the same task themselves.

In this chapter, we present a method for *unsupervised* third-person imitation learning. Here third-person refers to training an agent to correctly achieve a simple goal in a simple environment when it is provided a demonstration of a teacher achieving the same goal but from a different viewpoint; and unsupervised refers to the fact that the agent receives only these third-person demonstrations, and is *not* provided a correspondence between teacher states and student states. Our methods primary insight is that recent advances from domain confusion can be utilized to yield domain agnostic features which are crucial during the training process. To validate our approach, we report successful experiments on learning from third-person demonstrations in a pointmass domain, a reacher domain, and inverted pendulum. This work was published as [118].

6.1 Introduction

Reinforcement learning (RL) is a framework for training agents to maximize rewards in large, unknown, stochastic environments. In recent years, combining techniques from deep learning with reinforcement learning has yielded a string of successful applications in game playing and robotics [77, 76, 110, 69]. These successful applications, and the speed at which the abilities of RL algorithms have been increasing, makes it an exciting area of research with significant potential for future applications.

One of the major weaknesses of RL is the need to manually specify a reward function. For each task we wish our agent to accomplish, we must provide it with a reward function whose maximizer will precisely recover the desired behavior. This weakness is addressed by the field of Inverse Reinforcement Learning (IRL). Given a set of expert trajectories, IRL algorithms produce a reward function under which these the expert trajectories enjoy the property of optimality. Recently, there has been a significant amount of work on IRL, and current algorithms can infer a reward function from a very modest number of demonstrations (e.g,. [1, 92, 142, 68, 50, 34]).

While IRL algorithms are appealing, they impose the somewhat unrealistic requirement that the demonstrations should be provided from the *first-person* point of view with respect to the agent. Human beings learn to imitate entirely from third-person demonstrations – i.e., by observing other humans achieve goals. Indeed, in many situations, first-person demonstrations are outright impossible to obtain. Meanwhile, third-person demonstrations are often relatively easy to obtain.

The goal of this chapter is to develop an algorithm for third-person imitation learning. Future advancements in this class of algorithms would significantly improve the state of robotics, because it will enable people to easily teach robots news skills and abilities. Importantly, we want our algorithm to be *unsupervised*: it should be able to observe another agent perform a task, infer that there is an underlying correspondence to itself, and find a way to accomplish the same task.

We offer an approach to this problem by borrowing ideas from domain confusion [133] and generative adversarial networks (GANs) [40]. The high-level idea is to introduce an optimizer under which we can recover both a domain-agnostic representation of the agent's observations, and a cost function which utilizes this domain-agnostic representation to capture the essence of expert trajectories. We formulate this as a third-person RL-GAN problem, and our solution builds on the first-person RL-GAN formulation by [50].

Surprisingly, we find that this simple approach has been able to solve the problems that are presented in this chapter (illustrated in Figure 13), even though the student's observations are related in a complicated way to the teacher's demonstrations (given that the observations and the demonstrations are pixel-level). As techniques for training GANs become more stable and capable, we expect our algorithm to be able to infer solve harder third-person imitation tasks without any direct supervision.

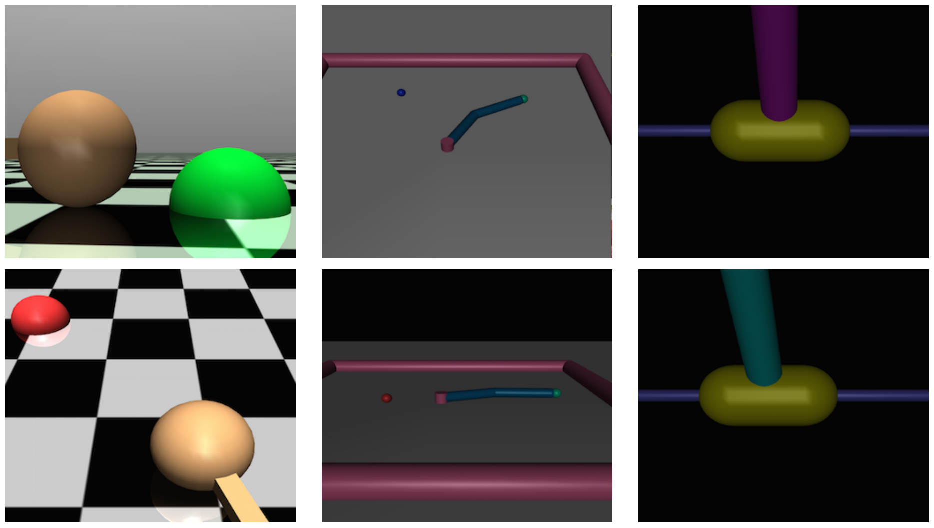


Figure 13: From left to right, the three domains we consider in this chapter: pointmass, reacher, and pendulum. Top-row is the third-person view of a teacher demonstration. Bottom row is the agent's view in their version of the environment. For the point and reacher environments, the camera angles differ by approximately 40 degrees. For the pendulum environment, the color of the pole differs.

6.2 Related Work

Imitation learning (also learning from demonstrations or programming by demonstration) considers the problem of acquiring skills from observing demonstrations. Imitation learning has a long history, with several good survey articles, including [100, 21, 5]. Two main lines of work within imitation learning are: 1) behavioral cloning, where the demonstrations are used to directly learn a mapping from observations to actions using supervised learning, potentially with interleaving learning and data collection (e.g., [89, 96]). 2) Inverse reinforcement learning [81], where a reward function is estimated that explains the demonstrations as (near) optimal behavior. This reward function could be represented as nearness to a trajectory [22, 2], as a weighted combination of features [1, 92, 91, 142, 16, 54, 28], or could also involve feature learning [93, 68, 140, 34, 50].

This past work, however, is not directly applicable to the *third* person imitation learning setting. In third-person imitation learning, the observations and actions obtained from the demonstrations are *not* the same as what the imitator agent will be faced with. A typical scenario would be: the imitator agent watches a human perform a demonstration, and then has to execute that same task. As discussed in [80] the "what and how to imitate" questions become significantly more challenging in this setting. To directly apply existing behavioral cloning or inverse reinforcement learning techniques would require knowledge of a mapping between observations and actions in the demonstrator space to observations and actions in the imitator space. Such a mapping is often difficult to obtain, and it typically relies on providing feature representations that captures the invariance between both environments [24, 112, 22, 79, 39, 45]. Contrary to prior work, we consider third-person imitation learning from raw sensory data, where no such features are made available.

The most closely related work to ours is by [34, 50, 140], who also consider inverse reinforcement learning directly from raw sensory data. However, the applicability of their approaches is limited to the first-person setting. Indeed, matching raw sensory observations is impossible in the 3rd person setting.

Our work also closely builds on advances in generative adversarial networks [40], which are very closely related to imitation learning as explained in [34, 50]. In our optimization formulation, we apply the gradient flipping technique from [36].

The problem of adapting what is learned in one domain to another domain has been studied extensively in computer vision in the supervised learning setting [141, 74, 64, 8, 31, 51, 71]. It has also been shown that features trained in one domain can often be relevant to other domains [29]. The work most closely related to ours is [133, 132], who also consider an explicit domain confusion loss, forcing trained classifiers to rely on features that don't allow to distinguish between two domains. This work in turn relates to earlier work by [20, 26], which also considers supervised training of deep feature embeddings.

Our approach to third-person imitation learning relies on reinforcement learning from raw sensory data in the imitator domain. Several recent advances in deep reinforcement learning have made this practical, including Deep Q-Networks [77], Trust Region Policy Optimization [110], A3C [76], and Generalized Advantage Estimation [111]. Our approach uses Trust Region Policy Optimization.

6.3 Background and Preliminaries

A discrete-time finite-horizon discounted Markov decision process (MDP) is represented by a tuple $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \rho_0, \gamma, T)$, in which $\mathcal{S}$ is a state set, $\mathcal{A}$ an action set, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_+$ a transition probability distribution, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ a reward function, $\rho_0 : \mathcal{S} \to \mathbb{R}_+$ an initial state distribution, $\gamma \in [0, 1]$ a discount factor, and T the horizon.

In the reinforcement learning setting, the goal is to find a policy $\pi_\theta : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_+$ parametrized by θ that maximizes the expected discounted sum of rewards incurred, $\eta(\pi_\theta) = \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{T} \gamma^t c(s_t)]$, where $s_0 \sim \rho_0(s_0)$, $a_t \sim \pi_\theta(a_t|s_t)$, and $s_{t+1} \sim \mathcal{P}(s_{t+1}|s_t, a_t)$.

In the (first-person) imitation learning setting, we are not given the reward function. Instead we are given traces (i.e., sequences of states traversed) by an expert who acts according to an unknown policy π_E. The goal is to find a policy π_θ that performs as well as the expert against the unknown reward function. It was shown in [1] that this can be achieved through inverse reinforcement learning by finding a policy π_θ that matches the expert's empirical expectation over discounted sum of all features that might contribute to the reward function. The work by [50] generalizes this to the setting when no features are provided as follows: Find a policy π_θ that makes it impossible for a discriminator (in their work a deep neural net) to distinguish states visited by the expert from states visited by the imitator agent. This can be formalized as follows:

$$\max_{\pi_\theta} \min_{\mathcal{D}_R} \quad - \mathbb{E}_{\pi_\theta}[\log \mathcal{D}_R(s)] - \mathbb{E}_{\pi_E}[\log(1 - \mathcal{D}_R(s))] \tag{17}$$

Here, the expectations are over the states experienced by the policy of the imitator agent, π_θ, and by the policy of the expert, π_E, respectively. $\mathcal{D}_R$ is the discriminator, which outputs the probability of a state having originated from a trace from the imitator policy π_θ. If the discriminator is perfectly able to distinguish which policy originated state-action pairs, then $\mathcal{D}_R$ will consistently output a probability of 1 in the first term, and a probability of 0 in the second term, making the objective its lowest possible value of zero. It is the role of the imitator agent π_θ to find a policy that makes it difficult for the discriminator to make that distinction. The desired equilibrium has the imitator agent making it impractical for the discriminator to distinguish, hence forcing the discriminator to assign probability 0.5 in all cases. [50] present a practical approach for solving this type of game when representing both π_θ and $\mathcal{D}_R$ as deep neural networks. Their approach repeatedly performs gradient updates on each of them. Concretely, for a current policy π_θ traces can be collected, which together with the expert traces form a data-set on which $\mathcal{D}_R$ can be trained with supervised learning minimizing the negative log-likelihood (in practice only performing a modest number of updates). For a fixed $\mathcal{D}_R$, this is a policy optimization problem where $-\log \mathcal{D}_R(s, a)$ is the reward, and policy gradients can be computed from those same traces. Their approach uses trust region policy optimization [110] to update the imitator policy π_θ from those gradients.

In our work we will have more terms in the objective, so for compactness of notation, we will realize the discriminative minimization from Eqn. (17) as follows:

$$\max_{\pi_\theta} \min_{\mathcal{D}_R} \mathcal{L}_R = \sum_i CE(\mathcal{D}_R(s_i), c_{\ell_i}) \tag{18}$$

Where s_i is state i, c_{ℓ_i} is the correct class label (was the state s_i obtained from an expert vs. from a non-expert), and CE is the standard cross entropy loss.

6.4 A Formal Definition Of The Third-Person Imitation Learning Problem

Formally, the third-person imitation learning problem can be stated as follows. Suppose we are given two Markov Decision Processes M_{π_E} and M_{π_θ}. Suppose further there exists a set of traces $\rho = \{(s_1, \dots, s_n)\}_{i=0}^n$ which were generated under a policy π_E acting optimally under some unknown reward

R_{π_E}. In third-person imitation learning, one attempts to recover by proxy through ρ a policy $\pi_\theta = f(\rho)$ which acts optimally with respect to R_{π_θ}.

6.5 A Third-Person Imitation Learning Algorithm

6.5.1 Game Formulation

In this section, we discuss a simple algorithm for third-person imitation learning. This algorithm is able to successfully discriminate between expert and novice policies, even when the policies are executed under different environments. Subsequently, this discrimination signal can be used to train expert policies in new domains via RL by training the novice policy to fool the discriminator, thus forcing it to match the expert policy.

In third-person learning, observations are more typically available rather than direct state access, so going forward we will work with observations o_t instead of states s_t as representing the expert traces. The top row of Figure 15 illustrates what these observations are like in our experiments.

We begin by recalling that in the algorithm proposed by [50] the loss in Equation 18 is utilized to train a discriminator $\mathcal{D}_R$ capable of distinguishing expert vs non-expert policies. Unfortunately, (18) will likely fail in cases when the expert and non-expert act in different environments, since $\mathcal{D}_R$ will quickly learn these differences and use them as a strong classification signal.

To handle the third-person setting, where expert and novice are in different environments, we consider that $\mathcal{D}_R$ works by first extracting features from o_t, and then using these features to make a classification. Suppose then that we partition $\mathcal{D}_R$ into a feature extractor $\mathcal{D}_F$ and the actual classifier which assigns probabilities to the outputs of D_F. Overloading notation, we will refer to the classifier as $\mathcal{D}_R$ going forward. For example, in case of a deep neural net representation, $\mathcal{D}_F$ would correspond to the earlier layers, and $\mathcal{D}_R$ to the later layers. The problem is then to ensure that D_F contains no information regarding the rollout's domain label d_ℓ (i.e., expert vs. novice domain). This can be realized as

$$\max_{\pi_\theta} \min \mathcal{L}_R = \sum_i CE(\mathcal{D}_R(\mathcal{D}_F(o_i)), c_{\ell_i})$$
$$\text{s.t. MI}(D_F(o_i); d_l) = 0$$

Where MI is mutual information and hence we have abused notation by using $\mathcal{D}_R$, D_F, and d_ℓ to mean the classifier, feature extractor, and the domain label respectively as well as distributions over these objects.

The mutual information term can be instantiated by introducing another classifier $\mathcal{D}_D$, which takes features produced by D_F and outputs the probability that those features were produced by in the expert vs. non-expert environment. (See [19, 10, 63, 25] for further discussion on instantiating the information term by introducing another classifier.) If $\sigma_i = D_F(o_i)$, then the problem can be written as

$$\max_{\pi_\theta} \min_{\mathcal{D}_R} \max_{\mathcal{D}_D} \mathcal{L}_R + \mathcal{L}_D = \sum_i CE(\mathcal{D}_R(\sigma_i), c_{\ell_i}) + CE(\mathcal{D}_D(\sigma_i), d_{\ell_i}) \quad (19)$$

In words, we wish to minimize class loss while maximizing domain confusion.

Often, it can be difficult for even humans to judge a static image as expert vs. non-expert because it does not convey any information about the environmental change affected by the agent's actions. For example, if a pointmass is attempting to move to a target location and starts far away from its goal state, it can be difficult to judge if the policy itself is bad or the initialization was simply unlucky. In response to this difficulty, we give $\mathcal{D}_R$ access to not only the image at time t, but also at some future time $t + n$. Define $\sigma_t = D_F(o_t)$ and $\sigma_{t+n} = D_F(o_{t+n})$. The classifier then makes a prediction $\mathcal{D}_R(\sigma_t, \sigma_{t+n}) = \hat{c}_\ell$.

This renders the following formulation:

$$\max_{\pi_\theta} \min_{\mathcal{D}_R} \max_{\mathcal{D}_D} \mathcal{L}_R + \mathcal{L}_D = \sum_i CE(\mathcal{D}_R(\sigma_i, \sigma_{i+n}), c_{\ell_i}) + CE(\mathcal{D}_D(\sigma_i), d_{\ell_i}) \quad (20)$$

Note we also want to optimize over $\mathcal{D}_F$, the feature extractor, but it feeds both into $\mathcal{D}_R$ and into $\mathcal{D}_D$, which are competing (hidden under σ), which we will address now.

To deal with the competition over $\mathcal{D}_F$, we introduce a function $\mathcal{G}$ that acts as the identity when moving forward through a directed acyclic graph and flips the sign when backpropagating through the graph. This technique has enjoyed recent success in computer vision. See, for example, [36]. With this trick, the problem reduces to its final form

$$\max_{\pi_\theta} \min_{\mathcal{D}_R, \mathcal{D}_D, \mathcal{D}_F} \mathcal{L}_R + \mathcal{L}_D = \sum_i CE(\mathcal{D}_R(\sigma_i, \sigma_{i+n}), c_{\ell_i}) + \lambda \; CE(\mathcal{D}_D(\mathcal{G}(\sigma_i), d_{\ell_i}) \tag{21}$$

In Equation (21), we flip the gradient's sign during backpropagation of D_F with respect to the domain classification loss. This corresponds to stochastic gradient ascent away from features that are useful for domain classification, thus ensuring that D_F produces domain agnostic features. Equation 21 can be solved efficiently with stochastic gradient descent. Here λ is a hyperparameter that determines the trade-off made between the objectives that are competing over $\mathcal{D}_F$.

To ensure sufficient signal for discrimination between expert and non-expert, we collect third-person demonstrations in the expert domain from both an expert and from a non-expert.

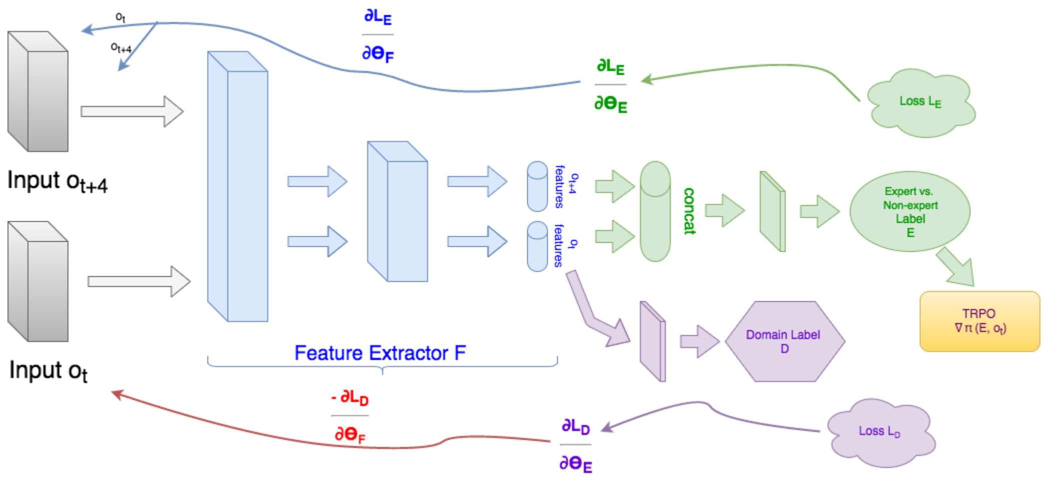
Our complete formulation is graphically summarized in Figure 14.

6.5.2 Algorithm

To solve the game formulation in Equation (21), we perform alternating (partial) optimization over the policy π_θ and the reward function and domain confusion encoded through $\mathcal{D}_R, \mathcal{D}_D, \mathcal{D}_F$.

The optimization over $\mathcal{D}_R, \mathcal{D}_D, \mathcal{D}_F$ is done through stochastic gradient descent with ADAM [58].

Our generator (π_θ) step is similar to the generator step in the algorithm by [50]. We simply use $-\log \mathcal{D}_R$ as the reward. Using policy gradient meth-



o_t
o_{t+4}
$\frac{\partial L_E}{\partial \theta_F}$
$\frac{\partial L_E}{\partial \theta_E}$
Loss L_E
Input o_{t+4}
Input o_t
o_{t+4} features
o_t features
concat
Expert vs. Non-expert Label E
TRPO $\nabla\pi$ (E, o_t)
Domain Label D
Feature Extractor F
$\frac{-\partial L_D}{\partial \theta_F}$
$\frac{\partial L_D}{\partial \theta_E}$
Loss L_D

Figure 14: Architecture diagram for third-person imitation learning. Images at time t and $t + 4$ are sent through a feature extractor to obtain $F(o_t)$ and $F(o_{t+4})$. Subsequently, these feature vectors are reused in two places. First, they are concatenated and used to predict whether the samples are drawn from expert or non-expert trajectories. Second, $F(o_t)$ is utilized to predict a domain label (expert vs. novice domain). During backpropogation, the sign on the domain loss L_D is flipped to destroy information that was useful for distinguishing the two domains. This ensures that the feature extractor F is domain agnostic. Finally, the classes probabilities that were computed using this domain-agnostic feature vector are utilized as a cost signal in TRPO; which is subsequently utilized to train the novice policy to take expert-like actions and collect further rollouts.

ods (TRPO), we train the generator to minimize this cost and thus push the policy further towards replicating expert behavior. Once the generator step is done, we start again with the discriminator step. The entire process is summarized in algorithm 1.

Algorithm 2 A third-person imitation learning algorithm.

```
 1: Let CE be the standard cross entropy loss.
 2: Let $\mathcal{G}$ be a function that flips the gradient sign during backpropogation and acts as
    the identity map otherwise.
 3: Initialize two domains, $E$ and $N$ for the expert and novice.
 4: Initialize a memory bank $\Omega$ of expert success and of failure in domain $E$. Each tra-
    jectory $\omega \in \Omega$ comprises a rollout of images $o = o_1, \ldots, o_t, \ldots o_n$, a class label $c_\ell$, and
    a domain label $d_\ell$.
 5: Initialize $\mathcal{D} = \mathcal{D}_F, \mathcal{D}_R, \mathcal{D}_D$, a domain invariant discriminator.
 6: Initialize a novice policy $\pi_\theta$.
 7: Initialize numiters, the number of inner policy optimization iterations we wish to run.
 8: for iter in numiters do
 9:     Sample a set of successes and failures $\omega_E$ from $\Omega$.
10:     Collect on policy samples $\omega_N$
11:     Set $\omega = \omega_E \cup \omega_N$.
12:     Shuffle $\omega$
13:     for $o, c_\ell, d_\ell$ in $\omega$ do
14:         for $o_t$ in $o$ do
15:             $\sigma_t = \mathcal{D}_F(o_t)$
16:             $\sigma_{t+4} = \mathcal{D}_F(o_{t+4})$
17:             $\mathcal{L}_R = CE(\mathcal{D}_R(\sigma_t, \sigma_{t+4}), c_\ell)$
18:             $\mathcal{L}_d = CE(\mathcal{D}_D(\mathcal{G}(\sigma_t)), d_\ell)$
19:             $\mathcal{L} = \lambda \cdot \mathcal{L}_d + \mathcal{L}_R$ [5]
20:             minimize $\mathcal{L}$ with ADAM.
21:         end for
22:     end for
23:     Collect on policy samples $\omega_N$ from $\pi_\theta$.
24:     for $\omega$ in $\omega_N$ do
25:         for $\omega_t$ in $\omega$ do
26:             $\sigma_t = \mathcal{D}_F(o_t)$
27:             $\sigma_{t+4} = \mathcal{D}_F(o_{t+4})$
28:             $\hat{c}_\ell = \mathcal{D}_R(\sigma_t, \sigma_{t+4})$
29:             $r = \hat{c}_\ell[0]$, the probability that $o_t, o_{t+4}$ were generated via expert rollouts.
30:             Use $r$ to train $\pi_\theta$ with via policy gradients (TRPO).
31:         end for
32:     end for
33: end for
34: return optimized policy $\pi_\theta$
```

6.6 Experiments

We seek to answer the following questions through experiments:

1. Is it possible to solve the third-person imitation learning problem in simple settings? I.e., given a collection of expert image-based rollouts in one domain, is it possible to train a policy in a different domain that replicates the essence of the original behavior?
2. Does the algorithm we propose benefit from both domain confusion and velocity?
3. How sensitive is our proposed algorithm to the selection of hyper-parameters used in deployment?
4. How sensitive is our proposed algorithm to changes in camera angle?
5. How does our method compare against some reasonable baselines?

6.6.1 Environments

To evaluate our algorithm, we consider three environments in the MuJoCo physics simulator. There are two different versions of each environment, an expert variant and a novice variant. Our goal is to train a cost function that is domain agnostic, and hence can be trained with images on the expert domain but nevertheless produce a reasonable cost on the novice domain. See Figure 1 for a visualization of the differences between expert and novice environments for the three tasks.

Point: A pointmass attempts to reach a point in a plane. The color of the target and the camera angle change between domains.

Reacher: A two DOF arm attempts to reach a designated point in the plane. The camera angle, the length of the arms, and the color of the target point are changed between domains. Note that changing the camera angle significantly alters the image background color from largely gray to roughly 30 percent black. This presents a significant challenge for our method.

Inverted Pendulum: A classic RL task wherein a pendulum must be made to balance via control. For this domain, We only change the color of the pendulum and not the camera angle. Since there is no target point, we found that changing the camera angle left the domain invariant representations with too little information and resulted in a failure case. In contrast to some traditional renderings of this problem, we do not terminate an episode when the agent falls but rather allow data collection to continue for a fixed horizon.

6.6.2 Evaluations

Is it possible to solve the third-person imitation learning problem in simple settings? In Figure 15, we see that our proposed algorithm is indeed able to recover reasonable policies for all three tasks we examined. Initially, the training is quite unstable due to the domain confusion wreaking havoc on the learned cost. However, after several iterations the policies eventually head towards reasonable local minima and the standard deviation over the reward distribution shrinks substantially. Finally, we note that the extracted feature representations used to complete this task are in fact domain-agnostic, as seen in Figure 16. Hence, the learning is properly taking place from a third-person perspective.

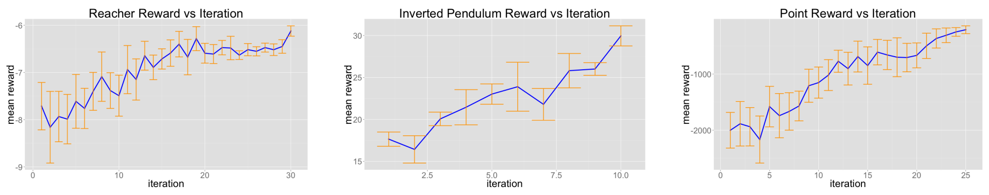


Figure 15: Reward vs training iteration for reacher, inverted pendulum, and point environments. The learning curves are averaged over 5 trials with error bars represent one standard deviation in the reward distribution at the given point.

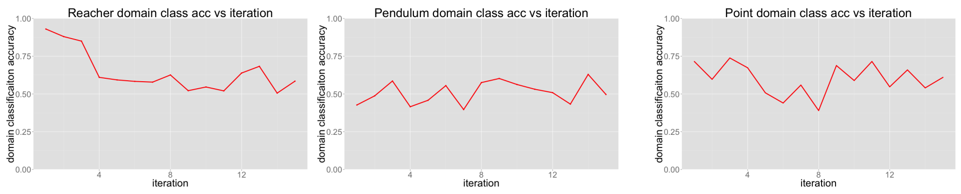


Figure 16: Domain accuracy vs. training iteration for reacher, inverted pendulum, and point environments.

Does the algorithm we propose benefit from both domain confusion and the multi-time step input? We answer this question with the experiments summarized in Figure 17. This experiment compares our approach with: (i) our approach without the domain confusion loss; (ii) our approach without the multi-time step input; (iii) our approach without the domain confusion loss and without the multi-time step input (which is very similar to the approach in [50]). We see that adding domain confusion is essential for getting strong performance in all three experiments. Meanwhile, adding multi-time step input marginally improves the results. See also Figure 19 for an analysis of the effects of multi-time step input on the final results.

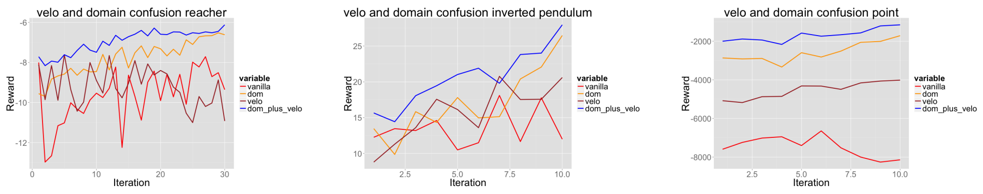


Figure 17: Reward vs iteration for reacher, inverted pendulum, and point environments with no domain confusion and no velocity (red), domain confusion (orange), velocity (brown), and both domain confusion and velocity (blue).

How sensitive is our proposed algorithm to the selection of hyper-parameters used in deployment? Figure 18 shows the effect of the domain confusion coefficient λ, which trades off how much we should weight the domain confusion objective vs. the standard cost-recovery objective, on the final performance of the algorithm. Setting λ too low results in slower learning and features that are not domain-invariant. Setting λ too high results in an objective that is too quick to destroy information, which makes it impossible to recover an accurate cost.

For multi-time step input, one must choose the number of look-ahead frames that are utilized. If too small a window is chosen, the agent's actions have not affected a large amount of change in the environment and it is difficult to discern any additional class signal over static images. If too large a time-frame passes, causality becomes difficult to interpolate and the agent does worse than simply being trained on static frames. Figure 19 illustrates that no number of look-ahead frames is consistently optimal across tasks. However, a value of 4 showed good performance over all tasks, and so this value was utilized in all other experiments.

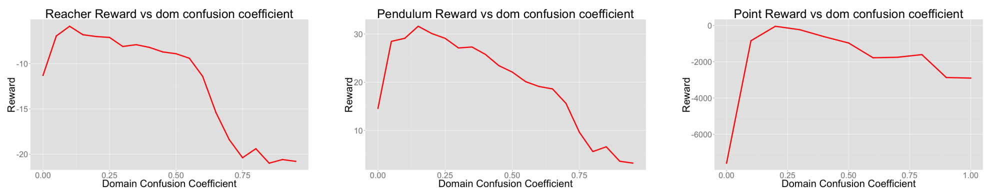


Figure 18: Reward of final trained policy vs domain confusion weight λ for reacher, inverted pendulum, and point environments.

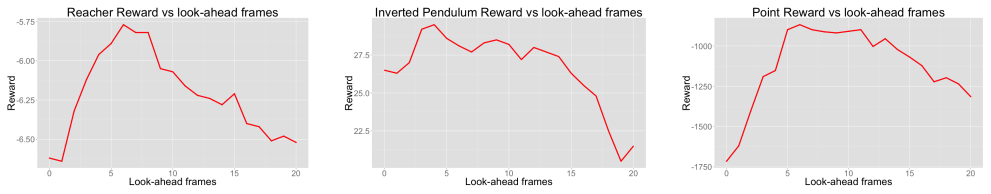
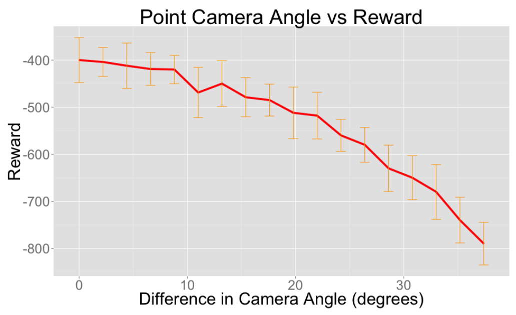


Figure 19: Reward of final trained policy vs number of look-ahead frames for reacher, inverted pendulum, and point environments.

How sensitive is our algorithm to changes in camera angle? We present graphs for the reacher and point experiments wherein we exam the final reward obtained by a policy trained with third-person imitation learning vs the camera angle difference between the first-person and third-person perspective. We omit the inverted double pendulum experiment, as the color and not the camera angle changes in that setting and we found the case of slowly transitioning the color to be the definition of uninteresting science.



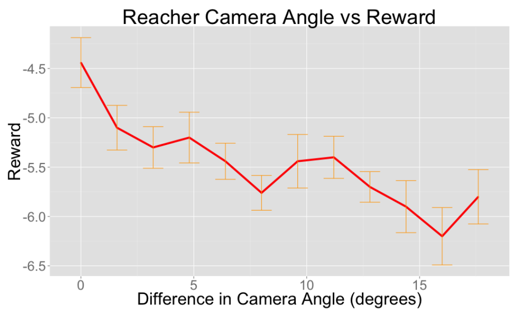
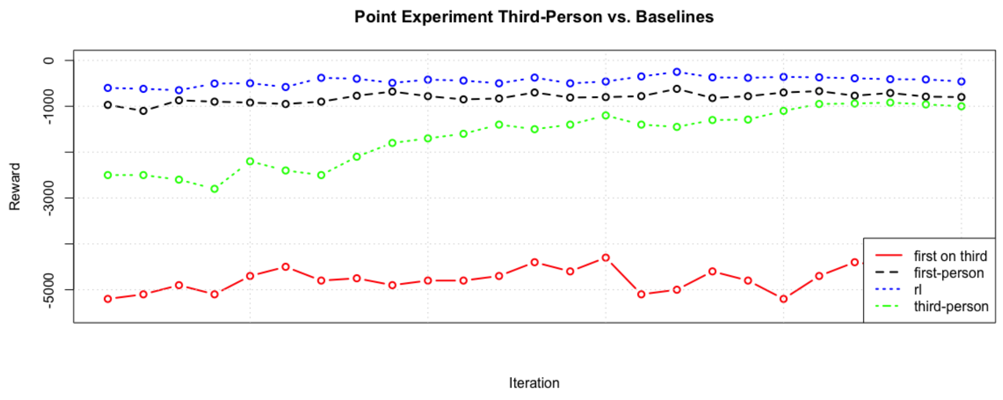


Figure 20: Point and reacher final reward after 20 epochs of third-person imitation learning vs the camera angle difference between the first and third-person perspective. We see that the point follows a fairly linear slope in regards to camera angle differences, whereas the reacher environment is more stochastic against these changes.



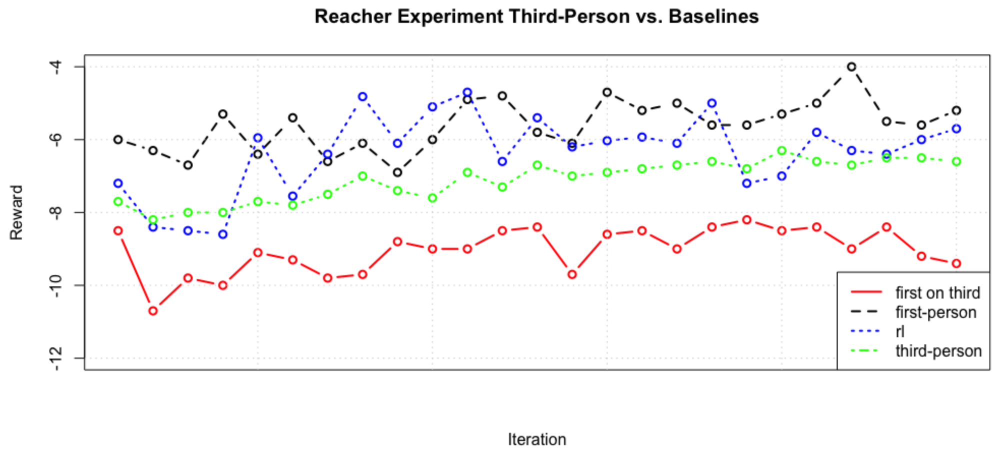


Figure 21: Learning curves for third-person imitation vs. three baselines: 1)RL with true reward, 2) first-person imitation, 3) attempting to use first-person features on the third-person agent.

How does our method compare against reasonable baselines? We consider the following baselines for comparisons against third-person imitation learning. 1) Standard reinforcement learning with using full state information and the true reward signal. This agent is trained via TRPO. 2) Standard GAIL (first-person imitation learning). Here, the agent receives first-person demonstration and attempts to imitate the correct behavior. This is an upper bound on how well we can expect to do, since we have the correct perspective. 3) Training a policy using first-person data and applying it to the third-person environment.

We compare all three of these baselines to third-person imitation learning. As we see in figure 9: 1) Standard RL, which (unlike the imitation learning approaches) has access to full state and true reward, helps calibrate performance of the other approaches. 2) First-person imitation learning is faced with a simpler imitation problem and accordingly outperforms third-person imitation, yet third-person imitation learning is nevertheless competitive. 3) Applying the first-person policy to the third-person agent fails miserably, illustrating that explicitly considering third-person imitation is important in these settings.

Somewhat unfortunately, the different reward function scales make it difficult to capture information on the variance of each learning curve. Consequently, below we have included the full learning curves for these experiments with variance bars, each plotted with an appropriate scale to examine the variance of the individual curves.

6.6.3 Learning Curves for Baselines

Here, we plot the learning curves for each of the baselines mentioned in the experiments section as a standalone plot. This allows one to better examine the variance of each individual learning curve.

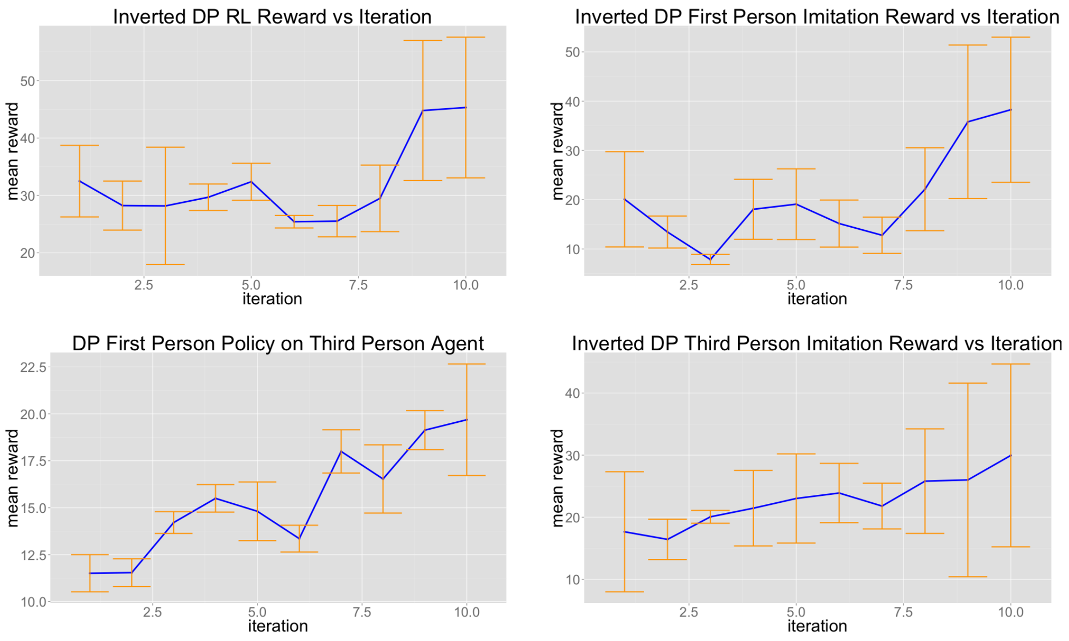


Figure 22: Inverted Pendulum performance under a policy trained on RL, first-person imitation learning, third-person imitation, and a first-person policy applied to a third-person agent.

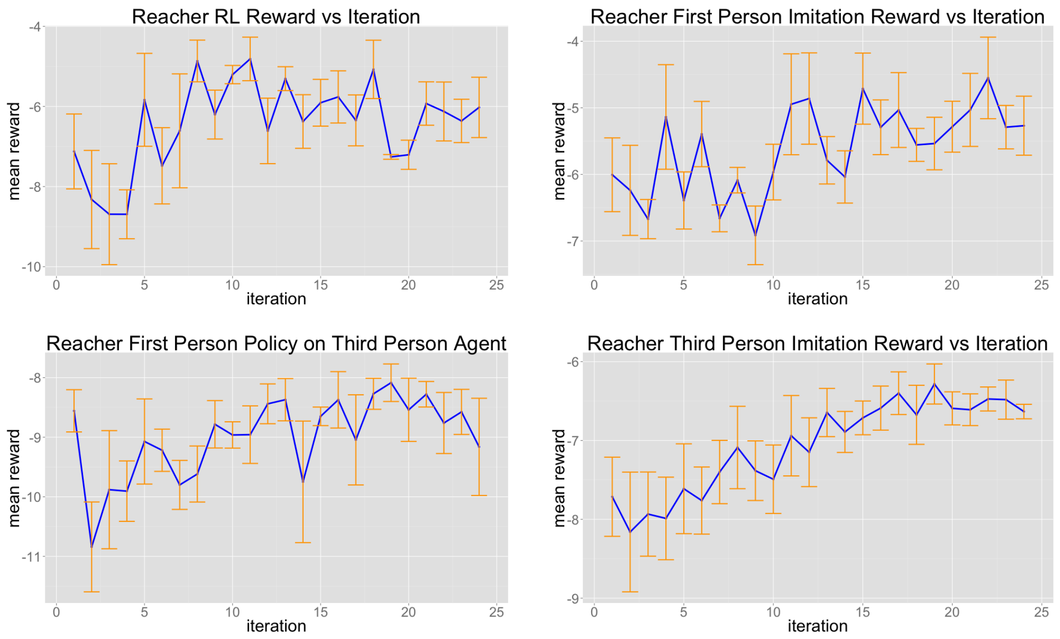


Figure 23: Reacher performance under a policy trained on RL, first-person imitation learning, third-person imitation, and a first-person policy applied to a third-person agent.

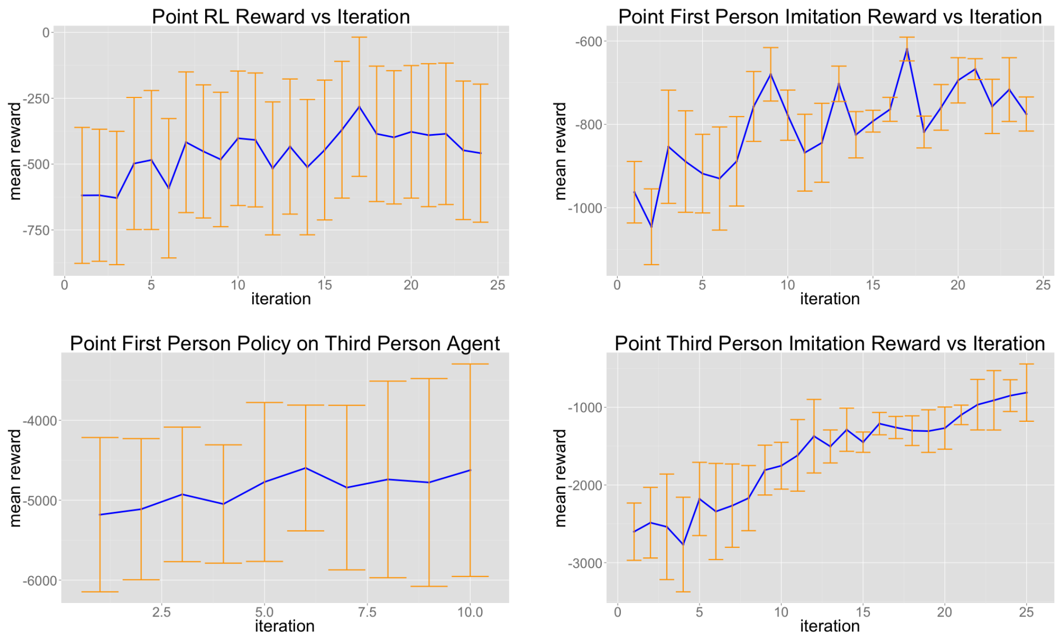


Figure 24: Point performance under a policy trained on RL, first-person imitation learning, third-person imitation, and a first-person policy applied to a third-person agent.

6.6.4 Architecture Parameters

Joint Feature Extractor: Input is images are size 50 x 50 with 3 channels, RGB. Layers are 2 convolutional layers each followed by a max pooling layer of size 2. Layers use 5 filters of size 3 each.

Domain Discriminator and the Class Discriminator: Input is domain agnostic output of convolutional layers. Layers are two feed forward layers of size 128 followed by a final feed forward layer of size 2 and a soft-max layer to get the log probabilities.

ADAM is used for discriminator training with a learning rate of 0.001. The RL generator uses the off-the-shelf TRPO implementation available in RL-Lab.

6.7 Discussion and Future Work

In this chapter, we presented the problem of third-person imitation learning. We argue that this problem will be important going forward, as techniques in reinforcement learning and generative adversarial learning improve and the cost of collecting first-person samples remains high. We presented an algorithm which builds on Generative Adversarial Imitation Learning and is capable of solving simple third-person imitation tasks.

One promising direction of future work in this area is to jointly train policy features and cost features at the pixel level, allowing the reuse of image features. Code to train a third-person imitation learning agent on the domains from this chapter is presented here: `https://github.com/bstadie/third_person_im`

7 Transfer Learning for Causal Inference

The previous work on this paper has focused on solving various learning problems in the fields of meta-learning, imitation learning, and reinforcement learning. In each case, we showed that the learning problem we were interested in solving arose naturally from the perspective that learning is really a sampling problem.

In this chapter, we take a step back. Instead of focusing on one particular learning problem, we instead consider one of the fundamental difficulties that arises from interpreting learning as a sampling problem: it is extremely difficult to understand causal relationships in the agent's environment through sampling alone. It is difficult to understand not only the causal relationships between states, but also the causal relationship between actions and state transitions. Propagating a policy's understanding of causal effects across hundreds of time-steps and across wide search trees of states is a nearly impossible problem.

Model based reinforcement learning exists in large part to rectify this problem. Model based algorithms allow the agent access to a model of the system's dynamics. This allows for better planning, a more consistent interpretation of relationships between states, and a more explicit use of the causal relationship between actions and state transitions. However, dynamics models are still not enough. They are too low-level and local, seeking to explicitly model the impacts of actions on immediately following state transitions. This ignores the need for a higher-level understanding of complex relationships between present and future states and the rules governing the agent's environment and the problem it is trying to solve. In the video game pong, a dynamics model will help the agent see that hitting the ball changes the direction of the ball. However, this dynamics model will not help the agent understand that its true goal is to hit the ball past its opponent's paddle. This deficiency is at the heart of the issue that arises from naively treating learning as a sampling problem over a state space. *Learning ought to be a sampling problem over the various causal relationships within an agent's environment, not a sampling problem over a low-level state space.* In this chapter, we will take a step towards this paradigm by considering the problem of transfer learning for causal inference. The algorithm we present allows causal relationships to be transported from one causal study to another. Such an algorithm could allow agent's to port their understanding of one causal relationship in their environment to another analogous causal relationship.

Below, we develop new algorithms for estimating heterogeneous treatment effects, combining recent developments in transfer learning for neural networks with insights from the causal inference literature. By taking advantage of transfer learning, we are able to efficiently use different data sources that are related to the same underlying causal mechanisms. We compare our algorithms with those in the extant literature using extensive simulation studies based on large-scale voter persuasion experiments and the MNIST database. Our methods can perform an order of magnitude better than existing benchmarks while using a fraction of the data. This work was published as [120].

7.1 Introduction

The rise of massive datasets that provide fine-grained information about human beings and their behavior provides unprecedented opportunities for evaluating the effectiveness of treatments. Researchers want to exploit these large and heterogeneous datasets, and they often seek to estimate how well a given treatment works for individuals conditioning on their observed covariates. This problem is important in medicine (where it is sometimes called personalized medicine) [47, 90], digital experiments [125], economics [7], political science [42], statistics [131], and many other fields. A large number of articles are being written on this topic, but many outstanding questions remain. In this thesis, we will consider the first application of transfer learning to this problem.

In the simplest case, treatment effects are estimated by splitting a training set into a treatment and a control group. The treatment group receives the treatment, while the control group does not. The outcomes in those groups are then used to construct an estimator for the Conditional Average Treatment Effect (CATE), which is defined as the expected outcome under treatment minus the expected outcome under control given a particular feature vector [6]. This is a challenging task because, for every unit, we either observe its outcome under treatment or control, but never both. Assumptions, such as the random assignment of treatment and additional regularity conditions, are needed to make progress. Even with these assumptions, the resulting estimates are often noisy and unstable because the CATE is a vector parameter. Recent research has shown that it is important to use estimators which consider both treatment groups simultaneously ([65, 136, 84, 48]). Unfortunately, these recent advances are often still insufficient to train robust CATE estimators because of the large sample sizes required when the number of covariates is not small.

However, researchers usually fail to use ancillary datasets that are available to them in applications. This is surprising, given the need for additional data to estimate CATE reliably. These ancillary datasets are related to the causal mechanism under investigation, but they are also partially distinct so they cannot be pooled naively, which explains why researches often do not use them. Examples of such ancillary datasets include observations from: experiments in different locations on different populations, different treatment arms, different outcomes, and non-experimental observational studies. The key idea underlying our contributions is that one can substantially improve CATE estimators by transferring information from other data sources.

Our contributions are as follows:

1. **We introduce the new problem of transfer learning for estimating heterogeneous treatment effects.**

2. **We develop the Y-learner for CATE estimation.** We consider the problem of CATE estimation with deep neural networks. We propose the Y-Learner, a CATE estimator designed from the ground up to take advantage of deep neural networks' ability to easily share information across layers. The Y-Learner often achieves state-of-the-art performance on CATE estimation. The Y-learner does not use transfer learning.

3. **MLRW Transfer for CATE Estimation** adapts the idea of meta-learning regression weights (MLRW) to CATE estimation. Using these learned weights, regression problems can be optimized much more quickly than with random initializations. Though a variety of MLRW algorithms exist, it is not immediately obvious how one should use these methods for CATE estimation. The principle difficulty is that CATE estimation requires the simultaneous estimation of outcomes under both treatment and control, when we only observe one of the outcomes

for any individual unit. However, most MLRW transfer methods optimize on a per-task basis to estimate a single quantity. We show that one can overcome this problem with clever use of the Reptile algorithm [83].While adapting Reptile to work with our problem, we discovered a slight modification to the original algorithm. To distinguish this modification, we refer to it in this chapter as **SF Reptile**, Slow-Fast Reptile.

4. **We provide several additional methods for transfer learning for CATE estimation:** warm start, frozen-features, multi-head, and joint training.
5. **We apply our methods to difficult data problems and show that they perform better than existing benchmarks.** We reanalyze a set of large field experiments that evaluate the effect of a mailer on voter turnout in the 2014 U.S. midterm elections [38]. This includes 17 experiments with 1.96 million individuals in total. We also simulate several randomized controlled trials using image data of handwritten digits found in the MNIST database [67]. We show that our methods, **MLRW** in particular, obtain better than state-of-the-art performance in estimating CATE, and that they require far fewer observations than extant methods.
6. **We provide open source code for our algorithms.**[6]

7.2 CATE Estimation

7.2.1 Background and Assumptions

We begin by formally introducing the CATE estimation problem. Following the potential outcomes framework [97], assume there exists a single experiment wherein we observe N i.i.d. distributed units from some super population, $(Y_i(0), Y_i(1), X_i, W_i) \sim \mathcal{P}$. $Y_i(0) \in \mathbb{R}$ denotes the potential outcome of unit i if it is in the control group, $Y_i(1) \in \mathbb{R}$ is the potential outcome of i if it is in the treatment group, $X_i \in \mathbb{R}^d$ is a d-dimensional feature vector,

[6]The software will be released once anonymity is no longer needed. We can also provide an anynomized copy to reviewers upon request.

and $W_i \in \{0,1\}$ is the treatment assignment. For each unit in the treatment group ($W_i = 1$), we only observe the outcome under treatment, $Y_i(1)$. For each unit under control ($W_i = 0$), we only observe the outcome under control. Crucially, there cannot exist overlap between the set of units for which $W_i = 1$ and the set for which $W_i = 0$. It is impossible to observe both potential outcomes for any unit. This is commonly referred to as the fundamental problem of causal inference.

However, not all hope is lost. We can still estimate the Conditional Average Treatment Effect (CATE) of the treatment. Let x be an individual feature vector. Then the CATE of x, denoted $\tau(x)$, is defined by

$$\tau(x) = \mathbb{E}[Y(1) - Y(0)|X = x].$$

Estimating τ is impossible without making further assumptions on the distribution of $(Y_i(0), Y_i(1), X_i, W_i)$. In particular, we need to place two assumptions on our data.

Assumption 1 (Strong Ignorability, [95])**.**

$$(Y_i(1), Y_i(0)) \perp W|X.$$

Assumption 2 (Overlap)**.** *Define the propensity score of x as,*

$$e(x) := \mathbb{P}(W = 1|X = x).$$

Then there exists constant $0 < e_{min}$, $e_{max} < 1$ such that for all $x \in$ Support(X),

$$0 < e_{min} < e(x) < e_{max} < 1.$$

In words, $e(x)$ is bounded away from 0 and 1.

Assumption 1 ensures that there is no unobserved confounder, a random variable which influences both the probability of treatment and the potential outcomes, which would make the CATE unidentifiable. The assumption is particularly strong and difficult to check in applications. Meanwhile, Assumption 2 rectifies the situation wherein a certain part of the population is always treated or always in the control group. If, for example, all women were in the control group, one cannot identify the treatment effect for women. Though both assumptions are strong, they are nevertheless satisfied by design in randomized controlled trials. While the estimators we discuss would be sensible in observational studies when the assumptions are satisfied, we warn practitioners to be cautious in such studies, especially when the number of covariates is large [27].

7.2.2 Basic CATE Estimation with the T-Learner

Given our two assumptions, there exist many valid CATE estimators. The crux of these methods is to estimate two quantities: the control response function,

$$\mu_0(x) = \mathbb{E}[Y(0)|X = x],$$

and the treatment response function,

$$\mu_1(x) = \mathbb{E}[Y(1)|X = x].$$

If we denote our learned estimates as $\hat{\mu}_0(x)$ and $\hat{\mu}_1(x)$, then we can form the CATE estimate as the difference between the two

$$\hat{\tau}(x) = \hat{\mu}_1(x) - \hat{\mu}_0(x).$$

The astute reader may be wondering why we don't simply estimate μ_0 and μ_1 with our favorite function approximation algorithm at this point and then all go home. After all, we have access to the ground truths μ_0 and μ_1 and the corresponding inputs x. In fact, it is commonplace to do exactly that. When people directly estimate μ_0 and μ_1 with their favorite model, we call the procedure a T-learner [65]. Common choices of models include linear models and random forests, though neural networks have recently been considered [84].

While it may seem like we've triumphed, the T-learner does have some drawbacks [6]. It is usually an inefficient estimator. For example, it will often perform poorly when one can borrow information across the treatment conditions. To overcome these deficiencies, a variety of alternative learners have been suggested. Closely related to the T-learner is the idea of estimating the outcome using all of the features and the treatment indicator, without giving the treatment indicator a special role [48]. The predicted CATE for an individual unit is then the difference between the predicted values when the treatment assignment indicator is changed from control to treatment, with all other features held fixed. This is called the S-learner, because it uses a single prediction model.

In the following section, we suggest another new learner called the **Y-learner** (See Figure 25). This learner has been engineered from the ground up to take advantage of some of the unique capabilities of neural networks. See below for a full description of the Y-learner, and additional learners found in the literature. Below, we will use these learners as base algorithms for transfer learning. That is to say, we will use the knowledge gained by training one learner on one experiment to help a new learner with a new underlying experiment train faster with less data.

7.2.3 More Advanced CATE Estimation with the Y-Learner

In this section, we show the favorable behavior of the Y-learner over the X-learner. In order to show this, we implemented the X-learner exactly as it is described in [65] and the Y-learner as it is described in this section. Figure 26 shows the MSE in proportion to its sample size. We can see that the X–learner is consistently outperformed on all these data sets by the Y-learner. We note that all these data sets were intentionally crated to be very similar to the GOTV data set we are interested in studying. Therefore these data sets are not extremely different from each other, and it is possible that the X-NN performs much better on different data sets.

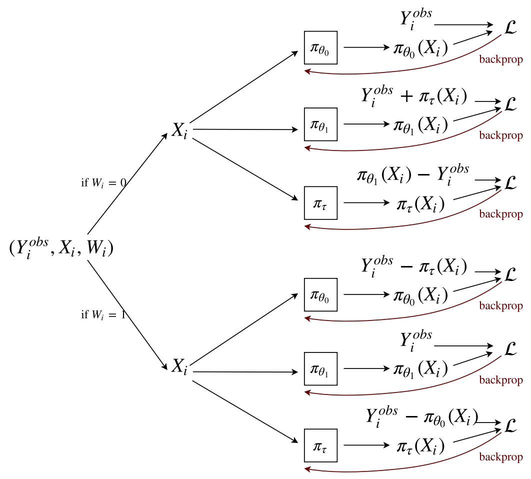


Figure 25: Y-learner with Neural Networks. One of many advanced methods for CATE estimation. See Section 7.2.3 and 7.6.2 for a more detailed overview.

Another important advantage of neural networks is that they can be trained jointly. This enables us to adapt well-performing meta-learners to perform even better. Specifically, we used the idea of X-NN to propose a new CATE estimator, which we call Y-NN.[7] The X-learner is essentially a two step procedure. In the first stage, the outcome functions, $\hat{\mu}_0$ and $\hat{\mu}_1$, are estimated and the individual treatment effects are imputed:

$$D_i^1 := Y(1) - \hat{\mu}_0(X_i) \qquad \text{and} \qquad D_i^0 := \hat{\mu}_1(X_i) - Y_i(0).$$

In the second stage, estimators for the CATE are derived by regressing the features X on the imputed treatment effects. [65] provides details. In the X-learner, the estimators of the first stage are held fixed and are not updated in the second stage. This is necessary since, unlike neural networks, many machine learning algorithms, such as RF and BART, cannot be updated in a meaningful way once they have been trained. For neural networks and similar gradient optimization-based algorithms, it is possible to jointly update the estimators in the first and the second stage.

This is exactly the motivation of the Y-learner. Instead of first deriving

[7]Y is chosen as it is the next letter in the alphabet after X. However, this is not a meta-learner because there is no obvious way to extend it to arbitrary base learners, such as RF or BART.

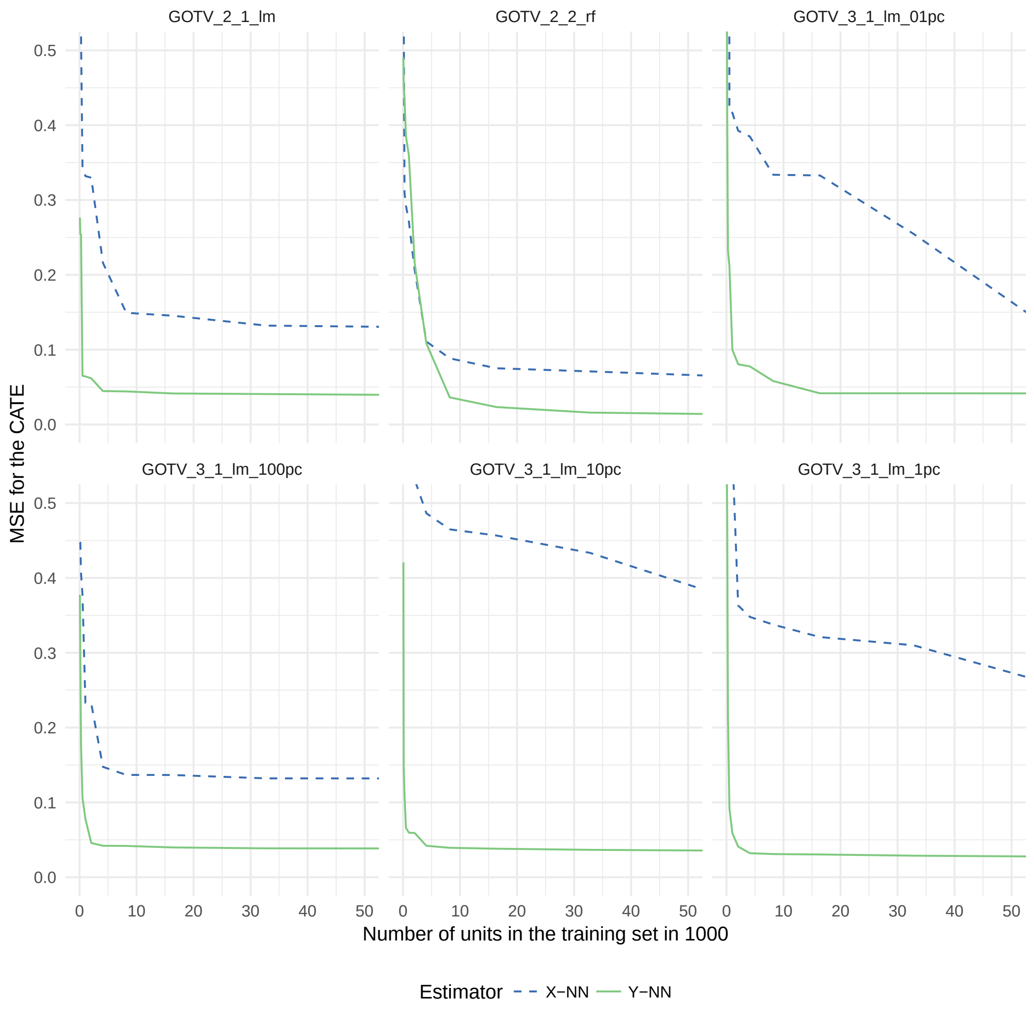


Figure 26: In this figure, we compare the Y and the X learner on six simulated data sets. A precise description on how the data was created can be found in Section 7.4.3

an estimator for the control response functions and then an estimator for the CATE function, these functions are optimized jointly. The pseudo-code in Algorithm 34 shows how these two stages are updated simultaneously. In Figure 26, we compare Y-NN with X-NN, and we find that Y-NN outperforms X-NN for our data sets.

7.3 Transfer Learning

7.3.1 Background

The key idea in transfer learning is that new experiments should transfer insights from previous experiments rather than starting learning anew. The most straightforward example of transfer comes from computer vision [137, 99, 17, 29]. Here, it is standard practice to train a neural network π_θ for one task and then use the trained network weights θ as initialization for a new task. The hope is that some basic low-level features of a vision system should be quite general and reusable. Starting optimization from networks that have already learned these general features should be faster than starting from scratch.

Despite its promise, fine-tuning often fails to produce initializations that are uniformly good for solving new tasks [33]. One potent fix to this problem is a class of algorithms that seek to optimize meta-learning initialization weights [33, 83]. In these algorithms, one meta-optimizes over many experiments to obtain neural network weights that can quickly find solutions to new experiments. We will use the Reptile algorithm to learn initialization weights for CATE estimation.

7.3.2 Transfer Learning CATE Estimators

In this chapter, we consider a scenario wherein one has access to many related causal inference experiments. Across all experiments, the input space X is the same. Let i index an experiment. Each experiment has its own distinct outcome when treatment is received, $\mu_1^i(x)$, and when no treatment is received, $\mu_0^i(x)$. Together, these quantities define the CATE $\tau^i = \mu_1^i - \mu_0^i$, which we want to estimate. We are usually interested in estimating the CATE

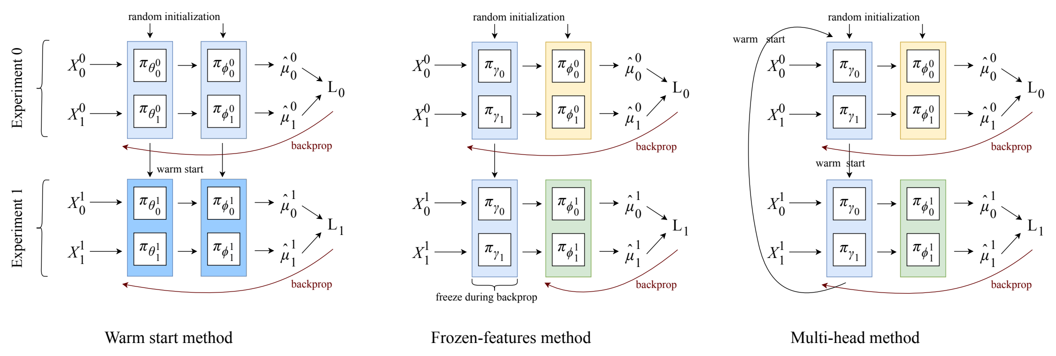


Figure 27: Warm start, frozen-features, and multi-head methods for CATE transfer learning. For these figures, we use the T-learner as the base learner for simplicity. All three methods attempt to reuse neural network features from previous experiments. See below for an illustration of joint-training.

by using X to predict μ_0^i and μ_1^i. However, in transfer learning, the hope is that we can transfer knowledge between experiments such that being able to predict μ_0^i, μ_1^i, and τ^i from experiment i accurately will help us predict μ_0^j, μ_1^j, and τ^j from experiment j.

Below, let π_θ be a generic expression for a neural network parameterized by θ. Sometimes, parameters will have a subscript indicating if their neural network predicts treatment or control (0 for control and 1 for treatment). Parameters may also have a superscript indicating the experiment number whose outcome is being predicted. For example, $\pi_{\theta_0^2}(x)$ predicts $\mu_0^2(x)$, the outcome under control for Experiment 2. All of the transfer algorithms described here are presented in detail below 7.6.2.

Warm start (also known as fine-tuning): Experiment 0 predicts $\pi_{\theta_0^0}(x) = \hat{\mu}_0^0(x)$ and $\pi_{\theta_1^0}(x) = \hat{\mu}_1^0(x)$ to form the CATE estimator $\hat{\tau} = \hat{\mu}_1^0(x) - \hat{\mu}_0^1(x)$. Suppose θ_0^0, θ_1^0 are fully trained and produce a good CATE estimate. For experiment 1, the input space X is identical to the input space for experiment 0, but the outcomes $\mu_0^1(x)$ and $\mu_1^1(x)$ are different. However, we suspect the underlying data representations learned by $\pi_{\theta_0^0}$ and $\pi_{\theta_1^0}$ are still useful. Hence, rather than randomly initialize θ_0^1 and θ_1^1 for experiment 1, we set $\theta_0^1 = \theta_0^0$ and $\theta_1^1 = \theta_1^0$. We then train $\pi_{\theta_0^1}(x) = \hat{\mu}_0^1(x)$ and $\pi_{\theta_1^1}(x) = \hat{\mu}_1^1(x)$. See Figure 27 and Algorithm 6 below.

Frozen-features: Begin by training $\pi_{\theta_0^0}$ and $\pi_{\theta_1^0}$ to produce good CATE estimates for experiment 0. Assuming θ_0^0 and θ_1^0 have more than k layers, let γ_0 be the parameters corresponding to the first k layers of θ_0^0. Define γ_1 analogously. Since we think the features encoded by $\pi_{\gamma_i}(X)$ would make a more informative input than the raw features X, we want to use those features as a transformed input space for $\pi_{\theta_0^1}$ and $\pi_{\theta_1^1}$. To wit, set $z_0 = \pi_{\gamma_0}(x)$ and $z_1 = \pi_{\gamma_1}(x)$. Then form the estimates $\pi_{\theta_0^1}(z_0) = \hat{\mu}_0^1$ and $\pi_{\theta_1^1}(z_1) = \hat{\mu}_1^1$. During training of experiment 1, we only backpropagate through θ_0^1, θ_1^1 and not through the features we borrowed from θ_0^0 and θ_1^0. See Figure 27 and Algorithm 7 below.

Multi-head: In this setup, all experiments share base layers that are followed by experiment-specific layers. The intuition is that the base layers should learn general features, and the experiment-specific layers should transform those features into estimates of μ_j^i. More concretely, let γ_0 and γ_1 be shared base layers. Set $z_0 = \pi_{\gamma_0}(x_0)$ and $z_1 = \pi_{\gamma_1}(x_1)$. The base layers are followed by experiment-specific layers ϕ_0^i and ϕ_1^i. Let $\theta_j^i = \left[\gamma_j, \phi_j^i\right]$. Then $\pi_{\theta_j^i}(x) = \pi_{\phi_j^i}\left(\pi_{\gamma_j}(x)\right) = \pi_{\phi_j^i}(z_j) = \hat{\mu}_j^i$. Training alternates between experiments: each θ_0^i and θ_1^i is trained for some small number of iterations, and then the experiment and head being trained are switched. Every head is usually trained several times. See Figure 27 Algorithm 8 below.

Joint training: All predictions share base layers θ. From these base layers, there are two heads per-experiment i: one to predict μ_0^i and one to predict μ_1^i. Let $\mu_i^j(r)$ represent the rth individual data-point from experiment i with treatment group j. Every head and the base features are trained simultaneously by optimizing with respect to the loss function $\mathcal{L} = \sum_i \sum_r \parallel (\hat{\mu}_0^i(r) - \mu_0^i(r)) \parallel + \sum_i \sum_p \parallel (\hat{\mu}_1^i(p) - \mu_1^i(p)) \parallel$ and minimizing over all weights. This will encourage the base layers to learn generally applicable features and the heads to learn features specific to predicting a single μ_j^i. Note that the summations are occurring over a different set of data, as we do not assume access to both μ_0^i and μ_1^i simultaneously. See Algorithm 4.

SF Reptile transfer for CATE estimators: Similarly to fine-tuning, we no longer provide each experiment with its own weights. Instead, we use data from all experiments to learn weights θ_0 and θ_1, which are good initializers. By good initializers, we mean that starting from θ_0 and θ_1, one can train neural networks π_{θ_0} and π_{θ_1} to estimate μ_0^i and μ_1^i for any arbitrary experiment much faster and with less data than starting from random initializations. To learn these good initializations, we use a transfer learning technique called

Reptile. The idea is to perform experiment-specific inner updates $U(\theta)$ and then aggregate them into outer updates of the form $\theta_{\text{new}} = \epsilon \cdot U(\theta) + (1-\epsilon) \cdot \theta$. In this chapter, we consider a slight variation of Reptile. In standard Reptile, ϵ is either a scalar or correlated to per-parameter weights furnished via SGD. For our problem, we would like to encourage our network layers to learn at different rates. The hope is that the lower layers can learn more general, slowly-changing features like in the frozen features method, and the higher layers can learn comparatively faster features that more quickly adapt to new tasks after ingesting the stable lower-level features. To accomplish this, we take the path of least resistance and make ϵ a vector which assigns a different learning rate to each neural network layer. Because our intuition involves slow and fast weights, we will refer to this modification in this chapter as SF Reptile: Slow Fast Reptile. Though this change is seemingly small, we found it boosted performance on our problems. See Figure 36 and Algorithm 9.

MLRW transfer for CATE estimation: In this method, there exists one single set of weights θ. There are no experiment-specific weights. Furthermore, we do not use separate networks to estimate μ_0 and μ_1. Instead, π_θ is trained to estimate one μ_j^i at a time. We train θ with SF Reptile so that in the future π_θ requires minimal samples to fit μ_j^i from any experiment. To actually form the CATE estimate, we use a small number of training samples to fit π_θ to μ_0^i and then a small number of training samples to fit π_θ to μ_1^i. We call θ **meta-learned regression weights (MLRW)** because they are meta-learned over many experiments to quickly regress onto any μ_j^i. The full MLRW algorithm is presented as Algorithm 3.

7.4 Evaluation on GOTV Data

We evaluate our transfer learning estimators on both real and simulated data. In our data example, we consider the important problem of voter encouragement. Analyzing a large data set of 1.96 million potential voters, we show how transfer learning across elections and geographic regions can dramatically improve our CATE estimators. This example shows that transfer learning can substantially improve the performance of CATE estimators. **To the best of our knowledge, this is the first successful demonstration of transfer learning for CATE estimation.**

7.4.1 GOTV Experiment Background

To evaluate transfer learning for CATE estimation on real data, we reanalyze a set of large field experiments with more than 1.96 million potential voters [38]. The authors conducted 17 experiments to evaluate the effect of a mailer on voter turnout in the 2014 U.S. Midterm Elections. The mailer informs the targeted individual whether or not they voted in the past four major elections (2006, 2008, 2010, and 2012), and it compares their voting behavior with that of the people in the same state. The mailer finishes with a reminder that their voting behavior will be monitored. The idea is that social pressure—i.e., the social norm of voting—will encourage people to vote. The likelihood of voting increases by about 2.2% (s.e.=0.001) when given the mailer.

Each of the experiments target a different state. This results in different populations, different ballots, and different electoral environments. In addition to this, the treatment is slightly different in each experiment, as the median voting behavior in each state is different. However, there are still many similarities across the experiments, so there should be gains from transferring information.

In this example, the input X is a voter's demographic data including age, past voting turnout in 2006, 2008, 2009, 2010, 2011, 2012, and 2013, marital status, race, and gender. The treatment response function $\hat{\mu}_1(x)$ estimates the voting propensity for a potential voter who receives a mailer encouraging them to vote. The control response function $\hat{\mu}_0$ estimates the voting propensity if that voter did not receive a mailer. The CATE τ is thus the change in the probability of voting when a unit receives a mailer. The complete dataset has this data over 17 different states. Treating each state as a separate experiment, we can perform transfer learning across them.

x	outcome	μ_0	μ_1	τ
a voter profile	The voter's propensity to vote	The voter's propensity to vote when they do not receive a mailer	The voter's propensity to vote when they do receive a mailer	Change in the voter's propensity to vote after receiving a mailer

Being able to estimate the treatment effect of sending a mailer is an important problem in elections. We may wish to only treat people whose likelihood of voting would significantly increase when receiving the mailer, to justify the cost for these mailers. Furthermore, we wish to avoid sending mailers to voters who will respond negatively to them. This negative response has been previously observed and is therefore feasible and a relevant problem—e.g., some recipients call their Secretary of State's office or local election registrar to complain [73, 75].

7.4.2 Evaluating CATE Estimators on Real GOTV data

Evaluating a CATE estimator on real data is difficult since one does not observe the true CATE or the individual treatment effect, $Y_i(1) - Y_i(0)$, for any unit because by definition only one of the two outcomes is observed for any unit. One could use the original features and simulate the outcome features, but this would require us to create a response model. Instead, we estimate the "truth" on the real data using linear models (version 1) or random forests (version 2), and we then draw the data based on these estimates. For a detailed description, we refer to section 7.4.3. We then ask the question: how do the various methods perform when they have less data than the entire sample?

We evaluate S-NN, T-NN, and Y-NN using our transfer learning methods. We also added a baseline benchmark which does not use any transfer learning for each of the CATE estimators. In addition to this, we added the S-RF and T-RF as random forest baselines, as well as the Joint estimator and the MLRW estimator, both of which use transfer learning. Figure 28

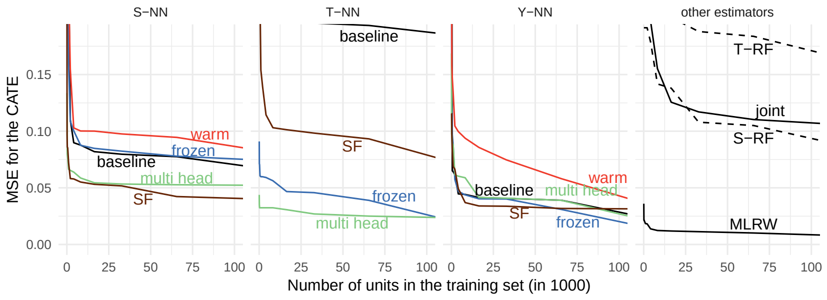


Figure 28: Social Pressure and Voter Turnout, Version 1. Our results far exceed the previous state of the art, which are represented here as S-RF, T-RF, and the baseline method for S-NN and T-NN. Our new methods are Y-NN and the transfer learning methods: warm, frozen, multi-head, joint, SF Reptile, and MLRW.

shows the performance of these estimators when the regression functions were created using a linear model, and Figure 29 shows the same, but the response functions are created using a random forest fitted on the real data.

In previous work, the non-transfer tree-based estimators such as T-RF and S-RF have achieved state of the art results on this problem [65]. For CATE estimation, these methods are very competitive baselines [42]. Happily for us, even non-transfer neural-network-based learners vastly outperform the prior art. In both examples, non-transfer S-NN, T-NN, and Y-NN learners are better or not much worse than T-RF and S-RF. S-NN and Y-NN perform extremely well in this example. Better still, our transfer learning approaches consistently outperform all classical baselines and non-transfer neural network learners on this benchmark. Positive transfer between experiments is readily apparent.

We find that multi-head, frozen features, and SF are usually the best methods to improve an existing neural network-based CATE estimator. **The best estimator is MLRW.** This algorithm consistently converges to a very good solution with very few observations.

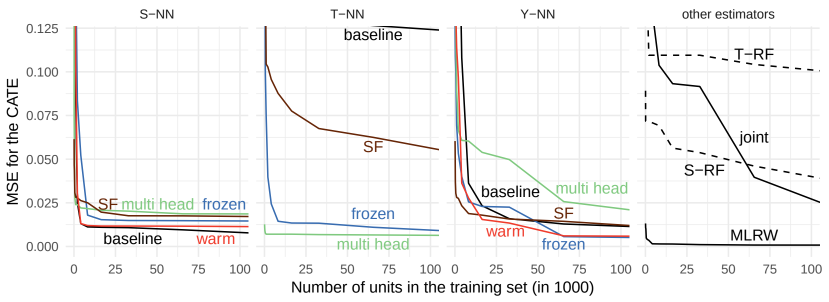


Figure 29: Social Pressure and Voter Turnout, Version 2. Our results exceed the previous state of the art results, which are represented here as S-RF, T-RF, and the baseline method for S-NN and T-NN. Our new methods are Y-NN and the transfer learning methods: warm, frozen, multi-head, joint, SF Reptile, and MLRW.

7.4.3 Data Generating Processes for Our Real World GOTV Data

In this section, we describe how the simulations for the GOTV example in the main paper were done

For our data example, we took one of the experiments conducted by [38]. The study took place in 2014 in Alaska and 252,576 potential voters were randomly assigned in a control and a treatment group. Subjects in the treatment group were sent a mailer as described in the main text and their voting turnout was recorded.

To evaluate the performance of different CATE estimators we need to know the true CATEs, which are unknown due to the fundamental problem of causal inference. To still be able to evaluate CATE estimators researchers usually estimate the potential outcomes using some machine learning method and then generate the data from this estimate. This is to some extent also a simulation, but unlike classical simulation studies it is not up to the researcher to determine the data generating distribution. The only choice of the researcher lies in the type of estimator she uses to estimate the response functions. To avoid being mislead by artifacts created by a particular method, we used a linear model in **Real World Data Set 1** and random forests estimator in **Real World Data Set 2**.

Specifically, we generate for each experiment a true CATE and we simulate new observed outcomes based on the real data in four steps.

1. We first use the estimator of choice (e.g., a random forests estimator) and train it on the treated units and on the control units separately to get estimates for the response functions, μ_0 and μ_1.
2. Next, we sample N units from the underlying experiment to get the features and the treatment assignment of our samples $(X_i, W_i)_{i=1}^N$.
3. We then generate the true underlying CATE for each unit using $\tau_i = \tau(X_i) = \mu_1(X_i) - \mu_0(X_i)$.
4. Finally we generate the observed outcome by sampling a Bernoulli distributed variable around mean μ_i.

$$Y_i^{obs} \sim \text{Bern}(\mu_i), \qquad \mu_i = \begin{cases} \mu_0(X_i) & \text{if } W = 0, \\ \mu_1(X_i) & \text{if } W = 1. \end{cases}$$

After this procedure, we have 17 data sets corresponding to the 17 experiments for which we know the true CATE function, which we can now use to evaluate CATE estimators and CATE transfer learners.

7.4.4 Data Generating Processes for Simulated GOTV Data

In this section, we discuss the results of a much bigger simulation study with 51 experiments which is summarized in Tables 3, 4, and 5.

Simulations motivated by real-world experiments are important to assess whether our methods work well for voter persuasion data sets, but it is important to also consider other settings to evaluate the generalizability of our conclusions.

To do this, we first specify the control response function, $\mu_0(x) = \mathbb{E}[Y(0)|X = x] \in [0, 1]$, and the treatment response function, $\mu_1(x) = \mathbb{E}[Y(1)|X = x] \in [0, 1]$.

We then use each of the 17 experiments to generate a simulated experiment in the following way:

1. We sample N units from the underlying experiment to get the features and the treatment assignment of our samples $(X_i, W_i)_{i=1}^N$.

2. We then generate the true underlying CATE for each unit using $\tau_i = \tau(X_i) = \mu_1(X_i) - \mu_0(X_i)$.
3. Finally we generate the observed outcome by sampling a Bernoulli distributed variable around mean μ_i.

$$Y_i^{obs} \sim \text{Bern}(\mu_i), \qquad \mu_i = \begin{cases} \mu_0(X_i) & \text{if } W = 0, \\ \mu_1(X_i) & \text{if } W = 1. \end{cases}$$

The experiments range in size from 5,000 units to 400,000 units per experiment and the covariate vector is 11 dimensional and the same as in the main part of the chapter. We will present here three different setup.

Simulation LM (Table 3): We choose here N to be all units in the corresponding experiment. Sample $\beta^0 = (\beta_1^0, \ldots, \beta_d^0) \overset{iid}{\sim} \mathcal{N}(0,1)$ and $\beta^1 = (\beta_1^1, \ldots, \beta_d^1) \overset{iid}{\sim} \mathcal{N}(0,1)$ and define,

$$\begin{aligned} \mu_0(x) &= \text{logistic}\left(x\beta^0\right), \\ \mu_1(x) &= \text{logistic}\left(x\beta^1\right). \end{aligned}$$

Simulation RF (Table 4): We choose here N to be all units in the corresponding experiment.

1. Train a random forests estimator on the real data set and define μ_0 to be the resulting estimator,
2. Sample a covariate f (e.g., age),
3. ample a random value in the support of f (e.g., 38),
4. Sample a shift $s \sim \mathcal{N}(0,4)$.

Now define the potential outcomes as follows:

$$\begin{aligned} \mu_0(x) &= \text{trained Random Forests algorithm} \\ \mu_1(x) &= \text{logistic}\left(\text{logit}\left(\mu_0(x) + s * 1_{f \geq v}\right)\right) \end{aligned}$$

Simulation RFt (Table 5): This experiment is the same as Simulation RF, but use only one percent of the data, $N = \frac{\#units}{100}$.

7.4.5 Full Results of 42 Simulated GOTV Experiments

Even though we combine each Simulation setup with 17 experiments, we only report the first 14, because the last three don't add any new insight, but they don't fit well on the page. Looking at Tables 3, 4, and 5, we observe that MLRW is the best performing transfer learner. In fact, for Simulation LM it is the best in 8 out of 17 experiments, in Simulation RF it is the best in 11 out of 17 experiments, and in Simulation RFt it is best in 10 out of 17 experiments. We also notice that in cases, where it is not the best performing estimator, it is usually very close to the best and it does not fail terribly anywhere. For the other transfer method, we note that frozen features, multi-head, and SF works very well and consistently improves upon the baseline learners which are not using outside information. Warm Start, however, does not work well and often even leads to worse results than the baseline estimators.

	Method	**LM-1**	**LM-2**	**LM-3**	**LM-4**	**LM-5**	**LM-6**	**LM-7**	**LM-8**	**LM-9**	**LM-10**	**LM-11**	**LM-12**	**LM-13**	**LM-14**
t-lm		15.95	7.82	20.14	6.62	46.15	17.24	9.88	10.65	44.77	7.63	8.43	9	11.21	20.9
s-rf		19.13	13.18	7.62	13.27	15.66	15.58	11.44	16.3	11.34	16.53	12.57	14.11	13.49	18.56
t-rf		20.04	13.56	7.79	13.62	16	15.99	11.76	16.65	11.54	17.11	13.66	14.38	13.67	18.96
R-NN		18.95	5.19	9.33	28.98	3.04	10.03	4.88	16.34	7.91	14.56	19.23	3.5	10.4	15.53
S-NN	frozen	6.74	6.17	2.75	5.76	5.68	6.27	4.23	7.17	4.15	6.77	4.88	5.9	6.63	9.87
	multi head	3.3	3.05	2.34	3.83	3.6	4.89	4.56	8.47	5.87	5.22	5.43	6.77	5.15	5.58
	SF	4.85	38.65	7.54	55.92	11.36	3.98	50.76	7.74	5.72	7.73	8.64	31.06	30.81	18.08
	baseline	6.84	5.29	3.77	6.86	5.94	7.19	4.6	8.08	5.61	7.12	4.34	6.05	8.59	10.75
	warm	7.29	6.44	4.08	7.25	6.74	7.17	6.03	8.8	5.63	7.6	5.82	6.53	8.55	12.02
T-NN	frozen	6.53	6.73	4.49	6.35	7.23	6.61	5.99	7.59	5.79	6.79	5.99	6.58	7.38	9.39
	multi head	2.73	2.34	1.34	2.11	2.49	2.3	1.97	2.73	1.94	2.36	1.95	2.32	2.65	3.64
	SF	20.72	27.71	8.54	20.18	23.06	15.35	14.27	13.05	9.37	27.94	40.03	16.3	20.81	6.78
	baseline	22.76	5.34	5.98	5.84	5.16	10.37	10.9	7.26	10.25	10.18	6.26	5.69	6.71	10.31
	warm	23.46	7.2	5.75	6.41	5.21	11.56	12.21	8.77	8.93	6.81	8.93	6.15	7.25	18.98
X-NN	frozen	6.63	14.04	10.73	19.57	17.94	14	13.67	18.83	14.36	10.81	12.25	16.61	39.96	32.81
	multi head	1.19	11.38	84.13	174.18	1.87	19.55	62.12	22.7	9.67	**0.85***	3.34	5.03	**0.94***	111.42
	SF	18.83	10.72	10.3	10.61	10.11	12.5	21.37	11.12	8.27	16.33	9.81	13.8	9.85	10.15
	baseline	19.52	8.25	4.68	5.06	6.6	11.5	10.78	12	10.26	6.25	13.11	7.27	9.2	18.45
	warm	20.06	8.54	5.57	6.37	10.77	9.34	11.36	13.16	8.03	8.66	10.96	7.52	11.57	16.7
Y-NN	frozen	1.54	**1***	2.1	3.3	**1.11***	46.07	27.63	5.47	7.48	7.21	1.02	**1.15***	0.97	43.82
	multi head	0.92	2.21	1.26	15.86	1.19	20.47	1.76	2.75	3.96	9.4	19.11	1.26	35.43	9.29
	SF	5.68	12	16.57	38	5.54	3.49	8.2	4.8	2.61	7.21	8.48	12.98	9.58	5.08
	baseline	**0.9***	1.31	5.24	31.43	6.8	**1.23***	7.5	**1.07***	1.32	1.35	8.19	1.2	4.11	7.72
	warm	48.91	1.26	5.79	3.61	29.43	2.71	3.94	12.87	21.76	13.25	15.78	17.45	1.12	29.01
joint	joint	13.75	5.81	3.46	4.12	3.46	12.22	9.58	14.96		4.75	8.55	13.13	9.91	11.68
MLRW		1	1.41	**1.05***	**1.94***	1.97	1.26	**1.03***	2.05	**0.9***	1.85	**1***	1.15	1.57	**2.75***

Table 3: MSE in percent for different CATE estimators.

	Method	RF-1	RF-2	RF-3	RF-4	RF-5	RF-6	RF-7	RF-8	RF-9	RF-10	RF-11	RF-12	RF-13	RF-14
t-lm		17.95	26.73	2.48		4.84	1.76	6.2	7.55	24.58	3.5	2.35	2.1	3.39	7.41
s-rf		7.18	7	4.18	6.86	8.43	9.15	6.03	9.28	5.95	8.49	6.7	8.1	6.46	8.64
t-rf		10.95	7.76	4.69	7.79	9.03	10.07	6.67	10.08	6.3	10.35	8.19	9.28	6.73	10.04
R-NN		11.41	1.18	5.7	1.26	0.7	3.79	9.11	8.01	7.17	5.14	4.15	0.52	1.26	6.88
S-NN	frozen	1.56	0.66	0.7	0.77	0.87	0.83	0.91	0.89	0.59	0.87	0.6	0.92	0.91	1.23
	multi head	0.69	1.23	0.4	1.59	1.58	0.65	0.81	0.57	**0.28***	1.98	1.08	1.24	2.27	3.66
	SF	1.36	9.64	4.26	4.53	6	6.74	11.68	2.8	12.01	41.6	45.72	4.59	1.98	6.75
	baseline	0.91	1.28	0.84	0.93	1.78	2.05	1.16	2.04	1.61	1.06	1.29	1.49	1.99	2.69
	warm	1.08	1.3	0.94	1.16	1.85	2.22	1.4	2.15	1.65	1.12	1.37	1.42	1.83	2.52
T-NN	frozen	1.55	1.02	0.62	0.95	1.11	0.99	0.89	1.18	0.86	1.03	0.89	1.01	1.14	1.49
	multi head	0.66	0.58	0.35	0.53	0.63	0.53	0.47	0.63	0.49	0.57	0.51	0.55	0.64	0.91
	SF	11.66	27.5	6.89	22.59	20.99	12.04	19.77	7.85	5.39	17.31	17.15	9.1	30.42	3.55
	baseline	7.83	4.25	1.44	1.47	1.35	1.33	2.05	8.55	1.79	3.21	2.29	3.62	2.03	8.7
	warm	12.74	3.32	1.33	1.58	1.06	1.6	2.19	9.28	1.52	2.77	3.73	4.99	1.64	8.74
X-NN	frozen	2.53	22.89	55.57	44.11	4.18	5.72	33.18	2.62	7.59	4.96	4.41	2.01	3.01	1.23
	multi head	3.45	2.53	47.09	39.7	2	27.62	10.39	0.78	11.8	10.85	0.93	9.72	11.74	30.62
	SF	4.34	1.85	5.47	6.82	4.21	11.89	7.91	5.02	4.67	6.82	16.39	6.99	5.22	2.11
	baseline	4.09	2.05	1.67	0.73	0.58	1.16	4.97	9.23	4.05	1.49	5.17	2.15	4.07	8.05
	warm	2.51	1.73	1.72	1.48	0.97	1.04	7.81	4.35	4.59	1.34	3.86	1.37	2.08	4.83
Y-NN	frozen	0.42	**0.32***	10.65	0.72	36	1.62	0.82	0.87	15.19	1.61	0.77	46.3	28.37	1.98
	multi head	29.41	9.71	2.74	12.27	48.87	16.59	50.21	73.74	1.65	1.76	3.52	24.26	0.76	216.69
	SF	5.45	2.51	1.53	4.76	9.61	20.81	3.68	4.84	10.8	2.87	2.78	1.36	9.07	4.08
	baseline	0.71	1.06	1.54	0.57	2.44	0.47	0.73	1.38	1.25	1.77	0.71	0.29	0.63	1.98
	warm	0.9	27.01	0.52	22.7	1.37	24.94	**0.3***	2.08	12.36	3.39	9.58	2.74	2.47	11.48
joint	joint		33.47	1.3	0.61	5.15	0.47	2.68		2.67	**0.37***	7.76	28.14	7.48	10.04
MLRW		**0.37***	1.12	**0.18***	**0.3***	**0.46***	**0.21***	0.42	**0.45***	0.37	1.18	**0.35***	**0.23***	0.26	**0.16***

Table 4: MSE in percent for different CATE estimators.

	Method	**RFt-1**	**RFt-2**	**RFt-3**	**RFt-4**	**RFt-5**	**RFt-6**	**RFt-7**	**RFt-8**	**RFt-9**	**RFt-10**	**RFt-11**	**RFt-12**	**RFt-13**	**RFt-14**
t-lm		22.65	2.74	2.3	5.07	4.78	8.3	9.77	38.73	23.94	8.94	83.37	4.26	4.31	31.76
s-rf		3.54	7.04	4.39	6.4	8.09	6.41	11.44	6.33	6.05	6.52	5.09	9.29	11.64	4.28
t-rf		13.3	13.55	8.41	11.9	14.11	15.48	14.42	12.54	10.19	18.99	12.48	14.64	13.61	12.02
R-NN		62.08	17.68	5.22	19.91	14.43	20.53	18.51	49.63	8.99	86.89	54.17	23.86	3.02	18.98
S-NN	frozen	2.49	51.39	37.47	49.5	43.17	29.19	21.81	27.25	61.34	27.02	17.25	64.22	24.94	11.24
	multi head	1.4	0.57	0.73	1.48	1.54	1.27	1.1	1.63	0.88	1.07	1.53	0.79	0.74	1.59
	SF	0.84	68.31	4.12	10.77	17.28	20.06	10.22	1.08	8.95	9.11	5.97	19.67	11.64	8.68
	baseline	14.66	2.08	0.95	1.38	2.24	7.3	3.17	1.08	0.88	5.29	1.44	2.37	2.85	1.78
	warm	17.64	2.76	1.38	2.31	1.99	8.64	4.32	1.37	0.85	5.5	3.76	5.44	3.13	3.49
T-NN	frozen	2.57	1.69	1.12	1.55	1.81	1.6	1.52	1.86	1.43	1.63	1.47	1.68	1.85	2.27
	multi head	0.69	0.55	0.34	0.49	0.63	0.64	0.51	0.64	0.5	0.53	**0.46***	**0.55***	0.66	0.8
	SF	18.04	9.54	7.33	6.61	22.11	9.21	20.2	10.65	15.06	13.87	27.64	17.96	14.08	4.49
	baseline	43.37	14.55	9.48	19.35	15.68	20.91	16	41.72	6.99	30.02	45.59	16.03	4.79	18.63
	warm	69.41	19.81	12.99	24.12	20.28	37.07	19.64	39.37	9.36	40.65	37.59	18.62	6.87	32.52
X-NN	frozen	12.41	221.49	215.01	131.46	94.89	328.53	199.48	41.14	398.33	83.75	132	434.25	125.88	168.92
	multi head	12.24	0.79	3.22	12.46	2.87	12.11	21.38	1.86	1.48	18.46	1.82	99.38	14.67	28.43
	SF	3.98	11.03	4.33	5.31	5.42	4.67	14.03	5.99	7	7	6.42	7.21	4.89	2.34
	baseline	60.44	15.03	4.93	7.08	9.51	21.22	14.7	16.83	8	80.71	23.89	25.5	4.67	17.1
	warm	30.46	13.02	5.11	9.95	9.21	13.12	9.95	12.8	4.48	50.45	19.27	12.92	8.1	15.73
Y-NN	frozen	1.72	**0.33***	0.87	2.47	1.8	5.44	1.35	19.37	37.98	0.81	0.7	33.11	1.34	13.64
	multi head	5.95	11.12	0.41	2.94	2.49	24.73	3.44	58.63	12.99	0.41	9.55	0.6	3.93	0.49
	SF	3.98	14.56	12.85	4.3	3.66	9.78	11.13	3.03	5.78	3.37	11.87	11.33	4.51	2.64
	baseline	0.88	10.92	0.28	1.32	1.89	11.35	8.42	1.6	5.66	0.54	0.54	2.01	0.61	2.13
	warm	8.03	0.48	0.36	0.71	2.53	1.21	**0.31***	1.77	**0.41***	0.74	23.48	1.4	1.15	17.72
joint	joint	102.83	38.47	133.88	15.31	37.32	410.04	38.28	14.96	5.02	11.97	33.71	26.33	11.99	147.45
MLRW		**0.31***	1.25	**0.21***	**0.27***	**0.35***	**0.29***	0.33	**0.61***	0.61	**0.4***	0.57	1.32	0.8	**0.24***

Table 5: MSE in percent for different CATE estimators.

7.5 Evaluation on MNIST Example

In the previous experiment, we observed that the MLRW estimator performed most favorably and transfer learning significantly improved upon the baseline. To confirm that this conclusion is not specific to voter persuasion studies, we consider in this section intentionally a very different type of data. The simulated data has been intentionally chosen to be different in character from our real-world example. In particular, the simulated input space is images and the estimated outcome variable is continuous.

Recently, [84] introduced a simulation study wherein MNIST digits are rotated by some number of degrees α; with α furnished via a single data generating process that depends on the value of the depicted digit. They then attempt to do CATE estimation to measure the heterogeneous treatment effect of a digit's label.

Motivated by this example, we develop a data generating process using MNIST digits wherein transfer learning for CATE estimation is applicable. In our example, the input X is an MNIST image. We have k data generating processes which return different outcomes for each input when given either treatment or control. Thus, under some fixed data generating process, μ_0 represents the outcome when the input image X is given the control, μ_1 represents the outcome when X is given the treatment, and τ is the difference in outcomes given the placement of X in the treatment or control group. Each data generating process has different response functions (μ_0 and μ_1) and thus different CATEs (τ), but each of these functions only depend on the image label presented in the image X. We thus hope that transfer learning could expedite the process of learning features which are indicative of the label. See section 7.5.1 for full details of the data generation process. In Figure 30 of section 7.5.1, we confirm that a transfer learning strategy outperforms its non-transfer learning counterpart, even on image data, and also that MLRW performs well.

7.5.1 Data Generation Procedure for MNIST Experiment

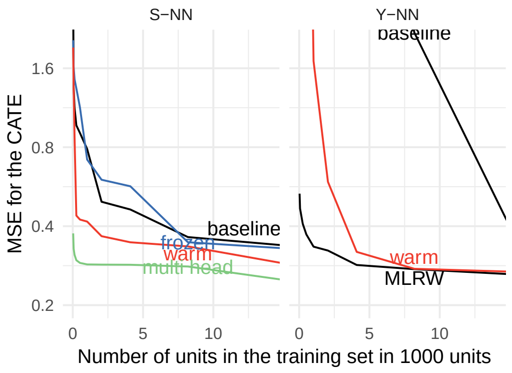


Figure 30: MNIST task

For our MNIST simulation study (Section 7.5), we used the MNIST database [67] which contains labeled handwritten images. We follow here the notation of [84], who introduce a very similar simulation study which is not trying to evaluate transfer learning for CATE estimation, but instead emulates a RCT with the goal to evaluate different CATE estimators.

The MNIST data set contains labeled image data (X_i, C_i), where X_i denotes the raw image of i and $C_i \in \{0, \ldots, 9\}$ denotes its label. We create k Data Generating Processes (DGPs), $D_1, \ \ldots, \ D_k$, each of which specifies a distribution of $(Y_i(0), Y_i(1), W_i, X_i)$ and represents different CATE estimation problems.

In this simulation, we let $W_i = 0$ if the image X_i is placed in the control, and $W_i = 1$ if the image X_i is placed in the treatment. $Y_i(W_i)$ quantifies the the outcome of X_i under W_i.

To generate a DGP D_j , we first sample weights in the following way,

$$\begin{aligned} m^j(0), \ m^j(1), \ldots, \ m^j(9) &\overset{iid}{\sim} \text{Unif}(-3, \ 3), \\ t^j(0), \ t^j(1), \ldots, \ t^j(9) &\overset{iid}{\sim} \text{Unif}(-1, \ 1), \\ p^j(0), \ p^j(1), \ldots, \ p^j(9) &\overset{iid}{\sim} \text{Unif}(0.3, \ 0.7), \end{aligned}$$

and we define the response functions and the propensity score as

$$\begin{aligned}
\mu_0^j(C_i) &= m^j(C_i) + 3C_i, \\
\mu_1^j(C_i) &= \mu_0^j(C_i) + t^j(C_i), \\
e^j(C_i) &= p^j(C_i).
\end{aligned}$$

To generate $(Y_i(0), Y_i(1), W_i, X_i)$ from D_j, we fist sample a (X_i, C_i) from the MNIST data set, and we then generate $Y_i(0), Y_i(1)$, and W_i in the following way:

$$\begin{aligned}
\varepsilon_i &\overset{iid}{\sim} \mathcal{N}(0,1) \\
Y_i(0) &= \mu_0(C_i) + \varepsilon_i \\
Y_i(1) &= \mu_1(C_i) + \varepsilon_i \\
W_i &\sim \text{Bern}(e(C_i)).
\end{aligned}$$

During training, X_i, W_i, and Y_i are made available to the convolutional neural network, which then predicts $\hat{\tau}$ given a test image X_i and a treatment W_i. τ is the difference in the outcome given the difference in treatment and control.

Having access to multiple DGPs can be interpreted as having access to prior experiments done on a similar population of images, allowing us to explore the effects of different transfer learning methods when predicting the effect of a treatment in a new image.

7.6 Psuedo Code

7.6.1 Pseudo Code for CATE Estimators

We will present pseudo code for the standard CATE estimators in this section. We present code for the transfer learning algorithms in Section 7.6.2. To be clear, the algorithms below are vanilla CATE estimators. There are no transfer learning algorithms in this section. We denote by Y^0 and Y^1 the observed outcomes for the control and the treated group. For example, Y_i^1 is the observed outcome of the ith unit in the treated group. X^0 and X^1 are the features of the control and treated units, and hence, X_i^1 corresponds to

Algorithm 1 T-learner

```
1: procedure T–LEARNER($X, Y^{obs}, W$)
2:     $\hat{\mu}_0 = M_0(Y^0 \sim X^0)$
3:     $\hat{\mu}_1 = M_1(Y^1 \sim X^1)$

4:     $\hat{\tau}(x) = \hat{\mu}_1(x) - \hat{\mu}_0(x)$
5: end procedure
```

M_0 and M_1 are here some, possibly different machine learning/regression algorithms.

Figure 31

Algorithm 2 S-learner

```
1: procedure S–LEARNER($X, Y^{obs}, W$)
2:     $\hat{\mu} = M(Y^{obs} \sim (X, W))$
3:     $\hat{\tau}(x) = \hat{\mu}(x, 1) - \hat{\mu}(x, 0)$
4: end procedure
```

$M(Y^{obs} \sim (X, W))$ is the notation for estimating $(x, w) \mapsto \mathbb{E}[Y|X = x, W = w]$ while treating W as a 0,1–valued feature.

Figure 32

the feature vector of the ith unit in the treated group. $M_k(Y \sim X)$ is the notation for a regression estimator, which estimates $x \mapsto \mathbb{E}[Y|X = x]$. It can be any regression/machine learning estimator, but in this chapter we only choose it to be a neural network or random forest.

Algorithm 3 X–learner

```
1: procedure X–LEARNER($X, Y^{obs}, W, g$)
2:     $\hat{\mu}_0 = M_1(Y^0 \sim X^0)$                                   ▷ Estimate response function
3:     $\hat{\mu}_1 = M_2(Y^1 \sim X^1)$
4:     $\tilde{D}_i^1 = Y_i^1 - \hat{\mu}_0(X_i^1)$                         ▷ Compute imputed treatment effects
5:     $\tilde{D}_i^0 = \hat{\mu}_1(X_i^0) - Y_i^0$
6:     $\hat{\tau}_1 = M_3(\tilde{D}^1 \sim X^1)$                           ▷ Estimate CATE for treated and control
7:     $\hat{\tau}_0 = M_4(\tilde{D}^0 \sim X^0)$
8:     $\hat{\tau}(x) = g(x)\hat{\tau}_0(x) + (1 - g(x))\hat{\tau}_1(x)$    ▷ Average the estimates
9: end procedure
```

$g(x) \in [0, 1]$ is a weighing function which is chosen to minimize the variance of $\hat{\tau}(x)$. It is sometimes possible to estimate Cov$(\tau_0(x), \tau_1(x))$, and compute the best g based on this estimate. However, we have made good experiences by choosing g to be an estimate of the propensity score, but also choosing it to be constant and equal to the ratio of treated units usually leads to a good estimator of the CATE.

Figure 33

Algorithm 4 Y-Learner Pseudo Code

```
1: if $W_i == 0$ then
2:     Update the network $\pi_{\theta_0}$ to predict $Y_i^{obs}$
3:     Update the network $\pi_{\theta_1}$ to predict $Y_i^{obs} + \pi_\tau(X_i)$
4:     Update the network $\pi_\tau$ to predict $\pi_{\theta_1}(X_i) - Y_i^{obs}$
5: end if
6: if $W_i == 1$ then
7:     Update the network $\pi_{\theta_0}$ to predict $Y_i^{obs} - \pi_\tau(X_i)$
8:     Update the network $\pi_{\theta_1}$ to predict $Y_i^{obs}$
9:     Update the network $\pi_\tau$ to predict $Y_i^{obs} - \pi_{\theta_0}(X_i)$
10: end if
```

This process describes training the Y-Learner for one step given a data point (Y_i^{obs}, X_i, W_i)

Figure 34

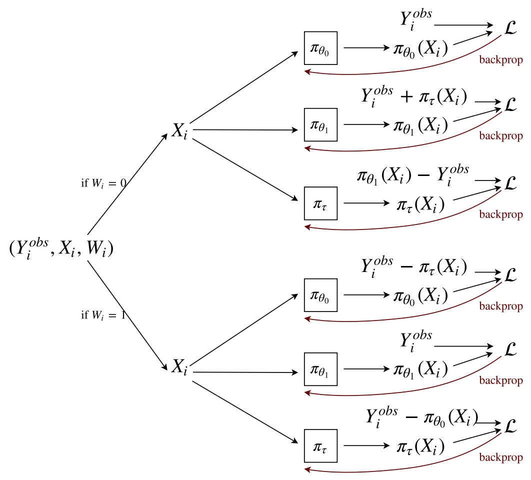


Figure 35: Y-learner with Neural Networks

7.6.2 MLRW and Joint Training Pseudo Code

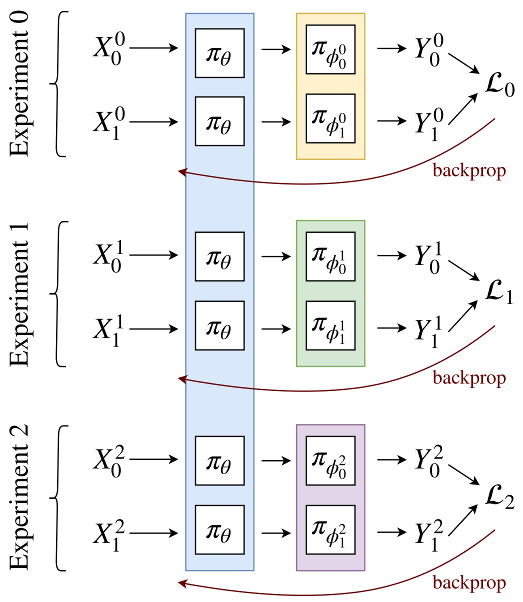


Figure 36: Joint Training - Unlike the Multi-head method which differentiates base layers for treatment and control, the Joint Training method has all observations and experiments (regardless of treatment and control) share the same base network, which extracts general low level features from the data.

Algorithm 3 MLRW Transfer for Cate Estimation.

```
 1: Let $\mu_0^{(i)}$ and $\mu_1^{(i)}$ be the outcome under treatment and control for experiment $i$.
 2: Let numexps be the number of experiments.
 3: Let $\pi_\theta$ be an $N$ layer neural network parameterized by $\theta = [\theta_0, \ldots, \theta_N]$.
 4: Let $\epsilon = [\epsilon_0, \ldots, \epsilon_N]$ be a vector, where $N$ is the number of layers in $\pi_\theta$.
 5: Let outeriters be the total number of training iterations.
 6: Let inneriters be the number of inner loop training iterations.
 7: for oiter < outeriters do
 8:     for i < numexps do
 9:         Sample $X_0$ and $X_1$: control and treatment units from experiment $i$
10:         for $j < 2$ do
11:             # # j iterating over treatment and control
12:             Let $U_0(\theta) = \theta$
13:             for k ¡ inneriters do
14:                 $\mathcal{L} = \|\pi_{U_k(\theta)}(X_j) - \mu_j(X_j)\|$
15:                 Compute $\nabla_\theta \mathcal{L}$.
16:                 Use ADAM with $\nabla_\theta \mathcal{L}$ to obtain $U_{k+1}(\theta)$.
17:                 $U_k(\theta) = U_{k+1}(\theta)$
18:             end for
19:             for $p < N$ do
20:                 $\theta_p = \epsilon_p \cdot U_k(\theta_p) + (1 - \epsilon_p) \cdot \theta_p$.
21:             end for
22:         end for
23:     end for
24: end for
25: To Evaluate CATE estimate, do
26: $C = []$
27: for i < numexps do
28:     Sample $X_0$ and $X_1$: control and treatment units from experiment $i$
29:     Sample $X$: test units from experiment $i$.
30:     for $j < 2$ do
31:         for $k <$ innteriters do
32:             $\mathcal{L} = \|\pi_{U_k(\theta)}(X_j) - \mu_j(X_j)\|$
33:             Compute $\nabla_\theta \mathcal{L}$.
34:             Use ADAM with $\nabla_\theta \mathcal{L}$ to obtain $U_{k+1}(\theta)$.
35:             $U_k(\theta) = U_{k+1}(\theta)$
36:         end for
37:         $\hat{\mu}_j = \pi_{U_k(\theta)}(X)$
38:     end for
39:     $\hat{\tau}_i = \hat{\mu}_0 - \hat{\mu}_1$
40:     C.append($\hat{\tau}_i$)
41: end for
42: return $C$
```

Algorithm 4 Joint Training

```
1: Let $\mu_0^{(i)}$ and $\mu_1^{(i)}$ be the outcome under control and treatment for experiment $i$.
2: Let numexps be the number of experiments.
3: Let $\pi_\rho$ be a generic expression for a neural network parameterized by $\rho$.
4: Let $\theta$ be base neural network layers shared by all experiments.
5: Let $\phi_0^{(i)}$ be neural network layers predicting $\mu_0^{(i)}$ in experiment $i$.
6: Let $\phi_1^{(i)}$ be neural network layers predicting $\mu_1^{(i)}$ in experiment $i$.
7: Let $\omega_0^{(i)} = \left[\theta, \phi_0^{(i)}\right]$ be the full prediction network for $\mu_0$ in experiment $i$.
8: Let $\omega_1^{(i)} = \left[\theta, \phi_1^{(i)}\right]$ be the full prediction network for $\mu_1$ in experiment $i$.
9: Let $\Omega = \bigcup_{j=0}^{1} \bigcup_{i=1}^{\text{numexps}} \omega_j^{(i)}$ be all trainable parameters.
10: Let numiters be the total number of training iterations
11: for iter < numiters do
12:     $\mathcal{L} = 0$
13:     for i < numexps do
14:         Sample $X_0$ and $X_1$: control and treatment units from experiment $i$
15:         for $j < 2$ do
16:             # # $j$ iterating over treatment and control
17:             $\mathcal{L}_j^{(i)} = \|\pi_{\omega_j^{(i)}}(X_j) - \mu_j(X_j)\|$
18:             $\mathcal{L} = \mathcal{L} + \mathcal{L}_j^{(i)}$
19:         end for
20:     end for
21:     Compute $\nabla_\Omega \mathcal{L} = \frac{\partial \mathcal{L}}{\partial \Omega} = \sum_i \sum_j \frac{\partial \mathcal{L}_j^{(i)}}{\partial \omega_j^{(i)}}$
22:     Apply ADAM with gradients given by $\nabla_\Omega \mathcal{L}$.
23:     for i < numexps do
24:         Sample $X$: test units from experiment $i$
25:     end for
26: end for
27: $\hat{\mu}_0 = \pi_{\omega_0^{(i)}}(X)$
28: $\hat{\mu}_1 = \pi_{\omega_1^{(i)}}(X)$
29: return CATE estimate $\hat{\tau} = \hat{\mu}_1 - \hat{\mu}_0$
```

7.6.3 Pseudo-code for T-learner Transfer CATE Estimators

Here, we present full pseudo code for the algorithms from Section 3 using the T-learner as a base learner. All of these algorithms can be extended to other learners including S, R, X, and Y. See the released code for implementations.

Algorithm 5 Vanilla T-learner

```
 1: Let $\mu_0$ and $\mu_1$ be the outcome under treatment and control.
 2: Let $X$ be the experimental data. Let $X_t$ be the test data.
 3: Let $\pi_{\theta_0}$ and $\pi_{\theta_1}$ be a neural networks parameterized by $\theta_0$ and $\theta_1$.
 4: Let $\theta = \theta_0 \cup \theta_1$.
 5: Let numiters be the total number of training iterations.
 6: Let batchsize be the number of units sampled. We use 64.
 7: for i < numiters do
 8:     Sample $X_0$ and $X_1$: control and treatment units. Sample batchsize
        units.
 9:     $\mathcal{L}_0 = \|\pi_\theta(X_0) - \mu_0(X_0)\|$
10:     $\mathcal{L}_1 = \|\pi_\theta(X_1) - \mu_1(X_1)\|$
11:     $\mathcal{L} = \mathcal{L}_0 + \mathcal{L}_1$
12:     Compute $\nabla_\theta \mathcal{L} = \frac{\partial \mathcal{L}}{\partial \theta}$.
13:     Apply ADAM with gradients given by $\nabla_\theta \mathcal{L}$.
14: end for
15: $\hat{\mu}_0 = \pi_{\theta_0}(X_t)$
16: $\hat{\mu}_1 = \pi_{\theta_1}(X_t)$
17: return CATE estimate $\hat{\tau} = \hat{\mu}_1 - \hat{\mu}_0$
```

Algorithm 6 Warm Start T-learner

```
1: Let $\mu_0^i$ and $\mu_1^i$ be the outcome under treatment and control for experiment $i$.
2: Let $X^i$ be the data for experiment $i$. Let $X_t^i$ be the test data for experiment $i$.
3: Let $\pi_{\theta_0}$ and $\pi_{\theta_1}$ be a neural networks parameterized by $\theta_0$ and $\theta_1$.
4: Let $\theta = \theta_0 \cup \theta_1$.
5: Let numiters be the total number of training iterations.
6: Let batchsize be the number of units sampled. We use 64.
7: for i < numiters do
8:     Sample $X_0^0$ and $X_1^0$: control and treatment units for experiment 0. Sample batchsize units.
9:     $\mathcal{L}_0 = \|\pi_{\theta_0}(X_0^0) - \mu_0(X_0^0)\|$
10:    $\mathcal{L}_1 = \|\pi_{\theta_1}(X_1^0) - \mu_1(X_1^0)\|$
11:    $\mathcal{L} = \mathcal{L}_0 + \mathcal{L}_1$
12:    Compute $\nabla_\theta \mathcal{L} = \frac{\partial \mathcal{L}}{\partial \theta}$.
13:    Apply ADAM with gradients given by $\nabla_\theta \mathcal{L}$.
14: end for
15: for i < numiters do
16:    Sample $X_0^1$ and $X_1^1$: control and treatment units for experiment 1. Sample batchsize units.
17:    $\mathcal{L}_0 = \|\pi_{\theta_0}(X_0^1) - \mu_0(X_0^1)\|$
18:    $\mathcal{L}_1 = \|\pi_{\theta_1}(X_1^1) - \mu_1(X_1^1)\|$
19:    $\mathcal{L} = \mathcal{L}_0 + \mathcal{L}_1$
20:    Compute $\nabla_\theta \mathcal{L} = \frac{\partial \mathcal{L}}{\partial \theta}$.
21:    Apply ADAM with gradients given by $\nabla_\theta \mathcal{L}$.
22: end for
23: $\hat{\mu}_0 = \pi_{\theta_0}(X_t^1)$
24: $\hat{\mu}_1 = \pi_{\theta_1}(X_t^1)$
25: return CATE estimate $\hat{\tau} = \hat{\mu}_1 - \hat{\mu}_0$
```

Algorithm 7 Frozen Features T-learner

```
1: Let $\mu_0^i$ and $\mu_1^i$ be the outcome under treatment and control for experiment $i$.
2: Let $X^i$ be the data for experiment $i$. Let $X_t^i$ be the test data for experiment $i$.
3: Let $\pi_\rho$ be a generic expression for a neural network parameterized by $\rho$.
4: Let $\theta_0^0, \theta_0^1, \theta_1^0, \theta_1^1$ be neural network parameters. The subscript indicates the outcome that $\theta$ is associated with predicting (0 for control and 1 for treatment) and the superscript indexes the experiment.
5: Let $\gamma_0$ be the first $k$ layers of $\pi_{\theta_0^0}$. Define $\gamma_1$ analogously.
6: Let $\theta^i = \theta_0^i \cup \theta_1^i$.
7: Let numiters be the total number of training iterations.
8: Let batchsize be the number of units sampled. We use 64.
9: **for** i < numiters **do**
10:     Sample $X_0^0$ and $X_1^0$: control and treatment units for experiment 0. Sample batchsize units.
11:     $\mathcal{L}_0 = \|\pi_{\theta_0^0}(X_0^0) - \mu_0(X_0^0)\|$
12:     $\mathcal{L}_1 = \|\pi_{\theta_1^0}(X_1^0) - \mu_1(X_1^0)\|$
13:     $\mathcal{L} = \mathcal{L}_0 + \mathcal{L}_1$
14:     Compute $\nabla_\theta \mathcal{L} = \frac{\partial \mathcal{L}}{\partial \theta}$.
15:     Apply ADAM with gradients given by $\nabla_{\theta^0} \mathcal{L}$.
16: **end for**
17: **for** i < numiters **do**
18:     Sample $X_0^1$ and $X_1^1$: control and treatment units for experiment 1. Sample batchsize units.
19:     Compute $Z_0^1 = \pi_\gamma(X_0^1)$ and $Z_1^1 = \pi_\gamma(X_1^1)$
20:     $\mathcal{L}_0 = \|\pi_{\theta_0^1}(Z_0^1) - \mu_0(Z_0^1)\|$
21:     $\mathcal{L}_1 = \|\pi_{\theta_1^1}(Z_1^1) - \mu_1(Z_1^1)\|$
22:     $\mathcal{L} = \mathcal{L}_0 + \mathcal{L}_1$
23:     Compute $\nabla_{\theta^1} \mathcal{L} = \frac{\partial \mathcal{L}}{\partial \theta^1}$. Do not compute gradients with respect to $\theta^0$ parameters.
24:     Apply ADAM with gradients given by $\nabla_{\theta^1} \mathcal{L}$.
25: **end for**
26: Compute $Z_t^1 = \pi_\gamma(X_t^1)$.
27: $\hat{\mu_0} = \pi_{\theta_0^1}(Z_t^1)$
28: $\hat{\mu_1} = \pi_{\theta_1^1}(Z_t^1)$
29: **return** CATE estimate $\hat{\tau} = \hat{\mu_1} - \hat{\mu_0}$
```

Algorithm 8 Multi-Head T-learner

```
1: Let $\mu_0^i$ and $\mu_1^i$ be the outcome under treatment and control for experiment $i$.
2: Let $X^i$ be the data for experiment $i$. Let $X_t^i$ be the test data for experiment $i$.
3: Let $\pi_\rho$ be a generic expression for a neural network parameterized by $\rho$.
4: Let $\theta_0$ be base neural network layers shared by all experiments for predicting outcomes
   under control.
5: Let $\theta_1$ be base neural network layers shared by all experiments for predicting outcomes
   under treatment.
6: Let $\phi_0^{(i)}$ be neural network layers receiving $\pi_{\theta_0}(x_0^i)$ as input and predicting $\mu_0^{(i)}(x_0^i)$ in
   experiment $i$.
7: Let $\phi_1^{(i)}$ be neural network layers receiving $\pi_{\theta_1}(x_1^i)$ as input and predicting $\mu_1^{(i)}(x_1^i)$ in
   experiment $i$.
8: Let $\omega_0^{(i)} = \left[\theta, \phi_0^{(i)}\right]$ be all trainable parameters used to predict $\mu_0^i$.
9: Let $\omega_1^{(i)} = \left[\theta, \phi_1^{(i)}\right]$ be all trainable parameters used to predict $\mu_1^i$.
10: Let $\Omega^i = \omega_0^{(i)} \cup \omega_1^{(i)}$.
11: Let numiters be the total number of training iterations.
12: Let batchsize be the number of units sampled. We use 64.
13: Let numexps be the number of experiments.
14: for i < numiters do
15:     for j < numexps do
16:         Sample $X_0^j$ and $X_1^j$: control and treatment units for experiment $j$. Sample
            batchsize units.
17:         Compute $Z_0^j = \pi_{\theta_0}(X_0^j)$ and $Z_1^j = \pi_{\theta_1}(X_1^j)$
18:         Compute $\hat{\mu}_0^j = \pi_{\phi_0^j}(z_0^j)$ and $\hat{\mu}_1^j = \pi_{\phi_1^j}(z_1^j)$
19:         $\mathcal{L}_0 = \|\hat{\mu}_0^j - \mu_0^j(X_0^j)\|$
20:         $\mathcal{L}_1 = \|\hat{\mu}_1^j - \mu_1^j(X_1^j)\|$
21:         $\mathcal{L} = \mathcal{L}_0 + \mathcal{L}_1$
22:         Compute $\nabla_{\Omega^i}\mathcal{L} = \frac{\partial \mathcal{L}}{\partial \Omega^i}$.
23:         Apply ADAM with gradients given by $\nabla_\theta \mathcal{L}$.
24:     end for
25: end for
26: Let $C = []$
27: for j < numexps do
28:     Compute $Z_0^j = \pi_{\theta_0}(X_t^j)$ and $Z_1^j = \pi_{\theta_1}(X_t^j)$
29:     Compute $\hat{\mu}_0^j = \pi_{\phi_0^j}(z_0^j)$ and $\hat{\mu}_1^j = \pi_{\phi_1^j}(z_1^j)$
30:     Estimate CATE $\hat{\tau} = \hat{\mu}_1^j - \hat{\mu}_0^j$.
31:     $C$.append($\hat{\tau}$)
32: end for
33: return $C$
```

Algorithm 9 SF Reptile T-learner

```
1: Let $\mu_0^i$ and $\mu_1^i$ be the outcome under treatment and control for experiment $i$.
2: Let $X^i$ be the data for experiment $i$. Let $X_t^i$ be the test data for experiment $i$.
3: Let $\pi_{\theta_0}$ and $\pi_{\theta_1}$ be a neural networks parameterized by $\theta_0$ and $\theta_1$.
4: Let $\theta = \theta_0 \cup \theta_1$.
5: Let $\epsilon = [\epsilon_0, \ldots, \epsilon_N]$ be a vector, where $N$ is the number of layers in $\pi_{\theta_i}$.
6: Let numouteriters be the total number of outer training iterations.
7: Let numinneriters be the total number of inner training iterations.
8: Let numexps be the number of experiments.
9: Let batchsize be the number of units sampled. We use 64.
10: **for** iouter < numouteriters **do**
11:     **for** i < numexps **do**
12:         $U_0(\theta_0) = \theta_0$
13:         $U_0(\theta_1) = \theta_1$.
14:         **for** k< numinneriters **do**
15:             Sample $X_0^i$ and $X_1^i$: control and treatment units. Sample batchsize units.
16:             $\mathcal{L}_0 = \|\pi_{U_k(\theta_0)}(X_0^i) - \mu_0(X_0^i)\|$
17:             $\mathcal{L}_1 = \|\pi_{U_k(\theta_1)}(X_1^i) - \mu_1(X_1^i)\|$
18:             $\mathcal{L} = \mathcal{L}_0 + \mathcal{L}_1$
19:             Compute $\nabla_\theta \mathcal{L} = \frac{\partial \mathcal{L}}{\partial \theta}$.
20:             Use ADAM with gradients given by $\nabla_\theta \mathcal{L}$ to obtain $U_{k+1}(\theta_0)$ and $U_{k+1}(\theta_1)$.
21:             Set $U_k(\theta_0) = U_{k+1}(\theta_0)$ and $U_k(\theta_1) = U_{k+1}(\theta_1)$
22:         **end for**
23:         **for** $p < N$ **do**
24:             $\theta_p = \epsilon_p \cdot U_k(\theta_p) + (1 - \epsilon_p) \cdot \theta_p$.
25:         **end for**
26:     **end for**
27: **end for**
28: To Evaluate CATE estimate, **do**
29: $C = []$.
30: **for** i < numexps **do**
31:     $U_0(\theta_0) = \theta_0$
32:     $U_0(\theta_1) = \theta_1$.
33:     **for** k< numinneriters **do**
34:         Sample $X_0^i$ and $X_1^i$: control and treatment units. Sample batchsize units.
35:         $\mathcal{L}_0 = \|\pi_{U_k(\theta_0)}(X_0^i) - \mu_0(X_0^i)\|$
36:         $\mathcal{L}_1 = \|\pi_{U_k(\theta_1)}(X_1^i) - \mu_1(X_1^i)\|$
37:         $\mathcal{L} = \mathcal{L}_0 + \mathcal{L}_1$
38:         Compute $\nabla_\theta \mathcal{L} = \frac{\partial \mathcal{L}}{\partial \theta}$.
39:         Use ADAM with gradients given by $\nabla_\theta \mathcal{L}$ to obtain $U_{k+1}(\theta_0)$ and $U_{k+1}(\theta_1)$.
40:         Set $U_k(\theta_0) = U_{k+1}(\theta_0)$ and $U_k(\theta_1) = U_{k+1}(\theta_1)$
41:     **end for**
42:     $\hat{\mu}_0^i = \pi_{U_k(\theta_0)}(X_0^i)$
43:     $\hat{\mu}_1^i = \pi_{U_k(\theta_1)}(X_1^i)$
44:     $\hat{\tau}^i = \hat{\mu}_1^i - \hat{\mu}_0^i$
45:     $C$.append($\hat{\tau}^i$).
46: **end for**
47: **return** $C$.
```

7.7 Discussion and Conclusion

In this chapter, we proposed the problem of transfer learning for CATE estimation. One immediate question the reader may be left with is why we chose the transfer learning techniques we did. We only considered two common types of transfer: (1) Basic fine tuning and weights sharing techniques common in the computer vision literature [137, 99, 17, 29, 59], (2) Techniques for learning an initialization that can be quickly optimized [33, 94, 83]. However, many further techniques exist. Yet, transfer learning is an extensively studied and perennial problem [103, 15, 129, 130, 127, 113]. In [135], the authors attempt to combine feature embeddings that can be utilized with non-parametric methods for transfer. [116] is an extension of this work that modifies the procedure for sampling examples from the support set during training. [3] and related techniques try to meta-learn an optimizer that can more quickly solve new tasks. [98] attempts to overcome forgetting during transfer by systematically introducing new network layers with lateral connections to old frozen layers. [78] uses networks with memory to adapt to new tasks. We invite the reader to review [33] for an excellent overview of the current transfer learning landscape. Though the majority of the discussed techniques could be extended to CATE estimation, our implementations of [98, 3] proved difficult to tune and consequently learned very little. Furthermore, we were not able to successfully adapt [116] to the problem of regression. We decided to instead focus our attention on algorithms for obtaining good initializations, which were easy to adapt to our problem and quickly delivered good results without extensive tuning.

8 Closing Remarks

Supervised learning has a curious property: it wants to work. Using a fairly standard set of algorithms and tools, an inexperienced researcher can solve a novel supervised learning problem. In my own experience, supervised learning worked even when I incorrectly coded my algorithms, did not correctly clean my data, and was bad at tuning my hyper-parameters. There were few surprises. For the most part, everything just worked even on novel problems.

Not surprisingly, many of the best results that utilize deep neural networks for robotics rely on supervised learning rather than the three learning problems we discussed in this dissertation [114]. When I coded Guided Policy Search [114], it worked beautifully and easily on difficult problems. Comparatively, the algorithms presented in this dissertation all required painstaking effort, bags of tricks, and thousands of hyper-parameter tuning sessions to get working at their full capacity. Even after all of this work, the results were often mixed. My contributions would improve performance on one task but hurt another. They could often solve one benchmark problem but would completely fail on a benchmark of seemingly similar difficulty. And even after all of my efforts, I'm still never quite sure why this is. After writing this thesis, I'm left with two questions. First, why are reinforcement learning, meta learning, and imitation learning so much more difficult than supervised learning? Second, is there anything we can do about it going forward?

As for my first question, I can postulate the following reasons why RL, imitation learning, and meta learning are much more difficult than supervised learning:

1. These learning problems are largely trying to learn over sequential data with complex causal relationships. Though supervised learning has shown that learning over sequential speech and text data is possible, analyzing this data does not require a deep understanding of the causal relationships between states.

2. On a related note, supervised learning never has to consider the credit assignment problem. It is very difficult for an RL agent to properly assign credit to which actions lead to an eventual reward. Of course, policy gradient methods are essentially doing credit assignment by re-

inforcing good actions. But, they may be doing this quite poorly. A human playing pong or tennis has a keen understanding that the key moment during their play is when their own racket strikes the ball, even though they do not receive credit for this strike until many time-steps later. Comparatively, our learning agents are trained to assign larger credit to times when their opponent misses the ball, even though they have long since lost control of the state space. As mentioned before, credit would be better assigned by sampling causal relationships in the agents environment, rather than directly sampling low-level states.

3. Policy gradient methods might be little better than random search. They are data inefficient, bad at exploration, and highly unstable. They do not compute the gradient with respect to the true reward, and are consequently orders of magnitude slower than model based approaches.

4. Our benchmarks for these learning problems are quite bad. Unfortunately, I spent around 33 percent of my time as a graduate student simply trying to find the right environments to test my new ideas. This problem is systematic and affected the majority of RL researchers I know. Comparatively, the supervised learning community seems to have very good benchmarks for their major problems. Moreover, the benchmarks we do have tend to be contrived. In supervised learning, the community tends to directly solve the problem they care about: classifying images, translating text, transcribing speech. In RL, we instead try and solve proxy problems such as Atari games. No one actually wants agents that are good at playing Atari. They have no value to us. What we probably want is something closer to androids that can perform most manual labor tasks. Our environments need to reflect this more.

5. We don't yet have enough compute. In particular, the experiments in Section 4 required over ten thousands hours of compute time. Since meta learning is a double optimization problem that optimizes over the entire learning process, this is not surprising. Such compute requirements are not unusual in RL research. In fact, they are commonplace. Given how inefficient policy gradient methods are, and how difficult the benchmark problems are, we probably need two orders of magnitude more compute to make real progress on meta learning.

6. Supervised learning is not a sampling problem. The underlying data

distribution is fixed and exogenous to the optimization process. In the learning problems we considered, the optimization algorithm itself influences the data that it will optimize over at future time-steps. This creates non-stationarity issues which in general make the problem much much more difficult. Optimizing over a dynamically changing set of data is difficult because the optimization landscape is constantly changing. Moreover, policy gradient methods essentially rely on only their ability to sample in order to compute gradients. This further complicates the relationship between the optimization process and sampling the state distribution. Though this thesis is titled *Learning as a Sampling Problem*, it would be remiss to suggest that such an interpretation is a good thing. Instead, I argue that learning as a sampling problem is one of the chief difficulties in the learning process.

This list might appear to make things quite hopeless. I often felt that way while completing this thesis. However, I do think there are interesting avenues to explore.

1. Better learning algorithms. Most of the work in this thesis was based on policy gradient methods or other model free reinforcement learning methods. While these methods are flexible and impressive, they are not poised for long term success. The world is too big and too complex for all learning to be model free. I would like to see more effort put into model-based RL and model based meta learning. In particular, I would like to expand the definition of models to include system other than dynamics.

2. More work on causal inference and credit assignment. A lot of past work in these areas has focused on meta learning or hierarchical learning, without much success. I would like to see work on RL that takes credit assignment and causal models seriously.

3. Better benchmarks. We should try and find problems that we actually care about solving. The solutions to our benchmarks should be valuable results, not stepping stones.

9 References

References

[1] P. Abbeel and A. Ng. Apprenticeship learning via inverse reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2004.

[2] Pieter Abbeel, Adam Coates, and Andrew Y Ng. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 2010.

[3] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, and Nando. de Freitas. Learning to learn by gradient descent by gradient descent. *Neural Information Processing Systems (NIPS)*, 2016.

[4] M. Araya, O. Buffet, and V. Thomas. Near-optimal brl using optimistic local transitions. *ICML*, 2012.

[5] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[6] Susan Athey and Guido W Imbens. Machine learning methods for estimating heterogeneous causal effects. *stat*, 1050(5), 2015.

[7] Susan Athey and Guido W Imbens. Recursive partitioning for heterogeneous causal effects. *Proceedings of the National Academy of Sciences of the United States of America*, 113(27):7353–60, 2016.

[8] Yusuf Aytar and Andrew Zisserman. Tabula rasa: Model transfer for object category detection. In *2011 International Conference on Computer Vision*, pages 2252–2259. IEEE, 2011.

[9] P. Bacon and D. Precup. The option-critic architecture. *In NIPS Deep RL Workshop*, 2015.

[10] D. Barber and F. V. Agakov. Kernelized infomax clustering. *NIPS*, 2005.

[11] Barto and Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 2003.

[12] M. Belkin, P. Niyogi, and V. Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *JMLR*, 2006.

[13] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. Unifying count based exploration and intrinsic motivation. *NIPS*, 2016.

[14] M.G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *JAIR*, 2013.

[15] Samy Bengio, Yoshua Bengio, Jocelyn Cloutier, and Jan Gecsei. On the optimization of a synaptic learning rule. *Biological Neural Networks*, 1992.

[16] A. Boularias, J. Kober, and J. Peters. Relative entropy inverse reinforcement learning. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

[17] Lubomir Bourdev, Subhransu Maji, and Jitendra Malik. Describing people: A poselet-based approach to attribute classification. *ICCV*, 2011.

[18] R. I. Brafman and M. Tennenholtz. R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 2002.

[19] J. S. Bridle, A. J. Heading, and D. J. MacKay. Unsupervised classifiers, mutual information and 'phantom targets'. *NIPS*, 1992.

[20] Jane Bromley, James W Bentz, Léon Bottou, Isabelle Guyon, Yann LeCun, Cliff Moore, Eduard Säckinger, and Roopak Shah. Signature verification using a “siamese” time delay neural network. *International Journal of Pattern Recognition and Artificial Intelligence*, 7(04):669–688, 1993.

[21] Sylvain Calinon. *Robot programming by demonstration.* EPFL Press, 2009.

[22] Sylvain Calinon, Florent Guenter, and Aude Billard. On learning, representing, and generalizing a task in a humanoid robot. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(2):286–298, 2007.

[23] D. Carmel and S. Markovitch. Exploration strategies for model-based learning in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 1999.

[24] Malinda Carpenter, Josep Call, and Michael Tomasello. Understanding "prior intentions" enables two–year–olds to imitatively learn a complex task. *Child development*, 73(5):1431–1441, 2002.

[25] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *NIPS*, 2016.

[26] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 539–546. IEEE, 2005.

[27] Alexander D'Amour, Peng Ding, Avi Feller, Lihua Lei, and Jasjeet Sekhon. Overlap in observational studies with high-dimensional covariates. *arXiv preprint arXiv:1711.02582*, 2017.

[28] A. Doerr, N. Ratliff, J. Bohg, M. Toussaint, and S. Schaal. Direct loss minimization inverse optimal control. In *Proceedings of Robotics: Science and Systems (R:SS)*, Rome, Italy, July 2015.

[29] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. *International Conference on Machine Learning (ICML)*, 2014.

[30] F. Doshi-Velez, D. Wingate, N. Roy, and J. Tenenbaum. Nonparametric bayesian policy priors for reinforcement learning. *NIPS*, 2014.

[31] Lixin Duan, Dong Xu, and Ivor Tsang. Learning with augmented features for heterogeneous domain adaptation. *arXiv preprint arXiv:1206.4660*, 2012.

[32] Y. Duan, M. Andrychowicz, B. Stadie, J. Ho, J. Schneider, I. Sutskever, P. Abbeel, and W. Zaremba. One-shot imitation learning. *NIPS*, 2017.

[33] C. Finn, P. Abbeel, and S. Levine. Model-agnostic metalearning for fast adaptation of deep networks. *ICML*, 2017.

[34] C. Finn, S. Levine, and P. Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. *ICML*, 2016.

[35] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Insights and applications. *Deep Learning Workshop, ICML*, 2015.

[36] Y. Ganin and V. Lempitsky. Unsupervised domain adaptation by backpropagation. *Arxiv preprint 1409.7495*, 2014.

[37] M. Geist and O. Pietquin. Managing uncertainty within value function approximation in reinforcement learning. *W. on Active Learning and Experimental Design*, 2010.

[38] Alan S Gerber, Gregory A Huber, Albert H Fang, and Andrew Gooch. The generalizability of social pressure effects on turnout across high-salience electoral contexts: Field experimental evidence from 1.96 million citizens in 17 states. *American Politics Research*, 45(4):533–559, 2017.

[39] G Gioioso, G Salvietti, M Malvezzi, and D Prattichizzo. An object-based approach to map human hand synergies onto robotic hands with dissimilar kinematics. *Robotics: Science and Systems VIII*, page 97, 2013.

[40] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

[41] Vincent Graziano, Tobias Glasmachers, Tom Schaul, Leo Pape, Giuseppe Cuccu, Jurgen Leitner, and Jürgen Schmidhuber. Artificial

Curiosity for Autonomous Space Exploration. *Acta Futura*, 1(1):41–51, 2011.

[42] Donald P Green and Holger L Kern. Modeling heterogeneous treatment effects in survey experiments with bayesian additive regression trees. *Public opinion quarterly*, 76(3):491–511, 2012.

[43] A. Guez, D. Silver, and P. Dayan. Efficient bayes-adaptive reinforcement learning using sample-based search. *NIPS*, 2014.

[44] X. Guo, S. Singh, H. Lee, R. Lewis, and X. Wang. Deep learning for real-time atari game play using offline monte-carlo tree search planning. *NIPS*, 2014.

[45] Abhishek Gupta, Clemens Eppner, Sergey Levine, and Pieter Abbeel. Learning dexterous manipulation for a soft robotic hand from human demonstration. *arXiv preprint arXiv:1603.06348*, 2016.

[46] T. Hastie, R. Tibshirani, and J. Friedman. The elements of statistical learning. *Springer series in statistics*, 2001.

[47] Nicholas C. Henderson, Thomas A. Louis, Chenguang Wang, and Ravi Varadhan. Bayesian analysis of heterogeneous treatment effects for patient-centered outcomes research. *Health Services and Outcomes Research Methodology*, 16(4):213–233, Dec 2016.

[48] Jennifer L Hill. Bayesian nonparametric modeling for causal inference. *Journal of Computational and Graphical Statistics*, 20(1):217–240, 2011.

[49] G. E. Hinton and R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 2006.

[50] J. Ho and S. Ermon. Generative adversarial imitation learning. *arXiv pre-print: 1606.03476*, pages 1061–1068, 2016.

[51] Judy Hoffman, Erik Rodner, Jeff Donahue, Trevor Darrell, and Kate Saenko. Efficient learning of domain-invariant image representations. *arXiv preprint arXiv:1301.3224*, 2013.

[52] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel. Vime: Variational information maximizing exploration. *NIPS*, 2016.

[53] S. Kakade, M. Kearns, and J. Langford. Exploration in metric state spaces. *ICML*, 2003.

[54] M. Kalakrishnan, P. Pastor, L. Righetti, and S. Schaal. Learning objective functions for manipulation. In *International Conference on Robotics and Automation (ICRA)*, 2013.

[55] M. Kearns and D. Koller. Efficient reinforcement learning in factored mdps. *Proc. IJCAI*, 1999.

[56] M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning Journal*, 2002.

[57] M. J. Kearns and S. P. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 2002.

[58] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2014.

[59] Gregory. Koch. Siamese neural networks for one-shot image recognition. *ICML Deep Learning Workshop*, 2015.

[60] J. Z. Kolter and A. Y. Ng. Near-bayesian exploration in polynomial time. *ICML*, 2009.

[61] Kompella, Stollenga, Luciw, and Schmidhuber. Exploring the predictable. *Advances in Evolutionary Computing*, 2002.

[62] Jan Koutník, Giuseppe Cuccu, Jürgen Schmidhuber, and Faustino Gomez. Evolving large-scale neural networks for vision-based reinforcement learning. *GECCO 2013*, 15:1061–1068, 2013.

[63] A. Krause, P. Perona, and R. G. Gomes. Discriminative clustering by regularized information maximization. *NIPS*, 2010.

[64] Brian Kulis, Kate Saenko, and Trevor Darrell. What you saw is not what you get: Domain adaptation using asymmetric kernel transforms. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1785–1792. IEEE, 2011.

[65] Sören Künzel, Jasjeet Sekhon, Peter Bickel, and Bin Yu. Meta-learners for estimating heterogeneous treatment effects using machine learning. *arXiv preprint arXiv:1706.03461*, 2017.

[66] T. Lang, M. Toussaint, and K. Keristing. Exploration in relational domains for model-based reinforcement learning. *Proc. AAMAS*, 2014.

[67] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[68] S. Levine, Z. Popovic, and V. Koltun. Nonlinear inverse reinforcement learning with gaussian processes. In *Advances in Neural Information Processing Systems (NIPS)*, 2011.

[69] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

[70] T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv:1509.02971*, 2015.

[71] Mingsheng Long and Jianmin Wang. Learning transferable features with deep adaptation networks. *CoRR, abs/1502.02791*, 1:2, 2015.

[72] M. Lopes, T. Lang, M. Toussaint, and P.-Y. Oudeyer. Exploration in model-based reinforcement learning by empirically estimating learning progress. *NIPS*, 2012.

[73] Christopher B Mann. Is there backlash to social pressure? a large-scale field experiment on voter mobilization. *Political Behavior*, 32(3):387–407, 2010.

[74] Yishay Mansour, Mehryar Mohri, and Afshin Rostamizadeh. Domain adaptation: Learning bounds and algorithms. *arXiv preprint arXiv:0902.3430*, 2009.

[75] Melissa R Michelson. The risk of over-reliance on the institutional review board: An approved project is not always an ethical project. *PS: Political Science & Politics*, 49(02):299–303, 2016.

[76] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy P Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *arXiv:1602.01783*, 2016.

[77] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[78] Tsendsuren Munkhdalai and Hong. Yu. Meta networks. *ICML*, 2017.

[79] Chrystopher L Nehaniv. Nine billion correspondence problems. *Imitation and Social Learning in Robots, Humans and Animals: Behavioural, Social and Communicative Dimensions, Cambridge University Press*, 8:10, 2007.

[80] Chrystopher L Nehaniv and Kerstin Dautenhahn. Like me?-measures of correspondence and imitation. *Cybernetics & Systems*, 32(1-2):11–51, 2001.

[81] A. Ng, S. Russell, et al. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2000.

[82] Hung Ngo, Matthew Luciw, Alexander Forster, and Juergen Schmidhuber. Learning skills from play: Artificial curiosity on a Katana robot arm. *Proceedings of the International Joint Conference on Neural Networks*, pages 10–15, 2012.

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

[84] Xinkun Nie and Stefan Wager. Learning objectives for treatment effect estimation. *arXiv preprint arXiv:1712.04912*, 2017.

[85] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy. Deep exploration via bootstrapped dqn. *NIPS*, 2016.

[86] G. Ostrovski, M. G. Bellemare, A. v. d. Oord, and R. Munos. Count-based exploration with neural density models. *arXiv:1703.01310*, 2017.

[87] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. *ICML*, 2017.

[88] J. Pazis and R. Parr. Pac optimal exploration in continuous space markov decision processes. *Proc. AAAI*, 2013.

[89] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems*, pages 305–313, 1989.

[90] Scott Powers, Junyang Qian, Kenneth Jung, Alejandro Schuler, Nigam H Shah, Trevor Hastie, and Robert Tibshirani. Some methods for heterogeneous treatment effect estimation in high dimensions. *Statistics in medicine*, 2018.

[91] D. Ramachandran and E. Amir. Bayesian inverse reinforcement learning. In *AAAI Conference on Artificial Intelligence*, volume 51, 2007.

[92] N. Ratliff, J. A. Bagnell, and M. A. Zinkevich. Maximum margin planning. In *International Conference on Machine Learning (ICML)*, 2006.

[93] N. Ratliff, D. Bradley, J. A. Bagnell, and J. Chestnutt. Boosting structured prediction for imitation learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2007.

[94] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. *International Conference on Learning Representations (ICLR)*, 2017.

[95] Paul R Rosenbaum and Donald B Rubin. The central role of the propensity score in observational studies for causal effects. *Biometrika*, 70(1):41–55, 1983.

[96] Stéphane Ross, Geoffrey J Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, page 6, 2011.

[97] Donald B Rubin. Estimating causal effects of treatments in randomized and nonrandomized studies. *Journal of educational Psychology*, 66(5):688, 1974.

[98] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. *CoRR, vol. abs/1606.04671*, 2016.

[99] Kulis B. Fritz M. Saenko, K. and T. Darrell. Adapting visual category models to new domains. *ECCV*, 2010.

[100] Stefan Schaal. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 3(6):233–242, 1999.

[101] Schmidhuber. Evolutionary principles in self-referential learning, or on learning how to learn: The meta-meta-... hook. *Diploma thesis, TUM*, 1987.

[102] Schmidhuber. Gödel machines: Fully self-referential optimal universal self-improvers. *Artificial General Intelligence*, 2006.

[103] J. Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 1992.

[104] J. Schmidhuber. Continual curiosity-driven skill acquisition from high-dimensional video inputs for humanoid robots. *Artificial Intelligence*, 2015.

[105] J. Schmidhuber, J. Zhao, and N. Schraudolph. Reinforcement learning with self-modifying policies. *Learning to learn, Kluwer*, 1997.

[106] Juergen Schmidhuber. Developmental robotics, optimal artificial curiosity, creativity, music, and the fine arts. *Connection Science*, 2006.

[107] Juergen Schmidhuber. On Learning to Think: Algorithmic Information Theory for Novel Combinations of Reinforcement Learning Controllers and Recurrent Neural World Models. *arXiv*, pages 1–36, 2015.

[108] Jürgen Schmidhuber. A Possibility for Implementing Curiosity and Boredom in Model-Building Neural Controllers. *Meyer, J.A. and Wilson, S.W. (eds) : From Animals to animats*, pages 222–227, 1991.

[109] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017.

[110] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *Arxiv preprint 1502.05477*, 2015.

[111] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[112] Aaron Shon, Keith Grochow, Aaron Hertzmann, and Rajesh P Rao. Learning shared latent structure for image synthesis and robotic imitation. In *Advances in Neural Information Processing Systems*, pages 1233–1240, 2005.

[113] Silver, Yand, and Li. Lifelong machine learning systems: Beyond learning algorithms. *DAAAI Spring Symposium-Technical Report, 2013.*, 2013.

[114] Finn Chelsea Darrell Trevor SLevine, Sergey and Pieter Abbeel. End-to-end training of deep visuomotor policies. *JMLR*, 2016.

[115] W. D. Smart and L. P. Kaelbling. Practical reinforcement learning in continuous spaces. *ICML*, 2000.

[116] J. Snell, K. , Swersky, and R. Zemel. Prototypical networks for few-shot learning. *arXiv preprint arXiv:1703.05175*, 2017.

[117] J. Sorg, S. Singh, and R. L. Lewis. Variance-based rewards for approximate bayesian reinforcement learning. *Proc. UAI*, 2010.

[118] Bradly C. Stadie, Pieter Abbeel, and Ilya Sutskever. Third-person imitation learning. *ICLR*, 2017.

[119] Bradly C. Stadie, S. Levine, and P. Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv:1507.00814*, 2015.

[120] Bradly C. Stadie, Sören R. Künzel, Nikita Vemuri, Varsha Ramakrishnan, Jasjeet S. Sekhon, and Pieter Abbeel. Transfer learning for

estimating causal effects using neural networks. *Pre-print. submitted to NIPS*, 2018.

[121] Bradly C. Stadie, Ge Yang, Rein Houthooft, Xi Chen, Yan Duan, Wu Yuhuai, Pieter Abbeel, and Ilya Sutskever. Some considerations on learning to explore via meta-reinforcement learning. *arXiv: 1803.01118*, 2018.

[122] Jan Storck, Sepp Hochreiter, and Jürgen Schmidhuber. Reinforcement driven information acquisition in non-deterministic environments. *Proceedings of the International ...*, 2:159–164, 1995.

[123] A. L. Strehl and M. L. Littman. An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 2008.

[124] Yi Sun, Faustino Gomez, and Jürgen Schmidhuber. Planning to be surprised: Optimal Bayesian exploration in dynamic environments. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6830 LNAI:41–51, 2011.

[125] Matt Taddy, Matt Gardner, Liyun Chen, and David Draper. A nonparametric bayesian analysis of heterogenous treatment effects in digital experimentation. *Journal of Business & Economic Statistics*, 34(4):661–672, 2016.

[126] H. Tang, R. Houthooft, D. Foote, A. Stooke, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel. exploration: A study of count-based exploration for deep reinforcement learning. *arXiv:1611.04717*, 2016.

[127] Taylor and Stone. Transfer learning for reinforcement learning domains: A survey. *DAAAI Spring Symposium-Technical Report, 2013.*, 2009.

[128] S. Tessler, C. Givony, T. Zahavy, D.J. Mankowitz, and S. Mannor. A deep hierarchical approach to lifelong learning in minecraft. *arXiv:1604.07255*, 2016.

[129] Thrun. Is learning the n-th thing any easier than learning the first? *NIPS*, 1996.

[130] S. Thrun and L. Pratt. Learning to learn. *Springer Science and Business Media*, 1998.

[131] Lu Tian, Ash A Alizadeh, Andrew J Gentles, and Robert Tibshirani. A simple method for estimating interactions between a treatment and a large number of covariates. *Journal of the American Statistical Association*, 109(508):1517–1532, 2014.

[132] Eric Tzeng, Coline Devin, Judy Hoffman, Chelsea Finn, Xingchao Peng, Pieter Abbeel, Sergey Levine, Kate Saenko, and Trevor Darrell. Towards adapting deep visuomotor representations from simulated to real environments. *arXiv preprint arXiv:1511.07111*, 2015.

[133] Eric Tzeng, Judy Hoffman, Ning Zhang, Kate Saenko, and Trevor Darrell. Deep domain confusion: Maximizing for domain invariance. *arXiv preprint arXiv:1412.3474*, 2014.

[134] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161*, 2017.

[135] Oriol Vinyals, Charles Blundell, Tim Lillicrap, Daan Wierstra, and et al. Matching networks for one shot learning. *Neural Information Processing Systems (NIPS)*, 2016.

[136] Stefan Wager and Susan Athey. Estimation and inference of heterogeneous treatment effects using random forests. *Journal of the American Statistical Association*, 2017.

[137] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-ucsd birds 200. technical report cns-tr-2010-001. *California Institute of Technology*, 2010.

[138] Wiering and Schmidhuber. Hq-learning. *Adaptive Behavior*, 1997.

[139] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning 8, 3-4 (1992), 229–256*, 1992.

[140] M. Wulfmeier, P. Ondruska, and I. Posner. Maximum entropy deep inverse reinforcement learning. *arXiv preprint arXiv:1507.04888*, 2015.

[141] Jun Yang, Rong Yan, and Alexander G Hauptmann. Cross-domain video concept detection using adaptive svms. In *Proceedings of the 15th ACM international conference on Multimedia*, pages 188–197. ACM, 2007.

[142] B. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey. Maximum entropy inverse reinforcement learning. In *AAAI Conference on Artificial Intelligence*, 2008.