Title
On and Off-Policy Deep Imitation Learning for Robotics

Author
Laskey, Michael David

Publication Date
2018

Peer reviewed|Thesis/dissertation

On and Off-Policy Deep Imitation Learning for Robotics

by Michael Laskey

A dissertation submitted in partial satisfaction of the requirements for the degree of:

Doctor of Philosophy in
Engineering – Electrical Engineering and Computer Sciences
in the
Graduate Division of the
University of California at Berkeley

Committee in charge:
Prof. Kenneth Goldberg, Chair
Prof. Anca Dragan
Prof. Paul Grigas

Summer 2018

Abstract

On and Off-Policy
Deep Imitation Learning for Robotics

by Michael Laskey
Doctor of Philosophy in Engineering – Electrical Engineering and Computer Sciences
University of California, Berkeley
Ken Goldberg, Chair

As an alternative to explicit programming for robots, Deep Imitation learning has two drawbacks: sample complexity and covariate shift. One approach to Imitation Learning is Behavior Cloning, in which a robot observes a supervisor and then infers a control policy. A known problem with this approach is that even slight departures from the supervisor's demonstrations can compound over the policy's roll-out resulting in errors; this concept of drift and resulting error is commonly referred to as covariate shift On-policy techniques reduce covariate shift by iteratively collecting corrective actions for the current robot policy. To reduce sample complexity of these approaches, we propose a novel active learning algorithm, SHIV (Svm-based reduction in Human InterVention). While evaluating SHIV, we reconsider the trade-off between Off- and On-Policy methods and find that: 1) On-Policy methods are challenging for humans supervisors and 2) performance varies with the expressiveness of the policy class. To make Off-Policy methods more robust for expressive policies we propose a second algorithm, DART (Disturbances Augmenting Robot Trajectories), which injects optimized noise into the supervisor's control stream to simulate error during data collection. This dissertation contributes two aforementioned algorithms, experimental evaluation with three robots evaluating their performance on tasks ranging from grasping in clutter to singulation to bed-making, and the design of a novel first-order urban driving simulator (FLUIDS) that can fill gaps in existing benchmarks for Imitation Learning to rapidly test algorithm performance in terms of generalization.

Dedication

I have been fortunate to have a significant number of excellent mentors in my Ph.D. I want to thank Anca Dragan, Florian Pokorny, Sachin Patil and Kevin Jamieson for all their guidance. I am especially thankful to my advisor Dr. Ken Goldberg. I've learned many valuable life lessons from him and his countless hours of feedback were influential on this work.

I also want to thank my colleagues; Jeff Mahler, Sanjay Krishnan, Steve McKinley, David Gealy, Zoey McCarthy, Animesh Garg, Lauren Miller, and Sam Staszak. My time in Soda and Etcheverry Hall was filled with endless debate and creative energy. We had a lot of fun the past five years ranging from nights spent in haunted houses to adventures in Bangkok. I will never forget the lab bonding experiences while camping in the desert.

I had the pleasure to mentor a large number of people in my Ph.D., and it was by far one of the most fulfilling experiences. I want to thank Jonathan Lee, Caleb Chuck, Wesley Hsieh, Chris Powers, Nami Sadigh, Ruta Joshi, Jerry Zhao, Andy Cui, Jim Ren, Adithya Murali and David Wang for all their effort. A large part of this work is due to their contributions.

Lastly, I want to thank my parents: Marsha and Scot Laskey, my sister Katelyn Laskey and my fiance Anne Browne for all their support and encouragement.

Contents

Chapter 1

Introduction

Rather than pure unsupervised learning with data-inefficient techniques like Reinforcement Learning (RL), in Imitation Learning (IL) the robot learns the policy directly from supervisor examples, mapping states to controls (Argall, Chernova, Veloso, & Browning, 2009). IL is useful when the robot does not have access to either the cost function that it should optimize or the dynamics model of the environment. The former occurs when its difficult to specify how to trade-off task attributes of equal importance, like a car trying to drive on the road while also avoiding other vehicles (Abbeel & Ng, 2004). The second situation occurs when either the system or the interaction with the world is difficult to characterize, like when a robot is trying to grasp an object in clutter and does not have an accurate model of how each object will behave in the space (Laskey, Lee, et al., 2016a)

IL algorithms can be categorized as Off-Policy or On-Policy. In Off-Policy IL, the robot learns the policy based on a batch of examples provided by a supervisor and then executes it to achieve the task. During execution though, a small error can accumulate, leading the robot away from the region of state space where it was given examples, leading to unrecoverable failures. For example, a robot driving may be trained on examples driving safely down the center of a lane, but even slight deviations will eventually put the robot into states near the side of the road where its policy could fail (Pomerleau, 1989). Commonly, this distribution mismatch between training and execution is known as covariate shift.

On-Policy IL addresses this issue by iteratively gathering more examples from the supervisor in states the robot encounters (Grollman & Jenkins, n.d.; Ross & Bagnell, 2010; Ross, Gordon, & Bagnell, 2011a). One such algorithm, DAgger, learns a series of policies. At each iteration, the robot trains a policy based on the existing examples, then rolls out (executes) that policy and the supervisor provides demonstrations for all states the robot visits. The new state/control examples are aggregated with the old examples for the next iteration. DAgger and related algorithms have been applied in a wide range of applications, from quadrotor flight to

natural language to Atari games (Guo, Singh, Lee, Lewis, & Wang, 2014; Duvallet, Kollar, & Stentz, 2013; Ross et al., n.d.).

A drawback to On-Policy approaches is the need to provide corrective feedback for each state the robot visits, which can be quite tedious. Thus, it is imperative to explore ways to reduce sample complexity without having to rely solely on the supervisor to collect examples. One standard technique to reduce data collection by the supervisor is active learning (Settles & Craven, 2008), in which the robot decides which sample should receive a label.

While active learning is an effective way to reduce data, the performance of these techniques can change when combined with On-Policy methods. Unlike traditional supervised learning, on-policy sampling creates a non-stationary distribution over which the data is sampled from, which can confuse conventional active learning schemes, such as those based on a confidence score or a query by committee. To alleviate this, we contribute an algorithm, SHIV (Svm-based reduction in Human InterVention), which conservatively proposes to estimate the support of the training data and ask for labels when new data is sampled outside the support.

When deploying and testing SHIV, we found that On-Policy methods can be quite challenging for human supervisors. In many On-Policy methods, a human retroactively must provide corrective feedback to the robot without observing the effect of their proposed control. Off-Policy methods, on the other hand, do not exhibit this limitation because they require the supervisor only to provide demonstrations. We observe in practice, via evaluation with human supervisors teaching a Zymark robot to perform planar simulation strategies, that On-Policy methods can significantly degrade performance.

In light of this limitation, we study theoretically when Off-Policy methods can be preferable to On- Policy. Intuitively, On-Policy are preferable when the policy is not expressive enough to capture the supervisor. In this setting, the robot will converge to a state distribution potentially far away from the supervisor and will need to collect examples in these new states. However, as the policy becomes more expressive, with the use of techniques like deep-learning, it is possible to converge quite close to the supervisor's distribution with enough data.

We can formalize the relation of model expressiveness between Off and On-Policy as a prediction problem. In this formulation, the goal of choosing a sampling distribution is to increase the likelihood of sampling states under the robot's learned policy. Thus it can be shown, in the infinite data setting, the choice between Off and On-Policy is directly related to the expressiveness of the estimator, or the size of the policy class. Specifically, as the expressiveness increases Off-Policy methods have a higher likelihood of converging near the supervisor.

However, Off-Policy methods with limited data may still not have an objectively high likelihood of sampling states under the robot's final policy. Therefore, we show that by injecting artificial noise into the supervisor's control stream, it is possible to increase the likelihood via simulating errors occurring during data collection. We

further show the optimal amount of noise to be injected is related to the variance of the estimator, or how much it is expected to change in the random design setting (Hsu, Kakade, & Zhang, 2012). We use these results to create an approximate algorithm, known as DART (Disturbances Augmenting Robot Training), which automatically adjusts the level of noise to inject in the control stream.

We evaluate DART both experimentally and in a study with human supervisors. In our study, human supervisors' train a Toyota HSR robot to retrieve an object from a cluttered shelf. When optimized noise is injected using DART, we find that the robustness of the learned policy is increased from a policy trained with no noise (i.e. Behavior Cloning). This study suggests humans can tolerate noise in teleoperation systems and properly calibrated noise-injection can increase robustness.

Finally, we conclude with a discussion on benchmarking of current Imitation Learning algorithms and the limitations of directly applying RL benchmarks, such as MuJoCo. Typically, RL benchmarks (Todorov, Erez, & Tassa, 2012) generally test for mastery of a single instance of a task. They do not benchmark how well the agent generalizes to new settings, which is the goal of IL. Furthermore, current benchmarks do not come with a fixed supervisor, which leads to the use of learned agents trained with RL. These learned supervisors may be quite unstable in some regions of the workspace, which can create unfair comparisons between algorithms.

To help encourage better benchmarking, we offer a driving simulator, known as FLUIDS (First-Order Local Urban Intersection Driving Simulator), which models the complex and variable interaction at traffic intersections. FLUIDS is designed to have a highly variable initial state distribution, which allows for complex sensitivity analyses of how well a learned agent does when tested on new domains. Furthermore, FLUIDS exposes the built-in algorithmic planner as a supervisor, to provide an interpretable baseline and way to collect training data.

In conclusion, the choice between whether using an On-Policy or Off-Policy sampling to collect data affects major design choices in robot learning systems. To train robots efficiently, we need to provide robust Off-Policy algorithms and present ways to reduce the data needed for On-Policy algorithms. This dissertation contributes: two algorithms;, experimental evaluation with three robots evaluating their performance on tasks ranging from grasping in clutter to singulation to bed-making; and, the design of a novel first-order urban driving simulator (FLUIDS) that can fill gaps in existing benchmarks for Imitation Learning to rapidly test algorithm performance in terms of generalization.

Problem Statement

The objective of Imitation Learning is to learn a policy that matches the supervisor's policy.

Modeling Choices and Assumptions: We model the system dynamics as Marko-

vian and stochastic. We model the initial state as sampled from a distribution over the state space. We assume a known state space and set of actions. We also assume access to a robot, such that we can sample from the state sequences induced by a policy. Lastly, we assume access to a supervisor who can provide a demonstration of the task.

Policies and State Densities. We denote by $\mathcal{X}$ the set consisting of observable states for a robot, and by $\mathcal{U}$ the set of actions. We model dynamics as Markovian, such that the probability of visiting state $\mathbf{x}_{t+1} \in \mathcal{X}$ can be determined from the previous state $\mathbf{x}_t \in \mathcal{X}$ and action $\mathbf{u}_t \in \mathcal{U}$:

$$p(\mathbf{x}_{t+1}|\mathbf{x}_0, \mathbf{u}_0, \ldots \mathbf{x}_t, \mathbf{u}_t) = p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t).$$

We assume a probability density over initial states $p(\mathbf{x}_0)$. An environment is thus defined as a specific instance of action and state spaces, initial state distribution, and dynamics.

Given a time horizon $T \in \mathbb{N}$, a trajectory ξ is a finite sequence of T pairs of states visited and corresponding control inputs at these states, $\xi = (\mathbf{x}_0, \mathbf{u}_0, \mathbf{x}_1, \mathbf{u}_1, \ldots, \mathbf{x}_T)$, where $\mathbf{x}_t \in \mathcal{X}$ and $\mathbf{u}_t \in \mathcal{U}$ for each $t \in \mathbb{N}$.

A deterministic policy is a measurable function $\pi : \mathcal{X} \to \mathcal{U}$ from states to controls. We consider policies $\pi_\theta : \mathcal{X} \to \mathcal{U}$ parameterized by some $\theta \in \Theta$. Under our assumptions, any such policy π_θ induces a probability density over the set of trajectories of length T:

$$p(\xi|\theta) = p(\mathbf{x}_0) \prod_{t=0}^{T-1} p(\mathbf{x}_{t+1}|\pi_\theta(\mathbf{x}_t), \mathbf{x}_t)$$

While we do not assume knowledge of the distributions corresponding to $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ or $p(\mathbf{x}_0)$, we assume that we have a real robot or a simulator. Therefore, when 'rolling out' trajectories under a policy π_θ, we utilize the robot or the simulator to sample from the resulting distribution over trajectories rather than estimating $p(\xi|\pi_\theta)$ itself.

Objective. In Imitation Learning, we do not assume access to a reward function, like we do in Reinforcement Learning (Sutton & Barto, 1998), but instead a supervisor, $\pi_{\theta*}$, where θ^* may not be contained in Θ. We assume the supervisor achieves a desired level of performance on the task, although it may not be optimal.

We measure the difference between controls using a surrogate loss $l : \mathcal{U} \times \mathcal{U} \to \mathbb{R}$ (Ross et al., 2011a; Ross & Bagnell, 2010). A commonly considered surrogate loss is the squared L2-norm on the control vectors $l(\mathbf{u}_1, \mathbf{u}_2) = ||\mathbf{u}_1 - \mathbf{u}_2||_2^2$ (Laskey, Lee, et al., 2016b). We measure total loss along a trajectory with respect to two policies π_{θ_1} and π_{θ_2} by $J(\theta_1, \theta_2|\xi) = \sum_{t=0}^{T-1} l(\pi_{\theta_1}(\mathbf{x}_t), \pi_{\theta_2}(\mathbf{x}_t))$.

The objective of Imitation Learning is to minimize the expected surrogate loss along the distribution induced by the robot's policy:

$$\min_\theta E_{p(\xi|\pi_\theta)} J(\theta, \theta^*|\xi). \tag{1}$$

The Two Types of Errors in Imitation Learning

In Eq. 1, the distribution on trajectories and the cumulative surrogate loss are coupled, which makes this a challenging optimization problem. Instead of directly solving this objective, a common thing to do is select a sampling distribution from which the data is collected from. In general, the sampling distribution comes from either being concentrated around the supervisor's policy or the robots.

We can describe a sampling distribution as the following Markov chain:

$$p(\xi|\psi) = p(\mathbf{x}_0) \prod_{t=0}^{T-1} p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) p(\mathbf{u}_t|\mathbf{x}_t, \psi)$$

In this distribution, the initial state distribution $p(\mathbf{x}_0)$ and $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ are fixed because they are parameters of the environments. However the probability over the control $p(\mathbf{u}_t|\mathbf{x}_t, \psi)$ is now paramterized by the sufficient statistic $\psi \in \Psi$. For example, one choice of a sampling distribution may be noise injected into the supervisor's policy, i.e. $\mathbf{u} = \pi_{\theta^*}(\mathbf{x}) + \epsilon$, where $\epsilon \sim \mathcal{N}(\mu, \Sigma)$. In this case, ψ corresponds to the following $\psi = \{\theta^*, \mu, \Sigma\}$. In Chapter 2, we will review how different algorithms such as Behavior Cloning, DAgger and DART can be seen as parameter settings of ψ.

Once a sampling distribution is defined a fix number, or N, trajectories are sampled from it and the expected risk is minimized. This can be formulated as follows:

$$\theta^R = \underset{\theta}{\operatorname{argmin}} \sum_{n=0}^{N-1} J(\theta, \theta^*|\xi_n), \;\; \xi_n \sim p(\xi|\psi)$$

where θ^R corresponds to the parameter vector describing the robot's learned policy π_{θ^R}. To understand the performance of π_{θ^R}, we can introduce the following decomposition.

$$\begin{aligned} &E_{p(\xi|\theta^R)} J(\theta^R, \theta^*|\xi) \\ &= \underbrace{E_{p(\xi|\theta^R)} J(\theta^R, \theta^*|\xi) - E_{p(\xi|\psi)} J(\theta^R, \theta^*|\xi)}_{\text{Covariate Shift}} + \underbrace{E_{p(\xi|\psi)} J(\theta^R, \theta^*|\xi)}_{\text{Generalization}}], \end{aligned}$$

This decomposition is interesting because it shows two types of errors. The second is the Generalization error, which corresponds to the standard type of error in Machine Learning (Schölkopf & Smola, 2002), where the distribution and loss are decoupled and the data is collected in an i.i.d setting. In Part 2, we discuss several novel algorithms to reduce this type of error.

The first type of error is more unique to IL and is commonly referred to as covariate shift (Osa et al., 2018), which means a distribution mismatch. Covariate shift reflects how close the sampling distribution, $p(\xi|\psi)$, is to the learned robot's

distribution,$p(\xi|\theta^R)$. A large number of IL algorithms have been proposed to reduce this type of error (Ross et al., n.d.; Ho & Ermon, 2016; Laskey, Lee, Fox, Dragan, & Goldberg, 2017). In Part 1, we will review this work and discuss when to expect certain algorithms to be preferable over others.

Chapter 2

Off and On-Policy Algorithms

The choice of sampling distribution can be divided into two categories On and Off Policy (Osa et al., 2018). In Off-Policy sampling the sampling distribution is concentrated around the supervisor's distribution, or $\forall \mathbf{x} \in \mathcal{X}$, $E_{p(\mathbf{u}|\mathbf{x},\psi)}\mathbf{u} = \pi_{\theta^*}(\mathbf{x})$. In On-Policy sampling the distribution is concentrated around the current robot's distribution, or $\forall \mathbf{x} \in \mathcal{X}$, $E_{p(\mathbf{u}|\mathbf{x},\psi)}\mathbf{u} = \pi_{\theta^k}(\mathbf{x})$. Note π_{θ^k} corresponds to the robot's policy at the current iteration.

In practice this creates two modes of feedback for the supervisor to provide; corrective or demonstrative. Demonstrative is typically performed via tele-operation, while corrective feedback is provided through some sort of labeling interface. In Fig. 2.1, we show an example of Off and On-Policy data collection systems for teaching manipulation strategies to a planar robot. While, both approaches teach the robot to perform the task, it is clear that the implementation of them is very different. We will will now review the types of algorithms prescribed to each method. Note that regardless of the sampling distribution, the supervisor is always expected to apply the true control for each state visited.

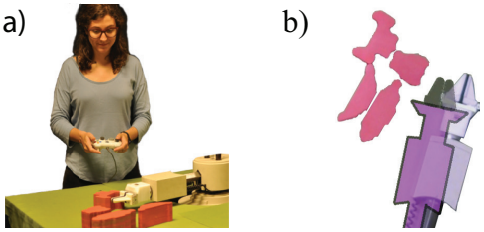


Figure 2.1: Two ways to provide feedback to the robot to teach it part singulation strategies. a) In Off Policy sampling, the human teleoperates the robot and performs the desired task. b) In On Policy sampling, the human observes a video of the robot's policy executing and applies retroactive feedback detailing what the robot should have done. In the image shown, the person is telling the robot to go backward towards the cluster.

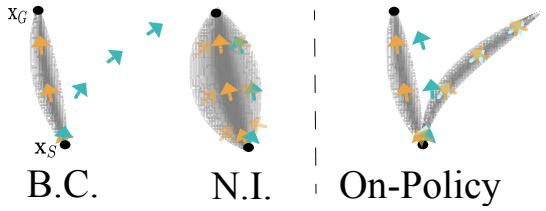


Figure 2.2: Robot Learning to reach a goal state $\mathbf{x}_G$. The grey denotes the distribution over trajectories. Left: Behavior Cloning (B.C.) in which the supervisor, the orange arrows, provides demonstrations. The robot, the teal arrows, deviates from the distributions and incurs high error. Middle: Noise Injection (N.I.), which injects noise to widen the supervisor's distribution and provides corrective examples. Noise Injection is off-policy but robust. Left: On-Policy which samples from the current robot's policy, the light teal arrows, to receive corrective examples from the supervisor.

Off-Policy Sampling

Off-Policy algorithms operate by having the supervisor tele-operate the robot and collect demonstrations. They are typically batch-style algorithms where N demonstrations are collected and the final robot's policy is trained on the aggregate dataset using empirical risk minimization. Currently the two common techniques are Behavior Cloning and Noise Injection.

Behavior Cloning In Behavior Cloning, the sampling distribution is defined as $p(\mathbf{u}|\mathbf{x}, \psi) = \delta(\pi_{\theta*}(\mathbf{x}))$, where δ is Dirac delta distribution. In practice, this corresponds to allowing the supervisor to tele-operate the robot with no disturbances to their control signal. Behavior Cloning is one of the oldest Imitation Learning algorithms and has been used in teaching robots to follow along a road (Pomerleau, 1989; Bojarski et al., 2016) and perform manipulation tasks like part singulation (Laskey, Chuck, et al., 2017) and scooping a ball (Liang, Mahler, Laskey, Li, & Goldberg, n.d.).

Noise Injection Historically, Behavior Cloning has been noted to obtain high covariate shift and suffer from poor performance in practice. A recent alternative to vanilla Behavior Cloning, is to inject artificial noise into the control stream (Shiarlis, Wulfmeier, Salter, Whiteson, & Posner, 2018; Codevilla, Müller, Dosovitskiy, López, & Koltun, 2017; Laskey, Lee, et al., 2017; Duan et al., 2017).

Noise injected into the supervisor's policy simulates error occurring during testing. Under this noise, the supervisor is forced to take corrective actions in order to successfully perform the task. The corrective examples allow the robot to learn how to recover when it deviates from the supervisor's distribution. However, because the demonstrations are still concentrated around the supervisor's policy, it can have less cognitive load on the supervisor than on-policy methods, which are concentrated around the robot's policy. The intuition of providing corrective examples during data collection is shown in Fig. 2.

One common noise distribution to consider is Gaussian noise, where the covariance matrix is fixed through out the state space (Shiarlis et al., 2018; Laskey, Lee, et al., 2017; Duan et al., 2017). The covariance matrix is commonly a hyper-parameter that is tuned, however recently Laskey et al. proposed an algorithm known as DART, which attempts to automatically set the level of noise (Laskey, Lee, et al., 2017). We present an analysis of how DART behaves in Ch. 3. More complex correlated noise terms have also been considered by (Codevilla et al., 2017), for the task of training a self-driving car in simulation. .

On-Policy Sampling

On-Policy algorithms operate by executing the current robot's policy and allowing it to visit states under its current distribution. A supervisor then provides the correct control signal after the state has been visited. In physical robot systems, this feedback is usually performed via watching a video of the execution and providing a control signal (Laskey, Lee, et al., 2016a; Ross et al., n.d.). On-Policy algorithms are commonly iterative with K iterations, thus the robot final's policy is the same as that at iteration K (i.e. $\pi_{\theta^R} = \pi_{\theta^K}$). Two common On-Policy algorithms are DAgger and Policy Gradient.

DAgger Inspired by the Online Optimization algorithm, Follow-The-Leader (FTL), DAgger operates by collecting data from the current robot's policy at each iteration k, (i.e. $p(\mathbf{u}|\mathbf{x}, \psi) = \delta(\pi_{\theta^{k-1}}(\mathbf{x}))$) and then retraining the policy on the aggregrate dataset (Ross et al., 2011a):

$$\theta^{k+1} = \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{k} E_{p(\xi|\theta^i)} J(\theta, \theta^*|\xi)$$

DAgger has been applied successfully to robotics tasks, such as flying a quadrotor (Ross et al., n.d.) or grasping in clutter (Laskey, Lee, et al., 2016b). Several variants of DAgger exist that have shown to empirically increase performance. These variants include either stochastic mixing between the robot and supervisor's policy at decaying rate or initializing the aggregate data-set with samples from the supervisor's demonstrations. Recently, Cheng et al. showed the advantage of these initialization schemes is to stabilize the performance of the algorithms, so that DAgger converges to a fixed policy. However, a fixed cost is incurred by deviating from the original FTL algorithm (Cheng & Boots, 2018).

Many extensions to DAgger exist in the literature to deal with problems such as reducing data needed (Kim & Pineau, 2013; Laskey, Staszak, et al., 2016; Judah, Fern, & Dietterich, 2011) or safety (Zhang & Cho, 2016). These extensions generally add more complexity to the algorithm, but retain the original On-Policy paradigm. For example, in MMD-IL, (Kim & Pineau, 2013), proposes to use a novelty detector

based on density estimation to decide whether to query the supervisor for corrective feedback at each state visited.

Policy Gradient Policy Gradients are another on-policy algorithm that samples from a delta distribution around the current robot policy, $p(\mathbf{u}|\mathbf{x}, \psi) = \delta(\pi_{\theta^{k-1}}(\mathbf{x}))$. However, instead of performing finding the global solution at each iteration, an alternative is to use a policy gradient of the current surrogate loss to update to the next policy (Cheng, Yan, Wagener, & Boots, 2018). The motivation for this is to reduce the computational requirement of On-Policy algorithms, which can require a substantial number of updates to the model.

$$\theta^{k+1} = \theta^k + \eta \nabla_\theta E_{p(\xi|\theta^k)} J(\theta, \theta^*|\xi)$$

In the Online Optimization literature this approach is known as Online Gradient Descent (OGD) (Shalev-Shwartz et al., 2012). See (Cheng et al., 2018) for a more detailed discussion on this approach and how the original gradient for more complex variants such as the natural policy gradient (Kakade, 2002) using a mirror descent formulation.

Guided Policy Search One variation of On-Policy Imitation Learning algorithms, is Guided Policy Search (GPS). In GPS, the supervisor is typically some sort of planning algorithm that can be found via optimization of a cost function, such as iLQG. In order to collect data on the current robot's distribution, GPS modifies the original cost function with an additional term that penalizes the supervisor if it differs from the current robot's control at that state. This penalization term is typically the KL-divergence between the supervisor's current distribution and the robot's (Levine, Finn, Darrell, & Abbeel, 2015; Levine & Koltun, 2013).

While GPS is a way to collect data on the robot's current distribution, it operates by forcing the supervisor to trade-off between completion of the task and applying the controls of the robot. For arbitrary black box supervisor's such as humans, it becomes unclear how to perform this trade-off since the internal planner of the supervisor cannot be modified. Thus, in this study, we do not consider Guided Policy Search algorithms.

Chapter 3

Noise Injection in Off-Policy Sampling

An Estimation Perspective on Imitation Learning

One way to interpret On and Off-policy algorithms is that they are trying to make a prediction about where the final robot's policy is likely to visit after training. Off-policy methods argue that the robot is likely to be concentrated around the supervisor's current robot policy, while On-policy predicts it will visit states near the current policy.

To formulate this prediction problem, we can treat π_{θ^R} as a random variable. The randomness in θ^R comes from two sources; data collection and optimization. In data collection, N trajectories are sampled from $p(\xi|\psi)$ to form a data set $D = \{\xi\}_{n=1}^{N}$ of examples. The probability over this data set can be denoted as $p(D|\psi)$. After a data set is sampled a policy is then optimized via minimizing the empirical loss. This procedure can also be random due to techniques like stochastic gradient descent, we denote the probability over a policy as $p(\theta^R|D)$.

We can see the effect this randomness on θ^R has on the error by recalling the decomposition in the original objective:

$$
\begin{aligned}
&E_{p(\xi|\theta^R)} J(\theta^R, \theta^*|\xi) \\
&= \underbrace{E_{p(\xi|\theta^R)} J(\theta^R, \theta^*|\xi) - E_{p(\xi|\psi)} J(\theta^R, \theta^*|\xi)}_{\text{Covariate Shift}} + \underbrace{E_{p(\xi|\psi)} J(\theta^R, \theta^*|\xi)}_{\text{Generalization}},
\end{aligned}
$$

We can now consider the expected performance given this randomness over θ^R, to do so we can take the expectation over the $p(\theta^R|\psi) = p(\theta^R|D)p(D|\psi)$. We can write this as follows:

$$E_{p(\theta^R|\psi)}[E_{p(\xi|\theta^R)}J(\theta^R,\theta^*|\xi)]$$
$$= E_{p(\theta^R|\psi)}[\underbrace{E_{p(\xi|\theta^R)}J(\theta^R,\theta^*|\xi) - E_{p(\xi|\psi)}J(\theta^R,\theta^*|\xi)}_{\text{Shift}} + \underbrace{E_{p(\xi|\psi)}J(\theta^R,\theta^*|\xi)}_{\text{Loss}}],$$

The new expectation over the loss term can be interpreted as the familiar bias-variance trade-off in the random design setting (Hsu et al., 2012). However, the Covariate Shift term has a more subtle interpretation.

In order to study this, we introduce the following Lemma to bound the shift. This bound holds when the surrogate loss $\forall \mathbf{u}_1, \mathbf{u}_2 \in \mathcal{U}\ l(\mathbf{u}_1, \mathbf{u}_2) \in [0, 1]$. Examples of when this occurs is if the robot has a discrete set of actions or bounded continuous controls and they are normalized during learning.

Lemma 1 *If* $\forall \mathbf{u}_1, \mathbf{u}_2 \in \mathcal{U},\ 0 \leqslant l(\mathbf{u}_1, \mathbf{u}_2) \leqslant 1$*, the following is true:*

$$E_{p(\theta^R|\psi)}[|E_{p(\xi|\psi)}J(\theta,\theta^*|\xi) - E_{p(\xi|\psi)}J(\theta,\theta^*|\xi)|] \leqslant \frac{T}{2}E_{p(\theta^R|\psi)}\mathcal{D}_{\mathcal{KL}}\big(p(\xi|\pi_\theta), p(\xi|\psi)\big) + \frac{T}{2}$$

[See Appendix for Proof]

According to Lemma 1, in order to reduce covariate shift, we should minimize the KL-divergence between the sampling distribution and the current robot's distribution across the likely samples of θ^R. We can optimize the upper-bound to achieve this as follows:

$$\psi^* = \underset{\psi}{\text{argmin}}\ \frac{T}{2}E_{p(\theta^R|\psi)}[\mathcal{D}_{\mathcal{KL}}\big(p(\xi|\pi_{\theta^R}), p(\xi|\psi)\big)]$$
$$= \underset{\psi}{\text{argmax}}\ E_{p(\theta^R|\psi)}[E_{p(\xi|\theta^R)}\sum_{t=0}^{T-1}\log p(\mathbf{u}_t|\mathbf{x}_t,\psi)]$$

In order to condense notation, we can rewrite the upper-bound with respect the the distribution with $p(\theta^R, D)$ marginalized out. We can write this new distribution as follows:

$$p_{\bar{\theta}^R}(\xi) = \int_\Theta \int_\mathcal{D} p(\xi|\theta^R)p(\theta^R|D)p(D|\psi)dDd\theta$$

Under this new Markov chain the element $p_{\bar{\theta}^R}(\mathbf{u}_t|, \mathbf{x}_t)$ is now a distribution over the possible controls likely under the possible instances of θ^R. We can rewrite our objective now as follows:

$$\max_\psi E_{p_{\bar{\theta}^R}(\xi)}\sum_{t=0}^{T-1}\log p(\mathbf{u}_t|\mathbf{x}_t,\psi)$$

This objective can be seen as increasing the sampling policy's log-likelihood of applying controls similar to the current robot's policy. Thus, minimizing the covariate shift means increasing the likelihood of sampling the robot's final policy under the sampling distribution.

Off-Policy vs. On-Policy

The choice between On and Off-policy methods can be posed as what approach best optimizes Eq. 3. We can answer this by considering the following sampling distribution $p(\mathbf{u}|\mathbf{x}, \psi) = \mathcal{N}(\mu(\mathbf{x}), \Sigma)$, where $\mu(\mathbf{x}) \in \{\pi_{\theta*}(\mathbf{x}), \pi_{\theta^{k-1}}(\mathbf{x})\}$. This distribution is interesting because it can be set to represent a variety of the algorithms described in the previous section. We can now select whether the mean should be set to be On or Off policy with the following optimization:

$$\min_{\mu \in \{\pi_{\theta*}, \pi_{\theta^{n-1}}\}} E_{p_{\bar{\theta}^R}(\xi)} \sum_{t=0}^{T-1} (\mathbf{u}_t|\mathbf{x}_t - \mu(\mathbf{x}_t))^T \Sigma^{-1} (\mathbf{u}_t|\mathbf{x}_t - \mu(\mathbf{x}_t)) \tag{2}$$

, where $\mathbf{u}_t|\mathbf{x}_t$ reflects a random control that is possible under $p_{\bar{\theta}^R}(\mathbf{u}_t|, \mathbf{x}_t)$. To understand when On-Policy is preferred over Off-Policy, we introduce the following result.

Lemma 2 *On-Policy has a higher likelihood of sampling trajectories under $p_{\bar{\theta}^R}(\xi)$ if and only if the following is true:*

$$\sum_{t=0}^{T-1} E_{p_{\bar{\theta}^R}(\mathbf{x}_t)} ||\pi_{\bar{\theta}^R}(\mathbf{x}_t) - \pi_{\theta*}(\mathbf{x}_t)||_2^2 \geqslant \sum_{t=0}^{T-1} E_{p_{\bar{\theta}^R}(\mathbf{x}_t)} ||\pi_{\bar{\theta}^K}(\mathbf{x}_t) - \pi_{\theta^{K-1}}(\mathbf{x}_t)||_2^2$$

where $p_{\bar{\theta}^R}(\mathbf{x}_t)$ is the marginal distribution for the state $\mathbf{x}$ at the t time-step and $\pi_{\bar{\theta}^R}(\mathbf{x}_t) = \pi_{\bar{\theta}^K}(\mathbf{x}_t) = E_{p_{\bar{\theta}^R}(\mathbf{u}_t|, \mathbf{x}_t)} \mathbf{u}$, or the mean outcome of the robot's final policy. This result is interesting because it shows the two terms that dictate whether On-Policy or Off-Policy sampling is preferable.

The left-hand side of the inequality has an interesting interpretation related to the Bias of an estimator (Geurts, 2009). The Bias is generally a measure of how well in expectation the function matches the true label and is normally reduced by increasing the expressiveness of the policy. To illustrate this concept, we consider the following Ridge Regression example in the infinite sample setting. In this example. the policy representation is linear, $\pi_\theta(\mathbf{x}) = \theta^T \mathbf{x}$, which may be in some Hilbert space, and the Supervisor provides control signals, $\tilde{\mathbf{u}} = \theta^{*T} \mathbf{x} + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma^2)$. Note for simplicity $\mathbf{u} \in \mathbb{R}$, or the controls are 1-dimensional.

In this example, when an Off-Policy algorithm is run, we can derive in close form the solution.

$$\bar{\theta}^R = \underset{\theta}{\text{argmin}}\, E_{p(\xi|\psi)} E_{\tilde{\mathbf{u}}} \sum_{t=0}^{T} ||\theta^T \mathbf{x}_t - \tilde{\mathbf{u}}_t||_2^2 + \lambda ||\theta||_2^2$$

$$\bar{\theta}^R = \underset{\theta}{\text{argmin}}\, E_{p(\xi|\psi)} \sum_{t=0}^{T} ||\theta^T \mathbf{x}_t - \theta^{*T} \mathbf{x}_t||_2^2 + T\sigma^2 + \lambda ||\theta||_2^2$$

$$\bar{\theta}^R = (E_{p(\mathbf{x}_t|\psi)}[\mathbf{x}_t \mathbf{x}_t^T] + \lambda I)^{-1} E_{p(\mathbf{x}_t|\psi)}[\mathbf{x}_t \mathbf{x}_t^T] \theta^*$$

, where λ is the regularization term used in Ridge Regression. We can drop the noise term since, it's a constant unaffected by the optimization and consider the covariance matrix representation of the optimization.

Under this expression it is clear that the Left-hand side can be minimized as $\lambda \to 0$, since $\bar{\theta}^R \to \theta^*$. Thus, suggesting Off-Policy methods have a larger likelihood of sampling robot's distribution as the expressiveness of the model is increased. It should be noted though that this is in the infinite sample setting, which assumes sufficient data has been collected. In practice it may be difficult to achieve this for sufficiently large function classes. In the next section, we show how noise injection is needed to further increase the likelihood of these expressive models.

The right-hand side of the equation, corresponds to On-Policy sampling, and measures how close the previous policy is to the final. This quantity is more nuanced than the Off-Policy term and requires a more detailed in analysis, which we present in Chapter 4. However, the key result is that in order for this quantity to converge to zero, it may require increasing λ or reducing the expressiveness of the model class. The intuition for this is that smaller function classes are limited in the amount of variation per iteration.

By combining these two results we can infer the following, for the infinite sample case: Off-Policy methods perform better when the policy is expressive enough and On-Policy methods may prefer better when expressiveness is decreased. Thus, suggesting that as the expressiveness is varied, so should the choice between On and Off-Policy algorithms.

Experiment

To illustrate the above theoretical results, we construct a simple grid-world environment, where its possible to collect enough data to perfectly recover the supervisor's policy. In grid world, we have a robot that is trying to reach a goal state, whereat it receives $+10$ reward. The robot receives -10 reward if it touches a penalty state. The robot has a state space of (x, y) coordinates and a set of actions consisting of {Left, Right, Forward, Backward, Stay}. The grid size for the environment is

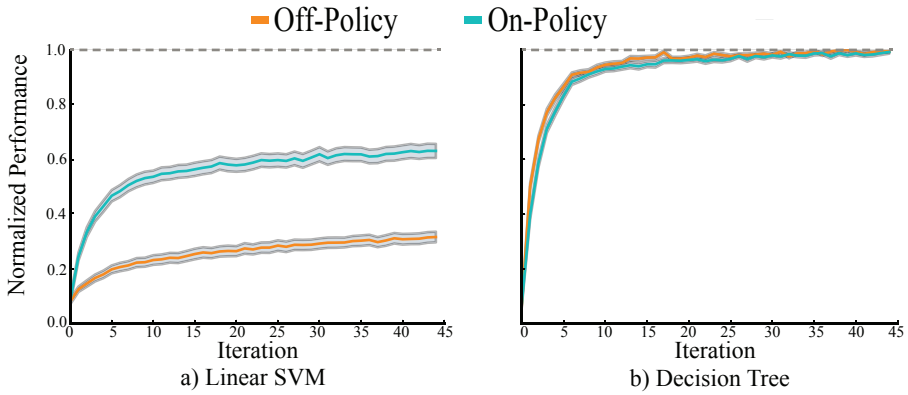


Figure 3.1: We compare On-Policy and Off-Policy for low- and high-expressiveness policy classes (a and b respectively) over 100 randomly generated 2D grid world environments, as a function of the amount of data provided to them. On-Policy outperforms in the low-expressive condition, but the performance gap is negligible in the high-expressive condition, when the policy class contains the expected supervisor policy.

15×15. 8% of randomly drawn states are marked as a penalties, while only one is a goal state. For the transition dynamics, $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$, the robot goes to an adjacent state different from the one desired uniformly at random with probability 0.16. The robot must learn to be robust to the noise in the dynamics, reach the goal state and then stay there. The time horizon for the policy is $T = 30$.

We use Value Iteration to compute an optimal supervisor. In all settings, we provided On-Policy with one initial demonstration from Off-Policy sampling before iteratively rolling out its policy. This initial demonstration set from Off-Policy sampling is common in On-Policy methods like DAgger (Ross et al., 2011a).

We run all trials over 100 randomly generated environments. We measure normalized performance, where 1.0 represents the expected cumulative reward of the optimal supervisor.

Low Expressiveness: Fig. 3.1(a) shows a case when the robot's policy class is empirically not expressive enough to represent the supervisor's policy. We used a Linear SVM for the policy class representation, which is commonly used in On-Policy (Ross & Bagnell, 2010; Ross et al., 2011a, n.d.). On-Policy outperforms Off-Policy , which is consistent with prior literature (Ross & Bagnell, 2010; Ross et al., 2011a). This outcome suggests that when the robot is not able to learn the supervisor's policy, On-Policy has a large advantage. Note that neither method converges to the supervisor's performance.

High Expressiveness: We next consider the situation where the robot's policy class is more expressive. We use decision trees with a depth of 100 to obtain a highly expressive function class.

As shown in Fig. 3.1(b), On-Policy and Off-Policy both converge to the true supervisor at the same rate. This suggests that when model is more expressive and has enough data, it is no longer as beneficial to obtain demonstrations with On-

Policy. However, we stress this domain is relatively easy and achieving sufficient data is possible to recover the supervisor. In the following, section we will study the finite sample case where data is limited.

Noise Injection in Sampling

In the previous section, we observed how increasing the expressiveness affects the choice between Off and On-Policy sampling in the infinite sample case. Off-Policy methods have a higher likelihood as the estimator becomes unbiased and On-Policy methods can become potentially unstable.

While, this is theoretically interesting, in practice estimators will be trained on finite data points. Thus, technique like Behavior Cloning, which only sample from the supervisor, may be brittle because the robot has to perfectly match at all states visited. One technique to correct for this is the injection of artificial noise into the sampling distribution of the supervisor. Noise injection can cause the supervisor to visit states due to errors made and make them provide corrective examples in order to recover.

The estimation perspective can also be used to analyze the effects of noise injection and determine when it could be beneficial. In order to do this, we first can consider what the best covariance matrix is with respect to Eq. 3 for our Gaussian Sampling distribution.

By substituting the Gaussian distribution into the objective, we can write the maximization for the covariance matrix as follows:

$$\Sigma^* = \arg\min_{\Sigma} -\frac{T}{2}\log|\Sigma^{-1}| + \frac{1}{2}E_{p_{\bar{\theta}^R}(\xi)}\sum_{t=0}^{T-1}\left(\mathbf{u}_t|\mathbf{x}_t - \pi_{\theta^*}(\mathbf{x}_t)\right)^T\Sigma^{-1}\left(\mathbf{u}_t|\mathbf{x}_t - \pi_{\theta^*}(\mathbf{x}_t)\right)$$

, which results in the maximum likelihood estimate for a covariance matrix. Before the optimization can be solved, there needs to be a choice for the mean $\mu(\mathbf{x})$. In order to consider, the best covariance matrix, we can set the mean to be the true mean of the estimators distribution at that state $\mu(\mathbf{x}_t) = \pi_{\bar{\theta}^R}(\mathbf{x}_t)$. We can then derive the optimal solution to this problem.

$$\Sigma^* = \frac{1}{T}E_{p_{\bar{\theta}^R}(\xi)}\sum_{t=0}^{T-1}(\mathbf{u}_t|\mathbf{x}_t - \pi_{\bar{\theta}^R}(\mathbf{x}_t))(\mathbf{u}_t|\mathbf{x}_t - \pi_{\bar{\theta}^R}(\mathbf{x}_t))^T$$

An natural interpretation of Σ^* is that it is the covariance matrix of the estimator's distribution (i.e. $\Sigma^* = \mathrm{Cov}(\pi_{\theta^R})$). The covariance matrix is very closely tied to the traditional idea of an estimator's variance, in the canonical bias-variance trade-off (Geurts, 2009) and measures the expected changes in each direction of the estimator as θ^R is sampled.

Common rates on terms like variance typically, show it increases with the expressiveness of the policy class and decreases with the number of data points (Hsu et al., 2012). Thus, if a very expressive estimator is used with a small data-set, we should expect to inject a large amount of noise. However, if the data-set is quite substantial (or appropriate in size for the expressiveness of the estimator) then only a small amount of noise should be needed.

In practice though computation of this term is not feasible due to it requiring knowledge of the final robot's policy. Thus, in the next section, we present a heuristic algorithm that is feasible for computing an estimate of this quantity.

DART

The above objective cannot be solved because $p_{\bar{\theta}^R}(\xi)$ and $\pi_{\bar{\theta}^R}$ are not known until after the robot has been trained. Thus, we will introduce three approximations to create a implementable algorithm.

Approximation 1 First, instead of using $\pi_{\bar{\theta}^R}$ for the mean of the sampling distribution $\mu(\mathbf{x})$, we will replace this with $\mu(\mathbf{x}) = \pi_{\theta*}(\mathbf{x})$. The intuition behind this is that we expect the robot to converge close to the supervisor's distribution, especially when an expressive estimator is used. Thus, we expect the average final robot's policy to be quite close to the supervisor's.

Approximation 2 The next approximation is the distribution for computing the optimization over. We will choose to replace $p_{\bar{\theta}^R}(\xi)$ with $p(\xi|\psi)$. The practical reason for this is that $p(\xi|\psi)$ is where the training data is collected from, thus it is the convenient choice for estimation. However, it also can be an appropriate choice because the parameters of $p(\xi|\psi)$ are being chosen to try and match $p_{\bar{\theta}^R}(\xi)$.

Approximation 3 The final approximation need is the relaxation of using the full distribution, $\mathbf{u}_t|\mathbf{x}_t$ to just a single sample of the robot's policy $\pi_{\theta^k}(\mathbf{x}_t)$. The practical reason for this is to avoid having to preform computationally expensive retraining of the robot's policy to sample from $p(\theta^k|\psi)$. The intuition for why this is reasonable lies in the fact that the covariance matrix is shared across all states, thus the policy variation among states is likely to dominate over changes from randomness in the dataset.

Since the robot's policy changes as more data is added, we propose an iterative solution, where after large batch sizes the covariance matrix is updated. We can combine all these approximations to form the following iterative update for the covaraince matrix.

$$\Sigma_{k+1} = \frac{1}{T} E_{p(\xi|\psi^k)} \sum_{t=0}^{T-1} (\pi_{\theta^k}(\mathbf{x}_t) - \pi_{\theta*(\mathbf{x}_t)})(\pi_{\theta^k}(\mathbf{x}_t) - \pi_{\theta*}(\mathbf{x}_t))^T \tag{3}$$

In order to potentially obtain a better estimate of the covariance matrix, we can leverage the intuition that the noise injected is meant to simulate the variance of the

estimator. Since, the variance typically decays as more data is added, we can scale the overall magnitude of the noise injected to a lower amount, if we think the current dataset is too small to reflect the accuracy of the final robot's policy. The intuition behind this prior is that the relative shape of the noise will remain similar, but the magnitude will decrease as more data is added. Given a target noise magnitude, α, the scaling can be set as follows:

$$\Sigma_{k+1}^{\alpha} = \frac{\alpha}{\text{tr}(\Sigma_{k+1})}\Sigma_{k+1}$$

We can now describe our algorithm DART (Disturbances Augmenting Robot Training), which iteratively solves for ψ_{k+1}^{α} to collect data and train the final robot policy. First N demonstrations are collected from a supervisor with an initial noise parameter set. Then a policy, $\pi_{\theta^{k+1}}$, is learned via empirical risk minimization on the aggregate dataset. The learned policy is then used to optimize Eq. 3 based on sample estimates and the outcome is optionally scaled based on α. Once the noise term is found N demonstrations are collected with the noise-injected supervisor and the robot is trained on the aggregate dataset. The algorithm is repeated for K iterations.

To understand how a the chosen Σ affects performance with respect to the Σ^*, we can study the curvature of the optimization function. The following Lemma provides an initial step to showing that the closer Σ is to Σ^* in Frobenius norm the closer their likelihoods are.

Lemma 3 *Given a valid covariance matrix for Σ, where $\Sigma \in S$, the following is true*

$$|E_{p_{\tilde{\theta}^R}(\xi)} \sum_{t=0}^{T-1} log\, p(\mathbf{u}_t|\mathbf{x}_t, \psi = \{\mu(\mathbf{x}_t), \Sigma^*\}) - E_{p_{\tilde{\theta}^R}(\xi)} \sum_{t=0}^{T-1} log\, p(\mathbf{u}_t|\mathbf{x}_t, \psi = \{\mu(\mathbf{x}_t), \Sigma^*\})|$$
$$\leqslant G_2||\Sigma^* - \Sigma||_F$$

where $G_2 = \max_{\Sigma \in S} ||\nabla_{\Sigma} E_{p_{\tilde{\theta}^R}(\xi)} \sum_{t=0}^{T-1} log\, p(\mathbf{u}_t|\mathbf{x}_t, \psi = \{\mu(\mathbf{x}_t), \Sigma^*\})||_F$

One possible direction for future work, is to try and derive a bound on the Frobenius norm. However, in the next section we will see empirically that DART can significantly increase the performance over standard Behavior Cloning, thus suggesting that the likelihood is able to be increased in practice.

Experiments

Our experiments are designed to explore:

1. Does DART reduce covariate shift as effectively as On-Policy methods?

2. How much does DART reduce the computational cost and how much does it decay the supervisor's performance during data collection?

3. If the policy becomes less expressive is DART able to perform well?

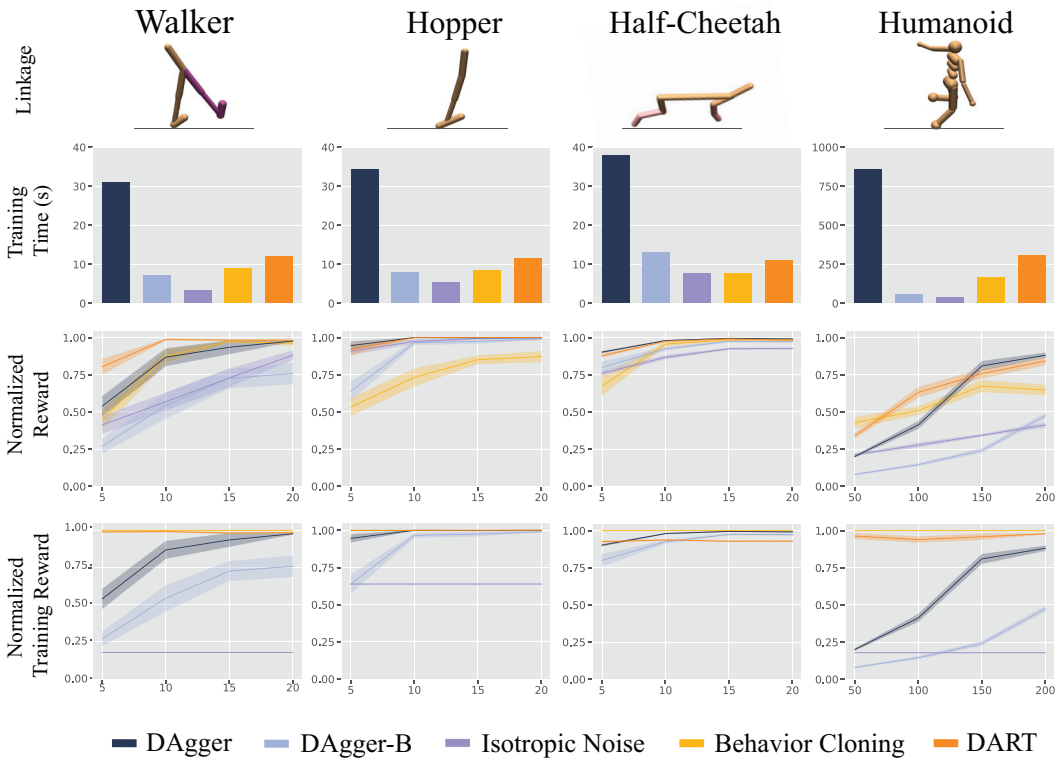


Figure 3.2: Top: The four different locomotive domains in MuJoCo we evaluated DART on: Walker, Hopper, Half-Cheetah and Humanoid. Top Middle: The time, in seconds, to achieve the performance level reported below. DART achieves similar performance to DAgger in all 4 domains, but requires significantly less computation because it doesn't require retraining the current robot policy after each demonstration. DAgger-B reduces the computation required by less frequently training the robot, but suffers significantly in performance in domains like Humanoid. Bottom Middle: The learning curve with respect to reward obtained during training with each algorithm plotted across the number of demonstrations. Bottom: The reward obtained during collection of demonstrations for learning. DART receives near the supervisor's reward at all iterations whereas DAgger can be substantially worse in the beginning.

To test how well DART performs compared with an on-policy methods such as DAgger and off-policy methods like Behavior Cloning, we use MuJoCo locomotion environments (Todorov et al., 2012). The challenge for these environments is that the learner does not have access to the dynamics model and must learn a control

policy that operates in a high-dimensional continuous control space and moves the robot in a forward direction without falling.

We use an algorithmic supervisor in these domains: a policy which is trained with TRPO (Schulman, Levine, Moritz, Jordan, & Abbeel, 2015) and is represented as a neural network with two hidden layers of size 64. Note, while TRPO uses a stochastic policy, the supervisor is the deterministic mean of the learner. This is the same supervisor used in (Ho & Ermon, 2016). For all 4 domains, we used the same neural network as the supervisor and trained the policies in Tensorflow. At each iteration we collected one demonstration. In order to make the task challenging for Imitation Learning, we used the same technique as in (Ho & Ermon, 2016), which is to sub-sample 50 state and control pairs from the demonstrated trajectories, making the learners receive less data at each iteration.

We used the following MuJoCo environments: Walker, Hopper, Humanoid, and Half-Cheetah. The size of the state and control space for each task is $\{|\mathcal{X}| = 17, |\mathcal{U}| = 6\}$, $\{|\mathcal{X}| = 11, |\mathcal{U}| = 3\}$, $\{|\mathcal{X}| = 376, |\mathcal{U}| = 17\}$, $\{|\mathcal{X}| = 117, |\mathcal{U}| = 6\}$, respectively. All experiments were run on a MacBook Pro with an 2.5 GHz Intel Core i7 CPU. We measured the cumulative reward of each learned policy by rolling it for 500 timesteps, the total computation time for each method and the cumulative reward obtained during learning.

We compare the following five techniques: Behavior Cloning, where data is collected from the supervisor's distribution without noise injected; DAgger, which is an on-policy method that stochastically samples from the robot's distribution and then fully retrains the robot's policy after every demonstration is collected; DAgger-B, which is DAgger but the retraining is done intermittently to reduce computation time; Isotropic-Gaussian which injects non-optimized isotropic noise; and DART. See supplement material for how the hyper-parameters for each method are set.

DAgger traditionally calls for controlling the trade-off between supervisor and robot with a hyperparamter β, which parameterizes an exponential decaying distribution over the probability of taking the supervisor's action. In order to explore the effectiveness of on-policy approaches, or those that strongly sample from the robot's policy, we swept several values,$\{0.0, 0.1, 0.3, 0.5\}$, for β, the stochastic mixing hyper-parameter, and found that setting $\beta = 0.5$ yields the best results. Since, DART determines how much stochasticity to inject as part of the algorithm, we compare in Sec. **??**, how much data is used to perform this grid-search versus DART's online optimization.

For noise injection algorithms, we chose $\alpha = T\text{tr}(\hat{\Sigma}_k)$, which corresponds to no additional knowledge being used. For Isotropic-Gaussian noise injection, we set $\Sigma_k^\alpha = I$ for all k. Given a demonstration a learner only receives 50 examples that are sub-sampled in the manner described by Ho and Ermon (Ho & Ermon, 2016). The covariance matrix is computed on the remaining held-out data points from the demonstration. To understand how sensitive DART is to the amount of held-out data needed to compute the noise term, in the second experiment we compute it on

10% of held-out data and observe no significant difference in reward obtain, despite the fact DART is trained on less data than the other methods.

In the MuJoCo tasks once the agent falls over the environment terminates and the task ends. Thus, individual trajectories can have significantly different lengths depending on the sampling algorithm. In light of this, we consider two different scenarios for evaluation of DART; trajectory and state level. In the trajectory level, the amount of trajectories each algorithm observed is held constant. This level is interesting to consider because in a variety of domain there is significant overhead to starting a new instance of the task because it can require restarting an environment, which may need significant human intervention.

Trajectory Level In the Walker, Hopper and Half-Cheetah domains, we ran these algorithms for 20 iterations, evaluating the learners at 5, 10, 15, and 20 iterations. One initial supervisor demonstration was always collected in the first iteration. To remain consistent, we updated DAgger-B and DART at the same iterations: iteration 2 and iteration 8.

In the Humanoid domain, we ran the algorithms for 200 iterations and evaluated the learners at iterations 50, 100, 150, and 200. Again, we collected one initial supervisor demonstration for DAgger and Isotropic-Gaussian. For DAgger-B and DART, we collected 60 initial demonstrations, in their respective manner, and then updated the model every 15 iterations afterwards.

Fig. 3.3 shows the results. In all domains, DART achieves parity with DAgger, whereas Behavior Cloning and DAgger-B are below this performance level in Walker and Humanoid. For Humanoid, DART is $3x$ faster in computation time and during training only decreases the supervisor's cumulative reward by 5%, whereas DAgger executes policies that have over 80% less cumulative reward than the supervisor. The reason for this is that DAgger requires constantly updating the current robot policy and forces the robot into sub-optimal states. While, one way to reduce this computation is to decrease the number of times the policy is updated. DAgger-B illustrates that this can significantly deteriorate performance in domains like Walker. Lastly, naively applying isotropic noise does not perform well, and leads to unsafe policies during execution, which suggests the need for optimizing the level of noise.

State Level While DART was able to perform significantly better over techniques like DAgger-B in the previous experiment, this could be attributed to DART seeing more data during learning, since the trajectories varied in length. To control for this variable, we will now consider running each algorithm with fixed amount of examples from the supervisor. To accomplish this, we will execute each agent until a fix amount of examples is gather and then update the model. Furthermore, to ensure DART doesn't see any more data when computing the noise term, we will hold out 10% of its total data for computation of the noise term.

In the Walker, Hopper and Half-Cheetah domains, we ran the algorithms until they each collected 400 data points, evaluating the learners every 50 data points. One initial supervisor demonstration was always collected at the beginning, account-

ing for the first 50 data points. We updated the DAgger policy after every 50 data points collected. To remain consistent, we updated both DAgger-B and DART after every 300 data points collected.

In the Humanoid domain, we ran the algorithms until they each collected 10,000 data points and evaluated the learners after every 1250 data points. Again, we collected one initial supervisor demonstration. For DAgger we updated the policy after every 200 data points collected. For DAgger-B and DART we updated the policies after every 1,000 data points collected.

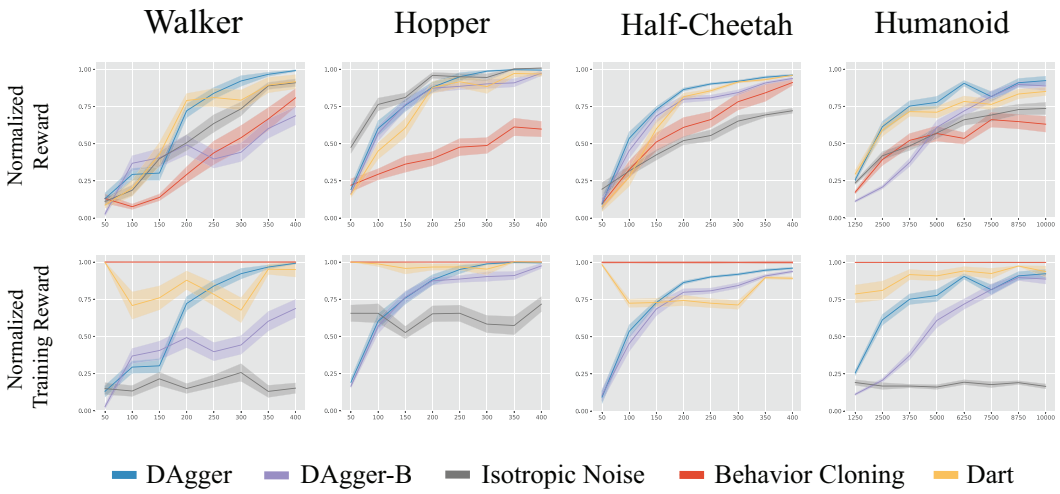


Figure 3.3: The comparison of the five algorithms when the amount of labeled examples seen is held fixed. Top: The learning curve with respect to reward obtained during training with each algorithm plotted across the number of labeled examples. Bottom: The reward obtained during collection of demonstrations for learning. DART receives close to the supervisor's reward in all iterations, however, in some domains such as Half-Cheetah, it can decay up to 25%, whereas DAgger can be substantially worse in the beginning.

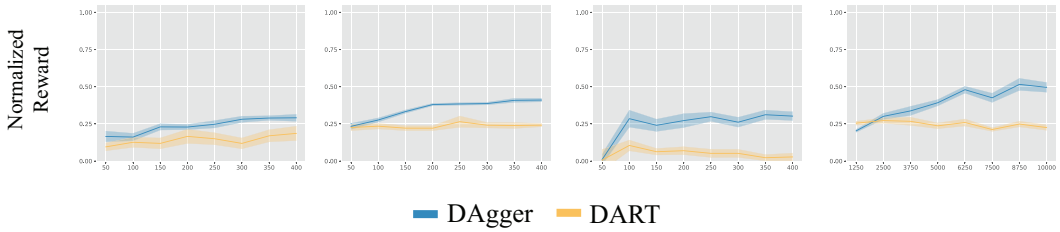


Figure 3.4: The comparison of DART and DAgger when the policy class is not expressive, or in this case linear. The learning curve with respect to reward obtained during training with each algorithm plotted across the number of labeled examples. In all domains DAgger outperforms DART, which is consistent with our theoretical analysis that suggest Off-Policy methods perform well when the expressiveness is increased.

As shown above the results are similar to that in the trajectory level, however

in Walker and Half-Cheetah DART does suffer slightly worse in performance during data collection compared to the trajectory level. Part of this could be the tiny amount of data used to estimate the noise term, this could have lead to a higher level of noise being injected earlier. One technique to correct for this is to use the regularization term α to add a prior over the magnitude of noise. Another difference observed is that in the Humanoid experiment DAgger-B performs at an improved rate. Thus, suggesting if On-Policy methods are allowed to fail more times they can be improved. However in Walker, DAgger-B still suffers significantly in performance, which indicates On-Policy methods work best when updated frequently.

Low Expressiveness Our theoretical analysis suggest that Off-Policy methods would perform poorly as the expressiveness is decreased. Given that DART makes several assumptions that reflect having a policy with low bias, we should expect DART to perform badly as the expressiveness is decreased. To test this we compare DART and DAgger with the same parameters as before, but now with linear policy class instead of a neural network.

In Fig. 3.4, we report the performance with the linear model versus examples labeled. DAgger significantly outperforms DART in all domains, thus suggesting as the model expressiveness is decreased On-Policy methods are preferable.

Chapter 4

Convergence of On-Policy Methods

The choice of On-policy methods has often been motivated from their theoretical analysis which shows they have superior performance to Off-Policy methods (Levine & Koltun, 2013; Kim & Pineau, 2013; Laskey, Staszak, et al., 2016). Traditionally, On-Policy algorithms have been analyzed with a regret style analysis taken from the Online Optimization literature.

A limitation with this current analysis though, is that it doesn't necessarily say whether the policy π_{θ^R} is optimal in that:

$$\theta^R = \underset{\theta}{\operatorname{argmin}} E_{p(\xi|\theta^R)} J(\theta, \theta^*|\xi)$$

, or the robot's policy returned is the best under its distribution. Instead the majority of analysis for On-Policy techniques state that there exists a policy that has error bounded by the loss on the aggregate dataset and an additional term.

In Sec. 4, we will show that this bounded loss can at times be substantially higher than Off-Policy methods performance. Then building on ideas from (Cheng & Boots, 2018), we will show how a new regret formulation can be used to understand when On-Policy methods achieve this optimality condition. Finally, in Sec.4 , we will see one way to obtain this condition is to decrease the expressiveness of the policy class.

Limitations with Static Regret Analysis

The analysis of On-Policy Sampling in (Ross et al., 2011a) is performed in the context of the regret framework of online learning. In online learning, at each iteration $k \in \{1, \ldots, N\}$ the algorithm chooses an action $\theta_k \in \Theta$ (e.g. a policy to roll

out) (Shalev-Shwartz et al., 2012), and observes some loss $f_k(\theta_k)$. Online learning is analyzed in the context of a regret guarantee: an upper bound (e.g. $\sqrt{N}$) on the cumulative loss incurred by the algorithm relative to taking just the best single action in Θ:

$$\sup_{\theta \in \Theta} \sum_{k=1}^{N} f_k(\theta_k) - f_k(\theta).$$

In the context of robotics and specifically in the case of On-Policy, taking an action θ_k at iteration k is rolling out a policy dictated by θ_k which induces a series of states $\mathbf{x}_{k,1}, \ldots, \mathbf{x}_{k,T}$ and taking the aggregate loss evaluated with respect to *these* states. Because these particular states could have been produced by a poor policy due to initialization, they could have little relevance to the task of interest and consequently, it makes little sense to compare to the policy that performs best on these particular states. What is important is the absolute performance of a policy on the task of interest. In the notation of Ross et al. (Ross et al., 2011a) this notion of relative regret is encoded in the ϵ_N error term that appears in their bounds and is defined relative to the particular set of rollouts observed by the On Policy LfD policy.

As an example for why low-regret may be uninformative, consider a car-racing game where the car has constant speed and the policy only chooses between straight, left, or right actions at the current state. Suppose at some point in the race the car encounters a "v" in the road where one path is a well-paved road and the other path is a muddy dirt road with obstacles. The car will finish the race faster by taking the paved road, but due either to the stochastic dynamics or imperfectness of the supervisor it is possible that after just a small number of supervisor demonstrations given to the robot to initialize On-Policy, a poor initial policy will be learned that leads the car down the dirt road instead of the paved road. When this policy is rolled out the supervisor will penalize the decision at the time point of taking the dirt versus paved road. But if the policy's actions agree with the supervisor once the car is on the dirt road (i.e., making the best of a bad situation) this policy will incur low-regret because the majority of the time the policy was acting in accordance with the supervisor. Thus, in this example a globally bad policy will be learned (because it took the dirt road instead of the paved road) but relative to the best policy acting on the dirt road, it performs pretty well.

The next theorem shows an instance in which an initial policy can be different enough from the optimal policy that the states visited by the initial policy – even with corrective feedback from the supervisor – are not informative or relevant enough to the task to guide On-Policy to the optimal policy.

Theorem 1 *For a given environment (state space, action space, dynamics mapping state-action pairs to states, and a loss function) and policy class Θ, let θ^N_{Off} be the*

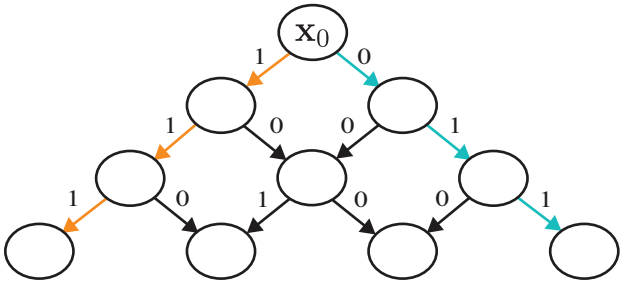


Figure 4.1: A Directed Acyclic Graph, where a robot is being taught by a supervisor to descend down and achieve maximum cumulative reward, which is shown via the numbers on each branch. Each node has the action the supervisor would select, either left or right, $\{L, R\}$. Off-Policy converges to the Orange path, which is optimal. However, the On Policy method converges to the Teal path, because it tries to learn on examples from that side of the tree.

policy learned from N supervisor demonstrations, and let θ_{On}^N be the policy learned by Off policy procedure, Behavior Cloning, with m supervisor demonstrations. Then there exists an environment and policy class Θ such that

- *θ^* is the unique minimizer of $E_{p(\xi|\theta)}[J(\theta)]$,*
- *$\lim_{N\to\infty} \theta_{Off}^N = \theta^*$ with probability 1,*
- *$\lim_{N\to\infty} \theta_{On}^N \neq \theta^*$ with probability at least ce^{-m}*

for some universal constant c. In other words, even with infinite data On-Policy may converge to a sub-optimal policy while Off-Policy converges to the best policy in the class that uniquely achieves the lowest loss.

Let $\{\{(\mathbf{x}_{k,t}, \mathbf{u}_{k,t}\}_{t=0}^3\}_{k=1}^N$ denote N trajectories. Consider an environment with a deterministic initial state at the root of the DAG of Figure 4.1 so that $p(\mathbf{x}_{k,0} = \texttt{root}) = 1$ for all k. The space of policies are constant functions $\Theta = \{L, R\}$ where if $\theta = L$, then regardless of the state, the control input $\mathbf{u}_{k,t}$ will be to take the left child (and analogously for $\theta = R$ taking the right child). For any state in the DAG with children, let $\phi(\mathbf{x}_{k,t}, \theta)$ denote the left child of $\mathbf{x}_{k,t}$ if $\theta = L$. Otherwise, denote it the right child. The dynamics are described as follows: for some $\mu \in (0, 1/4]$ to be defined later, if $\theta = L$ and $\mathbf{x}_{k,t} = \texttt{root}$ then $p(\mathbf{x}_{k,1} = \phi(\mathbf{x}_{k,0}, R)) = \mu$ and $p(\mathbf{x}_{k,1} = \phi(\mathbf{x}_{k,0}, L)) = 1 - \mu$, but if $\theta = R$ then the right child is chosen with probability 1. If $\mathbf{x}_{k,t} \neq \texttt{root}$ then $\mathbf{x}_{k,t+1} = \phi(\mathbf{x}_{k,t}, \theta)$.

Assume that the supervisor $\pi^* : \mathcal{X} \to \{L, R\}$ acts greedily according to the given rewards given in the DAG so that $\pi^*(\mathbf{x}_{k,t}) = L$ if the reward of the left child exceeds the right child and $\pi^*(\mathbf{x}_{k,t}) = R$ otherwise. Finally, define the state loss function $\ell(\cdot, \cdot)$ as the 0/1 loss so that after N trajectories, the loss is given as $J_N(\theta) = \sum_{k=1}^N \sum_{t=0}^2 \mathbf{1}\{\pi^*(\mathbf{x}_{k,t}) \neq \theta\}$ for all $\theta \in \{L, R\}$. Note that $\widehat{\theta}_N = \arg\min_{\theta\in\{L,R\}} J_N(\theta)$ is equivalent to looking at all actions by the supervisor over the states and taking the majority vote of L versus R (i.e., the states in which these actions are taken has no

impact on the minimizer). Note that we have not yet specified how the states $\mathbf{x}_{k,t}$ were generated.

We can compute the true loss when the trajectories are generated by $\theta \in \{L, R\}$. Let the empirical distribution of observed states under a fixed action $\theta \in \{L, R\}$ be given by $p(\xi \mid \theta)$, then

$$\begin{aligned} E_{p(\xi|\theta=L)} J(\theta = L) &= p(\mathbf{x}_{k,1} = \phi(\mathbf{x}_{k,0}, L)) \cdot 0 + \\ &\qquad p(\mathbf{x}_{k,1} = \phi(\mathbf{x}_{k,0}, R)) \cdot 2 = 2\mu \\ E_{p(\xi|\theta=R)} J(\theta = R) &= 1 \end{aligned}$$

which implies that $\theta_* = L$ and performs strictly better than R whenever $\mu < 1/2$.

It follows by the stochastic dynamics that the demonstrated action sequence by the supervisor equals $\{L, R, R\}$ with probability μ and $\{L, L, L\}$ with probability $1 - \mu$. After m supervisor sequences, the expected number of L actions is equal to $\mu + 3(1 - \mu) = 3 - 2\mu$ while the expected number of R actions is equal to just 2μ. By the law of large numbers, if only supervisor sequences are given then $\arg\min_{\theta \in \{L,R\}} J_m(\theta) \to L = \theta^*$ as $m \to \infty$ since $3 - 2\mu > 2\mu$ for all $\mu < 1/2$. In other words, $\theta^N_{Off} \to \theta^*$ as $N \to \infty$.

We now turn our attention to the On-Policy agent. Note that if after m supervisor action sequences we have that the number of observed R's exceeds the number of observed L's, then the On policy will define $\theta^m_{On} = R$. This proof assumes $\beta = 0$ in the On Policy (DAgger) algorithm (Ross et al., 2011a), however it can be extended to include β without loss of generality. It is easy to see that a policy rolled out with $\theta = R$ will receive the supervisor's action sequence $\{L, R, R\}$ and thus R will remain the majority vote and consequently $\theta^N_{On} = R$ for all $N \geqslant m$. What remains is to lower bound the probability that given m supervisor demonstrations, $\theta^m_{On} = R$.

For $k = 1, \ldots, m$ let $Z_k \in \{0, 1\}$ be independent Bernoulli(μ) random variables where $Z_k = 1$ represents observing the supervisor sequence $\{L, R, R\}$ and $Z_k = 0$ represents observing $\{L, L, L\}$. Given m supervisor sequences, note that the event $\arg\min_{\theta \in \{L,R\}} J_m(\theta) = R$ occurs if $\frac{1}{m} \sum_{k=1}^m Z_k > 3/4$. Noting that $\sum_{k=1}^m Z_k$ is a binomial(m,μ) random variable, the probability of this event is equal to

$$\sum_{k=\lfloor 3m/4 \rfloor + 1}^{m} \binom{m}{k} \mu^k (1 - \mu)^{m-k} \geqslant \Phi\left(\frac{\lfloor 3m/4 \rfloor + 1 - \mu m}{\sqrt{m\mu(1 - \mu)}} \right)$$

where we have used Slud's inequality (Slud, 1977) to lower bound a binomial tail by a Gaussian tail. Setting $\mu = 1/4$ we can further lower bound this probability by $\Phi\left(\frac{m+2}{\sqrt{3m/4}} \right) \geqslant ce^{-m}$ for some universal constant c. Consequently, with probability at least ce^{-m} we have that $\lim_{N \to \infty} \theta^N_{On} = R \neq \theta^*$ which implies that the expected loss of $\lim_{N \to \infty} \theta^N_{On}$ must exceed the expected loss of $\theta_* = L$.

We note there exist techniques to correct for this specific problem. One approach is to consider the value of each action and select actions that lead to higher reward during roll-out (Ross & Bagnell, 2014).However, the example is meant to illustrate the limitations of static regret analysis and the need for stronger rates.

Convergence with Respect to Dynamic Regret

In order to show convergence of an on-policy algorithm, we are interested in showing that the policies generated by the algorithm perform well on the loss on their own induced state distributions. To measure this, we turn to the dynamic regret, defined as

$$R_D(\theta_1, \ldots, \theta_K) := \sum_{k=1}^{K} f_k(\theta_k) - \sum_{k=1}^{K} \min_{\theta \in \Theta} f_k(\theta). \tag{4}$$

In comparison to the more well known static regret, which compares the algorithm's sequence of parameters to the single fixed parameter, dynamic regret compares the k th policy to the instantaneous best policy on the k th distribution (Zinkevich, 2003). The advantage of the dynamic regret metric is that the optima track the changes in state distribution so that a policy's performance is always evaluated with respect to the most relevant state distribution, which is the current one. Thus we can examine optimality properties of an algorithm by observing the convergence of the *average* dynamic regret, defined as $\frac{1}{K} R_D$.

The dynamic regret of an algorithm is fundamentally dependent on the change in the loss functions over iterations, often expressed in terms of quantities called variations. If the loss functions change in an unpredictable manner, we can expect large variation terms leading to large regret and sub-optimiality.

In imitation learning, the variation of the loss functions is related to the amount change in the state distribution induced by the sequence of policies. One way to control the changes is to restrict the possible options the model has to choose or its expressiveness. In the next section, we will illustrate this with Ridge Regression.

Assumptions

Before we can introduce our result, we need to make several assumptions which are common for On-Policy analysis (Cheng & Boots, 2018; Ross, Gordon, & Bagnell, 2011b). The first assumption is that the loss function is strongly convex with respect to the policy parameters and the change is bounded, or

Assumption 1 Let $\nabla_\theta E_{p(\xi|\theta^k)} J(\theta, \theta^*|\xi)$ be the derivative with respect to the policy evaluated. The following is true:

1. J is uniformly α-strongly convex in the second argument: $\forall \psi \in \Psi, \theta^1, \theta^2 \in \Theta, E_{p(\xi|\theta^k)} J(\theta^2, \theta^*|\xi) \geqslant E_{p(\xi|\theta^k)} J(\theta^1, \theta^*|\xi) + < \nabla_{\theta^1} E_{p(\xi|\theta^k)} J(\theta^1, \theta^*|\xi), \theta^1 - \theta^2 > + \frac{\alpha}{2}||\theta^1 - \theta^2||^2$.

2. $\forall \theta^1, \theta^2 \in \Theta, ||\nabla_{\theta^1} E_{p(\xi|\theta^k)} J(\theta^1, \theta^*|\xi) - \nabla_{\theta^2} E_{p(\xi|\theta^k)} J(\theta^2, \theta^*|\xi)|| \leqslant \gamma ||\theta^1 - \theta^2||$ and $\exists G > 0$ such that $\|\nabla_{\theta^1} E_{p(\xi|\theta^k)} J(\theta^1, \theta^*|\xi)\| \leqslant G$.

Additionally, the analysis requires an assumption about how sensitive the loss is to the change in the underlying sampling distribution. Intuitively, this is related to how large the covariate shift can be given a change in the policy parameters.
Assumption 2 Let $\nabla_\theta E_{p(\xi|\psi)} J(\theta, \theta^*|\xi)$ be the derivative with respect to the policy evaluated and $\theta_1, \theta_2 \in \Theta$. If $\beta \geqslant 0$, the following is true

$$||\nabla_\theta E_{p(\xi|\theta^1)} J(\theta, \theta^*|\xi) - \nabla_\theta E_{p(\xi|\theta^2)} J(\theta, \theta^*|\xi)||_* \leqslant \beta ||\theta^1 - \theta^2||$$

Convergence Results

We now present our main novel results about well-studied algorithms from online optimization in the context of imitation learning for the infinite sample case. Let $\theta_k^* = \arg\min_{\theta \in \Theta} E_{p(\xi|\theta^k)} J(\theta, \theta^*|\xi)$ be the optimal parameter at iteration k. We begin with a result concerning a stability constant $\lambda := \frac{\beta}{\alpha}$. (Cheng & Boots, 2018).

Lemma 4 *Given the assumptions, the following equality holds on the difference between consecutive optimal parameters for follow-the-leader and online gradient descent algorithms at any k:*

$$\|\theta_*^{k+1} - \theta_*^k\| \leqslant \lambda \|\theta^{k+1} - \theta^k\|.$$

[See Appendix For Proof]

This lemma suggests that in the case where $\lambda < 1$, we know with certainty that $\|\theta_*^{k+1} - \theta_*^k\| < \|\theta^{k+1} - \theta^k\|$. In other words, the optimal parameters cannot runaway faster than the algorithm's parameters. This intuition is also consistent with the findings of prior work (Cheng & Boots, 2018), which shows that convergence of the Kth policy can be guaranteed when $\lambda < 1$ for follow-the-leader. We will now examine how this idea of stability can be applied to common On-Policy IL algorithms.

DAgger

The first algorithm of interest is DAgger, which is defined in Chapter 2. We show convergence in dynamic regret via a corollary to Theorem 2 of (Cheng & Boots, 2018).

Corollary 1 *For DAgger under the assumptions, if* $\lambda < 1$*, then* $\frac{1}{K}R_D \to 0$ *as* $K \to \infty$*.*

The proof is immediate from the result of Theorem 2 of (Cheng & Boots, 2018). We have $E_{p(\xi|\theta^k)}J(\theta^k, \theta^*|\xi) - E_{p(\xi|\theta^k)}J(\theta^k_*, \theta^*|\xi) \leqslant \frac{\left(\lambda e^{1-\lambda}G\right)^2}{2\alpha k^{2(1-\lambda)}}$. Summing from 1 to K, we get $\sum_{k=1}^K E_{p(\xi|\theta^k)}J(\theta^k, \theta^*|\xi) - \sum_{k=1}^K E_{p(\xi|\theta^k)}J(\theta^k_*, \theta^*|\xi) \leqslant \sum_{k=1}^K \frac{\left(\lambda e^{1-\lambda}G\right)^2}{2\alpha k^{2(1-\lambda)}} = O(\max(1, K^{2\lambda-1}))$. Then the average dynamic regret is $\frac{1}{K}R_D = O(\max(1/K, K^{2\lambda-2}))$, which goes to zero.

Policy Gradient

We can also study the convergence of the policy gradient algorithm with respect to dynamic regret. Policy gradient as defined in Chapter 2 is when a gradient update is made to the policy based on the current surrogate loss.

Theorem 2 *For policy gradient under the assumptions, if* $\lambda < 1$*,* $2\lambda < \psi$ *and the step size,* $\eta = \frac{\alpha(\alpha^2 - 2\gamma\beta)}{2\beta^2(\alpha^2 - \gamma^2)}$*, then* $\frac{1}{K}R_D \to 0$ *as* $K \to \infty$*.*

Note the above result indicates for a policy gradient, we require a stronger condition that $\alpha^2 > 2\gamma\beta$. Written another way, the condition is $2\lambda < \psi$ where λ is the stability constant and $\psi = \frac{\alpha}{\gamma}$ is the condition number of f_k. So we require that the problem is both stable and well-conditioned.

These results are interesting because they show under certain conditions common On-Policy algorithms can converge in dynamic regret. The conditions though depend on quantities such as the strong convexity term and properties of the environment's dynamics (i.e. β). In the next section, we will show how to modify these properties via decreasing the expressiveness of the model.

The Effect of Model Expressiveness on Convergence

To understand how these convergence proofs affect performance in practice and the effects of model expressiveness on optimality, we will return to the Linear Regression example presented in Chapter 2. In this example. the policy representation is linear,$\pi_\theta(\mathbf{x}) = \theta^T\mathbf{x}$, which which may be in some Hilbert space, and the Supervisor provides control signals, $\tilde{\mathbf{u}} = \theta^{*T}\mathbf{x} + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma^2)$. Note for simplicity $\mathbf{u} \in \mathbb{R}$, or the controls are 1-dimensional. However a similar result can be shown for the multi-dimensional setting.

Our loss function or f_n defined in the previous section expected loss over the distribution on trajectories and the control labels.

$$f_k = \frac{1}{2} E_{p(\xi|\theta^k)} E_{\tilde{\mathbf{u}}} \sum_{t=0}^{T} ||\theta^T \mathbf{x}_t - \mathbf{u}_t||_2^2$$
$$\frac{1}{2} E_{p(\xi|\theta^k)} \sum_{t=0}^{T} ||\theta^T \mathbf{x}_t - \theta^{*T} \mathbf{x}_t||_2^2 + T\sigma^2$$

Given this loss it is interesting to consider how large α ,or the strong convexity term, is in this setting. To do this, we will rewrite it in the covariance matrix form with respect to the data. Note that we will drop the constant σ^2, for simplicity.

$$\begin{aligned}
&\frac{1}{2} \sum_{t=0}^{T} E_{p(\mathbf{x}_t|\theta^k)} ||\theta^T \mathbf{x}_t - \theta^{*T} \mathbf{x}_t||_2^2 \\
&= \frac{1}{2} \sum_{t=0}^{T} E_{p(\mathbf{x}_t|\theta^k)} (\theta^T - \theta^*)^T \mathbf{x}_t \mathbf{x}_t^T (\theta^T - \theta^*) \\
&= \frac{1}{2} \sum_{t=0}^{T} (\theta^T - \theta^*)^T E_{p(\mathbf{x}_t|\theta^k)} \mathbf{x}_t \mathbf{x}_t^T (\theta^T - \theta^*) \\
&= \frac{1}{2} \sum_{t=0}^{T} (\theta^T - \theta^*)^T \Sigma_{\mathbf{x}_t,k} (\theta^T - \theta^*)
\end{aligned}$$

Whats interesting about this derivation is that the strong-convexity and smoothness of the function are now coupled to the covariance matrix of the data at each time-step, $\Sigma_{\mathbf{x}_t,k}$. Specifically, the strong convexity term corresponds to the minimum eigenvalue $\alpha = \lambda_{\min}(\Sigma_{\mathbf{x}_t,k})$.

The implication of this is that convergence in dynamic-regret may be entirely dependent on the environment of the world for this linear regression formulation, which is troubling for practitioners. One way to remedy this though is to restrict the expressiveness of the function class via regularization. We can modify the loss function with a ridge regression penalty,

$$f_n = \frac{1}{2} E_{p(\xi|\psi)} E_{\tilde{\mathbf{u}}} \sum_{t=0}^{T} ||\theta^T \mathbf{x}_t - \mathbf{u}_t||_2^2 + \alpha_1 ||\theta||_2^2.$$

The strong convexity of this new loss function is now $\alpha = \lambda_{\min}(\Sigma_{\mathbf{x}_t,k}) + \alpha_1$, which suggests the strong convexity can be increased by setting α_1 to be large, which restricts the expressiveness of the function class. Thus, if a challenging setting is presented, stability can be achieved by restricting the function class for on-policy methods.

Note, though if the On-Policy algorithm was to converge before regularization was added than regularization may prevent arriving at the best possible solution in the original function class. A more detailed analysis of this incurred penalty for Follow-the-Leader based On-Policy algorithms can be found in (Cheng & Boots, 2018).

Chapter 5

Evaluation with Human Supervisors

In the previous chapter, we theoretically consider the trade-offs between On and Off-Policy Sampling and the effects of noise injection. While, our analysis can provide insight into the performance of these algorithms. It makes a strong assumption that the supervisor will be able to provide the same control signal regardless of the chosen sampling distribution.

In practice, the choice of sampling distribution can have a strong effect on the supervisor because it can change the way they provide feedback to the robot. To understand how this effect's performance in practice; we perform the following studies. First, we test the difference between pure Off and On-Policy sampling, where the user either provides demonstrative feedback or corrective. After that, we will study the effect of injecting artificial noise into the supervisor's control stream with Off-Policy methods.

On-Policy Vs. Off-Policy

To understand the choice between On and Off-Policy sampling, we perform a pilot user study on a real robot to test performance in practice. Participants teach the robot to perform a singulation task (i.e., separate an object from its neighbors), illustrated in Fig. 5.1. A successful singulation means at least one object has its center located 10 cm or more from all other object centers.

The robot has a two-dimensional control space, $\mathcal{U}$, that consists of base rotation and arm extension. The state space of the environment, $\mathcal{X}$, is captured by an overhead Logitech C270 camera, which is positioned to capture the workspace that contains all cluttered objects and the robot arm. The objects are red extruded polygons with an an average 4" diameter and 3" in height. They are made of

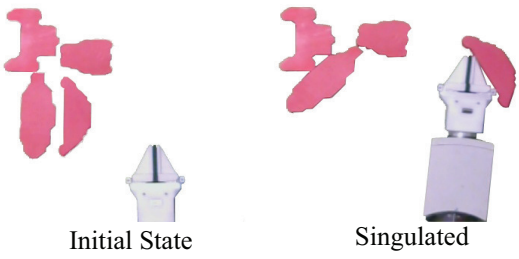


Figure 5.1: Left: An example initial state observed by the robot. The initial state can vary the relative position of the objects and pose of the pile. Right: A human is asked to singulate the object. Singulation is to have the robot learn to push one object away from its neighbors. A successful singulation means at least one object has its center located 10 cm or more from all other object centers.

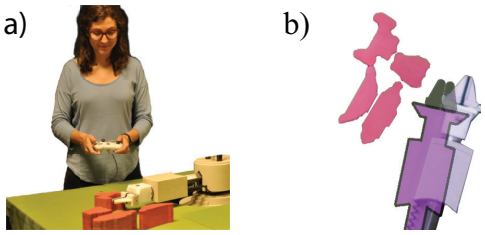


Figure 5.2: Two ways to provide feedback to the robot. a) In Off Policy sampling, the human teleoperates the robot and performs the desired task. For the singulation task, the human supervisor used an Xbox Controller. b) In On Policy sampling, the human observes a video of the robot's policy executing and applies retroactive feedback detailing what the robot should have done. In the image shown, the person is telling the robot to go backward towards the cluster.

Medium Density Fiberboard, which has a uniform density. The robot's policy is a deep neural network with the architecture from (Laskey, Lee, et al., 2016a). The network is trained using TensorFlow (*Tensor Flow*, n.d.) on a Tesla K40 GPU.

The robot is moved via positional control implemented with PID. Similar to (Laskey, Staszak, et al., 2016), the control space $\mathcal{U}$ consists of bounded changes in rotation and translation. The control signals for each degree of freedom are continuous values with the following ranges: base rotation, $[-1.5°, 1.5°]$, arm extension $[-1cm, 1cm]$.

During training and testing the initial state distribution, $p(\mathbf{x}_0)$ consisted of sampling the translation of the cluster from a multivariate isotropic Gaussian with variance of 20cm and the rotation was selected uniformly from the range $[-15°, 15°]$. The relative position of the 4 objects are chosen randomly. To help a human operator place objects in the correct pose, we used a virtual overlay on the webcam feed.

We selected 10 UC Berkeley students as human subjects. The subjects were familiar with robotics, but not the learning algorithms being assessed. Each par-

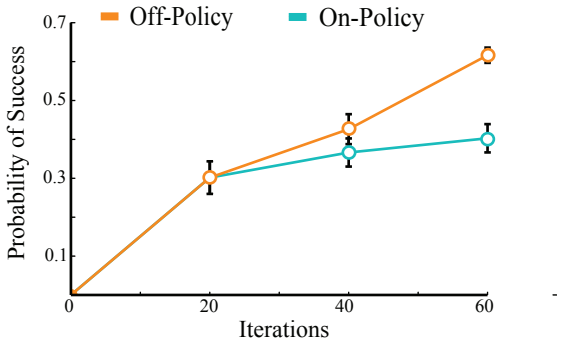


Figure 5.3: Average success at the singulation task over the 10 human subjects as a function of number of demonstrations. Each policy is evaluated 30 times on the a held out set of test configurations. The first 20 rollouts are from the supervisor rolling out there policy and the next 40 are collected via retro-active feedback for On Policy and tele-operated demonstrations for Off Policy. Off Policy LfD shows a 20% improvement in success at the end. The error bars shown are standard error on the mean.

ticipant first watched a trained robot perform the task successfully. Then they practiced providing feedback with On Policy sampling for 5 demonstrations and Off Policy sampling for 5 demonstrations. Next, each subject performed the first 20 demonstrations via Off Policy sampling, then performed 40 Off Policy demonstrations and 40 On Policy demonstrations in a counter-balanced order. In On Policy, we chose $K = 2$ iterations of 20 demonstrations each. The experiment took 2 hours per person on average.

In Off Policy, we asked participants to provide 60 demonstrations to the robot using an Xbox Controller, as shown in Fig. 5.2a. In On Policy, participants provided 20 initial demonstrations via the Xbox Controller, and then provided retroactive feedback for $K = 2$ iterations of 20 demonstrations each.

Retroactive feedback was provided through a labeling interface similar to our previous work (Laskey, Lee, et al., 2016a) illustrated in Fig 5.2b. In this interface, we showed a video at half speed of the robot's rollout to the participant. The supervisor then usse a mouse to provide feedback in the form of translation and rotation. To help the supervisor provide feedback, a virtual overlay is dislayed so that they can visualize the magnitude of their given control. A video that illustrates this setup and describes the different sampling approaches can be found at `https://berkeleyautomation.github.io/lfd_icra2017/`.

In Fig. 5.3 , we show the average performance of the policies trained with On Policy and Off Policy LfD. Each policy is evaluated on a holdout set of 30 initial states sampled from the same distribution as training. The policies learned with On Policy have approximately 40% probability of success versus 60% for Off Policy. This suggests that Off Policy may outperform On Policy when supervision is provided through actual human demonstrations.

To better understand why Off-Policy methods outperformed On-Policy, we hy-

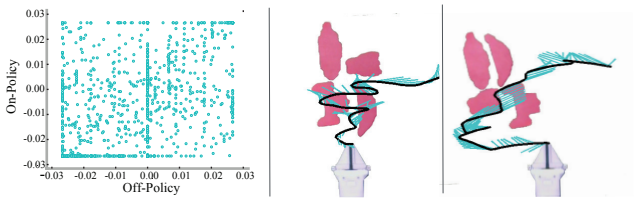


Figure 5.4: Results from the post analysis examining how well retroactive feedback matched teleoperation. The scatter plot shows the normalized angle of the control applied for both Off Policy (teleoperation) and On Policy (retroactive). The large dispersion in the graph indicates that the five participants had a difficult time matching their retroactive and teleoperated controls. Two example trajectories are also shown. The black line indicates the path from teleoperation and the teal line is the direction and scaled magnitude of the feedback given. If they matched perfectly, the teal line would be tangent to the path.

pothesized that the participants could have had trouble providing retroactive feedback consistent with their true policy. To test this, we asked 5 of the participants to provide 5 demonstrations via teleoperation. We then asked them to match their controls via the On Policy labeling interface.

We measured the correlation between the controls applied via retroactive feedback and teleo-peration. When calculated over all the participants and trajectories, the Pearson Correlation Coefficient in rotation and translation was 0.60 and 0.22, respectively. A smaller correlation coefficient suggests that it is harder for people to match their control in tele-operation.

In Fig. 5.4, we plot the control angle from Off Policy sampling versus the control angle from On Policy sampling, showing that the two are not correlated. We also show two trajectories that a participant tele-operated with their retroactive labels overlaid, both of which suggest disagreement between the tele-operated and retroactive controls. Overall, our analysis suggests that On Policy sampling can lose the intent of the human supervisor.

The intuition behind this is that when humans provide a demonstration they adjust their control signal similar to a feedback controller, where at each timestep the control is adjusted based on the observed effect. However, in the corrective feedback required by On-Policy sampling, there is correlation between the observed state and the provided control. Thus, the magnitude may be significantly off, but the supervisor may be unaware.

Behavior Cloning Vs. DART

In the previous study, the Off-Policy method, Behavior Cloning, was shown to be superior to On-Policy. However, Behavior Cloning by itself has been reported to be a sub-optimal algorithm in practice (Pomerleau, 1989). In this experiment, we will study if noise injection can improve upon the performance of Behavior Cloning with human supervisors. We will use the DART algorithm to set the level of noise.

We will these algorithms with human supervisors in a grasping in clutter task on a Toyota HSR robot. The goal of the task is for the robot to retrieve a goal object from a cupboard. The task is challenging because the goal object is occluded by obstacle objects and the robot must reason about how to clear a path based on observations taken from an eye-in-hand perspective. The objects are 6 common household food items, which consist of boxes and bottles with varying textures and mass distributions. The target object is fixed to always be a mustard bottle. The robot, task and image viewpoint are shown in Fig. 5.5. See supplement material for additional information on the task.

We use 4 supervisors who have robotics experience but not specifically in the field of Imitation Learning and compare Behavior Cloning and DART. When performing the study, we first collect $N = 10$ demonstrations with Behavior Cloning (i.e. no noise) and then in a counter-balanced ordering collect $N = 30$ more demonstrations with each technique. Our experiment was within-subject, have every supervisor perform all three methods.

We only updated the noise parameter after the first 10 demonstrations (i.e. $K = 2$). The final robot policy was then trained on the total of 40 demonstrations. We consider two different choices of α: $\alpha = 3\text{tr}(\hat{\Sigma}_1)$ and $\alpha = 3\text{tr}(\hat{\Sigma}_1)$. These choices correspond to the intuition that with small datasets in high dimensional image space the robot will have significant shift from the current loss on the supervisor's distribution. Through the rest of the paper, we will write our choices as $\alpha = 3$ and $\alpha = 6$ for brevity in notation.

During policy evaluation, we measured success as 1) the robot is able to identify the goal object and 2) there is a clear path between its gripper and the object. Once these conditions are identified an open-loop motion plan is generated to execute a grasp around the target object. Once a path is cleared a human supervisor tells the robot to execute the open-loop motion plan towards a fix position that the goal object is at. The human supervisor is used to ensure that the success criteria is not prone to error in the system, but the supervisor is blinded to which policy is being evaluated to prevent biases.

To identify the pose of the target object, we use the Faster R-CNN trained on the PASCAL VOC 2012 dataset, which has a bottle object category and is fine-tuned on 100 labeled images to detect the mustard bottle (Ren, He, Girshick, & Sun, 2015).

In Fig. 5.5, we report the average performance, over 20 trials, of the three techniques with each trial occurring on a different initial state sample. DART with $\alpha = 3$ performs the best out of the three techniques, with a 79% success rate over traditional Behavior Cloning's 49% success rate. Interestingly, DART with $\alpha = 6$ performs better than Behavior Cloning, but only has a 72% success rate. This may suggest this level of noise was potentially too high for the human supervisor. A video of the learned policy and the robot setup can be found here: `https://youtu.be/LfMD69llesg`.

In order to check that both methods had equal amounts of data. In Fig. 5.6,

Figure 5.5: Left: Experimental setup for the grasping in clutter task. A Toyota HSR robot uses a head-mounted RGBD camera and its arm to push obstacle objects out of the way to reach the goal object, a mustard bottle. The robot's policy for pushing objects away uses a CNN trained on images taken from the robot's Primesense camera, an example image from the robot's view point is shown in the orange box. Right: the Success Rate for Behavior Cloning, DART($\alpha = 3$) and DART($\alpha = 6$). DART($\alpha = 3$) achieves the largest success rate.

we report the average dataset collected per demonstrations and the generalization error. Both measurements contain no significant differences, which suggests dataset size and the quality of data were not affected across methods.

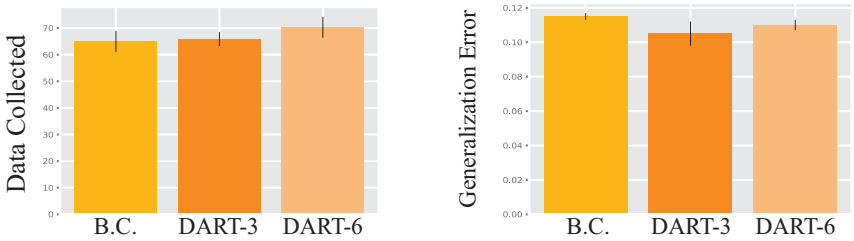


Figure 5.6: (Left) The amount of data collected on average with each method. On average the data shows no significant variations between the method, which suggests all methods were given roughly similar amounts of data during the experiment. (Right) Generalization error for the tasks, which was computed with 2 fold cross-validation. Similar generalization error suggests the empirical win in performance was due to covariate shift reduction.

This study suggest that human supervisor can tolerate noise injected into the tele-operation system and it can lead to a performance win. One reason for this is that the sampling distribution is still concentrated around the supervisor's control, thus an appropriate amount of feedback is still observed during tele-operation. Additionally, noise injection may help "regularize" the supervisor's policy, because it forces them to be robust to perturbation in order to complete the task. Since, this study similar results have been shown in the navigation domain (Codevilla et al., 2017).

Chapter 6

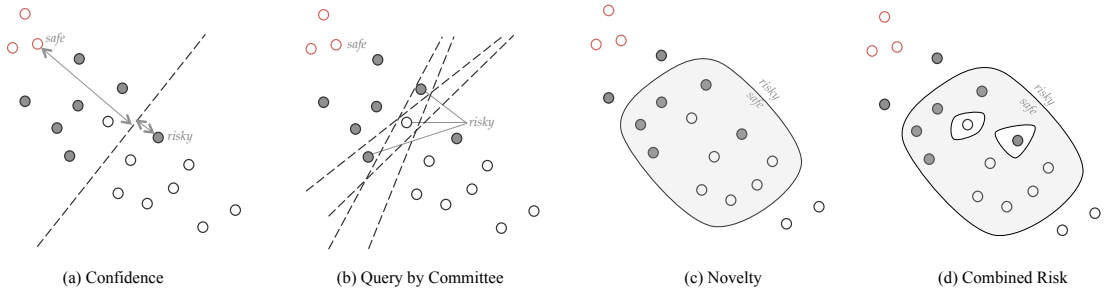
SHIV: Active Learning for On-Policy Sampling

One potential drawback with On-Policy sampling is that it requires providing corrective feedback for all states the robot visits. For human supervisors, this can be potentially quite tedious to provide feedback at all states. To reduce this burden, we can leverage active learning techniques, where the robot selects which examples would be informative. Interestingly, while active learning is a well studied field, it typically assumes the distribution over data collected is fixed during training (?, ?). In On-Policy sampling though the distribution can change substantially as the model is updated. In this chapter, we will study how this affects current active learning techniques and propose a more conservative approach that makes decisions based on the data itself, rather than the estimator. We will then study how to combine this with the common On-Policy algorithm DAgger.

Active Learning on Non-Stationary Distributions

We can formulate reducing supervisor burden in On-Policy sampling as a stream-based active learning problem (Atlas, Cohn, & Ladner, 1990; Cohn, Atlas, & Ladner, 1994). In stream based active learning, the decision of whether to query the supervisor or not is not over the entire state space (like in traditional pool-based active learning), but on states drawn one at a time from some data stream. In On-Policy sampling, this data stream is the states encountered when the robot is executing the current best policy.

Aside from the data stream, stream-based active learning has another ingredient: a query selection method, deciding whether or not to query. Typical query selection methods are estimator-centric, evaluating risk using the estimator, e.g. distance from the classification hyperplane (Tong & Koller, 2002), as in Fig. 6.1(a); or query



(a) Confidence (b) Query by Committee (c) Novelty (d) Combined Risk

Figure 6.1: A comparison of different query selection strategies for active learning on a non-stationary distribution. The shaded and empty circles are training data from two classes, 1 and 0 respectively. The red empty circles are samples from a new distribution (produced by executing the learned policy), and belong to class 0. Typical strategies classify states close the decision boundary as risky(a), or for which a set of estimators disagree (b). Neither of these apply to our new samples in red. In contrast, we use a strategy that is amenable to non-stationary distributions by classifying novel states as safe (i.e not risky) (c) and states in historically mislabeled regions.

by committee (Breiman, 1996), which uses a committee of hypothesized estimators that are trained on different subsets of the training data. Risk in query by committee is based on the level of agreement or disagreement among these hypotheses, with higher levels of disagreement for a state leading to higher chances of querying that state (Fig. 6.1(b)).

Both approaches implicitly assume a stationary state distribution — that the new data is sampled from the same distribution as the previous training data. Although such methods have been previously proposed for On-Policy sampling (see (Chernova & Veloso, 2009; Grollman & Jenkins, n.d.) for the former and (Judah et al., 2011; Judah, Fern, & ietterich, 2012) for the latter), On-Policy sampling violates the stationary distribution assumption because each new policy induces a new distribution of states. This can have negative consequences for learning: it has been shown that when training and test distributions are different, query by committee can perform worse than randomly selecting which states to query (Burbidge, Rowland, & King, 2007).

The problem lies in being estimator-centric. The estimator is a function of the current distribution. Therefore, it no longer provides a good measure of confidence when that distribution changes. And the distribution does change when the robot is rolling out a learned policy and starts encountering further away states. Instead of relying on the estimator, our measure of confidence explicitly identifies when states are drawn from a different distribution, i.e. when they are novel. Our experiments in Section 6 suggest that our data-centric approach has significantly lower false positives in On-Policy sampling problems than estimator-centric measures.

In order to detect when data is outside the distribution, our work builds on Kim et al. (Kim & Pineau, 2013) who apply novelty detection to safe On-Policy Sampling to enable a supervisor to take over and prevent unsafe states. In their MMD-IL algorithm, risk is evaluated with Maximal Mean Discrepancy. For a single

state, this is the sum of a kernel density estimate and the variance of the dataset, which works well in low dimensional state spaces. MMD-IL also reduces supervisor burden because the supervisor only takes over in risky states, as opposed to providing labels for all new states.

In contrast, we focus on active learning for On-Policy sampling in high-dimensional state spaces. Unlike safe learning, our problem does not require the supervisor to take over. This enables us to avoid the supervisor biasing the states sampled (which can in practice lead to a loss in performance (Ross et al., 2011a)), and also avoid assuming the supervisor is always available. On the other hand, our problem does require novelty detection in high dimensional state spaces, where non-parametric density estimation does not scale very well: it can require data exponential in the dimension of the state space (Nadaraya, 1964). Our results (Section 6) suggests that in our high dimensional state spaces, SHIV achieves an error rate half of that of kernel density estimate-bases techniques like MMD.

An alternative to kernel density estimates is to measure distance to its nearest neighbors (Knox & Ng, 1998). However, this approach was shown to be susceptible to issues, since nearest neighbors incorporates only local information about the data. For example, a group of outliers can be close together, but significantly far from the majority of the data and nearest neighbors would mark them as not novel (Hodge & Austin, 2004).

Our approach is based on the One Class SVM proposed by Scholköpf et al., which estimates a particular quantile level set for the training data by solving a convex quadratic program to the find support vectors (Schölkopf, Platt, Shawe-Taylor, Smola, & Williamson, 2001). The method has been theoretically shown to approximate the quantile levelset of a density estimate asymptotically for correctly chosen bandwidth settings and in the case of a normalized Gaussian kernel function (Vert & Vert, 2006). In (Liu, Hua, & Smith, 2014), the One Class SVM has furthermore been used for novelty detection in high-dimensional image data.

Risky and Safe States

Providing correct control inputs for (or "labeling") all states encountered at each iteration can impose a large burden on the supervisor. Instead of asking the supervisor for labels at all visited states, SHIV uses a measure of risk to actively decide whether a label is necessary.

In contrast to the standard measure risk, we define a state as "risky" for 2 reasons: 1) it lies in an area with a low density of previously trained states, which can cause the current policy to mis-predict the supervisor and incur high surrogate loss (Tokdar & Kass, 2010), or 2) the surrogate loss, or training error, of the current policy at the state is high, so that the state is unlikely to model the supervisor's control inputs correctly. States that are not classified as "risky" are deemed "safe"

.Our definition of risk can be visualized in Fig. 6.1(d).

The amount of data needed to estimate the density, scales exponentially in the dimension of the state space (Nadaraya, 1964). Thus, to evaluate risk in high-dimensional state spaces, such as the HOG features in our driving simulator, Fig. 6.2(a), we use a modified version of the technique known as the One Class SVM that estimates a regularized boundary of a user defined quantile on the training data in $\mathcal{X}$ (Schölkopf et al., 2001).

We consider the problem of estimating the quantile level-sets of a distribution P on a set $\mathcal{X}$ by means of a finite set of independent and identically distributed samples $\mathbf{x}_1, ..., \mathbf{x}_n \in \mathcal{X}$. In most general terms, the quantile function for P and subject to a class of measurable subsets $\mathcal{G}$ of $\mathcal{X}$ is defined by

$$U(\gamma) = \inf\{\lambda(G) : P(G) \geqslant \gamma, G \in \mathcal{G}\}\ 0 < \gamma \leqslant 1 \tag{5}$$

$\lambda : \mathcal{G} \to \mathbb{R}$ above denotes a volume measure. Suppose furthermore that $G : [0, 1] \to \mathcal{G}$ assigns a set $G(\gamma) \in \mathcal{G}$ that attains the infinum measure (i.e. volume) for each $\gamma \in [0, 1]$ (this set is in general not necessarily unique). $G(\gamma)$ denotes a set of minimum measure $G \in \mathcal{G}$ with $P(G(\gamma)) \geqslant \gamma$. Note in particular that $G(1)$ is the support of the density p corresponding to P, if p exists.

To handle distributions defined on high-dimensional spaces $\mathcal{X}$, work by Scholköpf et al. represents the class $\mathcal{G}$ via a kernel k as the set of half-spaces in the support vector (SV) feature space (Schölkopf et al., 2001). By minimizing a support vector regularizer controlling the smoothness of the estimated level set function this work derives an approximation of the quantile function described in Eq. 5. This approach can be thought of as employing $\lambda(G) = ||w||^2$, where $G_w = \{x : f_w(x) \geqslant \rho\}$, $f_w(\mathbf{x}) = \sum_i w_i k(\mathbf{x}_i, \mathbf{x})$ and (w, ρ) denote the weight vector and offset parameterizing a hyperplane in the feature space associated with a Gaussian kernel $k(\mathbf{x}_0, \mathbf{x}_1) = e^{-||\mathbf{x}_0 - \mathbf{x}_1||^2/2\sigma^2}$.

Let $\Phi : \mathcal{X} \to \mathcal{F}$ denote the feature map corresponding to our exponential kernel, mapping the observation space $\mathcal{X}$ into a Hilbert space $(\mathcal{F}, \langle , \rangle)$ such that $k(\mathbf{x}, \mathbf{x}') = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle$.

The One Class SVM proposed by (Schölkopf et al., 2001) determines a hyperplane in feature space $\mathcal{F}$ maximally separating the input data from the origin:

$$\underset{w \in \mathcal{F}, \xi \in \mathbb{R}, \rho \in \mathbb{R}}{\text{minimize}} \frac{1}{2}||w||^2 + \frac{1}{\nu n}\sum_i^n \xi_i - \rho \tag{6}$$

$$\text{s.t}\, \langle w, \Phi(x_i) \rangle \geqslant \rho - \xi_i,\ \xi_i \geqslant 0.$$

Here, the parameter ν controls the penalty or 'slack term' and is equivalent to γ (Vert & Vert, 2006) in the quantile definition, Eq. 5, as the number of samples increases. The decision function, determining point membership in the approximate quantile levelset is given by $g(\mathbf{x}) = \text{sgn}(\langle w, \Phi(x) \rangle - \rho)$. Here, for $x \in \mathcal{X}$, $g(x) = 0$ if x lies on the quantile levelset, $g(x) = 1$ if x is strictly in the interior of the quantile super-levelset and $g(x) = -1$ if x lies strictly in the quantile sub-levelset.

The dual form of the optimization yields a Quadratic Program that has worst case computational complexity of $O(n^3)$. However, Schölkopf et al. developed an improved optimization method that has empirically been shown to scale quadratically (Schölkopf et al., 2001). In the dual, the decision function is given by $g(\mathbf{x}) = \text{sgn}(\sum_{i=1}^{N} \alpha_i k(\mathbf{x}_i, \mathbf{x}) - \rho)$ where α_i corresponds to the dual variables. The novelty detection method can be visualized in Fig. 6.1(c). However, even when sufficient data is available, the associated control inputs may be inconsistent or noisy and a resulting policy optimizing Eq. 9 may still incur a large surrogate loss. To account for this, we propose a modification to the One Class SVM:

$$y_i = \begin{cases} 1 & : l(\pi_\theta(\mathbf{x}_i), \mathbf{u}_i) \leqslant \varepsilon \\ -1 & : l(\pi_\theta(\mathbf{x}_i), \mathbf{u}_i) > \varepsilon \end{cases} \tag{7}$$

Where, in the case when l denotes discrete $0-1$ loss, we set $\varepsilon = 0$, while in the continuous L_2 loss case, ε is a user defined threshold specifying allowable surrogate loss. We use y_i to modify the One Class SVM decision function as follows:

We divide up our data in to two sets those correctly classified: $\mathcal{D}_s = \{\{\mathbf{x}_i, \mathbf{u}_i\} \in \mathcal{D}_k, y_i = 1\}$ and those states incorrectly classified: $\mathcal{D}_r = \{\{\mathbf{x}_i, \mathbf{u}_i\} \in \mathcal{D}_k, y_i = -1\}$ A separate One-Class SVM is then trained on each set of states, (D_s and D_r) and providing measures of the level sets, g_s and g_r. Specified by parameters (ν, σ) and (ν_r, σ_r), respectively.

We then define the overall decision function as:

$$g_\sigma(\mathbf{x}) = \begin{cases} 0 & : g_s(\mathbf{x}) == 1 \text{ and } g_r(\mathbf{x}) == -1 \\ -1 & : \text{otherwise} \end{cases} \tag{8}$$

points are deemed risky if $g_\sigma(\mathbf{x}) \neq 0$. Practically, this modification corresponds to 'carving out holes' in the estimated quantile super-levelset such that neighborhoods around states with $y_i = -1$ are excluded from the super-levelset. An illustration of this can be seen in Fig. 6.1(d).

The decision function parametrization consists of the kernel bandwidth σ in g_s. We treat σ as a "risk sensitivity" parameter (and study its implications in Section 6). For two reasons: 1)The expected number of examples, after a policy roll out, the supervisor can be asked is $T * \int_{\mathbf{x}} \mathbf{1}(g_\sigma(\mathbf{x}) == 0) p(\mathbf{x}|\theta) d\mathbf{x}$. Thus, smaller σ corresponds to asking for more examples. 2) A relation exists between how smooth the supervisor's policy, $\tilde{\pi}$ and how many examples are needed to learn it. Thus, a large σ can be dangerous for policies with sharp variation because it will treat points as safe that are really risky.

SHIV:Svm-based reduction in Human InterVention

We can now combine our query selection mechanism with the On-Policy method, DAgger. Both SHIV and DAgger (Ross et al., 2011a) solve the minimization over

the aggregate data-set, described in Chapter 2, by iterating two steps: 1) compute a θ using the training data $\mathcal{D}$ thus far, and 2) execute the policy induced by the current θ, and ask for labels for the encountered states. However, instead of querying the supervisor for every new state, SHIV actively decides whether the state is risky enough to warrant a query.

Step 1 The first step of any iteration k is to compute a θ_k that minimizes surrogate loss on the current dataset $\mathcal{D}_k = \{(x_i, u_i)|i \in \{1, \ldots, N\}\}$ of demonstrated state-control pairs (initially just the set $\mathcal{D}$ of initial trajectory demonstrations):

$$\theta_k = \arg\min_{\theta} \sum_{n=1}^{N} l(\pi_\theta(\mathbf{x}_n), \mathbf{u}_n). \tag{9}$$

This sub-problem is a supervised learning problem, solvable by estimators like a support vector machine or a neural net [1]. Performance can vary though with the selection of a the estimator (Schölkopf & Smola, 2002)

Step 2 The second step SHIV and DAgger rolls out their policies, π_{θ_k}, to sample states that are likely under $p(\mathbf{x}|\theta_k)$.

What happens next, however, differs between SHIV and DAgger. For every state visited, DAgger requests the supervisor to provide the appropriate control/label. Formally, for a given sampled trajectory $\hat{\tau} = (\mathbf{x}_0, \mathbf{u}_0, ..., \mathbf{x}_T, \mathbf{u}_T)$, the supervisor provides labels $\tilde{\mathbf{u}}_t$, where $\tilde{\mathbf{u}}_t \sim \tilde{\pi}(\mathbf{x}_t) + \epsilon$, where ϵ is a small zero mean noise term, for $t \in \{0, \ldots, T\}$. The states and labeled controls are then aggregated into the next data set of demonstrations $\mathcal{D}_{k+1}$:

$$D_{k+1} = \mathcal{D}_k \cup \{(\mathbf{x}_t, \tilde{\mathbf{u}}_t) \| t \in \{0, \ldots, T\}\}$$

SHIV only asks for supervision on states for which are risky, or $g_\sigma(\mathbf{x}) \neq 0$:

$$D_{k+1} = \mathcal{D}_k \cup \{(\mathbf{x}_t, \tilde{\mathbf{u}}_t) \| t \in \{0, \ldots, T\}, g(\mathbf{x}_t) = -1\}$$

Steps 1 and 2 are repeated for K iterations or until the robot has achieved sufficient performance on the task[2].

[1]To handle the fact that the supervisor's policy can be noisy, a zero-mean noise term ϵ can be considered as present in the policy's output. A regularization technique in the optimization is used to control the smoothness of the function that is fit to the sampled data. In practice this regularization corresponds to a penalty term on either the L2 norm on the weights for regression based techniques or the slack coefficient for support vector machines (Schölkopf & Smola, 2002).

[2]In the original DAgger the policy rolled out was stochastically mixed with the supervisor, thus with probability β it would either take the supervisor's action or the robots. The use of this stochastically mix policy was for theoretical analysis. In practice, it is recommended to set $\beta = 0$ to avoid biasing the sampling (Guo et al., 2014; Ross et al., 2011a)

Experiments

All experiments were run on a machine with OS X with a 2.7 GHz Intel core i7 processor and 16 GB 1600 MHz memory in Python 2.7. The policies, π_θ are trained using Scikit-Learn (Pedregosa et al., 2011). Our modified One Class SVM contains two different ν parameters, ν and ν_r. We set $\nu = 0.1$ and $\nu_r = 10^{-3}$ for all experiments. We tuned σ and σ_r by performing a grid search over different values on the surrogate loss for a single trial of SHIV for 3 iterations.

We compare SHIV and DAgger in two domains: a driving simulator and push-grasping in clutter with a 4DOF arm. Each domain test different aspects of our algorithm: the driving simulator has a high dimensional visual state space, grasping in clutter has a human demonstrator provide the labels and is a challenging manipulation problem, surgical needle insertion uses data from a real robot.

We then compare our query selection method with those typically used in active learning. We show that for a non-stationary state distribution like ours, the notion of risk based on novelty and misclassified regions performs better than confidence, query-by-committee based methods, Maximum Mean Discrepancy, and the One Class SVM without the carving out misclassified holes modification. We continue with a sensitivity analysis, which suggests that the performance of SHIV is robust to the choice of how risky the robot is allowed to be (the σ parameter from Eq. 8).

Comparing SHIV with DAgger

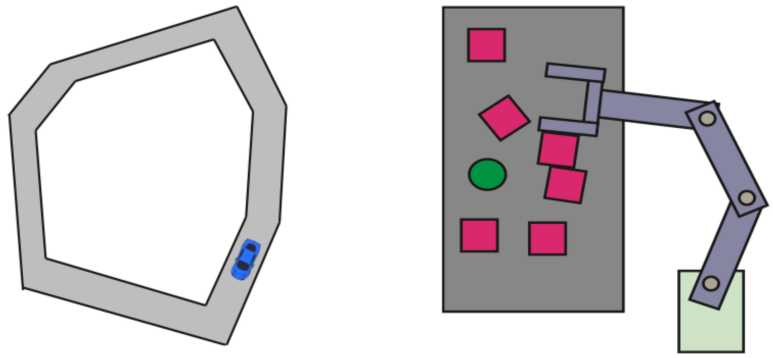
We compare SHIV with DAgger on three domains: a driving simulator and push-grasping in clutter. We hypothesize that SHIV achieves the same performance as DAgger, but by asking for fewer examples and thus reducing supervisor burden.

The independent variables are two variables. First, we manipulate the On-Policy algorithm we use: SHIV vs. DAgger. Second, we manipulate how many examples the algorithms are allowed to ask for: we set a budget of labeled states, and analyze performance as this budget increases.

For each algorithm and budget of states labeled, we measure the normalized performance (e.g. 1 corresponds to matching the supervisor's performance on a task). Performance is a domain specific term, such as number of times the car crashes in the driving simulator.

Driving Simulator

Our first domain is a common benchmark in Imitation Learning: learning a policy for a car to drive around a track (Argall et al., 2009; Ross & Bagnell, 2010). We implemented a driving simulator where the car must follow a polygonal track. We generate a polygonal track by repeatedly sampling from a Gaussian with mean that is the center of the video game workspace centered in the middle of the workspace, and



(a) Driving Simulator (b) Grasping in Clutter in Box2D

Figure 6.2: a) a driving simulator where the goal is to learn a controller on HOG features extracted from synthetic images to keep a car on a polygonal track; b) push-grasping in clutter in a physics simulator, where the goal is to learn a controller from human demonstrations to grasp the green object while not letting the red square objects fall off the gray boundary representing a table.

computing the convex hull of the sampled points. Then a convex hull is computed on the sampled points. This produces tracks composed of five to seven edges, an example is shown in Fig. 6.2(a). If the car accidentally leaves the track, it is placed back on the center of the track at a nearby position. The car's steering is $\mathcal{U} = \{-15^\circ, 0, 15^\circ\}$. A control input instantly changes the angle at a unit speed. The internal state space of the car is given by the xy-coordinates and the angle it is facing. In our experiments, the supervisor is provided by an algorithm that uses state space search through the driving simulator to plan the next control. The supervisor is only allowed to search a finite amount a time ahead in the game and is prone to error, thus on averages crashes 5.4 times per lap.

The supervisor drives around the track twice. We collect raw images of the simulation from a 2D bird's eye view and use Gaussian Pyramids to down-sample the images to 125×125 RGB pixels and then extract Histogram of Oriented Gradients (HOG) features using OpenCV. This results in a 27926 dimensional state space description. For both DAgger and , we use a Linear Support Vector Machine (SVM) to parameterize allowable policies π_θ, with $\gamma = 0.01$ as a regularization term on the slack variables, which was set via cross validation on the initial training examples. We set SHIV's parameters of the exponential kernel's bandwidth as $\sigma = 200$ and $\sigma_r = 200$, which are set via the grid search defined above.

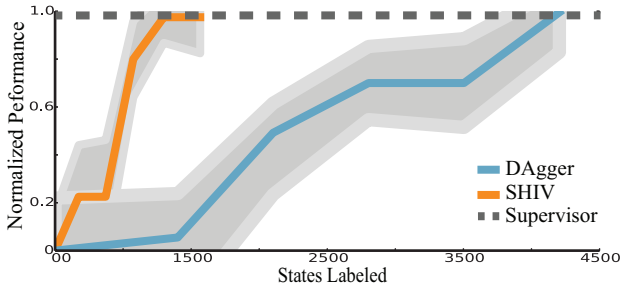


Figure 6.3: We compare normalized performance (i.e. matching the performance of the supervisor) for the Driving Simulator , performance is defined as the number of times the car left the track versus the number of queries made to the supervisor. We plot the performance of DAgger and SHIV, the teal and orange lines. We also plot the performance of our supervisor, who makes non-optimal decisions and has an average performance of 5.4 crashes. Initial results, which are run for 6 iterations each and are averaged over 40 levels, shown in Fig. 6.3 suggest an 71% reduction in the number of queries needed for SHIV compared to DAgger. Error bars in grey measure the standard error on the mean

In Fig. 6.3, we visualize the performance of DAgger and SHIV. We run 6 iterations, which is a completion of both Step 1 and Step 2 described in Section 6 of each algorithm over 40 different randomized polygonal tracks. Figure 6.3 presents averaged results from these experiments, suggesting a 71% reduction in the number of queries needed for SHIV compared to DAgger, in order to reach approximately the same performance as our non-optimal supervisor.

Grasping In Clutter in Box2D

We investigate having a human demonstrator control a simulated robot arm in 2D to reach a target object with out knocking other objects off a table. Grasping an object in a cluttered environment is a common task for a robot in an unstructured environment and has been considered a benchmark for robotic manipulation (Kitaev, Mordatch, Patil, & Abbeel, 2015; King, Haustein, Srinivasa, & Asfour, n.d.). The task is difficult because modeling the physics of pushing an object is non-trivial and requires knowing the shape, mass distribution and friction coefficient of all objects on the table. We are interested in learning such a policy via human demonstrations.

We used Box2D a physics simulator to model a virtual world. We simulate a 4 DOF robot arm with three main joints and a parallel jaw gripper as displayed in Fig. 6.2(b). SHIV and DAgger do not have access to the underlying dynamics of the simulator and must learn a policy from only demonstrations.

For input the human demonstrator provides controls through an XBox game controller. The right joystick was used to provide horizontal and vertical velocity inputs for the center of the end-effector which were then translated into robot arm motions by means of a Jacobian transpose controller for the 3 main joint angles. The left 'bumper' button on the joystick was used to provide a binary control signal

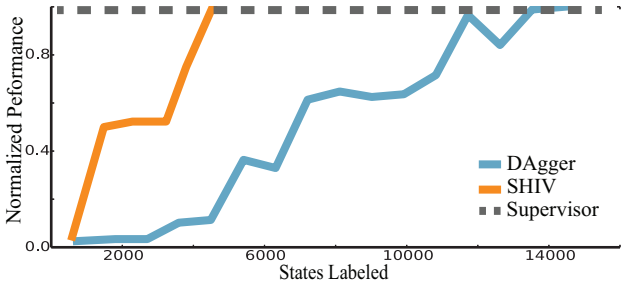


Figure 6.4: We compare normalize performance (i.e. matching the performance of the supervisor) for Grasping in Clutter domain, performance is defined as the sum of the number of objects knocked off the table plus 10 times the binary value indicating if the object is grasped or not. Initial results, which are averaged over 8 different trials suggest a 64% reduction in the number of queries needed for SHIV compared to DAgger,

to close the parallel jaw gripper. The control inputs are hence modeled by the set $\mathcal{U} = \{[-1, 1], [-1, 1], \{0, 1\}\}$.

A state $\mathbf{x} \in \mathcal{X}$ consisted of the 3 dimensional pose of the six objects on the table (translation and rotation), the 3 joint angles of the arm and a scalar value in the range $[0, 1]$ that measured the position of the gripper, 1 being fully closed and 0 being opened. For our representation of π_θ, we used kernelized ridge regression with the radial basis function as the kernel with the default Sci-Kit learn parameters. We defined performance as the sum of the number of objects knocked off the table plus 10 times the binary value indicating if the object is grasped or not. The order of magnitude difference in cost for grasping the object is to place emphasize on that portion of the task. The bandwidth parameters for SHIV were set to $\sigma_r = 5$ and $\sigma = 6$. For the ϵ term in the our risk method, we used the median in regression error which was the L2 distance between the predicted control and the true supervisor's control.

In our experiment, a human demonstrator provided one demonstration and then iterated until the performance was zero during the policy roll out. At each iteration, we sampled the pose of the target object from an isotropic Gaussian with a standard deviation that is 3% of the width of the table.

In Fig. 6.4 , we show the normalized performance averaged over 8 rounds for SHIV and DAgger. Supporting our hypothesis, our results suggests that SHIV can achieve the same performance with a 64% reduction in the number of examples needed.

SHIV Analysis

Comparison to active learning approaches.

We compare five active learning methods. We compare our combined notion of risk (Fig. 6.1(d)), with risk based on novelty alone (Fig. 6.1(c)) in order to test whether carving out regions that have been mis-classified previously is valuable.

We also compare against two baselines typically used in active learning. The first is confidence based on distance from the classification hyperplane (Tong & Koller, 2002) (Fig. 6.1(a)). We set the threshold distance to the average distance from the hyperplane for the mis-classified points in $\mathcal{D}_0$, which consisted of two demonstrations from our solver.

The second baseline is Query By Committee (Fig. 6.1(d)), which has a committee of different hypothesis estimators and points are marked risky if the committee disagrees. Our committee which was trained via bagging (Breiman, 1996). To obtain a committee, we divided the training data into 3 overlapping subsets, each with 80% of the data. We trained a Linear SVM on each subset. If the three classifiers agreed with each other the point was determined low risk and if they disagree it was determined high risk.

We compare against another novelty detection method as well called Maximal Mean Discrepancy (MMD), which evaluates how close a point is to the distribution using kernel density estimate and adds the variance of the distribution in the dataset $\mathcal{D}$ (Kim & Pineau, 2013). We set bandwidth of the kernel as the same as our modified One Class SVM and the α decision boundary is set by sorting the MMD value of the states and picking an α such that the lowest 10% are marked as risky. This is in similar spirit to $\nu = 0.1$.

We run each query selection method over 50 different car tracks in the driving simulator domain. We measured the percentage of truly risky states, encountered during the first policy roll out, that are estimated to be safe by the active learning technique, or false negatives. The active learning techniques are trained on the initial demonstrations $\mathcal{D}_0$, which is different then the distribution being sampled from, $p(\xi|\theta^1)$, in this experiment.

Fig. 6.5 plots the performance for each query selection method, averaged over 50 tracks. We observe a significant performance improvement with methods based on novelty detection compared to confidence,query by committee and MMD. Furthermore, using the combined measure of risk performs better than relying solely on the One Class SVM.

Sensitivity Analysis to Risk Sensitivity Parameter

To analyze the sensitivity of our method we varied the risk sensitivity parameter or σ of the decision function g_σ, or Eq. 8. σ is a measure of how much risk we allow,

Risk Sensitivity (σ)	Performance	States Labeled
1	5.8	4122
50	5.7	3864
150	5.9	1859
200	6.1	1524
250	6.3	1536
350	13.2	521

Table 6.1: An analysis of the sensitivity risk sensitivity parameter,σ, of Eq. 8, or the decision function for risky or safe. SHIV was run for 6 iterations and averaged over 40 different randomized polygonal tracks. Small σ, $\sigma = 1$, corresponds to always asking for help and very large sigma, $\sigma = 350$, relates to a lot less data being used, but decreased performance.

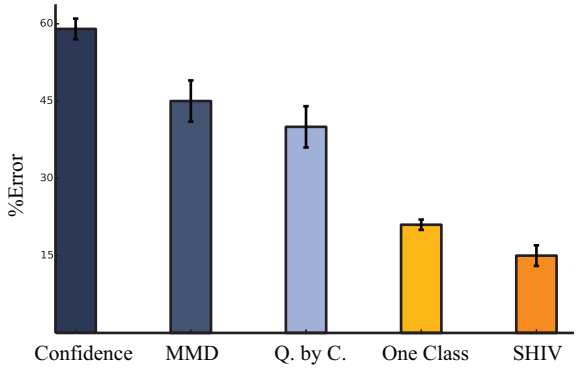


Figure 6.5: A comparison of different active learning approaches in terms of the percentage of risky states that are estimated to be safe by the active learning technique during the first policy roll out. We compare against a confidence estimate of distance to hyperplane,Maximal Mean Discrepancy, query by committee for 3 hypothesis classifiers, the One Class SVM, and our modified One Class SVM, marked as SHIV. Results are averaged over 50 tracks and the policy π_θ is represented as a Linear SVM. Error bars measure the standard error on the mean.

with smaller σs leading to more risk-adverse behavior. SHIV was ran for 6 iterations and averaged over 40 different randomized polygonal tracks.

For each σ, we measure the final cost performance of the policy π_θ after 6 iterations and the number of examples SHIV requested from the supervisor.

As shown in Table 6.1, small σ, $\sigma = 1$, corresponds to always asking for help (many states labeled) and very large sigma, $\sigma = 350$, relates to less data being used, but worse performance. However, σ values between 150 and 250 all achieve similarly good performance, suggesting that SHIV is robust to the choice of a particular σ.

Conclusion

On-Policy sampling has been successful on an array of tasks (Guo et al., 2014; Ross et al., 2011a), however it can place significant burden on a supervisor. Our algorithm, SHIV, implements stream-based active learning with a query selection method that evaluates risk in a manner tailored to non-stationary and high-dimensional state distributions. We empirically evaluated our method on three different domains: a driving simulator, grasping in clutter and surgical needle insertion. Results suggests up to a 70% reduction in the number of queries to the supervisor.

SHIV has several limitations. For instance, the selection of the value of σ can affect performance and a poor choice could result in the algorithm becoming offline imitation learning, as shown in Table 6.1. Future work will look at better understanding the relationship of σ to the function π_θ and how to automatically select it.

Furthermore, to update the decision function g_σ requires solving a quadratic program, which has an upper bound computationally complexity of $O(n^3)$ in the number of data points. This can effect the ability of using our modified One Class SVM for larger problems. While, Scholköpf et al. provides an efficient optimization that empirically scale quadratically (Schölkopf et al., 2001) for very large datasets this can still be burdensome. Future work will look at using techniques such as random features or PCA on the kernel Gram matrix to allow for better scaling (Schölkopf & Smola, 2002).

Chapter 7

FLUIDS, A Driving Simulator for IL Benchmarking

With the success of OpenAI Gym, the field of Reinforcement Learning and Imitation Learning has seen a standardization of light-weight simulators for comparing algorithms. For robotic tasks, which are continuous action spaces, the standard benchmark has been the MuJoCo domain (Laskey, Lee, et al., 2017; Ho & Ermon, 2016; Cheng et al., 2018; Sun, Venkatraman, Gordon, Boots, & Bagnell, 2017). In MuJoCo, a robot composed of a series of actuated linkages is trained to move forward in the desired direction. While MuJoCo is useful for testing Reinforcement Learning algorithms, it has several potential short-comings when applied to the Imitation Learning setting. In this section, we will detail these shortcomings and then present a new prototype simulator to help address them.

Concerns with Current Benchmarks

Testing Generalization

Benchmarks, like MuJoCo, typically measure the number of trials needed to perform an instance of the task well. These benchmarks are useful for RL algorithms because it tests how effective an agent is at exploring the state space to find a reasonable policy. However, in Imitation Learning the goal is not to test exploration but rather how well the policy generalizes to unseen instances of the task.

The intuition for this is that the supervisor has already determined the correct sequence of controls to apply to perform the task. Thus the robot needs merely to recover the supervisor's policy rather than discover a new one. Most RL benchmarks though do not prioritize testing on new instances of the problem and have relatively low variance in the conditions varied during learning. The effect of this low variance

is that the benchmarks may be too easy for IL algorithms.

For example, consider the Hopper task in MuJoco, where the goal is to learn to walk forward. In Fig. **??**, we apply Behavior Cloning with a nearest neighbor policy representation. As shown, with just a single demonstration the robot can match the supervisor's performance. The fact that only one demonstration is needed suggests that the robot is shown any new states during execution, which is very unrealistic.

Figure 7.1: An example of how domains, such as Hopper, in MuJoCo do not test Generalization. The learned policy, which is a nearest neighbor representation, is trained with Behavior Cloning. As shown, with a single demonstration the policy can match the performance of the supervisor, which indicates the benchmark does not vary the task enough to test generalization.

To fix this issue with generalization, it has become common practice to heavily sub-sample the trajectories returned from the supervisor (Ho & Ermon, 2016). The sub-sampling is implemented by throwing away most of the trajectory except for observations at every k^{th} interval. While this sub-sampling can create the appearance of unseen data at test time, it is not reflective of the real type of randomness a robot is likely to encounter in practice. In most robotic tasks, the randomness of the environment comes in the form of initial state and dynamics distribution. Only testing for randomness in this periodic sub-sampling, may create potential artifacts in the benchmark that can lead to useless algorithms dominating. Thus, it is imperative to have benchmarks with the correct type of randomness naturally occurring within them.

Another type of generalization to consider is how well the methods perform *out of sample*. Out of sample testing is when the robot sees a different distribution than

what was observed in training. For example, if a self-driving car is trained at a traffic intersection with no pedestrians and then pedestrians are allowed to enter the area , this would be considered out of sample, because it was never shown any instances of pedestrians during training. Considering that in unstructured environments like the home, its not possible to show on all types of situations during data collection. It is important to understand how algorithms respond to these scenarios.

Learned Supervisors

In Imitation Learning, a supervisor is required to perform the experiments. For domains like MuJoCo, it can be hard to derive an algorithmic supervisor, which has lead to the use of learning policy to act as the supervisor. The learned supervisor is trained with a Reinforcement Learning algorithm, such as TRPO (Schulman et al., 2015) or PPO (Laskey, Lee, et al., 2017; Ho & Ermon, 2016; Cheng et al., 2018; Sun et al., 2017). While this approach is convenient for producing a supervisor, it may lead to potential issues when the learned supervisor is queried at unseen states during learning.

For example, if an algorithm such as DAgger is applied, the robot could enter parts of the state space that the learned supervisor never saw. In these unseen states, the supervisor may provide very low-quality examples of what the robot should do, which could result in the robot learning a sub-optimal policy. Thus, when comparing two different algorithms, the performance may differ not because of the algorithms, but because the supervisor was providing low-quality labels to one of them. Ideally, a supervisor should be consistent in the quality of examples regardless of the state queried.

Promising Benchmarks

One example of a good benchmark for IL, is the Mario domain used in the original DAgger paper (Ross et al., 2011b). In Mario, an agent is trained to traverse a level and avoid dying by jumping on top of enemies and over obstacles. The goal is to stay alive as long as possible.

Mario correctly tests generalization because each new level is randomly generated from some distribution. Thus during every policy rollout, the position and types of obstacles and enemies are varied. The effect of this is the robot cannot replay a sequence from the supervisor, such as in the Hopper example, but must learn a policy that generalizes to new unseen states. Furthermore, given the number of combinations possible to create a level is exponentially large, it is very unlikely that space could ever be covered even with substantial computing power. Thus, any new algorithm must explicitly be designed to increase generalization to perform well.

The Mario benchmark is also able to provide an algorithmic supervisor, which prevents the need to train one. Since Mario is a discrete action state space with a

relatively low number of possible moves, a supervisor can be computed by running a search algorithm A^* leveraging the games internal dynamics. An A^* supervisor is advantageous because it eithers provides high-quality labels or determines no possible moves exist. Thus, users of the benchmark have a much better understanding of data being provided to each algorithm.

Mario does have several downsides, which prevent it from being the standard benchmark. First, it uses a discrete action space instead of continuous, so it cannot test control based algorithms. Second, it is not reflective of most robotics tasks, since it is only simulating a children's video game. Finally, it is written in Java, while most machine learning is performed in Python.

The FLUIDS Simulator

To help facilitate better testing of IL algorithms, we design the FLUIDS simulator. FLUIDS (First-Order Lightweight Urban Intersection Driving Simulator) is a simulator for modeling the interactions at traffic intersections. Navigating traffic intersections are interesting because in order to robustly navigate an intersection a robot needs to learn a complex hierarchical plan that can be executed on sensor data in a wide variety of configurations. Thus making it challenging and interesting task to benchmark IL algorithms on.

Similar to the Mario environment, FLUIDS supports a wide initial state distribution and built-in planner. However, in contrast it focuses on the continuous control, a task that is more relevant for robotics and is buitl entirely in Python, which allows for easy installation. In the next, section we will discuss the design of FLUIDS.

Terminology

Intersection: A layout of sidewalks, lanes, terrain, and traffic signals which govern when and where cars and pedestrians may travel.
Agent: A car or pedestrian which receives a state representation, and responds with an action.
State: The collection of all simulated objects in the environment, including their position, orientation, and velocity.

Architecture

FLUIDS uses OpenAI's Gym API and Python to allow for easy interfacing with popular learning frameworks such as Tensorflow (Abadi et al., 2015), PyTorch (Paszke et al., 2017), and SciKit Learn (Pedregosa et al., 2011).

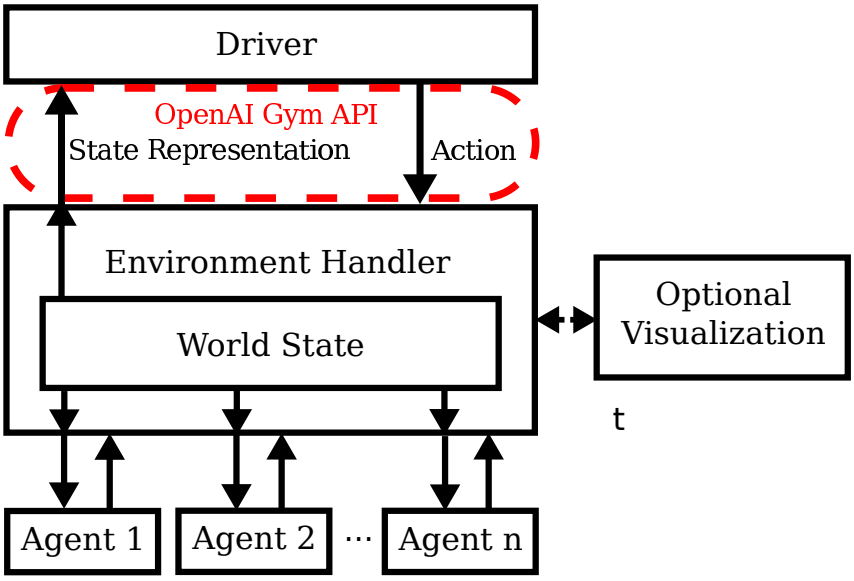


Figure 7.2: FLUIDS' system architecture. The evaluated driver interfaces to the world via the OpenAI Gym interface. In the back-end, the simulator coordinates the behavior of additional background drivers and pedestrians.

FLUIDS can be run using parallel frameworks such as Python's native multiprocessing package for fast data collection and evaluation.

FLUIDS also provides algorithmic supervisors for controlling the background cars and pedestrians in a scene. These background agents are designed to test the ability of a user agent in different multi-agent interactions.

Agent Dynamics

Agents fall in three classes: pedestrians, cars, and signal lights.

Pedestrian Dynamics

A pedestrian is parameterized by the state space $\mathbf{x}_p(t) = [x, y, \theta, v]$, which corresponds to its positional coordinates, orientation and velocity. The control signal for a pedestrian at time t is specified as $\mathbf{u}_c(t) = [\psi, a]$, which refers to walking direction and acceleration.

For pedestrian dynamics FLUIDS uses a point model to reflect the omni-directional

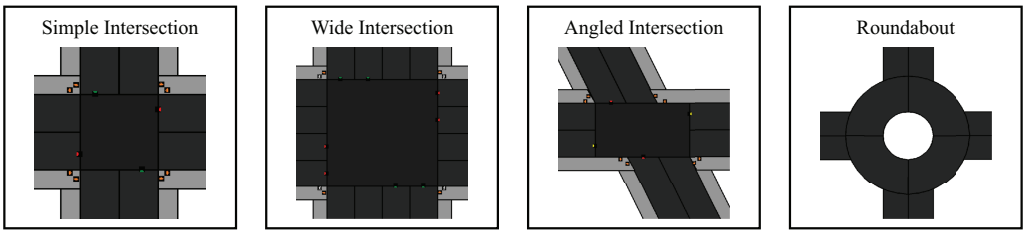


Figure 7.3: A sample of the different types of traffic intersections that can be designed in FLUIDS. Intersection types include common intersection like a 4-way stop and more intricate ones such as a roundabout. Intersections can also contain sidewalks, traffic lights, and crosswalk lights.

nature of pedestrian movement.

$$\begin{aligned}\dot{x} &= v\cos(\psi)\\ \dot{y} &= v\sin(\psi)\\ \dot{v} &= a\\ \dot{\theta} &= \psi\end{aligned}$$

Car Dynamics

The cars in FLUIDS are configurable in size, mass, and maximum velocity. The dynamic state of each car in FLUIDS is described by the state space $\mathbf{x}_c(t) = [x, y, \theta, v]$, which corresponds to positional coordinates, orientation and velocity. The control signal for each car at time t is specified as $\mathbf{u}_c(t) = [\psi, F]$, which refers to steering angle and acceleration force.

For cars, FLUIDS uses the kinematic bicycle model (Polack, Altché, d'Andréa Novel, & de La Fortelle, 2017). The differential equations that describe the model are as follows.

$$\begin{aligned}\dot{x} &= v\cos(\theta+\beta)\\ \dot{y} &= v\sin(\theta+\beta)\\ \dot{v} &= F/m\\ \dot{\theta} &= \frac{v}{l_r}\sin(\beta)\\ \beta &= \tan^{-1}\left(\frac{l_r}{l_f+l_r}\tan(\psi)\right)\end{aligned}$$

This model defines β as the angle between the current velocity of the center of mass with respect to the longitudinal axis of the car. l_r and l_f are the distance from the center of the mass of the vehicle to the front and rear axles. The control force F is scaled by the mass of the car m to provide control over the car's acceleration.

Traffic Lights and Crosswalk Lights

Traffic and crosswalk lights can be set up at the end of each lane or sidewalk. Traffic lights switch among the standard three colors: red, yellow, and green. Crosswalk lights are either white or red. Cars are aware of traffic light states and pedestrians are aware of crosswalk light states. FLUIDS allows for both types of lights to loop, stopping for any specified time on each color. This allows for a wide variety of light synchronization schemes.

Collision Interactions

FLUIDS performs collision checking using the Shapely geometry library (Popinet et al., 2014–2018). Shapely provides a fast, configurable interface for manipulating and designing two-dimensional polygons. Every object in FLUIDS is associated with a Shapely object that acts as its bounding volume. At every timestep, collision checking is performed and all collisions logged.

Observation Spaces in FLUIDS

FLUIDS offers three options for accessing the observations of the world; explicit, bird's-eye view, and quasi-LIDAR perspective. These spaces offer a wide variety in terms of the number of dimensions and geometric structure of the observed scene.
Explicit View In the explicit view the agent has access to the true state space (i.e., x_c and x_p of all N agents in the scene) Additionally, the agent can observe the state of all traffic lights and pedestrian crosswalk lights.

This view can be seen as having access to a global oracle where all information is available to an agent. FLUIDS also implements a Gaussian noise model over this space.
Bird's-Eye View In bird's-eye view, a top-down image of the intersection is available to the agent. These images use an RGB color representation of configurable resolution.

Viewpoints from above, such as those from unmanned aerial vehicles (Salvo, Caruso, Scordo, Guido, & Vitale, 2017) or traffic cameras (Koller et al., 1994) motivate this state space. In this state representation, agents must infer from pixel data the location of other agents in the scene and the current state of traffic lights. This state representation is relevant for potentially evaluating the efficiency of recent advances in Convolutional Neural Networks for planning (Mnih et al., 2013).

FLUIDS additionally provides a sensor noise model for this state representation. Specifically, FLUIDS supports adding a bounded zero-mean Gaussian noise to each pixel value, with variance defined by λ_p.
Quasi-LIDAR Representation Quasi-LIDAR representation is a state space that

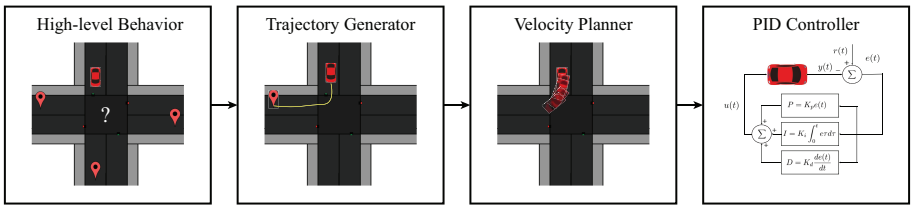


Figure 7.4: The planner for the built-in driving agent. First, a high-level destination, specified as a destination lane, is randomly selected for the agent. Then a positional trajectory is generated via OMPL to take the agent to the goal position. A velocity planner is then called to ensure the agent reaches the destination without collisions or violation of traffic laws with respect to the traffic light. Finally, a PID controller is used to execute the planned path.

models the world with information that mimics what an autonomous car extracts from the physical world. With recent advances in LIDAR and visual object detection, it is possible for a car to extract relative poses and class labels of other agents adjacent to it (Bergholz, Timm, & Weisser, 2000)(Fan, Yi, Hao, Mengyin, & Shunting, 2016).

To create the quasi-LIDAR state space m rays are projected from the agent's car at angular intervals along a 360° arc. If a ray collides with an object in the scene, it returns the following information $\{d, \text{label}, \psi_r, v_r\}$, which corresponds to the distance to, the class label of, the relative angle to, and the relative velocity to the object. Fig. 7.5 shows an illustration of this state space.

FLUIDS models the sensor noise of this state space, via additive element-wise zero-mean Gaussian noise to the relative distance, velocity, and angle. The parameter λ_l specifies the Gaussian noise. In addition to Gaussian noise, FLUIDS also models dropped observations. With probability ϵ a reading from a given ray is returned as empty space, to model the effect of missing data.

Built-in Hierarchical Supervisor

In order to support a baseline supervisor, for both comparison and to collect training data, FLUIDS implements a planner with four hierarchical components:a high-level behavior component, a nominal trajectory generator, a velocity planner, and a PID controller. These four components encompass what is commonly referred to as the high-level behavioral layer, motion planner, and local feedback control components of the autonomous car control stack (Paden, Čáp, Yong, Yershov, & Frazzoli, 2016). Fig. 7.4 displays how the components of the hierarchy contribute to the total autonomous control of a vehicle.

For each level of the hierarchy, FLUIDS provides an algorithmic supervisor that generates plausible behavior. Additionally, Users are also able to implement their

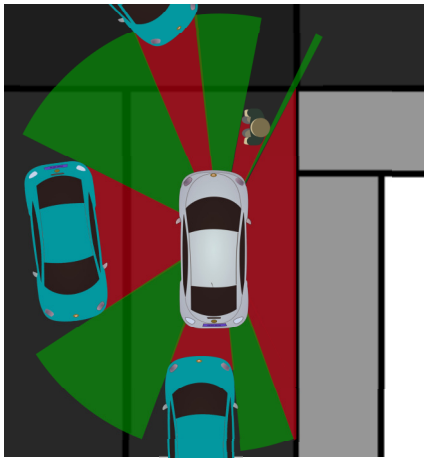
r 

Figure 7.5: Illustration of the quasi-LIDAR perspective state space. Rays are projected into the scene from the car's perspective. The red rays correspond to detected objects in the environment. The density of projected rays and noise model are both configurable.

own controllers at each level of the hierarchy, thus control algorithms can be tested at different levels of abstraction.

Behavioral Logic

Control Space Each agent in the driving simulator is assigned a target lane. Given K lanes, the control space for the agent is the selection of a target lane (i.e. $\mathcal{U} = \{l_0, ..., l_K\}$).
Algorithmic Supervisor The implemented planner operates by sampling a start lane uniformly and target lane with the following probability distribution.

$$p(l_t|l_g) = \begin{cases} \frac{1}{K-1} & \text{if } l_t \neq l_g \\ 0 & \text{if } l_t = l_g \end{cases},$$

Nominal Trajectory Generator

Control Space Once any car has an intended goal state, the next level in the hierarchy is to generate a path in Cartesian (x, y) space towards some designated goal state given an initial state representation of the world. Formally, the agent's control space is a set of 2D-positional points (i.e. $U \in \{\mathbb{R} \times \mathbb{R}\}^T$).
Algorithmic Supervisor The implemented planner uses deCastlejau spline interpolation to generate the nominal trajectory. We interpolate a path through a set of waypoints in the scene. Waypoints are located at each end of each lane. The

waypoints to hit are selected by the high-level behavioral planner. The same is used for pedestrians.

Velocity Planner

Control Space Similar to (Best, Narang, Pasqualin, Barber, & Manocha, 2017), we have an agent that plans the velocity of the trajectory after generation of the nominal trajectory. The agent on this level is given a current state representation of the world and must select the velocity from the bounded range $\mathcal{U} \in [0, v_{max}]$, where v_{max} corresponds to the maximum value.
Algorithmic Supervisor The implemented velocity planner performs a forward projection for each agent for T time-steps in the environment for every possible velocity the car may choose to maintain. Given these projections, the problem of multi-agent pathing is formulated as a constraint satisfaction problem (CSP). Interactions with traffic lights and pedestrians form unitary constraints in this CSP. This CSP is solved with Python-Constraint to assign a collision-free target velocity for every vehicle in the scene (Niemeyer, 2017).

Control Input

Control Space Given a trajectory of positional (x, y) points and target velocities, we now need to generate controls that can accurately track the trajectory. In this low-level of the hierarchy, the input is steering angle and acceleration, which an agent needs to determine given some current state representation of the world.
Algorithmic Supervisor FLUIDS uses two PID controllers. One controls the steering angle, ψ, to track the direction to the next positional point and the other controls the acceleration a to the next target velocity. We empirically observe the PID controllers can track over 99% of 200 test trajectories generated.

Testing Generalization in FLUIDS

One of the advantages of FLUIDS highly variable design is that it can be used to test how well a policy generalizes to situations that are out of the training distribution, or variations in the initial state distribution $p(\mathbf{x}_0)$ and the dynamics $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$.

We can illustrate this with a policy trained in the simulation with Behavior Cloning on 100 demonstrations from the supervisor. The policy is mapping the Q-LIDAR sensor observations to the a target velocity action. To evaluate the sensitivity of the learned velocity controller to parameters in the simulator. We perform a grid search over six parameters of the simulator:

- The number of cars in the scene, in the range $[2, 5]$.

- The number of pedestrians in the scene, in the range $[0, 4]$.
- The variance, λ_l of quasi-LIDAR noise, in the range $[0, 1]$.
- The probability of omitting a sensor reading from the quasi-LIDAR state, in the range $[0, 1]$.
- The presence of traffic lights, $[0, 1]$.
- The mass of the cars, in the range $[0, 200]$.

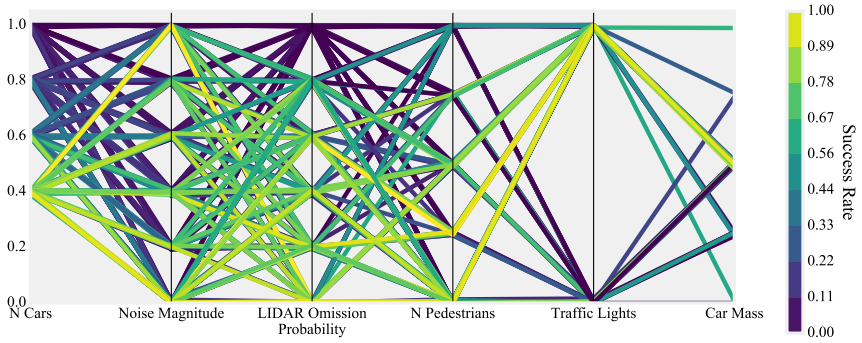


Figure 7.6: Sensitivity analysis of the imitation learner to state complexity and observation noise, performed over 256 configurations by evaluating the learned policy over half a million data points. For visualization we normalize all parameters between 0 and 1. We sweep between 2 and 5 cars, 0 and 4 pedestrians, and vary the noise and dynamics parameters. We report success rate as the percentage of evaluations in which the learned policy successfully guides all vehicles to their goals. The analysis reports that our imitation learned agent is tolerant to noisy observations, but fails when run in an environment with a large number of cars or pedestrians. We also observe model mismatch when we alter the car mass and remove traffic lights.

Fig. 7.6 reports the generated sensitivity analysis. FLUIDS evaluates the learned policy across 256 configurations, running in total over 5000 trajectories. The analysis reports that the performance of the learned policy is very sensitive to perturbations in the number of cars in the scene, with more cars providing a challenge and leading to a lower success rate. The sensitivity analysis also allows us to visualize the effect of quasi-LIDAR omissions, with our imitation learned agent being robust to increased omission probability until all the quasi-LIDAR observations are omitted, after which the success rate drops drastically. Additionally, the sensitivity analysis indicates the importance of traffic lights, which coordinate traffic through the intersection and prevent gridlock. Finally, the sensitivity analysis shows the importance of car mass,

with slightly different car masses providing a model mismatch and leading to lower success rates.

In total, this sensitive analysis provides an understanding of not just how well the policy performs on the distribution it was changed on, but also how sensitive it is to far out of sample perturbations. Having an understanding of this can help create more robust Imitation Learning algorithms in the future.

Discussion and Future Work

FLUIDS is a fast first-order simulator of multi-agent driving behaviors. The interface is lightweight, highly configurable, suitable for developing and evaluating autonomous agents in urban environments.FLUIDS can also be extended to cover a more diverse array of driving situations beyond intersections. Examples include navigating a dense parking lot or successfully merging onto a highway lane. The variety of agents can also be expanded to include bicyclists, emergency services, and cars with trailers. Adding a more diverse set of signals such as yield signs, railroad crossings signs, and turn only signals will also allow for more diverse intersections.

FLUIDS offer some new solutions to better test generalization of IL algorithms and provides a consistent supervisor. However, FLUIDS does have several limitations such as a low-dimensional action space and is still far from a realistic simulator. Going forward it would be interesting to examine how to run IL experiments efficiently in more complex simulators, such as CARLA (Dosovitskiy, Ros, Codevilla, López, & Koltun, 2017) or creating FLUIDS like simulator for manipulation tasks.

Chapter 8

Reducing Covariate Shift with Reward Information

In this Chapter, we review alternatives to reduce covariate shift. That cannot be classified as purely On of Off-Policy IL approaches because they assume access to more information. These techniques either exploit other type of information, such as a reward function, to decide what control to apply or having access to the Q-function of the expert's policy.

Leveraging Cost-To-Go Estimate

A variety of On-Policy algorithms leverage addition information about the expert's cost to go at each state. This can be written in the form of a Q-function or Advantage function. The intuition behind this information is that an agent can reason about the cost of taking an action with respect to the true task performance and not just what the supervisor would do.

Notable algorithms in this category are AGGRAVATED (Ross & Bagnell, 2014) and recently its policy gradient formulation variant Deeply AGGRAVATED (Sun et al., 2017). These algorithms are advantageous because they can allow the agent to act more like a planner and consider the trade-offs between taking different actions. Furthermore, the agent may not need to perfectly represent the supervisor to achieve similar task performance, since it can choose to take different routs of similar cost.

However, as noted by (Cheng et al., 2018) in robotic applications these algorithms can be challenging because obtaining good estimates of the expert's cost to go may require a large amount of data and a sufficiently structured reward function. Furthermore, some of these algorithms require being able to access arbitrary states to recompute path costs, which is quite challenging for robotic tasks where the dynamics are not known.

Leveraging Reinforcement Learning

One approach to bypass the need for reducing to covariate shift, is to run a Reinforcement Learning after a policy has been trained with Behavior Cloning. Reinforcement Learning forces the robot into states that do to errors made and provides feedback in the form of a reward signal. The benefit of this is that the robot receives a signal of what actions to apply on the current distribution without the need for a supervisor to provide corrective feedback. Recently, both On and Off-Policy approaches have been explored to achieve this (Rajeswaran et al., 2017; Gao et al., 2018). One of the most notable works of this approach is AlphaGo (?, ?).

One the the challenges with this approach though is that provided reward function may be sparse and require a large number of samples to optimize. Thus, a large amount of work has been examining ways to determine structure in the reward function via leveraging the demonstrations (Rajeswaran et al., 2017; Gao et al., 2018). This is quite similar to the ideas of Inverse Reinforcement Learning, in which a cost function is derived from demonstrations, however in this setting it is typically assumed their is no explicit transition model of the world.

One recent work to this approach, is known as Generative Adversarial Imitation Learning (Ho & Ermon, 2016) (GAIL). GAIL attempts to bypass the problem of sparse reward by treating Imitation Learning as a generative model formulation, where a given a set of initial distributions.The policy is treated as a generative model that tries to mimic the distribution and a discriminator is learned to try and distinguish between the real and fake demonstrations. In this the policy is learned with RL, but the reward function can be based on fooling the discriminator, rather than some notion of true success. Thus, avoiding the need to explicitly set the reward.

All of these approaches provide new and interesting ways to reduce covaraite shift, with minimal modifications to the supervisor's policy. However, it should be noted that unsupervised roll-outs of the robot in the physical world do come with additional hidden costs, such as the need to reset the environment. For example, consider a robot trying to tie a rope. Given the state of current RL algorithms we would expect the robot to have to tie the rope this task thousands of time before it is highly reliable (Rajeswaran et al., 2017).

However, after each each attempt, it would need to be untied to restart the world. Considering that untying a rope is potentially much harder than tying it (Liang et al., n.d.), either a human supervisor or a hardware contraption would need to be used to achieve this. Either one of these choices could require extensive human labor that needs to be taken into account when applying an RL technique. Thus, in some applications it may be that the hidden cost of creating a system to perform episodic RL in outweighs the cost of using pure Imitation Learning methods.

Chapter 9

Conclusion

Imitation Learning provides a set of solutions that promise to reduce programmer burden. We showed the advantages of Imitation Learning in robotic manipulation by training a robot to perform an object retrieval and planar part singulation. To advance these systems from prototypes to reliable products, we need a better understanding of the trade-offs between different algorithms for data collection.

This dissertation explores two classes of IL algorithms, On- and Off-Policy sampling, and presented several findings on each of their inherent trade-offs in achieving robustness. We concluded Off-Policy methods appear beneficial because they can be implemented in batch settings more efficiently and were shown to be more amenable to human supervisors than On-Policy methods. Off-Policy methods though require an expressive policy representation and noise to be injected into the supervisor's control stream to show similar performance outcomes as On-Policy.

In applications with deep neural network policy representations, this is a reasonable requirement. However, domains with very limited onboard computation–such as embedded systems or possibly very small datasets–a shallow policy representation is preferred. Under these conditions, On-Policy approaches can be quite effective at enabling a robot to perform robustly in new situations. Thus, neither method is preferable in all settings and both should be studied further.

Open Questions While our study led to a better understanding of the trade-offs between On- and Off-Policy, it also raised several new questions that are both theory and systems related. One of the significant theoretical question in Imitation Learning using Off-Policy techniques is better understanding the finite sample performance with noise injection. Our current analysis showed there is a connection between the likelihood of sampling the robot's final policy and the bias-variance of the policy representation. Potentially leveraging results from complexity analysis in random design setting could yield informative answers for rates on finite sample convergence (Hsu et al., 2012).

Interestingly, as we discussed in Chapter 4, we found that On-Policy methods

may not always converge to the best solution on induced distributions. To explore when this non-convergence can occur, we introduced the idea of stability in dynamic regret. This idea is a departure from the traditionally-used static regret. We saw that in some cases the expressiveness needs to be decreased for the policy to obtain a local optimum when deploying On-Policy techniques under this context. Still, many questions remain for cases when non-strongly convex policies are used, such as deep neural networks, and whether On-Policy methods can converge in these settings.

In evaluation with human supervisors, the task considered required the supervisor to control the robot at the end-effector position level. For a large number of potential manipulation tasks this might be not the right level of abstraction to provide feedback to the robot. In some tasks, it could be more advantageous to specify control at the grasp placement or trajectory level. In these settings, On-Policy methods may become more comfortable for a human supervisor's because the high-level of abstraction loosens the need for a tight observation feedback loop.

Finally, in simulated benchmarks expanding on the complexity and ability to test these algorithms is essential to progressing research and understanding these different approaches to IL. Our presented benchmark, FLUIDS, provides new ways to test generalization ability to out-of-sample perturbations as well as a fixed supervisor. However, it is limited in only modeling one type of task. Going forward, researchers should design more benchmarks primarily for Imitation Learning, not just Reinforcement Learning.

In conclusion, Imitation Learning has the potential to help robots quickly make decisions on complex high-dimensional sensor observations. If we expect large scale systems to be reliable, then it is imperative to understand the types of error prevalent in IL and what techniques are preferable under given conditions.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... Zheng, X. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems.* Retrieved from `https://www.tensorflow.org/` (Software available from tensorflow.org)

Abbeel, P., & Ng, A. Y. (2004). Apprenticeship learning via reinforcement learning. In *Proceedings of the twenty-first international conference on machine learning* (p. 1).

Argall, B. D., Chernova, S., Veloso, M., & Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and autonomous systems*, *57*(5), 469–483.

Atlas, L. E., Cohn, D. A., & Ladner, R. E. (1990). Training connectionist networks with queries and selective sampling. In *Nips* (pp. 566–573).

Bergholz, R., Timm, K., & Weisser, H. (2000, November 21). *Autonomous vehicle arrangement and method for controlling an autonomous vehicle.* Google Patents. (US Patent 6,151,539)

Best, A., Narang, S., Pasqualin, L., Barber, D., & Manocha, D. (2017). Autonovi: Autonomous vehicle planning with dynamic maneuvers and traffic constraints. *arXiv preprint arXiv:1703.08561*.

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., ... others (2016). End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*.

Breiman, L. (1996). Bagging predictors. *Machine learning*, *24*(2), 123–140.

Burbidge, R., Rowland, J. J., & King, R. D. (2007). Active learning for regression based on query by committee. In *Intelligent data engineering and automated learning-ideal 2007* (pp. 209–218). Springer.

Cheng, C.-A., & Boots, B. (2018). Convergence of value aggregation for imitation learning. *arXiv preprint arXiv:1801.07292*.

Cheng, C.-A., Yan, X., Wagener, N., & Boots, B. (2018). Fast policy learning through imitation and reinforcement. *arXiv preprint arXiv:1805.10413*.

Chernova, S., & Veloso, M. (2009). Interactive policy learning through confidence-based autonomy. *Journal of Artificial Intelligence Research*, *34*(1), 1.

Codevilla, F., Müller, M., Dosovitskiy, A., López, A., & Koltun, V. (2017). End-to-end driving via conditional imitation learning. *arXiv preprint arXiv:1710.02410*.

Cohn, D., Atlas, L., & Ladner, R. (1994). Improving generalization with active learning. *Machine learning*, *15*(2), 201–221.

Dosovitskiy, A., Ros, G., Codevilla, F., López, A., & Koltun, V. (2017). CARLA: An open urban driving simulator. In *Proceedings of the 1st annual conference on robot learning (corl)* (pp. 1–16).

Duan, Y., Andrychowicz, M., Stadie, B., Ho, O. J., Schneider, J., Sutskever, I., ... Zaremba, W. (2017). One-shot imitation learning. In *Advances in neural information processing systems* (pp. 1087–1098).

Duvallet, F., Kollar, T., & Stentz, A. (2013). Imitation learning for natural language direction following through unknown environments. In *Icra* (pp. 1047–1053).

Fan, Q., Yi, Y., Hao, L., Mengyin, F., & Shunting, W. (2016). Semantic motion segmentation for urban dynamic scene understanding. In *Automation science and engineering (case), 2016 ieee international conference on* (pp. 497–502).

Gao, Y., Xu, H., Lin, J., Yu, F., Levine, S., & Darrell, T. (2018). *Reinforcement learning from imperfect demonstrations.* Retrieved from `https://openreview.net/forum?id=BJJ9bz-0-`

Geurts, P. (2009). Bias vs variance decomposition for regression and classification. In *Data mining and knowledge discovery handbook* (pp. 733–746). Springer.

Grollman, D. H., & Jenkins, O. C. (n.d.). Dogged learning for robots. In *Icra, 2007 ieee.*

Guo, X., Singh, S., Lee, H., Lewis, R. L., & Wang, X. (2014). Deep learning for real-time atari game play using offline monte-carlo tree search planning. In *Nips* (pp. 3338–3346).

Ho, J., & Ermon, S. (2016). Generative adversarial imitation learning. In *Advances in neural information processing systems* (pp. 4565–4573).

Hodge, V. J., & Austin, J. (2004). A survey of outlier detection methodologies. *Artificial Intelligence Review*, *22*(2), 85–126.

Hsu, D., Kakade, S. M., & Zhang, T. (2012). Random design analysis of ridge regression. In *Conference on learning theory* (pp. 9–1).

Judah, K., Fern, A., & Dietterich, T. (2011). Active imitation learning via state queries. In *Proceedings of the icml workshop on combining learning strategies to reduce label cost.*

Judah, K., Fern, A., & ietterich, T. G. (2012). Active imitation learning via reduction to iid active learning. *arXiv preprint arXiv:1210.4876*.

Kakade, S. M. (2002). A natural policy gradient. In *Advances in neural information processing systems* (pp. 1531–1538).

Kim, B., & Pineau, J. (2013). Maximum mean discrepancy imitation learning. In *Robotics science and systems.*

King, J. E., Haustein, J. A., Srinivasa, S. S., & Asfour, T. (n.d.). Nonprehensile whole arm rearrangement planning on physics manifolds. *ICRA 2015 IEEE*.

Kitaev, N., Mordatch, I., Patil, S., & Abbeel, P. (2015). Physics-based trajectory optimization for grasping in cluttered environments. , 3102–3109.

Knox, E. M., & Ng, R. T. (1998). Algorithms for mining distancebased outliers in large datasets. In *Proceedings of the international conference on very large data bases.*

Koller, D., Weber, J., Huang, T., Malik, J., Ogasawara, G., Rao, B., & Russell, S. (1994). Towards robust automatic traffic scene analysis in real-time. In *Pattern recognition, 1994. vol. 1-conference a: Computer vision & image processing., proceedings of the 12th iapr international conference on* (Vol. 1, pp. 126–131).

Laskey, M., Chuck, C., Lee, J., Mahler, J., Krishnan, S., Jamieson, K., ... Goldberg, K. (2017). Comparing human-centric and robot-centric sampling for robot deep learning from demonstrations. In *Robotics and automation (icra), 2017 ieee international conference on* (pp. 358–365).

Laskey, M., Lee, J., Chuck, C., Gealy, D., Hsieh, W., Pokorny, F. T., ... Goldberg, K. (2016a). Robot grasping in clutter: Using a hierarchy of supervisors for learning from demonstrations. *Automation Science and Engineering (CASE), 2016 IEEE*, 827–834.

Laskey, M., Lee, J., Chuck, C., Gealy, D., Hsieh, W., Pokorny, F. T., ... Goldberg, K. (2016b). Robot grasping in clutter: Using a hierarchy of supervisors for learning from demonstrations. In *Automation science and engineering (case), 2016 ieee international conference on* (pp. 827–834).

Laskey, M., Lee, J., Fox, R., Dragan, A., & Goldberg, K. (2017). Dart: Noise

injection for robust imitation learning. In *Conference on robot learning* (pp. 143–156).

Laskey, M., Staszak, S., Hsieh, W. Y.-S., Mahler, J., Pokorny, F. T., Dragan, A. D., & Goldberg, K. (2016). Shiv: Reducing supervisor burden in dagger using support vectors for efficient learning from demonstrations in high dimensional state spaces. In *Robotics and automation (icra), 2016 ieee international conference on* (pp. 462–469).

Levine, S., Finn, C., Darrell, T., & Abbeel, P. (2015). End-to-end training of deep visuomotor policies. *arXiv preprint arXiv:1504.00702*.

Levine, S., & Koltun, V. (2013). Variational policy search via trajectory optimization. In *Advances in neural information processing systems* (pp. 207–215).

Liang, J., Mahler, J., Laskey, M., Li, P., & Goldberg, K. (n.d.). Using dvrk teleoperation to facilitate deep learning of automation tasks for an industrial robot. *Research report, UC Berkeley*.

Liu, W., Hua, G., & Smith, J. R. (2014). Unsupervised one-class learning for automatic outlier removal. In *Cvpr, 2014 ieee* (pp. 3826–3833).

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Nadaraya, E. A. (1964). On estimating regression. *Theory of Probability & Its Applications*, *9*(1), 141–142.

Niemeyer, G. (2017). *Python-Constraint: Solving constraint satisfaction problems in Python.* Retrieved from `https://labix.org/python-constraint` ([Online; accessed <today>])

Osa, T., Pajarinen, J., Neumann, G., Bagnell, J. A., Abbeel, P., Peters, J., et al. (2018). An algorithmic perspective on imitation learning. *Foundations and Trends® in Robotics*, *7*(1-2), 1–179.

Paden, B., Čáp, M., Yong, S. Z., Yershov, D., & Frazzoli, E. (2016, March). A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, *1*(1), 33-55.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., ... Lerer, A. (2017). Automatic differentiation in pytorch.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Polack, P., Altché, F., d'Andréa Novel, B., & de La Fortelle, A. (2017). The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles? In *Intelligent vehicles symposium (iv), 2017 ieee* (pp. 812–818).

Pomerleau, D. A. (1989). *Alvinn: An autonomous land vehicle in a neural network* (Tech. Rep.). Carnegie-Mellon University.

Popinet, S., et al. (2014–2018). *Shapely.* `https://pypi.python.org/pypi/Shapely`.

Rajeswaran, A., Kumar, V., Gupta, A., Schulman, J., Todorov, E., & Levine, S. (2017). Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*.

Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems* (pp. 91–99).

Ross, S., & Bagnell, D. (2010). Efficient reductions for imitation learning. In *International conference on artificial intelligence and statistics* (pp. 661–668).

Ross, S., & Bagnell, J. A. (2014). Reinforcement and imitation learning via interactive no-regret learning. *arXiv preprint arXiv:1406.5979*.

Ross, S., Gordon, G., & Bagnell, D. (2011b). A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics* (pp. 627–635).

Ross, S., Gordon, G. J., & Bagnell, J. A. (2011a). A reduction of imitation learning and structured prediction to no-regret online learning. *AISTATS. Vol. 1. No. 2*.

Ross, S., Melik-Barkhudarov, N., Shankar, K. S., Wendel, A., Dey, D., Bagnell, J. A., & Hebert, M. (n.d.). Learning monocular reactive uav control in cluttered natural environments. In *Icra, 2013 ieee.*

Salvo, G., Caruso, L., Scordo, A., Guido, G., & Vitale, A. (2017). Traffic data acquirement by unmanned aerial vehicle. *European Journal of Remote Sensing*, *50*(1), 343–351.

Schölkopf, B., Platt, J. C., Shawe-Taylor, J., Smola, A. J., & Williamson, R. C. (2001). Estimating the support of a high-dimensional distribution. *Neural computation*, *13*(7), 1443–1471.

Schölkopf, B., & Smola, A. J. (2002). *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. MIT press.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., & Abbeel, P. (2015). Trust region policy optimization.

Settles, B., & Craven, M. (2008). An analysis of active learning strategies for sequence labeling tasks. In *Proceedings of the conference on empirical methods in natural language processing* (pp. 1070–1079).

Shalev-Shwartz, S., et al. (2012). Online learning and online convex optimization. *Foundations and Trends® in Machine Learning*, *4*(2), 107–194.

Shiarlis, K., Wulfmeier, M., Salter, S., Whiteson, S., & Posner, I. (2018). Taco: Learning task decomposition via temporal alignment for control. *arXiv preprint arXiv:1803.01840*.

Slud, E. V. (1977, 06). Distribution inequalities for the binomial law. *Ann. Probab.*, *5*(3), 404–412. Retrieved

from http://dx.doi.org/10.1214/aop/1176995801 doi: 10.1214/aop/1176995801

Sun, W., Venkatraman, A., Gordon, G. J., Boots, B., & Bagnell, J. A. (2017). Deeply aggrevated: Differentiable imitation learning for sequential prediction. *arXiv preprint arXiv:1703.01030*.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. MIT press.

Tensor flow. (n.d.). https://www.tensorflow.org/.

Todorov, E., Erez, T., & Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *Intelligent robots and systems (iros), 2012 ieee/rsj international conference on* (pp. 5026–5033).

Tokdar, S. T., & Kass, R. E. (2010). Importance sampling: a review. *Wiley Interdisciplinary Reviews: Computational Statistics*, *2*(1), 54–60.

Tong, S., & Koller, D. (2002). Support vector machine active learning with applications to text classification. *The Journal of Machine Learning Research*, *2*, 45–66.

Vert, R., & Vert, J.-P. (2006). Consistency and convergence rates of one-class svms and related algorithms. *The Journal of Machine Learning Research*, *7*, 817–854.

Zhang, J., & Cho, K. (2016). Query-efficient imitation learning for end-to-end autonomous driving. *arXiv preprint arXiv:1605.06450*.

Zinkevich, M. (2003). Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the 20th international conference on machine learning (icml-03)* (pp. 928–936).

Chapter 10

Appendix

Chapter 3

Lemma 1 *If* $\forall \mathbf{u}_1, \mathbf{u}_2 \in \mathcal{U},\ 0 \leqslant l(\mathbf{u}_1, \mathbf{u}_2) \leqslant 1$*, the following is true:*

$$E_{p(\theta^R|\psi)}[|E_{p(\xi|\pi_{\theta^*},\psi)}J(\theta,\theta^*|\xi) - E_{p(\xi|\pi_\theta)}J(\theta,\theta^*|\xi)|] \leqslant \frac{T}{2}E_{p(\theta^R|\psi)}\mathcal{D}_{\mathcal{KL}}\big(p(\xi|\pi_\theta), p(\xi|\pi_{\theta^*},\psi)\big) + \frac{T}{2}$$

Proof:

We can complete the proof by bounding the quantity inside of $E_{p(\theta^R|\psi)}$

$$\begin{aligned}
&|E_{p(\xi|\pi_{\theta^*},\psi)}J(\theta,\theta^*|\xi) - E_{p(\xi|\pi_\theta)}J(\theta,\theta^*|\xi)| && (10)\\
&\leqslant T||p(\xi|\pi_{\theta^*},\psi) - p(\xi|\pi_\theta)||_{TV} && (11)\\
&\leqslant T\sqrt{\frac{1}{2}\mathcal{D}_{\mathcal{KL}}\big(p(\xi|\pi_\theta), p(\xi|\pi_{\theta^*},\psi)\big)} && (12)\\
&\leqslant \frac{T}{2}\mathcal{D}_{\mathcal{KL}}\big(p(\xi|\pi_\theta), p(\xi|\pi_{\theta^*},\psi)\big) + \frac{T}{2} && (13)
\end{aligned}$$

The first line of the proof follows from Lemma 9, which is stated below and the fact $\forall \mathbf{u}_1, \mathbf{u}_2\ 0 \leqslant l(\mathbf{u}_1, \mathbf{u}_2) \leqslant 1$. The second line follows from Pinsker's Inequality. ■

Lemma 9 *Let* P *and* Q *be any distribution on* $\mathcal{X}$*. Let* $f : \mathcal{X} \to [0, B]$*. Then*

$$|E_P[f(x)] - E_Q[f(x)]| \leqslant B||P - Q||_{TV}$$

Proof:

$$
\begin{aligned}
|E_P[f(x)] - E_Q[f(x)]| &= |\int_x p(x)f(x)dx - \int_x q(x)f(x)dx| \\
&= |\int_x (p(x) - q(x))f(x)dx| \\
&= |\int_x \big(p(x) - q(x)\big)(f(x) - \frac{B}{2})dx \\
&+ \frac{B}{2}\int_x \big(p(x) - q(x)\big)dx| \\
&\leqslant \int_x |p(x) - q(x)||f(x) - \frac{B}{2}|dx \\
&\leqslant \frac{B}{2}\int_x |p(x) - q(x)|dx \\
&\leqslant B||P - Q||_{TV}
\end{aligned}
$$

The last line applies the definition of total variational distance, which is $||P - Q||_{TV} = \frac{1}{2}\int_x |p(x) - q(x)|$. ■

Lemma 3 *On-Policy has a higher likelihood of sampling trajectories under $p_{\bar{\theta}^R}(\xi)$ if and only if the following is true:*

$$
\sum_{t=0}^{T-1} E_{p_{\bar{\theta}^R}(\mathbf{x}_t)}||\pi_{\bar{\theta}^R}(\mathbf{x}_t) - \pi_{\theta*}(\mathbf{x}_t)||_2^2 \geqslant \sum_{t=0}^{T-1} E_{p_{\bar{\theta}^R}(\mathbf{x}_t)}||\pi_{\bar{\theta}^K}(\mathbf{x}_t) - \pi_{\theta^{K-1}}(\mathbf{x}_t)||_2^2
$$

Proof:

$$
\min_{\mu \in \{\pi_{\theta*}, \pi_{\theta^{n-1}}\}} E_{p_{\bar{\theta}^R}(\xi)} \sum_{t=0}^{T-1} (\mathbf{u}_t|\mathbf{x}_t - \mu(\mathbf{x}_t))^T \Sigma^{-1} (\mathbf{u}_t|\mathbf{x}_t - \mu(\mathbf{x}_t))
$$

We can first rewrite this objective as follows:

$$
\begin{aligned}
& E_{p_{\bar{\theta}^R}(\xi)} \sum_{t=0}^{T-1} (\mathbf{u}_t|\mathbf{x}_t - \mu(\mathbf{x}_t))^T \Sigma^{-1} (\mathbf{u}_t|\mathbf{x}_t - \mu(\mathbf{x}_t)) && (14) \\
=& \frac{1}{\sigma} E_{p_{\bar{\theta}^R}(\xi)} \sum_{t=0}^{T-1} (\mathbf{u}_t|\mathbf{x}_t - \mu(\mathbf{x}_t))^T (\mathbf{u}_t|\mathbf{x}_t - \mu(\mathbf{x}_t)) && (15) \\
=& \frac{1}{\sigma} \sum_{t=0}^{T-1} E_{p_{\bar{\theta}^R}(\xi)} (\mathbf{u}_t|\mathbf{x}_t - \mu(\mathbf{x}_t))^T (\mathbf{u}_t|\mathbf{x}_t - \mu(\mathbf{x}_t)) && (16) \\
=& \frac{1}{\sigma} \sum_{t=0}^{T-1} E_{p_{\bar{\theta}^R}(\xi)} \mathrm{Tr}(M|\mathbf{x}_t) + (\pi_{\bar{\theta}^R}(\mathbf{x}_t) - \mu(\mathbf{x}_t))^T (\pi_{\bar{\theta}^R}(\mathbf{x}_t) - \mu(\mathbf{x}_t)) && (17)
\end{aligned}
$$

The first line applies the fact that $\Sigma = \sigma I$.The second line rewrites the distribution the expectation is computed over with respect to the marginalized $p(\mathbf{x}_t)$ and leverages the fact expectations commute with linearity. The final line is a known property for Quadratic Expectations. Note $M|\mathbf{x}_t$ is the covariance matrix of the distribution over $p(\mathbf{u}_t|\mathbf{x}_t, \bar{\theta}^R)$.

We can now use the fact that we are computing a minimizer and remove terms that are independent of the selection of $\mu(\mathbf{x}_t)$. We can then rewrite this objective as follows:

$$\min_{\mu \in \{\pi_{\theta^*}, \pi_{\theta^{K-1}}\}} \sum_{t=0}^{T-1} E_{p_{\bar{\theta}^R}(\xi)} ||\pi_{\bar{\theta}^R}(\mathbf{x}_t) - \mu(\mathbf{x}_t))||_2^2$$

We can now substitute the mean term for the sampling distribution of either On or Off-Policy to complete the proof.

■ **Optimal Covariance Matrix** In this section, we will derive the optimal solution for the noise to inject. Note that this is very similar to the common Maximum Likelihood of a covariance matrix. The the optimization problem can be written as:

$$\Sigma^* = \arg\min_{\Sigma} -\tfrac{T}{2} \log |\Sigma^{-1}| + \tfrac{1}{2} E_{p_{\bar{\theta}^R}(\xi)} \textstyle\sum_{t=0}^{T-1} \left(\mathbf{u}_t|\mathbf{x}_t - \pi_{\theta^*}(\mathbf{x}_t)\right)^T \Sigma^{-1} \left(\mathbf{u}_t|\mathbf{x}_t - \pi_{\theta^*}(\mathbf{x}_t)\right).$$

We then take the derivative with respect to Σ^{-1} and set it equal to zero.

$$0 = -\frac{T}{2}\Sigma + \frac{1}{2} E_{p_{\bar{\theta}^R}(\xi)} \sum_{t=0}^{T-1} \left(\mathbf{u}_t|\mathbf{x}_t - \pi_{\theta^*}(\mathbf{x}_t)\right) \left(\mathbf{u}_t|\mathbf{x}_t - \pi_{\theta^*}(\mathbf{x}_t)\right)^T.$$

Then the optimal covariance matrix is

$$\hat{\Sigma} = \frac{1}{T} E_{p_{\bar{\theta}^R}(\xi)} \sum_{t=0}^{T-1} \left(\mathbf{u}_t|\mathbf{x}_t - \pi_{\theta^*}(\mathbf{x}_t)\right) \left(\mathbf{u}_t|\mathbf{x}_t - \pi_{\theta^*}(\mathbf{x}_t)\right)^T.$$

Lemma 4 *Given a valid covariance matrix for Σ, where $\Sigma \in S$, the following is true*

$$|E_{p_{\bar{\theta}^R}(\xi)} \sum_{t=0}^{T-1} log\, p(\mathbf{u}_t|\mathbf{x}_t, \psi = \{\mu(\mathbf{x}_t), \Sigma^*\}) - E_{p_{\bar{\theta}^R}(\xi)} \sum_{t=0}^{T-1} log\, p(\mathbf{u}_t|\mathbf{x}_t, \psi = \{\mu(\mathbf{x}_t), \Sigma^*\})|$$
$$\leqslant G_2 ||\Sigma^* - \Sigma||_F$$

where $G_2 = \max_{\Sigma \in S} ||\nabla_\Sigma E_{p_{\bar{\theta}^R}(\xi)} \sum_{t=0}^{T-1} log\, p(\mathbf{u}_t|\mathbf{x}_t, \psi = \{\mu(\mathbf{x}_t), \Sigma^*\})||_F$

Proof:
We begin the proof by noting that

$$\max_{\Sigma} E_{p_{\bar{\theta}^R}(\xi)} \sum_{t=0}^{T-1} \log p(\mathbf{u}_t|\mathbf{x}_t, \psi = \{\mu(\mathbf{x}_t), \Sigma^*\})$$

is concave with respect to the parameter Σ. This follows from the fact that optimization is identical to the maximum likelihood of a Covariance matrix over the log-likelihood of a sum of Gaussian. By applying the property of concavity, we can obtain the following:

$$\begin{aligned}
&E_{p_{\bar{\theta}^R}(\xi)} \sum_{t=0}^{T-1} \log p(\mathbf{u}_t|\mathbf{x}_t, \psi = \{\mu(\mathbf{x}_t), \Sigma^*\}) - E_{p_{\bar{\theta}^R}(\xi)} \sum_{t=0}^{T-1} \log p(\mathbf{u}_t|\mathbf{x}_t, \psi = \{\mu(\mathbf{x}_t), \Sigma^*\}) \\
&\leqslant < \nabla_{\Sigma} E_{p_{\bar{\theta}^R}(\xi)} \sum_{t=0}^{T-1} \log p(\mathbf{u}_t|\mathbf{x}_t, \psi = \{\mu(\mathbf{x}_t), \Sigma^*\}), (\Sigma^* - \Sigma) > \\
&\leqslant G_2 ||\Sigma^* - \Sigma||_F
\end{aligned}$$

Where, the last line applies Cauchy-Schwartz with the Frobenius Norm and then leverages the maximum bound over the gradient G_2.

Chapter 4

For brevity, we will introduce several new notation in this section. First the loss on a given distribution will be denoted as $f_k(\theta) = E_{p(\xi|\theta^k} J(\theta, \theta^*|\xi)$. Additionally the gradient with respect to this loss (i.e. $\nabla_\theta E_{p(\xi|\theta^k} J(\theta, \theta^*|\xi)$, we will be shorten to $\nabla_k(\theta)$,

Lemma 5 *Given the assumptions, the following equality holds on the difference between consecutive optimal parameters for follow-the-leader and online gradient descent algorithms at any k:*

$$\|\theta^*_{k+1} - \theta^*_k\| \leqslant \lambda \|\theta_{k+1} - \theta_k\|.$$

By strong convexity of f_{k+1}, we have $\frac{\alpha}{2}\|\theta^*_{k+1} - \theta^*_k\|^2 \leqslant f_{k+1}(\theta^*_k) - f_{k+1}(\theta^*_{k+1}) \leqslant \|\nabla_{k+1}(\theta^*_k)\|\|\theta^*_k - \theta_{k+1}\| - \alpha/2\|\theta^*_{n+1} - \theta^*_k\|^2$. Then by rearranging terms, $\|\theta^*_{k+1} - \theta^*_k\| \leqslant \frac{1}{\alpha}\|\nabla_{k+1}(\theta^*_k) - \nabla_k(\theta^*_k)\| \leqslant \frac{\beta}{\alpha}\|\theta_{k+1} - \theta_k\|$, where the last inequality uses Assumption 2.

Theorem 2 *For policy gradient under the assumptions, if $\lambda < 1$, $2\lambda < \psi$ and the step size, $\eta = \frac{\alpha(\alpha^2 - 2\gamma\beta)}{2\beta^2(\alpha^2 - \gamma^2)}$, then $\frac{1}{K} R_D \to 0$ as $K \to \infty$.*

In this proof, we will make use of a variation known as the path variation, which measures the amount of change in the optimal parameters of the loss functions.

Definition 1 (Path Variation) *For a sequence of optimal parameters from m to n given by $\theta^*_{m:n} := (\theta^*_i)_{m \leqslant i \leqslant n}$, the path variation is defined as:*

$$V(\theta^*_{m:n}) := \sum_{i=m}^{n-1} \|\theta^*_i - \theta^*_{i+1}\|.$$

Before directly proving this theorem, we establish several supporting results based on the path variation.

Lemma 6 *For a sequence of predictions made by the online gradient descent algorithm $\theta_{1:K}$ and a sequence of optimal parameters $\theta^*_{1:K}$, the following inequality holds on the path variation:*

$$V(\theta^*_{1:K}) \leqslant \eta \frac{\beta\gamma}{\alpha} \sum_{k=1}^{K} \|\theta_k - \theta^*_k\|.$$

From Lemma 5, we have $\|\theta^*_{k+1} - \theta^*_k\| \leqslant \frac{\beta}{\alpha}\|\theta_{k+1} - \theta_k\| = \frac{\beta}{\alpha}\|\eta\nabla_k(\theta_k)\| = \eta\frac{\beta}{\alpha}\|\nabla_k(\theta_k) - \nabla_k(\theta^*_k)\| \leqslant \eta\frac{\beta\gamma}{\alpha}\|\theta_k - \theta^*_k\|$, where the final inequality uses the smoothness assumption. Then the result follows immediately.

Lemma 7 *Let $\rho = (1 - \alpha\eta + \gamma^2\eta^2)^{1/2}$, which is always nonnegative for any positive choice of η because $\gamma \geqslant \alpha$ by definition. Then the following inequality holds*

$$\sum_{k=1}^{K} \|\theta_k - \theta^*_k\| \leqslant \|\theta_1 - \theta^*_1\| + \sum_{k=1}^{K} \rho\|\theta_k - \theta^*_k\| + V(\theta^*_{1:K}).$$

By strong convexity we have the following: $0 \leqslant 2(f_k(\theta_k) - f_k(\theta^*_k)) \leqslant 2\langle\nabla_k(\theta_k), x_k - \theta^*_k\rangle - \alpha\|\theta^*_k - \theta_k\|^2$. By the update rule given in the algorithm:

$$\begin{aligned}\|\theta_{k+1} - \theta^*_k\|^2 &= \|\theta_k - \eta\nabla_k(\theta_k) - \theta^*_k\|^2 \\ &= \|\eta\nabla_k(\theta_k)\|^2 + \|\theta_k - \theta^*_k\|^2 \\ &\quad - 2\eta\langle\nabla_k(\theta_k), \theta_k - \theta^*_k\rangle.\end{aligned}$$

By rearranging the terms in last line and combining with the very first inequality, we arrive at the following:

$$\|\theta_{k+1} - \theta^*_k\|^2 \leqslant (1 - \alpha\eta)\|\theta_k - \theta^*_k\|^2 + \|\eta\nabla_k(\theta_k)\|^2.$$

Using the smoothness assumption and the fact that $\nabla_n(\theta^*_n) = 0$:

$$\begin{aligned}\|\theta_{k+1} - \theta^*_k\|^2 &\leqslant \|\theta_k - \theta^*_k\|^2 - \alpha\eta\|\theta_k - \theta^*_k\|^2 \\ &\quad + \eta^2\|\nabla_k(\theta_k) - \nabla_k(\theta^*_k)\|^2 \\ &\leqslant \left(1 - \alpha\eta + \gamma^2\eta^2\right)\|\theta_k - \theta^*_k\|^2.\end{aligned}$$

Then let $\rho = (1 - \alpha\eta + \gamma^2\eta^2)^{1/2}$. Consider the series:

$$\begin{aligned}\sum_{k=1}^{K} \|\theta_k - \theta_k^*\| &= \|\theta_1 - \theta_1^*\| + \sum_{n=2}^{N} \|\theta_n - \theta_{k-1}^* + \theta_{k-1}^* - \theta_k^*\| \\ &\leqslant \|\theta_1 - \theta_1^*\| + \sum_{k=2}^{K} \|\theta_n - \theta_{k-1}^*\| + V(\theta_{1:K}^*) \\ &\leqslant \|\theta_1 - \theta_1^*\| + \sum_{k=1}^{K} \rho\|\theta_k - \theta_k^*\| + V(\theta_{1:K}^*),\end{aligned}$$

where the second line uses the definition of the path variation and the third line uses (**??**).

[Proof of Theorem 2] We begin by bounding the result from Lemma 7 above by Lemma 6:

$$\sum_{k=1}^{K} \|\theta_k - \theta_k^*\| \leqslant \|\theta_1 - \theta_1^*\| + \left(\rho + \eta\frac{\beta\gamma}{\alpha}\right) \sum_{k=1}^{K} \|\theta_k - \theta_k^*\|.$$

By rearranging the terms and bounding by the diameter of $\mathcal{X}$:

$$\sum_{k=1}^{K} \|\theta_k - \theta_k^*\| \leqslant \frac{D}{1 - \rho - \eta\frac{\beta\gamma}{\alpha}}.$$

It can be shown that, under the assumptions, the choice of $\eta = \frac{\alpha(\alpha^2 - 2\beta\gamma)}{2\gamma^2(\alpha^2 - \beta^2)}$ ensures that $\left(1 - \rho - \eta\frac{\beta\gamma}{\alpha}\right)$ is positive. By the G-Lipschitz continuity of f_k, we have

$$\sum_{k=1}^{K} f_k(\theta_k) - \sum_{n=1}^{N} f_n(\theta_k^*) \leqslant \frac{GD}{1 - \rho - \eta\frac{\beta\gamma}{\alpha}},$$

and so $R_D(\theta_1, \ldots, \theta_K) = O(1)$. So we have $\frac{1}{K}R_D = O(1/K)$ which goes to zero.