Model Free Safe Learning from Demonstration Using Support Estimation [v5]

Michael Laskey[1],Wesley Yu-Shu Hsieh, Jeff Mahler, Florian T. Pokorny[1], Ken Goldberg[2]

***Abstract*—** [TODO: WRITING ABSTRACT IN THE MORNING]

I. Introduction

[TODO: NEED TO BRAINSTORM A GOOD USE CASE] Consider a robot trying to learn a surgical sub task, such as cutting a circular piece of gauze. The deformable nature of the gauze makes this a complex manipulation task, where traditional control methods could fail because they assume known dynamics of the system. One way to handle this is to assume a competent actor is there to provide demonstrations of good behavior and have the robot to learn a controller to make sequential decisions. Learning from demonstration (LfD) techniques have achieved state of the art performance in in a variety of applications where no knowledge of a reward function or model is available [18], [15], [20]

One approach to LfD is to train a policy, or function estimator (either a classifier or regressor), to predict a competent actor's behavior given training data of the observations (input) and actions (output) performed while accomplishing a task. However the robot's actions affect the next state observed, which violates the standard i.i.d assumption that is common in most statistical learning approaches. Ross et al. proved that this can cause the error in the estimator to compound over the time and lead to performance degradation that is proportional to the time horizon of the task squared. The intuition behind this result is that as the robot makes mistakes with respect to the competent actor's policy it drifts from the distribution it was trained on and encounters new states it can't generalize to. Ross and Bagnell overcome this by presenting an iterative method that first trains a policy on the data observed using standard supervised learning techniques then tries the policy out in the environment to see what states are likely to occur under the robot's trained policy. The competent actor then tells the robot what controls it should applied after each iteration and the policy is retrained [17].

During learning, implementations of Dagger though require exploring states that the competent actor hasn't provided labels for. This in practice can lead the robot to deviate far from the competent actor's policy and encounter potentially dangerous states. For example in [18], DAgger was used to let a quadcopter fly through the forest. When the competent agent demonstrated the task, the robot remained safely away from the trees. However as the robot learned its policy, it deviated from the competent actor's policy and repeatedly crashed into trees. Thus we consider dangerous states to be where the robot is likely to mispredict the competent agent's actions. We are interested in having the robot ask for help when dangerous states are encountered and have the expert take over to demonstrate a proper action.

To estimate where the robot's current policy is likely to mispredict the expert, we leverage a property in statistical machine learning that a trained model will be able to generalize within the distribution it is trained on [22]. This property can be seen used in importance sampling when data is collected from one distribution, but tested on another and has to be re-weighted to the test distribution [8]. One approach then is to estimate the distribution the policy is trained on and then ask for the competent agent to take over if the probability of the current state is low under the estimated distribution. However, the amount of data needed to accurately estimate the density increases exponentially with the dimension of state space [14]. If we are working with partial observations like image data this could be impractical. Thus, we propose using a technique known as the One Class SVM that only tries estimate the level set of the density trained on [19] and has been shown to successfully work on image data before in the context of outlier detection [**?**].

We assume access to an environment where a robot can try out policies and sample trajectories. We also assume access to a competent actor that is able to take control of the robot and tele-operate it at any state in the environment. We assume that we are able to collect a series of partial observations from the environment and the controls applied. Furthermore, we assume the observations encapsulate the Markov assumption in such that the probability of one is only dependent on the last observation and the controls applied.

Our contributions to date are framing the problem of asking for help in an online setting as estimating the support of a distribution and providing an upper bound on the expected number of times the robot's decisions differ from the expert. Initial results on a Mario AI video game averaged over 50 randomly played levels suggest we can maintain the same performance of the final converged DAgger policy during training, while still learning at a similar rate for the underlying policy.

[1]Department of Electrical Engineering and Computer Sciences; {mdlaskey, sachinpatil}@berkeley.edu
[2]Department of Industrial Engineering and Operations Research and Department of Electrical Engineering and Computer Sciences; goldberg@berkeley.edu
[1−2] University of California, Berkeley; Berkeley, CA 94720, USA

II. Related Work

The field of learning from demonstration has achieved state of the art performance in a variety of areas within robotics [2]. Abbeel et al. used an iLQR controller around helicopter trajectories demonstrated by an expert to successfully learn a series of aerobatics stunts [1]. Billard and Matari used a hierarchy of neural networks on video data and tracking markers to imitate a human arms movements on a 37 degree of freedom humaniod robot [3]. Schulman et al. used a non-linear warping, thin plate splines, on collected trajectories of a human controlling a PR2 and tying knots with a rope. Then transferred the demonstrations to the new unseen initial configurations of the rope [20].

A subset of the learning from demonstration field is working within the assumption of no known dynamics and no access to a cost function. Under these assumptions, one approach to learning from demonstration is to collect a set of trajectories and controls from an expert demonstrator and then treat the problem as a supervised learning problem that regresses on a function from state space to controls [2]. Pomerleau et al. used this approach on raw image data and controls provided by a person driving a car and learned a policy that allowed a car to travel on a highway. However, they observed if the car deviated from the middle of the road it would not know how to recover, this behavior was attributed to the fact the car only drove in middle of the road while collecting demonstrations [15]. Ross et al. studied this effect and proved the amount of error accumulated is squared in the time horizon of a policy and proposed a learning algorithm, SMILE, that stochastically mixes the expert control with the robots during training. Thus allowing the robot to explore states the expert hasn't seen before [16].

Ross and Bagnell then improved upon this method with DAgger an iterative method that first trains a policy on the data observed using standard supervised learning techniques and tries the policy out in the environment to see what states are likely to occur under the robots policy. The competent actor then tells the robot what controls it should applied after each iteration and the policy is retrained with an aggregate of all data seen before [17]. A quad-copter using DAgger learned how to fly through a forest using only HOG features extracted from visual images [18]. More recently, Levine et al. extended the iterative learning paradigm through Guided Policy Search, which replaces the expert with iLQG and uses KL divergence constraints to train the policy to match the distributions of the controllers [12].

A potential limitation of DAgger is that it requires sampling from the distribution induced by the current trained policy, which could lead to visiting states where it fails to mimic that of the competent actor. In [18], the quad-copter repeatedly crashed into trees before learning how to navigate them safely. In the reinforcement learning community there is a notion of safe learning, which is learning to do a task while minimizing the cost received during the learning period. Safe learning in the reinforcement learning context has been an extensively study area [7], [10], [5]. One particular technique that has been explored recently for safe reinforcement learning is known as Hamilton-Jacobi-Isaacs (HJI) reachibility analysis, which estimates a safe reachable set using a given nominal model [6], [5]. We assume only access to a set of demonstrations and high dimensional partially observed states, thus use of a model is infeasible. Garcia and Fernando purposed a model free approach to safe learning, but requires the user to specify error states explicitly and estimate the distance from them [4]. This can be difficult in the case when the states are high dimensional partial observations like image data. We thus propose estimating where the policy is likely to mis-predict the expert's actions and have the expert take over to safely explore these regions.

In order to tell where a policy is likely to mis-predict,we leverage a property in statistical machine learning that a trained model will be able to generalize within the distribution it is trained on [22]. This property is used in importance sampling when data is collected from one distribution, but tested on another and has to be re-weighted to the test distribution [8]. One approach is to estimate the probability density the policy is trained on. However, the amount of data needed to accurately estimate the density scales exponentially in the number of [14].

Several other approaches have been proposed to tell when your estimator is likely to not generalize [**?**].Knox and Ng used a nearest neighbors approach to try and estimate if a point was close to k neighbors, however it was shown to be susceptible to outliers because nearest neighbor only uses local information about the data [11]. Manevitz and Yousef trained a neural network filter to try and reconstruct the data and if the reconstruction error is high, they mark the point as likely to be outside of the estimators generalization ability. Manevitz and Yousef mention that finding the right architecture can be task specific and require a lot of hyperparameter tuning [13]. The One Class SVM proposed by Scholkopf et al. estimates a user defined level set of the distribution it is trained on by solving a convex quadratic program to find the support vectors that encompasses the data. The method has been theoretically shown to approximate the level set of a density estimate in the asymptomatic limit of sample and also its regularization term has the nice property of being equivalent to the quantile function[23]. It has also been shown to succesfully work on outlier dectection for raw image data [**?**].

A. *Estimation of a Distribution's Level Set*

[TODO: GOING TO REWORD ONCE THE MODIFICATIONS ARE DONE AND MAKE IT EASIER TO READ] Support estimation can be defined as minimizing a function that contains the volume of a given probability distribution. Formally we define it as follows let $\mathbf{x}_1, ..., \mathbf{x}_n$ be i.i.d. random variables in a set $\mathcal{X}$ with distribution P. Let $\mathcal{G}$ be a class of measurable subsets of $\mathcal{X}$ and let λ be a real function defined on $\mathcal{G}$. The quantile function with respect to $(P, \lambda, \mathcal{G})$ is

$$U(\gamma) = \inf\{\lambda(G) : P(G) \geq \gamma, G \in \mathcal{G}\}\ 0 < \gamma \leq 1 \quad (1)$$

We denote by $G(\gamma)$ the $G \in \mathcal{G}$ that attains the infimum. The most common choice of λ is Lebesgue measure, in which case $G(\gamma)$ is the minimum volume $G \in \mathcal{G}$ that contains at least a fraction γ of the the probability mass. Thus, $G(1)$ is the support of the density p corresponding to P, assuming it exists.

To handle high dimensional densities P, work by Scholkopf et al. looked at representing the class $\mathcal{G}$ via a kernel k as the set of half-spaces in SV feature space [19]. They then minimize a SV style regularizor which, using a kernel, controls the smoothness of the estimated function describing G. In terms of the quantile function described in Eq. 1, the approach can be thought of as employing $\lambda(G) = ||w||^2$, where $G_w = \{x : f_w(x) \geq \rho\}$. Here (w, ρ) are a weight vector and offset parameterizing a hyperplane in the feature space associated with a kernel.

Let Φ be a feature map $\mathcal{X} \rightarrow \mathcal{F}$, i.e. a map into an inner product space F such that the inner product in the image of Φ can be computed by evaluating some kernel

$$k(\mathbf{x}, \mathbf{y}) = \Phi(\mathbf{x})^T \Phi(\mathbf{y}) \tag{2}$$

such as the Gaussian kernel

$$k(\mathbf{x}, \mathbf{y}) = e^{-||\mathbf{x}-\mathbf{y}||^2/c} \tag{3}$$

The main idea is to solve an optimization problem that separates the origin, we solve the following quadratic problem:

$$\min_{w \in F, \xi \in \mathcal{R}^l, \rho \in \mathcal{R}} \frac{1}{2}||w||^2 + \frac{1}{vl}\sum_i \xi_i - \rho \tag{4}$$

$$\text{s.t } (w^T \Phi(x_i)) \geq \rho - \xi_i, \; \xi_i \geq 0 \tag{5}$$

$$\tag{6}$$

The parameter ν controls the penalty on slack terms and asymptotically approaches is equivalent to γ [23]. The decision function then becomes

$$g(x) = \text{sgn}((w^T \Phi(x) - \rho) \tag{7}$$

Where $g(x) = 1$ indicates in the support region and $(g) = -1$ denotes being outside the support region. The dual form of the optimization takes the form of a Quadratic Program with respect to the Gram Matrix [19]. An example of this decision function can be seen in Fig. 1, where an expert drives a simulated race car around a track and the support of his demonstrations is estimated.

III. Preliminaries

A. *Notation and Background*

Define $\mathbf{x} \in \mathcal{X}$ as state space for a robot, for example $\mathbf{x}$ could be an image from a camera feed or a vector of joint angles. Define $\mathbf{u} \in \mathcal{U}$ as the class of controls for an agent, which could be for example torque control applied to a robot or discrete commands such as in a video game. The environment is modeled to have dynamics that are stationary and Markovian, such that the the probability of state $\mathbf{x_{t+1}}$ can be determined from $\mathbf{x}_t$ and $\mathbf{u}_t$, or $p(\mathbf{x}_{t+1}|\mathbf{u}_t, \mathbf{x}_t)$. We represent a probability distribution over the initial state of the environment $p(\mathbf{x}_0)$.. Denote a trajectory as series of states visited and the controls applied as $\tau = (\mathbf{x}_0, \mathbf{u}_1, ..., \mathbf{x}_T, \mathbf{u}_T)$. A trajectory that is defined with state space only and not controls is $\tau_{\mathbf{x}} = (\mathbf{x}_0, ..., \mathbf{x}_T)$.

A policy is a function from state space to controls, $\pi(\mathbf{x}) \rightarrow \mathbf{u}$. A policy could be $\pi_\theta(\mathbf{x}) \rightarrow \mathbf{u}$, which is a function that is specified by a vector of parameters θ.The vector of parameters θ can be for example weights in a neural network or the vectors in a Support Vector Machine. A policy, π_θ in an environment with $p(\mathbf{x}_0)$ and $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ induces a distribution on the possible trajectories:$p(\tau_{\mathbf{x}}|\theta) = p(\mathbf{x}_0)\prod_{i=0}^{T-1} p(\mathbf{x}_{t+1}|\pi_\theta(\mathbf{x}_t), \mathbf{x}_t)$. Also π_θ in an environment with $p(\mathbf{x}_0)$ and $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ induces a distribution on states visited $p(\mathbf{x}|\theta)$.

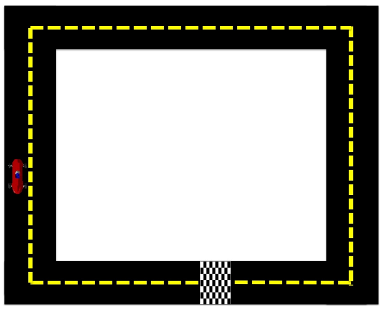
[TODO: THIS PARAGRAPH IS NOT NECCESARY TO READ BUT ITS HOW ONE GOES FROM $p(\tau_{\mathbf{x}}|\theta)$ TO $p(x|\theta)$.] Let $p(\mathbf{x}_t|\theta)$ be the distribution of states visited at time t if the robot follows the policy π_θ for $t-1$ time steps. Calculating this from $p(\tau_{\mathbf{x}}|\theta)$ is mean margnilizing over the previous timesteps $p(\mathbf{x}_t|\theta) = \int_{\mathbf{x}_{t-1}} ... \int_{\mathbf{x}_1} p((\mathbf{x}_t, ..., \mathbf{x}_1)|\theta) d\mathbf{x}_{t-1}...d\mathbf{x}_1$. The average distribution on states is now defined as $p(\mathbf{x}|\theta) = \frac{1}{T}\sum_{t=1}^T p(\mathbf{x}_t|\theta)$.[TODO: ENDS HERE]

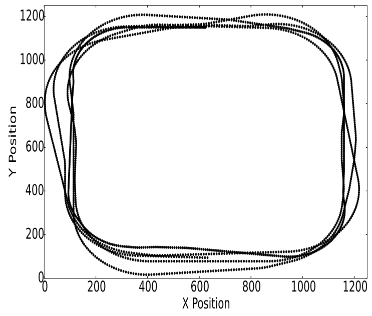
We do not know the distributions: $p(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$, $p(\mathbf{x}_0)$,$p(\tau_{\mathbf{x}}|\mathbf{x})$ or $p(\mathbf{x}|\theta)$. But we assume we have a stochastic real robot or a simulator such that given any state $\mathbf{x}_t$ and control $\mathbf{u}_t$, it will move into a state $\mathbf{x}_{t+1}$ such that $\mathbf{x}_{t+1} \sim p(\mathbf{x}_{t+1}|\pi_\theta(\mathbf{x}_t), \mathbf{x}_t)$. Note that $p(\tau_{\mathbf{x}}|\theta)$ is high dimensional because $\mathbf{x}$ is high dimensional.

Given π_θ we want to identify a set of highly probable trajectories and the associated high probable states. We assume we are given some initial state $\mathbf{x}_0$. We "roll-out" the policy using the robot with the policy to sample the resulting trajectory and set of states reached, which provides a sequence of highly probably states under $p(\tau_{\mathbf{x}}|\theta)$.Rather than estimating $p(\mathbf{x}|\theta)$ or $p(\tau_{\mathbf{x}}|\theta)$, we draw samples from them.

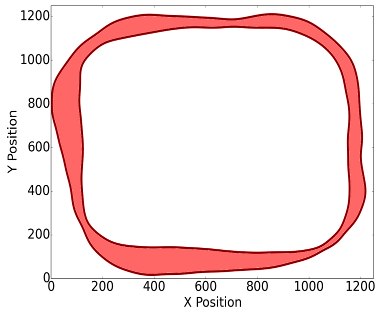
The objective in policy learning can be framed as minimizing some cost function $C(\tau) = \sum_{t=1}^T c(\mathbf{x}_t, \mathbf{u}_t)$. The cost function is user defined and task specific. For each state $\mathbf{x}_t$ and control $\mathbf{u}_t$ the robot receives a penalty related to how far it is from the goal state. For example, in task of inserting a peg into a hole the cost function can be distance to desired final state [12]. We also do not know the cost function. But we do have access to a "competent controller", an algorithm or human that uses some policy $\tilde{\pi}$ to minimize $C(\tau)$, and we are given an initial set of N training/demonstration trajectories resulting from this policy: $\mathcal{D} = \{\tilde{\tau}^1, ..., \tilde{\tau}^N\}$.

Next, we define a "surrogate" loss function l, which gives the distance between any pair of control values: in the continuous case $l(\mathbf{u}_0, \mathbf{u}_1) = ||\mathbf{u}_0 - \mathbf{u}_1||^2$ or in a discrete case


(a) Race Car Simulator



(b) Expert Demonstrations



(c) Estimated Support of Expert Demonstrations

Fig. 1: [TODO: WANT TO CHANGE THIS TO HIGHLIGHT ν EFFECT ON LEVEL SET ESTIMATION] A race car is driven around a track in Fig. 3(a) by an expert demonstrator. The state space of the car is represented as the x and y positions on the track or $\mathbf{s} = [x, y]$. In Fig. 3(b), the demonstrations are plotted in state space. In Fig. 1(c), the support is estimated using the decision function in 7. As illustrated the support accurately describes the boundary of the expert trajectory.

$$l(\mathbf{u}_0, \mathbf{u}_1) = \begin{cases} 1 & : \mathbf{u}_0 = \mathbf{u}_1 \\ 0 & : \mathbf{u}_0 \neq \mathbf{u}_1 \end{cases} \tag{8}$$

Given a candidate policy θ, we can use the surrogate loss function to compute how "close" it is to the competent controller's policy $\tilde{\pi}$.

We want to compute a policy π_θ that minimizes the expected surrogate loss where the expectation is taken over the distribution of states.

$$\min_\theta E_{p(\mathbf{x}|\theta)}[l(\pi_\theta(\mathbf{x}), \tilde{\pi}(\mathbf{x}))] \tag{9}$$

The optimization problem in Eq. 9 compares the action taken of the policy π_θ and the competent controller, $\tilde{\pi}$ weighted by the probability of the state occurring $p(\mathbf{x}|\theta)$.

Eq. 9 is hard to solve because the objective is not convex and we only have access to $\tilde{\pi}(x)$, where x is a state in one of the demonstrations or [TODO: CAN YOU PLEASE ELABORATE ON THE USE OF THE $\forall$ SYMBOL HERE]$\mathbf{x} \in \mathcal{D}$. To address these issues Stephane and Ross proposed an iterative method known as DAgger[17].

B. DAgger: Dataset Aggregation

DAgger is an iterative algorithm that repeats two steps for K iterations. In Eq. 9, the objective involves both the parameterized policy π_θ and the average distribution of states with respect to the policy, $p(\mathbf{x}|\theta)$. DAgger iteratively minimizes π_θ on the collected dataset of examples $\mathcal{D}$ and then adds new states of high probability under π_θ by sampling via policy roll outs from $p(\mathbf{x}|\theta)$.

1) Step 1: At iteration $k = 0$ given a dataset $\mathcal{D}_0$, the training set with N_0 demonstrated trajectories mapping $\mathbf{x}$ to $\mathbf{u}$. DAgger solves the following supervised learning problem with respect to the surrogate loss function. [TODO: YOU MENTIONED CHANGING $\theta_{k=0}$ TO $\theta_{k=1}$, BUT ON THE RIGHT HAND SIDE OF THE EQUATION θ IS A VARIABLE AND THE LEFT HAND SIDE ITS A SPECIFIC VALUE FOUND BY THE OPTIMIZATION, SO WE NEED THEM TO BE THE SAME. THE UPDATE IN MY MIND COMES AT THE END OF STEP 2]

$$\theta_{k=0} = \arg\min_{\theta_{k=0}} \sum_{i=1}^{N} \sum_{t=1}^{T} l(\pi_{\theta_{k=0}}(\mathbf{x}_t^i), \mathbf{u}_t^i) \tag{10}$$

This is a supervised learning problem and can be solved using standard machine learning techniques, for example a support vector machine or a neural network. And the surrogate losses mentioned early like Squared Euclidean distance or an Indicator function can can be appropriate optimization objective.

To handle the fact that the competent actor's policy contains noise ϵ, we employ regularization techniques in the optimization. Regularization is a technique to control the smoothness of the function that is fit to the sampled data. It is a common technique to make the estimator robust to zero mean noise on data. In practice it is a penalty term on either the L2 norm on the weights for regression based techniques or the slack coefficient for support vector machines [**?**].

2) Step 2: DAgger rolls out the policy $\pi_{\theta_{k=0}}$ to sample states that are likely to occur under $\theta_{k=0}$. For every state visited, DAgger requests the competent controller to provide the appropriate control/label. This can be very tedious for a human. To creates additional demonstrations to aggregate into the next dataset of demonstrations $\mathcal{D}_1$.

[TODO: MORE NOTATION HEAVY EXPLANATION] The trained policy π_{θ^k} has an associated average distribution of states $p(\mathbf{x}|\theta_k)$. In order to improve upon the policy the competent actor needs to provide feedback on states that occur with high probability under this distribution. Since we do not know $p(\mathbf{x}|\theta_k)$, we need to draw samples from it to determine what states occur with high probability. Drawing sample from $p(\mathbf{x}|\theta_k)$ involves rolling out the policy, or sampling an initial state $\mathbf{x}_1 \sim p(\mathbf{x}_1)$ and then executing action $\mathbf{u}_1 = \pi(\mathbf{x}_1)$ and entering state $\mathbf{x}_2$ this is repeated for T timesteps. The sampled trajectory is $\tau_\theta^k = (\mathbf{x}_1^k, \mathbf{u}_1^k, ..., \mathbf{x}_T^k, \mathbf{u}_T^k)$. The competent actor then provides feedback by labeling each state with the correct action $\tau^k = (\mathbf{x}_1^k, \tilde{\pi}(\mathbf{x}_1^k), ..., \mathbf{x}_T^k, \tilde{\pi}(\mathbf{x}_T^k))$. The dataset $\mathcal{D}_0$ is then updated via $\mathcal{D}_1 = \mathcal{D}_0 \cup \tau^k_{\hat{\theta}}$ and Step 1 is repeated. [TODO: ENDS HERE]

Steps 1 and 2 are repeated for K iterations or until the cumulative surrogate loss goes to zero. Since DAggger requires rolling out the policy at each iteration, one way to terminate the algorithm is when the performance of the rolled out policy matches the intended behavior, or more formally when the surrogate loss is below some predefined threshold γ.

Dagger works well in both simulation and real world experiments. It also has theoretical guarantees that it will converge in the limit to the expert's policy. However, using policy rollouts means the robot can enter dangerous states, which could damage a robot in a non simulation system. For example, when DAgger was used on a quad-copter the robot repeatedly crashed into trees before successfully learning to avoid them [18]. A state can be unsafe for 2 reasons: 1) its is far from the states we've trained on, which potentially means our policy will mis-predict the expert and occur high surrogate loss [22] 2) the surrogate loss assoicated with the state is high, so the state, although visited, doesn't have good control yet.

Furthermore iterations are tedious because if a human is the competent controller requires manual "labeling" of every state visited with the appropriate control. [TODO: IN THE LAST PARAGRAPH I COPIED THE NOTES SLIGHTLY DIFFERENT. FIRST I LISTED AN EXAMPLE OF A DANGEROUS STATE TO MAKE IT MORE CONCRETE AND SECOND I MADE IT CLEAR THAT THE COMPETENT CONTROLLER MIGHT NOT BE A HUMAN, WHICH COULD MAKE THE SECOND STEP EASIER LIKE IN MARIO]

To address these limitations, we propose to 1) avoid highly unsafe states entirely and 2) recognize slightly unsafe sights and request human input for these and to reduce the burden on the human trainer. One way to do this is to define safety in terms of distance to the nearest neighbor, but this vulnerable to outliers in the known set of states. Another option is to fit a distribution to the known set of states, which requires an exponential number of samples in the dimension of the state space [14]. We propose using a technique known as the One Class SVM that estimates a boundary of the density [19].

C. SHEATH

Our proposed method now not only trains a policy, π_{θ^k}, but also estimates where the policy is likely to accurately predict the competent actor,which we define as the safe region. In the statistics community, a known property is that estimators are function will mis-classify if they are evaluated at points not in the distribution sampled from.Techniques such as importance sampling use this results to train estimators on data drawn from a different distribution [22]. Thus, we try to estimate the a user-defined level set of the training distribution, which is explained in II-A and allow the competent actor to take over when the the robot has crossed this boundary. Then the expert will demonstrate the intended behave and guide the robot back to the safe region. Each of the two steps are explained in detail.

1) Step 1: Similar to DAgger at iteration k the algorithm estimates a policy, π_{θ^k} on the dataset $\mathcal{D}^k$ by solving Eq. 10. We then estimate the level set of the distribution being trained on by solving the optimization for the One Class SVM or 4 on the dataset $\mathcal{D}^k$. We now have a decision boundary $g^k(\mathbf{x})$, which is 1 if the region is in the safe area and 0 otherwise . The parameter ν is an estimate of how much probability mass is contained in the level set and can be use to control the size of the safe region.

2) Step 2: In DAgger the policy that was rolled out is π_{θ^k}, however this could potential encounter unsafe states. Using our notation of safety as being where we are likely to mispredict the expert, we instead roll out π^k, where $\pi^k(\mathbf{x}) = g^k(\mathbf{x})\pi^k_\theta(\mathbf{x}) + (1 - g^k(\mathbf{x}))\pi^k_{\hat{\theta}}(\mathbf{x})$. Thus if the policy leaves the safe region, then we ask the competent actor to take over and if it is in the safe region the robot remains in control. The sampled trajectory is then $\tau^k_\theta = (\mathbf{x}^k_1, \mathbf{u}^k_1, ..., \mathbf{x}^k_T, \mathbf{u}^k_T)$. The competent actor then provides feedback by labeling each state with the correct action $\tau^k_{\hat{\theta}} = (\mathbf{x}^k_1, \pi_{\hat{\theta}}(\mathbf{x}^k_1), ..., \mathbf{x}^k_T, \pi_{\hat{\theta}}(\mathbf{x}^k_T))$. The dataset $\mathcal{D}^{k+1}$ is then updated via $\mathcal{D}^{k+1} = \mathcal{D}^k \cup \tau^k_{\hat{\theta}}$ and Step 1 is repeated. [TODO: WE ARE WORKING ON REMOVING THE LAST STEP OF HAVING TO LABEL POINTS IN THE SAFE REGION, WE CAN DISCUSS THESE IDEAS WEDNESDAY]

An implicit trade off exists between what how much of the distribution is contained in the selected level set and the amount of exploration the robot is allowed. If the level set is large the robot will be allowed to explore in this region and potentially make more mistakes with respect to the expert, but learn faster. If the level set contains a smaller fraction of data, then the expert will be in control more and the robot will learn at a slower rate.

IV. Theoretical Analysis

[TODO: UPDATING THIS SECTION WITH NEW NOTATION, WILL BE FIXED TOMORROW]

Our goal is to find a the parameters θ which minimizes the observed surrogate loss under its induced distributions, (i.e.):

$$\theta = \arg\min_{\pi} E_{\pi_\theta}[l(\mathbf{x}, \pi)] \tag{11}$$

As system dynamics are assumed both unknown and complex, we cannot compute π_θ and can only sample it by executing τ, where $\tau \sim \pi_\theta$ on the system. Hence this is a non-i.i.d supervised learning problem due to the dependence of the input distribution on the policy π_θ itself. The interaction between policy and the resulting distribution makes optimization difficult as a results in a non-convex objective even if the loss $l(\mathbf{x}, *)$ is convex in π_θ for all states $\mathbf{x}$.

In the classification case, we have a $l((\mathbf{x}, \pi) = \sum_s (I(\pi_\theta(\mathbf{x}_t) \neq \pi_E(\mathbf{x}_t)))$, or an indicator function that is 1 if the policy is a mismatch and zero otherwise. In this case the cost function L is equivalent to matching the expert policy, thus $L(\mathbf{x}, \pi) = l(\mathbf{x}, \pi)$. Given a level set estimation function $g(s) \rightarrow \{0, 1\}$, we are interesting in bounding the cost of executing policy $\pi = g(s)\pi_\theta + (1 - g(s))\pi_E$, $J(\pi) = TE_\pi[l(s)]$.

Denote the expected loss of the policy π mimicking the expert under its own distribution of states $E_{d_\pi} = E_{s\sim d_\pi, a\sim\pi^*(s)}[l(s,a,\pi)]$. The dataset the policy π_θ was trained on is given by $\mathcal{D}$, which is the aggregrate of all data seen so far. Thus, we define the loss on this dataset as $E_{\mathcal{D},\pi_\theta} = \sum_{\mathbf{x}} l(\mathbf{x},\pi_\theta)d_{\pi_\theta}$ and the distribution it encounters at test time for iteration i is denoted as d_π. We now define ξ as the error in our level set estimation function $g(x) \rightarrow \{0,1\}$.

The density of d_{π_i} can be defined now for some $\alpha \in [0,1]$:

$$d_{\pi_i} = (1-\alpha)d_{\hat{\pi}} + \alpha d \tag{12}$$

A. Safe Learning Guarantees

In this work we are interested in interested in determining the expected loss per iteration on the policy rolled out π_i or $E_{d_{\pi_i}} = E_{s\sim d_{\pi_i}, a\sim\pi^*(s)}[l(s,a,\hat{\pi}_i)]$. Let $\epsilon_N = \frac{1}{N}\sum_{n=1}^{N} E_{\mathcal{D},\pi_\theta}$ or the average error encountered on the known data set.

In order to solve this proof we must use the following lemma

Lemma 4.1: Using A.H.U.S.E., where the error in $g(x) \rightarrow \{0,1\}$ is ξ

$$E_{d_\pi} \leq E_{\mathcal{D},\pi_\theta} + \xi \tag{13}$$

Proof:

$$\begin{aligned}
E_{d_{\pi_i}} =& \sum_x l(\mathbf{x},\pi)d_{\pi_i} \\
=& \sum_{\mathbf{x}}((1-\alpha)d_{\pi_\theta^i} + \alpha d)l(\mathbf{x},\pi) \\
=& \sum_{\mathbf{x}}(1-\alpha)d_{\hat{\pi}}l(\mathbf{x},\pi) + \sum_x \alpha d l(\mathbf{x},\pi) \\
=& (1-\alpha)E_{\mathcal{D},\pi^i_{theta}} + \alpha\xi \\
\leq& E_{\mathcal{D},\pi^i_{theta}} + \xi
\end{aligned}$$

■

Theorem 4.2: Using A.H.U.S.E., where the error in $g(x) \rightarrow \{0,1\}$ is ξ and the cost function $C = l$ or is upper bound by the surrogate loss function.

$$J(\pi) \leq T[\epsilon_N + \xi] \tag{14}$$

Proof:

We first demote that $J(\pi) \leq J(\bar{\pi})$, since $J(\pi) = \min_{i=1...N} J(\pi_\theta^i)$ and $J(\bar{\pi}) = \frac{1}{N}\sum_{i=1}^{N} J(\hat{\pi}_i)$. Since, the average is greater than or equal to the minimum.

$$\begin{aligned}
J(\bar{\pi}) =& \frac{1}{N}\sum_{i=1}^{N} J(\pi_i) \\
=& \frac{1}{N}\sum_{i=1}^{N} TE_{d_{\pi_i}} \\
\leq& \frac{1}{N}\sum_{i=1}^{N} T(E_{d_{\hat{\pi}}} + \xi) \\
\leq& T(\epsilon_N + \xi)
\end{aligned}$$

■

B. Convergence Rate

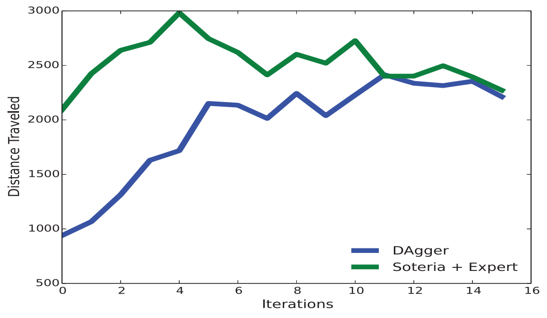
[TODO: THESE ARE IDEAS STILL IN PROGRESS] The convergance rate of DAgger is dependent on the probability of the expert taking over at each iteration β_i [17]. For Soteria, this corresponds to the probability of leaving the support or $p(g(\mathbf{x}) = -1|\pi_{\theta_i})$. The intuition is that the larger the level set parameter ν is the lower the probability of leaving is and thus faster convergence rate, but potentially less safe exploration. Not sure if a formal proof is going to be possible though with assuming known dynamics, still looking into it. However, we can experimentally demonstrate this.

V. Simulation Experiments

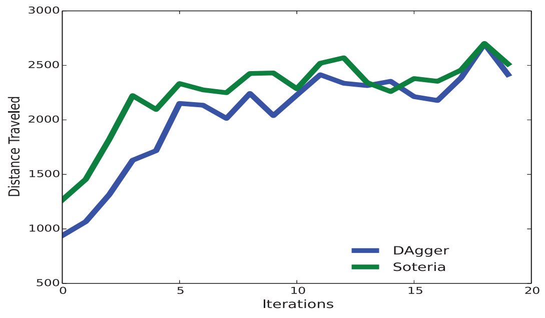
A. Super Mario Bros

Super Mario Bros. is a platform video game where the character, Mario, must move across each stage by avoiding being hit by enemies and falling into gaps, and before running out of time. We used the simulator from a recent Mario Bros. AI competition which can randomly generate stages of varying difficulty (more difficult gaps and types of enemies). Our goal is to train the computer to play the game based on current game features as input. Our expert is a near optimal A^* search algorithm that plans ahead using the game simulator. An action consists of 4 binary variables indicating which subset of buttons we should press in {left, right, jump, speed}. For both DAgger and Soteria, we use a Linear Support Vector Machine (SVM) as the policy representation, with the regularization term on the slack variable $C = 0.01$, which was set via cross validation on the initial training examples from our expert.. We initialized both Dagger and A.H.U.S.E with two expert demonstrations for each level. For the support estimation in Soteria we used a radial basis function as well for the kernel with a bandwidth of $c = 0.01$.

We compare performance in terms of average distance traveled by Mario per stage before dying, running out of time or completing the stage, on randomly generated stages of difficulty 1 with a time limit of 35s to complete the stage. Stages of difficulty 1 are fairly easy for an average human player but contain most types of enemies and gaps, except with fewer enemies and gaps than stages of harder difficulties. We compare performance of our method and



(a) Safe Learning with Policy, π



(b) Learning Rate of Policy, π_θ

Fig. 3: We compare performance in terms of average distance traveled by Mario per stage before dying, running out of time or completing the stage, on randomly generated stages of difficulty 1 with a time limit of 35s to complete the stage. Stages of difficulty 1 are fairly easy for an average human player but contain most types of enemies and gaps, except with fewer enemies and gaps than stages of harder difficulties. We compare performance of our method and DAgger. In Fig. 3, we report the performance of distance traveled vs. the amount of time a level was tried. We show the performance of Soteria's policy π during training and the underlying policiy π_θ. As illustrated, Soteria was able to learn at roughly the same rate as DAgger, but during learning approximately maintained the same reward as the converged DAgger policy. [TODO: WE HAVE BEEN STRUGGLING TO CONVEY THESE RESULTS BECAUSE EVEN THE EXPERT EVENTUALLY DIES SINCE THE GAME GOES ON FOREVER. THE IDEA IS THAT PERFORMANCE IS MEASURED IN DISCTANCE TRAVELED AND OUR METHOD MAINTAINS THE PERFORMANCE OF CONVERGED DAGGER THROUGH OUT TRAINING. THE UNDERLYING POLICY THOUGH LEARNS AT THE SAME RATE.]

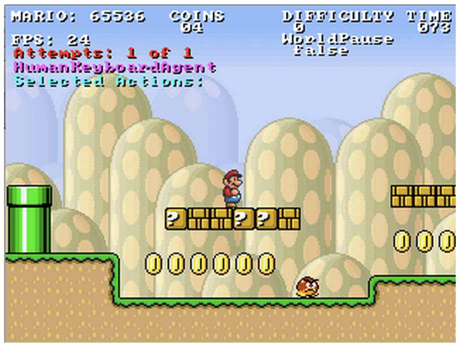


Fig. 2: An example image from the Mario AI simulator. Mario faces many challenges to reach an end of a level, such as gaps, jumps and enemies.

DAgger. In Fig. 3, we report the performance of distance traveled vs. the amount of time a level was tried. We show the performance of Soteria's policy π during training and the underlying policiy π_θ. As illustrated, Soteria was able to learn at roughly the same rate as DAgger, but during learning approximately maintained the same reward as the converged DAgger policy.

B. Driving Example

A common benchmark in RL is a car driving around a track avoiding other cars [TODO: CITE ALL]. In our driving simulator the car has dynamics $\mathbf{x}_{t+1} = A\mathbf{x}_t + Bu_t$, where $\mathbf{x} = [x\ y\ \theta]^T$. Our actions space $u = \{-15°, 0, 15°\}$. We have a human drive around the track 3 times and then collect the raw images observed. We use a Gaussian Pyramids to downsample the images to 125×125 RGB pixels.

[TODO: RIGHT NOW WE ARE TRYING TO GET THE NEURAL NETWORK TO WORK, IF WE CAN'T BY FRIDAY EVENING. I AM JUST GOING TO SWITCH TO HOG FEATURES WITH AN SVM] We feed the images into a feed-forward network with the same architecture used in [21]. The net consists of a convolution layer with 16 filters and two hidden

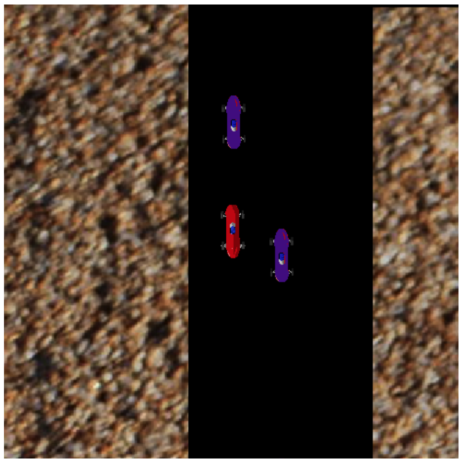


Fig. 4: An example image from the racing simulator. The car is meant to drive around a rectangular track and dodge other cars. The expert drives around the track for 3 iterations and the images are feed into a feedforward network to learn a policy.

layers seperated by ReLu units. We train the net with a Softmax loss function, which is common for classification tasks. The neural network was trained using Caffe [9].

C. Other Experiment

[TODO: FLORIAN AND I ARE DISCUSSING DIFFERENT IDEAS FOR THIS ONE, MAYBE A CONTROLLER FOR AN ARM OR SOMETHING ALONG THE LINES OF LEARNING TO GRASP IN CLUTTER BUT NOT KNOCKING OVER ANY OBJECTS]

VI. Discussion and Future Work

References

[1] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng, "An application of reinforcement learning to aerobatic helicopter flight," *Advances in neural information processing systems*, vol. 19, p. 1, 2007.

[2] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, “A survey of robot learning from demonstration,” *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.

[3] A. Billard and M. J. Matarić, “Learning human arm movements by imitation:: Evaluation of a biologically inspired connectionist architecture,” *Robotics and Autonomous Systems*, vol. 37, no. 2, pp. 145–160, 2001.

[4] J. Garcia and F. Fernández, “Safe exploration of state and action spaces in reinforcement learning,” *Journal of Artificial Intelligence Research*, pp. 515–564, 2012.

[5] J. H. Gillula, “Reducing conservativeness in safety guarantees by learning disturbances online: iterated guaranteed safe online learning,” *Robotics*, p. 81, 2013.

[6] J. H. Gillula and C. J. Tomlin, “Guaranteed safe online learning via reachability: tracking a ground target using a quadrotor,” in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 2723–2730.

[7] A. Hans, D. Schneegaß, A. M. Schäfer, and S. Udluft, “Safe exploration for reinforcement learning.” in *ESANN*, 2008, pp. 143–148.

[8] J. Huang, A. Gretton, K. M. Borgwardt, B. Schölkopf, and A. J. Smola, “Correcting sample selection bias by unlabeled data,” in *Advances in neural information processing systems*, 2006, pp. 601–608.

[9] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional architecture for fast feature embedding,” *arXiv preprint arXiv:1408.5093*, 2014.

[10] L. P. Kaelbling, M. L. Littman, and A. W. Moore, “Reinforcement learning: A,” *Journal of artificial intelligence research*, pp. 237–285, 1996.

[11] E. M. Knox and R. T. Ng, “Algorithms for mining distancebased outliers in large datasets.” Citeseer.

[12] S. Levine, C. Finn, T. Darrell, and P. Abbeel, “End-to-end training of deep visuomotor policies,” *arXiv preprint arXiv:1504.00702*, 2015.

[13] L. M. Manevitz and M. Yousef, “One-class svms for document classification,” *the Journal of machine Learning research*, vol. 2, pp. 139–154, 2002.

[14] E. A. Nadaraya, “On estimating regression,” *Theory of Probability & Its Applications*, vol. 9, no. 1, pp. 141–142, 1964.

[15] D. A. Pomerleau, “Alvinn: An autonomous land vehicle in a neural network,” DTIC Document, Tech. Rep., 1989.

[16] S. Ross and D. Bagnell, “Efficient reductions for imitation learning,” in *International Conference on Artificial Intelligence and Statistics*, 2010, pp. 661–668.

[17] S. Ross, G. J. Gordon, and J. A. Bagnell, “A reduction of imitation learning and structured prediction to no-regret online learning,” *arXiv preprint arXiv:1011.0686*, 2010.

[18] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert, “Learning monocular reactive uav control in cluttered natural environments,” in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 1765–1772.

[19] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, “Estimating the support of a high-dimensional distribution,” *Neural computation*, vol. 13, no. 7, pp. 1443–1471, 2001.

[20] J. Schulman, A. Gupta, S. Venkatesan, M. Tayson-Frederick, and P. Abbeel, “A case study of trajectory transfer through non-rigid registration for a simplified suturing scenario,” in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 4111–4117.

[21] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, “Trust region policy optimization,” *arXiv preprint arXiv:1502.05477*, 2015.

[22] S. T. Tokdar and R. E. Kass, “Importance sampling: a review,” *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 1, pp. 54–60, 2010.

[23] R. Vert and J.-P. Vert, “Consistency and convergence rates of one-class svms and related algorithms,” *The Journal of Machine Learning Research*, vol. 7, pp. 817–854, 2006.