

Constraint Learning with Inverse Reinforcement Learning

Author: Erfan Etesami

Supervisors: Baiyu Peng, Professor Aude Billard

I. INTRODUCTION

In the context of reinforcement learning (RL), reward functions play a crucial role as they are specifically designed to encourage the RL agent to achieve a desired behavior. However, blindly optimizing a reward function can result in poor real-world performance in case the environment is constrained. As such, the RL agent needs to be trained to maximize its reward function while abiding by the constraints.

The main goal of this work is to learn the constraints present in an environment utilizing given a set of expert demonstrations. This project, thus, is a variant of inverse reinforcement learning (IRL), where, unlike typical IRL, the reward function is explicitly known and injects prior knowledge about the agent's task. Here, the main assumption is that the expert demonstrations are a set of trajectories in which the reward is maximized while the constraints are respected. Therefore, if there is a trajectory with a high reward that the expert did not take, that trajectory must have violated some constraints in the environment.

This research project is mainly built upon the work done in [1], which aims to retrieve constraints of arbitrary complexity in a model-free continuous setting for both states and actions. In this report, states and actions are presented by $s \in \mathcal{S} \subset \mathbb{R}^n$ and $a \in \mathcal{A} \subset \mathbb{R}^m$, where n and m are the dimensions of state and action spaces, respectively. Moreover, γ is representative of the discount factor, and the real-valued reward is denoted by r . Furthermore, the agent is assumed to interact with the environment in a series of episodes with length T . The set $\tau = \{s_1, a_1, \dots, s_T, a_T\}$ is thus representative of a trajectory containing a sequence of state-action pairs in one episode. In addition, a policy, the mapping between states and actions, is denoted by $\pi(a|s)$ or $\pi(s, a)$.

For the trajectory τ , its probability under the policy π and the total discounted reward are defined as $\pi(\tau)$ and $r(\tau) = \sum_{t=1}^T \gamma^t r(s_t, a_t)$. Moreover, the constrained environment is assumed to be Markovian, in which the total discounted cost of the trajectory is denoted by $c(\tau) = \sum_{t=1}^T \gamma^t c(s_t, a_t)$, where $c : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}_+$ is the cost associated with violating constraints.

Therefore, the aim of a forward RL agent is to find the optimal π_c^* maximizing the expected total discounted reward $J(\pi) = \mathbb{E}_{\tau \sim \pi} \{r(\tau)\}$ while having the expected total discounted cost $J^c(\pi) = \mathbb{E}_{\tau \sim \pi} \{c(\tau)\} \leq \alpha$, where $\alpha \geq 0$ is the budget assigned to the agent for constraint violation. More specifically, $\alpha > 0$ allows the agent to take actions and visit states as long as $J^c(\pi) \leq \alpha$, whereas $\alpha = 0$ is associated with a situation where the agent cannot visit any state-action

pair with non-zero cost. As such, the constrained state-action pairs can have any positive cost. More information is provided about this formulation in [section II](#).

II. MAIN FRAMEWORK

As illustrated in [fig. 1](#), the framework for retrieving the constraints is composed of two consecutive steps of learning policy and learning constraints at each iteration. The forward RL policy is parameterized by ϕ and consists of two hidden layers with 16 units alongside the tanh activation function. In the backward step, the binary classifier ζ_θ is responsible for inferring whether the given state-action pairs are feasible or infeasible. The associated cost is subsequently defined as $c = \bar{\zeta}_\theta = 1 - \zeta_\theta$. As it will be detailed later, ζ_θ can be modeled by a neural network or a Gaussian mixture model (GMM). Note that in the current stage of this work, the cost $\bar{\zeta}_\theta$ is always binarized by comparing its value with a predetermined threshold such that $\bar{\zeta}_\theta(s_t, a_t) \in \{0, 1\}$, where 1 is the cost of visiting a constrained (infeasible) state-action pair.

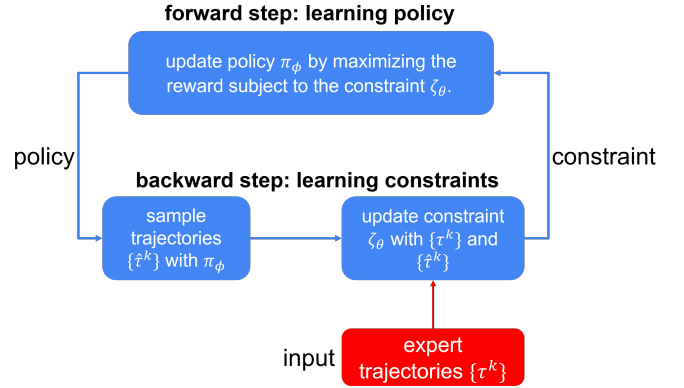


Fig. 1. The diagram of the two-step iterative framework of learning constraints.

As mentioned earlier, given a set of expert demonstrations, the goal of this work is to learn the constraint set $\mathcal{C}$ as an IRL problem where the reward is known. The set of expert trajectories is denoted as $\mathcal{D} = \{\tau^1, \dots, \tau^k\}$. These expert demonstrations are ideally sampled from the optimal constrained policy π_c^* defined earlier. Overall, the proposed two-step framework aims to recover constraints while the policy converges to the provided expert demonstrations. Nonetheless, it has been observed that it is possible for the algorithm to learn constraints similar to the true ones while the policy is not yet close to the expert trajectories. It is also noteworthy

that as the trajectories sampled from the policy become closer to the expert demonstrations, recovering constraints becomes harder since there would be fewer conflicts between the two sets of trajectories.

As to the forward step, [1] suggests solving the unconstrained min-max problem formulated as

$$\min_{\lambda \geq 0} \max_{\phi} J(\pi_{\phi}) + \frac{1}{\beta} \mathcal{H}(\pi_{\phi}) - \lambda (\mathbb{E}_{\hat{\tau} \sim \pi_{\phi}} \{\bar{\zeta}_{\theta}(\hat{\tau})\} - \alpha) \quad (1)$$

by employing the proximal policy optimization (PPO) algorithm [2]. Here, $\mathbb{E}_{\hat{\tau} \sim \pi_{\phi}} \{\bar{\zeta}_{\theta}(\hat{\tau})\}$ is equivalent to $J^c(\pi) \leq \alpha$ defined earlier. Moreover, $\mathcal{H}(\pi_{\phi}) = -\mathbb{E}_{\hat{\tau} \sim \pi_{\phi}} \{\log \pi_{\phi}(\hat{\tau})\}$ is the entropy measure. Increasing the entropy coefficient ($\frac{1}{\beta}$) indirectly encourages more exploration.

Furthermore, λ in eqn. (1) is the Lagrangian parameter which assigns a penalty weight to the amount of constraint violation. Learning λ requires some inner loops inside the forward step that makes the proposed framework unstable in practice. Thus, the λ is set to a fixed value in this work. As such, the budget α does not affect the gradient of the objective since $\lambda \cdot \alpha$ will only be a constant term.

III. CONSTRAINT NEURAL NETWORK

The binary classifier ζ_{θ} is initially modeled with a neural network whose inputs are the states and output is the scalar determining the feasibility of the states. It is noteworthy that the input can be extended to include actions if one wishes to recover constraints in the action space. Moreover, the leaky ReLU function is chosen as the intermediate activation between hidden layers, whereas the output activation is a sigmoid function that ensures having $0 \leq \zeta_{\theta} \leq 1$.

The maximum likelihood (ML) loss function using sample-based approximation is defined as (see [1])

$$\mathcal{L}(\theta) \approx \frac{1}{N} \sum_{i=1}^N \log \zeta_{\theta}(\tau^{(i)}) - \frac{1}{M} \sum_{j=1}^M \log \zeta_{\theta}(\hat{\tau}^{(j)}), \quad (2)$$

where τ and $\hat{\tau}$ correspond to expert demonstration and trajectories sampled from the policy, respectively. Another choice of loss function which works better in practice is the binary cross entropy (BCE) loss formulated as

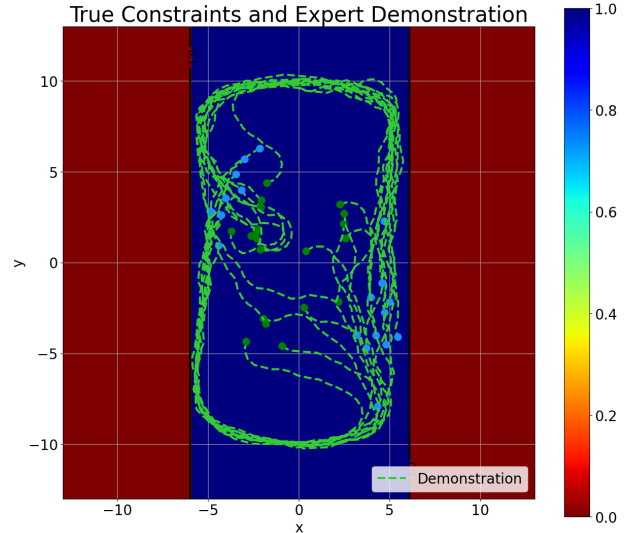
$$\mathcal{L}(\theta) \approx \frac{1}{N} \sum_{i=1}^N \log \zeta_{\theta}(\tau^{(i)}) + \frac{1}{M} \sum_{j=1}^M \log (1 - \zeta_{\theta}(\hat{\tau}^{(j)})). \quad (3)$$

During this project, several experiments have been done to test various network architectures. For instance, it has been observed that using the sigmoid function as an intermediate activation function significantly slows down the training and might prevent the network to learn any constraints. Furthermore, batch normalization results in a very narrow range of output that completely deteriorates the constraint learning and binarization of the cost. Testing different numbers of hidden layers and units has not also made any meaningful difference in the final outcome. One can also consider different weights for the loss associated with the expert demonstrations and policy trajectories described in eqns. (2) and (3). However, this

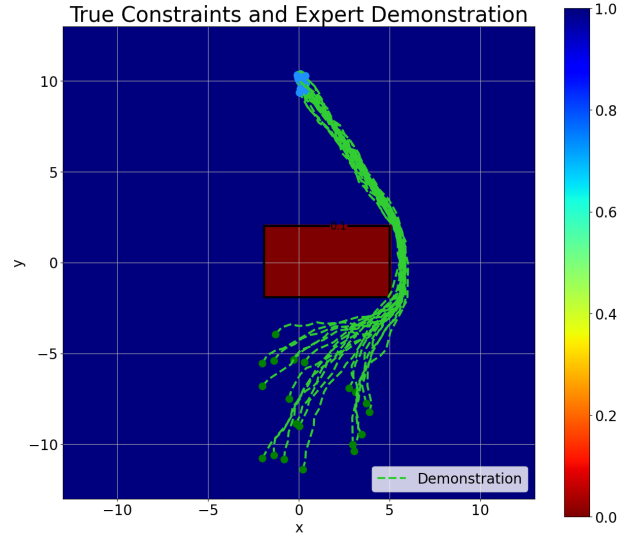
only introduces new hyperparameters to be tuned without any essential improvement. Nonetheless, regularizing both the ML and BCE loss plays a significant role in learning the minimal representation of the constraints (see section V-F).

IV. ENVIRONMENTS

Two environments, namely point-circle and point-obstacle, have been employed for examining the performance of the two-step framework. The states in both of these environments are comprised of positional x and y coordinates and the heading angle ψ . The actions are the magnitude of the linear velocity ($\|v\|_2$) and the scalar angular velocity (ω). Moreover, the states and actions are assumed to be continuous.



(a) point-circle environment.



(b) point-obstacle environment.

Fig. 2. The point-circle and point-obstacle environments in the x - y plane. The red region is representative of true constraints, and the blue-colored area corresponds to the feasible region. The start and end point of each expert demonstration is shown as green and blue dots, respectively. The number 0.1 in the contour corresponds to the binarization threshold. The color bar is associated with the value of ζ_{θ} .

In the point-circle environment, the agent's task is to follow a circle with a radius of 10 (the unit is arbitrary) in a clockwise trend. The normalized reward is thus designed as

$$r = \frac{ydx - xdy}{(1 + |\sqrt{x^2 + y^2} - 10|) \cdot (\sqrt{x^2 + y^2})}, \quad (4)$$

where $dx = \|v\|_2 \cos(\psi)$ and $dy = \|v\|_2 \sin(\psi)$.

In the point-obstacle environment, the aim is to encourage the agent to go from the initial position to the goal location G . The reward in this environment is defined as

$$r = -\frac{\sqrt{(x - G_x)^2 + (y - G_y)^2}}{d}, \quad (5)$$

where $d = 20$ is a normalization factor.

Fig. 2 illustrates point-circle and point-obstacle environment in the x - y plane. True constraint in the point-circle environment consists of two walls at $x = \pm 6$ location, which prevents the full circular motion encouraged by the reward. In the point obstacle environment, there is a rectangular obstacle blocking the path from the agent's initial position to the target location. Additionally, it is assumed that there is a known wall to the left of this obstacle. Note that the number 0.1, shown in fig. 2, is a hyperparameter used to consider states with $\zeta_\theta \geq 0.1$ as feasible. Note that the task in both environments is episodic with the length of 250 and 175 time steps in the point-circle and point-obstacle environments, respectively. Moreover, each trajectory is initialized randomly within a certain range at the beginning of each episode.

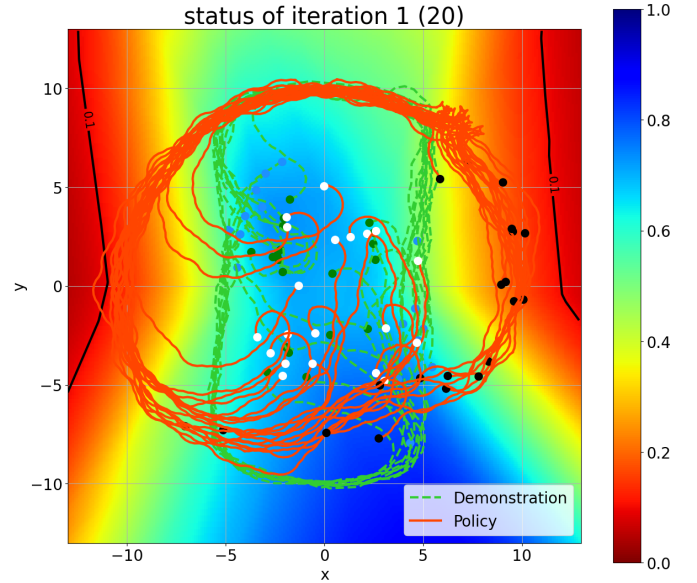
It is noteworthy that while training the agent in the environment, the episode is not terminated upon constraint violation; instead, the RL agent continues interacting with the environment until the maximum allowed time steps is reached in each episode. The reason is, this way, the entire trajectory can be used to make assumptions on the feasibility of its different segments. Moreover, if the agent violates the currently learned constraints in the environment, it will not directly receive a negative reward from the environment. This is desirable since it lets the agent explore different parts of the state space while recording the whole trajectory instead of discouraging it with a negative reward.

Furthermore, keeping the explicitly known reward separate from the constraint representation works better in practice. More specifically, at the beginning of the training, the learned constraints are not yet close to the true ones; therefore, using them directly in the reward might cause the agent to learn false constraints. Nonetheless, the cost function $\tilde{\zeta}_\theta$ informs the policy about the amount of constraint violation (see eqn. (1)). Furthermore, the cost function parameters will be updated at each iteration to represent the constraints existing in the environment better.

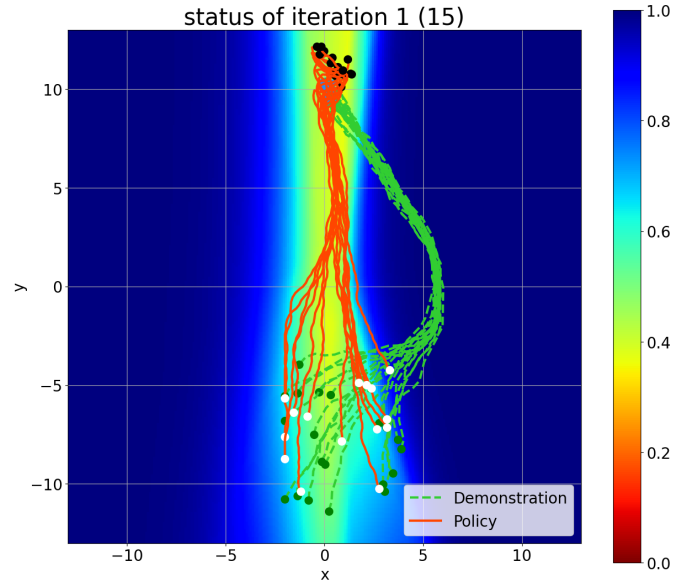
V. PRACTICAL MECHANISMS

A. Warmed-up Policy

Training policy in the forward step of the framework (see fig. 1) starts from a randomly initialized policy in [1].



(a) point-circle environment.



(b) point-obstacle environment.

Fig. 3. The warmed-up policy in point-circle and point-obstacle environments is shown in orange. The start and end point of each expert demonstration is shown as green and blue dots, respectively. The start and end points of policy trajectories are shown in white and black, respectively. The color bar is associated with the value of ζ_θ .

This, however, results in an unstable learning process. More concretely, the agent cannot learn any constraint at the early stages of training, and a random policy significantly affects the landscape of the recovered constraints. In this work, it is proposed to divide the training process into two phases. More specifically, one first ignores the existence of any constraint in the environment and, thus, trains the policy in order to maximize the reward. This mechanism, called the warmed-up policy, is only done once for each environment. In other words, the warmed-up policy is saved and then just loaded at the beginning of each run. As such, after loading the warmed-

up policy, the training proceeds by first a backward step to update the representation of constraints and then a forward step to update the policy accordingly.

The warmed-up policies for both point-circle and point-obstacle environments are presented in [fig. 3](#). Note that the number of generated policy trajectories is the same as the number of expert demonstrations in order to have a balanced distribution between the two.

B. Policy Excerpts

As training goes on, it often happens that some of the trajectories sampled from the policy have low rewards. Therefore, prior to using policy trajectories, only excerpts of them that have a higher total discounted reward than the averaged total discounted reward of expert demonstrations are selected to be used in the training process. The reason to do so is that these policy trajectories contain more useful information about the reward function and the constraints that exist in the environment. The number shown between parentheses in the title of [fig. 3](#) represents the number of selected policy trajectories at a particular iteration. This mechanism is observed to be more advantageous in the point-obstacle environment where the true constraint is more complex (multi-sided).

C. Memory

The backward step of the framework presented in [fig. 1](#) learns constraints by utilizing the difference between the policy trajectories and expert demonstrations (see [eqns. \(2\) and \(3\)](#)). Therefore, retrieving the constraints becomes more difficult as the policy more closely mimics the behavior of the expert. In such cases, it has been observed that the framework forgets about the constraints that are already learned. To prevent such phenomena, a memory mechanism inspired by the experience replay strategy [\[3\]](#) is proposed in this work.

One approach is to record the whole policy trajectory that has the highest reward among all the trajectories with a higher cumulative reward than the averaged total discounted reward of expert demonstration. Another method is to keep track of the individual points in the trajectories. More concretely, among all the policy trajectories with higher cumulative reward than the averaged cumulative reward of expert demonstrations, one can solely consider a predefined number of points that are more certain (have the highest cost) to be part of the constraint. It is also possible to additionally filter these recorded points such that only the ones that are far enough from the expert demonstrations be part of the constraint.

[Fig. 4](#) exhibits the learned constraints (red area) in the point-obstacle environment using the memory mechanism, which records individual samples with and without extra filtering by the distance to the expert demonstration. As depicted, the additional filtering helps to keep more informative samples in the memory. More specifically, one can observe a few yellow points (recorded in the memory) on the top side of the [fig. 4a](#), which causes the framework to memorize a small constraint above the goal location. Regardless, it is noteworthy that as the training goes on, there are more and more samples recorded

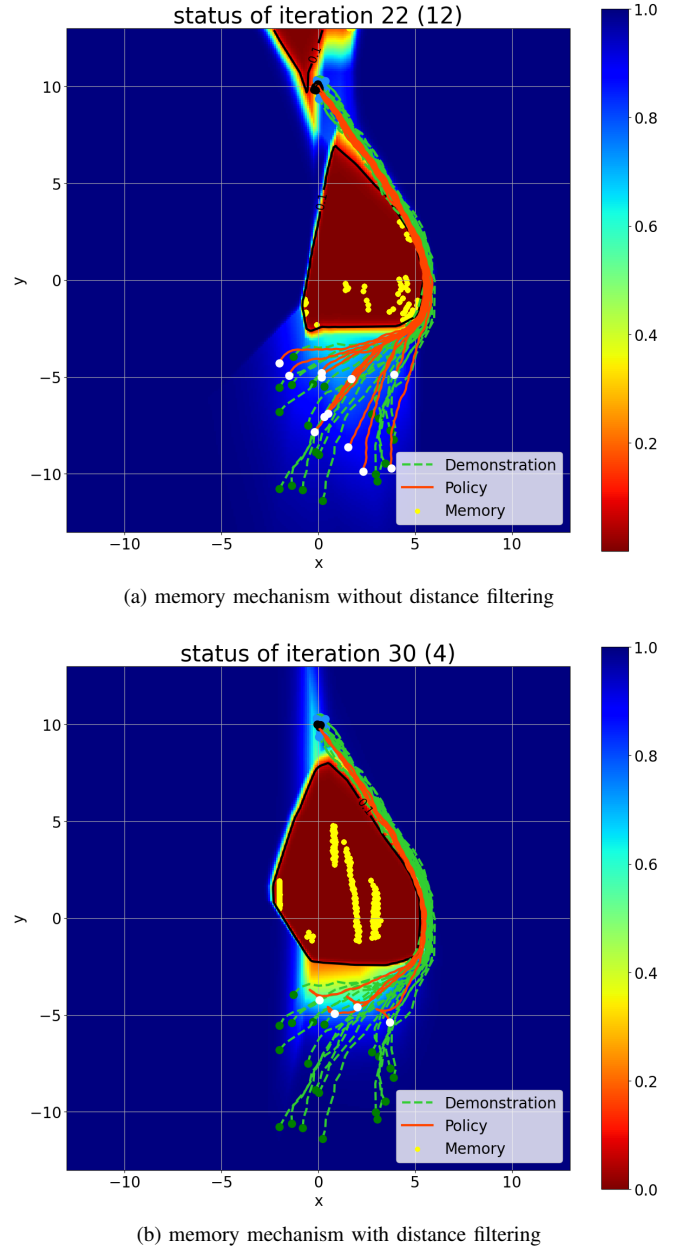


Fig. 4. The final version of the learned constraints in the point-obstacle environment using the memory mechanism with and without filtering based on the distance to the expert demonstrations. Expert demonstrations and policy trajectories are respectively shown in green and orange. The memorized samples are plotted in yellow. The start and end point of each expert demonstration is shown as green and blue dots, respectively. The start and end points of policy trajectories are shown in white and black, respectively. The color bar is associated with the value of ζ_θ . The learned constraint is presented in red alongside the binarization threshold on its contour.

in the memory, which gradually makes the training slower. Moreover, the memory mechanism introduces several new hyperparameters that should be carefully tuned; otherwise, one should expect to have some artifacts similar to the one presented in [fig. 4a](#) in the final outcome.

D. Switching Frequency

As mentioned earlier, recovering constraints in an environment is a dynamic task since the policy and the parameters of

the constraint network get updated at each iteration. This consecutive back-and-forth switching between forward and backward steps makes the training difficult. It has been observed that decreasing the frequency of this switching brings about a more stable and repeatable (in terms of output) learning process. As such, each iteration consists of a predetermined number (more than one) of backward and forward steps. Note that each backward step itself consists of a set of inner iterations to train the neural network as proposed in [1]. Similarly, the policy network is updated for a certain number of epochs at each forward step.

E. Early Stopping

Running the forward step during which the RL agent interacts with the environment is usually time-consuming. Moreover, the whole algorithm is originally [1] set to run for a fixed number of iterations. However, as mentioned earlier, the framework might suffer from the forgetting problem during the training phase. In this work, several early stopping measures have been tested in order to reduce the training time, mitigate the forgetting issue, and stop the constraint learning process in a more rigorous fashion.

One can gradually decrease the number of forward iterations as the training goes on since, after a while, the trajectories sampled from the policy get closer to the expert demonstrations. Another more reasonable approach is to stop the forward-step loop when the policy is already good enough, meaning that the averaged total discounted reward of the policy trajectories is higher than that of the expert demonstrations.

As to early-stopping the whole algorithm, the main method [4] is to stop training when the trajectories of the policy ($\hat{\tau}$) and those of the expert (τ) are close enough, i.e., when $\text{KL}(\tau|\hat{\tau}) \leq k$, where KL is the Kullback–Leibler divergence, and k is a predefined threshold. The probability distribution of τ and $\hat{\tau}$ can be learned with the Gaussian mixture models (GMMs). One can also consider the relative update of KL divergence as a stopping condition. Regardless, tuning the hyperparameter k is difficult and significantly data-dependent. Moreover, using this approach is based on the assumption that when policy trajectories are close to expert demonstrations, one can believe the constraints are correctly learned, which in practice, is not necessarily the case. Nonetheless, since the true constraints are unknown during the training phase, one can not explicitly use them to construct a stopping criterion. As it will be pointed out in section VII, constraint learning using GMMs can provide more flexibility in this regard.

F. Regularization

Because of the limited number of data points, the constraint network ζ_θ is susceptible to overfitting. To relieve this issue while encouraging the predicted output of ζ_θ be closer to one (smaller constraints), [1] proposes to add a regularization term formulated as

$$R(\theta) = -\delta \sum_{\tau \sim \{\mathcal{D}, \hat{\mathcal{D}}\}} |1 - \zeta_\theta(\tau)| \quad (6)$$

to the loss functions expressed in eqns. (2) and (3). Here, δ is the regularization weight, and $\mathcal{D}$ and $\hat{\mathcal{D}}$ are the set of expert and policy trajectories, respectively.

It is noteworthy that since the goal is to come up with a minimal representation of constraints, it makes more sense to regularize the state space containing the constraints instead of just applying a regularization term based on the sampled trajectories. Hence, in this work, the averaged predicted output of a mesh grid covering the state space is considered as the regularization term instead. Such regularization directly encourages ζ_θ to learn a smaller representation of constraints. One can also consider an L_1 regularization term to make the weights of the network more sparse and, thereby, achieve a less complex output. However, since ζ_θ modeled as a neural network has only two hidden layers with a few units, this does not bring a noticeable improvement. Regardless, the weight of the regularization (δ) should be tuned carefully since; otherwise, this can cause the norm of the gradient to be approximately zero, which stops the learning process.

VI. METRICS FOR PERFORMANCE ASSESSMENT

Several metrics have been proposed in the literature to measure the performance of the developed framework, i.e., how good the learned constraints are with respect to true constraints. [1], [5] propose the rate of constraint violation of the learned policy in a testing environment as a performance metric. In the testing environment, unlike the training one, the true constraints are known. This metric is significantly affected by how the policy is learned and does not directly take into account the currently learned constraints.

Another set of metrics can be designed by building a mesh grid over state space and computing the classification accuracy [4], [6]. Despite examining the learned constraints directly, these types of metrics treat all states equally, whereas the states near the boundary of the constraint are of greater importance. Alongside these proposed metrics, a few other measures have also been considered in this work, including the averaged reward of the feasible policy trajectories, false-positive rate (where infeasible points are considered positive), recall, precision, and intersection over union (also known as the Jaccard index).

Notably, the accuracy metric counts the one-to-one correspondence between the current label of data points in the mesh grid with their true labels. As such, the accuracy starts at the beginning of the training with a considerably high value (above 90% for the point-obstacle environment). The reason behind this can be understood by examining fig. 2b carefully. It is apparent that a large portion of state space is feasible; therefore, there are a lot of correct matches between the learned feasible labels and the true ones. On the other hand, the Jaccard index, proposed in this work, calculates the intersection between the areas with a cost of one (infeasible) over their union; thereby, it encapsulates both the shape of the learned constraint and its location.

VII. CONSTRAINT LEARNING WITH GMMs

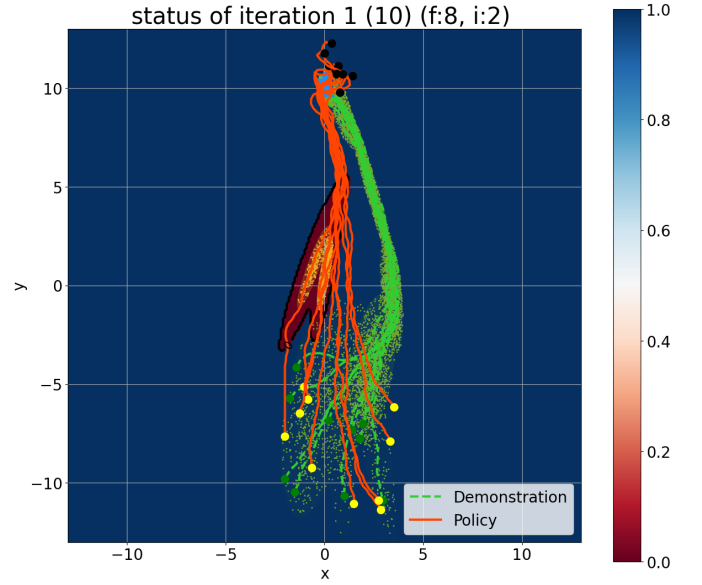
As discussed earlier, the problem of learning constraints in an environment can be interpreted as a binary classification task where the space should be classified into feasible and infeasible areas. As such, instead of modeling ζ_θ as a neural network, this work has investigated the application of the Gaussian mixture models (GMMs). The motivation behind this choice is the fact that GMM is a probabilistic approach that learns an estimation of the density of the data points and has the notion of likelihood based on which one can classify the space into feasible and constrained regions.

An extension to the nominal iterative expectation-maximization (EM) algorithm, i.e., variational inference, is employed in this project to fit a mixture of Gaussian models to the data. Variational (also called Bayesian) GMM has the advantage of automatically inferring the number of Gaussian components at the cost of having additional hyperparameters. Regardless, to avoid having Gaussian models with a relatively small weight, a threshold (1% or 5% in different experiments) is utilized to restrict the choice of the Gaussian models. In order to have the most generic models, the Gaussian covariance matrices are assumed to be full. Moreover, the weights are initialized using the k-means algorithm. Furthermore, the initialization has been performed five times, among which the best one is kept. Besides, the Dirichlet process is selected as the weight concentration prior.

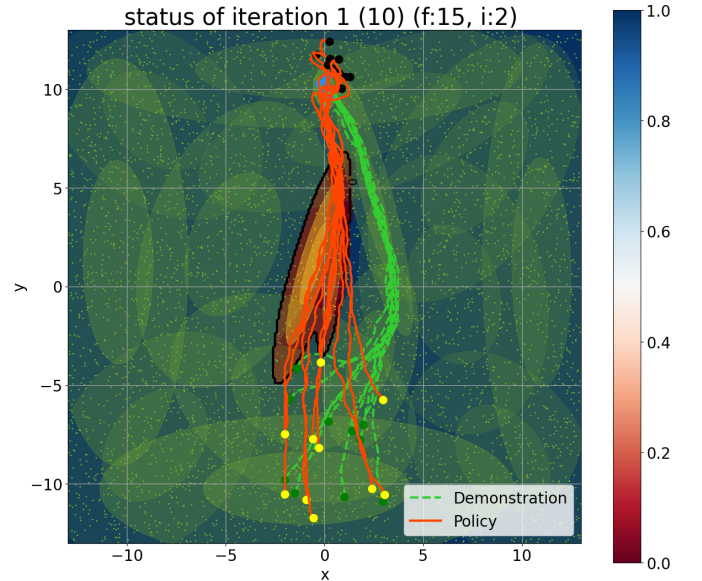
Each set of feasible and infeasible data is modeled separately by a set of Gaussian models. It is noteworthy that the initial GMMs are built upon two initial sampling strategies. In the first approach, all the points in the expert demonstrations are kept as the initial feasible data. The initial infeasible dataset consists of samples from the warmed-up policy that are far enough (based on a distance threshold) from the expert demonstrations and are on a trajectory with higher cumulative reward than the averaged total discounted reward of expert demonstrations. As such, the initial feasible and infeasible data points are pushed apart from one another. It is noteworthy two datasets built this way have a different number of data points. Therefore, two GMMs are trained separately at each initial feasible and infeasible data; afterward, a fixed number of samples is generated from each GMM in order to have balanced classes as the training continues. It is noteworthy that the samples are generated within two (or three) standard deviations of the Gaussian models since samples that are farther away can act as outliers with a low likelihood.

In the second approach of generating initial samples (which needs further investigation), instead of producing a fixed number of samples from each GMM, a grid of points is sampled from a uniform distribution over the state space to fill the holes that lack data. Then, the initially trained feasible and infeasible GMMs are used to label the points in this grid. More concretely, the points whose log-likelihood is higher under the feasible GMM compared to the infeasible one are considered to be feasible. Subsequently, the rest of the samples are labeled as infeasible. Note that to check feasibility, the log-likelihood

function is preferred over the likelihood function since taking the exponent of log-likelihood is observed to produce saturated values. Regardless, this new set of labeled data is afterward concatenated with the initial feasible and infeasible data points, and both GMMs are retrained accordingly.



(a) the first initial sampling method



(b) the second initial sampling method

Fig. 5. The initial iteration of the framework equipped with GMM to learn constraints in the point-obstacle environment with two different initial sampling methods. Feasible and infeasible GMMs, alongside their produced samples, are illustrated in green and yellow, respectively. The numbers inside the second parentheses in the titles are representative of the number of Gaussian components. Expert demonstrations and policy trajectories are respectively shown in green and orange. The start and end point of each expert demonstration is shown as green and blue dots, respectively. The start and end points of policy trajectories are shown in yellow and black, respectively. The learned constraint is presented in red, and the blue-colored area is representative of the feasible region.

Fig. 5 presents the difference between the two initial

sampling methods. More importantly, the initially learned constraint could arguably be accepted as a valid constraint if it were slightly closer to the expert demonstrations despite the fact the policy is still at the warmed-up stage. Whereas, in the case of using neural networks, the learned constraint during the initial iterations is mostly wrong. It is noteworthy that because of the randomness in initialization, the results of the initial classification are not necessarily repeatable.

At each iteration of the training phase, samples constituting the policy trajectories are classified with the GMMs updated at the previous iterations. The new samples are then accumulated in the corresponding feasible and infeasible dataset, and the GMMs are retrained accordingly. Keeping track of the policy trajectories in such a manner is similar to the memory mechanism introduced earlier and avoids the forgetting problem. However, it is noteworthy that the computational complexity of training GMMs with full covariance matrices is $\mathcal{O}(KMN^2)$, where K , M , and N are the number of Gaussian components, the number of data points, and the dimension of the data, respectively. Moreover, the Bayesian GMM consumes more time to infer K automatically. As such, retraining GMMs takes more time as the training goes on.

One solution to mitigate this is to solely retrain the GMM modeling the infeasible data. The reason is, in the considered environments, the feasible region is larger than the constrained area. Therefore, typically, at each iteration, a large portion of data will be labeled as feasible, causing the training of the corresponding GMM to last longer. In case one ceases the retraining of the feasible GMM in order to reduce the computational time, it is possible to produce a large number of initial feasible samples such that, as the training continues, both classes become more balanced.

Nonetheless, appending only infeasible points might cause the infeasible area to intersect with the expert demonstrations as presented in [fig. 6a](#). The reason is a large number of infeasible points make the infeasible GMM dominates (in terms of log-likelihood) over the feasible one such that the region close to the boundary would be labeled as infeasible. This phenomenon can be prevented if the region close (based on a threshold) to the expert demonstrations is always labeled as feasible irrespective of the log-likelihood computed under the GMMs. It is noteworthy that, at the moment, nothing directly prevents the GMMs from learning a constraint that coincides with expert demonstrations. Particularly, this is susceptible to happening near the starting point of the expert demonstrations (see [fig. 6](#)) since the density of the data points there is significantly lower than the rest of the trajectory and also lower than the accumulated number of infeasible points. Therefore, it is worth to further investigate some post-processing procedures on the expert demonstrations to make them have a more uniform representation.

Instead of retraining the GMMs with all the accumulated feasible and infeasible data points from the beginning, one can use the previously learned GMMs at each iteration to generate a fixed number of samples. With this method, each iteration takes nearly the same amount of time, and two classes will

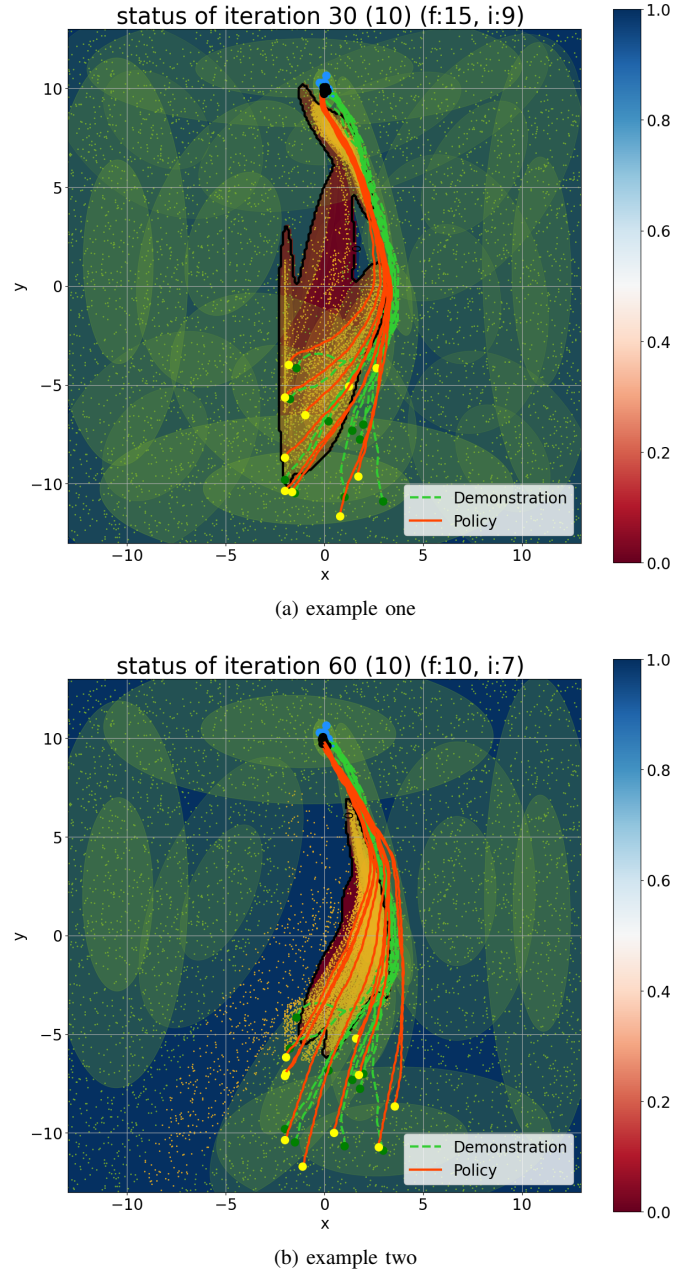


Fig. 6. The final version of the learned constraints in the point-obstacle environment using GMMs. Feasible and infeasible GMMs, alongside their produced samples, are illustrated in green and yellow, respectively. The numbers inside the second parentheses in the titles are representative of the number of Gaussian components, whereas the numbers inside the first parentheses correspond to the number of policy and expert trajectories. Expert demonstrations and policy trajectories are respectively shown in green and orange. The start and end point of each expert demonstration is shown as green and blue dots, respectively. The start and end points of policy trajectories are shown in yellow and black, respectively. The learned constraint is presented in red, and the blue-colored area is representative of the feasible region.

be balanced. However, since infeasible points are not being recorded in memory, the forgetting issue might arise again. Moreover, in the case of retraining the feasible GMM at each iteration, this approach can cause a shift (with respect to the expert demonstration) in the feasible samples generated by the feasible GMM at each iteration (see [fig. 7](#)). Therefore,

the policy diverges away from expert trajectories. Note that this figure corresponds to a run with the first initial sampling method, which does not work well for the point-obstacle environment since the constraint is particularly small and there are a lot of holes (lack of data points) in the state space.

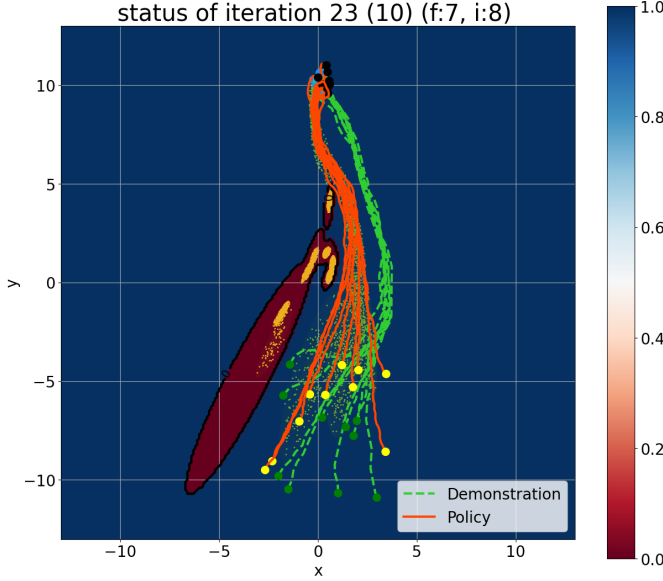


Fig. 7. Learned constraints (red area) in the point-obstacle environment using GMMs with the first initial sampling method alongside the retraining GMM method in which a fixed number of samples is generated at each iteration. Feasible and infeasible GMMs, alongside their produced samples, are illustrated in green and yellow, respectively. Expert demonstrations and policy trajectories are respectively shown in green and orange. The start and end point of each expert demonstration is shown as green and blue dots, respectively. The start and end points of policy trajectories are shown in yellow and black, respectively.

Another idea is to choose a fixed number of most likely feasible and infeasible data from the accumulated dataset instead of generating a fixed number of samples at each iteration. This approach, however, suffers from the non-uniform density of the expert demonstrations. More concretely, as mentioned earlier, the points in the first half (near the starting point) of the expert demonstration are not dense, and their log-likelihood under the feasible GMM is thus very low. Furthermore, if the number of selected highly likely samples is not large enough, the framework might learn a very small constrained region far from the expert demonstrations, which may also cause the policy not to be updated in a meaningful manner.

Furthermore, to save the computational time of the training phase, one can consider only retraining the infeasible and/or feasible GMM when a certain minimum number of points are added to the dataset at each iteration. More specifically, when the policy converges to the expert demonstrations, the number of newly labeled infeasible points will be quite small in case the learned constraint does not cover the expert trajectories, as pointed out earlier. Hence, it does not worth retraining the infeasible GMM since its representation will barely change with the addition of very few new points. More importantly, the number of new infeasible points can be tracked and em-

ployed as an early stopping condition of the whole framework alongside the methods discussed in section V-E.

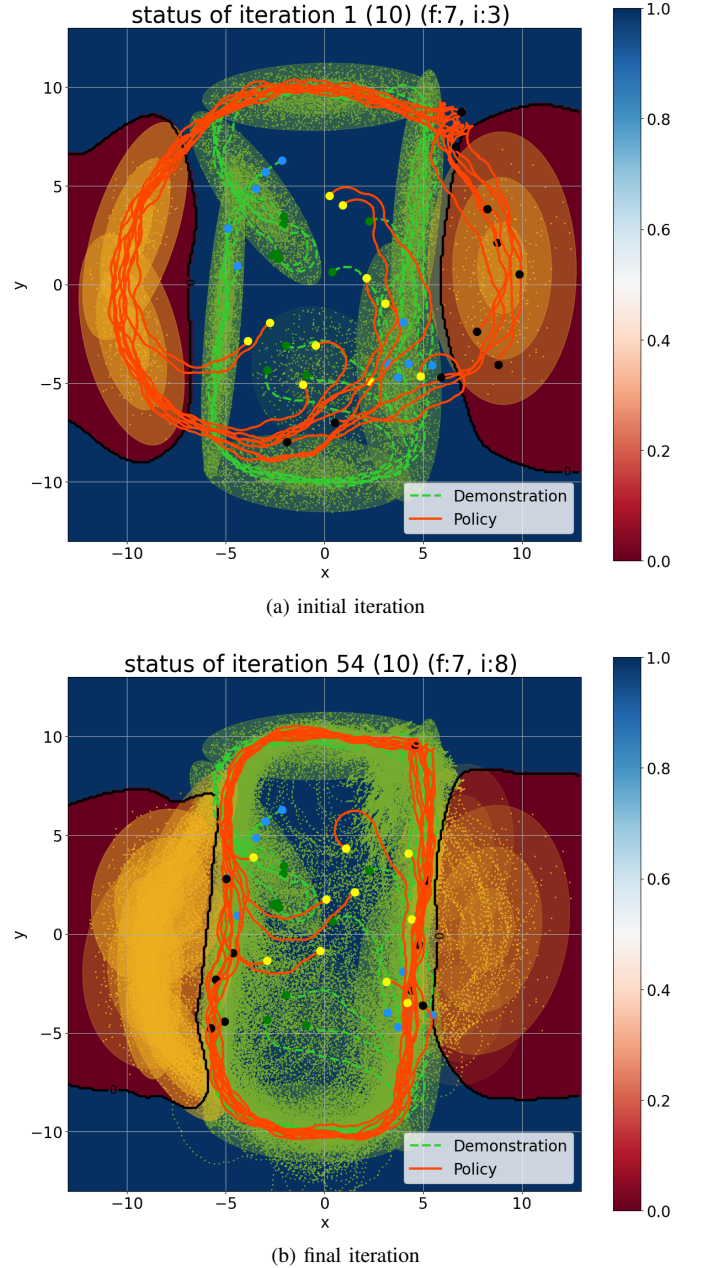


Fig. 8. The initial and final versions of the learned constraints (red region) in the point-circle environment using GMMs with three standard deviation sampling. Feasible and infeasible GMMs, alongside their produced samples, are illustrated in green and yellow, respectively. The numbers inside the second parentheses in the titles are representative of the number of Gaussian components, whereas the numbers inside the first parentheses correspond to the number of policy and expert trajectories. Expert demonstrations and policy trajectories are respectively shown in green and orange. The start and end point of each expert demonstration is shown as green and blue dots, respectively. The start and end points of policy trajectories are shown in yellow and black, respectively. The blue-colored area is representative of the feasible region.

One can also consider a threshold d such that if the likelihood of samples under infeasible GMM is below this value, those samples will be labeled as feasible irrespective

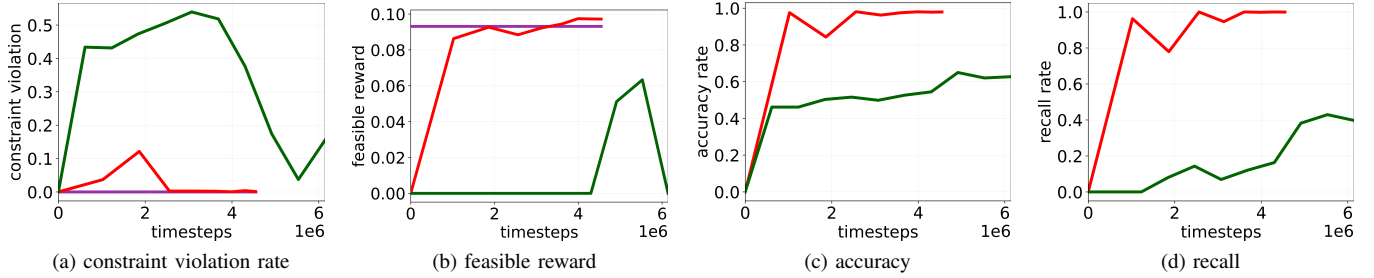


Fig. 9. Metrics used to compare the performance of the constraint recovery in the point-circle environment between this work (with the constraint neural network with BCE loss) and the framework proposed by [1]. The red, green, and purple curves correspond to this work, the original framework, and the expert demonstrations, respectively.

of the relative difference of likelihood under feasible and infeasible GMMs. This procedure, similar to the effect of having a threshold for the distance with respect to expert demonstration, can slightly prevent the occurrence of tight constraints, i.e., when the learned constraints are very close to the expert demonstrations as in fig. 6. Moreover, this strategy will automatically assume the areas far from the learned GMMs are feasible. This is a fine assumption because only the constraints near the expert demonstrations are important since those are the ones that prevent the expert from following a trajectory with maximum cumulative reward. Nevertheless, this method introduces the new hyperparameter d , which is very difficult to tune.

The learned constraints using GMMs in the point-circle environment are shown in fig. 8. Despite being different from the true constraints (see fig. 2a), the constraints learned initially are already good enough, meaning that these constraints are in agreement with the trajectory taken by a constraint-abiding expert. All in all, it has been observed that GMMs, when integrated into the backward step, have the capability of learning meaningful constraints in both point-circle and point-obstacle environments. Moreover, the results obtained with GMMs are competitive with those achieved when using the constraint neural network, given the fact that they do not require as many additional backward and forward iterations (reducing the switching frequency) to stabilize the training phase.

Furthermore, the GMMs output is usually more repeatable than when the constraint network is employed in conjunction with different mechanisms introduced earlier. Additionally, the GMM training phase has been performed on top of ten expert and policy trajectories, whereas the constraint network usually requires more trajectories (including the memory mechanism) alongside a regularization strategy to avoid overfitting and bring about reasonably learned constraints. Hence, one can argue that the GMM approach is more data-efficient. It is noteworthy that both the quantity and quality of the data become of greater importance in higher dimensions.

VIII. COMPARISON WITH THE ORIGINAL FRAMEWORK

Using some of the metrics introduced in section VI, the performance of the framework developed in this project is compared with that of the work done by [1], and the results

are illustrated in fig. 9. Here, only the constraint network with the BCE loss equipped with proposed mechanisms, such as warmed-up policy and reduced switching frequency, is considered. Evidently, all the metrics suggest a significant improvement over the work of [1]. More specifically, the trained policy violates the true constraint of the point-circle environment less than the nominal agent of [1] and brings about a greater average reward among feasible trajectories, which is also competitive with expert demonstrations. The accuracy and recall are also significantly higher, indicating the learned constraint is a better representative of the true one compared to [1].

Furthermore, the framework proposed by [1] needs to be run for about a hundred times larger number of forward time steps (total number of steps during which the RL agent interacts with the environment) compared to this work such that it would be able to learn a meaningful constraint. Moreover, to have a more fair comparison, the warmed-up policy is also employed in both cases, despite the fact that it is not considered in [1]. Yet, this work accomplishes significantly better results in a much shorter amount of computational time. It is also noteworthy that the reported experiments in [1] only illustrated learning a one-dimensional constraint on one side of the environment. This work, on the other hand, has been able to recover the constraints of a wider diversity, as detailed before.

IX. DISCUSSION AND FUTURE WORK

A. Support Vector Machine (SVM)

One alternative to GMM is the support vector machine (SVM) algorithm which is well-suited for binary classification. Since SVM problems can be formulated as convex optimization programs, they are significantly faster in the training phase compared to GMMs learned by the iterative EM algorithm.

Fig. 10 illustrated the result of a simple binary classification trial with SVM alongside the radial basis function (RBF) kernel in the point obstacle environment. Here, expert demonstrations (green points) are labeled as feasible data points, while the infeasible points (in yellow) are extracted from the warmed-up policy in such a way that their total discounted reward is higher than the maximum total discounted reward of expert demonstrations, and their distance to expert demonstration is higher than a threshold. Evidently, the recovered constraints (in red) are close to the true constraints (see fig. 2a).

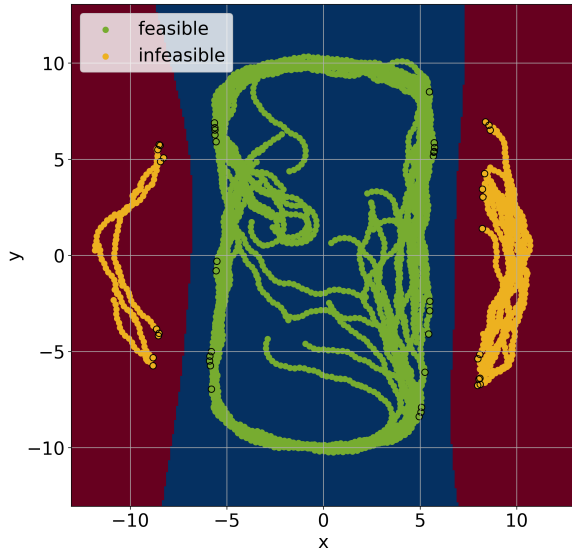


Fig. 10. The result of binary classification using SVM with RBF kernel. The green and yellow points are expert demonstrations and parts of the warmed-up policy, respectively. The blue and red regions are respectively representative of feasible and infeasible regions. The black hollow circles are the support vectors.

Note that, in addition to the kernel hyperparameters, the penalty factor C is another SVM hyperparameter that determines the cost of misclassifying the data points. Tuning different C values for the feasible and infeasible points alongside the threshold chosen to initially label data points as infeasible based on their distance to expert demonstration will affect the tightness of the learned constraints. For instance, associating a higher C for expert demonstrations as the feasible dataset forces SVM not to afford to misclassify those points. Moreover, since the decision function of SVM is continuous (before taking the sign), it might be possible to use its gradient to guide the RL agent by modifying its actions.

Moreover, since tuning C might be difficult in practice, particularly as the training goes on, the ν -SVM method can also be of interest. ν here is an upper bound on how much the data points are allowed to live within the SVM margin. Moreover, ν sets a lower bound on the ratio of the number of support vectors over the total number of data. Another variant of SVM is transductive SVM (TSVM), which is a semi-supervised learning technique designed for learning from partially labeled datasets. In this context, the expert demonstrations are labeled as feasible data points, whereas segments of policy trajectories with higher rewards than expert trajectories and far from them can be labeled as infeasible points. The rest of the dataset is thus left unlabeled. TSVM is suitable for such a scenario as it tends to come with a boundary that brings about perfect classification of the labeled data points while separating the unlabeled ones into either side of the margin.

Note that, unlike GMM, SVM and its variants proposed here do not entail a notion of likelihood. Hence, one cannot particularly reason about the validity of the predicted constraints, especially in areas where there is no data point. Moreover,

recall that in section VII, the feasible and infeasible GMM models are utilized respectively to label the policy trajectories into feasible and infeasible regions based on the log-likelihood of the newly observed data under previously learned GMMs. Hence, since SVM lacks this notion of confidence, it needs further investigation of how to label the policy trajectories at each iteration.

B. Main Concerns

Apart from the choice of the binary classifier, one of the main concerns is how to encourage or constrain the algorithms to learn a minimal yet accurate (with the correct shape and at the correct location) representation of the constraints. As pointed out earlier, the problem of constraint recovery does not necessarily have a unique solution, meaning that various realizations of constraints can correspond to the same set of expert demonstrations and an explicit reward function. What matters most is the constraint in the immediate vicinity of the expert trajectories. As such, retrieving the true constraint is quite challenging.

As to the proposed mechanisms, more investigation needs to be done into the regularization and memory strategies. Although it does not reveal a solid advantage over the usage of the neural network in the experiments that have been done so far, the GMM approach, as shown earlier, is capable of achieving promising results. There still exists a lot of room to improve this method if, for instance, one can learn the GMMs more intelligently. In addition, the quality of expert demonstrations (their density, for example) and the warmed-up policy have a huge effect on both of the constraint learning strategies, and the influence of each needs to be further explored.

Furthermore, designing a rigorous early stopping criterion instead of stopping after a fixed number of iterations is of interest. Additionally, more experiments need to be done to have a comprehensive and quantified comparison between different algorithms and also to better study the effect of each mechanism and the underlying hyperparameters. It is noteworthy that the current framework already has a lot of meta-parameters with coupled effects that need to be tuned precisely.

REFERENCES

- [1] U. Anwar, S. Malik, A. Aghasi, and A. Ahmed, "Inverse constrained reinforcement learning," *arXiv preprint arXiv:2011.09999*, 2020.
- [2] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.
- [3] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine learning*, vol. 8, pp. 293–321, 1992.
- [4] D. R. Scobee and S. S. Sastry, "Maximum likelihood constraint inference for inverse reinforcement learning," *arXiv preprint arXiv:1909.05477*, 2019.
- [5] C. Pérez-D'Arpino and J. A. Shah, "C-learn: Learning geometric constraints from demonstrations for multi-step manipulation in shared autonomy," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 4058–4065.
- [6] G. Chou, D. Berenson, and N. Ozay, "Learning constraints from demonstrations," in *Algorithmic Foundations of Robotics XIII: Proceedings of the 13th Workshop on the Algorithmic Foundations of Robotics 13*. Springer, 2020, pp. 228–245.