Energy-Based Continuous Inverse Optimal Control

Yifei Xu
UCLA
fei960922@ucla.edu

Jianwen Xie
Baidu Research
jianwen@ucla.edu

Tianyang Zhao
UCLA
tyzhao@ucla.edu

Chris Baker
iSee Inc.
chrisbaker@isee.ai

Yibiao Zhao
iSee Inc.
yz@isee.ai

Ying Nian Wu
UCLA
ywu@stat.ucla.edu

Abstract

The problem of continuous optimal control (over finite time horizon) is to minimize a given cost function over the sequence of continuous control variables. The problem of continuous inverse optimal control is to learn the unknown cost function from expert demonstrations. In this article, we study this fundamental problem in the framework of energy-based model, where the observed expert trajectories are assumed to be random samples from a probability density function defined as the exponential of the negative cost function up to a normalizing constant. The parameters of the cost function are learned by maximum likelihood via an "analysis by synthesis" scheme, which iterates the following two steps: (1) Synthesis step: sample the synthesized trajectories from the current probability density using the Langevin dynamics via back-propagation through time. (2) Analysis step: update the model parameters based on the statistical difference between the synthesized trajectories and the observed trajectories. Given the fact that an efficient optimization algorithm is usually available for an optimal control problem, we also consider a convenient approximation of the above learning method, where we replace the sampling in the synthesis step by optimization. We demonstrate the proposed method on autonomous driving tasks, and show that it can learn suitable cost functions for optimal control.

1 Introduction

The problem of continuous optimal control has been extensively studied. In this paper, we study the control problem of finite time horizon, where the trajectory is over a finite period of time. In particular, we focus on the problem of autonomous driving as a concrete example. In continuous optimal control, the control variables or actions are continuous. The dynamics is known. The cost function is defined on the trajectory and is usually in the form of the sum of stepwise costs and the cost of the final state. We call such a cost function Markovian. The continuous optimal control seeks to minimize the cost function over the sequence of continuous control variables or actions, and many efficient algorithms have been developed for various optimal control problems [29]. For instance, in autonomous driving, the iLQR (iterative linear quadratic regulator) algorithm is a commonly used optimization algorithm [20] [2]. We call such an algorithm the built-in optimization algorithm for the corresponding control problem.

In applications such as autonomous driving, the dynamics is well defined by the underlying physics and mechanics. However, it is a much harder problem to design or specify the cost function. One solution to this problem is to learn the cost function from expert demonstrations by observing their

Machine Learning for Autonomous Driving Workshop at the 34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

sequences of actions. Learning the cost function in this way is called continuous inverse optimal control problem.

In this article, we study the fundamental problem of continuous inverse optimal control in the framework of energy-based model. Originated from statistical physics, an energy-based model is a probability distribution where the probability density function is in the form of exponential of the negative energy function up to a normalizing constant. Instances with low energies are assumed to be more likely according to the model. For continuous inverse optimal control, the cost function plays the role of energy function, and the observed expert sequences are assumed to be random samples from the energy-based model so that sequences with low costs are more likely to be observed. We can choose the cost function either as a linear combination of a set of hand-designed features, or a non-linear and non-markovian neural network. The goal is to learn the parameters of the cost function from the expert sequences.

The parameters can be learned by the maximum likelihood method in the context of the energy-based model. The maximum likelihood learning algorithm follows an "analysis by synthesis" scheme, which iterates the following two steps: (1) Synthesis step: sample the synthesized trajectories from the current probability distribution using the Langevin dynamics [23]. The gradient computation in the Langevin dynamics can be conveniently and efficiently carried out by back-propagation through time. (2) Analysis step: update the model parameters based on the statistical difference between the synthesized trajectories and the observed trajectories. Such a learning algorithm is very general, and it can learn complex cost functions such as those defined by the neural networks.

For an optimal control problem where the cost function is of the Markovian form, a built-in optimization algorithm is usually already available, such as the iLQR algorithm for autonomous driving. In this case, we also consider a convenient modification of the above learning method, where we change the synthesis step (1) into an optimization step while keeping the analysis step (2) unchanged. We give justifications for this optimization-based method, although we want to emphasize that the sampling-based method is still more fundamental and principled, and we treat optimization-based method as a convenient modification.

We demonstrate the proposed energy-based continuous optimal control methods on autonomous driving and show that the proposed methods can learn suitable cost functions for optimal control.

2 Contributions and related work

The contributions of our work are as follows. (1) We propose an energy-based method for continuous inverse optimal control based on Langevin sampling via back-propagation through time. (2) We also propose an optimization-based method as a convenient approximation. (3) We evaluate the proposed methods on autonomous driving tasks for both single-agent system and multi-agent system, with both linear cost function and neural network non-linear cost function.

The following are research themes related to our work.

(1) Maximum entropy framework. Our work follows the maximum entropy framework of [39] for learning the cost function. Such a framework has also been used previously for generative modeling of images [38] and Markov logic network [25]. In this framework, the energy function is a linear combination of hand-designed features. Recently, [31] generalized this framework to deep version. In these methods, the state spaces are discrete, where dynamic programming schemes can be employed to calculate the normalizing constant of the energy-based model. In our work, the state space is continuous, where we use Langevin dynamics via back-propagation through time to sample trajectories from the learned model. We also propose an optimization-based method where we use gradient descent or a built-in optimal control algorithm as the inner loop for learning.

(2) ConvNet energy-based models(EBM). Recently, [32], [35], [36], [33] [34], applied energy-based model to various generative modeling tasks, where the energy functions are parameterized by ConvNets [17] [15]. Our method is different from ConvNet EBM. The control variables in our method form a time sequence. In gradient computation for Langevin sampling, back-propagation through time is used. Also, we propose an optimization-based modification and give justifications.

(3) Inverse reinforcement learning. Most of the inverse reinforcement learning methods [8, 7], including adversarial learning methods [9], [13], [21], [7], involve learning a policy in addition to

the cost function. In our work, we do not learn any policy. We only learn a cost function, where the trajectories are sampled by the Langevin dynamics or obtained by gradient descent or a built-in optimal control algorithm.

(4) Continuous inverse optimal control (CIOC). The CIOC problem has been studied by [22] and [19]. In [22], the dynamics is linear and the cost function is quadratic, so that the normalizing constant can be computed by a dynamic programming scheme. In [19], the Laplace approximation is used for approximation. However, the accuracy of the Laplace approximation is questionable for complex cost function. In our work, we assume general dynamics and cost function, and we use Langevin sampling for maximum likelihood learning without resorting to Laplace approximation.

(5) Trajectory prediction. A recent body of research has been devoted to supervised learning for trajectory prediction [1], [10], [30], [18], [6], [37]. These methods directly predict the coordinates and they do not consider control and dynamic models. As a result, they cannot be used for inverse optimal control.

3 Energy-based inverse control

3.1 Optimal control

We study the finite horizon control problem for discrete time $t \in \{1, ..., T\}$. Let x_t be the state at time t. Let $\mathbf{x} = (x_t, t = 1, ..., T)$. Let u_t be the continuous control variable or action at time t. Let $\mathbf{u} = (u_t, t = 1, ..., T)$. The dynamics is assumed to be deterministic, $x_t = f(x_{t-1}, u_t)$, where f is given, so that $\mathbf{u}$ determines $\mathbf{x}$. The trajectory is $(\mathbf{x}, \mathbf{u}) = (x_t, u_t, t = 1, ..., T)$. Let e be the environment condition. We assume that the recent history $h = (x_t, u_t, t = -k, ..., 0)$ is known.

The cost function is $C_\theta(\mathbf{x}, \mathbf{u}, e, h)$ where θ consists of the parameters that define the cost function. Its special case is of the linear form $C_\theta(\mathbf{x}, \mathbf{u}, e, h) = \langle \theta, \phi(\mathbf{x}, \mathbf{u}, e, h) \rangle$, where ϕ is a vector of hand-designed features, and θ is a vector of weights for these features. We can also parametrize C_θ by a neural network. The problem of optimal control is to find $\mathbf{u}$ to minimize $C_\theta(\mathbf{x}, \mathbf{u}, e, h)$ with given e and h under the known dynamics f. The problem of inverse optimal control is to learn θ from expert demonstrations $D = \{(\mathbf{x}_i, \mathbf{u}_i, e_i, h_i), i = 1, ..., n\}$.

3.2 Energy-based probabilistic model

The energy-based model assumes the following conditional probability density function

$$p_\theta(\mathbf{u}|e, h) = \frac{1}{Z_\theta(e, h)} \exp[-C_\theta(\mathbf{x}, \mathbf{u}, e, h)], \tag{1}$$

where $Z_\theta(e, h) = \int \exp[-C_\theta(\mathbf{x}, \mathbf{u}, e, h)] d\mathbf{u}$ is the normalizing constant. Recall that $\mathbf{x}$ is determined by $\mathbf{u}$ according to the deterministic dynamics, so that we only need to define probability density on $\mathbf{u}$. The cost function C_θ serves as the energy function. For expert demonstrations D, $\mathbf{u}_i$ are assumed to be random samples from $p_\theta(\mathbf{u}|e_i, h_i)$, so that $\mathbf{u}_i$ tends to have low cost $C_\theta(\mathbf{x}, \mathbf{u}, e_i, h_i)$.

3.3 Sampling-based inverse optimal control

We can learn the parameters θ by maximum likelihood. The log-likelihood is

$$L(\theta) = \frac{1}{n} \sum_{i=1}^{n} \log p_\theta(\mathbf{u}_i|e_i, h_i). \tag{2}$$

We can maximize $L(\theta)$ by gradient ascent, and the learning gradient is

$$L'(\theta) = \frac{1}{n} \sum_{i=1}^{n} \left[\mathrm{E}_{p_\theta(\mathbf{u}|e_i, h_i)} \left(\frac{\partial}{\partial \theta} C_\theta(\mathbf{x}, \mathbf{u}, e_i, h_i) \right) - \frac{\partial}{\partial \theta} C_\theta(\mathbf{x}_i, \mathbf{u}_i, e_i, h_i) \right], \tag{3}$$

which follows from the fact that $\frac{\partial}{\partial \theta} \log Z_\theta(e, h) = -\mathrm{E}_{p_\theta(\mathbf{u}|e,h)} \left(\frac{\partial}{\partial \theta} C_\theta(\mathbf{x}, \mathbf{u}, e_i, h_i) \right)$.

In order to approximate the above expectation, we can generate multiple random sample $\tilde{\mathbf{u}}_i \sim p_\theta(\mathbf{u}|e, h)$, which generates the sampled trajectory $(\tilde{\mathbf{x}}_i, \tilde{\mathbf{u}}_i)$ by unfolding the dynamics. We estimate

$L'(\theta)$ by

$$\hat{L}'(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left[\frac{\partial}{\partial\theta}C_\theta(\tilde{\mathbf{x}}_i, \tilde{\mathbf{u}}_i, e_i, h_i) - \frac{\partial}{\partial\theta}C_\theta(\mathbf{x}_i, \mathbf{u}_i, e_i, h_i)\right], \tag{4}$$

which is the stochastic unbiased estimator of $L'(\theta)$. Then we can run the gradient ascent algorithm $\theta_{\tau+1} = \theta_\tau + \gamma_\tau \hat{L}'(\theta_\tau)$ to obtain the maximum likelihood estimate of θ, where τ indexes the time step, γ_τ is the step size. According to the Robbins-Monroe theory of stochastic approximation [26], if $\sum_\tau \gamma_\tau = \infty$ and $\sum_\tau \gamma_\tau^2 < \infty$, the algorithm will converge to a solution of $L'(\theta) = 0$. For each i, we can also generate multiple copies of $(\tilde{\mathbf{x}}_i, \tilde{\mathbf{u}}_i)$ from $p_\theta(\mathbf{u}|e_i, h_i)$ and average them to approximate the expectation in (3). A small number is sufficient because the averaging effect takes place over time.

In linear case, where $C_\theta(\mathbf{x}, \mathbf{u}, e, h) = \langle\theta, \phi(\mathbf{x}, \mathbf{u}, e, h)\rangle$, we have $\frac{\partial}{\partial\theta}C_\theta(\mathbf{x}, \mathbf{u}, e_i, h_i) = \phi(\mathbf{x}, \mathbf{u}, e, h)$, making $\hat{L}'(\theta) = \frac{1}{n}\sum_{i=1}^{n}[\phi(\tilde{\mathbf{x}}_i, \tilde{\mathbf{u}}_i, e_i, h_i) - \phi(\mathbf{x}_i, \mathbf{u}_i, e_i, h_i)]$. It is the statistical difference between the observed trajectories and synthesized trajectories. At maximum likelihood estimate, the two match each other.

The synthesis step that samples from $p_\theta(\mathbf{u}|e, h)$ can be accomplished by Langevin dynamics, which iterates the following steps:

$$\mathbf{u}_{s+1} = \mathbf{u}_s - \frac{\delta^2}{2}C'_\theta(\mathbf{x}_s, \mathbf{u}_s, e, h) + \delta\mathbf{z}_s, \tag{5}$$

where s indexes the time step, $C'_\theta(\mathbf{x}, \mathbf{u}, e, h)$ is the derivative with respect to $\mathbf{u}$. δ is the step size, and $\mathbf{z}_s \sim \mathrm{N}(0, I)$ independently over s, where I is the identity matrix of the same dimension as $\mathbf{u}$. The Langevin dynamics is an inner loop of the learning algorithm.

The gradient term $C'_\theta(\mathbf{x}, \mathbf{u}, e, h) = \partial C_\theta(\mathbf{x}, \mathbf{u}, e, h)/\partial\mathbf{u}$ is computed via back-propagation through time, where $\mathbf{x}$ can be obtained from $\mathbf{u}$ by unrolling the deterministic dynamics. The computation can be efficiently and conveniently carried out by auto-differentiation on the current deep learning platforms.

3.4 Optimization-based inverse optimal control

We can remove the noise term in Langevin dynamics (5), to make it a gradient descent process, $\mathbf{u}_{s+1} = \mathbf{u}_s - \eta C'_\theta(\mathbf{x}_s, \mathbf{u}_s, e, h)$, and we can still learn the cost function that enables optimal control. This amounts to modifying the synthesis step into an optimization step. Moreover, a built-in optimization algorithm is usually already available for minimizing the cost function $C_\theta(\mathbf{x}, \mathbf{u}, e, h)$ over $\mathbf{u}$. For instance, in autonomous driving, a commonly used algorithm is iLQR. In this case, we can replace the synthesis step by an optimization step, where, instead of sampling $\tilde{\mathbf{u}}_i \sim p_{\theta_t}(\mathbf{u}|e_i, h_i)$, we optimize

$$\tilde{\mathbf{u}}_i = \arg\min_{\mathbf{u}} C_\theta(\mathbf{x}, \mathbf{u}, e_i, h_i). \tag{6}$$

The analysis step remains unchanged. In this paper, we emphasize the sampling-based method, which is more principled maximum likelihood learning, and we treat the optimization-based method as a convenient modification. We will evaluate both learning methods in our experiments.

A justification for the optimization-based algorithms in the context of the energy-based model (1) is to consider its tempered version: $p_\theta(\mathbf{u}|e, h) \propto \exp[-C_\theta(\mathbf{x}, \mathbf{u}, e, h)/T]$, where T is the temperature. Then the optimized $\tilde{\mathbf{u}}$ that minimizes $C_\theta(\mathbf{x}, \mathbf{u}, e, h)$ can be considered the zero-temperature sample, which is used to approximate the expectation in (3).

3.5 Energy-based inverse optimal control algorithm

Algorithm 1 presents the learning algorithm.

We treat the sampling-based method as a more fundamental and principled method, and the optimization-based method as a convenient modification. In our experiments, we shall evaluate both sampling-based method using Langevin dynamics and optimization-based method with gradient descent (GD) or iLQR as optimizer.

Algorithm 1 Energy-based inverse optimal control

input expert demonstrations $D = \{(\mathbf{x}_i, \mathbf{u}_i, e_i, h_i), \forall i\}$.
output cost function parameters θ, and synthesized or optimized trajectories $\{(\tilde{\mathbf{x}}_i, \tilde{\mathbf{u}}_i), \forall i\}$.
Let $\tau \leftarrow 0$, initialize θ.
repeat
    Synthesis step or optimization step: synthesizing $\tilde{\mathbf{u}}_i \sim p_{\theta_t}(\mathbf{u}|e_i, h_i)$ by Langevin sampling, or optimizing $\tilde{\mathbf{u}}_i = \arg\min_{\mathbf{u}} C_\theta(\mathbf{x}, \mathbf{u}, e_i, h_i)$, by gradient descent (GD) or iLQR, and then obtain $\tilde{\mathbf{x}}_i$, for each i.
    Analysis step: update $\theta_{\tau+1} = \theta_\tau + \gamma_\tau \hat{L}'(\theta_\tau)$, where $\hat{L}$ is computed according to (4).
    $\tau \leftarrow \tau + 1$.
until $\tau = \tau_{\max}$, the number of iterations.

4 Experiments

The code, more results and details can be found at : http://www.stat.ucla.edu/~yifeixu/meioc

4.1 Experimental setup

We evaluate the proposed energy-based inverse control methods on autonomous driving tasks. We test our methods on two datasets. Massachusetts driving dataset focuses on highways with curved lanes and static scenes while NGSIM US-101 dataset [5] focuses on rich vehicle interactions. Details about dataset statistics and preprocessing pipeline can be found in supplementary. We randomly split each dataset into training and testing sets.

In the task of autonomous driving, the state x_t consists of the coordinate, heading angle and velocity of the car, the control variables u_t consists of steering angle and acceleration, the environment e consists of road condition, speed limit, the curvature of the lane (which is represented by a cubic polynomial), as well as the coordinates of other vehicles. The trajectories of other vehicles are treated as known environment states and assumed to remain unchanged while the ego vehicle is moving, even though the trajectories of other vehicles should be predicted in reality. In this paper, we sidestep this issue and focus on the inverse control problem.

We first use a linear combination of some hand-designed features as the cost function. The vector of features ϕ includes the distance to goal (which is a virtual destination 5 seconds away from the starting point along the lane), the penalty for the collision with other vehicles, the distance to the center of the lane, the leading angle difference from the lane, the speed difference to the speed limit, the L2-norm of the change in controls and the L2-norm of the controls. Feature normalization is adopted to make sure that each feature has the same scale of magnitude. We also parameterize the cost function by a neural network in section 4.4.

As to learning, the weight parameters are initialized by a normal distribution. The controls are also initialized by zeros, which is keeping straight. We normalize the controls, i.e., the steering and acceleration, because their scales are different. Instead of sampling the controls, we sample their changes. We set the number of steps of the Langevin dynamics or the gradient descent to be $l = 64$ and set the step size to be $\delta = 0.2$. The choice of l is a trade-off between computational efficiency and prediction accuracy. For parameter training, we use the Adam optimizer [14].

We use Root Mean Square Error (RMSE) in meters with respect to each timestep t to measure the accuracy of prediction, i.e., $RMSE(t) = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \|\hat{y}_{it} - y_{it}\|^2}$, where N is the number of expert demonstrations, $\hat{y}_{it}$ is the predicted coordinate of the i-th demonstration at time t and y_{it} is the ground truth coordinate of the i-th demonstration at time t. A small RMSE is desired. As a stochastic method, our method is also evaluated by average RMSE and minimum RMSE.

4.2 Single-agent control

We first test our methods, including sampling-based and optimization-based ones, on single-agent control problem. We compare our method with three baseline methods.

- Constant velocity: the simplest baseline with a constant velocity and zero steerings.

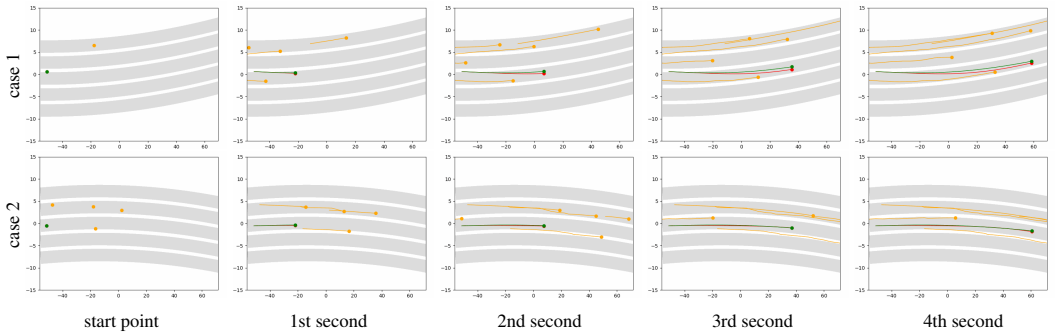


Figure 1: Predicted trajectories for single-agent control. (Green: Predicted trajectories. Red: Ground truths. Orange: Other vehicles. Gray: Lanes.)

- Generative adversarial imitation learning (GAIL) [13]: The original GAIL method was proposed for imitation learning. We use the same setting as in [16], which applies GAIL to the task of modeling human driving behavior. Besides, we change the policy gradient method from Trust Region Policy Optimization (TRPO) [27] to Proximal Policy Optimization (PPO) [28].
- CIOC with Laplace [19]: We implement this baseline with the same iLQR as that in our model.

We can predict the full trajectories with 64 steps of Langevin dynamics (LD) or gradient descent (GD) in roughly 0.1 second. Figure 1 displays 2 examples of qualitative results. For each example, 5 graphs show the positions of all vehicles in the scene as dots after 0 to 4 seconds respectively, along with the predicted trajectories (the green lines) and the ground truths (the red lines). Table 1 and 2 show the results for Massachusetts driving dataset and NGSIM US-101 dataset, respectively. In the last two rows, we provide both average RMSE along with the minimum RMSE for our sampling-based approach. Our methods achieve substantial improvements compared to baseline methods, such as CIOC [19] and GAIL, in terms of testing RMSE. We find that the sampling-based methods outperform the optimization-based methods among our energy-based approaches.

Table 1: Massachusetts driving dataset result.

Method	1s	2s	3s
Constant Velocity	0.340	0.544	0.870
CIOC	0.386	0.617	0.987
ours (iLQR)	0.307	0.491	0.786
ours (GD)	0.257	0.413	0.660
ours (LD) avg	**0.255**	**0.401**	**0.637**
ours (LD) min	*0.157*	*0.354*	*0.607*

Table 2: NGSIM US-101 dataset result.

Method	1s	2s	3s	4s
Constant Velocity	0.569	1.623	3.075	4.919
CIOC	0.503	1.468	2.801	4.530
GAIL	0.367	0.738	1.275	2.360
ours (iLQR)	0.351	0.603	0.969	1.874
ours (GD)	0.318	0.644	1.149	2.138
ours (LD) avg	**0.311**	**0.575**	**0.880**	**1.860**
ours (LD) min	*0.203*	*0.458*	*0.801*	*1.690*

The reason why the method "CIOC with Laplace" performs poorly on both two datasets is due to the fact that its Laplace approximation is not accurate enough for a complex cost function used in the current tasks. Our models are more genetic and do not make such an approximation. Instead, they use Langevin sampling for maximum likelihood training. Therefore, our methods can provide more accurate prediction results.

The problem of GAIL is due to its model complexity. GAIL parameterizes its discriminator, policy and value function by MLPs. Designing optimal MLP structures of these components for GAIL is challenging. Our method only needs to design a single architecture for the cost function. Additionally, our method for optimal control is performed by simulating the trajectory of actions and states according to the learned cost function taking into account the future information. In contrast, GAIL relies on its learned policy net for step-wise decision making.

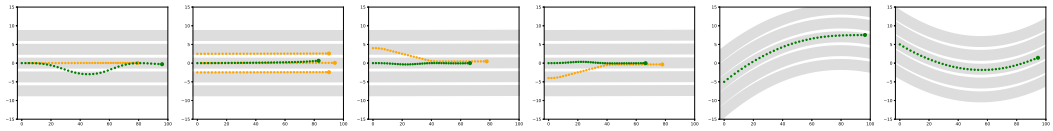
Compared with gradient descent (optimization-based approach), Langevin dynamics-based method can obtain smaller errors. One reason is that the sampling-based approach rigorously maximizes the log-likelihood of the expert demonstrations during training, while the optimization-based approach is just a convenient approximation. The other reason is that the Gaussian noise term in each Langevin step helps to explore the cost function and avoid sub-optimal solutions.

4.3 Corner case testing with toy examples

Corner cases are important for model evaluation. We construct 6 typical corner cases to test our model. Figure 2 shows the predicted trajectories by our method for several cases. Figures 2(a) and (b)

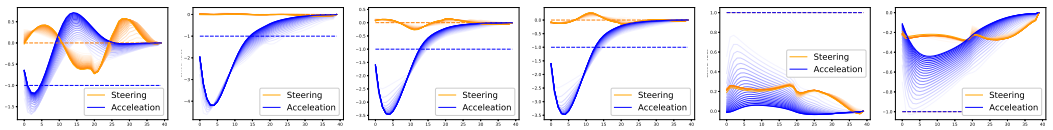
show two cases of sudden braking. In each of the cases, a vehicle (orange) in front of the ego vehicle (green) is making a sudden brake. In case (a), there are not any other vehicles moving alongside the ego vehicle, so it is predicted to firstly change the lane, then accelerate past the vehicle in front, and return to its previous lane and continue its driving. In case (b), two vehicles are moving alongside the ego vehicle. The predicted trajectory shows that the ego vehicle is going to trigger a brake to avoid a potential collision accident. Figures 2(c) and (d) show two cases in cut-in situation. In each case, a vehicle is trying to cut in from the left or right lane. The ego vehicle is predicted to slow down to ensure the safe cut-in of the other vehicle. Figures 2(e) and (f) show two cases in the large lane curvature situation, where our model can still perform well to predict reasonable trajectories.

Figure 3 shows the corresponding plots of the predicted controls over time steps. In each plot, blue lines stand for acceleration and orange lines stand for steering. The dash lines represent the initialization of the control for Langevin sampling, which is actually the control at the last time step in the history trajectory. We use 64 Langevin steps to sample the controls from the learned cost function. We plot the predicted control over time for each Langevin step. The curves with more numbers of Langevin steps appear darker. Thus, the darkest solid lines are the final predicted control.



(a) (b) (c) (d) (e) (f)

Figure 2: Predicted trajectories in corner cases. (Green : predicted trajectories. Orange : trajectories of other vehicles. Gray: Lanes.)



(a) (b) (c) (d) (e) (f)

Figure 3: Predicted control through time. (dash lines: initial values of the controls for Langevin sampling. solid lines: predicted controls over time steps. Blue: control of acceleration. Orange: control of steering.)

In short, this experiment demonstrates that our method is capable of learning a reasonable cost function to handle corner cases, such as situations of sudden braking, lane cut-in, and making turns in curved lanes.

4.4 Evaluation of different cost functions

The Neural network is a powerful function approximator. It is capable of approximating any complex nonlinear function given sufficient training data, and it is also flexible to incorporate prior information, which in our case are the manually designed features. In this experiment, we replace the linear cost function in our sampling-based approach with a neural network. Specifically, we design a cost function by multilayer perceptron (MLP), where we put three layers on top of the vector of hand-designed features: $C_\theta(\mathbf{x}, \mathbf{u}, e, h) = f(\phi(\mathbf{x}, \mathbf{u}, e, h))$, where f contains 2 hidden layers and 1 output layer. We also consider using a 1D CNN that takes into account the temporal relationship inside the trajectory for the cost function. We add four 1D convolutional layers on top of the vector of hand-designed features, where the kernel size in each layer is 1×4. The numbers of channels are $\{32, 64, 128, 256\}$ and the numbers of strides are $\{2, 2, 2, 1\}$ for different layers, respectively. One fully connected layer with a single kernel is attached at the end.

Table 3 shows the performance of different designs of cost functions. We can see that improvements can be obtained by using cost functions parameterized by either MLP or CNN. Neural network provides nonlinear connection layers as a transformation of the original input features. This implies that there are some internal connections between the features and some temporal connections among feature vectors at different time steps.

4.5 Multi-agent control

In the setting of single-agent control, the future trajectories of other vehicles are assumed to be known (e.g., they are predicted by a prediction method) and they remain unchanged no matter how the ego

Table 3: Performance of models with different cost functions.

Method (RMSE)	1s	2s	3s
Linear	0.255	0.401	0.637
MLP	0.237	0.379	0.607
CNN	**0.234**	**0.372**	**0.572**

Table 4: Performance comparison in multi-agent control on NGSIM US-101 dataset. RMSEs are reported.

Method (RMSE)	1s	2s	3s	4s
Constant Velocity	0.569	1.623	3.075	4.919
PS-GAIL	0.602	1.874	3.144	4.962
ours (multi-agent)	**0.365**	**0.644**	**1.229**	**2.262**

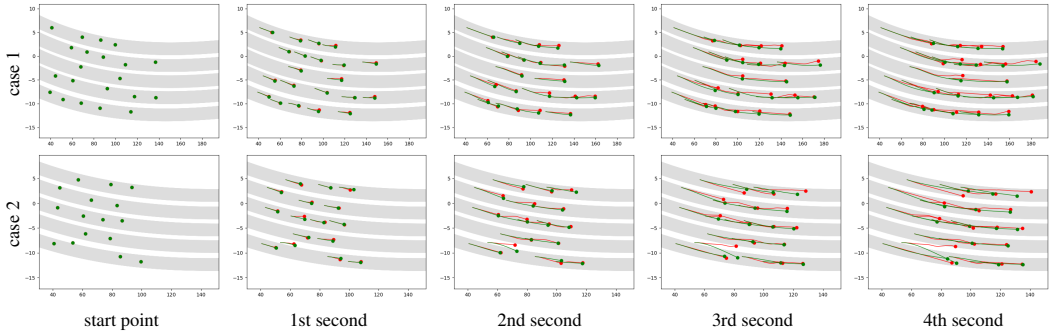


Figure 4: Predicted trajectories for multi-agent control. (Green: Predicted trajectories. Red: Ground truths. Gray: Lanes.)

vehicle moves. We extend our framework to multi-agent setting. In this setting, we simultaneously control all vehicles in the scene. The controls of other vehicles are used to predict the trajectories of other vehicles.

Suppose there are K agents, and every agent in the scene can be regarded as a general agent. The state and control space are Cartesian products of the individual states and controls respectively, i.e., $\mathbf{X} = (\mathbf{x}^k, k = 1, 2, ..., K), \mathbf{U} = (\mathbf{u}^k, k = 1, 2, ..., K)$. All the agents share the same dynamic function, which is $x_t^k = f(x_{t-1}^k, u_t^k) \forall k = 1, 2, ..., K$. The overall cost function are set to be the sum of each agent $C_\theta(\mathbf{X}, \mathbf{U}, e, h) = \sum_{k=0}^{K} C_\theta(\mathbf{x}^k, \mathbf{u}^k, e, h^k)$. Thus, the conditional probability density function becomes $p_\theta(\mathbf{U}|e, h) = \frac{1}{Z_\theta(e,h)} \exp[-C_\theta(\mathbf{X}, \mathbf{U}, e, h)]$, where $Z_\theta(e, h)$ is the intractable normalizing constant.

We compare our method with the following baselines for multi-agent control.

- Constant velocity: The simplest baseline with a constant velocity and zero steerings.
- The parameter sharing GAIL (PS-GAIL) [4] [3]: This method extends single-agent GAIL and Parameter Sharing Trust Region Policy Optimization (PS-TRPO) [11] to enable imitation learning in the multi-agent context.

We test our method on the NGSIM US-101 dataset. We use a linear cost function setting for each agent in this experiment. The maximum number of agents is 64. Figure 4 shows two examples of the qualitative results, one per row. The first to fifth columns show the positions of all vehicles in the scene as dots after 0 to 4 seconds respectively, along with the predicted trajectories (the green lines) and the ground truths (the red lines). Table 4 shows a comparison of performance between our method and the baselines. Results show that our method can work well in the multi-agent control scenario.

5 Conclusion

This paper studies the fundamental problem of learning the cost function from expert demonstrations for continuous optimal control. We study this problem in the framework of energy-based model, and we propose sampling-based method and optimization-based modification to learn the cost function.

Unlike previous method for continuous inverse control [19], we learn the model by maximum likelihood using Langevin sampling, without resorting to Laplace approximation. This is a possible reason for improvement over previous method. The Langevin sampling or MCMC sampling in general also has the potential to avoid sub-optimal modes.

Our method is generally applicable, and can learn non-linear and non-Markovian cost functions.

Acknowledgement

The work is supported by NSF DMS-2015577.

References

[1] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social lstm: Human trajectory prediction in crowded spaces. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[2] Alberto Bemporad, Manfred Morari, Vivek Dua, and Efstratios N Pistikopoulos. The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20, 2002.

[3] Raunak P. Bhattacharyya, Derek J. Phillips, Changliu Liu, Jayesh K. Gupta, Katherine Driggs-Campbell, and Mykel J. Kochenderfer. Simulating emergent properties of human driving behavior using multi-agent reward augmented imitation learning. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, May 2019.

[4] Raunak P Bhattacharyya, Derek J Phillips, Blake Wulfe, Jeremy Morton, Alex Kuefler, and Mykel J Kochenderfer. Multi-agent imitation learning for driving simulation. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1534–1539. IEEE, 2018.

[5] J. Colyar and J. Halkias. Us highway dataset. Federal Highway Administration (FHWA), Tech. Rep. FHWA-HRT-07-030, 2007.

[6] Nachiket Deo, Akshay Rangesh, and Mohan M Trivedi. How would surround vehicles move? a unified framework for maneuver classification and motion prediction. *IEEE Transactions on Intelligent Vehicles*, 3(2):129–140, 2018.

[7] Chelsea Finn, Paul Christiano, Pieter Abbeel, and Sergey Levine. A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. *NIPS Workshop on Adversarial Training*, 2016.

[8] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning (ICML)*, pages 49–58, 2016.

[9] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2672–2680. 2014.

[10] Agrim Gupta, Justin Johnson, Li Fei-Fei, Silvio Savarese, and Alexandre Alahi. Social gan: Socially acceptable trajectories with generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[11] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 66–83. Springer, 2017.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision (ICCV)*, pages 1026–1034, 2015.

[13] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 4565–4573, 2016.

[14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1097–1105, 2012.

[16] Alex Kuefler, Jeremy Morton, Tim Wheeler, and Mykel Kochenderfer. Imitating driver behavior with generative adversarial networks. In *Intelligent Vehicles Symposium (IV), 2017 IEEE*, pages 204–211, 2017.

[17] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[18] Namhoon Lee, Wongun Choi, Paul Vernaza, Christopher B Choy, Philip HS Torr, and Manmohan Chandraker. Desire: Distant future prediction in dynamic scenes with interacting agents. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 336–345, 2017.

[19] Sergey Levine and Vladlen Koltun. Continuous inverse optimal control with locally optimal examples. In *International Conference on Machine Learning (ICML)*, 2012.

[20] Weiwei Li and Emanuel Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *Proceedings of the First International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, pages 222–229, 2004.

[21] Yunzhu Li, Jiaming Song, and Stefano Ermon. Infogail: Interpretable imitation learning from visual demonstrations. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 3812–3822, 2017.

[22] Mathew Monfort, Anqi Liu, and Brian D. Ziebart. Intent prediction and trajectory forecasting via predictive inverse linear-quadratic regulation. In *Proceedings of the Twenty-Ninth Conference on Artificial Intelligence (AAAI)*, pages 3672–3678, 2015.

[23] Radford M Neal et al. Mcmc using hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2(11):2, 2011.

[24] Philip Polack, Florent Altché, Brigitte d'Andréa Novel, and Arnaud de La Fortelle. The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles? In *IEEE Intelligent Vehicles Symposium*, pages 812–818, 2017.

[25] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine learning*, 62(1-2):107–136, 2006.

[26] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[27] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, pages 1889–1897, 2015.

[28] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[29] Emanuel Todorov. Optimal control theory. *Bayesian brain: probabilistic approaches to neural coding*, pages 269–298, 2006.

[30] Anirudh Vemula, Katharina Muelling, and Jean Oh. Social attention: Modeling attention in human crowds. In *Proceedings of the International Conference on Robotics and Automation (ICRA) 2018*, May 2018.

[31] Markus Wulfmeier, Peter Ondruska, and Ingmar Posner. Maximum entropy deep inverse reinforcement learning. *arXiv preprint arXiv:1507.04888*, 2015.

[32] Jianwen Xie, Yang Lu, Song-Chun Zhu, and Yingnian Wu. A theory of generative convnet. In *International Conference on Machine Learning (ICML)*, pages 2635–2644, 2016.

[33] Jianwen Xie, Zilong Zheng, Ruiqi Gao, Wenguan Wang, Song-Chun Zhu, and Ying Nian Wu. Learning descriptor networks for 3d shape synthesis and analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8629–8638, 2018.

[34] Jianwen Xie, Zilong Zheng, Ruiqi Gao, Wenguan Wang, Song-Chun Zhu, and Ying Nian Wu. Revisiting video saliency prediction in the deep learning era. *IEEE transactions on pattern analysis and machine intelligence*, 2020.

[35] Jianwen Xie, Song-Chun Zhu, and Ying Nian Wu. Synthesizing dynamic patterns by spatial-temporal generative convnet. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7093–7101, 2017.

[36] Jianwen Xie, Song-Chun Zhu, and Ying Nian Wu. Learning energy-based spatial-temporal generative convnets for dynamic patterns. *IEEE transactions on pattern analysis and machine intelligence*, 2019.

[37] Tianyang Zhao, Yifei Xu, Mathew Monfort, Wongun Choi, Chris Baker, Yibiao Zhao, Yizhou Wang, and Ying Nian Wu. Multi-agent tensor fusion for contextual trajectory prediction. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[38] Song Chun Zhu, Yingnian Wu, and David Mumford. Filters, random fields and maximum entropy (FRAME): Towards a unified theory for texture modeling. *International Journal of Computer Vision*, 27(2):107–126, 1998.

[39] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 8, pages 1433–1438, 2008.

A Appendix: Theoretical understanding

A.1 Moment matching

For simplicity, consider the linear cost function $C_\theta(\mathbf{x}, \mathbf{u}, e, h) = \langle \theta, \phi(\mathbf{x}, \mathbf{u}, e, h) \rangle$. At the convergence of the optimization-based learning algorithm, which has the same analysis step (2) as the sampling-based algorithm, we have $\hat{L}'(\theta) = 0$, so that

$$\frac{1}{n} \sum_{i=1}^{n} \phi(\tilde{\mathbf{x}}_i, \tilde{\mathbf{u}}_i, e_i, h_i) = \frac{1}{n} \sum_{i=1}^{n} \phi(\mathbf{x}_i, \mathbf{u}_i, e_i, h_i), \tag{7}$$

where the left-hand side is the average of the optimal behaviors obtained by (6), and the right-hand side is the average of the observed behaviors. We want the optimal behaviors to match the observed behaviors on average. We can see the above point most clearly in the extreme case where all $e_i = e$ and all $h_i = h$, so that $\phi(\tilde{\mathbf{x}}, \tilde{\mathbf{u}}, e, h) = \frac{1}{n} \sum_{i=1}^{n} \phi(\mathbf{x}_i, \mathbf{u}_i, e, h)$, i.e., we want the optimal behavior under the learned cost function to match the average observed behaviors as far as the features of the cost function are concerned. Note that the matching is not in terms of raw trajectories but in terms of the features of the cost function.

A.2 Adversarial learning

We can also justify this optimization-based algorithm outside the context of probabilistic model as adversarial learning. To this end, we re-think about the inverse optimal control, whose goal is not to find a probabilistic model for the expert trajectories. Instead, the goal is to find a suitable cost function for optimal control, where we care about the optimal behavior, not the variabilities of the observed behaviors. Define the value function

$$V = \frac{1}{n} \sum_{i=1}^{n} \left[C_\theta(\tilde{\mathbf{x}}_i, \tilde{\mathbf{u}}_i, e_i, h_i) - C_\theta(\mathbf{x}_i, \mathbf{u}_i, e_i, h_i) \right], \tag{8}$$

Then $\hat{L}'(\theta) = \frac{\partial}{\partial \theta} V$, so that the analysis step (2) increases V. Thus the optimization step and the analysis step play an adversarial game $\max_\theta \min_{\tilde{\mathbf{u}}_i, \forall i} V$, where the optimization step seeks to minimize V by reducing the costs, while the analysis step seeks to increase V by modifying the cost function. More specifically, the optimization step finds the minima of the cost functions to decrease V, whereas the analysis step shifts the minima toward the observe trajectories in order to increase V.

B Appendix: Experiment details

B.1 Dataset

Here we provide the detailed dataset information of the two datasets we used in experiments.

(1) Massachusetts driving dataset: This is a private dataset collected from an autonomous vehicle during repeated trips on a stretch of highway. The dataset includes vehicle states, controls collected by the hardware on the vehicle, and environment information. This dataset has a realistic driving scenario, which includes curved lanes and complex static scenes. To solve the problem of noisy GPS signal, Kalman filtering is used to denoise the data. The number of trajectories is 44,000. Each trajectory is 4 seconds long with a time interval equal to 0.1 seconds.

(2) NGSIM US-101 [5]: This dataset consists of real highway traffics captured at 10Hz over a time span of 45 minutes. Compared to Massachusetts driving dataset, NGSIM US-101 contains richer

vehicle interactions. Each control needs to consider other nearby vehicles. We preprocess the data by dividing the data into blocks (or scenes), each of which is 50 frames (i.e., 5 seconds) long. For each block, we treat the first 10 frames as history and the rest 40 frames for prediction.) There are totally 831 scenes with 96,512 5-second-long vehicle trajectories. No control variables are provided. Thus, we need to infer the controls of each vehicle given the vehicle states. Assuming the bicycle model [24] dynamics, we perform an inverse-dynamics optimization using gradient descent to infer the controls and adjust the positions. The overall Root Mean Square Error (RMSE) between the ground truth GPS positions and the adjusted ones is 0.97 meters. The preprocessed trajectories have perfect dynamics with noiseless and smooth control sequences and GPS coordinates.

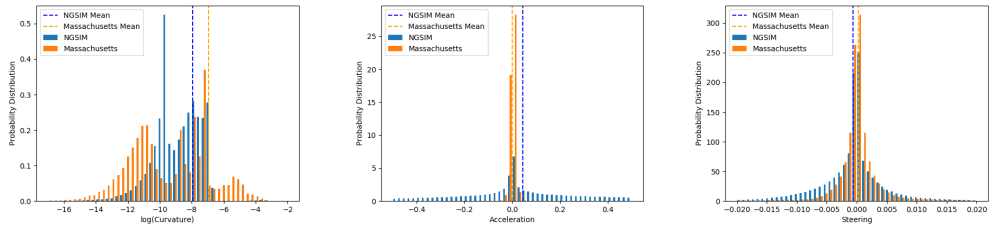


Figure 5: Statistics comparison between Massachusetts Driving dataset and NGSIM US-101 dataset

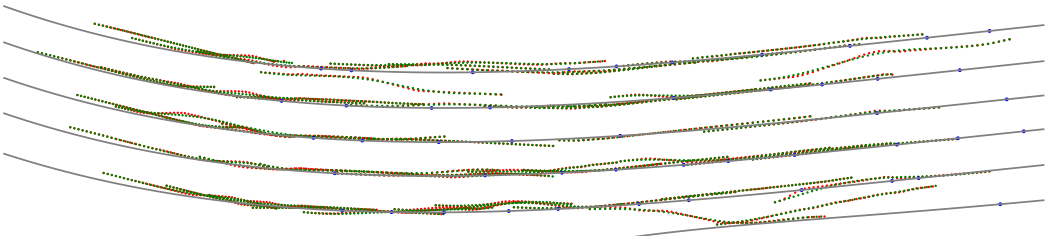


Figure 6: Preprocessed and raw trajectories in NGSIM US-101 dataset. (Green : preprocessed; Red: raw)

We provide a statistical comparison of two datasets. Figure 5 shows histograms of the road curvature, steering control, and acceleration control, respectively. We can see that Massachusetts driving dataset has bigger curvatures, while NGSIM US-101 dataset has bigger accelerations.

Figure 6 shows a sample of scene in the preprocessed NGSIM US-101 dataset. There are over 100 vehicles in this scene all moving from left to right. Each dot sequence represents the state trajectory of one vehicle from time frame 1 to 50. The green dot sequences represent the preprocessed trajectories, and the red ones represent the raw trajectories.

B.2 Network Structure

Table 5 shows a list of the hand-crafted features defined in section 4.1. They are differentiable feature extractors and construct the first layers of the networks that parameterize the cost functions, which are used in our methods (with Langevin, GD, and iLQR) and baselines. Table 6 and 7 are the MLP and CNN network structures we used in section 4.3, respectively. The network structure in Table 6 only considers information is each time frame, while the one shown in table 7 takes into account the temporal information. In table 7, the length of trajectory is 40. The numbers of dimensions of x, u and e are 6, 2, and 29, respectively.

B.3 Training Details

Normalization. For the control values, we normalize each control value to have zero mean and unit variance. We also normalize each hand-crafted feature by dividing the feature by its mean.

Optimizer. We use Adam optimizer to train our models. All β_1 and β_2 are set to be 0.5. All model parameters are initialized by the Kaiming He initialization [12] method. We provide the learning rates and decay rates for different energy functions below:

Table 5: A list of hand-crafted features

Distance to the goal point in longitude	L2 norm of the accelerating value
Distance to the goal point in latitude	L2 norm of the steering value
Distance to the center of the lane	Difference to last accelerating value
Difference to the speed limit	Difference to last steering value
Difference to the direction of the lane	Distance to the nearest obstacle

Table 6: MLP cost function network

Layer	Output Size
concat($[x, u, e]$)	6+2+29
hand-craft feature	10
Linear, LeakyReLU	64
Linear, LeakyReLU	64
Linear	1

Table 7: CNN cost function network

Layer	Output Size	Stride
concat($[x, u, e]$)	1 ×40×(6+2+29)	-
hand-craft feature	1 ×40×10	-
1×4 Conv1d, LeakyReLU	1 ×19×32	2
1×4 Conv1d, LeakyReLU	1×9×64	2
1×4 Conv1d, LeakyReLU	1×4×128	2
1×4 Conv1d, LeakyReLU	1×1×256	1
Linear	1×1×1	-

- Linear: Learning rate : 0.1; Exponential decay rate : 0.999
- MLP cost function: Learning rate : 0.005; No exponential decay
- CNN cost function: Learning rate : 0.005; Exponential decay rate : 0.999

Langevin Dynamics. To prevent from the exploding gradients problem, we use gradient clipping in each step for Langevin dynamics (or gradient descent), with a maximum limit equal to 0.1. The step size of Langevin dynamics is set to be 0.1 and the number of steps is 64. All settings are the same for the gradient descent setting. In appendix C, we study the influence of different choices of numbers of Langevin steps and step sizes.

iLQR. In iLQR solver, we apply grid search for the learning rate from 0.001 to 1. The maximum number of steps is set to be 100. When the difference of controls between two steps is smaller than 0.001, the early stop is triggered. In the experiment, the average number of iLQR steps is around 30.

Training time. We use a mini-batch of size 1,024 during training. For each epoch, we shuffle the whole training set. That is, in each epoch, the algorithm has a different mini-batch division. The following lists the running time of each epoch for each model while training on Massachusetts driving dataset. The training times are recorded in a PC with a CPU i9-9900 and a GPU Tesla P100.

- Linear: It takes around 3 minutes for each epoch, and 40 epochs in total. Training time is dominated by the number of steps we choose. If we decrease the number of steps to 8, it only takes 40 seconds per epoch.
- MLP / CNN cost function: It takes around 10 minutes for each epoch, and 40 epochs in total.
- GAIL: GAIL uses a smaller mini-batch size of 64. It takes around 10 minutes for each epoch, and 40 epochs in total.
- CIOC: It takes around 1 minute for each epoch, and 40 epochs in total.

C Appendix: Influence of hyperparameters

We investigate the influence of different choices of some hyperparameters in training, such as the number of Langevin steps l and the Langevin step size δ. Figures 7(a) and (b) depict training curves of the models with different l and δ, respectively. The models are trained on the Massachusetts driving dataset. Each curve reports the testing average RMSEs over training epochs. For testing, we use the same l and δ as those in training.

We observe that the learning is quite stable in the sense that the testing errors drop smoothly with an increasing number of training epochs. Figure 7(c) summarize the final average RMSEs for varying l and δ. We observe that in general, the testing performance increases as l or δ increases. However, the more Langevin steps we use, the more time consuming the sampling process is. A trade-off between accuracy and efficiency should be considered when choosing l and δ.

We also study, given a trained model, how different choices of l and δ in testing can affect the performance of the model. Figure 7(d) shows the average RMSEs of trajectories that are sampled from a learned model by using different numbers of Langevin steps l and step sizes δ. The model we use is with a linear cost function and trained with $l = 64$ and $\delta = 0.1$. We observe that: in the testing stage, using Langevin step sizes smaller than that in the training stage may take more Langevin steps to converge, while using larger ones may lead to a non-convergence issue. Thus, we suggest using the same l and δ in both training and testing stages for optimal performance.

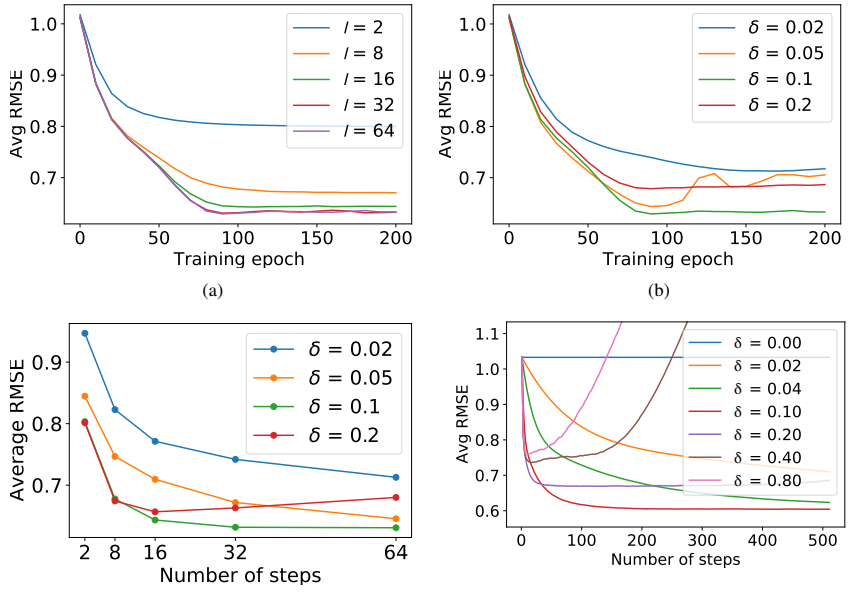


Figure 7: Influence of hyperparameters. (a) a line chart of testing average RMSEs over training epochs for different numbers of Langevin steps used in training. (b) a line chart of testing average RMSEs over training epochs for different Langevin step sizes used in training. (c) Influence of different numbers of Langevin steps and step sizes used in training. (d) Influence of different numbers of Langevin steps and step sizes used in testing.