Energy-Based Continuous Inverse Optimal Control

Yifei Xu, Jianwen Xie, Tianyang Zhao, Chris Baker, Yibiao Zhao and Ying Nian Wu

***Abstract*—The problem of continuous inverse optimal control (over finite time horizon) is to learn the unknown cost function over the sequence of continuous control variables 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 (1) synthesis step: sample the synthesized trajectories from the current probability density using the Langevin dynamics via back-propagation through time, and (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. Moreover, to make the sampling or optimization more efficient, we propose to train the energy-based model simultaneously with a top-down trajectory generator via cooperative learning, where the trajectory generator is used to fast initialize the synthesis step of the energy-based model. We demonstrate the proposed methods on autonomous driving tasks, and show that they can learn suitable cost functions for optimal control.**

***Index Terms*—Inverse optimal control; Energy-based models; Langevin dynamics; Cooperative learning.**

I. Introduction

A. Background and motivation

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 [1]. For instance, in autonomous driving, the iLQR (iterative linear quadratic regulator) algorithm is a commonly used optimization algorithm [2], [3]. 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 sequences of actions. Learning the cost function in this way is called continuous inverse optimal control (IOC) problem.

In this article, we study the fundamental problem of continuous inverse optimal control in the framework of energy-based model [4]. Originated from statistical physics, an energy-based model (EBM) 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. The energy function maps the input into a scalar, which is called energy. 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 estimation (MLE) 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 Markov chain Monte Carlo (MCMC), such as Langevin dynamics [5]. 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.

We need to point out that MLE is the most commonly used method for learning energy-based models, due to its asymptotic optimality. Among all the asymptotically unbiased estimators, MLE is the most accurate in terms of asymptotic variance [6]. Alternative methods for learning EBMs include contrastive divergence (CD) [7] and noise contrastive estimation (NCE) [8]. Contrastive divergence replaces MCMC by one or a few steps of MCMC sampling initialized from observed examples, and as a result, it has a big bias. NCE estimates the energy function discriminatively by recruiting a noise distribution to produce

Y. Xu is with the Department of Statistics, University of California, Los Angeles, CA 90095, USA. E-mail: fei960922@ucla.edu

J. Xie is with the Cognitive Computing Lab, Baidu Research, Bellevue, WA 98004, USA. E-mail: jianwen@ucla.edu

T. Zhao is with the Department of Statistics, University of California, Los Angeles, CA 90095, USA. E-mail: tyzhao@ucla.edu

C. Baker is with iSee Inc., Cambridge, MA 02139, USA. E-mail: chris-baker@isee.ai

Y. Zhao is with iSee Inc., Cambridge, MA 02139, USA. E-mail: yz@isee.ai

Y. N. Wu is with the Department of Statistics, University of California, Los Angeles, CA 90095, USA. E-mail: ywu@stat.ucla.edu

negative or contrastive examples against the observed examples which are treated as positive examples. For accurate estimation, the noise distribution should have substantial overlap with the data distribution. For high dimensional observations such as trajectories, it is difficult to find such a noise distribution. If the noise distribution does not have sufficient overlap with the data distribution, the estimate will have a big variance. Therefore, the maximum likelihood training is a more preferable algorithm to train EBMs.

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 into an optimization step while keeping the analysis step 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.

Moreover, we propose another novel energy-based IOC framework, where the energy-based model is trained with a top-down trajectory generator that serves as a fast initializer of the Langevin sampling of the energy-based model through a cooperative learning manner [9], [10]. Within each cooperative learning iteration, the trajectory generator generates initial trajectories to initialize a finite-step Langevin dynamics that samples from the energy-based model, and then the energy-based model is trained by comparing the expert trajectories and the synthesized trajectories. After that, the trajectory generator learns from how the MCMC changes its initial generated trajectories. The proposed framework belongs to the "fast thinking initializer and slow thinking solver" framework [11]. The trajectory generator plays the role of the fast thinking initializer because its generation of trajectory is accomplished by direct mapping, while the energy-based model plays the role of the slow thinking solver because it learns a cost function in the form of a conditional energy function, so that the trajectory can be synthesized by minimizing the cost function, or more rigorously by sampling from the energy-based model. The trajectory generator is like a policy, while the energy-based model is like a planner. Compared to GAN-type method, ours is equipped with an iterative refining process (slow thinking) guided by a learned cost (energy) function.

We empirically demonstrate the proposed energy-based IOC methods on autonomous driving in both single-agent and multi-agent scenarios, and show that the proposed methods can learn suitable cost functions for optimal control.

B. Related work

The following are research themes related to our work.

(1) Maximum entropy framework. Our work follows the maximum entropy framework of [12] for learning the cost function. Such a framework has also been used previously for generative modeling of images [13] and Markov logic network [14]. In this framework, the energy function is a linear combination of hand-designed features. Recently, [15] generalized this framework to a 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 the gradient descent algorithm or a built-in optimal control algorithm as the inner loop for learning.

(2) ConvNet-EBM. Recently, [16], [17], [18], [19], [20], [21] applied deep energy-based models to various generative modeling tasks, where the energy functions are parameterized by ConvNets [22], [23]. 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 [24], [25], including adversarial learning methods [26], [27], [28], [25], involve learning a policy in addition to the cost function. In our work, the energy-based IOC framework (without an extra trajectory generator) does not learn any policy, and it only learns a cost function (i.e., the energy 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 (IOC). The IOC problem has been studied by [29] and [30]. In [29], the dynamics is linear and the cost function is quadratic, so that the normalizing constant can be computed by a dynamic programming scheme. In [30], 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 [31], [32], [33], [34], [35], [36]. These methods directly predict the coordinates and do not consider control and dynamic models. Thus, they cannot be used for inverse optimal control.

(6) Cooperative Learning. Our joint training framework for IOC follows the generative cooperative learning algorithm (i.e., the CoopNets algorithm) of [10] for training the cost function in the EBM and the trajectory generator. Such a learning algorithm has also been applied previously to image generation [10], video generation [10], 3D shape generation [20], supervised conditional learning [11], and unsupervised image-to-image translation [37]. The CoopVAEBM [38] is a variant of the CoopNets algorithm by replacing the generic generator with a variational auto-encoder (VAE) [39]. The CoopFlow [40] is another variant of the CoopNets algorithm by changing the generator into a normalizing flow [41].

C. Contributions

The contributions of our work are as follows.

- We propose an energy-based method for continuous inverse optimal control based on Langevin sampling

via back-propagation through time. To the best of our knowledge, this is the first work that studies MCMC sampling-based inverse optimal control. Such an "analysis by synthesis" learning scheme makes our work essentially different from [12], [29], [30].
- We also propose an optimization-based method as a convenient approximation of the MCMC sampling under the proposed energy-based learning framework. The modified algorithm becomes an "analysis by optimization" scheme.
- We evaluate the proposed methods on autonomous driving tasks for trajectory prediction. We apply our framework to both single-agent system and multi-agent system, with both linear cost function and neural network non-linear cost function. This is the first work to study vehicle trajectory prediction under the energy-based framework.
- We also propose to train an energy-based model together with a policy-like trajectory generator, which serves as a fast initializer for the Langevin sampling, in a cooperative learning scheme.
- We conduct extensive ablation studies to analyze the effects of the key components and hyperparameters of the proposed frameworks to understated the model behaviors.

D. Organization

The rest of our paper is organized as follows: Section II presents the proposed framework of the energy-based inverse optimal control. Section III presents the proposed joint training framework, in which the energy-based model is trained simultaneously with a trajectory generator as amortized sampler. Qualitative and quantitative results of experiments are shown in Section IV. Conclusion of the paper is given in Section V.

II. Energy-based inverse optimal control

A. 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 parameterize 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 $D = \{(\mathbf{x}_i, \mathbf{u}_i, e_i, h_i), i = 1, ..., n\}$.

B. 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 $C_\theta(\mathbf{x}, \mathbf{u}, e_i, h_i)$.

C. Sampling-based inverse optimal control

The parameters θ can be learned by maximum likelihood. The log-likelihood is given by

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

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

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

which follows the property of the normalizing constant $\frac{\partial}{\partial \theta} \log Z_\theta(e, h) = -\mathrm{E}_{p_\theta(\mathbf{u}|e,h)} \left(\frac{\partial}{\partial \theta} C_\theta(\mathbf{x}, \mathbf{u}, e, h) \right)$.

In order to approximate the above expectation, we can generate multiple random samples via $\tilde{\mathbf{u}}_i \sim p_\theta(\mathbf{u}|e, h)$, which generates each 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 [42], 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 Equation (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 $\frac{\partial}{\partial \theta} C_\theta(\mathbf{x}, \mathbf{u}, e, h) = \phi(\mathbf{x}, \mathbf{u}, e, h)$, 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 an efficient gradient-based MCMC, the Langevin dynamics, which iterates the following steps:

$$\begin{aligned} \mathbf{u}_{s+1} &= \mathbf{u}_s + \frac{\delta^2}{2} \frac{\partial}{\partial \mathbf{u}} \log p_\theta(\mathbf{u}_s|e, h) + \delta \mathbf{z}_s, \\ &= \mathbf{u}_s - \frac{\delta^2}{2} \frac{\partial}{\partial \mathbf{u}} C_\theta(\mathbf{x}_s, \mathbf{u}_s, e, h) + \delta \mathbf{z}_s, \end{aligned} \tag{5}$$

where s indexes the time step, δ is the step size, and $\mathbf{z}_s \sim \mathcal{N}(0, I)$ the Brownian motion 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, with $\mathbf{u}_0$ ($\mathbf{u}$ at the initial time step) being initialized by Gaussian white noise. The gradient descent part $\mathbf{u}_{s+1} = \mathbf{u}_s - \frac{\delta^2}{2}\frac{\partial}{\partial \mathbf{u}} C_\theta(\mathbf{x}_s, \mathbf{u}_s, e, h)$ of Equation (5) is a mode seeking process that minimizes the cost function C_θ, while the added Gaussian noise $\mathbf{z}_s$ will prevent the samples from being trapped by local minima. According to the second law of thermodynamics [43], as $s \to \infty$ and $\delta \to 0$, $\mathbf{u}_s$ becomes an exact sample from $p_\theta(\mathbf{u}|e, h)$ under some regularity conditions. A Metropolis-Hastings [44] step can also be added to correct for the error due to discrete time step in Equation (5), but most existing works, such as [45], [4], [46], have shown that this can be ignored in practice if a small enough step size δ is used. Thus, for computational efficiency, in this work, we do not have the Metropolis-Hastings correction in our implementation.

The gradient term $\partial C_\theta(\mathbf{x}, \mathbf{u}, e, h)/\partial \mathbf{u}$ 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.

D. Optimization-based inverse optimal control

We can remove the noise term in Langevin dynamics in Equation (5), to make it a gradient descent process, i.e., $\mathbf{u}_{s+1} = \mathbf{u}_s - \eta \frac{\partial}{\partial \mathbf{u}} C_\theta(\mathbf{x}_s, \mathbf{u}_s, e, h)$, 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 algorithm in the context of the energy-based model in Equation (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 Equation (3).

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 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 Equation (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. In this matching, we do not care about modeling the variabilities in the observed behaviors. In the case of different (e_i, h_i) for $i = 1, ..., n$, the matching may not be exact for each combination of (e, h). However, such mismatches may be detected by new features which can be included in the features of the cost function.

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

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

then $\hat{L}'(\theta) = \frac{\partial}{\partial \theta} V$, so that the analysis step increases V. 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.

E. Energy-based IOC algorithm

Algorithm 1 and Algorithm 2 present the sampling-based and optimization-based learning algorithms, respectively. 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.

III. Joint training

A. A trajectory generator model as a fast initializer

Both sampling-based method via Langevin dynamics and optimization-based method via gradient descent are based on iterative process, which will benefit from good initialization. A good initial point can not only greatly shorten the number of iterative steps but also help find the optimal modes of the cost function. Therefore, we propose to train an energy-based model simultaneously with a trajectory generator model that serves as

Algorithm 1 Energy-based IOC with synthesis step

input expert demonstrations $D = \{(\mathbf{x}_i, \mathbf{u}_i, e_i, h_i), \forall i\}$.
output cost function parameters θ, and synthesized trajectories $\{(\tilde{\mathbf{x}}_i, \tilde{\mathbf{u}}_i), \forall i\}$.
Let $\tau \leftarrow 0$, randomly initialize θ.
repeat
  synthesis step: for each i, synthesize $\tilde{\mathbf{u}}_i \sim p_{\theta_t}(\mathbf{u}|e_i, h_i)$ by Langevin sampling and then obtain $\tilde{\mathbf{x}}_i$.
  analysis step: update $\theta_{\tau+1} = \theta_\tau + \gamma_\tau \hat{L}'(\theta_\tau)$, where $\hat{L}'$ is computed according to Equation (4).
  $\tau \leftarrow \tau + 1$.
until $\tau = \tau_{\max}$, the number of iterations.

Algorithm 2 Energy-based IOC with optimization step

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

a fast initializer for the Langevin dynamics or gradient descent of the energy-based model.

The basic idea is to use the trajectory generator model to generate trajectories via ancestral sampling to initialize a finite step Langevin dynamics or gradient descent for training the energy-based model. In return, the trajectory generator model learns from how the Langevin dynamics or gradient descent updates the initial trajectories it generates. Such a cooperative learning strategy is proposed in [10], [11], [40] for image generation.

To be specific, we propose the trajectory generator model that consists of the following two components

$$u_t = F_\alpha(x_{t-1}, \xi_t, e) \tag{9}$$

$$x_t = f(x_{t-1}, u_t) \tag{10}$$

where $t = 1, ..., T$, Equation (9) is the policy model, and Equation (10) is the known dynamic function. $\xi_t \sim \mathcal{N}(0, I)$ is the Gaussian noise vector. The Gaussian noise vectors at different times $(\xi_t, t = 1, ..., T)$ are independent of each other. Given the state x_{t-1} at the previous time step $t-1$ along with the environment condition e, the policy model outputs the action u_t at the current time step t, where the noise vector ξ_t accounts for the randomness in the mapping from x_{t-1} to u_t. F_α is a multi-layer perceptron, where α is the model parameters of the network. The initial state x_0 is assumed to be given.

We denote $\boldsymbol{\xi} = (\xi_t, t = 1, ..., T)$ and $p(\boldsymbol{\xi}) = \prod_{t=1}^{T} p(\xi_t)$. Given the state x_{t-1} and the environment condition e, although x_t is dependent on the action u_t, u_t is generated from ξ_t. In fact, we can write the trajectory generator in a compact form, i.e., $\mathbf{u} = G_\alpha(\boldsymbol{\xi}, e, h)$, where G_α composes F_α and f over time, and we use $h = x_0$ for simplicity in our implementation.

The algorithm for joint training of the energy-based model and the trajectory generator is that: at each iteration, (i) we first sample $\boldsymbol{\xi}_i$ from the Gaussian prior distribution, and then generate the initial trajectories by $\hat{\mathbf{u}}_i = G_\alpha(\boldsymbol{\xi}_i, e_i, h_i)$ for $i = 1, ..., n$. (ii) Starting from the initial trajectories $\{\hat{\mathbf{u}}_i\}$, we sample from the energy-based model by running a finite number of Langevin steps or optimize the cost function by running a finite steps of gradient descent to obtain the updated trajectories $\{\tilde{\mathbf{u}}_i\}$, and then obtain $\{\tilde{\mathbf{x}}_i\}$. (iii) We update the parameters θ of the energy-based model by maximum likelihood estimation, where the computation of the gradient of the likelihood is based on $\{\tilde{\mathbf{u}}_i\}$ and follows Equation (4). (iv) We update the parameters α of the trajectory generator by gradient descent on the loss

$$\hat{l}'_g(\alpha) = \frac{\partial}{\partial \alpha} \left[\frac{1}{n} \sum_{i=1}^{n} ||\tilde{\mathbf{u}}_i - G_\alpha(\boldsymbol{\xi}_i, e_i, h_i)||^2 \right]. \tag{11}$$

Algorithm 3 presents a detailed description of the cooperative training algorithm of an energy-based model and a trajectory generator for inverse optimal control. Synthesis step and optimization step are two options to generate $(\tilde{\mathbf{u}}, \tilde{\mathbf{x}})$.

Algorithm 3 Energy-based IOC with a trajectory generator

input expert demonstrations $D = \{(\mathbf{x}_i, \mathbf{u}_i, e_i, h_i), \forall i\}$.
output cost function parameters θ, trajectory generator parameters α, and synthesized or optimized trajectories $\{(\tilde{\mathbf{x}}_i, \tilde{\mathbf{u}}_i), \forall i\}$.
Let $\tau \leftarrow 0$, randomly initialize θ and α.
repeat
  Initialization step: Initialize $\hat{\mathbf{u}}_i = G_{\alpha_t}(\boldsymbol{\xi}_i, e_i, h_i)$, where $\boldsymbol{\xi}_i \sim p(\boldsymbol{\xi})$ by ancestral sampling, and then obtain $\hat{\mathbf{x}}_i$ for each i.
  Synthesis step or optimization step: Given the initial $\hat{\mathbf{u}}_i$, 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 cost function): Update $\theta_{\tau+1} = \theta_\tau + \gamma_\tau \hat{L}'(\theta_\tau)$, where $\hat{L}'(\theta)$ is computed according to Equation (4).
  Analysis step (update policy model): Update $\alpha_{\tau+1} = \alpha_\tau - \eta_\tau \hat{l}'_g(\alpha_\tau)$, where $\hat{l}'_g(\alpha)$ is computed according to Equation (11).
  $\tau \leftarrow \tau + 1$.
until $\tau = \tau_{\max}$, the number of iterations.

B. Bottom-up and top-down generative models of trajectories

Algorithm 3 presented in the main text is about a joint training of two types of generative models, the energy-based model (we also call it the trajectory evaluator) and the latent variable model (i.e., the trajectory generator). Both of these two models can be parameterized by deep neural networks and they are of opposite directions. The energy-based model has a bottom-up energy function that maps the trajectory to the cost, while the trajectory generator owns a top-down transformation that maps the sequence of noise vectors (i.e., the

latent variables) to the trajectory, as illustrated by the following diagram.

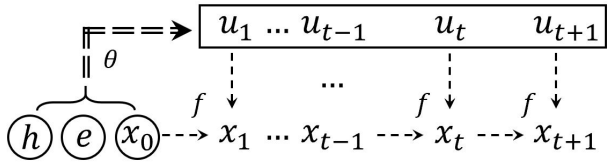
Bottom-up model	Top-down model
cost	**noise sequence**
(energy)	**(latent variables)**
⇑	⇓
trajectory	trajectory
(a) Trajectory evaluator	(b) Trajectory generator
(energy-based model)	(latent variable model)

C. Iterative and non-iterative generations of trajectories

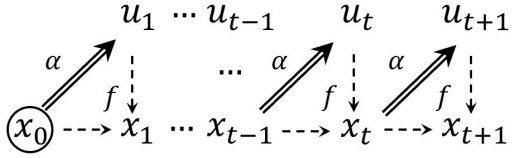
The energy-based model $p_\theta(\mathbf{u}|e,h)$ defines a cost function or an energy function $C_\theta(\mathbf{x},\mathbf{u},e,h)$, from which we can derive the Langevin dynamics to generate $\mathbf{u}$. This is an implicit generation process of $\mathbf{u}$ that iterates the Langevin step in Equation (5).

Figure 1 (a) illustrates the generation process of $(\mathbf{u},\mathbf{x})$. Given (h,e), the Langevin sampling seeks to find $\mathbf{u} = (u_1,...,u_T)$ to minimize the $C_\theta(\mathbf{x},\mathbf{u},e,h)$. The dashed double line arrows indicate iterative generation by sampling in the energy-based model, while the dashed line arrows indicate the known dynamic function $x_t = f(x_{t-1},u_t)$. With the generated action sequence $\mathbf{u} = (u_1,...,u_T)$, the state sequence $\mathbf{x} = (x_1,...,x_T)$ can be easily obtained by applying the dynamic function.

The trajectory generator generates $(\mathbf{u},\mathbf{x})$ via ancestral sampling, $(\mathbf{u},\mathbf{x}) = G_\alpha(\boldsymbol{\xi},e,h)$ which is a non-iterative process to produce $(\mathbf{u},\mathbf{x})$ from the recent history h (We assume $h = x_0$), environment e, and a sequence of noise vectors $\boldsymbol{\xi} = (\xi_1,...\xi_T)$ serving as the latent variables. The generator can unfold over time and can be decomposed into the policy model $u_t = F_\alpha(x_{t-1},\xi_t,e)$ and the dynamic function $x_t = f(x_{t-1},u_t)$ at each time step. The latent variable ξ_t accounts for variation in the policy model at time step t. Figure 1(b) illustrates the generation process of the trajectory generator. The double line arrows indicate the mapping of the policy model, while the dashed line arrows indicate the known dynamic function. The whole process of generating $(\mathbf{u},\mathbf{x})$ is of a dynamic or causal nature in that it directly evolves or unfolds over time.



(a) Trajectory generation via Langevin sampling



(b) Trajectory generation via ancestral sampling

Fig. 1: Trajectory generation by (a) iterative method and (b) non-iterative method.

D. Issue in maximum likelihood training of a single trajectory generator

Let $\boldsymbol{\xi} = (\xi_t, t = 1,...,T)$, where $\xi_t \sim \mathcal{N}(0,I)$ independently over time t. Let $\mathbf{u} = (u_t, t = 1,...,T)$. We have $\mathbf{u} = G_\alpha(\boldsymbol{\xi},e,h) + \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} = (\epsilon_t, t = 1,...,T)$ are observation errors and $\epsilon_t \sim \mathcal{N}(0,\sigma^2 I)$. For notational simplicity, we omit $\mathbf{x}$ and only keep $\mathbf{u}$ in the output of G_α, because $\mathbf{x}$ is just the intermediate output of G_α and $\mathbf{u}$ is determined by $\mathbf{x}$. The trajectory generator defines the joint distribution of $(\mathbf{u},\boldsymbol{\xi})$ conditioned on (e,h) as below,

$$q_\alpha(\mathbf{u},\boldsymbol{\xi}|e,h) = q_\alpha(\mathbf{u}|\boldsymbol{\xi},e,h)p(\boldsymbol{\xi}) \tag{12}$$

where $p(\boldsymbol{\xi}) = \prod_{i=t}^{T} p(\xi_t)$ is the prior distribution and $q_\alpha(\mathbf{u}|\boldsymbol{\xi},e,h) = \mathcal{N}(G_\alpha(\boldsymbol{\xi},e,h),\sigma^2 I)$. The marginal distribution of $\mathbf{u}$ conditioned on (e,h) is given by $q_\alpha(\mathbf{u}|e,h) = \int q_\alpha(\mathbf{u},\boldsymbol{\xi}|e,h)d\boldsymbol{\xi}$. The posterior distribution is $q_\alpha(\boldsymbol{\xi}|\mathbf{u},e,h) = q_\alpha(\mathbf{u},\boldsymbol{\xi}|e,h)/q_\alpha(\mathbf{u}|e,h)$. Suppose we observe expert demonstrations $D = \{(\mathbf{u}_i,\mathbf{x}_i,e_i,h_i), i = 1,...,n\}$. The maximum likelihood estimation of α seeks to maximize the log-likelihood function

$$L(\alpha) = \sum_{i=1}^{n} \log q_\alpha(\mathbf{u_i}|e_i,h_i). \tag{13}$$

The learning gradient can be computed according to

$$\frac{\partial}{\partial\alpha}\log q_\alpha(\mathbf{u}|e,h) = \frac{1}{q_\alpha(\mathbf{u}|e,h)}\frac{\partial}{\partial\alpha}\int q_\alpha(\mathbf{u},\boldsymbol{\xi}|e,h)d\boldsymbol{\xi} \tag{14}$$

$$= \int \left[\frac{\partial}{\partial\alpha}\log q_\alpha(\mathbf{u},\boldsymbol{\xi}|e,h)\right]\frac{q_\alpha(\mathbf{u},\boldsymbol{\xi}|e,h)}{q_\alpha(\mathbf{u}|e,h)}d\boldsymbol{\xi} \tag{15}$$

$$= \mathrm{E}_{q_\alpha(\boldsymbol{\xi}|\mathbf{u},e,h)}\left[\frac{\partial}{\partial\alpha}\log q_\alpha(\mathbf{u},\boldsymbol{\xi}|e,h)\right] \tag{16}$$

The expectation term in Equation (16) is under the posterior distribution $q_\alpha(\boldsymbol{\xi}|\mathbf{u},e,h)$ that is analytically intractable. One may draw samples from $q_\alpha(\boldsymbol{\xi}|\mathbf{u},e,h)$ via Langevin inference dynamics that iterates

$$\boldsymbol{\xi}_{s+1} = \boldsymbol{\xi}_s + \frac{\delta^2}{2}\frac{\partial}{\partial\boldsymbol{\xi}}\log q_\alpha(\boldsymbol{\xi}_s|\mathbf{u},e,h) + \delta\mathbf{z}_s, \tag{17}$$

where $\mathbf{z} \sim \mathcal{N}(0,I)$, δ is the step size and s indexes the time step. Also, ξ_0 is usually sampled from Gaussian white noise for initialization. After we infer $\boldsymbol{\xi}$ from each observation $(\mathbf{u}_i,e_i,h_i)$ by sampling from $q_\alpha(\boldsymbol{\xi}|\mathbf{u}_i,e_i,h_i)$ via Langevin inference process, the Monte Carlo approximation of the gradient of $L(\alpha)$ in Equation (13) is computed by

$$\frac{\partial}{\partial\alpha}L(\alpha) \approx \sum_{i=1}^{n}\left[\frac{\partial}{\partial\alpha}\log q_\alpha(\mathbf{u}_i,\boldsymbol{\xi}_i|e_i,h_i)\right] \tag{18}$$

Since $\frac{\partial}{\partial\boldsymbol{\xi}}\log q_\alpha(\boldsymbol{\xi}|\mathbf{u},e,h) = \frac{\partial}{\partial\boldsymbol{\xi}}\log q_\alpha(\mathbf{u},\boldsymbol{\xi}|e,h)$ in Equation (17). Both inference step in Equation (17) and learning step in Equation (18) need to compute derivative of $\log q_\alpha(\mathbf{u},\boldsymbol{\xi}|e,h) = \frac{1}{2\sigma^2}||G_\alpha(\boldsymbol{\xi},e,h) - \mathbf{u}||^2 + \text{const}$. The former is with respect to $\boldsymbol{\xi}$, while the latter is with respect to α, both of which can be computed by back-propagation through time. The resulting algorithm is called alternating back-propagation through time (ABPTT) algorithm [47].

Although the ABPTT algorithm is natural and simple, the difficulty of training the trajectory generator in this way might lie in the non-convergence issue of the short-run Langevin inference in Equation (17). Even long-run Langevin inference chains are easy to get trapped by local modes. Without fair samples drawn from the posterior distribution, the estimation of α will be biased.

E. Understanding the learning behavior of the cooperative training

In this section, we will present a theoretical understand of the learning behavior of the proposed Algorithm 2 shown in main text. We firstly start from the Contrastive Divergence (CD) algorithm that was proposed for efficient training of energy-based models. The CD runs k steps of MCMC initialized from the training examples, instead of the Gaussian white noise. Given the energy-based model for IOC $p_\theta(\mathbf{u}|e,h)$. Let M_θ be the transition kernel of the finite-step MCMC that samples from $p_\theta(\mathbf{u}|e,h)$. The original CD learning of $p_\theta(\mathbf{u}|e,h)$ seeks to minimize

$$\theta_{\tau+1} = \arg\min_\theta [\mathrm{KL}(p_{\text{expert}}(\mathbf{u}|e,h)\|p_\theta(\mathbf{u}|e,h)) - \mathrm{KL}(M_{\theta_\tau} p_{\text{expert}}(\mathbf{u}|e,h)\|p_\theta(\mathbf{u}|e,h))], \quad (19)$$

where $p_{\text{expert}}(\mathbf{u}|e,h)$ is the unknown distribution of the observed demonstrations of experts. Let $M_\theta p_{\text{expert}}(\mathbf{u}|e,h)$ denote the marginal distribution obtained after running M_θ starting from $p_{\text{expert}}(\mathbf{u}|e,h)$. If $M_\theta p_{\text{expert}}(\mathbf{u}|e,h)$ converges to $p_\theta(\mathbf{u}|e,h)$, then the second KL-divergence will become very small, and the CD estimate eventually is close to maximum likelihood estimate which minimizes the first KL-divergence in Equation (19).

In Algorithm 3, the MCMC sampling of the energy-based model is initialized from the trajectory generator $q_\alpha(\mathbf{u}|e,h)$, thus the learning of the energy-based model follows a modified CD estimate which, at learning step τ, seeks to minimize

$$\theta_{\tau+1} = \arg\min_\theta [\mathrm{KL}(p_{\text{expert}}(\mathbf{u}|e,h)\|p_\theta(\mathbf{u}|e,h)) - \mathrm{KL}(M_{\theta_\tau} q_\alpha(\mathbf{u}|e,h)\|p_\theta(\mathbf{u}|e,h))], \quad (20)$$

where we replace the $p_{\text{expert}}(\mathbf{u}|e,h)$ in Equation (20) by $q_\alpha(\mathbf{u}|e,h)$. That means we run a finite-step MCMC from a given initial distribution $q_\alpha(\mathbf{u}|e,h)$, and use the resulting samples as synthesized examples to approximate the gradient of the log-likelihood of the EBM.

At learning step τ, the learning of $q_\alpha(\mathbf{u}|e,h)$ seeks to minimize

$$\alpha_{\tau+1} = \arg\min_\alpha [\mathrm{KL}(M_\theta q_{\alpha_\tau}(\mathbf{u}|e,h)\|q_\alpha(\mathbf{u}|e,h))]. \quad (21)$$

Equation (21) shows that q_α learns to be the stationary distribution of M_θ. In other words, q_α seeks to be close to p_α, i.e., $q_\alpha \to p_\theta$. If so, the second KL-divergence term in Equation (20) will become zero. The Equation (20) is reduced to minimize the KL-divergence between the observed data distribution p_{expert} and the energy-based model p_θ. Eventually, q_α chases p_θ toward p_{expert}.

IV. Experiments

We evaluate the proposed energy-based continuous inverse optimal control methods on autonomous driving tasks. The code, dataset, more results and experiment details can be found in the project page: http://www.stat.ucla.edu/~yifeixu/ebm-ioc.

A. Experimental setup

In the task of autonomous driving, the state x_t consists of the coordinate, heading angle and velocity of the car, the control 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 optimal control problem.

We assume the dynamic function of all vehicles is a non-linear bicycle model [48], which considers longitudinal, lateral and yaw motions and assumes negligible lateral weight shift, roll and compliance steer while traveling on a smooth road. We assume all vehicles are standard two-axle, four-tire passenger cars with a 3-meter wheelbase. We set an understeering shift to be 0.043 when calculating heading angles.

As to learning, the model parameters are randomly initialized by a normal distribution. The control variables are initialized by zeros, which means keeping straight. We normalize the control variables, i.e., the steering and acceleration, because their scales are different. Instead of sampling the control variables, 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 [49].

We use Root Mean Square Error (RMSE) in meters with respect to each timestep t to measure the accuracy of prediction, i.e., $\mathrm{RMSE}(t) = \sqrt{\frac{1}{n}\sum_{i=1}^n \|\hat{y}_{it} - y_{it}\|^2}$, 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 timestep t. A small RMSE is desired. As a stochastic method, our method draws 5 samples from the learned model for prediction and the model performance is evaluated by the average RMSE and the minimum RMSE over 5 sampled trajectories.

B. Dataset

We test our methods on two datasets. The Massachusetts driving dataset focuses on highways with curved lanes and static scenes, while the NGSIM US-101 dataset focuses on rich vehicle interactions. We randomly split each dataset into training and testing sets. The introductions of these two datasets are given below.

(1) Massachusetts driving dataset: This is a private dataset collected through a vehicle during repeated trips on a stretch

of highway. The dataset includes vehicle states and controls, which are collected by the hardwares on the vehicles, as well as environment information. This dataset has a realistic driving scenario, including curved lanes and complex static scenes. To solve the problem of noisy GPS signal, Kalman filtering is used to denoise the data. There are 44,000 trajectories, each of which contains 40 0.1-second timesteps and is 4 seconds long.

(2) NGSIM US-101: NGSIM [50] contains real highway traffics captured at 10Hz over a time span of 45 minutes. Compared to Massachusetts driving dataset, NGSIM has rich vehicle interactions. The control needs to consider other nearby vehicles. We preprocess the data by dividing the data into 5-second/50-timestep trajectories. The first 10 timesteps are for history and the remaining 40 timesteps are used for prediction. There are 831 scenes with 96,512 5-second 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 [48] dynamics, we perform an inverse dynamics optimization using gradient descent to infer controls. In addition to minimizing the reconstruction error on states, we also minimize the L2 norm of the control variables and the difference between every two consecutive controls. The overall RMSE between the reconstructed positions and the ground truth GPS positions is 0.97 meters. The preprocessed trajectories are assumed to have perfect dynamics with noiseless and smooth sequences of controls and GPS coordinates.

C. Network structure

We first use a linear combination of some hand-designed features as the cost function. The features include: the distance from the current vehicle to the goal point (a virtual point set at front of the vehicle) in terms of longitude and latitude, the distance to the center of the lane, the difference between the current speed and the speed limit, the difference between the vehicle direction and the lane direction, the L2 norm of the control values (including acceleration and steering), the difference between the current control value and the control value at the previous timestep (including acceleration and steering), and the distance from the vehicle to the nearest obstacle. Feature normalization is adopted to make sure that each feature has the same scale of magnitude. These features are also used to design the cost function networks of our methods, as well as baseline methods for fair comparison.

Tables I and II present the multilayer perceptron (MLP) structure and the convolutional neural network (CNN) structure of cost functions, respectively, that we use in Section IV-G. As to the MLP structure, the number of hidden layers N_{hidden} is 64 and the number of layers is 3 (i.e., 2 hidden layers and 1 output layer) by default. The MLP cost function in Table I is defined on a single frame and the cost function of the whole trajectory is the summation of costs over all frames. The CNN cost function presented in Table II is defined on a trajectory with 40 time frames. Table III shows the structure of the generator model used in the joint training framework. It is similar to the actor network of the PPO policy [51] in the generative adversarial imitation learning (GAIL) [27].

TABLE I: Network structure of the MLP cost function

Layer	Output Size
concat($[x, u, e]$)	6 + 2 + 29
hand-designed features	10
Linear, LeakyReLU	N_{hidden}
Linear, LeakyReLU	N_{hidden}
Linear	1

TABLE II: Network structure of the CNN cost function

Layer	Output Size	Stride
♯ of frames × concat($[x, u, e]$)	1 × 40 × (6 + 2 + 29)	–
hand-designed features	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	–

TABLE III: Network structure of the generator model

Layer	Output Size
concat($[x, e, \xi]$)	6 + 29 + 4
Linear, ReLU	64
Linear, ReLU	16
Linear, ReLU	8
Linear, Tanh	2

D. Training details

Normalization. We apply normalization to the controls (i.e., acceleration and steering) and hand-designed features. For the controls, we normalize their values to have zero mean and unit variance. We also normalize each hand-designed feature by dividing by the mean. We normalize two datasets separately.

Optimizer. We use the Adam optimizer on training our models. Both β_1 and β_2 are set to be 0.5. All model parameters are randomly initialized by the He initialization method [52], which is a uniform distribution. In the linear setting, we set learning rate of the Adam to be 0.1 with an exponential decay rate 0.999. For the MLP cost function setting, we set the learning rate to be 0.005 without an exponential decay. In the CNN cost function setting, we set the learning rate to be 0.005 with an exponential decay rate 0.999. For the training of the generator model, the learning rate is 0.002 and the exponential decay rate is 0.998. For each epoch, we shuffle the whole dataset. The batch size is 1,024.

Langevin Dynamics. To prevent gradient values from being too large in each Langevin step, we set the maximum limit to be 0.1. The Langevin step size is set to be 0.1 and the number of Langevin steps is 64. All settings are the same for the gradient descent method to synthesize the controls.

iLQR As to the iLQR solver, we perform a grid search for the learning rate from 0.001 to 1. The maximum step is 100. If the difference between the current step and the previous step is smaller than 0.001, early stop is triggered. In the experiment, the average number of iLQR steps is around 30.

E. Single-agent control

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

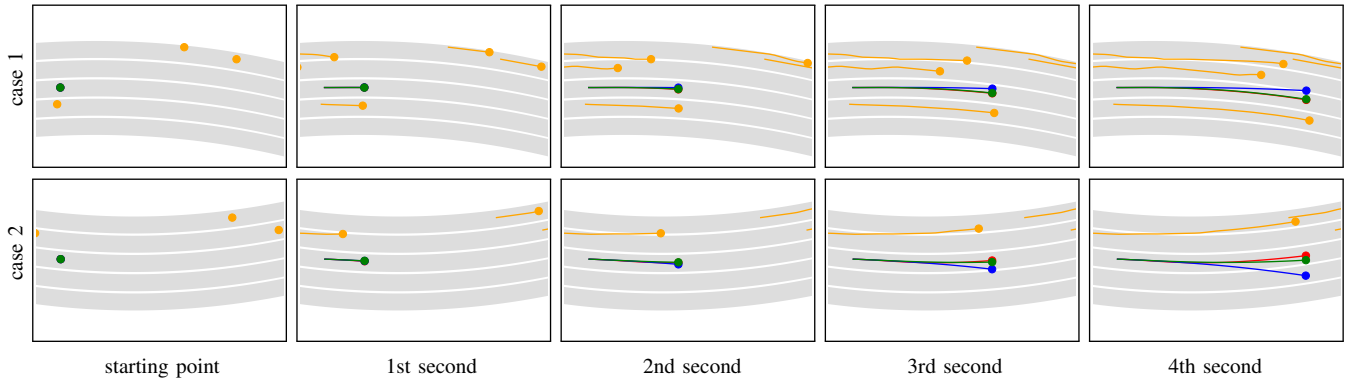


Fig. 2: Predicted trajectories for single-agent control on the Massachusetts driving dataset. The staring point is the last frame of the history trajectory. (Green: predicted trajectories by our model. Blue: predicted trajectories by GAIL. Red: ground truth trajectories. Orange: trajectories of other vehicles. Gray: lanes.)

- Constant velocity: the simplest baseline with a constant velocity and a zero steering.
- Generative adversarial imitation learning (GAIL) [27]: The original GAIL method was proposed for imitation learning. We use the same setting as in [53], which applies the GAIL to the task of modeling human highway driving behavior. Besides, we change the policy gradient method from Trust Region Policy Optimization (TRPO) [54] to Proximal Policy Optimization (PPO) [51].
- IOC with Laplace [30] (IOC-Laplace): We implement this baseline with the same iLQR method as that in our model.

It takes roughly 0.1 seconds to predict a full trajectory with a 64-step Langevin dynamics (or gradient descent). Figure 2 displays two qualitative results. Each row shows one 4-frame example with a frame interval equal to 1 second. Each frame shows trajectories over time for different vehicles as well as different baseline methods for comparison. Table IV and V show quantitative results for Massachusetts driving dataset and NGSIM, respectively. In the last two rows, we provide both average RMSE and minimum RMSE for our sampling-based approach. Our methods achieve substantial improvements compared to baseline methods, such as IOC-Laplace [30] 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 IV: Massachusetts driving dataset results (RMSE).

Method	1s	2s	3s
Constant Velocity	0.340	0.544	1.023
IOC-Laplace	0.386	0.617	0.987
GAIL	0.368	0.626	0.977
ours (via iLQR)	0.307	0.491	0.786
ours (via GD)	0.257	0.413	0.660
ours AVG (via Langevin)	**0.255**	**0.401**	**0.637**
ours MIN (via Langevin)	*0.157*	*0.354*	*0.607*

The reason why the method "IOC-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, they can provide more accurate prediction results.

TABLE V: NGSIM dataset results (RMSE).

Method	1s	2s	3s	4s
Constant Velocity	0.569	1.623	3.075	4.919
IOC-Laplace	0.503	1.468	2.801	4.530
GAIL	0.367	0.738	1.275	2.360
ours (via iLQR)	0.351	0.603	0.969	1.874
ours (via GD)	0.318	0.644	1.149	2.138
ours AVG (via Langevin)	**0.311**	**0.575**	**0.880**	**1.860**
ours MIN (via Langevin)	*0.203*	*0.458*	*0.801*	*1.690*

The problem of GAIL is its model complexity. GAIL parameterizes its discriminator, policy and value function by MLPs. Designing optimal MLP structures of these three components for GAIL is challenging. Our method only needs to design a single architecture for the cost function.

Additionally, the optimal control of our method is performed by simulating trajectories of actions and states according to the learned cost function that takes into account the future information. In contrast, the 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.

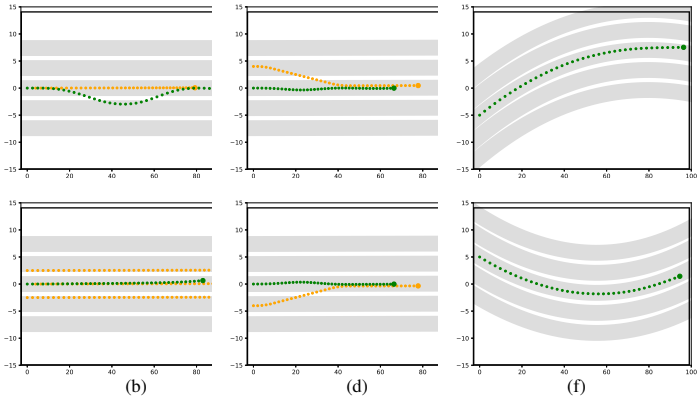
F. Corner case testing with toy examples

Corner cases are important for model evaluation. We construct 6 typical corner cases to test our model. Figure 3 shows the predicted trajectories by our method for several cases. Figures 3(a) and 3(b) show two cases of the 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 3(c) and 3(d) show two cases in the 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 3(e) and 3(f) show two cases in the large lane curvature situation, where our model can still perform well to predict reasonable trajectories.

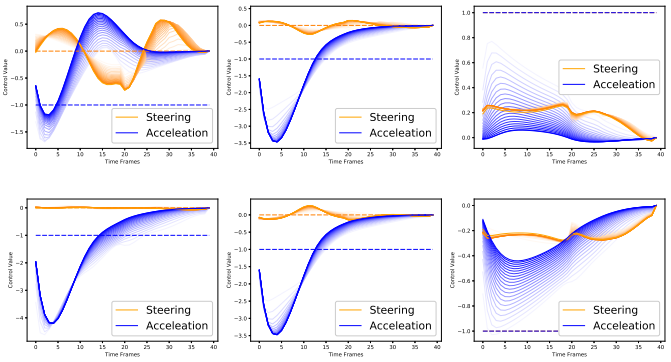
Figure 4 shows the corresponding plots of the predicted controls, i.e., steering and acceleration, over time steps. In each plot, blue lines stand for acceleration and orange lines stand for steering. The dash lines represent the initialization of the controls for Langevin sampling, which are actually the controls at the last time steps of the history trajectories. We use 64 Langevin steps to sample the controls from the learned cost function. We plot the predicted controls (i.e., acceleration and steering) 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 trajectories of controls.

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



(b) Sudden braking (d) Lane cut-in (f) Large lane curvature

Fig. 3: Prediction in corner cases. (Green : predicted trajectories. Orange : trajectories of other vehicles. Gray: lanes.)



(b) Sudden braking (d) Lane cut-in (f) Large lane curvature

Fig. 4: Predicted controls over 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.)

G. 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 cost function. 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, and θ contains all trainable parameters in f. 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 sequence of vectors 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 VI shows a comparison of performances 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.

TABLE VI: A comparison of performances of models with different cost functions (average RMSE).

Method	1s	2s	3s
Linear cost function	0.255	0.401	0.637
MLP cost function	0.237	0.379	0.607
CNN cost function	**0.234**	**0.372**	**0.572**

H. Multi-agent control

In the setting of single-agent control, the future trajectories of other vehicles are assumed to be given and they remain unchanged no matter how the ego vehicle moves. We extend our energy-based framework to the multi-agent setting, in which 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

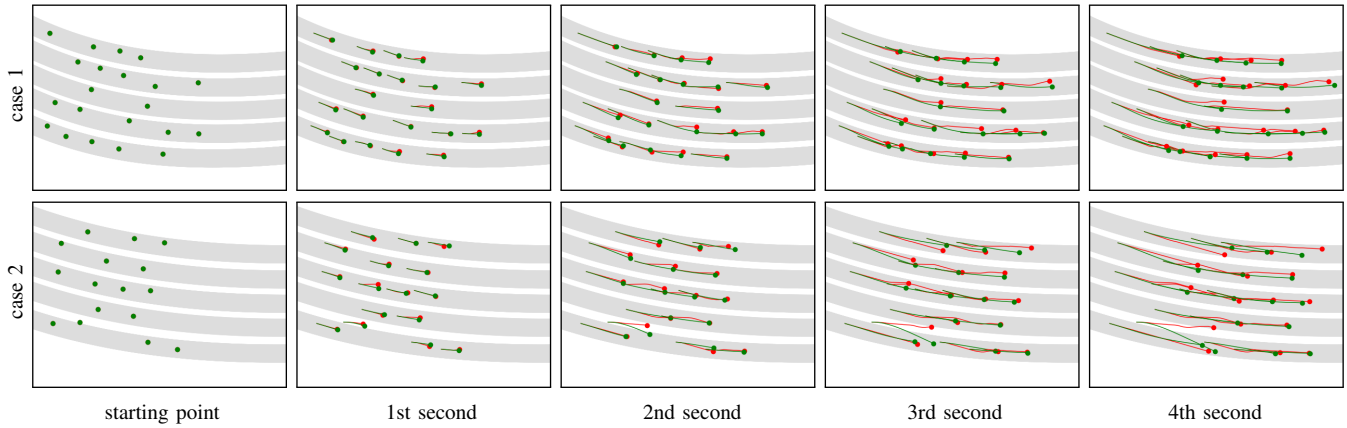


Fig. 5: Predicted trajectories for multi-agent control on the NGSIM dataset. The starting point is the last frame of the history trajectory. (Green: predicted trajectories by our model. Red: ground truth trajectories. Gray: lanes.)

$C_\theta(\mathbf{X}, \mathbf{U}, e, h) = \sum_{k=1}^{K} C_\theta(\mathbf{x}^k, \mathbf{u}^k, e, h^k)$. 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 a zero steering.
- The parameter sharing GAIL (PS-GAIL) [55] [56]: It extends the single-agent GAIL and the Parameter Sharing Trust Region Policy Optimization (PS-TRPO) [57] to enable imitation learning in the multi-agent control context.

We test our method on the NGSIM dataset. We use a linear cost function setting for each agent in this experiment. The maximum number of agents is 64. Figure 5 shows two examples of the qualitative results. Each row is one example. The rows from first to fifth show the positions of all vehicles in the scene as dots at different timesteps respectively, along with the predicted trajectories (the green lines) and the ground truths (the red lines). Table VII shows a comparison of performances between our method and the baselines in terms of RMSE. Results show that our method can also work very well in the multi-agent control scenario.

TABLE VII: Performance comparison in multi-agent control on the NGSIM dataset. Average RMSEs are reported.

Method	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**

I. Joint training with trajectory generator

In this section, we follow Algorithm 3 to introduce a trajectory generator as a fast initializer for our Langevin sampler. In the experiment, we design F_α as a 4-layer MLP with output dimensions 64, 16, 8 and 2, respectively, at different layers. The activation function is ReLU for each hidden layer and Tanh for the final layer. The learning rate of the trajectory generator is set to be 0.005. We update the generator 5 times for each cooperative learning iteration. The rest of the setting remains the same as in the model with a linear cost function.

Table VIII compares the proposed joint training method with the following baselines in terms of average RMSE. The methods include (1) "EBM w/o a generator": the single EBM method without using a trajectory generator. (2) "generator in joint training": the trajectory generator trained with an EBM via the proposed cooperative training algorithm. We train both baseline methods (1) and (2) as well as our joint training framework (which we refer to as "EBM with a generator" in Table VIII with different numbers of Langevin steps. Besides, we implement method (3), which is a single trajectory generator trained via maximum likelihood estimation with MCMC-based inference [47].

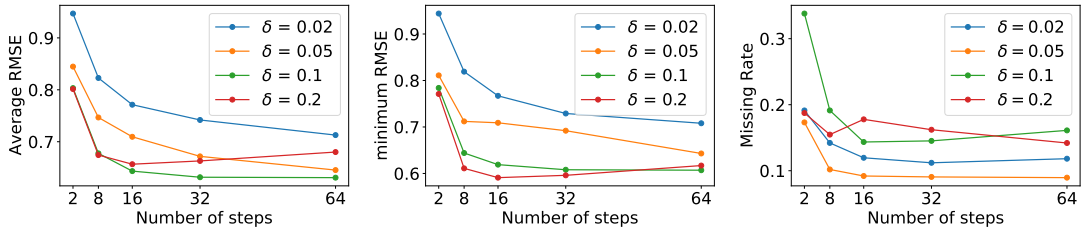
TABLE VIII: Results of the joint training with trajectory generator on the Massachusetts driving dataset. (average RMSE)

Number of steps	2	8	16	32	64
EBM w/o a generator	0.845	0.746	0.709	0.672	0.636
generator in joint training	0.956	0.835	0.844	0.845	0.854
EBM with a generator	**0.804**	**0.672**	**0.649**	**0.638**	**0.633**
generator only	0.911				

This comparison results show that a fast initializer can improve the performance even with less Langevin steps. For example, an EBM using 8 Langevin steps with a fast initializer is comparable with the one with a 32-step Langevin dynamics. Also, the method of "generator in joint training" performs better than the "generator only" setting because of the guidance of Langevin sampling of the EBM.

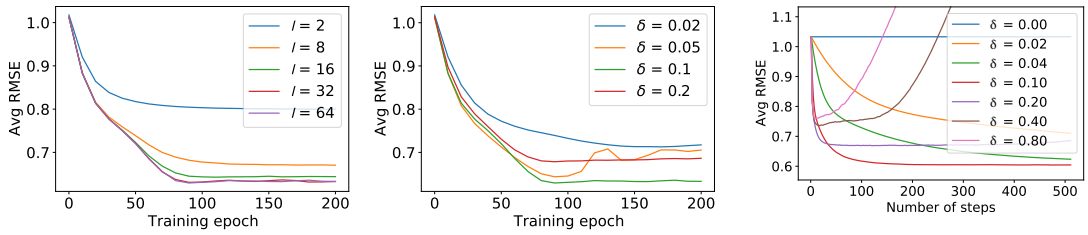
J. Training time and model size

We make a comparison of different methods in terms of computational cost and model size in the task of single-agent control on the Massachusetts driving dataset. We use a mini-batch of size 1,024 during training. For GAIL, the mini-batch size is 64. The total number of epochs is 40. Table IX lists



(a) (b) (c)

Fig. 6: Influence of hyperparameters. Performance comparison of energy-based models with different numbers of Langevin steps and Langevin step sizes is shown in each sub-figure. Each curve represents a model with a certain Langevin step size δ. We set δ=0.02, 0.05, 0.1 and 0.2. For each setting of δ, we choose different numbers of steps l=2, 8,16, 32 and 64. Performances are measured by (a) average RMSE, (b) minimum RMSE, and (c) missing rate on the Massachusetts driving dataset.



(a) (b) (c)

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

the time consumption per training epoch and the number of model parameters for different settings on the Massachusetts driving dataset. The training time is recorded in a PC with a CPU i9-9900 and a GPU Tesla P100. As to the energy-based framework, a simple cost function design can lead to less computation time and less parameters. However, complex cost function can result in better performance in terms of RMSE. Overall, compared with the GAIL and IOC-Laplace baselines, our energy-based IOC methods are competitive.

TABLE IX: Comparison of computation cost and model size.

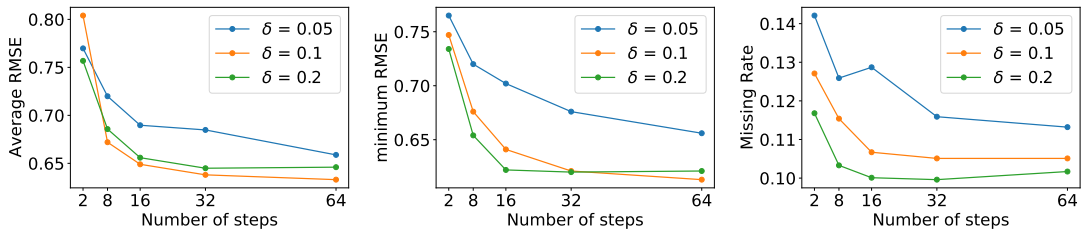
Method		time per epoch	♯ of parameters
EBM-IOC	Linear	~ 3 mins	11
	MLP	~ 10 mins	2817
	CNN	~ 12 mins	44017
	Joint Training	~ 3 mins	3790
GAIL		~ 10 mins	5893
IOC-Laplace		~ 1 min	11

K. Hyperparameter analysis for energy-based IOC models

1) Influence of the number of Langevin steps and the Langevin step size in the single EBM framework: We firstly study the influence of different choices of some hyperparameters, such as the number of Langevin steps l, and the step size δ of each Langevin step. Figure 6 depicts the performances of energy-based IOC models with different δ and l on the Massachusetts driving dataset. Each curve is associated with a certain step size δ and shows the testing performances over different numbers of Langevin steps. The performances are measured by (a) average RMSE, (b) minimum RMSE and (c) missing rate. In our experiments, we draw 5 samples from the learned model for prediction. Missing rate is the ratio of scenarios where none of all 5 sampled trajectories has an endpoint L2 error less than 1.0 meters. The three metrics are used in the sub-figures of Figure 6, respectively. In general, with the same δ, the model performance increases as the number of Langevin steps increases. However, the performance gains become smaller and smaller while using more Langevin steps. Using more Langevin steps will also increase the computational time of sampling. We use $l = 64$ to make a trade-off between performance and computational efficiency. We also choose $\delta = 0.1$ for a trade-off among performances measured by different metrics.

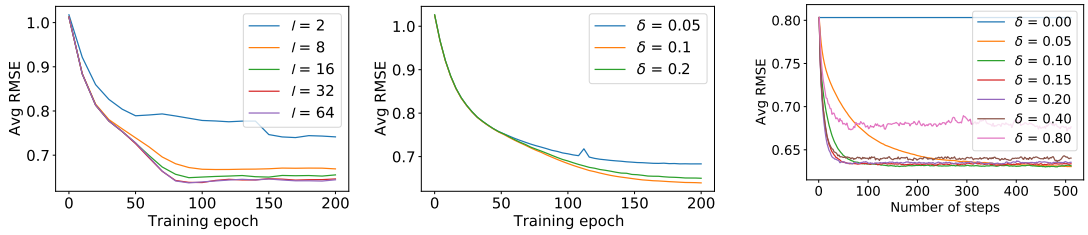
Figures 7(a) and 7(b) depict training curves of the models with different l and δ, respectively. The models are trained on the Massachusetts driving dataset. Each curve reports the testing average RMSEs over training epochs. For testing, we use the same l and δ as those in training. We observe that the learning is quite stable in the sense that the testing errors drop smoothly with an increasing number of training epochs.

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(c) 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



(a) (b) (c)

Fig. 8: Influence of hyperparameters for the cooperative training framework. Performance comparison of frameworks using different numbers of Langevin steps and different Langevin step sizes is given in each sub-figure. Each curve represents a framework with a certain Langevin step size δ. We set δ=0.05, 0.1 and 0.2. For each setting of δ, different numbers of Langevin steps are chosen, l=2, 8,16, 32 and 64. Performances are measured by three different metrics, which are (a) average RMSE, (b) minimum RMSE, and (c) missing rate.



(a) (b) (c)

Fig. 9: Influence of hyperparameters for the cooperative training framework. (a) Testing performances during the cooperative training with different Langevin steps l. (b) Testing performances during the cooperative training with different Langevin step sizes δ. (c) Testing performances after training. Each curve shows testing performances over different numbers of Langevin steps used to sample trajectories in testing. Each curve is associated with a step size chosen in testing. The model is trained with the Langevin step size $\delta = 0.2$ and the number of steps $l = 32$.

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.

2) Influence of the number of Langevin steps and the Langevin step size in the cooperative training framework: We hence study the influence of different choices of the number of Langevin steps l and the Langevin step size δ in the cooperative training framework. Figure 8 depicts the performances of cooperative training frameworks with different δ and l on the Massachusetts driving dataset. The performances shown in Figures 8(a), 8(b), and 8(c) are measured by average RMSE, minimum RMSE and missing rate, respectively. Each curve corresponds to a framework with a certain step size δ and shows the testing performances over different numbers of Langevin steps. Fixing δ, the model performance increases as the number of Langevin steps increases. We use $l = 32$ and $\delta = 0.1$ to make a trade-off between performance and computational efficiency.

Figures 9(a) and 9(b) shows the learning curves of the cooperative training with different numbers of Langevin steps and different step sizes, respectively. Each learning curve shows testing average RMSE over different of training epochs. We observe that the testing average RMSE decreases smoothly as the number of training epochs increases. We also study how different l and δ chosen in testing affect the performance of a learned energy-based IOC model. We first train an energy-based model with $\delta = 0.2$ and $l = 32$, and use the learned model in testing with varying Langevin step size δ and number of Langevin steps l. Figure 9(c) depicts the influences of varying δ and l in testing. We observe that given a learned cost function, Langevin sampling with a smaller step size and a larger number of Langevin steps may allow the model to generate better trajectories.

V. 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 the energy-based model, and propose a sampling-based method and optimization-based modification to learn the cost function. Unlike the previous method for continuous inverse optimal control [30], we learn the model by maximum likelihood using Langevin sampling, without resorting to Laplace approximation. This is a possible reason for improvement over the previous method. Langevin sampling in general also has the potential to avoid sub-optimal modes. Moreover, we propose to train the energy-based model with a trajectory generator as a fast initializer to improve the learning efficiency. The experiments show that our method is generally applicable, and can learn non-linear and non-Markovian cost functions.

In our future work, we shall explore other MCMC sampling or optimal control algorithms. We shall also experiment with recruiting a flow-based model [58], [59], [41] as a learned

approximate sampler to amortize the MCMC sampling. We shall also adapt our model to the scenario where the human drivers may be sub-optimal, and some human judges may assign scores to the trajectories of some of the drivers.

ACKNOWLEDGMENT

The work is supported by NSF DMS-2015577, DARPA SIMPLEX N66001-15-C-4035, ONR MURI N00014-16-1-2007, DARPA ARO W911NF-16-1-0579, DARPA N66001-17-2-4029, and XSEDE grant CIS210052.

REFERENCES

[1] E. Todorov, "Optimal control theory," *Bayesian brain: probabilistic approaches to neural coding*, pp. 269–298, 2006.

[2] W. Li and E. 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)*, 2004, pp. 222–229.

[3] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3–20, 2002.

[4] J. Xie, Y. Lu, S.-C. Zhu, and Y. Wu, "A theory of generative convnet," in *International Conference on Machine Learning (ICML)*, 2016, pp. 2635–2644.

[5] R. M. Neal *et al.*, "Mcmc using hamiltonian dynamics," *Handbook of markov chain monte carlo*, vol. 2, no. 11, p. 2, 2011.

[6] P. J. Bickel and K. A. Doksum, *Mathematical statistics: basic ideas and selected topics, volumes I-II package*. CRC Press, 2015.

[7] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Computation*, vol. 14, no. 8, pp. 1771–1800, 2002.

[8] A. Hyvärinen, "Estimation of non-normalized statistical models by score matching," *Journal of Machine Learning Research*, vol. 6, pp. 695–709, 2005.

[9] J. Xie, Y. Lu, R. Gao, S. Zhu, and Y. Wu, "Cooperative learning of energy-based model and latent variable model via mcmc teaching," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 32, no. 1, 2018.

[10] J. Xie, Y. Lu, R. Gao, S.-C. Zhu, and Y. N. Wu, "Cooperative training of descriptor and generator networks," *IEEE transactions on pattern analysis and machine intelligence (TPAMI)*, vol. 42, no. 1, pp. 27–45, 2018.

[11] J. Xie, Z. Zheng, X. Fang, S.-C. Zhu, and Y. N. Wu, "Cooperative training of fast thinking initializer and slow thinking solver for conditional learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2021.

[12] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning." in *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI)*, vol. 8. Chicago, IL, USA, 2008, pp. 1433–1438.

[13] S. C. Zhu, Y. Wu, and D. Mumford, "Filters, random fields and maximum entropy (frame): Towards a unified theory for texture modeling," *International Journal of Computer Vision (IJCV)*, vol. 27, no. 2, pp. 107–126, 1998.

[14] M. Richardson and P. Domingos, "Markov logic networks," *Machine learning*, vol. 62, no. 1-2, pp. 107–136, 2006.

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

[16] J. Xie, S.-C. Zhu, and Y. N. Wu, "Learning energy-based spatial-temporal generative convnets for dynamic patterns," *IEEE transactions on pattern analysis and machine intelligence (TPAMI)*, 2019.

[17] J. Xie, W. Hu, S.-C. Zhu, and Y. N. Wu, "Learning sparse frame models for natural image patterns," *International Journal of Computer Vision (IJCV)*, vol. 114, no. 2-3, pp. 91–112, 2015.

[18] J. Xie, S.-C. Zhu, and Y. N. Wu, "Synthesizing dynamic patterns by spatial-temporal generative convnet," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 7093–7101.

[19] J. Xie, Z. Zheng, R. Gao, W. Wang, S.-C. Zhu, and Y. N. Wu, "Generative voxelnet: Learning energy-based models for 3d shape synthesis and analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2020.

[20] J. Xie, Z. Zheng, R. Gao, W. Wang, S. Zhu, and Y. N. Wu, "Learning descriptor networks for 3d shape synthesis and analysis," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 8629–8638.

[21] J. Xie, Y. Xu, Z. Zheng, S. Zhu, and Y. N. Wu, "Generative pointnet: Deep energy-based learning on unordered point sets for 3d generation, reconstruction and classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 14 976–14 985.

[22] Y. LeCun, Y. Bengio *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems (NIPS)*, 2012, pp. 1097–1105.

[24] C. Finn, S. Levine, and P. Abbeel, "Guided cost learning: Deep inverse optimal control via policy optimization," in *International Conference on Machine Learning (ICML)*, 2016, pp. 49–58.

[25] C. Finn, P. Christiano, P. Abbeel, and S. Levine, "A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models," *arXiv preprint arXiv:1611.03852*, 2016.

[26] 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 (NIPS)*, 2014, pp. 2672–2680.

[27] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in Neural Information Processing Systems (NIPS)*, 2016, pp. 4565–4573.

[28] Y. Li, J. Song, and S. Ermon, "Infogail: Interpretable imitation learning from visual demonstrations," in *Advances in Neural Information Processing Systems (NIPS)*, 2017, pp. 3812–3822.

[29] M. Monfort, A. Liu, and B. D. Ziebart, "Intent prediction and trajectory forecasting via predictive inverse linear-quadratic regulation," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI)*, 2015, pp. 3672–3678.

[30] S. Levine and V. Koltun, "Continuous inverse optimal control with locally optimal examples," in *International Conference on Machine Learning (ICML)*, 2012, pp. 475–482.

[31] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social lstm: Human trajectory prediction in crowded spaces," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[32] A. Gupta, J. Johnson, L. Fei-Fei, S. Savarese, and A. Alahi, "Social gan: Socially acceptable trajectories with generative adversarial networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[33] A. Vemula, K. Muelling, and J. Oh, "Social attention: Modeling attention in human crowds," in *Proceedings of the International Conference on Robotics and Automation (ICRA) 2018*, May 2018.

[34] N. Lee, W. Choi, P. Vernaza, C. B. Choy, P. H. Torr, and M. Chandraker, "Desire: Distant future prediction in dynamic scenes with interacting agents," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 336–345.

[35] N. Deo, A. Rangesh, and M. M. Trivedi, "How would surround vehicles move? a unified framework for maneuver classification and motion prediction," *IEEE Transactions on Intelligent Vehicles*, vol. 3, no. 2, pp. 129–140, 2018.

[36] T. Zhao, Y. Xu, M. Monfort, W. Choi, C. Baker, Y. Zhao, Y. Wang, and Y. N. Wu, "Multi-agent tensor fusion for contextual trajectory prediction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[37] J. Xie, Z. Zheng, X. Fang, S. Zhu, and Y. N. Wu, "Learning cycle-consistent cooperative networks via alternating MCMC teaching for unsupervised cross-domain translation," in *Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI)*, 2021, pp. 10 430–10 440.

[38] J. Xie, Z. Zheng, and P. Li, "Learning energy-based model with variational auto-encoder as amortized sampler," in *Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI)*, 2021, pp. 10 441–10 451.

[39] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *International Conference on Learning Representations (ICLR)*, 2014.

[40] J. Xie, Y. Zhu, J. Li, and P. Li, "A tale of two flows: Cooperative learning of langevin flow and normalizing flow toward energy-based model," in *International Conference on Learning Representations (ICLR)*, 2022.

[41] D. P. Kingma and P. Dhariwal, "Glow: Generative flow with invertible 1x1 convolutions," *Advances in neural information processing systems (NeurIPS)*, vol. 31, 2018.

[42] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.

[43] T. M. Cover and J. A. Thomas, *Elements of information theory, Second Edition*. Wiley, 2006.

[44] W. K. Hastings, "Monte carlo sampling methods using markov chains and their applications," 1970.

[45] T. Chen, E. B. Fox, and C. Guestrin, "Stochastic gradient hamiltonian monte carlo," in *International Conference on Machine Learning (ICML)*, vol. 32, 2014, pp. 1683–1691.

[46] E. Nijkamp, M. Hill, T. Han, S. Zhu, and Y. N. Wu, "On the anatomy of mcmc-based maximum likelihood learning of energy-based models," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 34, no. 04, 2020, pp. 5272–5280.

[47] J. Xie, R. Gao, Z. Zheng, S.-C. Zhu, and Y. N. Wu, "Learning dynamic generator model by alternating back-propagation through time," in *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, vol. 33, 2019, pp. 5498–5507.

[48] P. Polack, F. Altché, B. d'Andréa Novel, and A. de La Fortelle, "The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?" in *Intelligent Vehicles Symposium (IV)*. IEEE, 2017, pp. 812–818.

[49] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2018.

[50] J. Colyar and J. Halkias, "US highway 101 dataset," vol. Federal Highway Administration (FHWA), Tech. Rep. FHWA-HRT-07-030, 2007.

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

[52] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1026–1034.

[53] A. Kuefler, J. Morton, T. Wheeler, and M. Kochenderfer, "Imitating driver behavior with generative adversarial networks," in *IEEE Intelligent Vehicles Symposium*, 2017, pp. 204–211.

[54] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning (ICML)*, 2015, pp. 1889–1897.

[55] R. P. Bhattacharyya, D. J. Phillips, B. Wulfe, J. Morton, A. Kuefler, and M. J. Kochenderfer, "Multi-agent imitation learning for driving simulation," in *International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 1534–1539.

[56] R. P. Bhattacharyya, D. J. Phillips, C. Liu, J. K. Gupta, K. Driggs-Campbell, and M. 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.

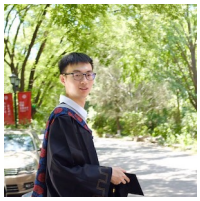
[57] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2017, pp. 66–83.

[58] L. Dinh, D. Krueger, and Y. Bengio, "NICE: non-linear independent components estimation," in *International Conference on Learning Representations (ICLR) Workshop*, 2015.

[59] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real NVP," in *International Conference on Learning Representations (ICLR)*, 2017.

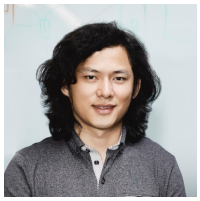
Yifei Xu received his Ph.D. degree in statistics from University of California, Los Angeles (UCLA) in 2022. He received his B.E. degree in computer science at Shanghai Jiao Tong University. His research interests focus on genertive model, reinforcement learning and computer vision.

Jianwen Xie received his Ph.D. degree in statistics from University of California, Los Angeles (UCLA) in 2016. He is currently a senior research scientist at Cognitive Computing Lab, Baidu Research, USA. Before joining Baidu, he was a senior research scientist at Hikvision Research Institute USA from 2017 to 2020, and a staff research associate and postdoctoral researcher in the Center for Vision, Cognition, Learning, and Autonomy (VCLA) at UCLA from 2016 to 2017. His research interests focus on generative modeling and unsupervised learning.



Tianyang Zhao is currently a Ph.D. candidate in the Center for Vision, Cognition, Learning and Autonomy at the University of California, Los Angeles (UCLA). He received his B.E. degree in computer science at Peking University. His research interests lie in generative model and representation learning.

Chris Baker received his Ph.D. degree in cognitive science from Massachusetts Institude of Technology (MIT) in 2012. He is currently a co-founder and chief scientist at iSee Inc.



Yibiao Zhao received his Ph.D. degree in statistics from University of California, Los Angeles (UCLA) in 2015. He is currently a co-founder and CEO at iSee Inc.

Ying Nian Wu received his Ph.D. degree in statistics from Harvard University in 1996. He was an assistant professor in the Department of Statistics, University of Michigan from 1997 to 1999. He joined University of California, Los Angeles (UCLA) in 1999, and is currently a professor in UCLA Department of Statistics. His research interests include generative models, representation learning, and computer vision. He received Honorable Mention for the David Marr Prize with S. C. Zhu et al. in 1999 and 2007 for generative modeling in computer vision.