Energy-Based Continuous Inverse Optimal Control

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

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 (EBM), 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 backpropagation 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 EBM simultaneously with a top-down trajectory generator via cooperative learning, where the trajectory generator is used to fast initialize the synthesis step of the EBM. We demonstrate the proposed methods on autonomous driving tasks and show that they can learn suitable cost functions for optimal control.

Index Terms—Cooperative learning, energy-based models (EBMs), inverse optimal control (IOC), Langevin dynamics.

I. INTRODUCTION

A. Background and Motivation

The problem of continuous optimal control has been extensively studied. In this article, 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.

Manuscript received 10 May 2021; revised 21 December 2021; accepted 25 March 2022. Date of publication 5 May 2022; date of current version 1 December 2023. This work was supported in part by NSF under Grant DMS-2015577, in part by the Defense Advanced Research Projects Agency’s Simplifying Complexity in Scientific Discovery (DARPA’s SIMPLEX) under Grant N66001-15-C-4035, in part by the Office of Naval Research Multidisciplinary University Initiative (ONR MURI) under Grant N00014-16-1-2007, in part by DARPA Army Research Office (ARO) under Grant W911NF-16-1-0579, DARPA under Grant N66001-17-2-4029, and in part by the Extreme Science and Engineering Discovery Environment (XSEDE) under Grant C11S210052. (Corresponding author: Yifei Xu.)

Yifei Xu, Tianyang Zhao, and Ying Nian Wu are with the Department of Statistics, University of California at Los Angeles, Los Angeles, CA 90095 USA (e-mail: fei960922@ucla.edu; tyzhao@ucla.edu; ywu@stat.ucla.edu).

Jianwen Xie is with the Cognitive Computing Laboratory, Baidu Research, Bellevue, WA 98004 USA (e-mail: jianwen@ucla.edu).

Chris Baker and Yibiao Zhao are with iSee, Inc., Cambridge, MA 02139 USA (e-mail: chrisbaker@see.ai; yz@see.ai).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNNLS.2022.3168795.

Digital Object Identifier 10.1109/TNNLS.2022.3168795.

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 iterative linear quadratic regulator (iLQR) 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 IOC in the framework of energy-based model (EBM) [4]. Originated from statistical physics, an 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 IOC, the cost function plays the role of energy function, and the observed expert sequences are assumed to be random samples from the EBM 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 nonlinear 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 EBM. 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 chain Monte Carlo (MCMC), such as Langevin dynamics [5]. The gradient computation in the Langevin dynamics can be conveniently and efficiently carried out by backpropagation 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 neural networks.
B. Related Work

The following are research themes related to our work.

1) Maximum Entropy Framework: Our work follows the maximum entropy framework mentioned in [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, Wulfmeier et al. [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 EBM. In our work, the state space is continuous, where we use Langevin dynamics via backpropagation through time to sample trajectories from the learned model. We also propose an optimization-based method where we use the gradient descent (GD) algorithm or a built-in optimal control algorithm as the inner loop for learning.

2) ConvNet-EBM: Recently, Xie et al. [16]–[21] applied deep EBMs 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, backpropagation 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 [25]–[28], 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 GD or a built-in optimal control algorithm.

4) Continuous IOC: The IOC problem has been studied in [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 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]–[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) mentioned in [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], 3-D 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 autoencoder (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 given as follows.

1) We propose an energy-based method for continuous IOC based on Langevin sampling via backpropagation through time. To the best of our knowledge, this is the first work that studies MCMC sampling-based IOC. Such an “analysis by synthesis” learning scheme makes our work essentially different from [12], [29], and [30].

2) 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.

3) We evaluate the proposed methods on autonomous driving tasks for trajectory prediction. We apply our framework to both single-agent system and multiagent system, with both linear cost function and neural network nonlinear cost function. This is the first work to study vehicle trajectory prediction under the energy-based framework.

4) We also propose to train an EBM together with a policy-like trajectory generator, which serves as a fast initializer for the Langevin sampling, in a cooperative learning scheme.

5) 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 this article is organized as follows. Section II presents the proposed framework of the energy-based IOC. Section III presents the proposed joint training framework, in which the EBM is trained simultaneously with a trajectory generator as amortized sampler. Qualitative and quantitative results of experiments are shown in Section IV. The conclusion of this article is given in Section V.

II. ENERGY-BASED IOC

A. Optimal Control

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

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

B. Energy-Based Probabilistic Model

The EBM assumes the following conditional probability density function:

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

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

C. Sampling-Based IOC

The parameters $\theta$ can be learned by maximum likelihood. The log likelihood is given by

$$L(\theta) = \frac{1}{n} \sum_{i=1}^{n} \log p_\theta(u_i|e_i, h_i).$$  \hspace{1cm} (2)

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

$$L'(\theta) = \frac{1}{n} \sum_{i=1}^{n} [E_{p_\theta(u|e_i, h_i)}(\frac{\partial}{\partial \theta}C_\theta(x, u, e_i, h_i)) $$

$$- \frac{\partial}{\partial \theta}C_\theta(x, u, e_i, h_i)]$$  \hspace{1cm} (3)

which follows the property of the normalizing constant $(\partial / \partial \theta) \log Z_\theta(e, h) = -E_{p_\theta(u|e_i, h_i)}((\partial / \partial \theta)C_\theta(x, u, e_i, h_i))$.

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

$$L'(\theta) = \frac{1}{n} \sum_{i=1}^{n} [\frac{\partial}{\partial \theta}C_\theta(\tilde{x}_i, \tilde{u}_i, e_i, h_i)]$$

$$- \frac{\partial}{\partial \theta}C_\theta(x, u, e_i, h_i)]$$  \hspace{1cm} (4)

which is the stochastic unbiased estimator of $L'(\theta)$. Then, we can run the gradient ascent algorithm $\theta_{t+1} = \theta_t + \gamma \tilde{L}'(\theta_t)$ to obtain the maximum likelihood estimate of $\theta$, where $\tau$ indexes the time step and $\gamma$ is the step size. According to the Robbins–Monroe theory of stochastic approximation [42],
if \( \sum \gamma = \infty \) and \( \sum \gamma^2 < \infty \), the algorithm will converge to a solution of \( L'(\theta) = 0 \). For each \( i \), we can also generate multiple copies of \((\tilde{x}_i, \tilde{u}_i)\) from \( p_\theta(u|e_i, h_i) \) and average them to approximate the expectation in (3). A small number is sufficient because the averaging effect takes place over time.

In the linear case, where \( C_\theta(x, u, e, h) = (\theta, \phi(x, u, e, h)) \), we have \( (\tilde{c}/\tilde{c}u)C_\theta(x, u, e, h) = \phi(x, u, e, h) \), making \( \hat{L}'(\theta) = (1/n) \sum_{i=1}^n \phi(\tilde{x}_i, \tilde{u}_i, e_i, h_i) - \phi(x, u, e, h)) \). 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 \( p_\theta(u|e, h) \) can be accomplished by an efficient gradient-based MCMC, the Langevin dynamics, which iterates the following steps:

\[
\begin{align*}
\mathbf{u}_{s+1} &= \mathbf{u}_s + \frac{\delta^2}{2} \frac{\partial}{\partial \mathbf{u}} \log p_\theta(\mathbf{u}|e, h) + \mathbf{z}_s \\
&= \mathbf{u}_s - \frac{\delta^2}{2} \frac{\partial}{\partial \mathbf{u}} \mathbf{C}_\theta(x_s, \mathbf{u}_s, e, h) + \mathbf{z}_s
\end{align*}
\]  

(5)

where \( s \) indexes the time step, \( \delta \) is the step size, and \( \mathbf{z}_s \sim \mathcal{N}(0, I) \) the Brownian motion independently over \( s \), with \( I \) 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 GD part \( \mathbf{u}_{s+1} = \mathbf{u}_s - (\delta^2/2)(\tilde{c}/\tilde{c}u)C_\theta(x_s, \mathbf{u}_s, e, h) \) of (5) is a mode seeking process that minimizes the cost function \( C_\theta \), while the added Gaussian noise \( \mathbf{z}_s \) prevents 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 \( p_\theta(u|e, h) \) under some regularity conditions. A Metropolis–Hastings [44] step can also be added to correct for the error due to discrete time step in (5), but most existing works, such as [4], [45], and [46], have shown that this can be ignored in practice if a small enough step size \( \delta \) is used. Thus, for computational efficiency, in this work, we do not have the Metropolis–Hastings correction in our implementation.

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

D. Optimization-Based IOC

We can remove the noise term in Langevin dynamics in (5), to make it a GD process, i.e., \( \mathbf{u}_{s+1} = \mathbf{u}_s - \eta(\tilde{c}/\tilde{c}u)C_\theta(x_s, \mathbf{u}_s, e, h) \), 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 \( C_\theta(x, u, e, h) \) over \( u \). 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 \( \tilde{u}_i \sim p_\theta(u|e_i, h_i) \), we optimize

\[
\tilde{u}_i = \arg\min_u C_\theta(x, u, e_i, h_i).
\]

(6)

The analysis step remains unchanged. In this article, we emphasize the sampling-based method, which is more principled maximum likelihood learning, and 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 EBM in (1) is to consider its tempered version \( p_\theta(u|e, h) \propto \exp[-C_\theta(x, u, e, h)/T] \), where \( T \) is the temperature. Then, the optimized \( \tilde{u} \) that minimizes \( C_\theta(x, u, e, h) \) can be considered the zero-temperature sample, which is used to approximate the expectation in (3).

1) Moment Matching: For simplicity, consider the linear cost function \( C_\theta(x, u, e, h) = (\theta, \phi(x, u, e, h)) \). 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

\[
\frac{1}{n} \sum_{i=1}^n \phi(\tilde{x}_i, \tilde{u}_i, e_i, h_i) = \frac{1}{n} \sum_{i=1}^n \phi(x_i, u_i, e_i, h_i)
\]

(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 \( \phi(\tilde{x}, \tilde{u}, e, h) = (1/n) \sum_{i=1}^n \phi(x_i, u_i, e_i, h_i) \), 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 \( i = 1, \ldots, n \), 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.

2) Adversarial Learning: We can also justify this optimization-based algorithm outside the context of probabilistic model as adversarial learning. To this end, we rethink about the IOC, 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

\[
V(\theta, \{\tilde{u}_i\}) = \frac{1}{n} \sum_{i=1}^n [C_\theta(\tilde{x}_i, \tilde{u}_i, e_i, h_i) - C_\theta(x_i, u_i, e_i, h_i)]
\]

(8)

and then, \( \hat{L}'(\theta) = (\tilde{c}/\tilde{c}u)V \) so that the analysis step increases \( V \). The optimization step and the analysis step play an adversarial game \( \max_{\theta} \min_{\tilde{u}, V} V \), 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 observed trajectory in order to increase \( V \).
Algorithm 1 Energy-Based IOC With Synthesis Step
1: input expert demonstrations $D = \{ (x_i, u_i, e_i, h_i), \forall i \}$.
2: output cost function parameters $\theta$, and synthesized trajectories $\{ (\tilde{x}_i, \tilde{u}_i), \forall i \}$.
3: Let $\tau \leftarrow 0$, randomly initialize $\theta$.
4: repeat
5: synthesis step: for each $i$, synthesize $\tilde{u}_i \sim p_{\theta}(u|e_i, h_i)$ by Langevin sampling and then obtain $\tilde{x}_i$.
6: analysis step: update $\theta_{t+1} = \theta_t + \gamma_t L'(\theta_t)$, where $L'$ is computed according to Equation (4).
7: $\tau \leftarrow \tau + 1$.
8: until $\tau = \tau_{\text{max}}$, the number of iterations.

Algorithm 2 Energy-Based IOC With Optimization Step
1: input expert demonstrations $D = \{ (x_i, u_i, e_i, h_i), \forall i \}$.
2: output cost function parameters $\theta$, and optimized trajectories $\{ (\tilde{x}_i, \tilde{u}_i), \forall i \}$.
3: Let $\tau \leftarrow 0$, randomly initialize $\theta$.
4: repeat
5: optimization step: for each $i$, optimize $\tilde{u}_i = \arg \min_u C_0(x, u, e_i, h_i)$ by gradient descent (GD) or iLQR, and then obtain $\tilde{x}_i$.
6: analysis step: update $\theta_{t+1} = \theta_t + \gamma_t L'(\theta_t)$, where $L'$ is computed according to Equation (4).
7: $\tau \leftarrow \tau + 1$.
8: until $\tau = \tau_{\text{max}}$, the number of iterations.

E. Energy-Based IOC Algorithm

Algorithms 1 and 2 present the sampling- 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 GD or iLQR as optimizer.

III. JOINT TRAINING

A. Trajectory Generator Model as a Fast Initializer

Both sampling-based method via Langevin dynamics and optimization-based method via GD are based on an 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 EBM simultaneously with a trajectory generator model that serves as a fast initializer for the Langevin dynamics or GD of the EBM.

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

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

$$u_t = F_u(x_{t-1}, \tilde{e}_t, e)$$

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

where $t = 1, \ldots, T$. (9) is the policy model and (10) is the known dynamic function. $\tilde{e}_t \sim N(0, I)$ is the Gaussian noise vector. The Gaussian noise vectors at different times ($\tilde{e}_t, t = 1, \ldots, T$) 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 $\tilde{e}_t$ accounts for the randomness in the mapping from $x_{t-1}$ to $u_t$. $F_u$ is a multilayer perceptron (MLP), where $\alpha$ is the model parameters of the network. The initial state $x_0$ is assumed to be given.

We denote $\xi = (\tilde{e}_t, t = 1, \ldots, T)$ and $p(\xi) = \prod_{t=1}^{T} p(\tilde{e}_t)$. 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 $\tilde{e}_t$. In fact, we can write the trajectory generator in a compact form, i.e., $u = G_u(\xi, e, h)$, where $G_u$ composes $F_u$ and $f$ over time, and we use $h = x_0$ for simplicity in our implementation.

The algorithm for joint training of the EBM and the trajectory generator is that at each iteration: 1) we first sample $\xi_t$ from the Gaussian prior distribution and then generate the initial trajectories by $\tilde{u}_i = G_u(\xi_t, e_i, h_i)$ for $i = 1, \ldots, n$; 2) starting from the initial trajectories $\{\tilde{u}_i\}$, we sample from the EBM by running a finite number of Langevin steps or optimize the cost function by running a finite step of GD to obtain the updated trajectories $\{\tilde{u}_i\}$ and then obtain $\{\tilde{x}_i\}$; 3) we update the parameters $\theta$ of the EBM by MLE, where the computation of the gradient of the likelihood is based on $\{\tilde{u}_i\}$ and follows (4); and 4) we update the parameters $\alpha$ of the trajectory generator by GD on the loss

$$\tilde{L}_g(\alpha) = \frac{1}{c} \sum_{i=1}^{n} \| \tilde{u}_i - G_u(\xi_i, e_i, h_i) \|^2.$$ (11)

Algorithm 3 presents a detailed description of the cooperative training algorithm of an EBM and a trajectory generator for IOC. Synthesis step and optimization step are two options to generate $\{\tilde{u}_i, \tilde{x}_i\}$.

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 EBM (we also call it the trajectory evaluator) and the latent variable model (i.e., the trajectory generator). Both these two models can be parameterized by deep neural networks and they are in opposite directions. The EBM 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,
Algorithm 3 Energy-Based IOC With a Trajectory Generator

1: input expert demonstrations $D = \{\{x_i, u_i, e_i, h_i\}, \forall i\}$.
2: output cost function parameters $\theta$, trajectory generator parameters $\alpha$, and synthesized or optimized trajectories $\{(\hat{x}_t, \hat{u}_t), \forall i\}$.
3: Let $\tau \leftarrow 0$, randomly initialize $\theta$ and $\alpha$.
4: repeat
5:  Initialization step: Initialize $\hat{u}_i = G_\alpha(x_i, e_i, h_i)$, where $x_i \sim \mathcal{N}(0, I)$ by ancestral sampling, and then obtain $\hat{x}_i$ for each $i$.
6:  Synthesis step or optimization step: Given the initial $\hat{u}_i$, synthesizing $\tilde{u}_i \sim p_\theta(u_i | e_i, h_i)$ by Langevin sampling, or optimizing $\hat{u}_i$ by gradient descent (GD) or iLQR, and then obtain $\hat{x}_i$ for each $i$.
7:  Analysis step (update cost function): Update $\theta_{t+1} = \theta_t + \gamma_t \hat{L}(\theta_t)$, where $\hat{L}(\theta)$ is computed according to Equation (4).
8:  Analysis step (update policy model): Update $\alpha_{t+1} = \alpha_t - \xi_t \hat{l}_\delta(\alpha_t)$, where $\hat{l}_\delta(\alpha)$ is computed according to Equation (11).
9:  $\tau \leftarrow \tau + 1$.
10: until $\tau = \tau_{\text{max}}$, the number of iterations.

as illustrated by the following diagram:

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

C. Iterative and Noniterative Generations of Trajectories

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

Fig. 1(a) shows the generation process of $(u, x)$. Given $(h, e)$, the Langevin sampling seeks to find $u = (u_1, \ldots, u_T)$ to minimize $C_\theta(x, u, e, h)$. The dashed double line arrows indicate iterative generation by sampling in the EBM, while the dashed line arrows indicate the known dynamic function. The whole process of generating $(u, x)$ is of a dynamic or causal nature in which it directly evolves or unfolds over time.

D. Issue in Maximum Likelihood Training of a Single Trajectory Generator

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

$$q_\alpha(u, \xi | e, h) = q_\alpha(u | \xi, e, h) p(\xi)$$

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

The learning gradient can be computed according to

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

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

$$= E_{q_\alpha(u | e, h)} \left[ \frac{\partial}{\partial \alpha} \log q_\alpha(u, \xi | e, h) \right].$$

The expectation term in (16) is under the posterior distribution $q_\alpha(u, \xi | e, h)$ that is analytically intractable. One may draw
samples from \( q_a(\xi|u,e,h) \) via Langevin inference dynamics that iterates
\[
\xi_{t+1} = \xi_t + \frac{\delta^2}{2} \log q_a(\xi_t|u,e,h) + \delta z_t
\]  
(17)
where \( z \sim \mathcal{N}(0, I) \), \( \delta \) is the step size, and \( s \) indexes the time step. Also, \( \xi_0 \) is usually sampled from Gaussian white noise for initialization. After we infer \( \xi \) from each observation \((u,e,h_i)\) by sampling from \( q_a(\xi|u,e,h_i) \) via Langevin inference process, the Monte Carlo approximation of the gradient of \( L(\alpha) \) in (13) is computed by
\[
\frac{\partial}{\partial \alpha} L(\alpha) \approx \sum_{i=1}^{n} \left( \frac{\partial}{\partial \alpha} \log q_a(u, \xi_i | e_i, h_i) \right).
\]  
(18)
Since \( \frac{\partial}{\partial \xi} \log q_a(\xi|u,e,h) = \frac{\partial}{\partial \xi} \log q_a(\xi, \xi|e,h) \) in (17). Both inference step in (17) and the learning step in (18) need to compute derivative of \( \log q_a(u, \xi|e,h) = (1/2\sigma^2)\|G_a(\xi, e, h) - u\|^2 + \text{const} \). The former is with respect to \( \xi \), while the latter is with respect to \( \alpha \), both of which can be computed by backpropagation through time. The resulting algorithm is called alternating backpropagation 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 nonconvergence issue of the short-run Langevin inference in (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 \( \alpha \) 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. We first start from the CD algorithm that was proposed for efficient training of EBM. The CD runs \( k \) steps of MCMC initialized from the training examples, instead of the Gaussian white noise. Given the EBM for IOC \( p_\theta(u|e,h) \). Let \( M_\theta \) be the transition kernel of the finite-step MCMC that samples from \( p_\theta(u|e,h) \). The original CD learning of \( p_\theta(u|e,h) \) seeks to minimize
\[
\theta_{t+1} = \arg \min_\theta \left[ \text{KL}(p_{\text{expert}}(u|e,h)||p_\theta(u|e,h)) - \text{KL}(M_\theta, p_{\text{expert}}(u|e,h)||p_\theta(u|e,h)) \right]
\]  
(19)
where \( p_{\text{expert}}(u|e,h) \) is the unknown distribution of the observed demonstrations of experts. Let \( M_\theta p_{\text{expert}}(u|e,h) \) denote the marginal distribution obtained after running \( M_\theta \) starting from \( p_{\text{expert}}(u|e,h) \). If \( M_\theta p_{\text{expert}}(u|e,h) \) converges to \( p_\theta(u|e,h) \), then the second Kullback-Leibler (KL) divergence will become very small, and the CD estimate eventually is close to maximum likelihood estimate, which minimizes the first KL divergence in (19).

In Algorithm 3, the MCMC sampling of the EBM is initialized from the trajectory generator \( q_a(u|e,h) \), and thus, the learning of the EBM follows a modified CD estimate, which, at learning step \( \tau \), seeks to minimize:
\[
\theta_{\tau+1} = \arg \min_\theta \left[ \text{KL}(p_{\text{expert}}(u|e,h)||p_\theta(u|e,h)) - \text{KL}(M_\theta, q_a(u|e,h)||p_\theta(u|e,h)) \right]
\]  
(20)
where we replace \( p_{\text{expert}}(u|e,h) \) in (20) by \( q_a(u|e,h) \). This means that we run a finite-step MCMC from a given initial distribution \( q_a(u|e,h) \) and use the resulting samples as synthesized examples to approximate the gradient of the log likelihood of the EBM.

At learning step \( \tau \), the learning of \( q_a(u|e,h) \) seeks to minimize
\[
a_{\tau+1} = \arg \min_a \left[ \text{KL}(M_\theta q_a(u|e,h)||q_a(u|e,h)) \right].
\]  
(21)

Equation (21) shows that \( q_a \) learns to be the stationary distribution of \( M_\theta \). In other words, \( q_a \) seeks to be close to \( p_\theta \), i.e., \( q_a \rightarrow p_\theta \). If so, the second KL divergence term in (20) will become zero. Equation (20) is reduced to minimize the KL divergence between the observed data distribution \( p_{\text{expert}} \) and the EBM \( p_\theta \). Eventually, \( q_a \) chases \( p_\theta \) toward \( p_{\text{expert}} \).

IV. EXPERIMENTS

We evaluate the proposed energy-based continuous IOC 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; and the environment \( e \) consists of road condition, speed limit, and 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 article, we sidestep this issue and focus on the IOC problem.

We assume that the dynamic function of all vehicles is a nonlinear 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 that all vehicles are standard two-axle, four-tire passenger cars with a 3-m wheelbase. We set an under-floor passenger cars with a 3-m wheelbase. We set an under-
We use root-mean-square error (RMSE) in meters with respect to each time step $t$ to measure the accuracy of prediction, i.e., \( \text{RMSE}(t) = \left( \frac{1}{n} \sum_{i=1}^{n} \| \hat{y}_{it} - y_{it} \|^2 \right)^{1/2} \), 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 step $t$. A small RMSE is desired.

As a stochastic method, our method draws five samples from the learned model for prediction and the model performance is evaluated by the average RMSE and the minimum RMSE over five 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 as follows.

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-s time steps and is 4 s long.

2) **NGSIM US-101**: NGSIM [50] contains real highway traffic captured at 10 Hz over a time span of 45 min. Compared to the 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-s/50-time-step trajectories. The first ten time steps are for history and the remaining 40 time steps are used for prediction. There are 831 scenes with 96,512 5-s 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 GD to infer controls. In addition to minimizing the reconstruction error on states, we also minimize the $L_2$ norm of the control variables (including acceleration and steering), the difference between the current control value and the control value at the previous time step (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 a fair comparison.

Tables I and II present the 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 $N_{\text{hidden}}$ is 64 and the number of layers is 3 (i.e., two hidden layers and one 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 proximal policy optimization (PPO) policy [51] in the generative adversarial imitation learning (GAIL) [27].

**D. Training Details**

1) **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 current speed and the speed limit, the difference between the vehicle direction and the lane direction, the $L_2$ norm of the control values (including acceleration and steering), the difference between the current control value and the control value at the previous time step (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 a fair comparison.
and unit variance. We also normalize each hand-designed feature by dividing by the mean. We normalize two datasets separately.

2) Optimizer: We use the Adam optimizer on training our models. Both $\beta_1$ and $\beta_2$ 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 the learning rate of Adam to be 0.1 with an exponential decay rate of 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 of 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 1024.

3) 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 GD method to synthesize the controls.

4) 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 as follows.

1) Constant Velocity: The simplest baseline with a constant velocity and a zero steering.

2) 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 PPO [51].

3) 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 s to predict a full trajectory with a 64-step Langevin dynamics (or GD). Fig. 2 shows two qualitative results. Each row shows one four-frame example with a frame interval equal to 1 s. Each frame shows trajectories over time for different vehicles as well as different baseline methods for comparison. Tables 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.

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.

| 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  | 0.766  |
| 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  |

| Method          | 1s     | 2s     | 3s     | 4s     |
|-----------------|--------|--------|--------|--------|
| Constant Velocity | 0.369  | 1.623  | 3.073  | 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  | 0.830  | 1.660  |
| 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.

In addition, the optimal control of our method is performed by simulating trajectories of actions and states according to the learned cost function that considers the future information. In contrast, the GAIL relies on its learned policy net for stepwise decision-making.

Compared with GD (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 suboptimal solutions.

F. Corner Case Testing With Toy Examples

Corner cases are important for model evaluation. We construct six typical corner cases to test our model. Fig. 3 shows the predicted trajectories by our method for several cases. Fig. 3(a) and (b) shows 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 Fig. 3(a), there are not any other vehicles moving alongside the ego vehicle, so it is predicted to first change the lane, then accelerate past the vehicle in front, and return to its previous lane and continue its driving. In Fig. 3(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. Fig. 3(c) and (d) shows 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. Fig. 3(e) and (f) shows two cases in the large lane curvature situation, where our model can still perform well to predict reasonable trajectories.

Fig. 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 dashed 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.

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. In particular, we design a cost function by MLP, where we put three layers on top of the sequence of vectors of hand-designed features, where the kernel size in each layer is 1×4. One fully connected layer with a single kernel is attached at the end.

The numbers of channels are {32, 64, 128, 256} and the numbers of strides are {2, 2, 2, 1} for different layers. 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.

**H. Multiagent 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 multiagent 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 that 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., $X = (x^k, k = 1, 2, \ldots, K), U = (u^k, k = 1, 2, \ldots, K)$. All the agents share the same dynamic function, which is $x^k_t = f(x^k_{t-1}, u^k_t), \forall k = 1, 2, \ldots, K$. The overall cost function is set to be the sum of each agent $C_\theta(X, U, e, h) = \sum_{k=1}^K C_\theta(x^k, u^k, e, h^k)$. Thus, the conditional probability density function becomes $p_\theta(U|e, h) = (1/(Z_\theta(e, h))) \exp[-C_\theta(X, U, e, h)],$ where $Z_\theta(e, h)$ is the intractable normalizing constant.

We compare our method with the following baselines for multiagent control.

1) **Constant Velocity**: The simplest baseline with a constant velocity and a zero steering.

2) **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 multiagent 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. Fig. 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 time steps 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 multiagent control scenario.

**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 $F_\alpha$ as a four-layer MLP with output dimensions 64, 16, 8, and 2 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 five 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 and 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.
Fig. 6. Influence of hyperparameters. Performance comparison of EBMs with different numbers of Langevin steps and Langevin step sizes is shown in each subfigure. Each curve represents a model with a certain Langevin step size $\delta$. We set $\delta = 0.02, 0.05, 0.1, \text{and} 0.2$. For each setting of $\delta$, we choose different numbers of steps $l = 2, 8, 16, 32, \text{and} 64$. Performances are measured by (a) average RMSE, (b) minimum RMSE, and (c) missing rate on the Massachusetts driving dataset.

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

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  |
|-----------------|-----|-----|-----|-----|-----|
| EB w/o a generator | 0.845 | 0.746 | 0.709 | 0.672 | 0.636 |
| EB w/ a generator | 0.804 | 0.672 | 0.649 | 0.638 | 0.633 |
| generator only   |     |     |     |     | 0.911 |

Besides, we implement method (3), which is a single trajectory generator trained via MLE with MCMC-based inference [47]. This comparison results show that a fast initializer can improve the performance even with less Langevin steps. For example, an EBM using eight 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 1024 during training. For GAIL, the mini-batch size is 64. The total number of epochs is 40. Table IX lists 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.

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 first study the influence of different choices of some hyperparameters, such as the number of Langevin steps $l$ and the step size $\delta$ of each Langevin step. Fig. 6 shows the performances of energy-based IOC models with different $\delta$ and $l$ on the Massachusetts driving dataset. Each curve is associated with a certain step size $\delta$ and shows the testing performances over different numbers of Langevin steps. The performances are measured by: 1) average RMSE; 2) minimum RMSE; and 3) missing rate. In our experiments, we draw five samples from the learned model for prediction. The missing rate is the ratio of scenarios where none of all five sampled trajectories has an endpoint $L2$ error less than 1.0 m. The three metrics are used in Fig. 6(a)–(c). In general, with the same $\delta$, 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 tradeoff between performance and computational efficiency. We also choose \( \delta = 0.1 \) for a tradeoff among performances measured by different metrics.

Fig. 7(a) and (b) shows the training curves of the models with different values of \( l \) and \( \delta \), 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 \( \delta \) 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 \( \delta \) in testing can affect the performance of the model. Fig. 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 \( \delta \). The model we use is with a linear cost function and trained with \( l = 64 \) and \( \delta = 0.1 \). We observe that, in the training 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 nonconvergence issue. Thus, we suggest using the same \( l \) and \( \delta \) 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 \( \delta \) in the cooperative training framework. Fig. 8 shows the performances of cooperative training frameworks with different values of \( \delta \) and \( l \) on the Massachusetts driving dataset. The performances shown in Fig. 8(a)–(c) are measured by average RMSE, minimum RMSE, and missing rate, respectively. Each curve corresponds to a framework with a certain step size \( \delta \) and shows the testing performances over different numbers of Langevin steps. Fixing \( \delta \), the model performance increases as the number of Langevin steps increases. We use \( l = 32 \) and \( \delta = 0.1 \) to make a tradeoff between performance and computational efficiency.

Fig. 9(a) and (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 training epochs. We observe that the testing average RMSE decreases smoothly as the number of training epochs increases. We also study how different values of \( l \) and \( \delta \) chosen in testing affect the performance of a learned energy-based IOC model. We first train an EBM with \( \delta = 0.2 \) and \( l = 32 \) and use the learned model in testing with varying Langevin step size \( \delta \) and a number of Langevin steps \( l \). Fig. 9(c) shows the influences of varying \( \delta \) 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 article 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 EBM and propose a sampling-based method and optimization-based modification to learn the cost function. Unlike the previous method for continuous IOC [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 suboptimal modes. Moreover, we propose to train the EBM 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 nonlinear 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 [41], [58], [59] 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 suboptimal, and some human judges may assign scores to the trajectories of some of the drivers.

REFERENCES

[1] E. Todorov, “Optimal control theory,” in Bayesian Brain: Probabilistic Approaches to Neural Coding, 2006, pp. 269–296.

[2] W. Li and E. Todorov, “Iterative linear quadratic regulator design for nonlinear biological movement systems,” in Proc. 1st Int. Conf. Informat. Control, Autom. Robot. (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, Jan. 2002.

[4] J. Xie, Y. Lu, S.-C. Zhu, and Y. Wu, “A theory of generative convnet,” in Proc. Conf. Learn. Autom. (ICLA), 2016, pp. 2635–2644.

[5] R. M. Neal et al., “MCMC using Hamiltonian dynamics,” in Handbook of Markov Chain Monte Carlo, vol. 2, no. 11. 2011. p. 2. [online]. Available: https://www.taylorfrancis.com/books/edit/10.1201/b19095/handbook-markov-chain-monte-carlo-steve-brooks-andrew-gelman-galin-jones-xiao-li-meng

[6] I. J. Goodfellow and K. A. Doshi, Mathematical Statistics: Basic Ideas and Selected Topics, Package, vols. 1–2. Boca Raton, FL, USA: CRC Press, 2015.

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

[8] A. Hyvärinen, “Estimation of non-normalized statistical models by score matching,” J. Mach. Learn. Res., 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 Proc. AAAI Conf. Artif. Intell. (AAAI), 2018, vol. 32, no. 1.

[10] J. Xie, Y. Lu, R. Gao, S.-C. Zhu, and Y. N. Wu, “Cooperative training of descriptor and generator networks,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 42, no. 1, pp. 27–45, Jan. 2020.

[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 Trans. Pattern Anal. Mach. Intell., early access, Mar. 26, 2021, doi: 10.1109/TPAMI.2021.3069023.

[12] B. D. Ziebart, A. L. Maas, J. A. Bagnell, and A. K. Dey, “Maximum entropy inverse reinforcement learning,” in Proc. 23rd AAAI Conf. Artif. Intell. (AAAI), Chicago, Ill., USA, vol. 8, 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,” Int. J. Comput. Vis., vol. 27, no. 2, pp. 107–126, 1998.

[14] M. Richardson and P. Domingos, “Markov logic networks,” Mach. Learn., vol. 62, no. 1, pp. 107–136, Feb. 2006.

[15] M. Wulfmeier, P. Endrudska, and I. Posner, “Maximum entropy deep inverse reinforcement learning,” 2015, arXiv:1507.04888.

[16] J. Xie, S.-C. Zhu, and Y. N. Wu, “Learning energy-based spatial-temporal generative ConvNets for dynamic patterns,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 43, no. 2, pp. 516–531, Feb. 2021.
[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 Proc. Int. Conf. Learn. Represent. (ICLR), 2022.

[41] D. P. Kingma and P. Dhariwal, “Glow: Generative flow with invertible 1×1 convolutions,” in Proc. Adv. Neural Inf. Process. Syst. (NeurIPS), vol. 31, 2018.

[42] H. Robbins and S. Monro, “A stochastic approximation method,” Ann. Math. Statist., vol. 22, no. 3, pp. 400–407, 1951.

[43] T. M. Cover and J. A. Thomas, Elements of Information Theory, 2nd ed. Hoboken, NJ, USA: Wiley, 2006.

[44] W. K. Hastings, “Monte Carlo sampling methods using Markov chains and their applications,” Tech. Rep., 1970.

[45] T. Chen, E. B. Fox, and C. Guestrin, “Stochastic gradient Hamiltonian Monte Carlo,” in Proc. Int. Conf. Mach. Learn. (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 Proc. AAAI Conf. Artif. Intell. (AAAI), 2020, vol. 34, no. 4, 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 Proc. AAAI Conf. Artif. Intell. (AAAI), vol. 33, 2019, pp. 5498–5507.

[48] P. Polack, F. Altiene, B. d’Andrea-Novel, and A. de La Fortelle, “The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?” in Proc. IEEE Intell. Vehicles Symp. (IV), Jun. 2017, pp. 812–818.

[49] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2014, arXiv:1412.6980.

[50] J. Colyar and J. Halkias, “U.S. highway 101 dataset,” Federal Highway Admin. (FHWA), Washington, DC, USA, Tech. Rep. FHWA-HRT-07-030, 2007.

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

[52] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification,” in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), Dec. 2015, pp. 1026–1034.

[53] A. Kuefler, J. Morton, T. Wheeler, and M. Kochenderfer, “Imitating driver behavior with generative adversarial networks,” in Proc. IEEE Intell. Vehicles Symp. (IV), Jun. 2017, pp. 204–211.

[54] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, “Trust region policy optimization,” in Proc. Int. Conf. Mach. Learn. (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 Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS), Oct. 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 Proc. Int. Conf. Robot. Autom. (ICRA), May 2019, pp. 789–795.

[57] J. K. Gupta, M. Egorov, and M. Kochenderfer, “Cooperative multi-agent control using deep reinforcement learning,” in Proc. Int. Conf. Auton. Agents Multiagent Syst. (AAMAS), 2017, pp. 66–83.

[58] L. Dinh, D. Krueger, and Y. Bengio, “NICE: Non-linear independent components estimation,” in Proc. Int. Conf. Learn. Representations (ICLR) Workshop, 2015.

[59] L. Dinh, J. Sohl-Dickstein, and S. Bengio, “Density estimation using real NVP,” in Proc. Int. Conf. Learn. Represent. (ICLR), 2017.

Yifei Xu received the B.E. degree in computer science from Shanghai Jiao Tong University, Shanghai, China, in 2017, and the Ph.D. degree in statistics from the University of California at Los Angeles (UCLA), Los Angeles, CA, USA, in 2022. His research interests focus on generative model, reinforcement learning, and computer vision.

Jianwen Xue (Member, IEEE) received the Ph.D. degree in statistics from the University of California at Los Angeles (UCLA), Los Angeles, CA, USA, in 2016. Before joining Baidu, he was a Senior Research Scientist at the Hikvision Research Institute, Santa Clara, CA, USA, from 2017 to 2020, and a Staff Research Associate and Post-Doctoral Researcher with the Center for Vision, Cognition, Learning, and Autonomy (VCLA), UCLA, from 2016 to 2017. He is currently a Senior Research Scientist at the Cognitive Computing Laboratory, Baidu Research, Bellevue, WA, USA. His research interests focus on generative modeling and unsupervised learning.

Tianyang Zhao received the B.S. degree in data science and big data technology from Peking University, Beijing, China, in 2019. He is currently pursuing the Ph.D. degree with the Center for Vision, Cognition, Learning and Autonomy, University of California at Los Angeles (UCLA), Los Angeles, CA, USA. His research interests lie in generative model and representation learning.

Yibiao Zhao received the Ph.D. degree in statistics from the University of California at Los Angeles (UCLA), Los Angeles, CA, USA, in 2015. He is currently a Co-Founder and the CEO of iSee, Inc., Cambridge.

Chris Baker received the Ph.D. degree in cognitive science from the Massachusetts Institute of Technology (MIT), Cambridge, MA, USA, in 2012. He is currently a Co-Founder and a Chief Scientist at iSee, Inc., Cambridge.

Ying Nian Wu (Member, IEEE) received the Ph.D. degree in statistics from Harvard University, Cambridge, MA, USA, in 1996. He was an Assistant Professor with the Department of Statistics, University of Michigan, Ann Arbor, MI, USA, from 1997 to 1999. He joined the University of California at Los Angeles (UCLA), Los Angeles, CA, USA, in 1999, where he is currently a Professor with the Department of Statistics. His research interests include generative models, representation learning, and computer vision.

Dr. Wu received the Honorable Mention for the David Marr Prize with S. C. Zhu et al. in 1999 and 2007 for generative modeling in computer vision.