Title

A Causality-Free Neural Network Method for High-Dimensional Hamilton-Jacobi-Bellman Equations

ISBN

9781538682661

Authors

Nakamura-Zimmerer, Tenavi
Gong, Qi
Kang, Wei

Publication Date

2020-07-03

DOI

10.23919/acc45564.2020.9147270

Peer reviewed

A Causality-Free Neural Network Method for High-Dimensional Hamilton-Jacobi-Bellman Equations

Tenavi Nakamura-Zimmerer[1], Qi Gong[1], and Wei Kang[2]

***Abstract*— Computing optimal feedback controls for nonlinear systems generally requires solving Hamilton-Jacobi-Bellman (HJB) equations, which, in high dimensions, are notoriously difficult. Existing strategies often rely on specific, restrictive problem structures, or are valid only locally around some nominal trajectory. In this paper, we propose a data-driven method to approximate semi-global solutions to HJB equations for general high-dimensional nonlinear systems and compute optimal feedback controls in real-time. To accomplish this, we model solutions to HJB equations with neural networks (NNs) trained on data generated without discretizing the state space. Training is made more effective and data-efficient by leveraging the known problem structure and using the partially-trained NN to aid in further data generation. We demonstrate the effectiveness of our method by learning solutions to HJB equations for nonlinear systems of dimension up to 30 arising from the stabilization of a Burgers'-type partial differential equation. The trained NNs are then used for real-time optimal feedback control of these systems.**

I. Introduction

For the optimal control of nonlinear dynamical systems, it is well-known that open-loop controls are not robust to model uncertainty or disturbances. For slowly evolving processes, it is possible to use model predictive control by recomputing the open-loop optimal solution for a relatively short time horizon in the future. However, for most applications one typically desires an optimal feedback control law, as feedback controls are inherently more robust to disturbances. Using dynamic programming, the optimal feedback control is computed by solving a (discretized) Hamilton-Jacobi-Bellman (HJB) equation, a partial differential equation (PDE) in n spatial dimensions plus time. The size of the discretized problem increases exponentially with n, making direct solution intractable for even moderately high-dimensional problems. This is the so-called "curse of-dimensionality."

For this reason, there is an extensive literature on methods of finding approximate solutions for HJB equations. We have no intention to give a full review of existing results except for a short list of some related publications, [1]–[8]. These existing methods suffer from one or more of the following drawbacks: the problem's dimension is limited; the accuracy of the solution is hard to verify for general systems; the solution may be valid only in a small neighborhood; or the system model must have certain special algebraic structure.

[1]Tenavi Nakamura-Zimmerer and Qi Gong are with the Department of Applied Mathematics, Baskin School of Engineering, University of California, Santa Cruz tenakamu@ucsc.edu, qigong@soe.ucsc.edu

[2]Wei Kang is with the Department of Applied Mathematics, Naval Postgraduate School, Monterery, CA wkang@nps.edu

Recently, deep learning approaches for solving high-dimensional PDEs and HJB equations have produced promising results; see e.g. [9]–[12]. Existing neural network-based techniques generally rely on either minimizing the residual of the PDE and (artificial) boundary conditions at randomly sampled collocation points [9], [10]; or modeling the solution and its gradient near a nominal trajectory [12]. For HJB equations arising in stochastic optimal control, [11] use a NN basis to solve stochastic differential equations connected to the solution of the HJB equation.

In this paper, we introduce a computational framework for solving high-dimensional HJB equations and generating fully nonlinear optimal feedback controls. Our approach is data-driven and consists of three main steps. First, a small set of open-loop optimal control solutions is generated by solving two-point boundary value problems (BVPs) derived from Pontryagin's Minimum Principle (PMP). This data generation algorithm is *causality-free*, i.e. the solution at each point can be computed without using the value of the solution at other points. This frees us from having to discretize the state space and permits perfectly parallelizable data generation. In the second step, we use the data set to train a neural network (NN) to approximate the solution to the HJB equation, called the *value function*, and its gradient. Supplying this gradient information encourages the NN to learn the shape of the value function, rather than just fitting point data. If needed, additional data can be obtained quickly with the aid of the NN. Lastly, the accuracy of the NN is verified on another data set that is generated using the causality free algorithm.

Solution of high-dimensional problems is enabled by causality-free data generation, physics-informed learning, and the inherent capacity of NNs for dealing with high-dimensional data. Furthermore, once the NN is trained offline, evaluation of the control is very fast. This allows computation of the control in real-time, which is essential for a feedback implementation. Unlike other NN-based methods, our approach requires computation of expensive PDE residuals, and the solution is valid over large spatial domains.

As an illustrative example, the method is applied problems of dimension $n = 10$, 20, and 30 arising from pseudospectral discretization of an open-loop unstable Burgers'-type PDE. Through this example, we demonstrate several advantages and potential capabilities of the method. These include solving high-dimensional HJB equations on semi-global domains and with empirically validated levels of accuracy, computationally efficient NN-based feedback control for real-time applications, and the ability to generate rich data sets.

II. A CAUSALITY-FREE METHOD FOR HJB

Consider the optimal control problem

$$\begin{cases} \underset{\boldsymbol{u}\in\mathcal{U}}{\text{minimize}} & F(\boldsymbol{x}(t_f)) + \int_0^{t_f} L(t,\boldsymbol{x},\boldsymbol{u})dt, \\ \text{subject to} & \dot{\boldsymbol{x}} = \boldsymbol{f}(t,\boldsymbol{x},\boldsymbol{u}). \end{cases} \tag{1}$$

Here $\boldsymbol{x}(t) : [0,t_f] \to \mathcal{X} \subseteq \mathbb{R}^n$ is the state, $\boldsymbol{u}(t,\boldsymbol{x}) : [0,t_f] \times \mathcal{X} \to \mathcal{U} \subseteq \mathbb{R}^m$ is the control, $\boldsymbol{f}(t,\boldsymbol{x},\boldsymbol{u}) : [0,t_f] \times \mathcal{X} \times \mathcal{U} \to \mathbb{R}^n$ is a Lipschitz continuous vector field, $F(\boldsymbol{x}(t_f)) : \mathcal{X} \to \mathbb{R}$ is the terminal cost, and $L(t,\boldsymbol{x},\boldsymbol{u}) : [0,t_f] \times \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ is the running cost. For simplicity let the final time t_f be fixed.

For a given initial condition $\boldsymbol{x}(0) = \boldsymbol{x}_0$, many methods exist to compute the optimal open-loop solution, $\boldsymbol{u} = \boldsymbol{u}^*(t;\boldsymbol{x}_0)$. Due to various sources of disturbance and real-time application requirements, for practical implementation one typically desires an optimal control in closed-loop feedback form, $\boldsymbol{u} = \boldsymbol{u}^*(t,\boldsymbol{x})$. To compute the optimal feedback control, we follow the standard procedure in optimal control (see e.g. [13]) and define the Hamiltonian

$$H(t,\boldsymbol{x},\boldsymbol{\lambda},\boldsymbol{u}) \coloneqq L(t,\boldsymbol{x},\boldsymbol{u}) + \boldsymbol{\lambda}^T \boldsymbol{f}(t,\boldsymbol{x},\boldsymbol{u}), \tag{2}$$

where $\boldsymbol{\lambda}(t) : [0,t_f] \to \mathbb{R}^n$ is the costate. The optimal control satisfies

$$\boldsymbol{u}^*(t,\boldsymbol{x},\boldsymbol{\lambda}) = \arg\min_{\boldsymbol{u}\in\mathcal{U}} H(t,\boldsymbol{x},\boldsymbol{\lambda},\boldsymbol{u}). \tag{3}$$

Now we define the value function,

$$V(t,\boldsymbol{x}) \coloneqq \inf_{\boldsymbol{u}\in\mathcal{U}} \left\{ F(\boldsymbol{x}(t_f)) + \int_t^{t_f} L(\tau,\boldsymbol{x},\boldsymbol{u})d\tau \right\}. \tag{4}$$

The value function is the optimal cost-to-go of (1) and is the unique (viscosity) solution to the Hamilton-Jacobi-Bellman (HJB) equation,

$$\begin{cases} -\left[V_t(t,\boldsymbol{x}) + H^*\left(t,\boldsymbol{x},V_{\boldsymbol{x}}\right)\right] = 0, \\ V(t_f,\boldsymbol{x}) = F(\boldsymbol{x}), \end{cases} \tag{5}$$

where $H^*(t,\boldsymbol{x},\boldsymbol{\lambda}) \coloneqq H(t,\boldsymbol{x},\boldsymbol{\lambda},\boldsymbol{u}^*)$, $V_t \coloneqq \partial V/\partial t$, and $V_{\boldsymbol{x}} \coloneqq \partial V/\partial \boldsymbol{x}$. If (5) can be solved, then the optimal control is computed by substituting

$$\boldsymbol{\lambda}(t) = V_{\boldsymbol{x}}(t,\boldsymbol{x}) \tag{6}$$

into (3) to get

$$\boldsymbol{u}^*(t,\boldsymbol{x}) = \arg\min_{\boldsymbol{u}\in\mathcal{U}} H\left(t,\boldsymbol{x},V_{\boldsymbol{x}},\boldsymbol{u}\right). \tag{7}$$

This means that with $V_{\boldsymbol{x}}(\cdot)$ available, the feedback control is obtained as the solution of an (ideally straightforward) optimization problem.

Unfortunately, directly solving (5) directly is computationally intractable for even moderately high-dimensional problems. However, following the strategy in [8], we exploit the fact that the characteristics of the value function evolve according to

$$\begin{cases} \dot{\boldsymbol{x}}(t) = \dfrac{\partial H}{\partial \boldsymbol{\lambda}} = \boldsymbol{f}(t,\boldsymbol{x},\boldsymbol{u}^*(t,\boldsymbol{x},\boldsymbol{\lambda})), \\ \dot{\boldsymbol{\lambda}}(t) = -\dfrac{\partial H}{\partial \boldsymbol{x}}(t,\boldsymbol{x},\boldsymbol{\lambda},\boldsymbol{u}^*(t,\boldsymbol{x},\boldsymbol{\lambda})), \\ \dot{v}(t) = -L(t,\boldsymbol{x},\boldsymbol{u}^*(t,\boldsymbol{x},\boldsymbol{\lambda})), \end{cases} \tag{8}$$

with two-point split boundary conditions

$$\begin{cases} \boldsymbol{x}(0) = \boldsymbol{x}_0, \\ \boldsymbol{\lambda}(t_f) = \dfrac{dF}{d\boldsymbol{x}}(\boldsymbol{x}(t_f)), \\ v(t_f) = F(\boldsymbol{x}(t_f)). \end{cases} \tag{9}$$

For any given initial condition $\boldsymbol{x}_0$, the optimal control and value function along the characteristic $\boldsymbol{x}(t;\boldsymbol{x}_0)$ are then given by

$$\boldsymbol{u}^*(t,\boldsymbol{x}) = \boldsymbol{u}^*(t;\boldsymbol{x}_0), \qquad V(t,\boldsymbol{x}) = v(t;\boldsymbol{x}_0). \tag{10}$$

Solutions to the two-point BVP (8–9) can be computed independently of one another, so this approach is causality-free. In [8], the authors construct a sparse grid of initial conditions and solve (8–9) at each grid point. They then interpolate the costate and apply PMP to obtain the feedback control. This is called the sparse grid characteristics method. However, even using a sparse grid the number of points grows like $O\left(N(\log N)^{n-1}\right)$, where n is the state dimension and N is the number of grid points in each dimension. Thus this approach still suffers from the curse of-dimensionality. In this paper, instead of sparse grid interpolation we use data from solved BVPs to train a NN to approximate the value function $V(t,\boldsymbol{x})$. The proposed method is thus completely grid-free and applicable in high dimensions.

Remark 1: The two-point BVP (8–9) provides a necessary condition for optimality and is well-known in optimal control theory as Pontryagin's Minimum Principle (PMP). In general, however, solutions of the BVP are not unique and may be sub-optimal, i.e. the characteristics of the value function satisfy (8–9), but there may be other solutions to these equations which are sub-optimal and therefore not characteristics of the value function. Optimality of solutions to the BVP can be guaranteed under certain convexity conditions, see e.g. [14], [15]. Addressing this challenging problem in a broader context is beyond the scope of the present work, but for the example problems we deal with, solutions of (8–9) satisfy the sufficient conditions as well.

A. Data generation algorithms

Although solving the BVP (8–9) is easier than solving the full HJB equation (5), it is still often very difficult, due to the well-known high sensitivity to the boundary conditions (9). So far, there is no general algorithm that is reliable and fast enough for real-time applications. However, in our approach the real-time feedback control computation is done by a NN trained *offline*. Thus we can solve the BVP offline to generate data for training and evaluating such a NN. For this purpose, numerically solving the BVP can be manageable although it may require some parameter tuning. In this paper, we use an implementation of the BVP solver introduced in [16]. This algorithm is based on a three-stage Lobatto IIIa discretization, a collocation formula which provides a solution that is fourth-order accurate. But the algorithm is highly sensitive to the initial guess for $\boldsymbol{x}(t)$ and $\boldsymbol{\lambda}(t)$: there is no guarantee of convergence with an arbitrary initial guess.

Furthermore, convergence is increasingly dependent on good initializations as we increase the length of the time interval.

To overcome this difficulty, we employ the *time-marching* trick from [8] in which the solution grows from an initially short time interval to the final time t_f. More specifically, we choose a time sequence

$$0 < t_1 < t_2 < \cdots < t_K = t_f,$$

in which t_1 is small. For the short time interval $[0, t_1]$, the BVP solver converges given most initial guesses near the initial state $\boldsymbol{x}$. Then, the resulting trajectory is rescaled over the longer time interval $[0, t_2]$. The rescaled trajectory is used as the initial guess to find a solution of the BVP for $0 \leq t \leq t_2$. We repeat this process until $t_K = t_f$, at which we obtain the full solution. Still, it is necessary to tune the time sequence $\{t_k\}_{k=1}^K$ to achieve convergence while maintaining acceptable efficiency.

Computing many such solutions becomes expensive, which means that generating the large data sets necessary to train a NN can be difficult. With this in mind, we use time-marching only to generate a (small) initial data set, and later increase the size of the data set if needed. The key to doing this efficiently is simulating the system dynamics using the partially-trained NN to close the loop. This quickly provides good guesses for the optimal state $\boldsymbol{x}^*(t)$ and costate $\boldsymbol{\lambda}(t)$ over the entire time interval $[0, t_f]$, so that we can immediately solve (8–9) for all of $[0, t_f]$. We refer to this technique as *NN warm start*. A numerical comparisons between this method and the time-marching trick is given in Section IV-B.

III. Neural network modelng

Neural networks have become a popular tool for modeling high-dimensional functions, since they are not dependent on discretizing the state space. In this paper, we apply NNs to approximate solutions of the HJB equation and evaluate the resulting feedback control in real-time. Specifically, we carry out the following steps:

1) *Data generation:* We compute the value function, $V(t, \boldsymbol{x})$, on trajectories $\boldsymbol{x}(t)$ from initial conditions chosen by Monte Carlo sampling. Data is generated by solving the BVP as discussed in Section II-A. In this initial data generation step, we require relatively few data points, since more data can be added later at little computational cost.
2) *Model training:* Given this data set, we train a NN to approximate the value function $V(\cdot)$. Learning is guided by the underlying structure of the problem, specifically by asking the NN to satisfy (6). In doing so, we regularize the model and make efficient use out of small data sets.
3) *Model validation and refinement:* Next we check the generalization accuracy of the trained NN on a new set of validation data computed at Monte Carlo sample points. If necessary, we use the partially-trained NN to aid in generating additional data and continue training the model on the expanded data set.
4) *Feedback control:* We compute the optimal feedback control online by evaluating the gradient of the trained NN, $V_{\boldsymbol{x}}^{NN}(\cdot)$, and applying PMP. Notably, evaluation of the gradient is extremely cheap even for large n, enabling implementation in high-dimensional systems.

The crux of the proposed method depends on modeling the value function (4) over a semi-global domain $\mathcal{X} \subset \mathbb{R}^n$. We present details of this process in the following subsections. In Section III-A, we review the basic structure of feedforward NNs and describe how we train a NN to model the value function. Then in Section III-B, we propose a simple way to incorporate information about the known problem structure into training. Finally, in Section IV the method is applied to synthesize optimal controls for an open-loop unstable Burgers-type PDE.

A. Feedforward neural networks

In this paper we use multilayer feedforward NNs. While more sophisticated architectures have been developed for some applications, we find this basic architecture to be more than adequate for our purposes. Let $V(\cdot)$ be the function we wish to approximate and $V^{NN}(\cdot)$ be its NN representation:

$$V(t, \boldsymbol{x}) \approx V^{NN}(t, \boldsymbol{x}) = g_M \circ \boldsymbol{g}_{M-1} \circ \cdots \circ \boldsymbol{g}_1(t, \boldsymbol{x}),$$

where each layer $\boldsymbol{g}_m(\cdot)$ is just

$$\boldsymbol{g}_m(\boldsymbol{y}) = \sigma_m(\boldsymbol{W}_m \boldsymbol{y} + \boldsymbol{b}_m).$$

Here $\boldsymbol{W}_m$ and $\boldsymbol{b}_m$ are the weight matrices and bias vectors, respectively. $\sigma_m(\cdot)$ represents a nonlinear *activation function* applied component-wise to its argument; popular choices include ReLU, tanh, and other similar functions. In this paper, we use tanh for all the hidden layers. The final layer, $g_M(\cdot)$, is typically linear, so $\sigma_M(\cdot)$ is the identity function.

We denote by $\boldsymbol{\theta}$ the collection of all parameters, i.e.

$$\boldsymbol{\theta} \coloneqq \{\boldsymbol{W}_m, \boldsymbol{b}_m\}_{m=1}^M.$$

The NN is trained by optimizing over the parameters $\boldsymbol{\theta}$ to best approximate $V(t, \boldsymbol{x})$ by $V^{NN}(t, \boldsymbol{x}; \boldsymbol{\theta})$. Specifically, by solving the BVP (8–9) from a set of randomly sampled initial conditions, we get a data set

$$\mathcal{D} = \left\{ \left(\left(t^{(i)}, \boldsymbol{x}^{(i)} \right), V^{(i)} \right) \right\}_{i=1}^{N_d},$$

where $\left(t^{(i)}, \boldsymbol{x}^{(i)}\right)$ are the inputs, $V^{(i)} \coloneqq V\left(t^{(i)}, \boldsymbol{x}^{(i)}\right)$ are the outputs to be modeled, and $i = 1, 2, \ldots, N_d$ are the indices of the data points. The NN is then trained by solving the nonlinear regression problem,

$$\min_{\boldsymbol{\theta}} \frac{1}{N_d} \sum_{i=1}^{N_d} \left[V^{(i)} - V^{NN}\left(t^{(i)}, \boldsymbol{x}^{(i)}; \boldsymbol{\theta} \right) \right]^2. \tag{11}$$

B. Physics-informed learning

Motivated by advances in physics-informed deep learning [17], we expect that we can improve on the rudimentary loss function in (11) by incorporating information about the underlying physics. In [17], and in particular in the context of HJB equations in [9] and [10], a known underlying

PDE and boundary conditions are imposed by minimizing a residual loss over spatio-temporal collocation points. In this approach, no data is gathered: the PDE is solved directly in the least-squares sense. However, the residual must be evaluated over a large number of collocation points and can be rather expensive to compute. Thus we propose a simpler approach of modeling the costate $\boldsymbol{\lambda}(\cdot)$ along with the value function itself, taking full advantage of the ability to gather data along the characteristics of the HJB PDE.

Specifically, we know that the costate must satisfy (6), so we train the NN to minimize

$$\left\|\boldsymbol{\lambda}(t;\boldsymbol{x}) - V_{\boldsymbol{x}}^{NN}(t,\boldsymbol{x};\boldsymbol{\theta})\right\|^2,$$

where $V_{\boldsymbol{x}}^{NN}(\cdot)$ is the gradient of the NN representation of the value function with respect to the state, which is calculated using automatic differentiation. In machine learning, automatic differentiation is usually used to compute gradients with respect to the model parameters, but is just as easy to apply to computing gradients with respect to inputs. The computational graph is also pre-compiled so evaluating the gradient is cheap.

Costate data $\boldsymbol{\lambda}(t)$ is obtained for each trajectory as a natural product of solving the BVP (8–9). Hence we have the augmented data set,

$$\bar{\mathcal{D}} = \left\{\left(\left(t^{(i)},\boldsymbol{x}^{(i)}\right),\left(V^{(i)},\boldsymbol{\lambda}^{(i)}\right)\right)\right\}_{i=1}^{N_d}, \tag{12}$$

where $\boldsymbol{\lambda}^{(i)} \coloneqq \boldsymbol{\lambda}\left(t^{(i)};\boldsymbol{x}^{(i)}\right)$. We now define the *physics-informed learning problem*,

$$\begin{cases} \min\limits_{\boldsymbol{\theta}} \quad \mathcal{L}\left(\boldsymbol{\theta};\bar{\mathcal{D}}\right), \\ \text{where} \quad \mathcal{L}\left(\boldsymbol{\theta};\bar{\mathcal{D}}\right) \coloneqq \mathcal{L}_V\left(\boldsymbol{\theta};\bar{\mathcal{D}}\right) + \mu\cdot\mathcal{L}_{\boldsymbol{\lambda}}\left(\boldsymbol{\theta};\bar{\mathcal{D}}\right). \end{cases} \tag{13}$$

Here $\mu \geq 0$ is a scalar weight, the loss with respect to data is

$$\mathcal{L}_V\left(\boldsymbol{\theta};\bar{\mathcal{D}}\right) \coloneqq \frac{1}{N_d}\sum_{i=1}^{N_d}\left[V^{(i)} - V^{NN}\left(t^{(i)},\boldsymbol{x}^{(i)};\boldsymbol{\theta}\right)\right]^2, \tag{14}$$

and the physics-informed gradient loss regularization is defined as

$$\mathcal{L}_{\boldsymbol{\lambda}}\left(\boldsymbol{\theta};\bar{\mathcal{D}}\right) \coloneqq \frac{1}{N_d}\sum_{i=1}^{N_d}\left\|\boldsymbol{\lambda}^{(i)} - V_{\boldsymbol{x}}^{NN}\left(t^{(i)},\boldsymbol{x}^{(i)};\boldsymbol{\theta}\right)\right\|^2. \tag{15}$$

A NN trained to minimize (13) learns not just to fit the value data, but it is rewarded for doing so in a way that respects the underlying structure of the problem. This physics-informed regularization takes the known problem structure into account, so is preferable to the usual L^1 or L^2 regularization, which are based on the (heuristic) principle that simpler representations of data are likely to generalize better. Furthermore, we recall that the optimal control depends explicitly on $V_{\boldsymbol{x}}(\cdot)$ through (7). Accurate approximation of $V_{\boldsymbol{x}}(\cdot)$ is therefore essential for calculating optimal controls. Our method achieves this through automatic differentiation to compute *exact* gradients and by minimization of the physics-informed loss term (15).

In common practice, one usually randomly partitions the given data set (12) into a training set $\bar{\mathcal{D}}_{\text{train}}$ and validation set $\bar{\mathcal{D}}_{\text{val}}$. During training, the loss functions (14) and (15) are calculated with respect to the training data $\bar{\mathcal{D}}_{\text{train}}$. We then evaluate the performance of the NN against the validation data $\bar{\mathcal{D}}_{\text{val}}$, which it did not observe during training. Good validation performance indicates that the NN generalizes well, i.e. it did not overfit the training data. We make the validation test more stringent by generating $\bar{\mathcal{D}}_{\text{train}}$ and $\bar{\mathcal{D}}_{\text{val}}$ from *independently drawn* initial conditions, so that the two data sets do not share any part of the same trajectories.

IV. Application to Burgers'-type PDE

In this section, we test our method on high-dimensional nonlinear systems of ODEs arising from Chebyshev pseudospectral (PS) discretization of a one-dimensional forced Burgers'-type PDE. An infinite-horizon version of this problem is studied in [7], in which the value function is approximated by a polynomial. We note that in [7], separability of the nonlinear dynamics is required to compute the high-dimensional integrals necessary in the Galerkin formulation. Our method does not require this restriction, although it does apply in this problem.

As in [7], let $X(t,\xi) : [0,t_f]\times[-1,1] \to \mathbb{R}$ satisfy the following one-dimensional controlled PDE with Dirichlet boundary conditions:

$$\begin{cases} X_t = XX_\xi + \nu X_{\xi\xi} + \alpha X e^{\beta X} + I_\omega(\xi)u, \\ \qquad \text{for } t>0, \xi\in(-1,1), \\ X(t,-1) = X(t,1) = 0, \\ \qquad \text{for } t>0, \\ X(0,\xi) = X_0, \\ \qquad \text{for } \xi\in(-1,1). \end{cases} \tag{16}$$

For notational convenience we have written $X = X(t,\xi)$. We denote $X_t = \partial X/\partial t$, $X_\xi = \partial X/\partial\xi$, and $X_{\xi\xi} = \partial^2 X/\partial\xi^2$. The scalar-valued control $u(t,X)$ is actuated only on ω, the support of the indicator function

$$I_\omega(\xi) \coloneqq \begin{cases} 1, & \xi\in\omega, \\ 0, & \xi\notin\omega\ . \end{cases}$$

The optimal control problem we consider is

$$\begin{cases} \min\limits_{u(\cdot)} \quad F(X(t_f,\xi)) + \displaystyle\int_t^{t_f} L(X,u)d\tau, \\ \text{s.t.} \quad X_t = XX_\xi + \nu X_{\xi\xi} + \alpha X e^{\beta X} + I_\omega(\xi)u, \\ \qquad X(t,-1) = X(t,1) = 0. \end{cases} \tag{17}$$

Here

$$F(X(t_f,\xi)) = \frac{W_2}{2}\left\|X(t_f,\xi)\right\|^2_{L^2_{(-1,1)}},$$

$$L(X,u) = \frac{W_1}{2}u^2(t,X) + \frac{1}{2}\left\|X(t,\xi)\right\|^2_{L^2_{(-1,1)}},$$

$$\left\|X(t,\xi)\right\|^2_{L^2_{(-1,1)}} \coloneqq \int_{-1}^{1}\left|X(t,\xi)\right|^2 d\xi,$$

and we set

$$\omega = (-0.5,-0.2), \quad \nu = 0.2, \quad \alpha = 1.5,$$
$$\beta = -0.1, \quad W_1 = 0.1, \quad W_2 = 1, \quad t_f = 8.$$

The objective of stabilizing $X(t,\xi)$ is made more challenging by the added source term, $\alpha X e^{\beta X}$, which renders the origin unstable. This can be seen clearly in Fig. 2a.

To solve (17) using our framework, we perform Chebyshev PS collocation to transform the PDE (16) into a system of ODEs. Following [18], we let

$$\xi_j = \cos(j\pi/N_c), \qquad j = 0, 1, \dots N_c,$$

where $N_c + 1$ is the number of collocation points. After accounting for boundary conditions, we collocate $X(t,\xi)$ at *internal* (non-boundary) Chebshev points, ξ_j, $j = 1, 2, \dots, n$, where $n = N_c - 1$. The discretized state $\boldsymbol{x}(t) : [0, t_f] \to \mathbb{R}^n$ is defined as

$$\boldsymbol{x}(t) \coloneqq \big(X(t,\xi_1), \quad X(t,\xi_2), \quad \dots, \quad X(t,\xi_n)\big)^T$$

and the PDE (16) becomes a system of ODEs in n dimensions:

$$\dot{\boldsymbol{x}} = \boldsymbol{x} \odot \boldsymbol{D}\boldsymbol{x} + \nu \boldsymbol{D}^2 \boldsymbol{x} + \alpha \boldsymbol{x} \odot e^{\beta \boldsymbol{x}} + \boldsymbol{I}_\omega u, \qquad (18)$$

In the above, "$\odot$" denotes element-wise multiplication (the Hadamard product), $\boldsymbol{I}_\omega$ is the discretized indicator function, and $\boldsymbol{D}, \boldsymbol{D}^2 \in \mathbb{R}^{n\times n}$ are the internal parts of the first and second order Chebyshev differentiation matrices, obtained by deleting the first and last rows and columns of the full matrices. This automatically enforces the boundary conditions. Finally, since $X(t,\xi)$ is collocated at Chebyshev nodes, we approximate the inner product appearing in the cost function by Clenshaw-Curtis quadrature [18]:

$$\|X(t,\xi)\|^2_{L^2_{(-1,1)}} = \int_{-1}^{1} |X(t,\xi)|^2 \, d\xi \approx \boldsymbol{w}^T \boldsymbol{x}^2(t),$$

where $\boldsymbol{w} \in \mathbb{R}^n$ are the internal Clenshaw-Curtis quadrature weights and $\boldsymbol{x}^2 \coloneqq \boldsymbol{x} \odot \boldsymbol{x}$. Now the original problem (17) can be reformulated as an ODE-constrained problem,

$$\begin{cases} \min\limits_{u(\cdot)} & \displaystyle\int_t^{t_f} \tilde{L}(\boldsymbol{x}, u) d\tau + \frac{W_2}{2} \boldsymbol{w}^T \boldsymbol{x}^2(t_f), \\ \text{s.t.} & \dot{\boldsymbol{x}} = \boldsymbol{x} \odot \boldsymbol{D}\boldsymbol{x} + \nu \boldsymbol{D}^2 \boldsymbol{x} + \alpha \boldsymbol{x} \odot e^{\beta \boldsymbol{x}} + \boldsymbol{I}_\omega u, \end{cases} \qquad (19)$$

where

$$\tilde{L}(\boldsymbol{x}, u) = \frac{1}{2}\left[\boldsymbol{w}^T \boldsymbol{x}^2(t) + W_1 u^2(t, \boldsymbol{x})\right].$$

A. Learning high-dimensional value functions

The state dimension n of the optimal control problem (19) introduced in the previous section can be adjusted, presenting a good opportunity to test the scalability of our algorithms. We learn the value function $V(t, \boldsymbol{x})$ over the spatial domain

$$\mathcal{X}_0 = \{\, \boldsymbol{x} \in \mathbb{R}^n | -2 \le x_j \le 2, j = 1, 2, \dots, n\}.$$

Using the proposed deep learning framework, we approximate solutions to (19) in $n = 10$, 20, and 30 dimensions. In [7] the infinite-horizon version of the problem is solved up to twelve dimensions, but the accuracy of the solution is not readily verifiable. The ability to easily measure model accuracy for high-dimensional problems with *no known analytical solution* is a key advantage of our approach.

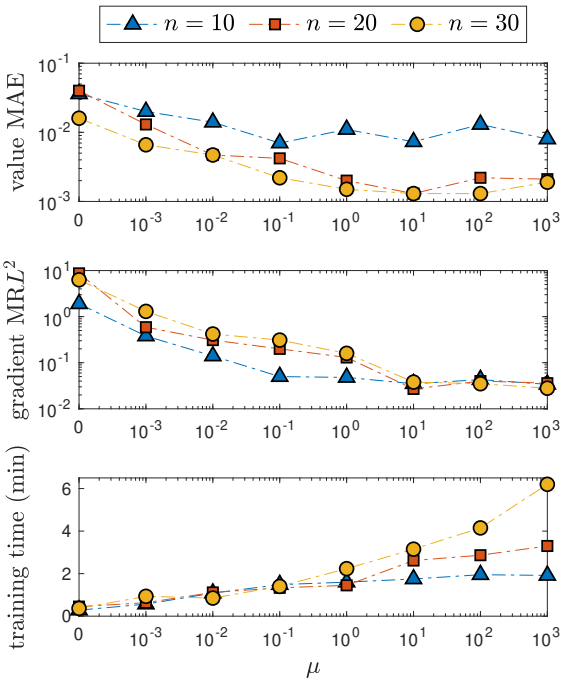


Fig. 1: Validation accuracy and training time of NNs for modeling the time-dependent value function $V(t, \boldsymbol{x})$ of (19). All NNs have the same parameter initialization and are run on an NVIDIA RTX 2080Ti GPU.

For each discretized optimal control problem, $n = 10$, 20, and 30, we apply the time-marching strategy and use the Scipy [19] implementation of the three-stage Lobatto IIIa algorithm in [16] to solve BVPs for 30 uniformly sampled initial conditions, $\boldsymbol{x}_0^{(i)} \in \mathcal{X}_0$, $i = 1, 2, \dots, 30$. For each $\boldsymbol{x}_0^{(i)}$, the BVP solver outputs an optimal trajectory $\boldsymbol{x}^{(i)}\left(t^{(k)}\right)$, where $t^{(k)} \in [0, t_f]$ are collocation points chosen by the solver. Typically this can be a few hundred points per initial condition, depending on the state dimension n and solver tolerances. We use a standard feedforward NN with three layers of 64 neurons each, implemented in TensorFlow [20].

In Fig. 1, we present validation accuracy results for the trained NNs. We include the mean absolute error (MAE) in predicting the value function and the mean relative L^2 error (MRL^2) in predicting the costate, $\boldsymbol{\lambda}(t; \boldsymbol{x}) \approx V_{\boldsymbol{x}}^{NN}(t, \boldsymbol{x})$. Accuracy is measured empirically on independently generated validation data sets comprised of trajectories from 50 randomly selected initial conditions. The results in Fig. 1 show that even in 30 dimensions and with tiny data sets, we are able to train accurate models. This level of accuracy with small data sets is obtained only with physics-informed learning: NNs trained with properly tuned gradient weight μ are orders of magnitude more accurate than those trained by pure regression (11), i.e. with $\mu = 0$. Fig. 1 also shows the training time for each NN, not including time spent generating the initial data. Besides this startup cost, the training time is short and scales reasonably with the problem dimension n and the gradient weight μ. This demonstrates the viability of the proposed method for solving high-dimensional optimal control problems.

TABLE I: Convergence of BVP solutions for (19) when using time-marching. BVP integration time is measured only on successful attempts.

n	K	% BVP convergence	mean integration time
10	4	40%	0.7 s
	6	83%	0.8 s
	10	90%	1.3 s
20	4	46%	3.6 s
	5	86%	4.2 s
	6	99%	4.7 s
30	4	47%	11.3 s
	6	90%	14.6 s
	8	100%	19.1 s

TABLE II: Convergence of BVP solutions for (19) when using NN warm start. BVP integration time is measured only on successful attempts.

n	μ	gradient MRL^2	% BVP conv.	mean int. time
10	0	1.9×10^{0}	96%	0.6 s
	10^{-1}	5.0×10^{-2}	95%	0.5 s
	10^{1}	3.5×10^{-2}	96%	0.6 s
20	0	8.7×10^{0}	63%	2.5 s
	10^{-1}	2.0×10^{-1}	99%	2.4 s
	10^{1}	2.7×10^{-2}	100%	2.3 s
30	0	6.3×10^{0}	89%	7.0 s
	10^{-1}	3.1×10^{-1}	97%	7.0 s
	10^{1}	3.8×10^{-2}	100%	6.8 s

B. Fast BVP solutions with NN warm start

Generating the initial training data set can be the most computationally expensive part of the process, especially as the problem dimension n increases. Consequently, for difficult high-dimensional problems it may be infeasible to generate a large-enough data set from scratch. This obstacle can be largely overcome by using low-fidelity NNs to aid in further data generation. In this section, we briefly compare the reliability and speed of BVP convergence between our two strategies: time-marching and NN warm start.

For each of $n = 10$, 20, and 30, we randomly sample a set of 1000 candidate points from the domain $\mathcal{X}_0$, from which we pick 100 points with the largest predicted value gradient norm. We find that the BVP tends to be more difficult to solve at such points. This set of initial conditions is fixed for each n. First in Table I, we give results for the time-marching trick, depending on the problem dimension, n, and the number of steps in the sequence $\{t_k\}_{k=1}^{K}$. Time-marching is effective once the sequence of time steps is properly tuned, but the speed of this algorithm scales poorly with n.

Next in Table II, we solve the same BVPs directly over the whole time interval $t \in [0, 8]$ with NN warm start. The NNs are trained on the same data set but with different gradient weights, μ, and thus have varying costate prediction accuracy. Now the advantage of utilizing NNs to aid in data generation becomes clear: the average time needed for convergence is drastically lower than with time-marching. Even with low-fidelity NNs, we get good initial guesses which allow us to reliably solve the BVP. Thus quickly training a low-fidelity NN to aid in data generation is the most efficient strategy for building larger data sets.

C. Closed-loop simulations

In this section we show that the feedback control output by the trained NN stabilizes the high-dimensional system and is close to the true optimal control. The optimal feedback control which minimizes (7) is given by

$$u^*(t, \boldsymbol{x}) = -\frac{1}{W_1}[\boldsymbol{I}_\omega]^T V_{\boldsymbol{x}}(t, \boldsymbol{x}), \tag{20}$$

which we approximate with

$$u^{NN}(t, \boldsymbol{x}) = -\frac{1}{W_1}[\boldsymbol{I}_\omega]^T V_{\boldsymbol{x}}^{NN}(t, \boldsymbol{x}). \tag{21}$$

We plot the uncontrolled and closed-loop dynamics in Figs. 2a and 2b, respectively, starting from two different initial conditions, $X(0, \xi) = 2\sin(\pi\xi)$ and $X(0, \xi) = -2\sin(\pi\xi)$. The dimension of the discretized system is $n = 30$. For both of these initial conditions (and almost all others tested), the NN controller successfully stabilizes the open-loop unstable origin. Further, as shown in Fig. 2c, the NN-generated controls are very close to the true optimal controls which are calculated by solving the associated BVPs. Finally, the speed of control computation is fast: independent of n, each evaluation of the control takes just milliseconds on both an NVIDIA RTX 2080Ti GPU and a 2012 MacBook Pro. This feature is essential for real-time feedback.

V. Conclusion

In this paper, we have developed a novel machine learning framework for solving HJB equations and synthesizing optimal feedback controls in real-time. Unlike most other state of the art techniques, our method requires no linearization or restrictions on the structure of the dynamics. The causality-free algorithm we use for data generation enables application to high-dimensional systems, as well as validation of model accuracy. We also emphasize that while our method is data-driven, by leveraging the costate data we are able to learn more physically-consistent models and better controls with surprisingly small data sets.

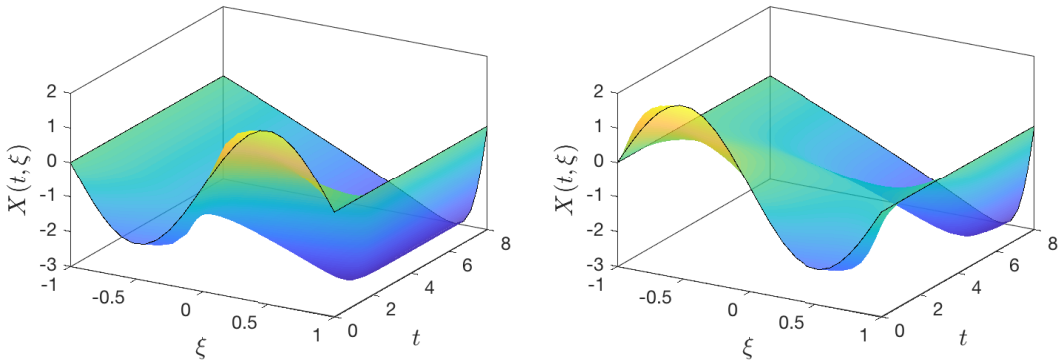
These promising results leave plenty of room for future development. Of special interest are extensions of the framework to solve problems with free final time, state and control constraints, and non-differentiable value functions. These appear ubiquitously in practical applications and present substantial challenges for data generation and NN modeling. Overcoming these obstacles would open the door to solving many important and difficult optimal control problems.

References

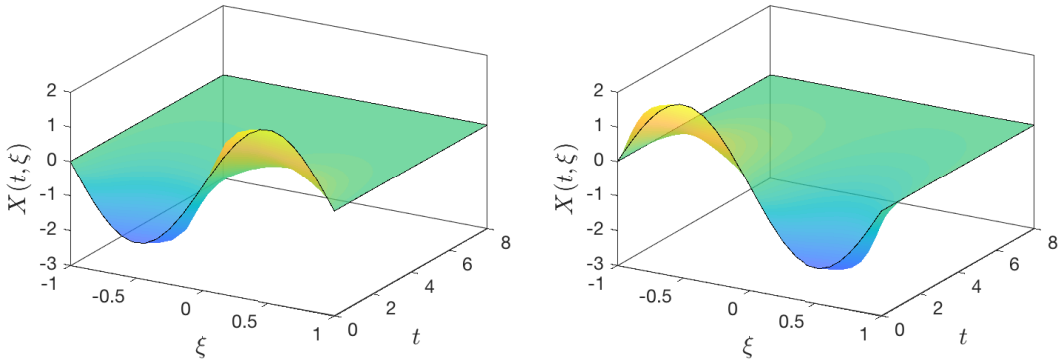
[1] S. Cacace, E. Cristiani, M. Falcone, and A. Picarelli, "A patchy dynamic programming scheme for a class of Hamilton–Jacobi–Bellman equations," *SIAM J. Sci. Comput.*, vol. 34(5), pp. A2625–A2649, 2012.

[2] C. Navasca and A. J. Krener, "Patchy solutions of Hamilton-Jacobi-Bellman partial differential equations," in *Modeling, Estimation and Control*, ser. Lecture Notes in Control and Information Sciences, A. Chiuso, S. Pinzoni, and A. Ferrante, Eds. Springer-Verlag Berlin Heidelberg, 2007, vol. 364, pp. 251–270.

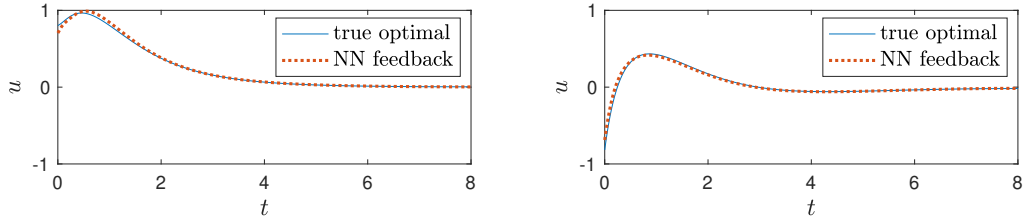
[3] O. Bokanowski, J. Garcke, M. Griebel, and I. Klompmaker, "An adaptive sparse grid semi-Lagrangian scheme for first order Hamilton-Jacobi Bellman equations," *J. Sci. Comput.*, vol. 55, no. 3, pp. 575–605, 2013.



(a) Uncontrolled dynamics.



(b) NN-controlled dynamics.



(c) Comparison of true optimal control (open-loop BVP solution) and NN control profiles.

Fig. 2: Simulations of (18) in $n = 30$ dimensions. Left: $X(0,\xi) = 2\sin(\pi\xi)$. Right: $X(0,\xi) = -2\sin(\pi\xi)$.

[4] M. Falcone and R. Ferretti, *Semi-Lagrangian Approximation Schemes for Linear and Hamilton-Jacobi Equations*. Society for Industrial and Applied Mathematics, 2013.

[5] I. Yegorov and P. M. Dower, "Perspectives on characteristics based curse-of-dimensionality-free numerical approaches for solving Hamilton-Jacobi equations," *Appl. Math. Optim.*, 2018.

[6] J. Darbon and S. Osher, "Algorithms for overcoming the curse of dimensionality for certain Hamilton-Jacobi equations arising in control theory and elsewhere," *Res. Math. Sci.*, vol. 3, no. 1, 2016.

[7] D. Kalise and K. Kunisch, "Polynomial approximation of high-dimensional Hamilton-Jacobi-Bellman equations and applications to feedback control of semilinear parabolic PDEs," *SIAM J. Sci. Comput.*, vol. 40, no. 2, pp. A629–A652, 2018.

[8] W. Kang and L. C. Wilcox, "Mitigating the curse of dimensionality: Sparse grid characteristics method for optimal feedback control and HJB equations," *Comput. Optim. Appl.*, vol. 68, no. 2, pp. 289–315, 2017.

[9] Y. Tassa and T. Erez, "Least squares solutions of the HJB equation with neural network value-function approximators," *IEEE Trans. Neural Netw.*, vol. 18, no. 4, pp. 1031–1041, 2007.

[10] J. Sirignano and K. Spiliopoulos, "DGM: A deep learning algorithm for solving partial differential equations," *J. Comput. Phys.*, vol. 375, pp. 1339 – 1364, 2018.

[11] J. Han, A. Jentzen, and W. E, "Solving high-dimensional partial differential equations using deep learning," *Proceedings of the National Academy of Sciences*, vol. 115, no. 34, pp. 8505–8510, 2018.

[12] D. Izzo, E. Öztürk, and M. Märtens, "Interplanetary transfers via deep representations of the optimal policy and/or of the value function," *arXiv:1904.08809*, 2019.

[13] D. Liberzon, *Calculus of Variations and Optimal Control Theory: A Concise Introduction*. Princeton, NJ, USA: Princeton University Press, 2011.

[14] A. Bressan and B. Piccoli, *Introduction to the Mathematical Theory of Control*. American Institute of Mathematical Sciences, 2007.

[15] R. Hartl, S. Sethi, and R. Vickson, "A survey of the maximum principles for optimal control problems with state constraints," *SIAM Rev.*, vol. 37, no. 2, pp. 181–218, 1995.

[16] J. Kierzenka and L. F. Shampine, "A BVP solver based on residual control and the MATLAB PSE," *ACM Trans. Math. Softw.*, vol. 27, no. 3, pp. 299–316, 2001.

[17] M. Raissi, P. Perdikaris, and G. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *J. Comput. Phys.*, vol. 378, pp. 686–707, 2019.

[18] L. N. Trefethen, *Spectral Methods in MATLAB*. Society for Industrial and Applied Mathematics, 2000.

[19] E. Jones, T. Oliphant, P. Peterson, *et al.*, "SciPy: Open source scientific tools for Python," http://www.scipy.org/, 2001–.

[20] M. Abadi, A. Agarwal, P. Barham, *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," http://www.tensorflow.org/, 2015–.