

UC Santa Cruz

UC Santa Cruz Previously Published Works

Title

Adaptive deep learning for high-dimensional hamilton-jacobi-bellman equations

Permalink

<https://escholarship.org/uc/item/0ws4x49n>

Journal

SIAM Journal on Scientific Computing, 43(2)

ISSN

1064-8275

Authors

NAKAMURA-ZIMMERER, T

GONG, Q

KANG, W

Publication Date

2021

DOI

10.1137/19M1288802

Peer reviewed

ADAPTIVE DEEP LEARNING FOR HIGH-DIMENSIONAL HAMILTON-JACOBI-BELLMAN EQUATIONS*

TENAVI NAKAMURA-ZIMMERER[†], QI GONG[†], AND WEI KANG[‡]

Abstract. Computing optimal feedback controls for nonlinear systems generally requires solving Hamilton-Jacobi-Bellman (HJB) equations, which are notoriously difficult when the state dimension is large. Existing strategies for high-dimensional problems 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 candidate 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 physics of the problem and using the partially-trained NN to aid in adaptive data generation. We demonstrate the effectiveness of our method by learning solutions to HJB equations corresponding to the attitude control of a six-dimensional nonlinear rigid body, and 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 feedback control of these systems.

Key words. Hamilton-Jacobi-Bellman Equations, Optimal Feedback Control, Nonlinear Dynamical Systems, Deep Learning, Neural Networks, Optimization

AMS subject classifications. 49K15, 49L20, 49N35, 68T05, 90C30, 93C15, 93C20

1. 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 solutions for relatively short time horizons in the future. However, for most applications one typically desires a feedback control law, as feedback controls are inherently more robust to disturbances. In principle, optimal feedback controllers can be synthesized 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 large 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. Some key examples include series expansions [3, 28, 23], level set methods [32], patchy dynamic programming [9, 31], semi-Lagrangian methods [5, 17], method of characteristics and Hopf formula-based algorithms [15, 11, 38], and polynomial approximation [22]. These existing methods suffer 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 region; or the system model must have certain special algebraic structure.

In [24, 25], semi-global solutions to HJB equations are computed by combining the method of characteristics with sparse state space discretization. In this approach,

*Submitted to the journal’s Methods and Algorithms for Scientific Computing section September 20, 2019. Preliminary results of this work appeared in the American Control Conference 2020 [30].

Funding: The work of the first and second authors was partially supported with funding from the Defense Advanced Research Projects Agency (DARPA) grant FA8650-18-1-7842.

[†]Department of Applied Mathematics, Baskin School of Engineering, University of California, Santa Cruz (tenakamu@ucsc.edu, qgong@ucsc.edu).

[‡]Department of Applied Mathematics, Naval Postgraduate School, Monterey, CA, (wkang@nps.edu).

41 a two-point boundary value problem (BVP) is solved at each point in a sparse grid.
 42 These BVPs can be solved independently, making the algorithm *causality-free*. This
 43 property is attractive because the computation does not depend on a grid, and hence
 44 they can be applied to high-dimensional problems. The Hopf formula methods [15, 11,
 45 38] also have this property, though it is achieved in a different way and under certain
 46 convexity/concavity assumptions. Causality-free methods are usually too slow for
 47 online computation, but they are perfectly parallelizable so can be used to generate
 48 large data sets offline. Such data sets can then be used to construct faster solutions
 49 such as sparse grid interpolants [24, 25] or, as in this paper, neural networks.

50 Using neural networks (NNs) as a basis for solving HJB equations is not by
 51 itself a new idea, and deep learning approaches have led to promising results; see
 52 for instance [2, 10, 35, 21, 34, 20]. To the best of our knowledge, state-of-the-art
 53 NN-based techniques generally rely on either minimizing the residual of the PDE and
 54 (artificial) boundary conditions at randomly sampled collocation points [2, 10, 35, 34];
 55 or, due to computational limitations, approximating the control and/or HJB solution
 56 and its gradient in a small neighborhood of a nominal trajectory [21, 20]. In [13, 14],
 57 a specialized NN architecture is proposed to solve some classes of Hamilton-Jacobi
 58 equations, but this method has yet to be generalized to state-dependent HJB equations
 59 arising in optimal control. Deep learning techniques have also been proposed for
 60 solving high-dimensional stochastic optimal control problems (see e.g. [18, 19, 4]).

61 In this paper, we develop a computational method for solving high-dimensional
 62 HJB equations and synthesizing candidate optimal feedback controllers. Our approach
 63 is data-driven and consists of three main steps. First, we generate a small set of open-
 64 loop optimal control solutions using a causality-free algorithm based on Pontryagin’s
 65 Minimum Principle (PMP). In the second step, we use the data set to train a NN
 66 to approximate the solution to the HJB equation, called the *value function*. During
 67 training, we supply information about the value function gradient, which encourages
 68 the NN to learn the shape of the value function rather than just fitting point data.
 69 We also estimate the number of samples needed to obtain a good model. Additional
 70 samples are chosen in regions where the value function is difficult to learn, and are
 71 obtained quickly with the aid of the NN. In this sense our method involves *adaptive*
 72 *sampling*. Lastly, the accuracy of the NN is verified on independent data generated
 73 using the same causality free algorithm from the first step. Unlike other NN-based
 74 methods for deterministic HJB equations, our approach does not require computing
 75 expensive PDE residuals and the solution is valid over large spatial domains.

76 As an illustrative example, the method is applied to design an attitude controller
 77 of a rigid-body satellite equipped with momentum wheels. This is a highly nonlinear
 78 problem with $n = 6$ spatial dimensions and $m = 3$ control inputs. With the proposed
 79 method, we obtain a model of the value function with accuracy comparable to that
 80 obtained in [25], but require far fewer sample trajectories to do so. Scalability of
 81 the method is tested on problems of dimension $n = 10, 20$, and 30 arising from pseu-
 82 dospectral discretization of a Burgers’-type PDE. We show that the method is capable
 83 of handling these high-dimensional problems without simplifying the dynamics.

84 Through these examples, we demonstrate several advantages and potential capa-
 85 bilities of the proposed framework. These include solving HJB equations over semi-
 86 global domains with empirically validated levels of accuracy, progressive generation
 87 of rich data sets, and computationally efficient nonlinear feedback control for real-
 88 time applications. Solution of high-dimensional problems is enabled by efficient and
 89 adaptive causality-free data generation, physics-informed learning, and the inherent
 90 capacity of NNs for dealing with high-dimensional data.

91 **1.1. Abbreviations and notation.** Here we present a brief list of some of the
 92 abbreviations, terminology, and notation used in this paper.

	OCP	...	optimal control problem
	HJB	...	Hamilton-Jacobi-Bellman equation
	PMP	...	Pontryagin's Minimum Principle
	NN	...	neural network
93	RMAE	...	relative mean absolute error
	RML ²	...	relative mean L^2 error
	\mathcal{D}	...	data set
	μ	...	gradient regularization weight
	C	...	adaptive sampling convergence parameter

94 **2. A causality-free method for HJB equations.** We consider fixed final
 95 time optimal control problems (OCP) of the form

$$96 \quad (2.1) \quad \begin{cases} \text{minimize} & J[\mathbf{u}(\cdot)] = F(\mathbf{x}(t_f)) + \int_0^{t_f} \mathcal{L}(t, \mathbf{x}, \mathbf{u}) dt, \\ \text{subject to} & \dot{\mathbf{x}}(t) = \mathbf{f}(t, \mathbf{x}, \mathbf{u}), \\ & \mathbf{x}(0) = \mathbf{x}_0. \end{cases}$$

97 Here $\mathbf{x}(t) : [0, t_f] \rightarrow \mathbb{X} \subseteq \mathbb{R}^n$ is the state, $\mathbf{u}(t, \mathbf{x}) : [0, t_f] \times \mathbb{X} \rightarrow \mathbb{U} \subseteq \mathbb{R}^m$ is the control,
 98 and $\mathbf{f}(t, \mathbf{x}, \mathbf{u}) : [0, t_f] \times \mathbb{X} \times \mathbb{U} \rightarrow \mathbb{R}^n$ is a Lipschitz continuous vector field. $J[\mathbf{u}(\cdot)]$ is
 99 the cost functional which is composed of $F(\mathbf{x}(t_f)) : \mathbb{X} \rightarrow \mathbb{R}$, the terminal cost, and
 100 $\mathcal{L}(t, \mathbf{x}, \mathbf{u}) : [0, t_f] \times \mathbb{X} \times \mathbb{U} \rightarrow \mathbb{R}$, the running cost. We assume that the cost functional
 101 is convex in \mathbf{x} and \mathbf{u} . In this paper we consider the case where the final time $t_f < \infty$
 102 is fixed.

103 For a given initial condition $\mathbf{x}(0) = \mathbf{x}_0$, many numerical methods exist to compute
 104 the optimal open-loop solution,

$$105 \quad (2.2) \quad \mathbf{u} = \mathbf{u}^*(t; \mathbf{x}_0).$$

106 The open-loop control (2.2) which solves (2.1) is valid for all $t \in [0, t_f]$, but only
 107 for the fixed initial condition $\mathbf{x}(0) = \mathbf{x}_0$. Due to various sources of disturbance and
 108 real-time application requirements, for practical implementation one typically desires
 109 an optimal control in closed-loop feedback form,

$$110 \quad (2.3) \quad \mathbf{u} = \mathbf{u}^*(t, \mathbf{x}),$$

111 which can be evaluated online given any $t \in [0, t_f]$ and a measurement of $\mathbf{x} \in \mathbb{X}$.

112 To compute the optimal feedback control, we follow the standard procedure in
 113 dynamic programming (see e.g. [27]) and define the value function $V(t, \mathbf{x}) : [0, t_f] \times$
 114 $\mathbb{X} \rightarrow \mathbb{R}$ as the optimal cost-to-go of (2.1) starting at (t, \mathbf{x}) . That is,

$$115 \quad (2.4) \quad V(t, \mathbf{x}) := J[\mathbf{u}^*(\cdot)] = \begin{cases} \inf_{\mathbf{u}(\cdot) \in \mathbb{U}} & F(\mathbf{y}(t_f)) + \int_t^{t_f} \mathcal{L}(\tau, \mathbf{y}, \mathbf{u}) d\tau, \\ \text{s.t.} & \dot{\mathbf{y}}(\tau) = \mathbf{f}(\tau, \mathbf{y}, \mathbf{u}), \\ & \mathbf{y}(t) = \mathbf{x}. \end{cases}$$

116 It can be shown that the value function is the unique viscosity solution [12] of the
 117 Hamilton-Jacobi-Bellman (HJB) PDE,

$$118 \quad (2.5) \quad \begin{cases} -V_t(t, \mathbf{x}) - \min_{\mathbf{u} \in \mathbb{U}} \{ \mathcal{L}(t, \mathbf{x}, \mathbf{u}) + [V_{\mathbf{x}}(t, \mathbf{x})]^T \mathbf{f}(t, \mathbf{x}, \mathbf{u}) \} = 0, \\ V(t_f, \mathbf{x}) = F(\mathbf{x}), \end{cases}$$

119 where we denote $V_t := \partial V / \partial t$ and $V_x := [\partial V / \partial \mathbf{x}]^T$. Note that if the value function is
 120 C^2 , then it is the unique classical solution of (2.5).

121 To compute the control given the value function $V(\cdot)$, we start by defining the
 122 Hamiltonian

$$123 \quad (2.6) \quad \mathcal{H}(t, \mathbf{x}, \boldsymbol{\lambda}, \mathbf{u}) := \mathcal{L}(t, \mathbf{x}, \mathbf{u}) + \boldsymbol{\lambda}^T \mathbf{f}(t, \mathbf{x}, \mathbf{u}),$$

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

$$126 \quad (2.7) \quad \mathbf{u}^*(t) = \mathbf{u}^*(t, \mathbf{x}; \boldsymbol{\lambda}) = \arg \min_{\mathbf{u} \in \mathbb{U}} \mathcal{H}(t, \mathbf{x}, \boldsymbol{\lambda}, \mathbf{u}).$$

127 If we denote the minimized Hamiltonian by $\mathcal{H}^*(t, \mathbf{x}, \boldsymbol{\lambda}) := \mathcal{H}(t, \mathbf{x}, \boldsymbol{\lambda}, \mathbf{u}^*(t, \mathbf{x}; \boldsymbol{\lambda}))$, then
 128 (2.5) can be expressed as

$$129 \quad (2.8) \quad \begin{cases} -V_t(t, \mathbf{x}) - \mathcal{H}^*(t, \mathbf{x}, V_x) = 0, \\ V(t_f, \mathbf{x}) = F(\mathbf{x}). \end{cases}$$

130 If (2.8) can be solved (in the viscosity sense), then it provides both necessary and suf-
 131 ficient conditions for optimality. Moreover, the optimal feedback control is computed
 132 by substituting

$$133 \quad (2.9) \quad \boldsymbol{\lambda}(t) = V_x(t, \mathbf{x})$$

134 into (2.7) to get

$$135 \quad (2.10) \quad \mathbf{u}^*(t, \mathbf{x}) = \mathbf{u}^*(t, \mathbf{x}; V_x) = \arg \min_{\mathbf{u} \in \mathbb{U}} \mathcal{H}(t, \mathbf{x}, V_x, \mathbf{u}).$$

136 This means that with $V_x(\cdot)$ available, the feedback control is obtained as the solution
 137 of an (ideally straightforward) optimization problem.

138 **2.1. Pontryagin's Minimum Principle.** To make use of (2.10), we need an
 139 efficient way to approximate the value function and its gradient. Like [24, 25], rather
 140 than solve the full HJB equation (2.8) on a grid, we exploit the fact that the char-
 141 acteristics of solutions to (2.8) evolve according to a two-point BVP, well-known in
 142 optimal control as Pontryagin's Minimum Principle (PMP):

$$143 \quad (2.11) \quad \begin{cases} \dot{\mathbf{x}}(t) = \mathcal{H}_\lambda = \mathbf{f}(t, \mathbf{x}, \mathbf{u}^*(t, \mathbf{x}; \boldsymbol{\lambda})), & \mathbf{x}(0) = \mathbf{x}_0, \\ \dot{\boldsymbol{\lambda}}(t) = -\mathcal{H}_x(t, \mathbf{x}, \boldsymbol{\lambda}, \mathbf{u}^*(t, \mathbf{x}; \boldsymbol{\lambda})), & \boldsymbol{\lambda}(t_f) = F_x(\mathbf{x}(t_f)), \\ \dot{v}(t) = -\mathcal{L}(t, \mathbf{x}, \mathbf{u}^*(t, \mathbf{x}; \boldsymbol{\lambda})), & v(t_f) = F(\mathbf{x}(t_f)). \end{cases}$$

144 The two-point BVP provides a necessary condition for optimality. If we further assume
 145 that the solution is optimal, then along the characteristic $\mathbf{x}(t; \mathbf{x}_0)$ we have that

$$146 \quad (2.12) \quad \mathbf{u}^*(t, \mathbf{x}) = \mathbf{u}^*(t; \mathbf{x}_0), \quad V(t, \mathbf{x}) = v(t; \mathbf{x}_0), \quad V_x(t, \mathbf{x}) = \boldsymbol{\lambda}(t; \mathbf{x}_0).$$

147 In [24, 25], the two-point BVP (2.11) is solved for each point in a sparse grid.
 148 Applying (2.12), the value function and its gradient are then calculated using high-
 149 dimensional interpolation. This technique is called the sparse grid characteristics
 150 method. But even in a sparse grid the number of points grows like $O(N(\log N)^{n-1})$,
 151 where n is the state dimension and N is the number of grid points in each dimen-
 152 sion. Thus one may have to solve a prohibitively large number of BVPs for higher-
 153 dimensional problems. Instead of sparse grid interpolation, we use data from solved
 154 BVPs to train a NN to approximate the value function. This approach is completely
 155 grid-free and hence applicable in high dimensions.

156 *Remark 2.1.* In general, the BVP admits multiple solutions which can sometimes
 157 be sub-optimal. The characteristics of the value function satisfy (2.11), but there
 158 may be other solutions to these equations which are sub-optimal and therefore not
 159 characteristics of the value function. In many problems it is also possible for the
 160 characteristics to intersect, giving rise to non-smooth value functions and difficulties
 161 in applying (2.9).

162 Optimality of solutions to the BVP can be guaranteed under some convexity
 163 conditions (see e.g. [29]). For most dynamical systems it is difficult to verify such
 164 conditions globally, but we can guarantee optimality locally around an equilibrium
 165 point [28]. Addressing the challenge of global optimality in a broader context is beyond
 166 the scope of the present work, so in this paper we assume that solutions to the two-
 167 point BVP (2.11) are optimal. Under this assumption, the relationship between PMP
 168 and the value function as given in (2.12) holds everywhere.

169 Note the proposed method can still be applied to problems where this assumption
 170 cannot be verified. In such cases PMP remains the prevailing tool for computing
 171 candidate optimal solutions, and from these the proposed method will yield a feedback
 172 controller which satisfies necessary conditions for optimality.

173 **2.2. Causality-free data generation.** While solving the BVP is easier than
 174 solving the full HJB equation, we know of no general algorithm that is reliable and
 175 fast enough for real-time applications. However, in our approach the real-time feed-
 176 back control computation is done by a NN which is trained *offline*. Thus we can solve
 177 the BVP offline to generate data for training and evaluating such a NN. For this pur-
 178 pose, numerically solving the BVP can be manageable although it may require some
 179 parameter tuning. In this paper, we use an implementation of the BVP solver intro-
 180 duced in [26]. This algorithm is based on a three-stage Lobatto IIIa discretization, a
 181 collocation formula which provides a solution that is fourth-order accurate. But the
 182 algorithm is highly sensitive to the initial guess for $\mathbf{x}(t)$ and $\boldsymbol{\lambda}(t)$: there is no guar-
 183 antee of convergence with an arbitrary initial guess, and in most cases a good initial
 184 guess for $\boldsymbol{\lambda}(t)$ cannot be derived from the problem physics. Furthermore, convergence
 185 is increasingly dependent on good initializations as we increase the length of the time
 186 interval.

To overcome this difficulty, we employ the *time-marching* trick from [24, 25].
 This is a continuation technique in which we sequentially extend the solution from an
 initially short time interval to the final time t_f . Specifically, we choose a sequence of
 intermediate times

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

187 in which t_1 is small. For the short time interval $[0, t_1]$, the BVP solver converges given
 188 most initial guesses near the initial state \mathbf{x}_0 . Then the resulting trajectory is rescaled
 189 over the longer time interval $[0, t_2]$. The rescaled trajectory is used as the initial guess
 190 to solve the BVP over $0 \leq t \leq t_2$. We repeat this process until $t_K = t_f$, at which we
 191 obtain the full solution. By appropriately tuning the time sequence $\{t_k\}_{k=1}^K$, we can
 192 largely overcome the problem of sensitivity to initial guesses.

193 Computing many such solutions becomes expensive, which means that generating
 194 the large data sets necessary to train a NN can be difficult. With this in mind, we use
 195 the time-marching trick only to generate a small initial data set, and adaptively adding
 196 more points during training. The key to doing this efficiently is simulating the system
 197 dynamics using the partially-trained NN to close the loop. The closed-loop trajectory
 198 and predicted costate provide good guesses for the optimal state and costate, so that
 199 we can immediately solve (2.11) for all of $[0, t_f]$. Besides being more computationally

200 efficient than time-marching, this approach also requires no parameter tuning. Details
 201 are presented in [section 4](#), and numerical comparisons between this method and time-
 202 marching are given in [subsections 5.2](#) and [6.2](#).

203 As an alternative to either of these two approaches, one could use backpropagati-
 204 on as suggested in [\[20\]](#). However, this method does not allow one to choose initial
 205 conditions independently and so cannot be considered fully causality-free.

206 **3. Neural network approximation of the value function.** Neural networks
 207 have become a popular tool for modeling high-dimensional functions, since they are
 208 not dependent on discretizing the state space. In this paper, we apply NNs to ap-
 209 proximate solutions of the HJB equation and evaluate the resulting feedback control
 210 in real-time. Specifically, we carry out the following steps:

- 211 1. *Initial data generation:* We compute the value function, $V(t, \mathbf{x})$, along tra-
 212 jectories $\mathbf{x}(t)$ from initial conditions chosen by Monte Carlo sampling. Data
 213 is generated by solving the BVP as discussed in [subsection 2.2](#). In this initial
 214 data generation step, we require relatively few data points since more data
 215 can be added later at little computational cost.
- 216 2. *Model training:* Given this data set, we train a NN to approximate the value
 217 function. Learning is guided by the underlying structure of the problem,
 218 specifically by asking the NN to satisfy Eq. [\(2.9\)](#). In doing so, we regularize
 219 the model and make efficient use out of small data sets.
- 220 3. *Adaptive data generation:* In the initial training phase we only have a small
 221 data set, so the NN only roughly approximates the value function. We now
 222 expand the data set by generating data in regions where the value function
 223 is likely to be steep or complicated, and thus difficult to learn. Generating
 224 additional data is made efficient by good initial guesses obtained from NN-
 225 in-the-loop simulations of the system dynamics.
- 226 4. *Model refinement and validation:* We continue training the model and in-
 227 creasing the size of the data set until we satisfy some convergence criteria.
 228 Then, we check the generalization accuracy of the trained NN on a new set
 229 of validation data computed at Monte Carlo sample points.
- 230 5. *Feedback control:* We compute the feedback control online by evaluating the
 231 gradient of the trained NN and applying PMP. Notably, evaluation of the
 232 gradient is *exact* and it is extremely cheap even for large n , enabling real-
 233 time implementation in high-dimensional systems.

234 The crux of the proposed method depends on modeling the value function [\(2.4\)](#)
 235 over a semi-global domain $\mathbb{X} \subset \mathbb{R}^n$. We present details of this process in the following
 236 subsections. In [subsection 3.1](#), we review the basic structure of feedforward NNs and
 237 describe how we train a NN to model the value function. Then in [subsection 3.2](#), we
 238 propose a simple way to incorporate information about the known solution structure
 239 into training. Finally in [subsection 3.4](#), we demonstrate how to use the trained NN
 240 for feedback control. The adaptive data generation scheme is treated separately in
 241 [section 4](#). The proposed method is illustrated in [section 5](#) by solving a practical
 242 optimal attitude control problem for a rigid body satellite, and then applied to solve
 243 larger problems in [section 6](#).

3.1. Feedforward neural networks. In this paper we use multilayer feedfor-
 ward NNs. While many more sophisticated architectures have been developed for
 other 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 $V^{\text{NN}}(\cdot)$ be its NN
 representation. Feedforward NNs approximate complicated nonlinear functions by a

composition of simpler functions, namely

$$V(t, \mathbf{x}) \approx V^{\text{NN}}(t, \mathbf{x}) = g_L \circ \mathbf{g}_{L-1} \circ \cdots \circ \mathbf{g}_\ell \circ \cdots \circ \mathbf{g}_1(t, \mathbf{x}),$$

where each layer $\mathbf{g}_\ell(\cdot)$ is defined as

$$\mathbf{g}_\ell(\mathbf{y}) = \sigma_\ell(\mathbf{W}_\ell \mathbf{y} + \mathbf{b}_\ell).$$

244 Here \mathbf{W}_ℓ and \mathbf{b}_ℓ are the weight matrices and bias vectors, respectively. $\sigma_\ell(\cdot)$ repre-
 245 sents a nonlinear *activation function* applied component-wise to its argument; popular
 246 choices include ReLU, tanh, and other similar functions. In this paper, we use tanh
 247 for all the hidden layers. The final layer, $g_L(\cdot)$, is typically linear, so $\sigma_L(\cdot)$ is the
 248 identity function.

Let $\boldsymbol{\theta}$ denote the collection of the parameters of the NN, i.e.

$$\boldsymbol{\theta} := \{\mathbf{W}_\ell, \mathbf{b}_\ell\}_{\ell=1}^L.$$

The NN is trained by optimizing over the parameters $\boldsymbol{\theta}$ to best approximate $V(t, \mathbf{x})$ by $V^{\text{NN}}(t, \mathbf{x}; \boldsymbol{\theta})$. Specifically, by solving the BVP (2.11) from a set of randomly sampled initial conditions, we get a data set

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

249 where $(t^{(i)}, \mathbf{x}^{(i)})$ are the inputs, $V^{(i)} := V(t^{(i)}, \mathbf{x}^{(i)})$ are the outputs to be modeled,
 250 and $i = 1, 2, \dots, N_d$ are the indices of the data points. In the most naïve setting, the
 251 NN is then trained by solving the nonlinear regression problem,

$$252 \quad (3.1) \quad \underset{\boldsymbol{\theta}}{\text{minimize}} \quad \frac{1}{N_d} \sum_{i=1}^{N_d} \left[V^{(i)} - V^{\text{NN}} \left(t^{(i)}, \mathbf{x}^{(i)}; \boldsymbol{\theta} \right) \right]^2.$$

253 **3.2. Physics-informed machine learning.** Motivated by the development of
 254 physics-informed neural networks [33], we expect that we can improve on the rudimen-
 255 tary loss function in (3.1) by incorporating information about the underlying physics.
 256 In [33], and in particular in the context of HJB equations in [2, 10, 35, 34], the known
 257 underlying PDE and boundary conditions are imposed by minimizing a residual loss
 258 over spatio-temporal collocation points. In this approach, no data is gathered: the
 259 PDE is solved directly in the least-squares sense. But this residual must be evaluated
 260 over a large number of collocation points and can be rather expensive to compute.
 261 Thus we propose a simpler approach of modeling the costate $\boldsymbol{\lambda}(\cdot)$ along with the
 262 value function itself, taking full advantage of the ability to gather data along the
 263 characteristics of the HJB PDE.

Specifically, we know that the costate must satisfy Eq. (2.9), so we train the NN to minimize

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

264 where $V_{\mathbf{x}}^{\text{NN}}(\cdot)$ is the gradient of the NN model with respect to the state. This quantity
 265 is calculated using automatic differentiation. In machine learning, automatic differ-
 266 entiation is usually used to compute gradients with respect to the model parameters,
 267 but is just as easy to apply to computing gradients with respect to inputs. This
 268 gradient is exact, so no finite difference approximations are needed. In addition, the
 269 computational graph is pre-compiled so evaluating the gradient is cheap.

270 Costate data $\boldsymbol{\lambda}(t)$ is obtained for each trajectory as a natural product of solving
 271 the BVP (2.11). Hence we have the augmented data set,

$$272 \quad (3.2) \quad \mathcal{D} = \left\{ \left(t^{(i)}, \mathbf{x}^{(i)} \right), \left(V^{(i)}, \boldsymbol{\lambda}^{(i)} \right) \right\}_{i=1}^{N_d},$$

273 where $\boldsymbol{\lambda}^{(i)} := \boldsymbol{\lambda}(t^{(i)}; \mathbf{x}^{(i)})$. We now define the *physics-informed learning problem*,

$$274 \quad (3.3) \quad \underset{\boldsymbol{\theta}}{\text{minimize}} \quad \text{loss}(\boldsymbol{\theta}; \mathcal{D}) := \underset{V}{\text{loss}}(\boldsymbol{\theta}; \mathcal{D}) + \mu \cdot \underset{\boldsymbol{\lambda}}{\text{loss}}(\boldsymbol{\theta}; \mathcal{D}).$$

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

$$276 \quad (3.4) \quad \underset{V}{\text{loss}}(\boldsymbol{\theta}; \mathcal{D}) := \frac{1}{N_d} \sum_{i=1}^{N_d} \left[V^{(i)} - V^{\text{NN}} \left(t^{(i)}, \mathbf{x}^{(i)}; \boldsymbol{\theta} \right) \right]^2,$$

277 and the gradient regularization is defined as

$$278 \quad (3.5) \quad \underset{\boldsymbol{\lambda}}{\text{loss}}(\boldsymbol{\theta}; \mathcal{D}) := \frac{1}{N_d} \sum_{i=1}^{N_d} \left\| \boldsymbol{\lambda}^{(i)} - V_{\mathbf{x}}^{\text{NN}} \left(t^{(i)}, \mathbf{x}^{(i)}; \boldsymbol{\theta} \right) \right\|^2.$$

279 Following standard practice, when computing the loss functions (3.4) and (3.5), the
 280 output data is linearly scaled to the range $[-1, 1]$ to improve the scaling of the opti-
 281 mization problem.

282 A NN trained to minimize (3.3) learns not just to fit the value data, but it is
 283 rewarded for doing so in a way that respects the underlying structure of the problem.
 284 Gradient regularization takes the known solution structure into account; this makes
 285 it preferable to the usual L^1 or L^2 regularization, which are based on the (heuristic)
 286 principle that simpler representations of data are likely to generalize better. Further-
 287 more, we recall that the optimal control depends explicitly on $V_{\mathbf{x}}(\cdot)$ – see Eqs. (2.10)
 288 and (3.8). Accurate approximation of $V_{\mathbf{x}}(\cdot)$ is therefore essential for calculating opti-
 289 mal controls. Our method achieves this through automatic differentiation to compute
 290 *exact* gradients and by minimization of the gradient loss term (3.5).

291 **3.3. Model validation.** In common practice, one randomly partitions the given
 292 data set (3.2) into a training set $\mathcal{D}_{\text{train}}$ and validation set \mathcal{D}_{val} . During training, the
 293 loss functions (3.4) and (3.5) are calculated with respect to the training data $\mathcal{D}_{\text{train}}$.
 294 We then evaluate the performance of the NN against the validation data \mathcal{D}_{val} , which
 295 it did not observe during training. Good validation performance indicates that the
 296 NN generalizes well, i.e. it did not overfit the training data. We make the validation
 297 test more stringent by generating $\mathcal{D}_{\text{train}}$ and \mathcal{D}_{val} from *independently drawn* initial
 298 conditions, so that the two data sets do not share any part of the same trajectories.

299 We consider the following error metrics for validation. First, the relative mean
 300 absolute error (RMAE) of value function prediction, which is defined as

$$301 \quad (3.6) \quad \text{RMAE}(\boldsymbol{\theta}; \mathcal{D}_{\text{val}}) := \frac{\sum_{i=1}^{N_d} |V^{(i)} - V^{\text{NN}}(t^{(i)}, \mathbf{x}^{(i)}; \boldsymbol{\theta})|}{\sum_{i=1}^{N_d} |V^{(i)}|}.$$

302 We also measure the relative mean L^2 error (RML²) of gradient prediction, which is
 303 defined as

$$304 \quad (3.7) \quad \text{RML}^2(\boldsymbol{\theta}; \mathcal{D}_{\text{val}}) := \frac{\sum_{i=1}^{N_d} \left\| \boldsymbol{\lambda}^{(i)} - V_{\mathbf{x}}^{\text{NN}} \left(t^{(i)}, \mathbf{x}^{(i)}; \boldsymbol{\theta} \right) \right\|_2}{\sum_{i=1}^{N_d} \left\| \boldsymbol{\lambda}^{(i)} \right\|_2}.$$

305 We consider these error metrics instead of pointwise relative errors in order to em-
 306 phasize predictive accuracy in regions where a lot of control effort is needed. This
 307 is important because we are interested in designing nonlinear controllers which are
 308 effective and efficient far away from the equilibrium.

309 **3.4. Neural network in the closed-loop system.** Once the NN is trained,
 310 evaluating $V_{\mathbf{x}}^{\text{NN}}(t, \mathbf{x})$ at new inputs is highly efficient. Moreover, since we minimized
 311 the gradient loss (3.5) during training, we also expect $V_{\mathbf{x}}^{\text{NN}}(t, \mathbf{x})$ to approximate the
 312 true gradient well. At runtime, whenever the feedback control needs to be computed,
 313 we evaluate $V_{\mathbf{x}}^{\text{NN}}(t, \mathbf{x})$ and then solve (2.10) based on this approximation.

For many problems of interest, the optimization problem (2.10) admits an ana-
 lytic or semi-analytic solution. In particular, for the important class of control affine
 systems with running cost convex in \mathbf{u} , we can solve (2.10) analytically. Suppose that
 the system dynamics can be written in the form

$$\dot{\mathbf{x}} = \mathbf{f}(t, \mathbf{x}) + \mathbf{g}(t, \mathbf{x})\mathbf{u},$$

where $\mathbf{f}(t, \mathbf{x}) : [0, t_f] \times \mathbb{X} \rightarrow \mathbb{R}^n$, $\mathbf{g}(t, \mathbf{x}) : [0, t_f] \times \mathbb{X} \rightarrow \mathbb{R}^{n \times m}$, and the control is
 unconstrained. Further, suppose that the running cost is of the form

$$\mathcal{L}(t, \mathbf{x}, \mathbf{u}) = h(t, \mathbf{x}) + \mathbf{u}^T \mathbf{W} \mathbf{u},$$

for some convex function $h(t, \mathbf{x}) : [0, t_f] \times \mathbb{X} \rightarrow \mathbb{R}$ and some positive definite weight
 matrix $\mathbf{W} \in \mathbb{R}^{m \times m}$. Then the Hamiltonian is

$$\mathcal{H}(t, \mathbf{x}, \boldsymbol{\lambda}, \mathbf{u}) = h(t, \mathbf{x}) + \mathbf{u}^T \mathbf{W} \mathbf{u} + \boldsymbol{\lambda}^T \mathbf{f}(t, \mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{g}(t, \mathbf{x})\mathbf{u}.$$

Now we apply PMP, which for unconstrained control requires

$$\mathbf{0}_{m \times 1} = \mathcal{H}_{\mathbf{u}}(t, \mathbf{x}, \boldsymbol{\lambda}, \mathbf{u}^*) = 2\mathbf{W}\mathbf{u}^* + \mathbf{g}^T(t, \mathbf{x})\boldsymbol{\lambda}.$$

314 Letting $\boldsymbol{\lambda} = V_{\mathbf{x}}(t, \mathbf{x})$ and solving for \mathbf{u}^* yields the optimal feedback control law in
 315 explicit form:

$$316 \quad (3.8) \quad \mathbf{u}^*(t, \mathbf{x}; V_{\mathbf{x}}) = -\frac{1}{2}\mathbf{W}^{-1}\mathbf{g}^T(t, \mathbf{x})V_{\mathbf{x}}(t, \mathbf{x}).$$

317 The resulting NN controller is then simply

$$318 \quad (3.9) \quad \mathbf{u}^{\text{NN}}(t, \mathbf{x}) = \mathbf{u}^*(t, \mathbf{x}; V_{\mathbf{x}}^{\text{NN}}) = -\frac{1}{2}\mathbf{W}^{-1}\mathbf{g}^T(t, \mathbf{x})V_{\mathbf{x}}^{\text{NN}}(t, \mathbf{x}).$$

319 **4. Adaptive sampling and model refinement.** Since generating just a single
 320 data point requires solving a challenging BVP, it can be expensive to generate large
 321 data sets which adequately represent the value function. This necessitates training
 322 using limited data and a method to generate new data in a smart and efficient way.
 323 In this paper, effective training with small data sets is accomplished by incorporating
 324 information about the costate as discussed in subsection 3.2, but also by combining
 325 progressive data generation with an efficient adaptive sampling technique.

326 Optimization methods in machine learning (see e.g. [6] for a comprehensive sur-
 327 vey) are typically divided into second and first order methods. Second order methods
 328 like L-BFGS [8] rely on accurate gradient computations, and hence generally have to
 329 use the entire data set. For this reason they are often referred to as batch or full-batch
 330 methods. On the other hand, first order methods based on stochastic gradient descent

(SGD) use only small subsets, or mini-batches, of the full data set. That is, at each optimization iteration k , the loss functions in (3.1) and (3.3) are evaluated only on a subset $\mathcal{S}_k \subset \mathcal{D}_{\text{train}}$ with $|\mathcal{S}_k| \ll |\mathcal{D}_{\text{train}}|$. Here $|\mathcal{D}|$ denotes the number of data points in a data set \mathcal{D} . Although second order methods converge much more quickly than first order methods, the necessary gradient calculations are prohibitively expensive for large data sets. Consequently, SGD variants have become the de facto standard for machine learning applications.

But in the context of deep learning, our NNs are small and data sets smaller. Thus we expect second order methods to be superior for our purposes. With a small initial data set, which we denote by $\mathcal{D}_{\text{train}}^1$, we find that training a low-fidelity model is very fast using L-BFGS. After this initial round, we want to increase the size of the data set so that it better captures the features of the value function. We then continue training the model using this larger data set, $\mathcal{D}_{\text{train}}^2$. We continue this process until some convergence conditions are satisfied.

Our approach is similar to and inspired by a progressive batching method proposed in [7]. The primary difference is that the problem addressed in [7] is a standard machine learning problem, where a massive data set is available from the start. This allows one to increase the sample size every few iterations, and take a completely different sample from the available data. In our problem, start with only a small amount of data and we can generate more as we go, but since data generation is expensive, we would like to generate only as much as is needed.

4.1. Convergence test and sample size selection. In this section we derive a convergence test and sample size selection scheme for the purpose of progressive data generation. To start, suppose that the internal optimizer (e.g. L-BFGS) converges in optimization round r and let $\mathcal{D}_{\text{train}}^r$ be the available training data set. Given convergence of the internal optimizer, the first order necessary condition for optimality holds, so

$$(4.1) \quad \left\| \frac{\partial \text{loss}}{\partial \boldsymbol{\theta}} (\boldsymbol{\theta}; \mathcal{D}_{\text{train}}^r) \right\| \ll 1.$$

Here $\text{loss}(\cdot)$ is the physics-informed loss defined in Eq. (3.3), and $\frac{\partial \text{loss}}{\partial \boldsymbol{\theta}}(\cdot)$ is its gradient with respect to the NN parameters $\boldsymbol{\theta}$. For true first order optimality, we would like the gradient to be small when evaluated over the entire continuous domain of interest, $[0, t_f] \times \mathbb{X}$. In other words, we want

$$(4.2) \quad \left\| \frac{\partial \text{loss}}{\partial \boldsymbol{\theta}} (\boldsymbol{\theta}; [0, t_f] \times \mathbb{X}) \right\| \ll 1,$$

where the Monte Carlo sums in Eqs. (3.4) and (3.5) become integrals in the limit as the size of the data set approaches infinity.

The simplest way to see if (4.2) holds is to generate a validation data set \mathcal{D}_{val} . Then using the fact that $\frac{\partial \text{loss}}{\partial \boldsymbol{\theta}} (\boldsymbol{\theta}; \mathcal{D}_{\text{val}}) \rightarrow \frac{\partial \text{loss}}{\partial \boldsymbol{\theta}} (\boldsymbol{\theta}; [0, t_f] \times \mathbb{X})$ in the limit as $|\mathcal{D}_{\text{val}}| \rightarrow \infty$, one checks if, for example,

$$(4.3) \quad \left\| \frac{\partial \text{loss}}{\partial \boldsymbol{\theta}} (\boldsymbol{\theta}; \mathcal{D}_{\text{val}}) \right\| < \epsilon,$$

for some small parameter $\epsilon > 0$. Convergence tests like (4.3) are standard in machine learning and are useful for testing generalization performance. But for many practical problems, it may be too expensive to generate enough validation data to make the

373 test meaningful. More importantly, such tests provides no clear guidance in selecting
 374 the sample size $|\mathcal{D}_{\text{train}}^{r+1}|$ should they not be satisfied.

375 In this paper, we use validation tests to quantify model accuracy after training is
 376 complete (see [subsection 3.3](#)). Indeed, the ability to empirically validate solutions is a
 377 key benefit of the causality-free approach. For the purpose of determining convergence
 378 between training rounds, however, we propose a different statistically motivated test
 379 which provides information on choosing $|\mathcal{D}_{\text{train}}^{r+1}|$. The idea is simple: since we already
 380 assume [\(4.1\)](#) holds, then to ensure that [\(4.2\)](#) is also satisfied, it suffices to check that
 381 the error in approximating [\(4.2\)](#) by [\(4.1\)](#) is relatively small.

382 To motivate this more rigorously, consider a finite sample set $\mathcal{D} \subset [0, t_f] \times \mathbb{X}$ with
 383 fixed size $|\mathcal{D}|$, and assume that the sample points $(t^{(i)}, \mathbf{x}^{(i)}) \in \mathcal{D}$ are independent
 384 and identically distributed (i.i.d.)¹. By design, if $(t^{(i)}, \mathbf{x}^{(i)})$ are i.i.d. then the sample
 385 gradient $\frac{\partial \text{loss}}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}; \mathcal{D})$ is an unbiased estimator for the true gradient (evaluated over the
 386 entire continuous domain). That is,

$$387 \quad (4.4) \quad \mathbb{E}_{\mathcal{D}} \left[\frac{\partial \text{loss}}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}; \mathcal{D}) \right] = \frac{\partial \text{loss}}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}; [0, t_f] \times \mathbb{X}),$$

388 where $\mathbb{E}_{\mathcal{D}}[\cdot] := \mathbb{E}_{\mathcal{D} \subset [0, t_f] \times \mathbb{X}}[\cdot]$ denotes the population mean over all possible finite
 389 sample sets $\mathcal{D} \subset [0, t_f] \times \mathbb{X}$ with fixed size $|\mathcal{D}|$. Intuitively, [\(4.4\)](#) implies that if [\(4.1\)](#)
 390 holds, then on average we also have [\(4.2\)](#), as desired. But we must control the mean
 391 square error (MSE) of the estimator, which is given by

$$392 \quad (4.5) \quad \text{MSE} \left[\frac{\partial \text{loss}}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}; \mathcal{D}) \right] := \mathbb{E}_{\mathcal{D}} \left[\left\| \frac{\partial \text{loss}}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}; \mathcal{D}) - \frac{\partial \text{loss}}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}; [0, t_f] \times \mathbb{X}) \right\|^2 \right]$$

$$393 \quad = \mathbb{E}_{\mathcal{D}} \left[\sum_{j=1}^{|\boldsymbol{\theta}|} \left(\frac{\partial \text{loss}}{\partial \theta_j}(\boldsymbol{\theta}; \mathcal{D}) - \frac{\partial \text{loss}}{\partial \theta_j}(\boldsymbol{\theta}; [0, t_f] \times \mathbb{X}) \right)^2 \right].$$

394

To simplify this, using linearity of the expectation we obtain

$$\text{MSE} \left[\frac{\partial \text{loss}}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}; \mathcal{D}) \right] = \sum_{j=1}^{|\boldsymbol{\theta}|} \text{Var}_{\mathcal{D} \subset [0, t_f] \times \mathbb{X}} \left[\frac{\partial \text{loss}}{\partial \theta_j}(\boldsymbol{\theta}; \mathcal{D}) \right],$$

and then by construction of the loss function,

$$\text{MSE} \left[\frac{\partial \text{loss}}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}; \mathcal{D}) \right] = \sum_{j=1}^{|\boldsymbol{\theta}|} \text{Var}_{\mathcal{D} \subset [0, t_f] \times \mathbb{X}} \left[\frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \frac{\partial \text{loss}}{\partial \theta_j}(\boldsymbol{\theta}; (t^{(i)}, \mathbf{x}^{(i)})) \right].$$

¹In practice, while initial conditions are i.i.d., points at future times lie along the optimal trajectories coming from these initial conditions and are thus spatially correlated. Adaptive sampling (see [subsection 4.2](#)) also introduces sample dependence. This likely reduces sample variance compared to i.i.d. data, but we still find the numerical tests useful for providing sample size guidelines. In addition, if we learn only the initial-time value function as in [section 5](#), then sample independence can be upheld if we forego adaptive sample placement.

395 Using the simplifying assumption that $(t^{(i)}, \mathbf{x}^{(i)})$ are i.i.d., this becomes

$$\begin{aligned}
 396 \quad \text{MSE} \left[\frac{\partial \text{loss}}{\partial \boldsymbol{\theta}} (\boldsymbol{\theta}; \mathcal{D}) \right] &= \frac{1}{|\mathcal{D}|^2} \sum_{j=1}^{|\boldsymbol{\theta}|} \sum_{i=1}^{|\mathcal{D}|} \text{Var}_{(t, \mathbf{x}) \in [0, t_f] \times \mathbb{X}} \left[\frac{\partial \text{loss}}{\partial \theta_j} (\boldsymbol{\theta}; (t, \mathbf{x})) \right] \\
 397 \quad (4.6) \quad &= \frac{1}{|\mathcal{D}|} \sum_{j=1}^{|\boldsymbol{\theta}|} \text{Var}_{(t, \mathbf{x}) \in [0, t_f] \times \mathbb{X}} \left[\frac{\partial \text{loss}}{\partial \theta_j} (\boldsymbol{\theta}; (t, \mathbf{x})) \right]. \\
 398
 \end{aligned}$$

399 If the estimation error is small, then the sample mean is likely to be a good approx-
 400 imation of the true mean. Hence we expect that $\left\| \frac{\partial \text{loss}}{\partial \boldsymbol{\theta}} (\boldsymbol{\theta}; [0, t_f] \times \mathbb{X}) \right\|$ will also be
 401 small as desired. To this end, we require that the root MSE not be too large compared
 402 to the expected gradient. Specifically, we check if

$$403 \quad (4.7) \quad \sqrt{\text{MSE} \left[\frac{\partial \text{loss}}{\partial \boldsymbol{\theta}} (\boldsymbol{\theta}; \mathcal{D}) \right]} \leq C \left\| \mathbb{E}_{\mathcal{D}} \left[\frac{\partial \text{loss}}{\partial \boldsymbol{\theta}} (\boldsymbol{\theta}; \mathcal{D}) \right] \right\|_1,$$

404 where $C > 0$ is a scalar parameter. On the right hand side we use the L^1 norm instead
 405 of the L^2 as it is less sensitive to outliers in the loss gradient. In practice we find that
 406 this makes the test less likely to suggest unreasonably large sample sizes.

In practice, evaluating of the true population variances on the left hand side of
 407 (4.7) is computationally intractable. But we can approximate these by the corre-
 408 sponding *sample* variances² taken over all data $(t^{(i)}, \mathbf{x}^{(i)}) \in \mathcal{D}_{\text{train}}^r$, which we denote
 409 by $\text{Var}_{\mathcal{D}_{\text{train}}^r}[\cdot] := \text{Var}_{(t^{(i)}, \mathbf{x}^{(i)}) \in \mathcal{D}_{\text{train}}^r}[\cdot]$:

$$\text{MSE} \left[\frac{\partial \text{loss}}{\partial \boldsymbol{\theta}} (\boldsymbol{\theta}; \mathcal{D}) \right] \approx \frac{1}{|\mathcal{D}_{\text{train}}^r|} \sum_{j=1}^{|\boldsymbol{\theta}|} \text{Var}_{\mathcal{D}_{\text{train}}^r} \left[\frac{\partial \text{loss}}{\partial \theta_j} (\boldsymbol{\theta}; (t^{(i)}, \mathbf{x}^{(i)})) \right].$$

407 Similarly, we approximate the expected gradient on the right hand side of (4.7) by
 408 the sample gradient and arrive at the following practical convergence criterion:

$$409 \quad (4.8) \quad \sqrt{\sum_{j=1}^{|\boldsymbol{\theta}|} \text{Var}_{\mathcal{D}_{\text{train}}^r} \left[\frac{\partial \text{loss}}{\partial \theta_j} (\boldsymbol{\theta}; (t^{(i)}, \mathbf{x}^{(i)})) \right]} \leq C \left\| \frac{\partial \text{loss}}{\partial \boldsymbol{\theta}} (\boldsymbol{\theta}; \mathcal{D}_{\text{train}}^r) \right\|_1 \sqrt{|\mathcal{D}_{\text{train}}^r|}.$$

If the convergence test (4.8) is satisfied, then it is likely that the expected gradient
 $\left\| \frac{\partial \text{loss}}{\partial \boldsymbol{\theta}} (\boldsymbol{\theta}; [0, t_f] \times \mathbb{X}) \right\|$ is also small. In other words, we expect that the parameters $\boldsymbol{\theta}$
 satisfies the first order optimality conditions evaluated over the entire domain, so we
 can stop optimization. Satisfaction of (4.8) does not imply that the trained model is
 good – merely that seeing more data would probably not improve it significantly. On
 the other hand, when the criterion is not met, then it guides us in selecting the next
 sample size $|\mathcal{D}_{\text{train}}^{r+1}|$. Concretely, suppose that the ratio of the sample variance to the
 sample gradient doesn't change significantly by increasing the size of the data set, i.e.

$$\frac{\sqrt{\sum_{j=1}^{|\boldsymbol{\theta}|} \text{Var}_{\mathcal{D}_{\text{train}}^{r+1}} \left[\frac{\partial \text{loss}}{\partial \theta_j} (\boldsymbol{\theta}; (t^{(i)}, \mathbf{x}^{(i)})) \right]}}{\left\| \frac{\partial \text{loss}}{\partial \boldsymbol{\theta}} (\boldsymbol{\theta}; \mathcal{D}_{\text{train}}^{r+1}) \right\|_1} \approx \frac{\sqrt{\sum_{j=1}^{|\boldsymbol{\theta}|} \text{Var}_{\mathcal{D}_{\text{train}}^r} \left[\frac{\partial \text{loss}}{\partial \theta_j} (\boldsymbol{\theta}; (t^{(i)}, \mathbf{x}^{(i)})) \right]}}{\left\| \frac{\partial \text{loss}}{\partial \boldsymbol{\theta}} (\boldsymbol{\theta}; \mathcal{D}_{\text{train}}^r) \right\|_1}.$$

²Computing a large number of individual gradients can still be too costly, so we often evaluate
 sample variances over a smaller subset of the training data.

410 Then the appropriate choice of $|\mathcal{D}_{\text{train}}^{r+1}|$ to satisfy (4.8) after the next round is such
 411 that

$$412 \quad (4.9) \quad M |\mathcal{D}_{\text{train}}^r| \geq |\mathcal{D}_{\text{train}}^{r+1}| \geq \frac{\sum_{j=1}^{|\theta|} \text{Var}_{\mathcal{D}_{\text{train}}^r} \left[\frac{\partial \text{loss}}{\partial \theta_j} (\boldsymbol{\theta}; (t^{(i)}, \mathbf{x}^{(i)})) \right]}{\left(C \left\| \frac{\partial \text{loss}}{\partial \boldsymbol{\theta}} (\boldsymbol{\theta}; \mathcal{D}_{\text{train}}^r) \right\|_1 \right)^2},$$

413 where $M > 1$ is a scalar parameter which prevents the data set size from growing too
 414 quickly. Throughout this paper we use $M = 2$.

415 The convergence test (4.8) and sample size selection scheme (4.9) derived above
 416 are close to that used in [7], except that we employ the L^1 norm of the sample
 417 gradient in the denominator instead of the L^2 norm. We prefer the L^1 norm because
 418 it is less sensitive to outliers in the loss gradient. Intuitively, this improves robustness
 419 by making the test less likely to suggest unreasonably large sample sizes. We also
 420 contribute a different derivation, coming from the perspective of progressive data
 421 generation as opposed to sampling from a large pre-existing data set. Finally, like [7]
 422 our results are not specific to learning solutions to the HJB equation. They can be
 423 applied to many data-driven optimization problems where data is scarce but can be
 424 generated over time. Notably, these results facilitate the use of existing algorithms
 425 for second order and constrained optimization in such applications.

426 **4.2. Adaptive data generation with NN warm start.** The sample size
 427 selection criterion (4.9) we propose indicates how many data are necessary to satisfy
 428 the convergence test (4.8), assuming a uniform sampling from the domain. In practice,
 429 since all the data we generate will be new, we can choose to generate new data where
 430 it is needed most, hence the term *adaptive sampling*. This condition for generating
 431 new data can be interpreted in many ways. In this paper, we concentrate samples
 432 where $\|V_{\mathbf{x}}^{\text{NN}}(\cdot)\|$ is large. Regions of the value function with large gradients tend to
 433 be steep or complicated, and thus may benefit from having more data to learn from.
 434 Furthermore, these regions correspond to places where the control effort is large and
 435 hence we would like controllers to be especially accurate there.

Specifically, for each initial condition we want to integrate, we can first randomly
 sample a set of N_c candidate initial conditions from \mathbb{X} . A quick pass through the NN
 yields the predicted gradient at all candidate points:

$$\left\{ V_{\mathbf{x}}^{\text{NN}} \left(0, \mathbf{x}_0^{(i)} \right) \right\}_{i=1}^{N_c}.$$

436 We then choose the point(s) with the largest predicted gradient norms and solve the
 437 BVP (2.11) for each of these. To aid in solving these BVPs, instead of using the time-
 438 marching trick described in subsection 2.2, we simulate the system dynamics using the
 439 partially-trained NN as the closed-loop controller and predicting $\boldsymbol{\lambda}(t) \approx V_{\mathbf{x}}^{\text{NN}}(t, \mathbf{x}(t))$
 440 along the trajectory. In most cases, this yields an approximate solution which is
 441 reasonably close to the optimal state and costate. By supplying this trajectory as
 442 an initial guess to the BVP solver, we then quickly and reliably obtain a solution
 443 to the BVP for the full time interval $[0, t_f]$. This process is repeated for new initial
 444 conditions until we obtain the desired amount of data (each trajectory may contain
 445 hundreds of data points). We refer to this technique as a *NN warm start*. A summary
 446 of the full training procedure is given in Algorithm 4.1.

447 Algorithm 4.1 enables us to build up a rich data set and a high-fidelity model of
 448 $V(\cdot)$. Moreover, the data set is not constrained to lie within a small neighborhood
 449 of some nominal trajectory. It can contain points from the entire domain \mathbb{X} , and we

Algorithm 4.1 Adaptive sampling and model refinement

```

1: Generate  $\mathcal{D}_{\text{train}}^1$  using time-marching
2: for  $r = 1, 2, \dots$  do
3:   Solve (3.3) for  $\theta$ 
4:   if (4.8) is satisfied then
5:     return optimized parameters  $\theta$  and NN validation accuracy
6:   else
7:     while (4.9) is not satisfied do
8:       Sample candidate initial conditions  $\mathbf{x}_0^{(i)}$ ,  $i = 1, \dots, N_c$ , from  $\mathbb{X}$ 
9:       In parallel, predict  $\|V_{\mathbf{x}}^{\text{NN}}(0, \mathbf{x}_0^{(i)})\|$ ,  $i = 1, \dots, N_c$ 
10:      Choose the initial condition(s) with largest predicted gradient norm and
      use NN warm start to solve the BVP (2.11)
11:      Add the resulting trajectories to  $\mathcal{D}_{\text{train}}^{r+1}$ 
12:    end while
13:  end if
14: end for

```

450 can concentrate more data near complicated features of the value function. As we
451 progressively refine the NN model, we can adjust the gradient loss weight μ , as well
452 as other hyperparameters such as the internal optimizer convergence tolerance and
453 the number of terms in the L-BFGS Hessian approximation. As the NN is already
454 partially-trained, fewer iterations should be needed for convergence in each round so
455 we can afford to make each iteration more expensive.

456 **5. Application to rigid body attitude control.** To illustrate the capabilities
457 of proposed method, we consider the six-state rigid body model of a satellite studied
458 by Kang and Wilcox [24, 25]. With the sparse grid characteristics method, they
459 interpolate the value function at initial time, $V(t = 0, \mathbf{x})$, and use this for moving
460 horizon feedback control of the nonlinear system. We use their successful results as a
461 baseline for evaluating our method.

Let $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$ be an inertial frame of orthonormal vectors and let $\{\mathbf{e}'_1, \mathbf{e}'_2, \mathbf{e}'_3\}$
be a body frame. The state of the satellite is then written as $\mathbf{x} = (\mathbf{v} \ \boldsymbol{\omega})$. Here \mathbf{v} is
the attitude of the satellite represented in Euler angles,

$$\mathbf{v} = (\phi \ \theta \ \psi)^T,$$

in which ϕ , θ , and ψ are the angles of rotation around \mathbf{e}'_1 , \mathbf{e}'_2 , and \mathbf{e}'_3 , respectively,
in the order (1, 2, 3). These are also commonly called roll, pitch, and yaw. $\boldsymbol{\omega}$ denotes
the angular velocity in the body frame,

$$\boldsymbol{\omega} = (\omega_1 \ \omega_2 \ \omega_3)^T.$$

For details see [16]. The state dynamics are

$$\begin{pmatrix} \dot{\mathbf{v}} \\ \mathbf{J}\dot{\boldsymbol{\omega}} \end{pmatrix} = \begin{pmatrix} \mathbf{E}(\mathbf{v})\boldsymbol{\omega} \\ \mathbf{S}(\boldsymbol{\omega})\mathbf{R}(\mathbf{v})\mathbf{h} + \mathbf{B}\mathbf{u} \end{pmatrix}.$$

Here $\mathbf{E}(\mathbf{v}), \mathbf{S}(\boldsymbol{\omega}), \mathbf{R}(\mathbf{v}) : \mathbb{R}^3 \rightarrow \mathbb{R}^{3 \times 3}$ are matrix-valued functions defined as

$$\mathbf{E}(\mathbf{v}) := \begin{pmatrix} 1 & \sin \phi \tan \theta & \cos \phi \tan \theta \\ 0 & \cos \phi & -\sin \phi \\ 0 & \sin \phi / \cos \theta & \cos \phi / \cos \theta \end{pmatrix}, \quad \mathbf{S}(\boldsymbol{\omega}) := \begin{pmatrix} 0 & \omega_3 & -\omega_2 \\ -\omega_3 & 0 & \omega_1 \\ \omega_2 & -\omega_1 & 0 \end{pmatrix},$$

and

$$\mathbf{R}(\mathbf{v}) := \begin{pmatrix} \cos \theta \cos \psi & \cos \theta \sin \psi & -\sin \theta \\ \sin \phi \sin \theta \cos \psi - \cos \phi \sin \psi & \sin \phi \sin \theta \sin \psi + \cos \phi \cos \psi & \cos \theta \sin \phi \\ \cos \phi \sin \theta \cos \psi + \sin \phi \sin \psi & \cos \phi \sin \theta \sin \psi - \sin \phi \cos \psi & \cos \theta \cos \phi \end{pmatrix}.$$

Further, $\mathbf{J} \in \mathbb{R}^{3 \times 3}$ is a combination of the inertia matrices of the momentum wheels and the rigid body without wheels, $\mathbf{h} \in \mathbb{R}^3$ is the total constant angular momentum of the system, and $\mathbf{B} \in \mathbb{R}^{3 \times m}$ is a constant matrix where m is the number of momentum wheels. To control the system, we apply a torque $\mathbf{u}(t, \mathbf{v}, \boldsymbol{\omega}) : [0, t_f] \times \mathbb{R}^3 \times \mathbb{R}^3 \rightarrow \mathbb{R}^m$.

We consider the fully-actuated case where $m = 3$. Let

$$\mathbf{B} = \begin{pmatrix} 1 & 1/20 & 1/10 \\ 1/15 & 1 & 1/10 \\ 1/10 & 1/15 & 1 \end{pmatrix}, \quad \mathbf{J} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 4 \end{pmatrix}, \quad \mathbf{h} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

The optimal control problem is

$$(5.1) \quad \begin{cases} \text{minimize}_{\mathbf{u}(\cdot)} & J[\mathbf{u}(\cdot)] = \int_t^{t_f} \mathcal{L}(\mathbf{v}, \boldsymbol{\omega}, \mathbf{u}) d\tau + \frac{W_4}{2} \|\mathbf{v}(t_f)\|^2 + \frac{W_5}{2} \|\boldsymbol{\omega}(t_f)\|^2, \\ \text{subject to} & \dot{\mathbf{v}} = \mathbf{E}(\mathbf{v})\boldsymbol{\omega}, \\ & \mathbf{J}\dot{\boldsymbol{\omega}} = \mathbf{S}(\boldsymbol{\omega})\mathbf{R}(\mathbf{v})\mathbf{h} + \mathbf{B}\mathbf{u}. \end{cases}$$

Here

$$\mathcal{L}(\mathbf{v}, \boldsymbol{\omega}, \mathbf{u}) = \frac{W_1}{2} \|\mathbf{v}\|^2 + \frac{W_2}{2} \|\boldsymbol{\omega}\|^2 + \frac{W_3}{2} \|\mathbf{u}\|^2$$

and

$$W_1 = 1, \quad W_2 = 10, \quad W_3 = \frac{1}{2}, \quad W_4 = 1, \quad W_5 = 1, \quad t_f = 20.$$

Finally, we consider initial conditions in the domain

$$(5.2) \quad \mathbb{X}_0 = \left\{ \mathbf{v}, \boldsymbol{\omega} \in \mathbb{R}^3 \mid -\frac{\pi}{3} \leq \phi, \theta, \psi \leq \frac{\pi}{3} \text{ and } -\frac{\pi}{4} \leq \omega_1, \omega_2, \omega_3 \leq \frac{\pi}{4} \right\}.$$

In [25], to avoid discretizing time the value function is approximated only at initial time $t = 0$. In order to facilitate a fair comparison we do the same. This means that we model $V(0, \mathbf{v}, \boldsymbol{\omega}) \approx V^{\text{NN}}(\mathbf{v}, \boldsymbol{\omega})$, i.e. the NN does *not* take time as an input variable. Consequently the control is implemented with a time-independent moving horizon rather than as a time-dependent optimal control. In other words, at each time t when we evaluate the control, we assume $t = 0$ and return $\mathbf{u}(t) = \mathbf{u}^{\text{NN}}(\mathbf{v}(t), \boldsymbol{\omega}(t))$. Controlling the system using moving horizon feedback is standard practice. It is also reasonable for the present case because the problem dynamics are time-invariant and the time horizon is relatively long. Because of this we observe near-optimal performance from the moving horizon controller.

5.1. Learning the value function. In this section, we present numerical results of our implementation of a NN for modeling the initial-time value function of the rigid body attitude control problem (5.1). To obtain data, we uniformly sample initial conditions $(\mathbf{v}^{(i)}, \boldsymbol{\omega}^{(i)})$ from the domain \mathbb{X}_0 defined in (5.2), and for each initial value, solve the two-point BVP (2.11) using time-marching and the SciPy [37] implementation of the three-stage Lobatto IIIa algorithm in [26]. Each integrated trajectory contains around 100 data points on average, but we use only initial time data, $V(0, \mathbf{v}^{(i)}, \boldsymbol{\omega}^{(i)})$. For validation, we generate a data set containing $|\mathcal{D}_{\text{val}}| = 1000$

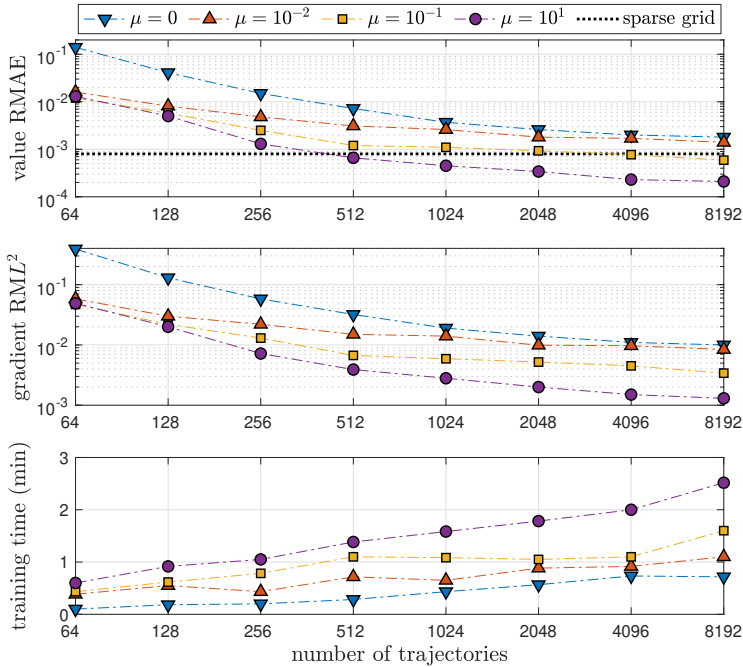


Figure 1: Validation accuracy and training time of NNs for modeling the initial time value function $V(0, \mathbf{v}, \boldsymbol{\omega})$ of the optimal attitude control problem (5.1). All NNs have the same parameter initialization and are run on an NVIDIA RTX 2080Ti GPU.

488 data points (at $t = 0$), and keep this fixed throughout all the tests. As a baseline,
 489 the sparse grid characteristics method with $|G_{\text{sparse}}^{13}| = 44,698$ grid points achieves a
 490 RMAE of 8.00×10^{-4} on this validation data set.

491 We implement a standard feedforward NN in TensorFlow 1.11 [1] and train it to
 492 approximate $V(0, \mathbf{v}, \boldsymbol{\omega})$. The NN has three hidden layers with 64 neurons in each, but
 493 many alternate configurations of depth and width also work. For optimization, we
 494 use the SciPy interface for the L-BFGS optimizer [37, 8]. Figure 1 displays the results
 495 of a series of tests in which we vary the weight μ on the value gradient loss term (3.5)
 496 and the size of the training data set. Results are compared to those obtained in [25].

497 We highlight that with just 512 data points, we can train NNs with better accu-
 498 racy than the sparse grid characteristics method with $|G_{\text{sparse}}^{13}| = 44,698$ points. Thus
 499 for this problem, the proposed method is about 90 times as data-efficient. With 8192
 500 data points, the NN can be almost four times as accurate as the sparse grid charac-
 501 teristics method. This level of accuracy with small data sets is obtained only with
 502 physics-informed learning. In particular, NNs trained by pure regression (3.1) cannot
 503 match the accuracy of the sparse grid characteristics method, as shown in Figure 1
 504 for the case with $\mu = 0$. Accuracy improves as we increase μ but with diminishing
 505 returns for $\mu \geq 10$. While physics-informed learning is more costly, it permits the use
 506 of much smaller data sets, and the increased training time is still quite short.

507 **5.2. Training with adaptive data generation.** Performing a thorough sys-
 508 tematic study of the adaptive sampling and model refinement technique proposed in

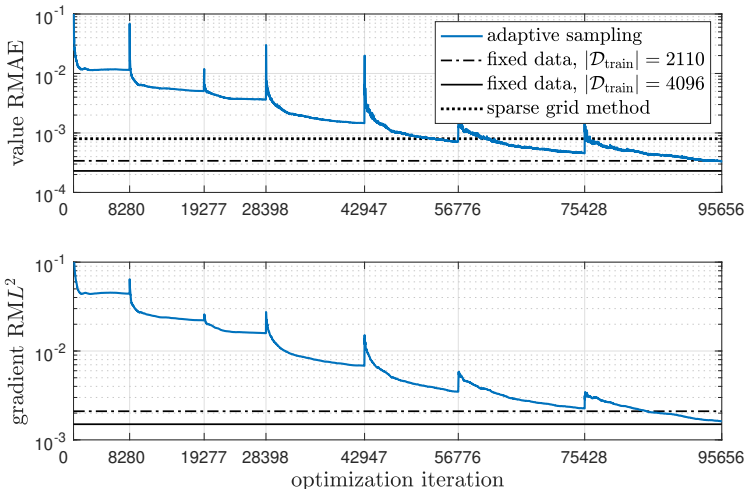


Figure 2: Progress of adaptive sampling and model refinement for the rigid body problem (5.1), compared to training on fixed data sets and the sparse grid characteristics method. Spikes in the error correspond to the start of new training rounds and expansion of the training data set.

509 section 4 is rather complicated, since a successful implementation depends on various
 510 hyperparameter settings, which can and perhaps should change each optimization
 511 round. Results also depend on random chance, since data points are chosen in a
 512 (partially) random way and the randomly-initialized NN training problem is highly
 513 non-convex. For this reason, in this section we show a just few conservative results
 514 which we feel illustrate the potential of the method.

515 Figure 2 shows the progress of the validation error during training when using
 516 adaptive sampling starting from a data set with $|\mathcal{D}_{\text{train}}^1| = 64$ points. We set the
 517 gradient loss weight to $\mu = 10$ and the convergence parameter in (4.8) to $C = 0.25$.
 518 After each round, we check the convergence criterion (4.8) and increase the number
 519 of training data according to (4.9). Each data set includes all previously generated
 520 data, and we generate additional data as needed through Algorithm 4.1. With these
 521 configurations, the model passes the convergence test after seven training rounds and
 522 observing a total of $|\mathcal{D}_{\text{train}}^7| = 2110$ samples.

523 The final value function accuracy is 3.3×10^{-4} : over twice as accurate as the
 524 sparse grid method with about twenty times fewer data, and the gradient prediction
 525 accuracy is 1.6×10^{-3} . As shown in Figure 2, the gradient predictions of the network
 526 trained using the adaptive algorithm are just as accurate as a network trained on a
 527 fixed data set of $|\mathcal{D}_{\text{train}}| = 4096$ samples. That is to say, the adaptive sampling method
 528 facilitates more accurate gradient predictions using fewer data. These results highlight
 529 the main advantages of the adaptive sampling and model refinement method: the
 530 ability to overcome an initial lack of data, efficiently generate a large data set, and
 531 improve gradient prediction accuracy which is needed for effective control. To fully
 532 realize the potential of the method, hyperparameters like μ , C , and internal optimizer
 533 parameters need to be adjusted in each round. Development of algorithms to do this
 534 adaptively remains a topic for future research.

K	% BVP convergence	mean integration time
1	0.3%	0.37 s
2	38.7%	0.44 s
3	76.2%	0.40 s
4	92.9%	0.45 s
8	98.4%	0.53 s

Table 1: Convergence of BVP solutions for (5.1) when using the time-marching trick, depending on the number of steps in the sequence $\{t_k\}_{k=1}^K$. The case $K = 1$ corresponds to a direct solution attempt over the whole time interval with no time-marching. BVP integration time is measured only on successful attempts – failed solution attempts usually take much longer.

μ	training time	gradient RML^2	% BVP conv.	mean int. time
10^{-8}	7 s	2.5×10^{-1}	88.0%	0.50 s
10^{-4}	19 s	1.4×10^{-1}	98.6%	0.48 s
1	23 s	4.5×10^{-2}	99.7%	0.44 s

Table 2: Convergence of BVP solutions for (5.1) when using NN warm start with NNs of varying gradient prediction accuracy. BVP integration time is measured only on successful attempts.

535 Next, we investigate the convergence of the BVP solver with time-marching and
 536 NN warm start. Results are given in Table 1 and Table 2, respectively. For these tests,
 537 we use 1000 initial conditions with the largest predicted gradient norm, $\|V_{\mathbf{x}}^{\text{NN}}(\cdot)\|$,
 538 picked from a set of 10^6 randomly sampled candidate points. Initial conditions with
 539 large gradient norm tend to be located in regions where the value function is steep or
 540 complicated, and may thus be more difficult to solve. The set of initial conditions is
 541 fixed for all tests.

542 In the first row of Table 1, we attempt to solve the BVP with no time-marching,
 543 i.e. over the entire time interval without constructing any initial guess. In this case,
 544 the proportion of convergent solutions is extremely small, obviating the need for good
 545 initial guesses. As shown in Table 1, we reliably obtain solutions for this problem
 546 when we use at least $K = 4$ time intervals. We note that the initial conditions
 547 are purposefully chosen to be difficult – if we simply take uniform samples from the
 548 domain \mathbb{X}_0 , the proportion of convergent solutions increases significantly.

549 In Table 2, we present results using NN warm start. We train several NNs on a
 550 data set of only 64 points. Because the data set is so small, *each NN takes only seconds*
 551 *to finish training*. We also experiment with using different gradient loss weights μ for
 552 each NN. This directly impacts the accuracy in predicting the initial-time costate,
 553 $\boldsymbol{\lambda}(0; \mathbf{v}_0, \boldsymbol{\omega}_0) \approx V_{\mathbf{x}}^{\text{NN}}(\mathbf{v}_0, \boldsymbol{\omega}_0)$, which in turn is key to synthesizing optimal controls.

554 Even with these low-fidelity models, the rate of BVP convergence is just as high
 555 as when using $K = 4$ time intervals for time-marching. The quality of initial guesses
 556 improves with better costate prediction, and it is not difficult to exceed 99% conver-
 557 gence. For this problem, the speed of the two methods is about the same. However,
 558 when we consider higher-dimensional problems in subsection 6.2, we find that NN
 559 warm start significantly improves both reliability and efficiency.

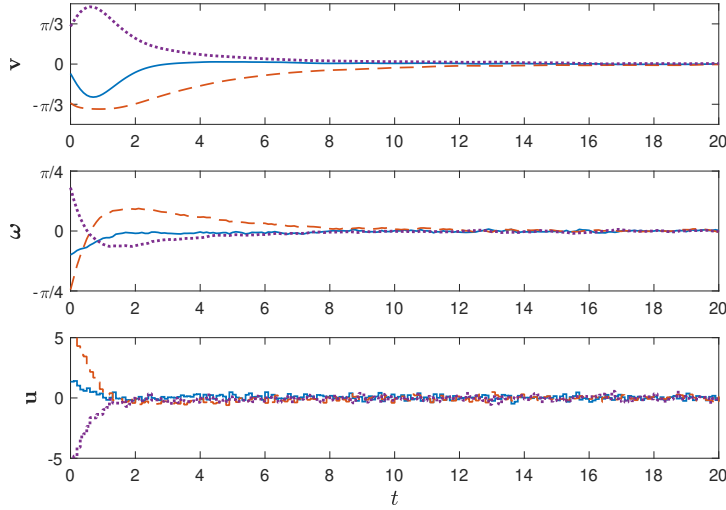


Figure 3: Sample closed-loop trajectories of the rigid body system controlled by a NN feedback controller implemented with a zero-order-hold and subject to measurement noise. Solid: ϕ , ω_1 , and u_1 . Dashed: θ , ω_2 , and u_2 . Dotted: ψ , ω_3 , and u_3 .

560 **5.3. Closed-loop simulation.** In this section we perform numerical simulations
 561 of the rigid body dynamics, demonstrating that the NN feedback controller is capable
 562 of stabilizing the system. Using (3.9) we calculate the optimal feedback control law

$$563 \quad (5.3) \quad \mathbf{u}^{\text{NN}}(\mathbf{v}, \boldsymbol{\omega}) = -\frac{1}{W_3} [\mathbf{J}^{-1} \mathbf{B}]^T V_{\boldsymbol{\omega}}^{\text{NN}}(\mathbf{v}, \boldsymbol{\omega}).$$

564 Recall that because we are using a time-independent value function model, the control
 565 is implemented as time-independent moving horizon feedback. Since \mathbf{J} and \mathbf{B} are
 566 constant matrices, we pre-compute the product $-\mathbf{J}^{-1} \mathbf{B}]^T / W_3$. Hence evaluation
 567 of the control requires only a forward pass through the computational graph of $V_{\boldsymbol{\omega}}^{\text{NN}}(\cdot)$
 568 and a matrix multiplication.

In Figure 3, we plot a typical closed-loop trajectory starting from a randomly sampled initial condition. To make the simulation more realistic, we implement the controller using a zero-order-hold with a sample rate of 10 [Hz]. In addition, we corrupt inputs to the controller with Gaussian white noise with standard deviation $\sigma = 0.01\pi$. That is, for all $t \in [t_k, t_k + 0.1]$, we apply the control

$$\mathbf{u}(t) = \mathbf{u}^{\text{NN}}(\hat{\mathbf{v}}(t_k), \hat{\boldsymbol{\omega}}(t_k)),$$

where

$$\begin{pmatrix} \hat{\mathbf{v}}(t_k) \\ \hat{\boldsymbol{\omega}}(t_k) \end{pmatrix} := \begin{pmatrix} \mathbf{v}(t_k) \\ \boldsymbol{\omega}(t_k) \end{pmatrix} + \mathbf{n}(t_k), \quad \mathbf{n}(t_k) \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}).$$

569 In spite of this, the NN controller successfully stabilizes the system. Furthermore, the
 570 total cost of the closed-loop trajectory is $J[\mathbf{u}^{\text{NN}}(\cdot)] = 12.67$, about 1% more than the
 571 optimal cost $J[\mathbf{u}^*(\cdot)] = 12.52$. For comparison, a linear quadratic regulator (LQR)
 572 for (5.1) accumulates a total cost of $J[\mathbf{u}^{\text{LQR}}(\cdot)] = 15.95$, which is 27% more than

573 the optimal cost. Finally, short computation time is critical for implementation in
 574 real systems, and this is achieved here as each evaluation of the control takes only a
 575 couple milliseconds on both an NVIDIA RTX 2080Ti GPU and a 2012 MacBook Pro.

576 **6. Application to control of Burgers'-type PDE.** In this section, we test
 577 our method on high-dimensional nonlinear systems arising from a Chebyshev pseu-
 578 dospectral (PS) discretization of a one-dimensional forced Burgers'-type PDE. An
 579 infinite-horizon version of this problem is studied in [22], in which the value function
 580 is approximated using a polynomial Galerkin technique. We note that in [22], separa-
 581 bility of the nonlinear dynamics is required to compute the high-dimensional integrals
 582 necessary in the Galerkin formulation. Our method does not require this restriction,
 583 although it does apply to this problem.

584 As in [22], let $X(t, \xi) : [0, t_f] \times [-1, 1] \rightarrow \mathbb{R}$ satisfy the following one-dimensional
 585 controlled PDE with Dirichlet boundary conditions:

$$586 \quad (6.1) \quad \begin{cases} X_t = XX_\xi + \nu X_{\xi\xi} + \alpha X e^{\beta X} + I_\Omega(\xi)u, & t > 0, \xi \in (-1, 1), \\ X(t, -1) = X(t, 1) = 0, & t > 0, \\ X(0, \xi) = X_0, & \xi \in (-1, 1). \end{cases}$$

For notational convenience we have written $X = X(t, \xi)$, and as before 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

$$I_\Omega(\xi) := \begin{cases} 1, & \xi \in \Omega, \\ 0, & \xi \notin \Omega. \end{cases}$$

587 The PDE-constrained optimal control problem is

$$588 \quad (6.2) \quad \begin{cases} \underset{u(\cdot)}{\text{minimize}} & J[u(\cdot)] = \int_t^{t_f} \mathcal{L}(X, u) d\tau + \frac{W_2}{2} \|X(t_f, \xi)\|_{L^2_{(-1,1)}}^2, \\ \text{subject to} & X_t = XX_\xi + \nu X_{\xi\xi} + \alpha X e^{\beta X} + I_\Omega(\xi)u, \\ & X(\tau, -1) = X(\tau, 1) = 0. \end{cases}$$

Here

$$\|X(\tau, \xi)\|_{L^2_{(-1,1)}}^2 := \int_{-1}^1 X^2(\tau, \xi) d\xi, \quad \mathcal{L}(X, u) = \frac{1}{2} \|X(\tau, \xi)\|_{L^2_{(-1,1)}}^2 + \frac{W_1}{2} u^2(\tau, X),$$

and we set

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

589 In this problem, the goal of stabilizing $X(t, \xi)$ is made more challenging by the added
 590 reaction term, $\alpha X e^{\beta X}$, which renders the origin unstable. This can be seen clearly in
 591 [Figure 4a](#).

To solve (6.2) using our framework, we perform Chebyshev PS collocation to
 transform the PDE (6.1) into a system of ordinary differential equations (ODEs).
 Following [36], let

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

where $N_c + 1$ is the number of collocation points. After accounting for boundary
 conditions, we collocate $X(t, \xi)$ at internal (non-boundary) Chebyshev points, ξ_j ,
 $j = 1, 2, \dots, n$, where $n = N_c - 1$. The discretized state is defined as

$$\mathbf{x}(t) := (X(t, \xi_1), X(t, \xi_2), \dots, X(t, \xi_n))^T : [0, t_f] \rightarrow \mathbb{R}^n,$$

and the PDE (6.1) becomes a system of ODEs in n dimensions:

$$\dot{\mathbf{x}} = \mathbf{x} \odot \mathbf{D}\mathbf{x} + \nu \mathbf{D}^2\mathbf{x} + \alpha \mathbf{x} \odot e^{\beta \mathbf{x}} + \mathbb{I}_\Omega u,$$

In the above, “ \odot ” denotes element-wise multiplication (Hadamard product), \mathbb{I}_Ω is the discretized indicator function, and $\mathbf{D}, \mathbf{D}^2 \in \mathbb{R}^{n \times n}$ are the internal parts of the first and second order Chebyshev differentiation matrices, which are obtained by deleting the first and last rows and columns of the full matrices. This discretization automatically enforces the boundary conditions. Finally, since $X(t, \xi)$ is collocated at Chebyshev nodes, the inner product appearing in the cost function is conveniently approximated by Clenshaw-Curtis quadrature [36]:

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

592 where $\mathbf{w} \in \mathbb{R}^n$ are the internal Clenshaw-Curtis quadrature weights and $\mathbf{x}^2(t)$ is calcu-
593 lated element-wise, i.e. $\mathbf{x}^2 = \mathbf{x} \odot \mathbf{x}$. Now the original OCP (6.2) can be reformulated
594 as an ODE-constrained problem,

$$595 \quad (6.3) \quad \begin{cases} \underset{u(\cdot)}{\text{minimize}} & \int_t^{t_f} \frac{1}{2} [\mathbf{w}^T \mathbf{x}^2(\tau) + W_1 u^2(\tau, \mathbf{x})] d\tau + \frac{W_2}{2} \mathbf{w}^T \mathbf{x}^2(t_f), \\ \text{subject to} & \dot{\mathbf{x}} = \mathbf{x} \odot \mathbf{D}\mathbf{x} + \nu \mathbf{D}^2\mathbf{x} + \alpha \mathbf{x} \odot e^{\beta \mathbf{x}} + \mathbb{I}_\Omega u. \end{cases}$$

596 **6.1. Learning high-dimensional value functions.** The state dimension n of
597 the OCP (6.3) can be adjusted, presenting a good opportunity to test the scalability
598 of our algorithms. For this problem, we learn the value function $V = V(t, \mathbf{x})$ with
599 time-dependence, rather than just $V(0, \mathbf{x})$ as in section 5. Consequently, the resulting
600 controls can be implemented as time-dependent controls or with a moving horizon.
601 We consider the following domain of initial conditions:

$$602 \quad (6.4) \quad \mathbb{X}_0 = \{\mathbf{x} \in \mathbb{R}^n \mid -2 \leq x_j \leq 2, j = 1, 2, \dots, n\}.$$

603 Using the proposed adaptive deep learning framework, we approximate solutions
604 to (6.3) in $n = 10, 20,$ and 30 dimensions. We focus on demonstrating what is possible
605 using our approach, rather than carrying out a detailed study of its effectiveness under
606 different parameter tunings. In [22] an infinite-horizon version of the problem is solved
607 up to twelve dimensions, but the accuracy of the solution is not readily verifiable. The
608 ability to conveniently measure model accuracy for general high-dimensional problems
609 with *no known analytical solution* is a key advantage of our framework.

610 For each discretized OCP, $n = 10, 20,$ and $30,$ we apply the time-marching strat-
611 egy to build an initial training data set $\mathcal{D}_{\text{train}}^1$ from 30 uniformly sampled initial
612 conditions, $\mathbf{x}_0^{(i)} \in \mathbb{X}_0, i = 1, 2, \dots, 30.$ For each initial condition $\mathbf{x}_0^{(i)},$ the BVP solver
613 outputs an optimal trajectory $\{\mathbf{x}^{(i)}(t_k)\},$ evaluated at collocation points $t_k \in [0, t_f]$
614 chosen by the solver. Typically this can be a few hundred per initial condition, de-
615 pending on the state dimension n and the BVP solver tolerances. Since these data sets
616 can be get quite large, we often train on randomly selected subsets of the data. This
617 can significantly improve training speed without sacrificing accuracy. When needed,
618 we solve additional BVPs to expand the data set as described in subsection 4.2. We
619 use the same NN architecture as in section 5, with three hidden layers with 64 neurons
620 each. We set $C = 0.3, 1.3,$ and 1.8 for $n = 10, 20,$ and $30,$ respectively, and use $\mu = 10$
621 in all cases.

n	num. trajectories	training time	value RMAE	gradient RML^2
10	163	25 min	1.3×10^{-3}	5.7×10^{-3}
20	128	48 min	5.0×10^{-3}	1.1×10^{-2}
30	145	62 min	1.3×10^{-2}	2.2×10^{-2}

Table 3: Validation accuracy of NNs for solving the collocated Burgers'-type OCP (6.3), depending on the state dimension n . Training time includes time spent generating additional data according to Algorithm 4.1. All NNs are trained on an NVIDIA RTX 2080Ti GPU.

622 In Table 3, we present validation accuracy results for the trained NNs. We include the RMAE in predicting the value function and the RML^2 error in predicting the costate, $\lambda(t; \mathbf{x}_0) \approx V_{\mathbf{x}}^{\text{NN}}(t, \mathbf{x}(t; \mathbf{x}_0))$. Accuracy is measured empirically on independently generated validation data sets comprised of trajectories from 50 randomly selected initial conditions. We find that the trained NNs accurately predict both the value function and its gradient, even in 30 dimensions.

628 Table 3 also shows the total number of sample trajectories seen by the NN, including the initial data $\mathcal{D}_{\text{train}}^1$. It may seem surprising that we are able to reach the same level of accuracy in higher dimensions with similar numbers of sample trajectories. This happens because the BVP solver usually needs more collocation points for larger problems, thus producing more data per trajectory. Consequently, fewer trajectories are needed to fulfill the data set size recommendation (4.9). Similarly, in section 5 we use data only for $t = 0$, so we need thousands of trajectories to fill in the state space and train the NN. This suggests that learning the time-dependent value function can be more efficient than learning $V(0, \mathbf{x})$ only. Note that, if preferred, the time-dependent controller can still be used with a moving horizon like in section 5. Such an implementation can be useful in the presence of noise.

639 Lastly, Table 3 shows the training time for each NN, including time spent testing convergence and generating additional trajectories on the fly, but not time spent generating the initial data. Generating data becomes the most expensive computation as n increases, but even so we find that computational effort scales reasonably with the problem dimension. Furthermore, it is possible to obtain a rough low-fidelity NN model in just minutes as shown in Table 5, which in turn allows for more efficient data generation. This demonstrates the viability of the proposed method for solving high-dimensional optimal control problems.

647 **6.2. NN warm start for fast and reliable BVP solutions.** In our experience, generating the initial training data set can be the most computationally demanding part of the process, especially as the problem dimension n increases. Consequently, for difficult high-dimensional problems it may be impractical to generate a large-enough data set from scratch. This obstacle can be largely overcome by using partially-trained/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. These experiments demonstrate the importance of NN guesses for high-dimensional data generation.

656 For each of $n = 10, 20$, and 30 , we randomly sample a set of 1000 candidate points from the domain \mathbb{X}_0 defined in (6.4). From these we choose 100 points with the largest predicted value gradient. The set of initial conditions is fixed for each n . Next we

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 4: Convergence of BVP solutions for (6.3) when using the time-marching trick, depending on the problem dimension, n , and the number of steps in the sequence $\{t_k\}_{k=1}^K$. BVP integration time is measured only on successful attempts.

n	μ	training time	gradient RML ²	% BVP conv.	mean int. time
10	10^{-8}	20 s	2.1×10^{-1}	96%	0.8 s
	10^{-4}	31 s	9.8×10^{-2}	99%	0.8 s
	1	56 s	4.2×10^{-2}	88%	0.6 s
20	10^{-8}	29 s	3.7×10^{-1}	74%	2.9 s
	10^{-4}	47 s	9.8×10^{-2}	91%	2.5 s
	1	76 s	6.5×10^{-2}	98%	2.5 s
30	10^{-8}	38 s	3.0×10^{-1}	79%	7.1 s
	10^{-4}	125 s	7.6×10^{-2}	94%	6.9 s
	1	189 s	7.4×10^{-2}	96%	7.1 s

Table 5: Convergence of BVP solutions for (6.3) when using NN warm start with NNs of varying gradient prediction accuracy. BVP integration time is measured only on successful attempts.

659 proceed as in subsection 5.2, solving each BVP by time-marching with various K .
 660 Results are summarized in Table 4. We then solve the same BVPs directly over the
 661 whole time interval $t \in [0, 8]$ with NN warm start. These NNs are trained on fixed
 662 data sets containing only 30 trajectories, but with different gradient loss weights μ ,
 663 resulting in varying costate prediction accuracy. We also limit the number of L-BFGS
 664 iterations so that each model is trained only for a short time. Results are given in
 665 Table 5.

666 As before, we find that even NNs with relatively large costate prediction error
 667 enable consistently convergent BVP solutions. Time-marching also works once the
 668 sequence of time steps $\{t_k\}_{k=1}^K$ is properly tuned, but the speed of this method scales
 669 poorly with n . Now the advantage of utilizing NNs to aid in data generation becomes
 670 clear: when n is large, the average time needed for convergence when using NN warm
 671 start is drastically lower than that of the time-marching trick. This approach also
 672 requires no tuning of the time-marching sequence. Because low-fidelity NNs are quick
 673 to train, training such a NN and then using it to aid in data generation is the most
 674 efficient strategy for building larger data sets.

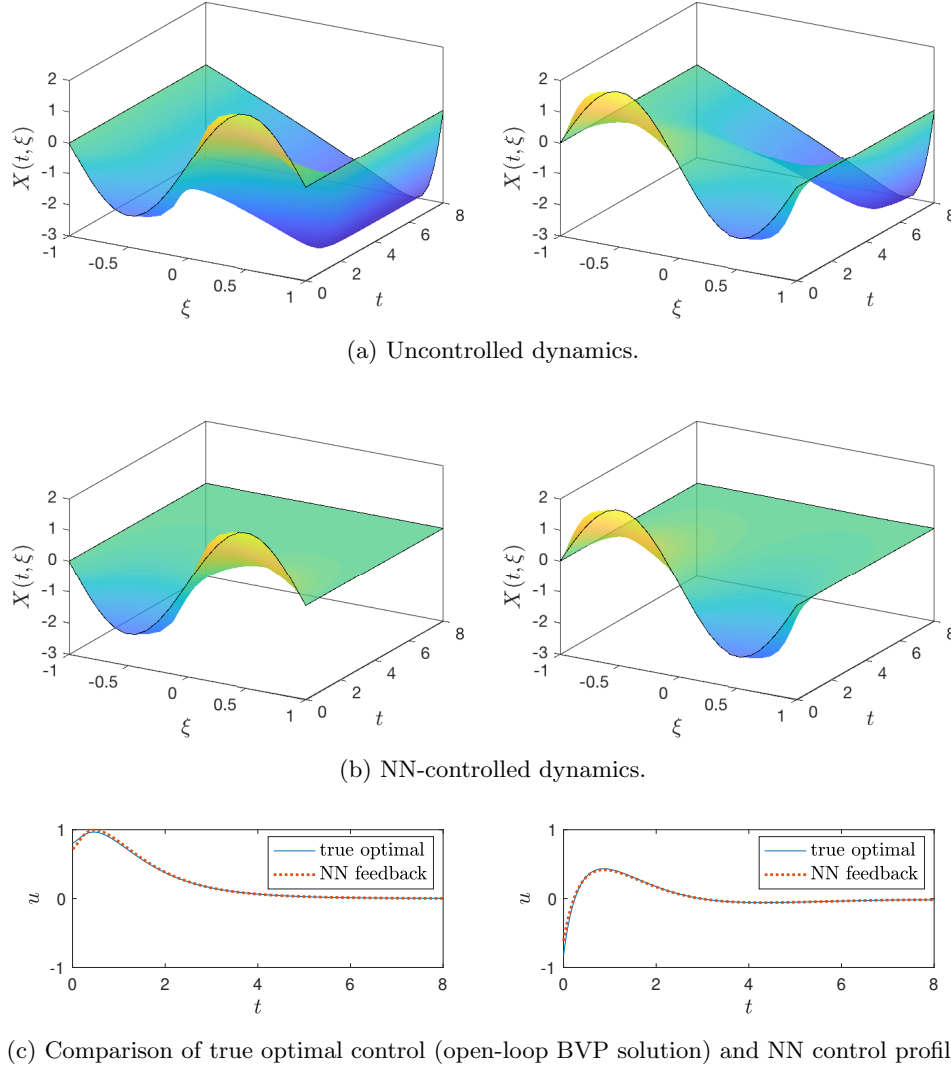


Figure 4: Simulations of the collocated Burgers'-type PDE (6.1) in $n = 30$ dimensions. Left column: $X(0, \xi) = 2 \sin(\pi\xi)$. Right column: $X(0, \xi) = -2 \sin(\pi\xi)$.

675 **6.3. Closed-loop simulations.** In this section we use simulations to demon-
 676 strate that the feedback control output by the trained NN not only stabilizes the
 677 high-dimensional system, but that it is close to the true optimal. The optimal
 678 feedback control law can again be calculated with (3.9), from which we obtain

$$679 \quad (6.5) \quad u^{\text{NN}}(t, \mathbf{x}) = -\frac{1}{W_1} \mathbb{I}_{\Omega}^T V_{\mathbf{x}}^{\text{NN}}(t, \mathbf{x}).$$

680 In Figure 4, we plot the uncontrolled (Figure 4a) and closed-loop controlled dy-
 681 namics (Figure 4b), starting from two different initial conditions, $X(0, \xi) = 2 \sin(\pi\xi)$

682 and $X(0, \xi) = -2 \sin(\pi\xi)$, where the dimension of the discretized system is $n = 30$.
 683 For both of these initial conditions (and almost all others tested), the NN controller
 684 successfully stabilizes the open-loop unstable origin. Further, as shown in [Figure 4c](#),
 685 the NN-generated controls are very close to the true optimal controls which are calcu-
 686 lated by solving the associated BVPs. Finally, the speed of online control computation
 687 is not sensitive to the problem dimension: each evaluation still takes just milliseconds
 688 on both an NVIDIA RTX 2080Ti GPU and a 2012 MacBook Pro.

689 **7. Conclusion.** In this paper, we have developed a novel machine learning
 690 framework for solving HJB equations and designing candidate optimal feedback con-
 691 trollers. Unlike many other state of the art techniques, our method does not require
 692 finite difference approximations of the gradient nor strict restrictions on the structure
 693 of the dynamics. The causality-free algorithm we use for data generation enables
 694 application to high-dimensional systems and validation of model accuracy. We also
 695 emphasize that while our method is data-driven, by leveraging the costate data we
 696 are able to train more physically-consistent models and better controllers with sur-
 697 prisingly small data sets.

698 The proposed method is not only a consumer of data, but through adaptive data
 699 generation it can also be used build rich data sets with points anywhere in a semi-
 700 global domain. Thus the value function and control are valid for large ranges of
 701 dynamic states, rather than just in the neighborhood of some nominal trajectory.
 702 Furthermore, data can be generated near complicated or non-smooth regions of the
 703 value function to aid in learning. This in turn allows us to train more accurate NN
 704 models or employ other data-driven methods.

705 We have demonstrated the possibility for use of the framework in a practical
 706 setting by synthesizing candidate optimal feedback controls of a six-dimensional non-
 707 linear rigid body. The potential for scalability of the method is demonstrated by
 708 solving HJB equations in up to 30 dimensions using limited data, and empirical vali-
 709 dation indicates that the NN models are good approximations of the value function.
 710 How well the proposed techniques work for even larger problems remains an open
 711 question. Indeed, understanding the scalability of deep learning methods in general
 712 is still an active area of research. Nevertheless, we are encouraged by the simula-
 713 tions in [section 6](#) which suggest that the method may scale quite well for moderately
 714 high-dimensional problems. In addition, the computational burden associated with
 715 an increase in dimensionality is incurred entirely offline: due to the structure of NNs,
 716 increasing the dimension has a negligible impact on the speed of online control calcu-
 717 lation.

718 These promising results leave plenty of room for future development. Of special
 719 interest are extensions of the framework to solve problems with free final time and
 720 state and control constraints, which appear ubiquitously in practical applications.
 721 Such problems typically give rise to non-unique solutions of PMP and non-smooth
 722 value functions, thus presenting substantial challenges for both data generation and
 723 neural network modeling. Overcoming these obstacles would open the door to solving
 724 many important and difficult optimal control problems.

725

REFERENCES

- 726 [1] M. ABADI, A. AGARWAL, P. BARHAM, ET AL., *TensorFlow: Large-scale machine learning on*
 727 *heterogeneous systems*, 2016, <https://arxiv.org/abs/1603.04467>.
 728 [2] M. ABU-KHALAF AND F. L. LEWIS, *Nearly optimal control laws for nonlinear systems with sat-*
 729 *urating actuators using a neural network HJB approach*, *Automatica*, 41 (2005), pp. 779–

- 730 791, <https://doi.org/10.1016/j.automatica.2004.11.034>.
- 731 [3] E. AL'BREKHT, *On the optimal stabilization of nonlinear systems*, J. Appl. Math. Mech., 25(5)
- 732 (1961), pp. 1254–1266, [https://doi.org/10.1016/0021-8928\(61\)90005-3](https://doi.org/10.1016/0021-8928(61)90005-3).
- 733 [4] A. BACHOUCH, C. HURÉ, N. LANGRENÉ, AND H. PHAM, *Deep neural networks algorithms*
- 734 *for stochastic control problems on finite horizon: numerical applications*, 2018, <https://arxiv.org/abs/1812.05916>.
- 735
- 736 [5] O. BOKANOWSKI, J. GARCKE, M. GRIEBEL, AND I. KLONPMAKER, *An adaptive sparse grid semi-*
- 737 *Lagrangian scheme for first order Hamilton-Jacobi Bellman equations*, J. Sci. Comput.,
- 738 55 (2013), pp. 575–605, <https://doi.org/10.1007/s10915-012-9648-x>.
- 739 [6] L. BOTTOU, F. E. CURTIS, AND J. NOCEDAL, *Optimization methods for large-scale machine*
- 740 *learning*, SIAM Rev., 60 (2018), pp. 223–311, <https://doi.org/10.1137/16M1080173>.
- 741 [7] R. H. BYRD, G. M. CHIN, J. NOCEDAL, AND Y. WU, *Sample size selection in optimization*
- 742 *methods for machine learning*, Math. Program., 134 (2012), pp. 127–155, <https://doi.org/10.1007/s10107-012-0572-5>.
- 743
- 744 [8] R. H. BYRD, P. LU, J. NOCEDAL, AND C. ZHU, *A limited memory algorithm for bound con-*
- 745 *strained optimization*, SIAM J. Sci. Comput., 16 (1995), pp. 1190–1208, <https://doi.org/10.1137/0916069>.
- 746
- 747 [9] S. CACACE, E. CRISTIANI, M. FALCONE, AND A. PICARELLI, *A patchy dynamic programming*
- 748 *scheme for a class of Hamilton-Jacobi-Bellman equations*, SIAM J. Sci. Comput., 34
- 749 (2012), pp. A2625–A2649, <https://doi.org/10.1137/110841576>.
- 750 [10] T. CHENG, F. L. LEWIS, AND M. ABU-KHALAF, *Fixed-final-time-constrained optimal control*
- 751 *of nonlinear systems using neural network HJB approach*, IEEE Trans. Neural Netw., 18
- 752 (2007), pp. 1725–1737, <https://doi.org/10.1109/TNN.2007.905848>.
- 753 [11] Y. T. CHOW, J. DARBON, S. OSHER, AND W. YIN, *Algorithm for overcoming the curse of di-*
- 754 *mensionality for state-dependent Hamilton-Jacobi equations*, J. Comput. Phys., 387 (2019),
- 755 pp. 376–409, <https://doi.org/10.1016/j.jcp.2019.01.051>.
- 756 [12] M. G. CRANDALL AND P.-L. LIONS, *Viscosity solutions of Hamilton-Jacobi equations*, Trans.
- 757 Amer. Math. Soc., 277 (1983), pp. 1–42, <https://doi.org/10.2307/1999343>.
- 758 [13] J. DARBON, G. P. LANGLOIS, AND T. MENG, *Overcoming the curse of dimensionality for some*
- 759 *Hamilton-Jacobi partial differential equations via neural network architectures*, Res. Math.
- 760 Sci., 7 (2020), p. 20, <https://doi.org/10.1007/s40687-020-00215-6>.
- 761 [14] J. DARBON AND T. MENG, *On some neural network architectures that can represent viscosity*
- 762 *solutions of certain high dimensional Hamilton-Jacobi partial differential equations*, J.
- 763 Comput. Phys., 425 (2021), p. 109907, <https://doi.org/https://doi.org/10.1016/j.jcp.2020.109907>.
- 764
- 765 [15] J. DARBON AND S. OSHER, *Algorithms for overcoming the curse of dimensionality for certain*
- 766 *Hamilton-Jacobi equations arising in control theory and elsewhere*, Res. Math. Sci., 3
- 767 (2016), <https://doi.org/10.1186/s40687-016-0068-7>.
- 768 [16] J. DIEBEL, *Representing attitude: Euler angles, unit quaternions, and rotation vec-*
- 769 *tors*, 2006, https://www.astro.rug.nl/software/kapteyn-beta/_downloads/attitude.pdf (ac-
- 770 cessed 2020-05-16).
- 771 [17] M. FALCONE AND R. FERRETTI, *Semi-Lagrangian Approximation Schemes for Linear and*
- 772 *Hamilton-Jacobi Equations*, Society for Industrial and Applied Mathematics, Philadelphia,
- 773 PA, 2013, <https://doi.org/10.1137/1.9781611973051>.
- 774 [18] J. HAN, A. JENTZEN, AND W. E, *Solving high-dimensional partial differential equations using*
- 775 *deep learning*, Proc. Natl. Acad. Sci. USA, 115 (2018), pp. 8505–8510, <https://doi.org/10.1073/pnas.1718942115>.
- 776
- 777 [19] C. HURÉ, H. PHAM, A. BACHOUCH, AND N. LANGRENÉ, *Deep neural networks algorithms for*
- 778 *stochastic control problems on finite horizon, part I: convergence analysis*, 2018, <https://arxiv.org/abs/1812.04300>.
- 779
- 780 [20] D. IZZO, E. ÖZTÜRK, AND M. MÄRTENS, *Interplanetary transfers via deep representations of*
- 781 *the optimal policy and/or of the value function*, in Genetic and Evolutionary Computation
- 782 Conference, 2019, pp. 1971–1979, <https://doi.org/10.1145/3319619.3326834>.
- 783 [21] F. JIANG, G. CHOU, M. CHEN, AND C. J. TOMLIN, *Using neural networks to compute ap-*
- 784 *proximate and guaranteed feasible Hamilton-Jacobi-Bellman PDE solutions*, 2016, <https://arxiv.org/abs/1611.03158>.
- 785
- 786 [22] D. KALISE AND K. KUNISCH, *Polynomial approximation of high-dimensional Hamilton-Jacobi-*
- 787 *Bellman equations and applications to feedback control of semilinear parabolic PDEs*, SIAM
- 788 J. Sci. Comput., 40 (2018), pp. A629–A652, <https://doi.org/10.1137/17M1116635>.
- 789 [23] W. KANG, P. DE, AND A. ISIDORI, *Flight control in a windshear via nonlinear h_∞ methods*, in
- 790 Proceedings of the 31st IEEE Conference on Decision and Control, vol. 1, 1992, pp. 1135–
- 791 1142.

- 792 [24] W. KANG AND L. C. WILCOX, *A causality free computational method for HJB equations with*
793 *application to rigid body satellites*, in AIAA Guidance, Navigations, and Control Confer-
794 *ence*, 2015, pp. 1–10, <https://doi.org/10.2514/6.2015-2009>.
- 795 [25] W. KANG AND L. C. WILCOX, *Mitigating the curse of dimensionality: Sparse grid character-*
796 *istics method for optimal feedback control and HJB equations*, *Comput. Optim. Appl.*, 68
797 (2017), pp. 289–315, <https://doi.org/10.1007/s10589-017-9910-0>.
- 798 [26] J. KIERZENKA AND L. F. SHAMPINE, *A BVP solver based on residual control and the MATLAB*
799 *PSE*, *ACM Trans. Math. Softw.*, 27 (2001), pp. 299–316, [https://doi.org/10.1145/502800.](https://doi.org/10.1145/502800.502801)
800 [502801](https://doi.org/10.1145/502800.502801).
- 801 [27] D. LIBERZON, *Calculus of Variations and Optimal Control Theory: A Concise Introduction*,
802 Princeton University Press, Princeton, NJ, 2011, <https://doi.org/10.2307/j.ctvcvm4g0s>.
- 803 [28] D. LUKES, *Optimal regulation of nonlinear dynamical systems*, *SIAM J. Control*, 7 (1969),
804 pp. 75–100, <https://doi.org/10.1137/0307007>.
- 805 [29] O. L. MANGASARIAN, *Sufficient conditions for the optimal control of nonlinear systems*, *SIAM*
806 *J. Control*, 4 (1966), pp. 139–152, <https://doi.org/10.1137/0304013>.
- 807 [30] T. NAKAMURA-ZIMMERER, Q. GONG, AND W. KANG, *A causality-free neural network method*
808 *for high-dimensional Hamilton-Jacobi-Bellman equations*, in American Control Conference
809 (ACC), 2020, pp. 787–793, <https://doi.org/10.23919/ACC45564.2020.9147270>.
- 810 [31] C. NAVASCA AND A. J. KRENER, *Patchy Solutions of Hamilton-Jacobi-Bellman Partial Differ-*
811 *ential Equations*, Springer, Berlin-Heidelberg, 2007, pp. 251–270, [https://doi.org/10.1007/](https://doi.org/10.1007/978-3-540-73570-0_20)
812 [978-3-540-73570-0_20](https://doi.org/10.1007/978-3-540-73570-0_20).
- 813 [32] S. OSHER AND J. A. SETHIAN, *Fronts propagating with curvature-dependent speed: Algorithms*
814 *based on Hamilton-Jacobi formulations*, *J. Comput. Phys.*, 79 (1988), pp. 12–49, [https://doi.org/10.1016/0021-9991\(88\)90002-2](https://doi.org/10.1016/0021-9991(88)90002-2).
- 815 [33] M. RAISSI, P. PERDIKARIS, AND G. KARNIADAKIS, *Physics-informed neural networks: A deep*
816 *learning framework for solving forward and inverse problems involving nonlinear partial*
817 *differential equations*, *J. Comput. Phys.*, 378 (2019), pp. 686–707, [https://doi.org/10.1016/](https://doi.org/10.1016/j.jcp.2018.10.045)
818 [j.jcp.2018.10.045](https://doi.org/10.1016/j.jcp.2018.10.045).
- 819 [34] J. SIRIGNANO AND K. SPILIOPOULOS, *DGM: A deep learning algorithm for solving partial dif-*
820 *ferential equations*, *J. Comput. Phys.*, 375 (2018), pp. 1339–1364, [https://doi.org/10.1016/](https://doi.org/10.1016/j.jcp.2018.08.029)
821 [j.jcp.2018.08.029](https://doi.org/10.1016/j.jcp.2018.08.029).
- 822 [35] Y. TASSA AND T. EREZ, *Least squares solutions of the HJB equation with neural network*
823 *value-function approximators*, *IEEE Trans. Neural Netw.*, 18 (2007), pp. 1031–1041, <https://doi.org/10.1109/TNN.2007.899249>.
- 824 [36] L. N. TREFETHEN, *Spectral Methods in MATLAB*, Society for Industrial and Applied Mathe-
825 *matics*, Philadelphia, PA, 2000, <https://doi.org/10.1137/1.9780898719598>.
- 826 [37] P. VIRTANEN, R. GOMMERS, T. E. OLIPHANT, AND ET. AL., *SciPy 1.0: Fundamental algorithms*
827 *for scientific computing in Python*, *Nat. Methods*, 17 (2020), pp. 261–272, [https://doi.org/](https://doi.org/10.1038/s41592-019-0686-2)
828 [10.1038/s41592-019-0686-2](https://doi.org/10.1038/s41592-019-0686-2).
- 829 [38] I. YEGOROV AND P. M. DOWER, *Perspectives on characteristics based curse-of-dimensionality-*
830 *free numerical approaches for solving Hamilton-Jacobi equations*, *Appl. Math. Optim.*,
831 (2018), <https://doi.org/10.1007/s00245-018-9509-6>.
- 832
- 833