Near-optimal control of dynamical systems with neural ordinary differential equations

IOP Publishing Mach. Learn.: Sci. Technol. 3 (2022) 045004 https://doi.org/10.1088/2632-2153/ac92c3

MACHINE LEARNING
Science and Technology

OPEN ACCESS

RECEIVED
3 July 2022

REVISED
1 September 2022

ACCEPTED FOR PUBLICATION
16 September 2022

PUBLISHED
17 October 2022

PAPER

Near-optimal control of dynamical systems with neural ordinary differential equations

Lucas Böttcher[1,*] **and Thomas Asikis**[2]

[1] Centre for Human and Machine Intelligence, Frankfurt School of Finance and Management, 60322 Frankfurt am Main, Germany
[2] Game Theory, University of Zurich, 8006 Zurich, Switzerland
[*] Author to whom any correspondence should be addressed.

E-mail: l.boettcher@fs.de

Keywords: dynamical systems, neural ODEs, optimal control, optimization, implicit regularization

Abstract

Optimal control problems naturally arise in many scientific applications where one wishes to steer a dynamical system from an initial state $\mathbf{x}_0$ to a desired target state $\mathbf{x}^*$ in finite time T. Recent advances in deep learning and neural network–based optimization have contributed to the development of numerical methods that can help solve control problems involving high-dimensional dynamical systems. In particular, the framework of neural ordinary differential equations (neural ODEs) provides an efficient means to iteratively approximate continuous-time control functions associated with analytically intractable and computationally demanding control tasks. Although neural ODE controllers have shown great potential in solving complex control problems, the understanding of the effects of hyperparameters such as network structure and optimizers on learning performance is still very limited. Our work aims at addressing some of these knowledge gaps to conduct efficient hyperparameter optimization. To this end, we first analyze how truncated and non-truncated backpropagation through time affect both runtime performance and the ability of neural networks to learn optimal control functions. Using analytical and numerical methods, we then study the role of parameter initializations, optimizers, and neural-network architecture. Finally, we connect our results to the ability of neural ODE controllers to implicitly regularize control energy.

1. Introduction

The optimal control (OC) of complex dynamical systems is relevant in many scientific disciplines [1], including biology [2–7], epidemiology [8, 9], quantum engineering [10–12], power systems [13, 14], and supply chains [15].

Mathematically, OC problems can be formulated using variational calculus [16]. The goal is to find a Lebesgue-measurable control signal $\mathbf{u}(t) \in \mathbb{R}^m$ $(0 \leqslant t \leqslant T)$ that steers a dynamical system from its initial state $\mathbf{x}_0 \in \mathbb{R}^n$ to a desired target state $\mathbf{x}^* \in \mathbb{R}^n$ in finite time T, minimizing the control energy

$$E_T[\mathbf{u}] = \frac{1}{2}\int_0^T \|\mathbf{u}(t)\|_2^2 \mathrm{d}t \tag{1}$$

over the time interval $[0, T]$. Depending on the application, it may be useful to consider additional constraints and integrated cost functionals that are different from equation (1). A corresponding example is provided in appendix A. If additional constraints on the control function $\mathbf{u}(t)$ are imposed, it has to be chosen from a corresponding set of admissible controls [17].

The outlined OC problem aims at finding

$$\mathbf{u}^*(t) = \arg\min_{\mathbf{u}(t)} E_T[\mathbf{u}]\,, \tag{2}$$

subject to the constraint

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t), \quad \mathbf{x}(0) = \mathbf{x}_0, \quad \mathbf{x}(T) = \mathbf{x}^*, \tag{3}$$

where $\mathbf{f}(\cdot)$ denotes a vector field that depends on the system state $\mathbf{x}(t)$, control input $\mathbf{u}(t)$, and time t. According to Pontryagin's maximum principle (PMP) [18], a necessary condition for OC, the control that satisfies equations (2) and (3) is found by minimizing the corresponding control Hamiltonian

$$H(\mathbf{x}(t), \mathbf{u}(t), \boldsymbol{\lambda}(t), t) = \boldsymbol{\lambda}(t)^\top \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t) + \frac{1}{2}\|\mathbf{u}(t)\|_2^2, \tag{4}$$

at every time point t. Here, $\boldsymbol{\lambda}(t)$ is a Lagrange multiplier vector whose time-dependent components are called the adjoint (or costate) variables of the system [1, 19]. Linear systems admit a closed-form solution of $\mathbf{u}^*(t)$ [20]. For general, non-linear dynamical systems, one typically aims at minimizing

$$J = L(\mathbf{x}(T), \mathbf{x}^*) + \mu E_T[\mathbf{u}], \tag{5}$$

where $L(\mathbf{x}(T), \mathbf{x}^*)$ denotes the distance between the reached state $\mathbf{x}(T)$ and the desired target state $\mathbf{x}^*$, e.g. the mean-squared error (MSE) $L(\mathbf{x}(T), \mathbf{x}^*) \propto \|\mathbf{x}(T) - \mathbf{x}^*\|_2^2$. The Lagrange multiplier μ models the impact of control energy on the total loss. Clearly, in the limit $\mu \to \infty$, we have $\mathbf{u}^* = 0$.

While PMP provides a necessary condition for OC [18], the Hamilton–Jacobi–Bellman (HJB) equation provides both a necessary and sufficient condition for optimality [21, 22]. To solve non-linear OC problems, one typically resorts to indirect and direct numerical methods. More recently, transformation methods that convert non-linear control problems into linear ones have been proposed [23]. Examples of indirect OC solvers include different kinds of shooting methods [24] that use PMP and a control Hamiltonian to construct a system of equations describing the evolution of state and adjoint variables. In addition to indirect methods, one may also parameterize state and control functions and directly solve the resulting optimization problem. Possible function parameterizations include piecewise constant functions and other suitable basis functions [25]. Over the past two decades, pseudospectral methods have emerged as an effective approach for solving non-linear OC problems [26] with applications in aerospace engineering [27]. However, it has been shown that certain pseudospectral methods are not able to solve standard benchmark control problems [28]. In this work, we represent time-dependent control signals by artificial neural networks (ANNs) [29]. To parameterize and learn control functions, we use neural ordinary differential equations (neural ODEs) [30–34]. A schematic of the application of neural ODE control (NODEC) to a networked dynamical system is shown in figure 1.

Control methods that are based on ANNs have been applied to both discrete and continuous-time dynamical systems [35]. To keep calculations associated with gradient updates tractable, applications of ANNs in control have often focused on shallow architectures and linear dynamics. For high-dimensional and non-linear dynamics with intractable gradient updates, 'identifier' ANNs are useful to learn and replace the dynamical system underlying a given control task [35]. Recent advances in neural ODEs [31] and automatic differentiation [36] allow for the direct application of deep neural network architectures to high-dimensional non-linear models [33], thereby avoiding identifier networks [35] and limitations associated with shallow ANN structures. Other popular uses of ANNs for control are found in the fields of deep reinforcement learning [37] and neural HJB methods [38]. Deep reinforcement learning is often applied to model-free control tasks where the model dynamics are unknown or non-differentiable [39]. Neural HJB methods have been applied to state-feedback control and rely on the existence of smooth value functions [38]. Instead of explicitly minimizing control energy or other loss functionals, neural ODE controllers have been shown to exhibit implicitly regularization properties [33, 34]. In this work, we study the effect of learning protocols and hyperparameters such as network structure and optimizers on the ability of NODEC to implicitly learn near-optimal control solutions by minimizing the distance between reached and target state.

The remainder of this paper is organized as follows. Section 2 summarizes the basic concepts associated with NODEC and discusses two different backpropagation protocols for learning control functions: (i) truncated backpropagation through time (TBPTT) and (ii) backpropagation through time (BPTT) [40–42]. In section 3, we discuss how parameter initialization and activation functions affect NODEC's ability to implicitly learn OC solutions. Invoking tools from dynamical systems theory, we discuss how different neural network structures affect the learnability of a constant control solution. We will show that single neurons are able to learn OC solutions only for certain initial weights and biases. Numerical results presented in section 4 indicate that as the depth of the employed neural network increases, initial conditions have a smaller impact on the ability of NODEC to learn a near-OC solution. In section 5, we analyze how gradients in the neural network parameters induce gradient updates in both the control function and control energy. By applying NODEC to a control problem with a time-dependent OC solution,

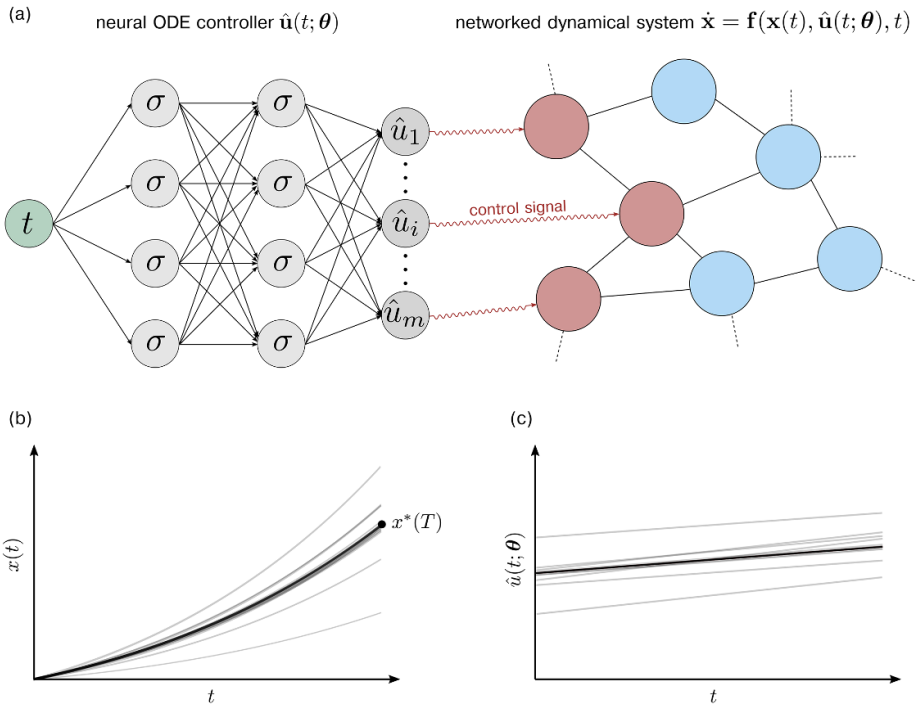

Figure 1. Controlling networked dynamical systems with neural ODEs. (a) A neural ODE controller takes the time t as an input variable and produces a control signal $\hat{\mathbf{u}}(t;\boldsymbol{\theta}) \in \mathbb{R}^m$. A networked dynamical system is then controlled by connecting control inputs $u_1(t;\boldsymbol{\theta}), \ldots, u_m(t;\boldsymbol{\theta})$ to all or a subset of nodes. Activation functions in the neural ODE controller are denoted by σ. (b), (c) Evolution of $x(t)$, the state of a one-dimensional dynamical system, and of $\hat{u}(t;\boldsymbol{\theta})$, the corresponding control function. Solid grey lines represent $x(t)$ and $\hat{u}(t;\boldsymbol{\theta})$ at different training epochs. Solutions after convergence are shown by solid black lines.

we find that adaptive gradient methods such as Adam [43] are able to approximate OC well while steepest descent (SD) fails to do so. We complement this part of our analysis with one and two-dimensional loss projections [44–49] that are useful to geometrically interpret the tradeoff between (i) a small distance between reached and target state and (ii) a small control energy. Section 6 concludes our paper.

Our source codes are publicly available at [50].

2. Backpropagation protocols

The basic steps underlying NODEC solutions of OC problems (3) are summarized below [34].

(a) Parameterize the control input $\hat{\mathbf{u}}(t;\boldsymbol{\theta})$ using a neural network such that equation (3) becomes

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t), \hat{\mathbf{u}}(t;\boldsymbol{\theta}), t)\,, \quad \mathbf{x}(0) = \mathbf{x}_0\,, \quad \mathbf{x}(T) = \mathbf{x}^*. \tag{6}$$

The vector $\boldsymbol{\theta} \in \mathbb{R}^N$ denotes the neural-network parameters.

(b) Solve the dynamical system (6) numerically for a given initial condition $\mathbf{x}(0) = \mathbf{x}_0$.

(c) Calculate the MSE $L \equiv L(\mathbf{x}(T), \mathbf{x}^*)$ between reached state $\mathbf{x}(T)$ and target state $\mathbf{x}^*$ and backpropagate gradients to update neural-network weights.

(d) Iterate steps (a)–(c) until convergence is reached (i.e. until $L(\mathbf{x}(T), \mathbf{x}^*)$ is smaller than a certain threshold).

Instead of learning the vector field $\mathbf{f}$ as in related applications of neural ODEs [30, 31], we use ANNs to represent time-dependent control functions $\hat{\mathbf{u}}(t;\boldsymbol{\theta})$ (see figure 1(a)).

We will discuss in the following sections that control solutions with a small control energy $E_T[\mathbf{u}]$ can be obtained without explicitly including $E_T[\mathbf{u}]$ in the loss function. Even if the control energy is included directly in the loss function, one may still have to search for a Lagrange multiplier μ (see equation (5)) that is associated with a desired tradeoff between control energy and distance from the target state. We provide a corresponding example in appendix A. Identifying the optimal value of a Lagrange multiplier [51] or Lagrange costate might prove challenging or intractable in practice [52]. Implicit energy regularization properties that have been reported for NODEC [33, 34] may therefore be useful for the design of effective control methods.

To learn control functions that are suitable to solve a certain control problem minimizing the MSE between reached and target state, there exist two main classes of backpropagation protocols: (i) truncating backpropagation and approximating a desired control function within a certain subinterval of $[0, T]$, and (ii) learning the control input over the whole interval $[0, T]$ using BPTT. The first option is also referred to as TBPTT.

In the following two subsections, we will discuss the differences between BPTT and TBPTT with respect to (w.r.t.) learning control functions. We will also provide an example to compare the performance of BPTT and TBPTT.

2.1. Truncated backpropagation through time

For notational brevity, we will consider a TBPTT protocol for a one-dimensional flow $\dot{x} = f(x(t), \hat{u}(t;\boldsymbol{\theta}), t)$ where neural-network parameters $\boldsymbol{\theta}$ are being updated for a time t' in the loss function $L = 1/2(x(T) - x^*)^2$. Gradients are calculated through the loss function if the underlying dynamical system time t is equal to t'. Generalizations for multiple evaluation times and higher-dimensional flows follow directly from the equations below. The TBPTT gradient of L w.r.t. $\boldsymbol{\theta}$ is

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}}^{(t')} L = \nabla_{\boldsymbol{\theta}}^{(t')} \frac{1}{2} \left(x(T) - x^*\right)^2 &= \nabla_{\boldsymbol{\theta}}^{(t')} \frac{1}{2} \left(\int_0^T f(x(t), \hat{u}(t;\boldsymbol{\theta}), t)\, \mathrm{d}t + x_0 - x^* \right)^2 \\
&= \nabla_{\boldsymbol{\theta}}^{(t')} \frac{1}{2} \left(\int_0^{t'-\epsilon/2} f(x(t), \hat{u}(t;\boldsymbol{\theta}), t)\, \mathrm{d}t + \int_{t'+\epsilon/2}^T f(x(t), \hat{u}(t;\boldsymbol{\theta}), t)\, \mathrm{d}t \right. \\
&\quad \left. + \epsilon f(x(t'), \hat{u}(t', \boldsymbol{\theta}), t') + x_0 - x^* \right)^2 \\
&= \epsilon \left. \frac{\mathrm{d}f}{\mathrm{d}\hat{u}} \mathcal{J}_{\hat{u}}^\top \right|_{t=t'} \left(\int_0^T f(x(t), \hat{u}(t;\boldsymbol{\theta}), t)\, \mathrm{d}t + x_0 - x^* \right) \\
&= \epsilon \left. \frac{\mathrm{d}f}{\mathrm{d}\hat{u}} \mathcal{J}_{\hat{u}}^\top \right|_{t=t'} \frac{\partial L}{\partial x(T)} = \epsilon \left(\frac{\partial x}{\partial \hat{u}} \frac{\partial f}{\partial x} + \frac{\partial f}{\partial \hat{u}} \right) \mathcal{J}_{\hat{u}}^\top \Big|_{t=t'} \frac{\partial L}{\partial x(T)}\,,
\end{aligned} \tag{7}$$

where ϵ is a small positive number that determines the width of the TBPTT interval. Note that ϵ can be absorbed in the learning rate associated with gradient-descent updates of $\boldsymbol{\theta}$. The Jacobian of $\hat{u}(t;\boldsymbol{\theta})$ w.r.t. $\boldsymbol{\theta}$ is

$$\mathcal{J}_{\hat{u}} = \left(\partial \hat{u}/\partial \theta_1, \ldots, \partial \hat{u}/\partial \theta_N\right). \tag{8}$$

In the described TBPTT protocol, we used that

$$\nabla_{\boldsymbol{\theta}}^{(t')} f(x(t), \hat{u}(t;\boldsymbol{\theta}), t) := \begin{cases} \frac{\mathrm{d}f}{\mathrm{d}\hat{u}} \mathcal{J}_{\hat{u}}^\top & t = t' \\ 0 & \text{otherwise} \end{cases}. \tag{9}$$

2.2. Backpropagation through time

Without truncating the BPTT protocol, the gradient of L w.r.t. $\boldsymbol{\theta}$ is

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} L = \nabla_{\boldsymbol{\theta}} \frac{1}{2} \left(x(T) - x^*\right)^2 &= \nabla_{\boldsymbol{\theta}} \frac{1}{2} \left(\int_0^T f(x(t), \hat{u}(t;\boldsymbol{\theta}), t)\, \mathrm{d}t + x_0 - x^* \right)^2 \\
&= \int_0^T \frac{\mathrm{d}f}{\mathrm{d}\hat{u}} \mathcal{J}_{\hat{u}}^\top\, \mathrm{d}t \left(\int_0^T f(x(t), \hat{u}(t;\boldsymbol{\theta}), t)\, \mathrm{d}t + x_0 - x^* \right) \\
&= \int_0^T \left(\frac{\partial x}{\partial \hat{u}} \frac{\partial f}{\partial x} + \frac{\partial f}{\partial \hat{u}} \right) \mathcal{J}_{\hat{u}}^\top\, \mathrm{d}t \left(\int_0^T f(x(t), \hat{u}(t;\boldsymbol{\theta}), t)\, \mathrm{d}t + x_0 - x^* \right) \\
&= I\left[D_{\hat{u}} x(T), \mathcal{J}_{\hat{u}}^\top \right] \frac{\partial L}{\partial x(T)}\,,
\end{aligned} \tag{10}$$

where the notation $I\left[D_{\hat{u}} x(T), \mathcal{J}_{\hat{u}}^\top\right]$ is used to indicate that the integration associated with the derivative $D_{\hat{u}} x(T) := \mathrm{d}x(T)/\mathrm{d}\hat{u}$ has to be applied to $\mathcal{J}_{\hat{u}}^\top$. Note that $\partial L/\partial x(T)$ is not time-dependent, while the derivative of the loss function L w.r.t. $\hat{u}$ is an operator that acts on the time-dependent quantity $\mathcal{J}_{\hat{u}}^\top$.

We will now compare BPTT and TBPTT protocols for controlling the two-dimensional linear flow

$$\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u}\,, \quad \mathbf{x}(0) = (0.5, 0.5)^\top\,, \quad \mathbf{x}^* = (1, -1)^\top\,, \tag{11}$$

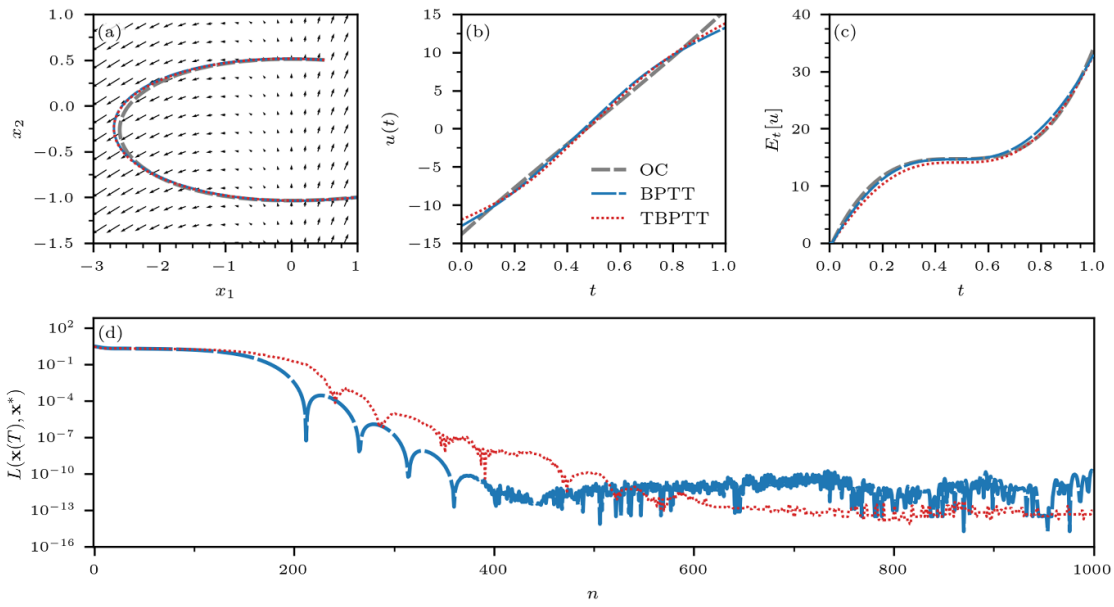

Figure 2. Learning control functions with different backpropagation protocols. (a) The evolution of $\mathbf{x}(t)$ of the solution of equation (11) in the (x_1, x_2) plane. (b), (c) The evolution of the corresponding control function and control energy. The OC solution is indicated by a dashed grey line. Dashed blue lines and dotted red lines show the solutions associated with BPTT and TBPTT protocols, respectively. The underlying neural network has 2 hidden exponential linear unit (ELU) layers and 14 neurons per hidden layer. We trained the neural network for 1000 epochs using Adam. The learning rates are $\eta = 3 \times 10^{-3}$ (BPTT) and $\eta = 5 \times 10^{-3}$ (TBPTT). (d) The loss $L(\mathbf{x}(T), \mathbf{x}^*)$ as a function of training epochs n. Both backpropagation protocols can achieve low loss values after a few hundred epochs of training.

with

$$A = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 1 \\ 0 \end{pmatrix}. \tag{12}$$

The OC $\mathbf{u}^*(t)$ associated with solving equation (11) and minimizing $E_T[\mathbf{u}]$ (see equation (1)) has been derived in [20]. We compare the optimal solution with the solutions obtained using (T)BPTT and the MSE loss

$$L = \frac{1}{2} \|\mathbf{x}(T) - \mathbf{x}^*\|_2^2 . \tag{13}$$

Figure 2(a) shows the evolution of $\mathbf{x}(t)$ in the (x_1, x_2) plane. Both backpropagation protocols, BPTT and TBPTT, produce solutions after 1000 epochs of training that are similar to the OC solution. This similarity is also reflected in the evolution of the control signal and control energy (see figures 2(b) and (c)). To solve equation (11) numerically, we use a forward Euler method with step size 0.01. In the TBTT protocol, we use a different gradient evaluation time t' in each backpropagation step. While carrying out our numerical experiments, we observe that convergence of both methods is sensitive to choice of learning rate and optimizer. Figure 2(d) shows that BPTT converges faster with a smaller learning rate, but at later stages in training it becomes less stable. On the contrary, TBPTT, converges slower with a higher learning rate, but it is more stable after convergence. Under the employed parametrization, both methods are capable to converge fast to low error values. In terms of computation time, on the same hardware and according to `tqdm` library measurements, BPTT performs around 65 training epochs per second, whereas TBPTT achieves approximately 125 epochs per second. The difference in total computation time may be attributed to the fewer gradient evaluations required by TBPTT.

3. Parameter initialization and activation functions

As shown in section 2 and in previous work [33, 34], NODEC is able to learn near-OC solutions without explicitly minimizing $E_T[\mathbf{u}]$. To provide insights into the learning dynamics underlying NODEC, we study two control problems associated with a one-dimensional linear flow. The first dynamical system has a constant OC solution while that of the second dynamical system is time-dependent. In this section, we will show that shallow architectures require one to tune initial weights and biases to learn OC solutions without

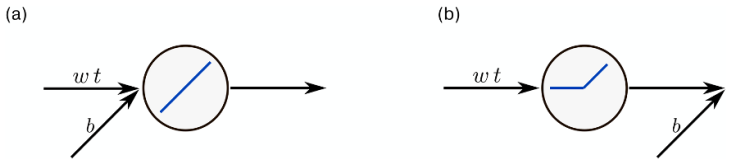

Figure 3. Examples of single-neuron structures. (a) The input is the time t and the output is $wt + b$. (b) The input is the time t and the output is $\max(0, wt) + b$.

accounting for a control energy term in the loss function. The next section then provides numerical evidence that deeper architectures are less prone to such weight and bias initialization effects.

3.1. Constant control

We first consider the OC problem associated with the scalar control function

$$u^*(t) = \arg\min_{u(t)} E_T[u] \tag{14}$$

subject to a constraint that is given by the state-independent, one-dimensional flow

$$\dot{x} = u\,, \quad x(0) = x_0\,, \quad x(T) = x^*\,. \tag{15}$$

The corresponding control Hamiltonian (see equation (4)) is

$$H = \lambda u(t) + \frac{1}{2}u(t)^2\,. \tag{16}$$

We use PMP to derive $u^*(t)$, which we will assume to be the OC associated with equations (14) and (15). We obtain

$$0 = \left.\frac{\partial H}{\partial u}\right|_{u=u^*} = \lambda + u^*(t)\,, \tag{17}$$

$$\dot{\lambda} = -\frac{\partial H}{\partial x} = 0\,. \tag{18}$$

Integrating equation (15) with $u^*(t) = -\lambda$ yields

$$x(t) = x_0 - \int_0^t \lambda\,\mathrm{d}t' = x_0 + \frac{t(x^* - x_0)}{T}\,, \tag{19}$$

where we used that the reached state $x(T)$ is equal to the target state x^*. The OC is $u^* = (x^* - x_0)/T$.

A simple neural network that consists of a single rectified linear unit (ReLU) and uses the time t as an input is, in principle, able to represent the constant OC u^*. For a neural network–generated control signal $\hat{u}(t;\boldsymbol{\theta}) = \mathrm{ReLU}(wt + b) = \max(0, wt + b)$, where $\boldsymbol{\theta} = (w, b)^\top$, we have $\hat{u}(t;\boldsymbol{\theta}) = u^*(t)$ if $w^* = 0$ and $b^* = (x^* - x_0)/T$.

Near-optimal control solutions have been obtained representing control functions by neural ODEs and optimizing

$$L = \frac{1}{2}(x(T) - x^*)^2\,, \tag{20}$$

without explicitly accounting for the integrated cost or control energy (1) [34]. Under what conditions (e.g. initial weights and biases, optimization algorithm, and neural-network structure) can a neural network learn a control function $\hat{u}(t;\boldsymbol{\theta})$ that is close to $u^*(t)$ by optimizing equation (20)? As a starting point for addressing this question, we will focus on analytically tractable learning algorithms and neural network structures. In the following sections, we will then use these gained insights to draw conclusions for control problems that involve higher-dimensional neural network structures and generalized optimization algorithms.

3.1.1. Single-neuron structure with linear activation

In the following example, we use simple neural network that consists of a linear activation and a weight and bias parameter (see figure 3(a)). The corresponding loss function is

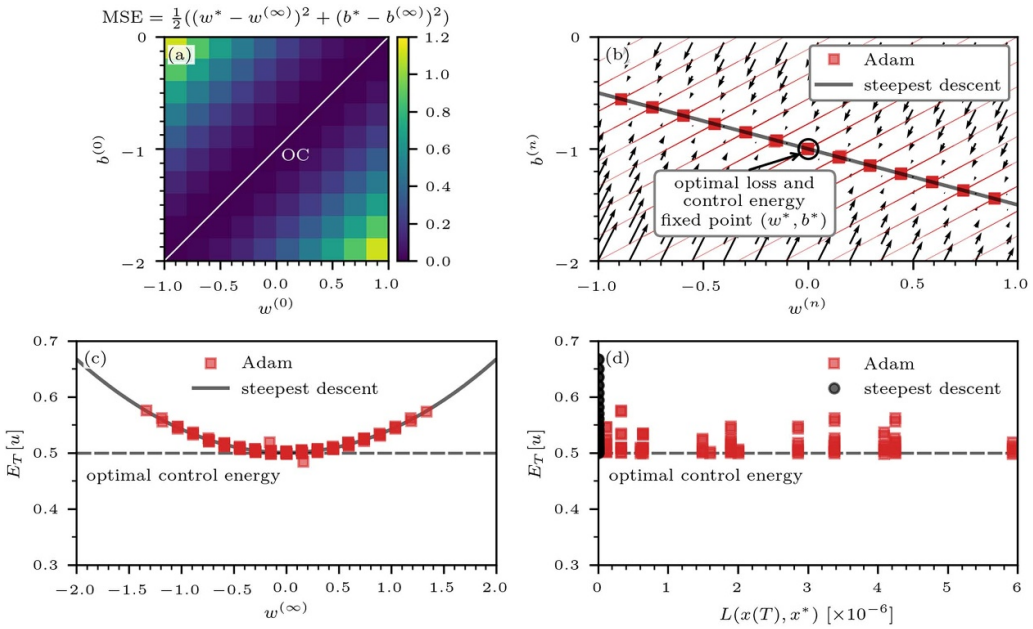

Figure 4. Approximating constant control with a linear activation function. (a) The mean squared error (MSE) associated with the deviation of the learned weight and bias, $(w^{(\infty)}, b^{(\infty)})$, from the OC weight and bias, $(w^*, b^*) = (0, -1)$, for different initial weights $w^{(0)}$ and biases $b^{(0)}$. The solid white line indicates the location of initial weights and biases for which gradient-descent learning converges to OC. (b) Solid red lines and filled red squares indicate learning trajectories and learned parameters $(w^{(\infty)}, b^{(\infty)})$, respectively. The solid black line represents fixed points of steepest-descent learning [i.e. $w^{(\infty)} = -2(1 + b^{(\infty)})$ as defined in equation (24)]. The fixed point $(w^*, b^*) = (0, -1)$ corresponding to the OC solution is marked by a black circle. (c) The control energy $E_T[u]$ (see equation (1)) as a function of learned neural-network weights $w^{(\infty)}$. Energies associated with Adam-based simulation results and steepest-descent fixed points are indicated by filled red squares and a solid black line, respectively. The dashed grey line marks the energy associated with OC. (d) Control energy as a function of the loss function (21). In all simulations, we set $T = 1$, $x_0 = 0$, and $x^* = -1$. We trained the controller with Adam using a learning rate $\eta = 0.1$ for 100 epochs.

$$L(x(T), x^*) = \frac{1}{2}(x(T) - x^*)^2 = \frac{1}{2}\left(\int_0^T (wt + b)\,\mathrm{d}t + x_0 - x^*\right)^2$$
$$= \frac{1}{2}\left(\frac{1}{2}wT^2 + bT + x_0 - x^*\right)^2. \tag{21}$$

Using steepest-descent learning and BPTT, weights and biases are iteratively updated according to

$$w^{(n+1)} = w^{(n)} - \eta\frac{\partial L}{\partial w^{(n)}},$$
$$b^{(n+1)} = b^{(n)} - \eta\frac{\partial L}{\partial b^{(n)}}, \tag{22}$$

where $w^{(0)}$ and $b^{(0)}$ denote initial weight and bias, respectively. The quantity η denotes the learning rate. Using the loss function (21), the gradients in equation (22) are

$$\frac{\partial L}{\partial w^{(n)}} = \frac{1}{2}T^2\left(\frac{1}{2}w^{(n)}T^2 + b^{(n)}T + x_0 - x^*\right),$$
$$\frac{\partial L}{\partial b^{(n)}} = T\left(\frac{1}{2}w^{(n)}T^2 + b^{(n)}T + x_0 - x^*\right). \tag{23}$$

The fixed points $(w^{(\infty)}, b^{(\infty)})$ associated with equation (23) satisfy

$$w^{(\infty)} = -\frac{2}{T^2}(b^{(\infty)}T + x_0 - x^*). \tag{24}$$

For $T = 1$, $x_0 = 0$, and $x^* = -1$, the fixed points are given by $w^{(\infty)} = -2(1 + b^{(\infty)})$. Recall that, in this example, the weight and bias of OC are $w^* = 0$ and $b^* = -1$. This fixed point can only be reached for specific initializations as shown in figures 4(a) and (b). Other fixed points are associated with certain tradeoffs between small control energies

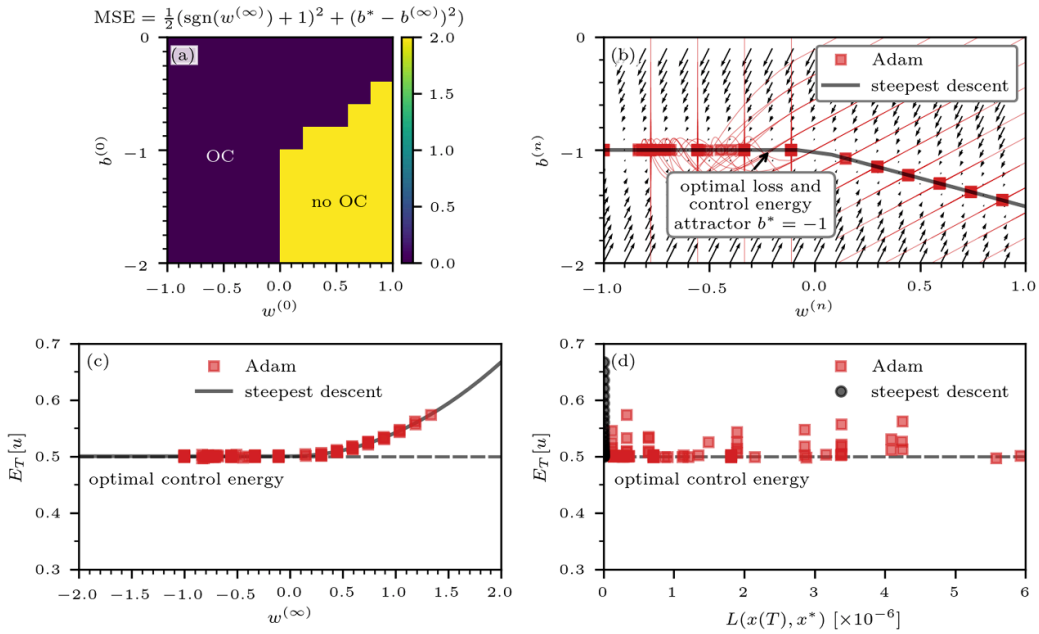

Figure 5. Approximating constant control with a ReLU activation function. (a) The mean squared error (MSE) associated with the deviation of the learned weight and bias, $(w^{(\infty)}, b^{(\infty)})$, from the OC weight and bias, $w^* \leqslant 0, b^* = -1$, for different initial weights $w^{(0)}$ and biases $b^{(0)}$. The deep purple region indicates initial weights and biases for which the neural network learns OC. In the yellow region, OC cannot be learned. (b) Solid red lines and filled red squares indicate learning trajectories and learned parameters $(w^{(\infty)}, b^{(\infty)})$, respectively. The solid black line represents the attractor of steepest-descent learning [i.e. $b^* = -1$ if $w^{(\infty)} \leqslant 0$ and $w^{(\infty)} = -2(1 + b^{(\infty)})$ if $w^{(\infty)} > 0$]. The attractor $w^* \leqslant 0, b^* = -1$ corresponds to OC. (c) The control energy $E_T[u]$ (see equation (1)) as a function of learned neural-network weights $w^{(\infty)}$. Energies associated with Adam-based simulation results and steepest-descent fixed points are indicated by filled red squares and the solid black line, respectively. The dashed grey line marks the energy associated with OC. (d) Control energy as a function of the loss function (21). In all simulations, we set $T = 1$, $x_0 = 0$, and $x^* = -1$. We trained the controller with Adam using a learning rate $\eta = 0.1$ for 100 epochs.

$$E_{T=1}[u] = \frac{1}{2}\int_0^1 \left[b^{(\infty)} - 2(1 + b^{(\infty)})\, t\right]^2 \mathrm{d}t = \frac{1}{6}[4 + b^{(\infty)}(2 + b^{(\infty)})] \tag{25}$$

and small losses $L(x(1), x^*)$ (see figures 4(c) and (d)).

3.1.2. Single-neuron structure with ReLU activation

We now use a slightly different neural-network structure that consists of a single ReLU (see figure 3(b)). The loss function in this example is

$$\begin{aligned} L(x(T), x^*) &= \frac{1}{2}(x(T) - x^*)^2 \\ &= \frac{1}{2}\left(\int_0^T (\max(0, wt) + b)\,\mathrm{d}t + x_0 - x^*\right)^2 \\ &= \begin{cases} \frac{1}{2}\left(\frac{1}{2}wT^2 + bT + x_0 - x^*\right)^2 & \text{if } w \geqslant 0 \\ \frac{1}{2}\left(bT + x_0 - x^*\right)^2 & \text{otherwise} \end{cases}. \end{aligned} \tag{26}$$

If the weights are larger than or equal to 0, the steepest-descent equations are equivalent to equation (23). Otherwise, we have

$$\begin{aligned} w^{(n+1)} &= w^{(n)}, \\ b^{(n+1)} &= b^{(n)} - \eta T(b^{(n)} T + x_0 - x^*). \end{aligned} \tag{27}$$

If $w^{(n)} < 0$ for some n, the fixed point of the gradient descent (27) is equivalent to the OC $b^{(\infty)} = b^* = \frac{x^* - x_0}{T}$. Figures 5(a) and (b) shows that the optimal-control domain of equation (26) is substantially larger than that resulting from a gradient descent in equation (21). The control energy approaches that of OC for $w^{(\infty)} \leqslant 0$, while larger values of $w^{(\infty)}$ are associated with larger control energies (see figure 5(c)). As in the previous example, one can observe that tradeoffs between small control energies and small losses are possible (see figure 5(d)).

Equation (27) also shows that the OC solution associated with equation (15) is a global attractor if one would just learn a bias term, reflecting the fact that the OC is given by a constant.

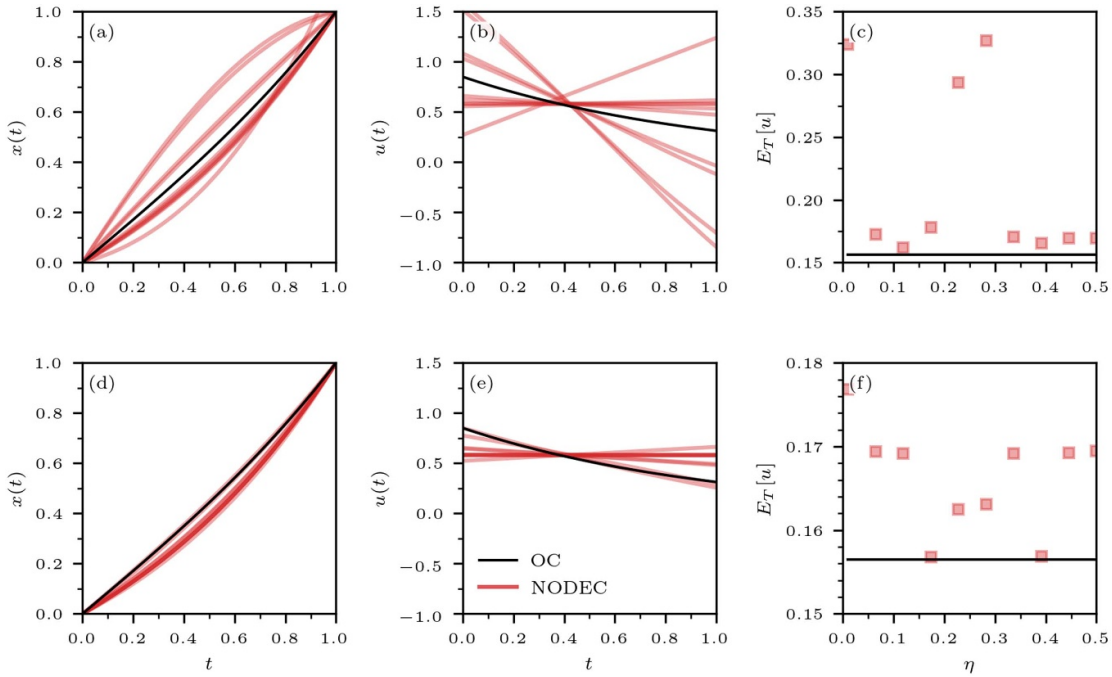

Figure 6. Approximating time-dependent control. We use NODEC to control equation (29) for $x_0 = 0$ and $T = a = b = x^* = 1$. The neural network that we use in this example consists of 2 hidden layers with 6 ELU neurons each. (a)–(c) Initial weights and biases are set to 1. (d)–(f) Initial weights and biases are set to 0.1. Different solid red lines are associated with different learning rates. Solid black lines indicate OC solutions (30)–(32). We trained NODEC for 1000 epochs using Adam.

3.2. Time-dependent control

To study the ability of NODEC to learn time-dependent control functions, we consider the OC problem

$$u^*(t) = \arg\min_{u(t)} E_T[u] \tag{28}$$

subject to

$$\dot{x} = ax + bu\,, \quad x(0) = x_0\,, \quad x(T) = x^*\,. \tag{29}$$

A practical application of this OC problem is temperature control in a room [53]. The corresponding OC signal is

$$u^*(t) = \frac{ae^{-at}}{b\sinh(aT)}\left(x^* - x_0 e^{aT}\right)\,. \tag{30}$$

Note that for $a > 0$ the magnitude of $u^*(t)$ decays exponentially because larger values of t are associated with larger state values of the uncontrolled dynamics $\dot{x} = ax$. The opposite holds for $a < 0$. The evolution of the system state $x(t)$ under the influence of $u^*(t)$ is

$$x^*(t) = x_0 e^{at} + \frac{\sinh(at)}{\sinh(aT)}\left(x^* - x_0 e^{aT}\right)\,, \tag{31}$$

and the control energy associated with the OC signal (30) is

$$E_T[u^*] = \frac{1}{2}\int_0^T u^*(t)^2 \mathrm{d}t = \frac{a(1-e^{-2aT})(x^* - x_0 e^{aT})^2}{4b^2\sinh^2(aT)}\,. \tag{32}$$

As in the prior examples, we use NODEC to learn $\hat{u}(t;\boldsymbol{\theta})$ by minimizing the MSE loss (20). We set $x_0 = 0$ and $T = a = b = x^* = 1$, such that $x^*(t) = \sinh(t)/\sinh(1)$, $u^*(t) = e^{-t}/\sinh(1)$, and $E_T[u^*] = 1/(e^2 - 1) \approx 0.157$. To capture the time-dependence of the control input, we use a neural network with two hidden layers and six exponential linear units (ELUs) per hidden layer. Numerical experiments with smaller architectures showed that the exponential decay of the OC solution could not be captured well. The number of timesteps in our simulations is 100. If bias and weight values are initialized to 1, NODEC is able to steer the dynamical system toward the desired target state (see figure 6(a)). For a learning rate of about 0.12,

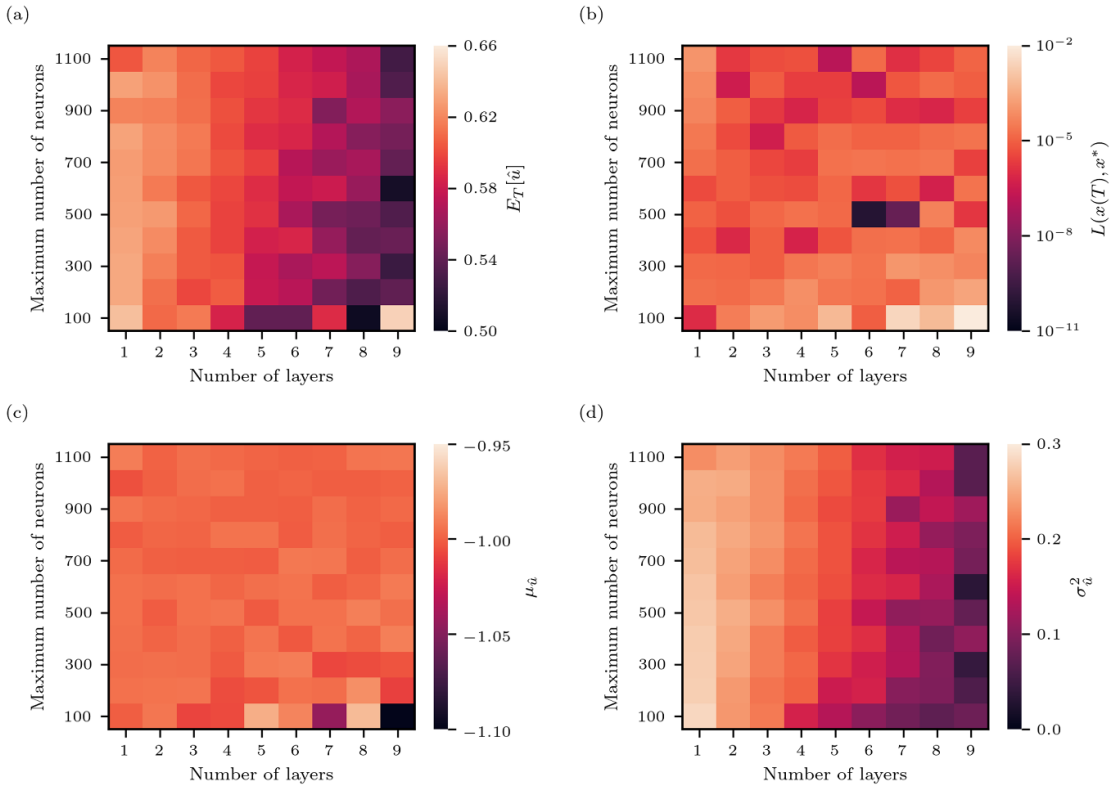

Figure 7. Effect of variations in number of layers and neurons per layer on the ability of NODEC to learn a constant control signal. The underlying architectures use hyperbolic tangent activations in all hidden layers. Bias terms are not included. Each ANN is trained for 100 epochs using Adam and a learning rate $\eta = 10^{-3}$. We set the number of neurons per layer to be the greatest integer less than or equal to the maximum number of neurons (vertical axes) divided by the number of layers (horizontal axes). The optimal constant control is -1 and the OC energy is $1/2$. Heatmaps show the (a) control energy $E_T[\hat{u}]$, (b) loss $L(x(T), x^*)$, (c) mean control signal over time $\mu_{\hat{u}}$, and (d) control signal variance over time, $\sigma^2_{\hat{u}}$.

we observe that NODEC approximates the OC solution by a constant control with a similar control energy (see figures 6(b) and (c)). The relative difference between the two control energies is about 3.5%. Using smaller initial weights and biases can help NODEC approximate the OC solution even more closely (see figures 6(d)–(f)). We find a minimum relative control energy difference of less than 0.2%. Earlier work [34] also suggested that small initial weights and biases can be helpful for learning OC solutions without explicitly regularizing control energy.

4. Neural network depth and width

After having discussed the effect of parameter initialization on the ability of NODEC to implicitly learn OC solutions, we will now study how different numbers of layers and neurons per layer can help improve learning performance in the context of the examples from section 3.

4.1. Constant control

We first focus on the ability of deeper ANN architectures to implicitly learn a constant control function for the example that we discussed in section 3.1. The OC signal is $u^* = -1$, and we use an ANN with hyperbolic tangent activations, between one to nine layers, and up to 1100 neurons. Weights and biases are initially distributed according to $\mathcal{U}(-\sqrt{k}, \sqrt{k})$, where k denotes the number of input features of a certain layer. Figure 7(a) shows that an increase in the number of layers is associated with a decrease in control energy toward the optimal value of 0.5. Even if the number of layers increases, loss values are still small and almost unaffected by the change of the ANN architecture (see figure 7(b)). Increasing the number of layers also helps in learning constant controls, as the temporal mean $\mu_{\hat{u}}$ is close to the optimal constant control value for most simulated scenarios with many layers (see figure 7(c)) while the corresponding variance $\sigma^2_{\hat{u}}$ gets closer to 0 (see figure 7(d)). To summarize, we observe that adding layers is more beneficial than adding more neurons in order to approximate a constant OC function.

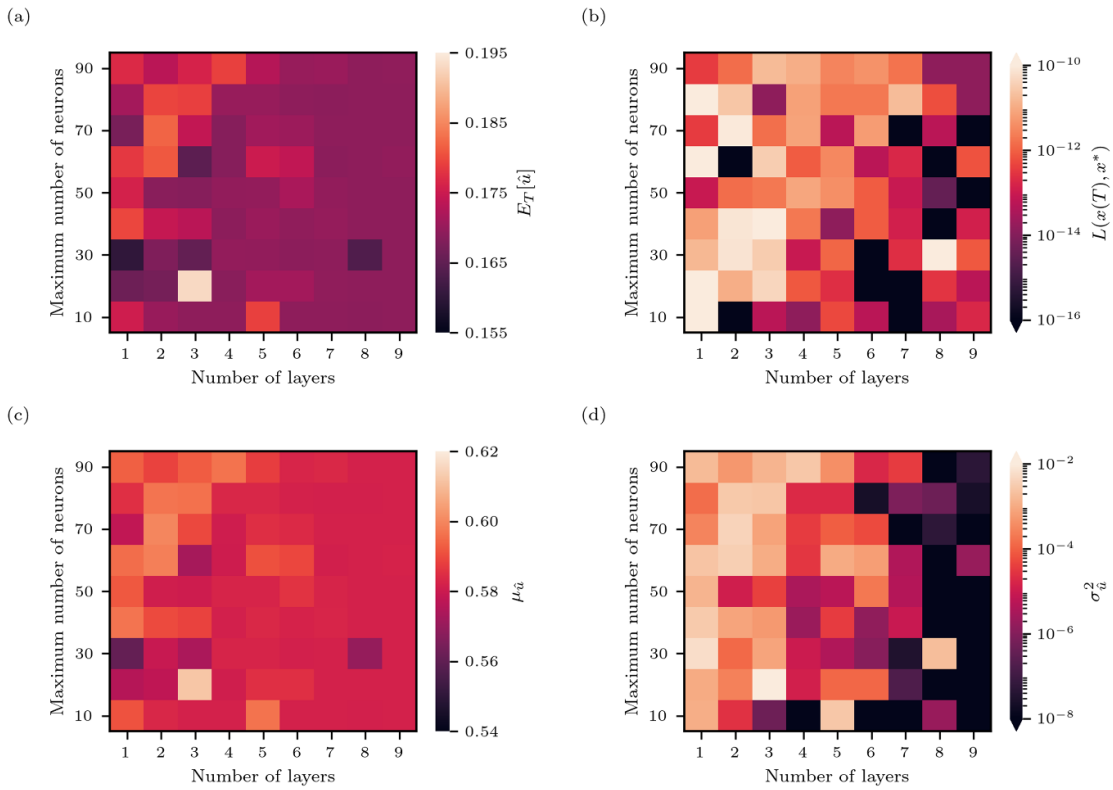

Figure 8. Effect of variations in number of layers and neurons per layer on the ability of NODEC to learn the time-dependent control signal (30) for $x_0 = 0$ and $T = a = x^* = 1$. The underlying architectures use bias terms and ELU activations in all hidden layers. Each ANN is trained for 500 epochs using Adam and a learning rate $\eta = 3 \times 10^{-3}$. We set the number of neurons per layer to be the greatest integer less than or equal to the maximum number of neurons (vertical axes) divided by the number of layers (horizontal axes). The OC energy is approximately 0.157. Heatmaps show the (a) control energy $E_T[\hat{u}]$, (b) loss $L(x(T), x^*)$, (c) mean control signal over time $\mu_{\hat{u}}$, and (d) control signal variance over time, $\sigma^2_{\hat{u}}$.

4.2. Time-dependent control

For learning the time-dependent control signal that we studied in section 3.2, we use an ANN with ELU activations, between one to nine layers, and up to 90 neurons. Weights and biases are initially distributed according to $\mathcal{U}(-\sqrt{k}, \sqrt{k})$, where k denotes the number of input features of a certain layer. Figure 8(a) shows that smaller control energies can be achieved on average as the number of layers increases. For one or eight layers and 30 neurons in total, we observe that the ANN controller is able to implicitly learn a solution with a small loss and a control energy that is close to that of the optimal solution. However, the loss may increase with the number of layers (see figure 8(b)). Increasing the number of layers while keeping a relatively small number of neurons per layer seems to affect convergence as most ANNs seem to converge to a non-optimal constant control—the corresponding variances in figure 8(d) are close to 0 (darker color). We show in section 5.3 that the constant OC approximation of equation (30) has a control energy of about 0.169 (see figure 8). In this example, we find that implicit regularization of control energy can be achieved by selecting appropriate hyperparameters (e.g. one or eight layers and 30 neurons in total).

In appendix B, we study the effect of different numbers of layers and neurons on the performance of NODEC in implicitly learning near-optimal control solutions associated with the two-dimensional flow discussed in section 2.

5. Implicit energy regularization

For sufficiently small changes in the neural-network parameters $\Delta\boldsymbol{\theta}^{(n)}$ (i.e. a sufficiently small learning rate η), a steepest-descent update in $\boldsymbol{\theta}^{(n)}$ changes a control input according to

$$\hat{\mathbf{u}}(t;\boldsymbol{\theta}^{(n+1)}) = \hat{\mathbf{u}}(t;\boldsymbol{\theta}^{(n)}) + \mathcal{J}_{\hat{\mathbf{u}}^{(n)}}\Delta\boldsymbol{\theta}^{(n)} = \hat{\mathbf{u}}(t;\boldsymbol{\theta}^{(0)}) + \sum_{i=0}^{n}\mathcal{J}_{\hat{\mathbf{u}}^{(i)}}\Delta\boldsymbol{\theta}^{(i)}$$

$$= \hat{\mathbf{u}}(t;\boldsymbol{\theta}^{(0)}) - \eta\sum_{i=0}^{n}\mathcal{J}_{\hat{\mathbf{u}}^{(i)}}\nabla_{\boldsymbol{\theta}^{(i)}}L\,, \tag{33}$$

where $\Delta\boldsymbol{\theta}^{(n)} = -\eta\nabla_{\boldsymbol{\theta}^{(n)}} L$. The gradient $\nabla_{\boldsymbol{\theta}^{(n)}} L$ can be evaluated with BPTT and TBPTT protocols (see equations (7) and (10)). We will now study the evolution of control inputs under neural-network parameter updates.

5.1. Induced gradient update

For the linear one-dimensional flow (29), we have

$$x(t) = e^{at}\left(x_0 + \int_0^t be^{-at'}\hat{u}(t';\boldsymbol{\theta}^{(n)})\,\mathrm{d}t'\right) \tag{34}$$

and

$$L = \frac{1}{2}\left(x(T) - x^*\right)^2 = \frac{1}{2}\left[e^{aT}\left(x_0 + \int_0^T be^{-at}\hat{u}(t;\boldsymbol{\theta}^{(n)})\,\mathrm{d}t\right) - x^*\right]^2. \tag{35}$$

Using the BPTT notation introduced in section 2, we obtain

$$\nabla_{\boldsymbol{\theta}^{(n)}} L = be^{aT}\int_0^T e^{-at}\nabla_{\boldsymbol{\theta}^{(n)}}\hat{u}(t;\boldsymbol{\theta}^{(n)})\,\mathrm{d}t\left[e^{aT}\left(x_0 + \int_0^T be^{-at}\hat{u}(t;\boldsymbol{\theta}^{(n)})\,\mathrm{d}t\right) - x^*\right] \tag{36}$$

$$\begin{aligned}
&= be^{aT}\int_0^T \mathcal{J}_{\hat{u}^{(n)}}^{\top} e^{-at}\,\mathrm{d}t\,\frac{\partial L}{\partial x(T)}\\
&= I\left[D_{\hat{u}^{(n)}}x(T), \mathcal{J}_{\hat{u}^{(n)}}^{\top}\right]\frac{\partial L}{\partial x(T)}.
\end{aligned} \tag{37}$$

Invoking equation (33), the induced gradient update in $\hat{u}(t;\boldsymbol{\theta}^{(n)})$ is

$$\begin{aligned}
\hat{u}(t;\boldsymbol{\theta}^{(n+1)}) &= \hat{u}(t;\boldsymbol{\theta}^{(n)}) - \eta\mathcal{J}_{\hat{u}^{(n)}} be^{aT}\int_0^T \mathcal{J}_{\hat{u}^{(n)}}^{\top} e^{-at}\,\mathrm{d}t\,\frac{\partial L}{\partial x(T)}\\
&= \hat{u}(t;\boldsymbol{\theta}^{(n)}) - \eta\mathcal{J}_{\hat{u}^{(n)}} I\left[D_{\hat{u}^{(n)}}x(T), \mathcal{J}_{\hat{u}^{(n)}}^{\top}\right]\frac{\partial L}{\partial x(T)}.
\end{aligned} \tag{38}$$

Based on equation (38), we define the weighted total change in $\hat{u}(t;\boldsymbol{\theta}^{(n)})$ for the linear dynamics (29) as

$$\begin{aligned}
\Delta\hat{U}^{(n)} &:= \int_0^T \hat{u}(t;\boldsymbol{\theta}^{(n+1)})e^{-at}\,\mathrm{d}t - \int_0^T \hat{u}(t;\boldsymbol{\theta}^{(n)})e^{-at}\,\mathrm{d}t\\
&= -\eta^{-1}b^{-1}e^{-aT}\|\Delta\boldsymbol{\theta}^{(n)}\|_2^2\left(\frac{\partial L}{\partial x(T)}\right)^{-1},
\end{aligned} \tag{39}$$

where $\Delta\boldsymbol{\theta}^{(n)}$ can be directly evaluated with standard backpropagation methods, thus avoiding the evaluation of the Jacobian elements $(\mathcal{J}_{\hat{u}^{(n)}})_i = \partial\hat{u}^{(n)}/\partial\theta_i^{(n)}$ in equation (33).

Figure 9 shows a comparison of learned control signals $\hat{u}(t;\boldsymbol{\theta}^{(n)})$ for the linear dynamics (29) with $x_0 = 0$ and $T = a = b = x^* = 1$. The neural network that we use in this example consists of 2 hidden layers with 6 ELU neurons each. Weights and biases are initialized to values of 0.1. SD fails to approximate the OC signal (see figure 9(a)), whereas Adam is able to approach the OC function (see figure 9(b)). We again emphasize that the learning loss is proportional to the squared difference between reached state and target state $(x(T) - x^*)^2$ (see equation (21)). Learning the OC solution is not a direct learning target. Still, in the discussed example, learning OC is possible with the Adam optimizer. As described by equations (33)–(38), a gradient descent in $\boldsymbol{\theta}^{(n)}$ may induce a gradient update in $\hat{u}(t;\boldsymbol{\theta}^{(n)})$. Because the Jacobian $\mathcal{J}_{\hat{u}^{(n)}}$ is usually not straightforward to evaluate, we use $\Delta\hat{U}^{(n)}$ to study if the linearization of $\hat{u}(t;\boldsymbol{\theta}^{(n)} + \Delta\boldsymbol{\theta}^{(n)})$ in $\Delta\boldsymbol{\theta}^{(n)}$ is justified. Figure 9(c) shows that $\Delta\hat{U}^{(n)} = \int_0^T \hat{u}(t;\boldsymbol{\theta}^{(n+1)})e^{-at}\,\mathrm{d}t - \int_0^T \hat{u}(t;\boldsymbol{\theta}^{(n)})e^{-at}\,\mathrm{d}t$ (red squares) is well-described by $-\eta^{-1}b^{-1}e^{-aT}\|\Delta\boldsymbol{\theta}^{(n)}\|_2^2\left(\frac{\partial L}{\partial x(T)}\right)^{-1}$ (solid red line). The evolution of $\Delta\hat{U}^{(n)}$ under Adam (solid blue line) undergoes oscillations.

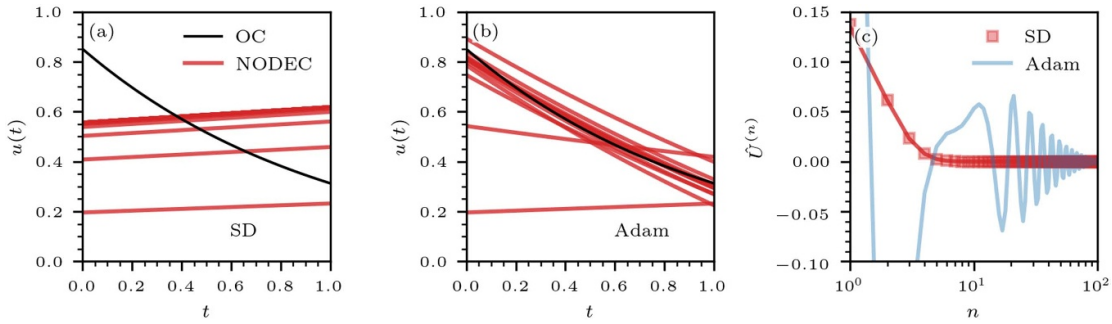

Figure 9. Learning control signals with SD and Adam. We use NODEC to control equation (29) for $x_0 = 0$ and $T = a = b = x^* = 1$. (a), (b) Learned and OC signals are indicated by solid red and black lines, respectively ((a) SD; (b) Adam). Different solid red lines correspond to control functions $\hat{u}(t;\boldsymbol{\theta}^{(n)})$ learned after different numbers of epochs n. The learning rate was set to $\eta = 0.15$. The neural network that we use in this example consists of 2 hidden layers with 6 ELU neurons each. Initial weights and biases are set to 0.1. Solid black lines indicate the OC signal (30). (c) The weighted total change $\Delta\hat{U}^{(n)}$ (see equation (39)) as a function of epochs n. Red squares (SD) and the blue solid line (Adam) are based on directly evaluating $\int_0^T \hat{u}(t;\boldsymbol{\theta}^{(n+1)})e^{-at}\,\mathrm{d}t - \int_0^T \hat{u}(t;\boldsymbol{\theta}^{(n)})e^{-at}\,\mathrm{d}t$. The solid red line is based on evaluating the right-hand side of equation (39) using neural-network parameter changes $\Delta\boldsymbol{\theta}^{(n)}$ associated with SD.

Table 1. Overview of gradient updates of neural-network parameters, control function, and control energy.

Neural-network parameters	$\boldsymbol{\theta}^{(n+1)} = \boldsymbol{\theta}^{(n)} - \eta\nabla_{\boldsymbol{\theta}^{(n)}} L$
Control function	$\hat{\mathbf{u}}(t;\boldsymbol{\theta}^{(n+1)}) = \hat{\mathbf{u}}(t;\boldsymbol{\theta}^{(n)}) - \eta\mathcal{J}_{\hat{\mathbf{u}}^{(n)}}\nabla_{\boldsymbol{\theta}^{(n)}} L$
Control energy	$E_T[\hat{\mathbf{u}}^{(n+1)}] = E_T[\hat{\mathbf{u}}^{(n)}] - \eta(\nabla_{\boldsymbol{\theta}^{(n)}} E_T[\hat{\mathbf{u}}^{(n)}])^\top \nabla_{\boldsymbol{\theta}^{(n)}} L$

5.2. Control energy regularization

The induced gradient update in the control input $\hat{\mathbf{u}}(t;\boldsymbol{\theta}^{(n)})$ and the evolution of the control energy $E_T[\hat{\mathbf{u}}]$ during training are connected via

$$E_T[\hat{\mathbf{u}}^{(n+1)}] = E_T[\hat{\mathbf{u}}^{(n)}] + \int_0^T \hat{\mathbf{u}}(t;\boldsymbol{\theta}^{(n)})^\top \mathcal{J}_{\hat{\mathbf{u}}^{(n)}}\,\mathrm{d}t\Delta\boldsymbol{\theta}^{(n)} + \frac{1}{2}\Delta\boldsymbol{\theta}^{(n)\top}\int_0^T \mathcal{J}_{\hat{\mathbf{u}}^{(n)}}^\top \mathcal{J}_{\hat{\mathbf{u}}^{(n)}}\,\mathrm{d}t\Delta\boldsymbol{\theta}^{(n)} \tag{40}$$

$$= E_T[\hat{\mathbf{u}}^{(n)}] - \eta\int_0^T \hat{\mathbf{u}}(t;\boldsymbol{\theta}^{(n)})^\top \mathcal{J}_{\hat{\mathbf{u}}^{(n)}}\,\mathrm{d}t\nabla_{\boldsymbol{\theta}^{(n)}} L + \mathcal{O}(\eta^2)$$
$$= E_T[\hat{\mathbf{u}}^{(0)}] - \eta\sum_{i=0}^{n}\int_0^T \hat{\mathbf{u}}(t;\boldsymbol{\theta}^{(i)})^\top \mathcal{J}_{\hat{\mathbf{u}}^{(i)}}\,\mathrm{d}t\nabla_{\boldsymbol{\theta}^{(i)}} L + \mathcal{O}(\eta^2)\,. \tag{41}$$

Using the identity $\nabla_{\boldsymbol{\theta}^{(n)}} E_T[\hat{\mathbf{u}}^{(n)}] = \int_0^T \mathcal{J}_{\hat{\mathbf{u}}^{(n)}}^\top \hat{\mathbf{u}}(t;\boldsymbol{\theta}^{(n)})\,\mathrm{d}t$, we obtain

$$E_T[\hat{\mathbf{u}}^{(n+1)}] = E_T[\hat{\mathbf{u}}^{(0)}] - \eta\sum_{i=0}^{n}(\nabla_{\boldsymbol{\theta}^{(i)}} E_T[\hat{\mathbf{u}}^{(i)}])^\top \nabla_{\boldsymbol{\theta}^{(i)}} L + \mathcal{O}(\eta^2)\,. \tag{42}$$

Equation (42) shows that, up to terms of order $\mathcal{O}(\eta^2)$, a gradient-descent update in $\boldsymbol{\theta}^{(n)}$ is associated with a control energy update that is equal to $(\nabla_{\boldsymbol{\theta}^{(n)}} E_T[\hat{\mathbf{u}}^{(n)}])^\top$ multiplied by $\Delta\boldsymbol{\theta}^{(n)} = -\eta\nabla_{\boldsymbol{\theta}^{(n)}} L$. Let ω denote the angle between $\nabla_{\boldsymbol{\theta}^{(n)}} E_T[\hat{\mathbf{u}}^{(n)}]$ and $\nabla_{\boldsymbol{\theta}^{(n)}} L$. If $\cos(\omega) > 0$ (i.e. $\omega < |\pi/2|$), a gradient update in $\boldsymbol{\theta}^{(n)}$ is associated with a decrease in control energy. The control energy increases for $\cos(\omega) < 0$ (i.e. $|\pi/2| < \omega < |\pi|$) and it stays constant for $\cos(\omega) = 0$ (i.e. $\omega = \pm\pi/2$).

Table 1 provides an overview of the gradient updates associated with the neural-network parameters $\boldsymbol{\theta}^{(n)}$, control function $\mathbf{u}(t;\boldsymbol{\theta}^{(n)})$, and control energy $E_T[\hat{\mathbf{u}}^{(n)}]$.

Using random projections of the loss function L [45] can help provide geometric intuition for the interplay between explicit minimization of L and implicit regularization of the control energy $E_T[\hat{\mathbf{u}}]$. To obtain two and three-dimensional projections of the 51-dimensional parameter space of the employed neural networks, we set the neural-network parameters $\boldsymbol{\theta} = \boldsymbol{\theta}^* + \alpha\boldsymbol{\delta} + \beta\boldsymbol{\eta}$ around a local optimum $\boldsymbol{\theta}^*$. Here, $\alpha, \beta \subseteq \mathbb{R}$ denote scaling parameters and $\boldsymbol{\delta}, \boldsymbol{\eta} \sim \mathcal{N}(0, \mathbb{1}_{N\times N})$ are random Gaussian vectors.

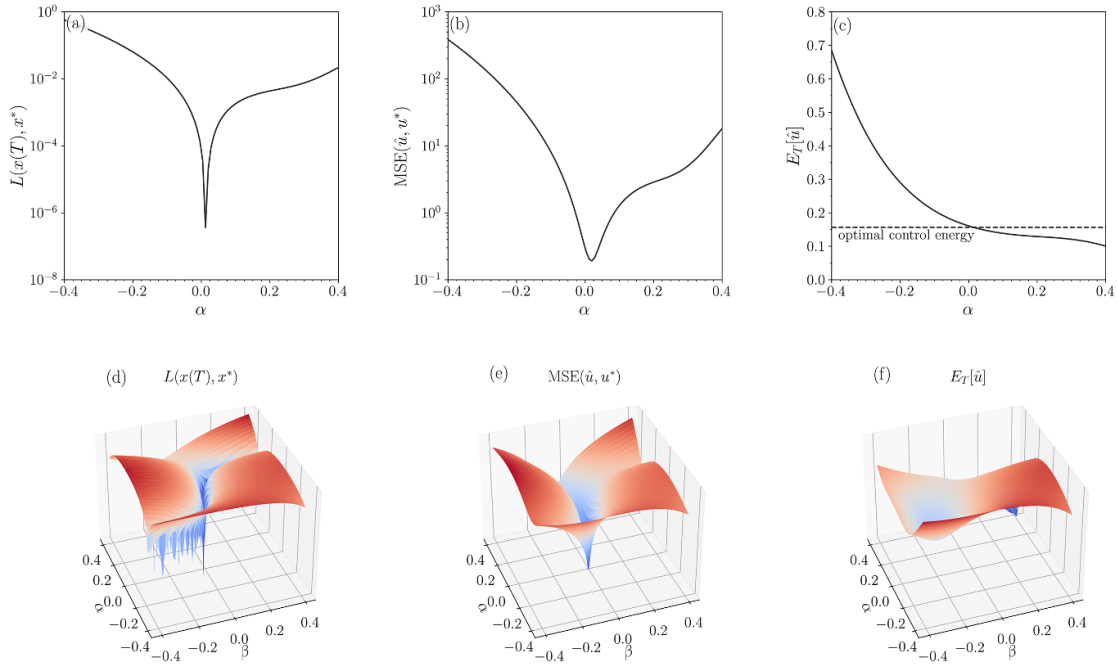

Figure 10. Loss landscapes for training with Adam. Around a local optimum $\boldsymbol{\theta}^*$ obtained after 1000 training epochs, we set the neural-network parameters $\boldsymbol{\theta} = \boldsymbol{\theta}^* + \alpha\boldsymbol{\delta} + \beta\boldsymbol{\eta}$, where $\boldsymbol{\delta}, \boldsymbol{\eta} \sim \mathcal{N}(0, \mathbb{1}_{N\times N})$. (a), (d) The loss $L(x(T), x^*)$ (see equation (20)) for $\alpha \in [-0.4, 0.4]$, $\beta = 0$ (a) and $\alpha, \beta \in [-0.4, 0.4]$ (b). (b), (e) The MSE associated with the difference between $\hat{u}(t;\boldsymbol{\theta})$ and u^*, $\mathrm{MSE}(\hat{u}, u^*)$ (see equation (43)), as a function of $\alpha \in [-0.4, 0.4]$ (b) and $\alpha, \beta \in [-0.4, 0.4]$ (e). (c), (f) The control energy $E_T[\hat{u}]$ (see equation (1)) as a function of $\alpha \in [-0.4, 0.4]$ (c) and $\alpha, \beta \in [-0.4, 0.4]$ (f). The OC energy, $1/(e^2 - 1)$, is indicated by a dashed black line in panel (c). The neural network that we use in this example consists of 2 hidden layers with 6 ELU neurons each. Parameters of the underlying dynamical system (29) are set to $x_0 = 0$ and $T = a = b = x^* = 1$. We used Adam to train the neural network and set the learning rate to $\eta = 0.15$. Initial weights and biases were set to 0.1.

For an Adam-based optimization of the OC problem (29), figure 10 shows two and three-dimensional random projections of $L(x(T), x^*)$, $E_T[\hat{u}]$, and

$$\mathrm{MSE}(u^*, \hat{u}) = \frac{1}{M}\sum_{i=1}^{M}[u^*(t_i) - \hat{u}(t_i)]^2 \approx \int_0^T [u^*(t) - \hat{u}(t)]^2 \mathrm{d}t, \tag{43}$$

quantifying the deviation of $\hat{u}$ from the optimum u^*. Here, M is the number of timesteps and $t_i = iT/M$. In the random projection shown in figure 10, we observe that Adam produces a local optimum that is associated with a small loss, $\mathrm{MSE}(u^*, \hat{u})$, and control energy. In comparison with the results obtained with SD (see figure 11), we observe that Adam produces sharper local optima. The loss projection that we show in figure 11 also suggests that SD produces two 'valleys' of local optima separated by a plateau. Escaping from a local optimum located in one valley to another optimum in the second valley may be challenging as even larger learning rates may not be able to overcome the plateau. Additional loss projections for neural networks with fewer parameters are provided in appendix C.

5.3. Constant OC approximation

In situations where the OC function is expected to only vary modestly in time, or when no good guess of the OC solution is available, one may use a constant control approximation as a baseline. For such a baseline, the underlying neural network consists of a bias term (i.e. $\hat{u}(t;\boldsymbol{\theta}^{(n)}) \equiv c^{(n)}$) and equation (33) becomes

$$c^{(n+1)} = c^{(n)} + \Delta c^{(n)}, \tag{44}$$

where $\Delta c^{(n)} = -\eta \frac{\mathrm{d}L}{\mathrm{d}c^{(n)}}$. For linear dynamics (29) with $x_0 = 0$ and $T = a = b = x^* = 1$, the loss function (35) becomes

$$L = \frac{1}{2}\left[\int_0^1 e^{-t+1}c^{(n)}\,\mathrm{d}t - 1\right]^2 = \frac{1}{2}\left[c^{(n)}(e-1) - 1\right]^2. \tag{45}$$

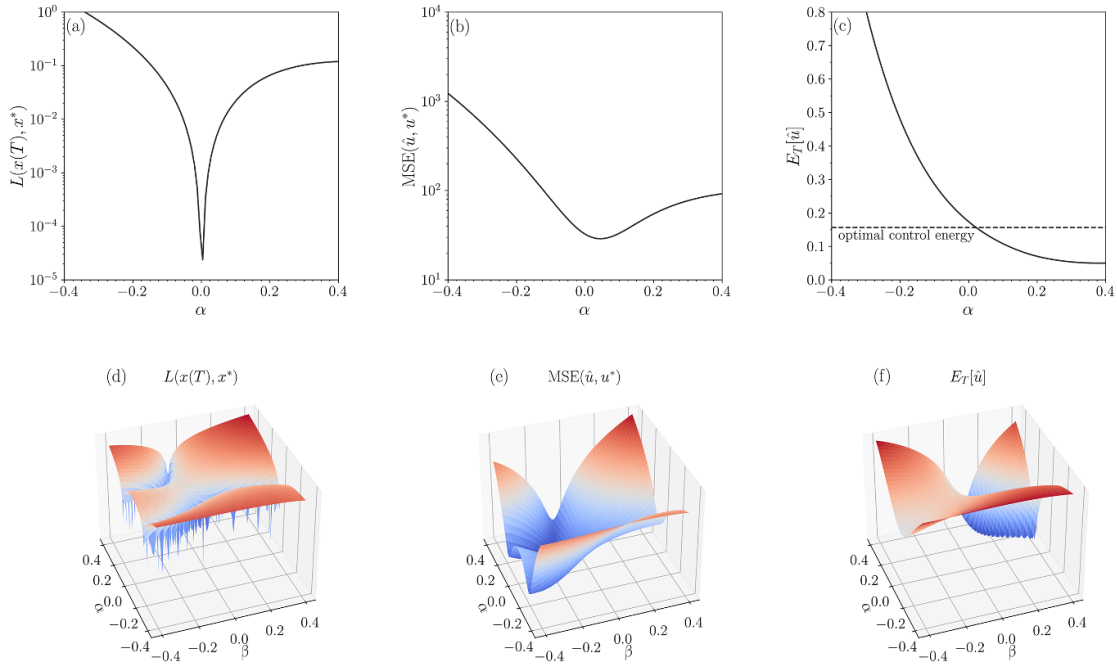

Figure 11. Loss landscapes for training with SD. Around a local optimum $\boldsymbol{\theta}^*$ obtained after 1000 training epochs, we set the neural-network parameters $\boldsymbol{\theta} = \boldsymbol{\theta}^* + \alpha\boldsymbol{\delta} + \beta\boldsymbol{\eta}$, where $\boldsymbol{\delta}, \boldsymbol{\eta} \sim \mathcal{N}(0, \mathbb{1}_{N\times N})$. (a), (d) The loss $L(x(T), x^*)$ (see equation (20)) as a function of $\alpha \in [-0.4, 0.4]$, $\beta = 0$ (a) and $\alpha, \beta \in [-0.4, 0.4]$ (b). (b), (e) The MSE associated with the difference between $\hat{u}(t;\boldsymbol{\theta})$ and u^*, $\mathrm{MSE}(\hat{u}, u^*)$ (see equation (43)), as a function of $\alpha \in [-0.4, 0.4]$ (b) and $\alpha, \beta \in [-0.4, 0.4]$ (e). (c), (f) The control energy $E_T[\hat{u}]$ (see equation (1)) as a function of $\alpha \in [-0.4, 0.4]$ (c) and $\alpha, \beta \in [-0.4, 0.4]$ (f). The OC energy, $1/(e^2 - 1)$, is indicated by a dashed black line in panel (c). The neural network that we use in this example consists of 2 hidden layers with 6 ELU neurons each. Parameters of the underlying dynamical system (29) are set to $x_0 = 0$ and $T = a = b = x^* = 1$. We used SD to train the neural network and set the learning rate to $\eta = 0.15$. Initial weights and biases were set to 0.1.

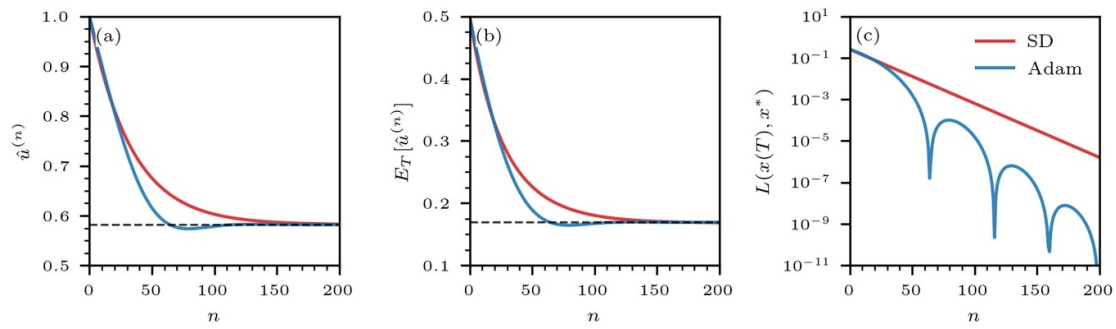

Figure 12. Approximating OC by a constant. (a) The constant control input $\hat{u}(t;\boldsymbol{\theta}^n) \equiv c^{(n)}$ as a function of the number of training epochs n. The dashed black line indicates the optimum $c^{(\infty)} = 1/(e-1) \approx 0.58$. (b) The control energy $E_T[\hat{u}^{(n)}]$ as a function of n. In the limit $n \to \infty$, the control energy approaches $1/2(e-1)^{-2} \approx 0.17$ (dashed black line). (c) The MSE loss $L(x(T), x^*)$ as a function of n. Solid red and blue lines represent SD and Adam solutions, respectively. We set the learning rate to $\eta = 10^{-2}$.

In the limit $n \to \infty$, the control function approaches $c^* = 1/(e-1)$ and in each gradient-descent step, the control energy changes according to

$$E_T[\hat{u}^{(n+1)}] = E_T[\hat{u}^{(n)}] - \eta c^{(n)} \frac{\mathrm{d}L}{\mathrm{d}c^{(n)}}. \tag{46}$$

$$\begin{aligned} E_T[\hat{u}^{(n+1)}] &= E_T[\hat{u}^{(n)}] - \eta c^{(n)} \int_0^1 e^{-t+1}\,\mathrm{d}t \left[\int_0^1 e^{-t+1} c^{(n)}\,\mathrm{d}t - 1\right] \\ &= E_T[\hat{u}^{(n)}] - \eta c^{(n)}(e-1)\left[c^{(n)}(e-1) - 1\right]. \end{aligned} \tag{47}$$

Figure 12 shows the evolution of $\hat{u}(t;\boldsymbol{\theta}^{(n)}) \equiv c^{(n)}$, $E_T[\hat{u}^{(n)}]$, and $L(x(T), x^*)$. Solid red and blue lines represent solutions that were obtained with SD and Adam, respectively. Dashed black lines in figures 12(a) and (b) indicate the OC solutions. The convergence behavior of Adam is non-monotonic. Between 60 and 70

training epochs, Adam achieves control energy values smaller than $E_T[\hat{u}^{(\infty)}] = 1/2(e-1)^{-2}$, owing to an increase in the loss $L(x(T), x^*)$. The ratio between $E_T[\hat{u}^{(\infty)}] = 1/2(e-1)^{-2}$ and the OC energy is 1.08, indicating that the constant control approximation achieves a control energy that is close to that of the corresponding OC solution.

The constant control baseline that we derived in this section also appears to be close to possible local optima that are learned by NODEC. For example, for certain learning rates and initial values of neural-network parameters, Adam approaches solutions that are similar to the constant baseline (see figures 6(b) and (e)). We also observe a similar behavior for steepest-descent learning (see figure 9(a)). One of the loss projections in appendix C shows that SD approaches a local optimum that corresponds to a constant OC approximation. A solution that is closer to OC and which was not found by SD is also visible in the shown projections in appendix C.

6. Discussion

Optimal control problems arise in various applications and often admit analytical solutions only under special assumptions (e.g. for linear systems [20]). For non-linear control problems, different direct and indirect numerical methods have been developed to approximate OC solutions. The loss function underlying OC problems consists of a final cost (e.g. difference between reached and target state) and an integrated cost (e.g. control energy). To achieve a certain tradeoff between these two cost contributions, one has to find suitable Lagrange multipliers (see equation (5)), which might prove challenging in practice. Neural ODE controllers provide an alternative to standard direct and indirect OC solvers in that they are able to implicitly learn near-optimal control functions [33, 34] if the underlying hyperparameters are chosen appropriately. The problem of optimizing Lagrange multipliers in OC loss functionals can be recast into a hyperparameter optimization problem, with the goal being that the basin of attraction of the desired OC solution in the underlying loss landscape is sufficiently large.

To study the ability of neural ODE controllers to perform implicit control energy regularization, we have analyzed the influence of backpropagation protocols, parameter initialization, optimizers, and activation functions on the learned control solutions. Using analytical and numerical arguments, we were able to identify different features that can help NODEC designers to efficiently perform hyperparameter optimization and learn near-optimal control solutions.

First, TBPTT has been shown to be computationally more efficient in terms of iterations per unit time than its untruncated counterpart. Although TBPTT protocols only provide an approximation of the actual gradient update, our results suggest that they can achieve loss values and control solutions that are similar to those obtained with untruncated gradient updates. Future work may evaluate the computational advantages of TBPTT over BPTT in larger-scale problems.

Second, a useful starting point for certain control problems may be a constant controller that can be represented by a bias term. Constant OC approximations appeared as local optima in deeper architectures that we used in our numerical experiments. For high-dimensional control tasks studied in earlier work [34], NODEC has also learned a constant OC approximation that provided better performance than solutions found by an adjoint-gradient method (i.e. an indirect OC solver). Increasing the width and depth of the underlying ANN can substantially improve the performance of NODEC in implicitly learning near-optimal control solutions as shown in section 4. Between four to six hidden fully-connected layers and about five to ten neurons per hidden layer have shown good performance in different control tasks. In accordance with earlier numerical work [34], our results also highlight the importance of appropriate initialization protocols. Large weight and bias values may often be associated with loss values that are far away from the basin of attraction of a near-OC solution. In this context it may be worthwhile to study combinations of neural-network parameter initialization protocols and momentum schedulers [54].

Third, adapative gradient methods like Adam have been shown to be able to learn near-OC functions in problems where SD fails to do so. These results therefore contribute to the discussion of the advantages of adaptive optimizers over standard gradient descent techniques [55–57]. Two and three dimensional loss projections provide insights into hyperparameter-dependent geometric features of the loss and control energy landscapes and may reveal the existence of better nearby local optima.

There are different possibilities for future work. Studying the ability of neural ODE controllers to implicitly learn control functions for loss functionals different from equation (5) can help provide insights into the properties of generalized implicit regularization approaches. It would also be interesting to use dynamical systems approaches to study the behavior of adaptive optimizers akin to our steepest-descent analysis in section 3.1. Instead of using ANNs to approximate control functions, control problems can function as a tool to better understand optimization dynamics within deep ANNs that are otherwise mainly studied in terms of image classification. Finally, an important direction for future work is to control complex

real-world systems by combining neural ODE controllers with machine-learning methods [58, 59] that can learn dynamical models from time series data.

Data availability statement

The data and source codes that support the findings of this study are openly available at https://gitlab.com/ComputationalScience/near-optimal-control.

Acknowledgments

Lucas Böttcher thanks Mingtao Xia for helpful discussions. Thomas Asikis acknowledges that the research was supported by NCCR Automation, a National Centre of Competence in Research, funded by the Swiss National Science Foundation (Grant No. 180545). Both authors would like to especially thank Nino Antulov-Fantulin for the helpful inputs, discussions, and guidance.

Appendix A. Moving particle subject to friction

The goal is to minimize

$$W[\mathbf{x}, u] = \int_0^1 v(t)u(t)\,\mathrm{d}t \tag{A.1}$$

subject to

$$\begin{aligned}
&\dot{x}(t) = v(t)\\
&\dot{v}(t) = -v(t) + u(t)\\
&v(t) \geqslant 0\\
&0 \leqslant u(t) \leqslant 2\\
&(x(0), v(0)) = (0, 1)\\
&(x(1), v(1)) = (1, 1)\,,
\end{aligned} \tag{A.2}$$

where $\mathbf{x}(t) = (x(t), v(t))^\top$ [26]. This control problem describes the movement of a particle under friction from $x(0) = 0$ to $x(1) = 1$ minimizing the work done. The control input $u(t)$ corresponds to a force that acts on the particle and the OC in this example is $u^*(t) = 1$.

To solve this control problem with NODEC, we represent $u(t)$ by a neural network $\hat{u}(t;\boldsymbol{\theta})$ with H layers and N_H neurons per layer. We train NODEC using Adam and the MSE loss and set $L(\mathbf{x}(T), \mathbf{x}^*) = \|\mathbf{x}(t) - \mathbf{x}^*\|_2^2$. That is, we do not explicitly account for control bounds and $W[\mathbf{x}, u]$. Figures A1(a) and (b) show the evolution of $x(t)$ and $u(t)$. OC and NODEC solutions are represented by solid grey and red lines, respectively. The neural network that we use in this example has $H = 64$ layers with $N_H = 6$ ELU neurons per layer. Bias terms are included in every layer.

To study the effect of different neural network structures and activations on the learning performance, we first set the number of neurons per layer $N_H = 6$ and vary the number of layers H from 2 to 64. All weights and biases are initially set to a value of 10^{-2}. We perform numerical experiments for both ELU and ReLU activations. We train NODEC for 100 epochs using Adam with a learning rate $\eta = 0.5 \times 10^{-2}$ and we evaluate the best model. We show the corresponding loss and MSE values after training in figures A1(c) and (d). For $H \geqslant 8$, the loss approaches values smaller than 10^{-7} and the MSE drops to values below 10^{-4}. For one layer and varying N_H, the neural network is not able approximate the OC solution. Our results are almost identical for both ReLU and ELU activations.

In addition to the prior numerical experiment, we now also directly account for the additional loss term $W[\mathbf{x}, u]$ and optimize

$$L(\mathbf{x}(T), \mathbf{x}^*) + \mu W[\mathbf{x}, u]\,, \tag{A.3}$$

where μ is a Lagrange multiplier that models the influence of $W[\cdot]$ on the total loss. To initialize weights without additional parameter optimization, we use the Kaiming uniform initializer [60], and we initially set all biases to a value of 10^{-2}. The neural network that we use to optimize equation (A.3) consists of $H = 8$ fully connected linear layers with $N_H = 6$ ELU neurons each. This way of setting up the network architecture ensures that NODEC is able to represent the constant control function. We use Adam with a learning rate of

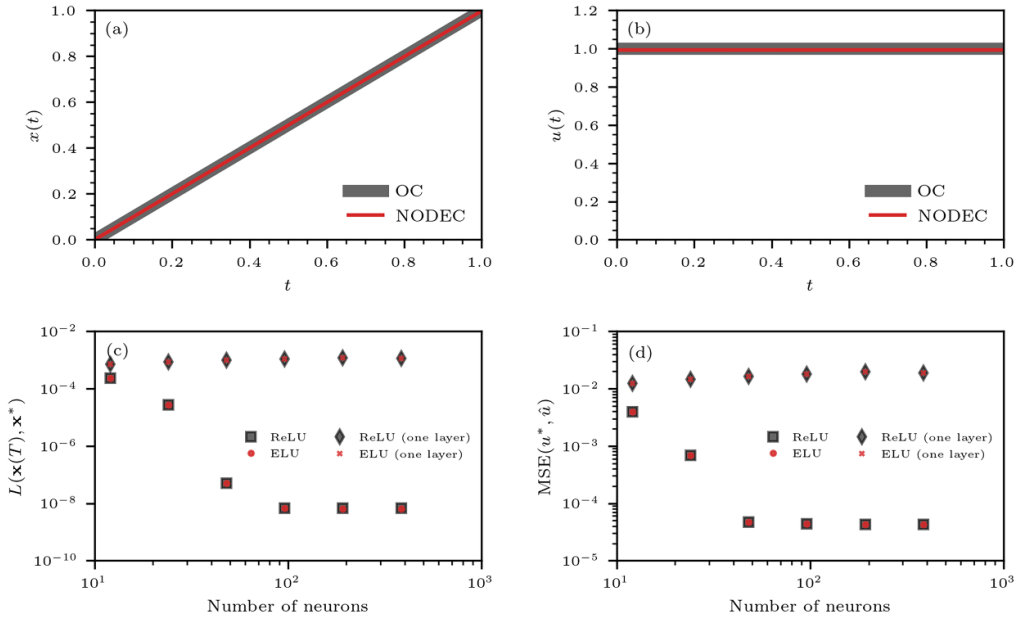

Figure A1. Controlling a moving particle that is subject to friction. (a), (b) Evolution of the state $x(t)$ and control function $u(t)$. OC and NODEC-based solutions are shown in grey and red, respectively. (c), (d) Loss $L(\mathbf{x}(T),\mathbf{x}^*) = \|\mathbf{x}(t) - \mathbf{x}^*\|_2^2$ and $\mathrm{MSE}(\mathbf{u}^*, \hat{\mathbf{u}})$ for different activations and architectures as a function of the number of neurons.

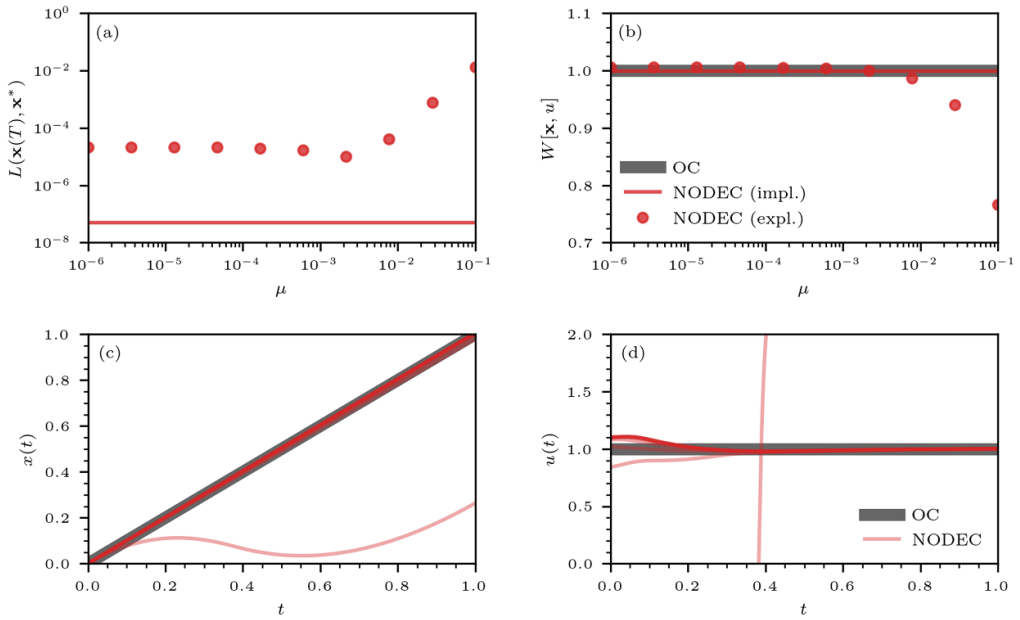

Figure A2. Lagrange multiplier optimization. (a), (b) The loss $L(\mathbf{x}(T),\mathbf{x}^*) = \|\mathbf{x}(T) - \mathbf{x}^*\|_2^2$ and work $W[\mathbf{x},u]$ (see equation (A.1)) for different values of μ (see equation (A.3)). Red disks are associated with a direct optimization of both $L(\mathbf{x}(T),\mathbf{x}^*)$ and $W[\mathbf{x},u]$ while solid red lines indicate implicit regularization solutions. (c), (d) Different values of μ are associated with different NODEC-based solutions of $x(t)$ and $\hat{u}(t;\boldsymbol{\theta})$ (solid red lines). As a reference, we also show the corresponding OC solutions (solid grey line).

10^{-1} to train the neural network for 100 epochs, and we evaluate the best model. In addition to the uniform weight initialization, the learning rate of $\eta = 10^{-1}$ was chosen to model a not fully optimized hyperparameter set. Otherwise, near-optimal solutions would be learned automatically as in the previous example without optimizing the multiplier μ. As shown in figures A2(a) and (b), the loss component $L(\mathbf{x}(T),\mathbf{x}^*)$ approaches a minimum for $\mu \approx 2 \times 10^{-3}$. The corresponding value of $W[\mathbf{x},u]$ is reasonably close to the optimal solution. However, figure A2(a) also shows that the implicit regularization of $W[\mathbf{x},u]$ can achieve smaller loss values and similar values of $W[\mathbf{x},u]$. Figures A2(c) and (d) shows different NODEC solutions (solid red lines) that are associated with different multiplier values μ. Although almost all solutions are aligned with the OC solution in terms of the evolution of $x(t)$, variations in μ produce visible deviations of $\hat{u}(t;\boldsymbol{\theta})$ from $u^*(t)$.

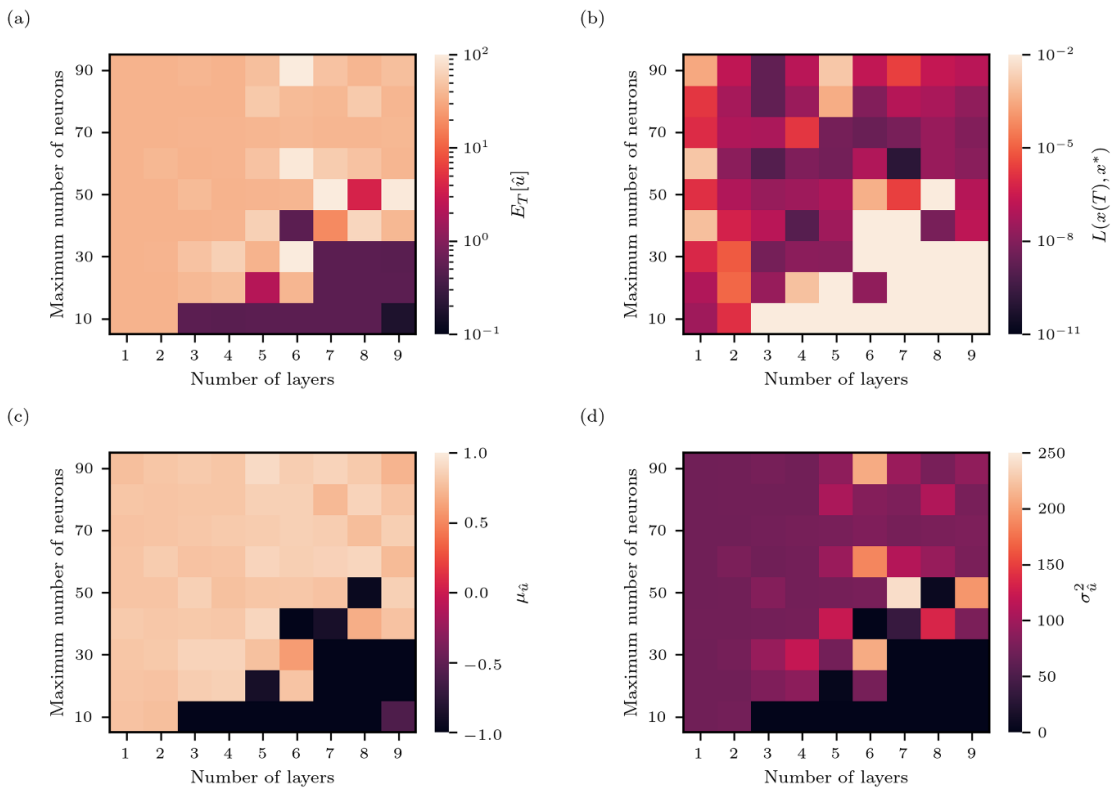

Figure B1. Effect of variations in number of layers and neurons per layer on the ability of NODEC to learn a time-dependent control signal for the two-dimensional flow (11). The underlying architectures use bias terms and leaky ReLU activations in all hidden layers. Each ANN is trained for 500 epochs using Adam and a learning rate $\eta = 3 \times 10^{-3}$. We set the number of neurons per layer to be the greatest integer less than or equal to the maximum number of neurons (vertical axes) divided by the number of layers (horizontal axes). The OC energy is approximately 34. Heatmaps show the (a) control energy $E_T[\hat{u}]$, (b) loss $L(x(T), x^*)$, (c) mean control signal over time $\mu_{\hat{u}}$, and (d) control signal variance over time, $\sigma^2_{\hat{u}}$.

Appendix B. Architectures for time-dependent control in two dimensions

For learning the time-dependent control of the two-dimensional flow (11), we use an ANN with leaky ReLU activations, between one to nine layers, and up to 90 neurons. Weights and biases are initially distributed according to $\mathcal{U}(-\sqrt{k}, \sqrt{k})$, where k denotes the number of input features of a certain layer. Figure B1(a) shows that smaller control energies can be achieved as the number of layers increases. For six layers and 18 neurons in total, we observe that the ANN controller is able to implicitly learn a solution with a small loss and a control energy that is close to that of the optimal solution. However, in general, the loss may increase with the number of layers (see figure B1(b)). Increasing the number of layers while keeping a low number of neurons seems to affect convergence as most ANNs seem to converge to a non-optimal constant control—the corresponding variances in the bottom right corner of figure B1(d) are close to 0 (darker color). To summarize, implicit regularization of control energy can be achieved by selecting appropriate hyperparameters (e.g. six layers and 18 neurons in total).

Appendix C. Loss projections

To complement our geometric analysis of implicit energy regularization (see section 5), we provide additional loss, MSE, and control energy visualizations in figures C1 and C2. The control task is the same as in section 5. We aim at controlling the one-dimensional flow (29) with $x_0 = 0$ and $T = a = b = x^* = 1$. The neural network that we use here consists of 2 hidden layers with 5 ELU neurons each. Weights and biases are initialized to 0.1. The total number of neural-network parameters is 46 instead of 51 as in section 5.

Figure C1 shows two and three-dimensional loss projections around a local optimum found by Adam. The local optimum is associated with a small loss, MSE, and control energy. Other surrounding local optima are not closer to OC solution.

For the same neural network and initial conditions, we observe in figure C2 that SD gets stuck in a local optimum that is associated with a larger MSE than the optimum found by Adam. Another optimum that is visible in the projections in figure C2 has a significantly smaller MSE and control energy than the one found

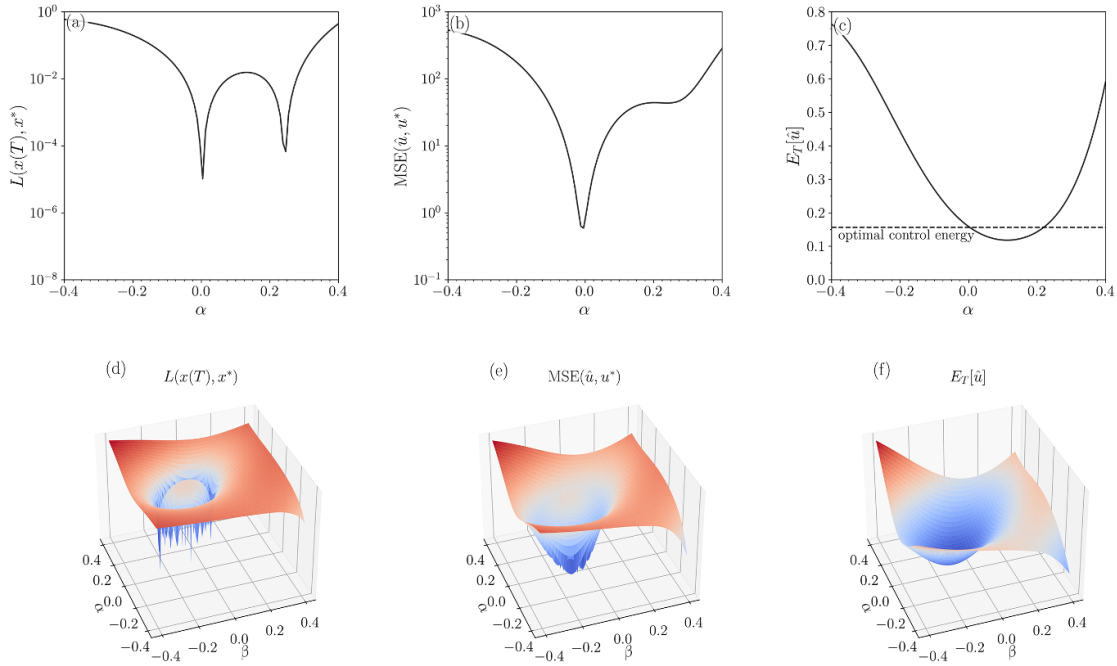

Figure C1. Loss landscapes for training with Adam. Loss landscapes for training with Adam. Around a local optimum $\boldsymbol{\theta}^*$ obtained after 1000 training epochs, we set the neural-network parameters $\boldsymbol{\theta} = \boldsymbol{\theta}^* + \alpha\boldsymbol{\delta} + \beta\boldsymbol{\eta}$, where $\boldsymbol{\delta}, \boldsymbol{\eta} \sim \mathcal{N}(0, \mathbb{1}_{N\times N})$. (a), (d) The loss $L(x(T), x^*)$ (see equation (20)) as a function of $\alpha \in [-0.4, 0.4]$, $\beta = 0$ (a) and $\alpha, \beta \in [-0.4, 0.4]$ (b). (b), (e) The MSE associated with the difference between $\hat{u}(t;\boldsymbol{\theta})$ and u^*, $\mathrm{MSE}(\hat{u}, u^*)$ (see equation (43)), as a function of $\alpha \in [-0.4, 0.4]$ (b) and $\alpha, \beta \in [-0.4, 0.4]$ (e). (c), (f) The control energy $E_T[\hat{u}]$ (see equation (1)) as a function of $\alpha \in [-0.4, 0.4]$ (c) and $\alpha, \beta \in [-0.4, 0.4]$ (f). The OC energy, $1/(e^2 - 1)$, is indicated by a dashed black line in panel (c). The neural network that we use in this example consists of 2 hidden layers with 5 ELU neurons each. Parameters are set to $x_0 = 0$ and $T = a = b = x^* = 1$. We used Adam to train NODEC and we set the learning rate to $\eta = 0.1733$. Initial weights and biases are set to 0.1.

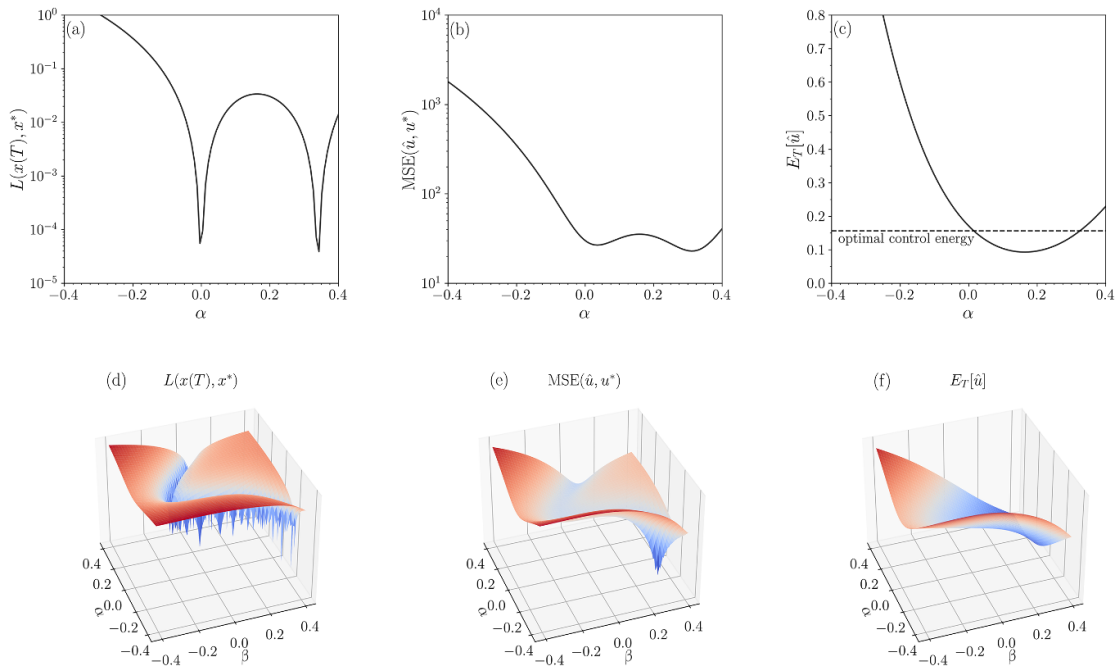

Figure C2. Loss landscapes for training with SD. Loss landscapes for training with Adam. Around a local optimum $\boldsymbol{\theta}^*$ obtained after 1000 training epochs, we set the neural-network parameters $\boldsymbol{\theta} = \boldsymbol{\theta}^* + \alpha\boldsymbol{\delta} + \beta\boldsymbol{\eta}$, where $\boldsymbol{\delta}, \boldsymbol{\eta} \sim \mathcal{N}(0, \mathbb{1}_{N\times N})$. (a), (d) The loss $L(x(T), x^*)$ (see equation (20)) as a function of $\alpha \in [-0.4, 0.4]$, $\beta = 0$ (a) and $\alpha, \beta \in [-0.4, 0.4]$ (b). (b), (e) The MSE associated with the difference between $\hat{u}(t;\boldsymbol{\theta})$ and u^*, $\mathrm{MSE}(\hat{u}, u^*)$ (see equation (43)), as a function of $\alpha \in [-0.4, 0.4]$ (b) and $\alpha, \beta \in [-0.4, 0.4]$ (e). (c), (f) The control energy $E_T[\hat{u}]$ (see equation (1)) as a function of $\alpha \in [-0.4, 0.4]$ (c) and $\alpha, \beta \in [-0.4, 0.4]$ (f). The OC energy, $1/(e^2 - 1)$, is indicated by a dashed black line in panel (c). The neural network that we use in this example consists of 2 hidden layers with 5 ELU neurons each. Parameters are set to $x_0 = 0$ and $T = a = b = x^* = 1$. We used SD to train NODEC and we set the learning rate to $\eta = 0.1733$. Initial weights and biases are set to 0.1.

by SD. However, in the shown example, SD converges toward a constant control approximation (see section 5.3) and not towards the local optimum that is closer to the OC solution

ORCID iDs

Lucas Böttcher https://orcid.org/0000-0003-1700-1897
Thomas Asikis https://orcid.org/0000-0003-0163-4622

References

[1] Bertsekas D 2012 *Dynamic Programming and Optimal Control* 4th edn (St Belmont, MA: Athena Scientific)
[2] Jarrah A, Vastani H, Duca K and Laubenbacher R 2004 An optimal control problem for *in vitro* virus competition *2004 43rd IEEE Conf. on Decision and Control (CDC) (IEEE Cat. No. 04CH37601)* vol 1 (IEEE) pp 579–84
[3] Lenhart S and Workman J T 2007 *Optimal Control Applied to Biological Models* (London: Chapman and Hall/CRC)
[4] Laubenbacher R, Hinkelmann F and Oremland M 2013 Agent-based models and optimal control in biology *Mathematical Concepts and Methods in Modern Biology* ed R Robeva and T Hodge (Boston, MA: Academic Boston) pp 143–78
[5] Ledzewicz U, Aghaee M and Schättler H 2016 Optimal control for a SIR epidemiological model with time-varying populations *2016 IEEE Conf. on Control Applications (CCA)* (IEEE) pp 1268–73
[6] Konstorum A, Vella A T, Adler A J and Laubenbacher R C 2017 *J. R. Soc. Interface* **14** 20170150
[7] An G, Fitzpatrick B, Christley S, Federico P, Kanarek A, Neilan R M, Oremland M, Salinas R, Laubenbacher R and Lenhart S 2017 *Bull. Math. Biol.* **79** 63–87
[8] Choi W and Shim E 2021 *J. Theor. Biol.* **512** 110568
[9] Sharomi O and Malik T 2017 *Ann. Oper. Res.* **251** 55–71
[10] Mabuchi H 2009 *New J. Phys.* **11** 105044
[11] Dong D and Petersen I R 2010 *IET Control Theory Appl.* **4** 2651–71
[12] Schäfer F, Sekatski P, Koppenhöfer M, Bruder C and Kloc M 2021 Control of stochastic quantum dynamics by differentiable programming *Mach. Learn.: Sci. Technol.* **2** 035004
[13] Minciardi R and Sacile R 2011 *IEEE Intell. Syst.* **6** 126–33
[14] Bienstock D 2011 Optimal control of cascading power grid failures *2011 50th IEEE Conf. on Decision and Control and European Control Conf.* (IEEE) pp 2166–73
[15] Böttcher L, Asikis T and Fragkos I 2022 (arXiv:2201.06126)
[16] Van Groesen E and Molenaar J 2007 *Continuum Modeling in the Physical Sciences* (Philadelphia, PA: SIAM)
[17] Wang L and Xu Y 2018 *Math. Control. Relat. Fields* **8** 1001
[18] Pontryagin L S 1987 *Mathematical Theory of Optimal Processes* (Boca Raton, FL: CRC Press)
[19] Speyer J L and Jacobson D H 2010 *Primer on Optimal Control Theory* (Philadelphia, PA: SIAM)
[20] Yan G, Ren J, Lai Y C, Lai C H and Li B 2012 *Phys. Rev. Lett.* **108** 218703
[21] Zhou X 1990 *J. Optim. Theory Appl.* **65** 363–73
[22] Bellman R E and Dreyfus S E 1962 *Applied Dynamic Programming* (Princeton, NJ: Princeton University Press)
[23] Kaiser E, Kutz J N and Brunton S L 2021 *Mach. Learn.: Sci. Technol.* **2** 035023
[24] Oberle H J and Grimm W 2001 *BNDSCO: A Program for the Numerical Solution of Optimal Control Problems* (Hamburg: Institut für Angewandte Mathematik der Universität Hamburg)
[25] Bock H G and Plitt K J 1984 *IFAC Proc. Vol.* **17** 1603–8
[26] Gong Q, Kang W and Ross I M 2006 *IEEE Trans. Autom. Control* **51** 1115–29
[27] Kang W and Bedrossian N 2007 *SIAM News* **40** 1–3
[28] Fahroo F and Ross I M 2008 Advances in pseudospectral methods for optimal control *AIAA Guidance, Navigation and Control Conf. and Exhibit* p 7309
[29] Kang W and Gong Q 2022 *SIAM J. Control Optim.* **60** 786–813
[30] Wang Y J and Lin C T 1998 *IEEE Trans. Neural Netw.* **9** 294–307
[31] Chen R T Q, Rubanova Y, Bettencourt J and Duvenaud D 2018 Neural ordinary differential equations *Advances in Neural Information Processing Systems 31: Annual Conf. on Neural Information Processing Systems 2018, NeurIPS 2018* (*Montréal, Canada, 3–8 December 2018*) pp 6572–83
[32] Cuchiero C, Larsson M and Teichmann J 2020 *SIAM J. Math. Data Sci.* **2** 901–19
[33] Asikis T, Böttcher L and Antulov-Fantulin N 2022 *Phys. Rev. Res.* **4** 013221
[34] Böttcher L, Antulov-Fantulin N and Asikis T 2022 *Nat. Commun.* **13** 333
[35] Lewis F, Jagannathan S and Yesildirak A 2020 *Neural Network Control of Robot Manipulators and Non-Linear Systems* (Boca Raton, FL: CRC Press)
[36] Baydin A G, Pearlmutter B A, Radul A A and Siskind J M 2018 *J. Mach. Learn. Res.* **18** 1–43
[37] Novati G, Mahadevan L and Koumoutsakos P 2019 *Phys. Rev. Fluids* **4** 093902
[38] Abu-Khalaf M and Lewis F L 2005 *Automatica* **41** 779–91
[39] Mizutani E and Dreyfus S E 2004 Two stochastic dynamic programming problems by model-free actor-critic recurrent-network learning in non-Markovian settings *2004 IEEE Int. Joint Conf. on Neural Networks (IEEE Cat. No. 04CH37541)* vol 2 (IEEE) pp 1079–84
[40] Williams R J and Peng J 1990 *Neural Comput.* **2** 490–501
[41] Werbos P J 1990 *Proc. IEEE* **78** 1550–60
[42] Feldkamp L and Puskorius G 1993 Neural network control of an unstable process *Proc. 36th Midwest Symp. on Circuits and Systems* (IEEE) pp 35–40
[43] Kingma D P and Ba J 2014 arXiv:1412.6980
[44] Goodfellow I J and Vinyals O 2015 Qualitatively characterizing neural network optimization problems *3rd Int. Conf. on Learning Representations, ICLR 2015 (San Diego, CA, USA, 7–9 May 2015) (Conf. Track Proc.)* ed Y Bengio and Y LeCun

[45] Li H, Xu Z, Taylor G, Studer C and Goldstein T 2018 Visualizing the loss landscape of neural nets *Advances in Neural Information Processing Systems 31: Annual Conf. on Neural Information Processing Systems 2018 (Montréal, Canada, 3–8 December 2018) (NeurIPS 2018)* ed S Bengio, H M Wallach, H Larochelle, K Grauman, N Cesa-Bianchi and R Garnett pp 6391–401
[46] Hao Y, Dong L, Wei F and Xu K 2019 Visualizing and understanding the effectiveness of BERT *Proc. 2019 Conf. on Empirical Methods in Natural Language Processing and the 9th Int. Joint Conf. on Natural Language Processing EMNLP-IJCNLP 2019 (Hong Kong, China, 3–7 November 2019)* ed K Inui, J Jiang, V Ng and X Wan (Association for Computational Linguistics) pp 4141–50
[47] Wu D, Xia S T and Wang Y 2020 *Advances in Neural Information Processing Systems* vol 33 pp 2958–69
[48] Horoi S, Huang J, Rieck B, Lajoie G, Wolf G and Krishnaswamy S 2022 Exploring the geometry and topology of neural network loss landscapes *Advances in Intelligent Data Analysis XX—20th Int. Symp. on Intelligent Data Analysis, IDA 2022 (Rennes, France, 20–22 April 2022) (Proc. Lecture Notes in Computer Science* vol 13205*)* ed T Bouadi, E Fromont and E Hüllermeier (Springer) pp 171–84
[49] Böttcher L and Wheeler G 2022 Visualizing high-dimensional loss landscapes with Hessian directions (arXiv:2208.13219 [cs.LG])
[50] Böttcher L and Asikis T 2022 Gitlab repository (available at: https://gitlab.com/ComputationalScience/near-optimal-control)
[51] Ito K and Kunisch K 2008 *Lagrange Multiplier Approach to Variational Problems and Applications* (Philadelphia, PA: SIAM)
[52] Thorne J D and Hall C D 1996 *J. Guid. Control Dyn.* **19** 283–8
[53] Lewis F L, Vrabie D and Syrmos V L 2012 *Optimal Control* (Hoboken, NJ: Wiley)
[54] Sutskever I, Martens J, Dahl G and Hinton G 2013 On the importance of initialization and momentum in deep learning *Proc. 30th Int. Conf. on Machine Learning, {ICML} 2013* (*Atlanta, GA, USA*, *16–21 June 2013*) (PMLR) pp 1139–47
[55] Hardt M, Recht B and Singer Y 2016 Train faster, generalize better: stability of stochastic gradient descent *Proc. of the 33rd Int. Conf. on Machine Learning, (ICML) 2016* (*New York, USA*, *19–24 June 2016*) (PMLR) pp 1225–34
[56] Wilson A C, Roelofs R, Stern M, Srebro N and Recht B 2017 The Marginal value of adaptive gradient methods in machine learning *Advances in Neural Information Processing Systems 30: Annual Conf. on Neural Information Processing Systems 2017* (*Long Beach, CA*, *4–9 December 2017*) pp 4148–58
[57] Choi D, Shallue C J, Nado Z, Lee J, Maddison C J and Dahl G E 2020 On empirical comparisons of optimizers for deep learning *Int. Conf. on Learning Representations* (*2019*) (arXiv:1910.05446)
[58] Hoffmann M *et al* 2021 *Mach. Learn.: Sci. Technol.* **3** 015009
[59] Brunton S L and Kutz J N 2022 *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems and Control* (Cambridge: Cambridge University Press)
[60] He K, Zhang X, Ren S and Sun J 2015 Delving deep into rectifiers: surpassing human-level performance on ImageNet classification *Proc. IEEE Int. Conf. on Computer Vision* pp 1026–34