Structure-Regularized Neural Networks for Hamilton-Jacobi-Bellman Optimal Control Problems

Anonymous Authors[1]

Abstract

Recent deep learning advances enable scaling solutions to Hamilton-Jacobi-Bellman (HJB) equations for higher-dimensional systems, facilitating optimal control policy synthesis across diverse applications. However, the flexibility in deep models introduces training complexities, requiring the use of various training heuristics. In this work, we investigate the challenges of training deep models in optimal control problems by drawing insights from the matrix-algebraic Riccati equation in infinite-time Linear Quadratic Regular (LQR) problems. We proposed a structure-regularized positive-definite neural network that guarantees learning the unique admissible value function, simplifying the training process. We conduct experiments on various classical nonlinear optimal control problems, demonstrating that our methods achieve high-quality solutions.

1. Introduction

Dynamic programming (Bellman, 1954; Bertsekas, 2012) and Hamilton-Jacobi-Bellman (HJB) equations form a solid theoretical foundation for analyzing optimal control problems and deriving feedback policies for various applications in engineering, economy and management (Sundar & Shiller, 1997; Björk et al., 2014). However, solving the HJB equation is challenging, and many methods (Mitchell & Templeton, 2005; Beard et al., 1997) encounter exponential computational complexity as the system dimension increases, commonly known as the "curse of dimensionality" (Bellman & Kalaba, 1959).

Recently, deep learning approaches have been introduced and demonstrate success in solving HJB equations up to hundreds of dimensions (Han et al., 2018; Darbon et al., 2020). However, the training process becomes more challenging when applying these to more complex systems. Specifically, when extending learning-based methods to general nonlinear dynamics, warm-up, and curriculum training are often required (Lutter et al., 2020; Bansal & Tomlin, 2021) which prolongs the training time. In addition, the performance of these algorithms become highly sensitive to various hyperparameter choices, and properly-tuned learning parameters are needed to achieve good results (Nakamura-Zimmerer et al., 2021).

In this work, we investigate why training neural networks are difficult for solving HJB equations in optimal control. We demonstrate that, even for a simple LQR problem, the solutions to the associated HJB equation can be infinitely many. Moreover, unlike reward-shaping techniques used in reinforcement learning, where different value functions result in unchanged policies (Ng et al., 1999), we demonstrate that most solutions to the HJB equation are ill-defined and lead to policies that make the system unstable. Prior literature (Tedrake, 2023; Smears, 2018) has shown that if we further constrain the solution to be positive-definite i.e. the output of the solution(value function) is consistently positive for any argument except at the origin (where it is zero), then the solution is unique to the given HJB equations, leading to the optimal policy. For LQR problems, positive-definite solutions can be obtained by Schur-decomposition (Laub, 1979) or iterative Newton method (Guo & Higham, 2007). However, these methods are hard to extend to general nonlinear dynamics, especially using deep neural models for value estimation. Without preserving the positive-definiteness, we argue that it is challenging for a generic neural network to learn the unique admissible solution among other ill-defined solutions, making training difficult.

To address the training difficulty in solving HJB equations for optimal control, we propose a structure-regularized positive-definite architecture that provably learns the unique admissible solution (rejecting all ill-defined solutions by construction) to the HJB equation. We experiment with several challenging nonlinear control problems and demonstrate that our method can achieve good performance directly through random initialization, requiring fewer computational iterations to converge.

The paper is organized as follows: Section 2, we provide

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

background and related work in optimal control and HJB equations. In Section 3, we present a motivated example illustrating possible solutions to HJB equations. Sections 4 and 5 cover discussions on control policy synthesis and learning solutions to HJB equations, respectively. Finally, we present the experimental results and conclude our work.

2. Background and Related Work

In this section, we give the formulation of the optimal control problem and background on HJB equations.

Problem statement We consider the time-invariant control affine system given by

$$\dot{x}(t) = f(x(t)) + g(x(t))u(x(t)), \tag{1}$$

with state $x(t) \in \mathcal{X} \subseteq \mathbb{R}^n$, drift $f(x(t))\colon \mathcal{X} \to \mathbb{R}^n$, actuator $g(x(t))\colon \mathcal{X} \to \mathbb{R}^{n\times m}$ and the policy $u(x(t))\colon \mathcal{X} \to \mathcal{U} = \{u\colon u_{\min} \le u \le u_{\max}\} \subseteq \mathbb{R}^m$, where $u_{\min}$ and $u_{\max}$ are control limit. For the notation convenience, we may omit the argument (e.g. x instead of $x(t)$), and use u to denote both policy and control input.

The goal is to find a feedback policy that minimizes the infinite-time horizon cost-to-go

$$\mathcal{J}(x_0, u(x)) = \int_{t_0}^{+\infty} r(x(\tau), u(x(\tau))d\tau, \tag{2}$$

where $c\colon \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ is the running cost and $x(\tau)$ is the solution of the dynamics (1) given initial condition $x(t_0) = x_0$ and feedback policy $u(x)$. To ensure the well-poses of the problem and the existence of solutions to the HJB equation, we assume the dynamics 1 is controllable and Lipschitz continuous on the compact set $\mathcal{X}$ that contains the origin. Additionally, we require that the drift f satisfies $f(0) = 0$, and running cost r satisfies $r(0, 0) = 0$.

To streamline our upcoming policy derivation, we further assume a quadratic form for the running cost: $r(x, u) = l(x) + u^\top Ru$, where $l\colon \mathcal{X} \to \mathbb{R}$ is the state-dependent cost, and $R \succ 0$ is a positive-definite matrix. These assumptions simplify our problems significantly, yet remain non-restrictive, as many applications can be written down in such a form (Tedrake, 2023).

HJB equation for optimal control Given dynamics (1) and the cost-to-go function (2), we define the value function as cost-to-go under optimal policy

$$\mathcal{V}(x_0) = \min_u \mathcal{J}(x_0, u(x)), \tag{3}$$

and the Hamiltonian

$$\mathcal{H}(x, u, \mathcal{V}) = r(x, u) + \frac{\partial \mathcal{V}^\top}{\partial x}\Big(f(x) + g(x)u\Big). \tag{4}$$

Then the HJB equations (Bertsekas, 2012; Tedrake, 2023)

$$u^* = \operatorname*{argmin}_{u\in\mathcal{U}} \mathcal{H}(x, u, \mathcal{V}); \tag{5a}$$

$$\mathcal{H}(x, u^*, \mathcal{V}) = 0, \quad \forall x \in \mathcal{X}, \tag{5b}$$

provide necessary conditions for the candidate function $V(x)$ and control policy $u(x)$ as value function and optimal policy. The HJB equation stems from dynamic programming theory. For those acquainted with the Bellman equation in discrete time, the HJB equation serves as its extension into a continuous-time system.

Learning positive-definite neural model Learning neural models with positive-definite properties is a promising direction for synthesizing policy in nonlinear control problems and has been investigated in recent work (Chow et al., 2018; Chang et al., 2019; Chang & Gao, 2021) with a focus on system stability and safety. However, most approaches achieve positive definiteness "softly" by incorporating penalty terms into the loss function. Such formulations complicate the training procedure due to multiple competing objectives (Mertikopoulos et al., 2018). The imbalance between numerous ill-defined solutions and scarce admissible solutions to our HJB problems further exacerbates the training difficulty. Therefore, a neural network warm start is still needed to obtain a good policy that stabilizes the system. Moreover, the penalty loss only applies to the training dataset. Consequently, the learned model provides little guarantee of positive definiteness on unseen states, and the policy's performance may decay due to the violation of positive definiteness.

In contrast, (Kolter & Manek, 2019) had constrained the stability upon the model through an input-convex model (Amos et al., 2017) and demonstrated success for learning non-expansive dynamics. We show that such a structured-regularized neural architecture can be applied to solve HJB equations in optimal control, facilitating easier training.

3. Infinite Solutions to HJB

In this section, we elucidate the potential solutions to the HJB equation through an example. We highlight that, even for simple cases, there are numerous ill-defined solutions compared to the limited admissible ones to the HJB equations, posing a challenge for learning the admissible solution.

Consider infinite-time LQR problems,

$$\min_{u(t)} \int_0^\infty x^\top Qx + u^\top Ru\, dt \tag{6}$$
$$\text{subject to } \dot{x} = Ax + Bu$$

where $Q \succeq 0$ and $R \succ 0$ are symmetric and real-valued matrices. The value function of LQR takes quadratic form

$V(x) = x^\top Px$. Substituting the quadratic value function into our HJB equation (5) yields

$$u(x) = -R^{-1}B^T Px; \tag{7a}$$

$$x^T(A^T P + PA - PBR^{-1}B^T P + Q)x = 0, \tag{7b}$$

Equation 7b must hold for all x, implying the satisfaction of the Algebraic Riccati Equation (ARE).

$$A^T P + PA - PBR^{-1}B^T P + Q = 0. \tag{8}$$

Now let's assign numerical values to ARE equations. We assume all matrices are 2-by-2 identity matrices i.e. $A = B = Q = R = I_{2\times 2}$, and $P = \begin{bmatrix} x, & z_1 \\ z_2, & y \end{bmatrix}$ is a real-value matrix. The matrices we choose are not specific but mainly for calculation and visualization convenience. Readers can verify the system is controllable and fully actuated.

Substituting the numbers into ARE (8) gives us four quadratic equations with four unknowns

$$\begin{array}{l} x^2 + z_2^2 = 2x + 1 \\ y^2 + z_1^2 = 2y + 1 \\ z_1 x + z_2 y = 2z_2 \\ z_1 x + z_2 y = 2z_1 \end{array}. \tag{9}$$

There is one positive-definite solution and one negative solution to the equations (9):

$$\begin{cases} x = 1 \pm \sqrt{2} \\ y = 1 \pm \sqrt{2} \\ z_1 = z_2 = 0 \end{cases}, \tag{10}$$

and infinite indefinite solutions

$$\begin{cases} x = c \\ y = 2 - c \\ z_1 = z_2 = \pm\sqrt{2c + 1 - c^2} \\ c \in [1 - \sqrt{2}, 1 + \sqrt{2}] \end{cases}. \tag{11}$$

We visualize the solutions in Figure 1 (a) and demonstrate that only positive-definite solutions result in a stabilizing policy. The non-uniqueness of ARE solutions commonly holds for arbitrary LQR problems (Willems, 1972). This phenomenon is also reported in general nonlinear dynamics (Misztela, 2018). We demonstrate in Figure 1 (c), without warmup, it is easy for a generic neural network to converge to a non-positive definite solution.

4. Control Policy Synthesized

In this section, we discuss how to synthesize the policy given the value function. The main idea is to solve the optimization problem 5a by exploring the structure of systems. The approach described in this section is then applied to learn the value function, which we will discuss in Section 5.

Under the assumptions outlined in Section 2 (i.e. control affine system, quadratic running cost on u, and input limit constraints) the optimization problem (5a) simplifies to solving a quadratic programming (QP) problem.

$$\begin{array}{ll} \underset{u}{\text{minimize}} & u^\top Ru + \frac{\partial \mathcal{V}^\top}{\partial x} g(x)u + l(x) + \frac{\partial \mathcal{V}^\top}{\partial x} f(x) \\ \text{subject to} & u_{\min} \le u \le u_{\max} \end{array}. \tag{12}$$

To start, we explore two special cases. First, we consider situations where the control input is unlimited ($u_{\min} = -\infty$ and $u_{\max} = +\infty$). In this scenario, the optimal control is obtained by setting the gradient of the objective function in (12) to zero, resulting in $u^* = -\frac{R^{-1}}{2} g(x)^\top \frac{\partial \mathcal{V}}{\partial x}$.

Secondly, when the control input is one-dimensional (i.e., the system has one actuator), the optimal control is given by $u^* = \text{clip}\left(-\frac{R^{-1}}{2} g(x)^\top \frac{\partial \mathcal{V}}{\partial x}, u_{\min}, u_{\max}\right)$, where the clip function truncates the control at the limits. This truncated control is optimal as it satisfies the Karush-Kuhn-Tucker (KKT) conditions, providing necessary and sufficient conditions for optimality in QP (**?**).

Proposition 4.1. *Let* $h_0(u) = u^\top Ru + \frac{\partial \mathcal{V}^\top}{\partial x} g(x)u + l(x) + \frac{\partial \mathcal{V}^\top}{\partial x} f(x)$ *denotes the objective function,* $h_1(u) = u - u_{max} \le 0$, $h_2(u) = -u + u_{min} \le 0$ *denote the upper and lower limit constraints, and* λ_1 *and* λ_2 *be the associated Lagrangian dual variables respectively. Then* $u^* = clip(-\frac{R^{-1}}{2} g(x)^\top \frac{\partial \mathcal{V}}{\partial x}, u_{min}, u_{max})$ *satisfy the KKT conditions*

Proof. The KKT conditions for optimality are listed below

1. The primary inequality constraints $h_i(u^*) \le 0$ holds for $i = 1, 2$.
2. The dual variables are non-negative $\lambda_i \ge 0,\ i = 1, 2$.
3. The complementary slackness hold $\lambda_i h_i(u) = 0,\ i = 1, 2$.
4. The gradient of Lagrangian vanishes with respect to u

$$\nabla h_0(u) + \lambda_1 \nabla h_1(u) + \lambda_2 \nabla h_2(u) = 0. \tag{13}$$

In considering three cases, the solution is either truncated by the upper limits (i.e., $-\frac{R^{-1}}{2} g(x)^\top \frac{\partial \mathcal{V}}{\partial x} > u_{\max}$), truncated by the lower limits, or the solution $u^* = -\frac{R^{-1}}{2} g(x)^\top \frac{\partial \mathcal{V}}{\partial x}$ lies within both limits. In all cases, the primary inequality constraints naturally hold.

Now, let's delve into the case of optimal control truncated at the upper limits. Since $h_2(u_{\max}) = u_{\min} - u_{\max} < 0$, by

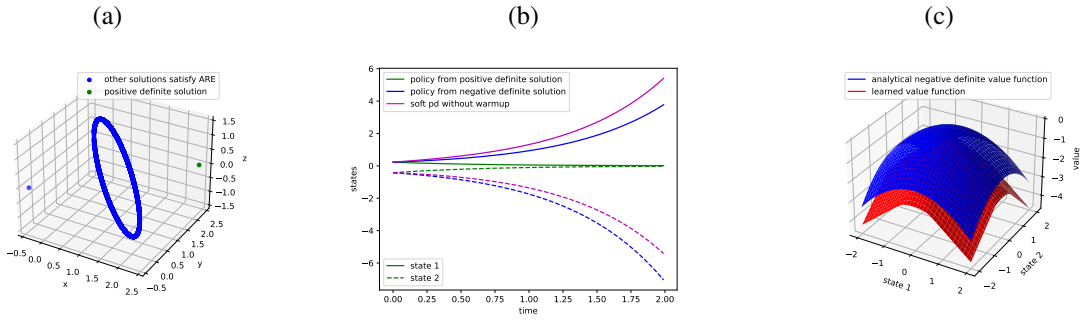

Figure 1. **(a) Visualization of solutions to ARE:** The positive and negative solutions are unique (Willems, 1972), while there are infinite-many indefinite solutions in the 1-D ring. **(b) Trajectories by policies obtained from different solutions:** Given the same initial condition, we execute policies derived from different solutions to ARE. **(c) Analytical solution versus learned solution:** We train a generic multilayer perceptron model to solve HJB equations, where it converges to the negative-definite solution.

complementary slackness $\lambda_2 = 0$. Substituting $u^* = u_{\max}$ and $\lambda_2 = 0$ into equation 13 yield $\lambda_1 = -(2Ru_{\max} + g(x)^\top \frac{\partial \mathcal{V}}{\partial x})$. By the assumption $-\frac{R^{-1}}{2} g(x)^\top \frac{\partial \mathcal{V}}{\partial x} > u_{\max}$, so $\lambda_1 = -2Ru_{\max} - g(x)^\top \frac{\partial \mathcal{V}}{\partial x} > 2R\Big(\frac{R^{-1}}{2} g(x)^\top \frac{\partial \mathcal{V}}{\partial x}\Big) - g(x)^\top \frac{\partial \mathcal{V}}{\partial x} = 0$. Therefore KKT condition holds.

The proof for the case when the optimal control is truncated at the lower limits is analogous. Now let's consider the case when $u_{\min} \leq u^* = -\frac{R^{-1}}{2} g(x)^\top \frac{\partial \mathcal{V}}{\partial x} \leq u_{\max}$. Because both the inequality constraints strictly hold, by complementary slackness $\lambda_1 = \lambda_2 = 0$. The gradient of the Lagrangian vanish condition also holds because $u^* = -\frac{R^{-1}}{2} g(x)^\top \frac{\partial \mathcal{V}}{\partial x} = 0$ and $\lambda_1 = \lambda_2 = 0$. Therefore, KKT conditions hold. □

For high-dimensional control inputs, we employ the active-set method (Wong, 2011), akin to the preceding proof. We first guess the active inequality constraints (i.e. $h_i(u) = 0$), then solve the equations (13), and verify if the resulting solution u satisfies KKT conditions.

Exploiting the structure of the optimal control problems and solving associated QP with the active-set method and KKT conditions enable efficient synthesis of control policies. Furthermore, leveraging auto-grad learning frameworks such as Google Jax (Bradbury et al., 2018) allows for batch computation of control inputs on GPUs, facilitating quick learning solutions to HJB equations, which we will discuss shortly.

5. Learning Solutions to HJB Equations

In this section, we describe how to learn the solution to HJB equations, namely value functions. The idea is to minimize a sample-based penalty that quantifies the extent to which the value function violates the HJB equations. We underscore the significance of learning positive-definite solutions in Section 3. Here, we also detail the methodology to enforce the model to learn positive-definite solutions.

5.1. HJB Loss Function

Let $\hat{\mathcal{V}}(x, \theta)$ be a neural function candidate, where x is the input and θ denotes the parameters. Ideally, we would want the function $\hat{\mathcal{V}}(x, \theta)$ to satisfy the HJB equations 5 for any state $x \in \mathcal{X}$. Since we cannot enforce HJB equations 5 straightforwardly by constructions. We introduce an auxiliary loss term

$$L_{\text{HJB}}(\theta, \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{x_i \in D} \| r(x_i, u(x_i, \theta)) + \frac{\partial \hat{\mathcal{V}}^\top}{\partial x}(x_i, \theta)[f(x_i) + g(x_i)u(x_i, \theta)] \|_2^2, \tag{14}$$

where $\mathcal{D} = \{x_i \in \mathcal{X}\}$ is the unlabeled dataset with size $|\mathcal{D}|$, $\frac{\partial \hat{\mathcal{V}}}{\partial x}(x, \theta)$ is the gradient of the neural function with respect to the input x, and $u(x, \theta)$ is the control policy associated with current value function (See section 4 for derivation). Both gradient $\frac{\partial \hat{\mathcal{V}}}{\partial x}(x, \theta)$ and policy $u(x, \theta)$ are functions of the state x and parameters θ, and the entire surrogate loss function (14) are differentiable. Therefore, we can apply gradient-based methods to minimize the loss (14) with respect to θ. The idea of relaxing functional constraints with sampling-based penalty is common in learning control-orientated structures (Singh et al., 2021; Richards et al., 2023) and more generally in solving PDE with (physically-informed) neural networks(PINN) (Raissi et al., 2019; Cuomo et al., 2022).

5.2. Learning Positive-definite Model with "Soft" Method

Here, we describe a "soft" method to learn positive-definite solutions by introducing an additional penalty term to the HJB loss (14). We define the HJB loss with positive-definite

requirement as

$$L_{\text{HJB}}^{\text{Soft PD}}(\theta, \mathcal{D}) = \alpha_1 L^{\text{HJB}}(\mathcal{D}) + \alpha_2 \|\hat{\mathcal{V}}(0, \theta)\|_2^2 + \alpha_3 \frac{1}{|\mathcal{D}|} \sum_{x_i \in D} \max(0, -\hat{\mathcal{V}}(x_i, \theta)), \quad (15)$$

where α_1, α_2, and α_3 are weights that balance the importance of satisfying HJB equations and positive-definiteness. This method is founded on literature focused on jointly learning stability certificates and control policies (Chang et al., 2019; Chang & Gao, 2021). Despite demonstrating promising results in learning admissible solutions, training the neural network proves challenging and necessitates a warm-up from an existing controller (e.g., LQR). Consequently, the focus shifts towards enhancing the performance of the current controller (e.g., expanding the region the controller can stabilize the system) rather than deriving a controller from scratch.

5.3. Positive-definite Neural Architecture

The training challenge of "soft" methods arises from multi-objective optimization and stochastic gradient descent (Mertikopoulos et al., 2018). The imbalance between numerous inadmissible solutions and scarce admissible ones, both resulting in zero HJB loss, exacerbates the situation. Moreover, even when the loss is minimized in the training data, positive definiteness is still not guaranteed due to unseen states. To address these challenges, we introduce the use of a positive-definite neural architecture

We consider a fully connected n-layer neural network over the state x

$$\begin{aligned} z_1 &= x, \\ z_{k+1} &= \sigma(W_i z_k), \quad k = 1, 2 \dots n-1, \\ \hat{\mathcal{V}}(x, \theta) &= z_n^T z_n + \epsilon \|x\|_2^2, \end{aligned} \quad (16)$$

where z_i denotes each ith layer, σ are nonlinear-activation functions with property $\sigma(0) = 0$ such as rectified linear unit, hyperbolic tangent function, and sine function, $\theta = \{W_1, W_2, \dots W_{n-1}\}$ is the parameter, and ϵ is a fixed positive scalar.

Proposition 5.1. *The function $\hat{\mathcal{V}}(x, \theta)$ is positive-definite respect to the input x.*

Proof. First, for any non-zero input x, the output $\hat{\mathcal{V}}(x, \theta)$ is strictly positive as $z_n^T z_n$ is non-negative(by dot product) and $\epsilon \|x\|_2^2$ is strictly positive (due to property of euclidian norm). Second, $\hat{\mathcal{V}}(0, \theta) = 0$. If $z_k = 0$, then $z_{k+1} = \sigma(W_k 0) = \sigma(0) = 0$. When $x = z_1 = 0$, it follows that $z_n = 0$, and euclidian norm of origin equals zero. Therefore $\hat{\mathcal{V}}(0, \theta) = 0$ and we conclude $\hat{\mathcal{V}}(x, \theta)$ is positive-definite respect to the input x. □

Our proposed architecture can provably learn positive-definite solutions, allowing us to directly minimize the HJB loss (14) without the need to balance different objectives. Additionally, it eliminates numerous ill-defined solutions by construction, allowing us to learn solutions directly from scratch and reduce computational time for convergence, as we will demonstrate shortly in our experiments.

6. Experiments

In this section, we experimentally compare our proposed method, "soft" methods with and without initialization, and a linearized LQR controller as a baseline across three classical optimal control problems: double integrator time-optimal control, cartpole balancing, and optimal flying for a 2D drone. We highlight that our method can directly synthesize the best control policy (in terms of cumulated cost) from a randomly initialized network among these examples and demonstrate quick convergence. Full details on system dynamics are provided in Appendix **??**.

Training Details We employ a 4-layer fully connected forward neural network with hidden layer sizes 128, 128, and 64 in the first three layers and a 1-dimensional output as our baseline architecture for "soft" methods. In addition to uniform sampling from the state space Ω, we adopt a common practice in reinforcement learning by rolling out trajectories with the current policy and adding the visited states to the dataset $\mathcal{D}$. We then warm up our baseline methods with a linearized LQR controller using the techniques introduced in () and train with the Adam optimizer with associated loss. For each system, we select data-sampling and warm-up methods, along with other hyperparameters, to maximize the performance (in terms of cumulated cost) of our baseline methods. Full implementation details and hyperparameters are included in Appendix **??**. The same hyperparameters are then applied to our proposed methods, where we initialize the positive-definite model randomly. We emphasize the robustness of our proposed methods to hyperparameter choices, producing similar results with different settings. All methods are implemented using Python and JAX, and training is conducted on a single NVIDIA GeForce RTX 3070 Ti GPU. Computational times, number of gradient updates, and data samples used for our methods to converge to good policy are reported in Table 1.

Testing Procedures We assess policies based on cumulated cost. First, we randomly sample multiple initial conditions within the state space Ω. Subsequently, we simulate the system forward using the given policies and calculate the cumulated costs for each trajectory. During training, we assess the performance of the learned policy after each epoch with 20 different initial conditions and generate a cost training curve. For the final comparison between learned and

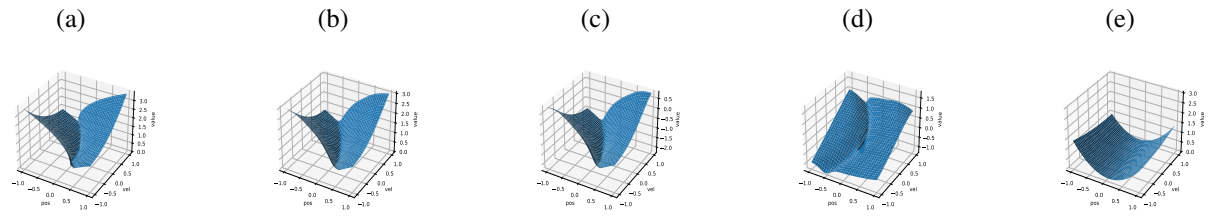

Figure 2. Value functions for double integrator time optimal control, where **(a): Analytical (b): Ours (c): Soft PD with warmup (d): Soft PD without warmup (e): LQR**.

Table 1. Runtime statistics of our methods on three nonlinear control examples.

DYNAMICS	TIME(SEC)	ITERS	SAMPLES
DOUBLE INTEGRATOR	6.2	10K	65K
CARTPOLE	1.1	390	24K
2D DRONE	10.7	18K	117K

model-based policies, we initialize 100 different initial conditions, providing each method with the same set of initial conditions, and simulate forward. We present the performance of the different policies in Table 2 and demonstrate qualitative results below.

Double Integrator Time-Optimal Control The double integrator time-optimal control problem involves pushing a unit mass brick system to the origin with bounded inputs. The time to the origin is defined as the infinite-time integral of the indicator function $t_f = \int_0^\infty \mathbb{1}_{\{\|x\|_2 \geq 10^{-2}\}}(x(t))dt$, where the running cost $r(x,u) = \mathbb{1}_{\{\|x\|_2 \geq 10^{-2}\}}(x)$ equal to zero when the brick is sufficient close to origin, and equal to 1 for other cases. For the LQR controller, we relax the indicator function to $r(x,u) = x^\top Q x + u^\top R u$, where Q is a 2-by-2 identity matrix and $R = 0.01Q$ to encourage saturated control. The problem is widely studied, and analytical solutions exist that we can utilize to evaluate the performance of different methods.

Our methods generate the most time-optimal trajectory compared to all baseline methods (see Figure 3 (a) and Table 2). The results align with the visualization of value functions, as shown in Figure 2. We observe that the gradients of the learned results are smaller than the analytical solutions around the origin, where the analytical solution exhibits a sharper valley. This discrepancy explains the optimal time gap between the learned method and the analytical solution. Figure 2 (d) shows the soft PD method can learn suitable value functions in some state spaces without warm-up. However, when we further train the models with 10 times more training iterations and obtain smaller PD HJB loss 15 compared to its warmup counterpart, the performance of the associated policy did not improve. Therefore, we argue it is unlikely for the soft PD method to learn admissible solutions in the entire state space without warm-up. We also observed a sudden decay in the performance of policies generated by soft PD methods during the training process. This may be attributed to the temporary violation of the positive-definiteness property when optimizing the HJB loss further. In contrast, our methods guarantee a positive-definite value function throughout the training process, resulting in a more stable cost training curve.

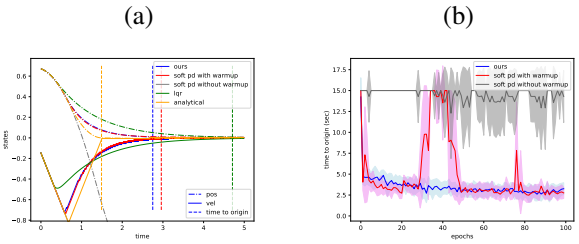

Figure 3. **(a): Trajectories for double integrator:** Our methods outperform all baseline methods to generate time optimal trajectory. **(b): Cost training curve for double integrator:** The shaded region represents one standard deviation among different initial conditions. We set the time to 15 if the system has not reached the origin by then.

Cartpole Balancing The cartpole balancing is a standard nonlinear control problem for testing different control methods. The goal is to keep the pole upright at the origin with the minimum energy. We define the running cost $r(x,u) = (x - x_f)^\top Q(x - x_f) + u^\top R u$, where x_f is denotes up-right equilibrium. Figure 4 (a) illustrates that both our method and the LQR controller successfully stabilize the system, generating similar trajectories. While the soft method, when properly initialized, learns to balance the pole in the upper-right position, the cart is not precisely controlled to zero.

Drone Control We consider the problem of controlling a 2D drone to the desired position as quickly and energy-efficient as possible given random initial conditions. The running cost is defined as $r(x,u) = (x - x_f)^\top Q(x - x_f) +$

Table 2. Performance of policies in terms of cost-to-go among three examples. For double integrator time optimal control, we cut off the cost-to-go at 15 if the system does not reach the origin by that time.

METHODS	DOUBLE INTEGRATOR	CARTPOLE	2D DRONE
OURS	2.69 ± 0.80	9.57 ± 6.73	4.80 ± 3.32
LQR	4.36 ± 1.24	9.57 ± 6.74	5.03 ± 3.61
SOFT PD WITH WARMUP	2.72 ± 0.80	9.59 ± 6.73	$55.33\text{K} \pm 72.13\text{K}$
SOFT PD WITHOUT WARMUP	13.58 ± 4.14	2082.66 ± 2685.06	$1460\text{K} \pm 61.87\text{K}$

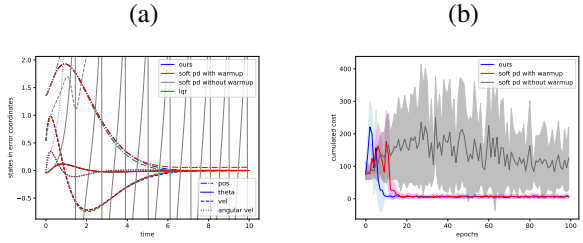

Figure 4. **(a): Trajectories for cartpole:** We show the states in the error coordinates $e = x - x_f$. Our methods learn a policy that balances the cartpole at origin. **(b): Cost training curve for cartpole:** The shaded region represents one standard deviation among different initial conditions.

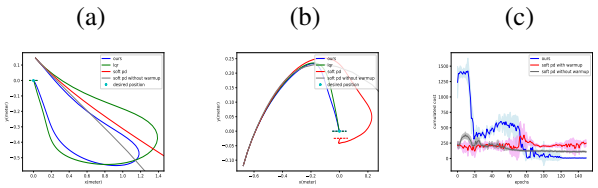

Figure 5. **(a): Trajectory for drone:** The dashed lines are the final orientations of Drones. Our method successfully controls the drone to the desired position, while the soft method fails. **(b): Trajectory for drone with another initial condition:** The soft methods stabilize the system and control the drone toward the desired position. **(c): Cost training curve for drone control**: The shaded region represents one standard deviation among different initial conditions. The cost-to-go grows rapidly for random controllers. To improve visualization, we halt the calculation of the cost-to-go if the drone flies too far away from Ω. See Appendix **??** for details.

$(u - u_f)^\top R(u - u_f)$, where x_f is the desired position and u_f is the control to maintain hovering. Our methods learn the policy that successfully controls the drone to the desired position for any initial condition. In contrast, the soft method, even with warm-up training, stabilizes the system only for some initial conditions (See Figure 5 (a) and (b)). We also observe our method outperforming the LQR controller. This is likely due to the LQR controller being limited by the linear approximation, while our method leverages full nonlinear dynamics.

7. Conclusion and Future work

In this paper, we studied what makes learning a good policy difficult in HJB optimal control problems. We identify potential solutions to the HJB equation and emphasize that many of these solutions can be ill-defined. We underscore the imbalance between inadmissible and admissible solutions, a factor contributing to training difficulties, and stress the significance of learning positive-definite solutions. For this purpose, we propose a positive-definite architecture for learning solutions to the HJB equation. Overall, our methods outperform baseline methods for optimal control policy synthesis. Moreover, our methods can directly learn solutions from random initialization, offering significantly improved ease of training.

Future work We view our methods as control-informed learning techniques that aim to synthesize optimal policies through deep learning approaches. We demonstrate the ability to efficiently learn complex controllers, outperforming naive model-based controllers like LQR. In the real world, neither detailed dynamic models nor specific parameters for the model are readily available. Thus, an interesting avenue for future research lies in simultaneously learning the feedback policy and control-orientated dynamics structure. This could build off of existing work in (Kolter & Manek, 2019; Richards et al., 2023). We envision that control-informed and structure-aware learning enables more data-efficient and generalizable control policy synthesis compared to a pure model-free approach.

References

Amos, B., Xu, L., and Kolter, J. Z. Input convex neural networks. In *International Conference on Machine Learning*, pp. 146–155. PMLR, 2017.

Bansal, S. and Tomlin, C. J. Deepreach: A deep learning approach to high-dimensional reachability. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1817–1824. IEEE, 2021.

Beard, R. W., Saridis, G. N., and Wen, J. T. Galerkin approximations of the generalized hamilton-jacobi-bellman equation. *Automatica*, 33(12):2159–2177, 1997.

Bellman, R. Dynamic programming and a new formalism in the calculus of variations. *Proceedings of the national academy of sciences*, 40(4):231–235, 1954.

Bellman, R. and Kalaba, R. On adaptive control processes. *IRE Transactions on Automatic Control*, 4(2):1–9, 1959.

Bertsekas, D. *Dynamic programming and optimal control: Volume I*, volume 4. Athena scientific, 2012.

Björk, T., Murgoci, A., and Zhou, X. Y. Mean–variance portfolio optimization with state-dependent risk aversion. *Mathematical Finance: An International Journal of Mathematics, Statistics and Financial Economics*, 24(1):1–24, 2014.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Chang, Y.-C. and Gao, S. Stabilizing neural control using self-learned almost lyapunov critics. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1803–1809. IEEE, 2021.

Chang, Y.-C., Roohi, N., and Gao, S. Neural lyapunov control. *Advances in neural information processing systems*, 32, 2019.

Chow, Y., Nachum, O., Duenez-Guzman, E., and Ghavamzadeh, M. A lyapunov-based approach to safe reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

Cuomo, S., Di Cola, V. S., Giampaolo, F., Rozza, G., Raissi, M., and Piccialli, F. Scientific machine learning through physics–informed neural networks: Where we are and what's next. *Journal of Scientific Computing*, 92(3):88, 2022.

Darbon, J., Langlois, G. P., and Meng, T. Overcoming the curse of dimensionality for some hamilton–jacobi partial differential equations via neural network architectures. *Research in the Mathematical Sciences*, 7:1–50, 2020.

Guo, C.-H. and Higham, N. J. Iterative solution of a nonsymmetric algebraic riccati equation. *SIAM journal on matrix analysis and applications*, 29(2):396–412, 2007.

Han, J., Jentzen, A., and E, W. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34): 8505–8510, 2018.

Kolter, J. Z. and Manek, G. Learning stable deep dynamics models. *Advances in neural information processing systems*, 32, 2019.

Laub, A. A schur method for solving algebraic riccati equations. *IEEE Transactions on Automatic Control*, 24 (6):913–921, 1979. doi: 10.1109/TAC.1979.1102178.

Lutter, M., Belousov, B., Listmann, K., Clever, D., and Peters, J. Hjb optimal feedback control with deep differential value functions and action constraints. In *Conference on Robot Learning*, pp. 640–650. PMLR, 2020.

Mertikopoulos, P., Papadimitriou, C., and Piliouras, G. Cycles in adversarial regularized learning. In *Proceedings of the twenty-ninth annual ACM-SIAM symposium on discrete algorithms*, pp. 2703–2717. SIAM, 2018.

Misztela, A. On nonuniqueness of solutions of hamilton–jacobi–bellman equations. *Applied Mathematics & Optimization*, 77:599–611, 2018.

Mitchell, I. M. and Templeton, J. A. A toolbox of hamilton-jacobi solvers for analysis of nondeterministic continuous and hybrid systems. In *International workshop on hybrid systems: computation and control*, pp. 480–494. Springer, 2005.

Nakamura-Zimmerer, T., Gong, Q., and Kang, W. Adaptive deep learning for high-dimensional hamilton–jacobi–bellman equations. *SIAM Journal on Scientific Computing*, 43(2):A1221–A1247, 2021.

Ng, A. Y., Harada, D., and Russell, S. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pp. 278–287. Citeseer, 1999.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

Richards, S. M., Slotine, J.-J., Azizan, N., and Pavone, M. Learning control-oriented dynamical structure from data. *arXiv preprint arXiv:2302.02529*, 2023.

Singh, S., Richards, S. M., Sindhwani, V., Slotine, J.-J. E., and Pavone, M. Learning stabilizable nonlinear dynamics with contraction-based regularization. *The International Journal of Robotics Research*, 40(10-11):1123–1150, 2021.

Smears, I. Hamilton-jacobi-bellman equations, analysis and numerical analysis. chapter 3. 2018.

Sundar, S. and Shiller, Z. Optimal obstacle avoidance based on the hamilton-jacobi-bellman equation. *IEEE transactions on robotics and automation*, 13(2):305–310, 1997.

Tedrake, R. Underactuated robotics. chapter 3 and 7. 2023. URL https://underactuated.csail.mit.edu.

Willems, J. Least squares optimal control and algebraic riccati equations. *Automatic Control, IEEE Transactions on*, AC-16:621 – 634, 01 1972. doi: 10.1109/TAC.1971.1099831.

Wong, E. *Active-set methods for quadratic programming*. University of California, San Diego, 2011.

A. You *can* have an appendix here.

You can have as much text here as you want. The main body must be at most 8 pages long. For the final version, one more page can be added. If you want, you can use an appendix like this one.

The `\onecolumn` command above can be kept in place if you prefer a one-column appendix, or can be removed if you prefer a two-column appendix. Apart from this possible change, the style (font size, spacing, margins, page numbering, etc.) should be kept the same as the main body.