A NEURAL NETWORK APPROACH FOR STOCHASTIC OPTIMAL CONTROL[*]

XINGJIAN LI[†], DEEPANSHU VERMA[†], AND LARS RUTHOTTO[†]

Abstract. We present a neural network approach for approximating the value function of high-dimensional stochastic control problems. Our training process simultaneously updates our value function estimate and identifies the part of the state space likely to be visited by optimal trajectories. Our approach leverages insights from optimal control theory and the fundamental relation between semi-linear parabolic partial differential equations and forward-backward stochastic differential equations. To focus the sampling on relevant states during neural network training, we use the stochastic Pontryagin maximum principle (PMP) to obtain the optimal controls for the current value function estimate. By design, our approach coincides with the method of characteristics for the non-viscous Hamilton-Jacobi-Bellman equation arising in deterministic control problems. Our training loss consists of a weighted sum of the objective functional of the control problem and penalty terms that enforce the HJB equations along the sampled trajectories. Importantly, training is unsupervised in that it does not require solutions of the control problem.

Our numerical experiments highlight our scheme's ability to identify the relevant parts of the state space and produce meaningful value estimates. Using a two-dimensional model problem, we demonstrate the importance of the stochastic PMP to inform the sampling and compare to a finite element approach. With a nonlinear control affine quadcopter example, we illustrate that our approach can handle complicated dynamics. For a 100-dimensional benchmark problem, we demonstrate that our approach improves accuracy and time-to-solution and, via a modification, we show the wider applicability of our scheme.

Key words. Hamilton-Jacobi-Bellman equation, high-dimensional stochastic optimal control, forward-backward stochastic differential equations, neural networks, stochastic maximum principle

AMS subject classifications. 35F21, 49M99, 68T07

1. Introduction.

We consider stochastic optimal control (SOC) problems that require finding a policy to control randomly perturbed dynamical systems to optimize a given objective functional. Problems of this type arise in many fields, including finance, biology, robotics, and many other engineering applications; see, for example, [11, 37] for references and extensive theoretical discussion on SOC problems.

Dynamic programming (DP) is a prominent framework for solving SOC problems. At its core, DP seeks to find the value function, which assigns every state of the system the optimal cost-to-go. For many problems, recovering the optimal control from the value function is straightforward. One way to compute the value function is by solving the Hamilton-Jacobi-Bellman (HJB) equation [2, 40], which is a semi-linear parabolic Partial Differential Equation (PDE). The two main challenges in solving the HJB equations are the forward-backward structure and the high dimensionality.

The space dimension of the HJB equation equals the state dimension of the dynamical system to be controlled. Hence, many numerical schemes for solving PDEs cannot be applied to realistic problem instances due to the Curse-of-Dimensionality (CoD); for example, the computational costs of approaches that discretize the state space with a mesh typically grow exponentially in the dimensions of the system. Hence, the applicability of existing approaches that employ spatial discretizations

[*]Submitted to the editors March 10, 2023.

Funding: This work was supported in part by NSF awards DMS 1751636 and DMS 2038118s, AFOSR grant FA9550-20-1-0372, and US DOE Office of Advanced Scientific Computing Research Field Work Proposal 20-023231.

[†]Department of Mathematics, Emory University, Atlanta, GA (xingjian.li@emory.edu, deepanshu.verma@emory.edu, lruthotto@emory.edu).

(for example, [4, 7, 17, 21, 35, 23, 24, 35, 38]) is limited to state dimensions $d \leq 3$. Mesh-free methods based on radial basis function methods (for example, [39]) can be effective for slightly larger d but ultimately also suffer from CoD for $d = \mathcal{O}(100)$, as we consider here.

A first requirement for mitigating the CoD in SOC problems is to alleviate the need for spatial discretization. This can be achieved by using a nonlinear version of the Feynman-Kac lemma and replacing the HJB equation with a system of Forward-Backward Stochastic Differential Equations (FBSDEs); see, for example, [6, 31, 40]. Discretizing the FBSDE system using the classical Euler Maruyama scheme [20] yields mesh-free numerical schemes that have gained importance for SOC problems and nonlinear second-order PDEs; see, for example, [3, 9, 10, 14, 28] and references therein.

A second requirement for mitigating the CoD is to parameterize the value function effectively in high dimensions. Due to their universal approximation property and many advances in deep learning, the use of neural networks (NNs) as function approximators has recently gathered significant attention. The idea of combining the BSDE approach with NNs has been pioneered by the seminal works [8, 13] that use neural networks to solve a wider class of high-dimensional semi-linear parabolic PDEs and include a stochastic optimal control example as a special case. Around the same time, [36] achieves similarly promising results for neural-network-based solution of the HJB equation. These works developed learning algorithms for approximating the value function or its gradient using the FBSDE system. Their success has sparked several extensions that improve the NN architectures and consider other types of PDEs; see, for example, [16, 18, 32].

Even after combining FBSDEs and deep learning, solving the HJB equation globally remains affected by the curse of dimensionality as it would require sampling the entire state space. Fortunately, the optimal state trajectories of many SOC problems do not cover the entire space. This motivates us to develop semi-global approaches; that is, we seek to reliably estimate the value function in those parts of the state space that are likely to be visited when following an optimal policy. Clearly, these tasks are interrelated: Finding the relevant parts of the state space depends on the value function, and, vice versa, the value function needs to be trained using samples from the state space.

The key idea of our approach is to use the stochastic Pontryagin maximum principle (PMP) [34] to link the sampling and the value function approximation; more precisely, we define the forward system in the FBSDE approach in terms of the value function. The PMP provides a set of necessary optimality conditions, known as the Hamiltonian system, and a closed loop feedback form that allows recovering the optimal control from the value function; see, for example, [40]. Our formulation is suitable for problems where the underlying Hamiltonian can be efficiently computed, such as those involving affine controls and convex Lagrangians with differentiable value functions. As we discuss in more detail later, the choice of the forward SDE is crucial as it is used to sample the parts of the state space along which the value function estimate is improved using a loss that includes the backward SDE. It is worth noting that the FBSDE framework allows choosing the forward system almost arbitrarily. Thus, our choice is theoretically on par with the standard Brownian motion used for exploring the state space in [13, 36]. However, as our numerical experiments suggest, our choice of the forward dynamics enables us to solve a wider class of SOC problems; see section 4.

A theoretical advantage of our proposed scheme over existing approaches is its consistency with deterministic optimal control problems. In the absence of uncertainty in the dynamics, our approach reduces to the method of characteristics for the HJB equation. Hence, our approach can also be seen as an extension of the neural network approaches for deterministic optimal control proposed, for example, in [30, 22].

The rest of the paper is organized as follows: In section 2, we introduce the parts of SOC theory used to obtain our proposed forward SDE. In section 3, we provide the NN approach to learn the value function using the FBSDE reformulation from section 2. Using an intuitive 2D example, we show the importance of sampling. For this low-dimensional problem, we use a finite element approach to compare and validate our method. To illustrate the potential of our method, we consider the 100-dimensional benchmark problem also used in [13, 36], and propose modifications that lead to more challenging sampling problems. We also test our method on a 12-dimensional problem with complex nonlinear dynamics. Finally, we conclude the paper and discuss future directions.

2. Stochastic Optimal Control Background.

In this section, we describe the stochastic optimal control (SOC) problems considered in this work and review the key results from SOC theory that motivate our approach. Our discussion follows [40] and we refer to this textbook for a more comprehensive background and for more general results. We first introduce the SOC problem and then review its underlying theory; to be specific, we review the stochastic Pontryagin Maximum Principle (PMP), the Hamilton-Jacobi-Bellman (HJB) equation, and its reformulation into a system of forward-backward stochastic differential equations (FBSDEs) obtained from a nonlinear version of the Feynman-Kac formula.

2.1. Stochastic Optimal Control Problem.

Let $(\Omega, \mathcal{F}, \mathbb{F} = \{\mathcal{F}_t\}_{t\geq 0}, \mathbb{P})$ be a given complete probability space, $W(s)$ be a d-dimensional Brownian motion on $(\Omega, \mathcal{F}, \mathbb{F}, \mathbb{P})$ where s denotes the time. For a fixed initial state $\boldsymbol{x}$ at some time $0 < t < T < \infty$, we seek to control the randomly perturbed dynamical system

$$\begin{cases} \mathrm{d}\boldsymbol{z}_{t,\boldsymbol{x}}(s) = f(s, \boldsymbol{z}_{t,\boldsymbol{x}}(s), \boldsymbol{u}_{t,\boldsymbol{x}}(s, \boldsymbol{z}_{t,\boldsymbol{x}}(s)))\mathrm{d}s + \sigma(s, \boldsymbol{z}_{t,\boldsymbol{x}}(s))\mathrm{d}W(s), s \in [t, T], \\ \boldsymbol{z}_{t,\boldsymbol{x}}(t) = \boldsymbol{x}. \end{cases} \tag{2.1}$$

Here, $\boldsymbol{z}_{t,\boldsymbol{x}} : [t, T] \to \mathbb{R}^d$ describes the state and $\boldsymbol{u}_{t,\boldsymbol{x}} : [t, T] \times \mathbb{R}^d \to U$ describes the control of the system, the function $\sigma : [t, T] \times \mathbb{R}^d \to \mathbb{R}^{d\times d}$ represents the diffusion coefficient, and $f : [t, T] \times \mathbb{R}^d \times U \to \mathbb{R}^d$ represents the drift of the system. We assume that the set of admissible controls $U \subset \mathbb{R}^k$ is closed. We seek to minimize the objective functional

$$J_{t,\boldsymbol{x}}(\boldsymbol{u}_{t,\boldsymbol{x}}) = \mathbb{E}\left\{ G\big(\boldsymbol{z}_{t,\boldsymbol{x}}(T)\big) + \int_t^T L\big(s, \boldsymbol{z}_{t,\boldsymbol{x}}(s), \boldsymbol{u}_{t,\boldsymbol{x}}(s, \boldsymbol{z}_{t,\boldsymbol{x}}(s))\big)\,\mathrm{d}s \right\}, \tag{2.2}$$

which is comprised of the running cost $L : [t, T] \times \mathbb{R}^d \times U \to \mathbb{R}$ and the terminal cost $G : \mathbb{R}^d \to \mathbb{R}$. Here, the expectation is taken with respect to perturbances of the dynamics (2.1) that is described by the probability measure $\mathbb{P}$. We assume sufficient regularity conditions on f, σ, G, and L, see [40, Chapter 2] for a list of assumptions.

The value function assigns the optimal cost-to-go to any initial state, that is,

$$\Phi(t, \boldsymbol{x}) = \inf_{\boldsymbol{u}_{t,\boldsymbol{x}}} J_{t,\boldsymbol{x}}(\boldsymbol{u}_{t,\boldsymbol{x}}), \tag{2.3}$$

and a solution $\boldsymbol{u}^*_{t,\boldsymbol{x}}$ to (2.3) incurring this minimum value is called an optimal control.

The *generalized Hamiltonian*, $H : [t,T] \times \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R}^{d\times d} \to \mathbb{R} \cup \{\infty\}$, is a key ingredient for the SOC theory in the following sections and the backbone of our numerical scheme. For the problem defined in (2.1) and (2.2) it reads

$$H(s, \boldsymbol{z}, \boldsymbol{p}, \boldsymbol{M}) = \sup_{\boldsymbol{u}\in U} \mathcal{H}(s, \boldsymbol{z}, \boldsymbol{p}, \boldsymbol{M}, \boldsymbol{u}), \tag{2.4}$$

where $\boldsymbol{p}$ and $\boldsymbol{M}$ are called adjoint variables and

$$\mathcal{H}(s, \boldsymbol{z}, \boldsymbol{p}, \boldsymbol{M}, \boldsymbol{u}) = \frac{1}{2}\mathrm{tr}\,(\sigma(s, \boldsymbol{z})\boldsymbol{M}) + \boldsymbol{p}\cdot f(s, \boldsymbol{z}, \boldsymbol{u}) - L(s, \boldsymbol{z}, \boldsymbol{u}).$$

We assume that there exists a unique minimizer of the Hamiltonian (2.4).

To make it notationally convenient, in the rest of the paper, we drop the second argument for the controls and denote controls by $\boldsymbol{u}_{t,\boldsymbol{x}}(s)$.

2.2. Stochastic Pontryagin Maximum Principle. The PMP provides first-order necessary conditions for the SOC problem and also states that the optimal control $\boldsymbol{u}^*_{t,\boldsymbol{x}}$ must satisfy an (extended) Hamiltonian system along the optimal state and adjoint trajectory. This is made precise by the following result from [40, Theorem 3.2, Chapter 3].

THEOREM 2.1. *[40, Theorem 3.2, Chapter 3] Assume that $(\boldsymbol{z}^*_{t,\boldsymbol{x}}, \boldsymbol{u}^*_{t,\boldsymbol{x}})$ is an optimal pair that solves (2.1) and (2.2). Then there exist adjoint states $\boldsymbol{p}_{t,\boldsymbol{x}} \colon [t,T] \to \mathbb{R}^d$ and $\boldsymbol{M}_{t,\boldsymbol{x}} \colon [t,T] \to \mathbb{R}^{d\times d}$ satisfying the adjoint equation*

$$\begin{cases} \mathrm{d}\boldsymbol{p}_{t,\boldsymbol{x}}(s) = & \boldsymbol{M}_{t,\boldsymbol{x}}(s)\mathrm{d}W(s) - \nabla_{\boldsymbol{z}}\mathcal{H}\big(s, \boldsymbol{z}^*_{t,\boldsymbol{x}}(s), \boldsymbol{p}_{t,\boldsymbol{x}}(s), \boldsymbol{M}_{t,\boldsymbol{x}}(s), \boldsymbol{u}^*_{t,\boldsymbol{x}}(s)\big)\mathrm{d}s \\ \boldsymbol{p}_{t,\boldsymbol{x}}(T) = & -\nabla_{\boldsymbol{z}}G\big(\boldsymbol{z}^*_{t,\boldsymbol{x}}(T)\big), \end{cases} \tag{2.5}$$

where $s \in [t,T]$ and the optimal control satisfies

$$\boldsymbol{u}^*_{t,\boldsymbol{x}}(s) = \arg\max_{\boldsymbol{u}\in U}\ \mathcal{H}\big(s, \boldsymbol{z}^*_{t,\boldsymbol{x}}(s), \boldsymbol{p}_{t,\boldsymbol{x}}(s), \boldsymbol{M}_{t,\boldsymbol{x}}(s), \boldsymbol{u}(s)\big) \tag{2.6}$$

for almost all $s \in [t,T]$, $\mathbb{P}$-almost surely.

We note that the optimal control defined in (2.6) only depends on the adjoint variable $\boldsymbol{p}_{t,\boldsymbol{x}}$ but not on $\boldsymbol{M}_{t,\boldsymbol{x}}$ since $\sigma(\cdot,\cdot)$ does not depend on the control.

We further assume that there exists a unique continuous closed-form solution to (2.6). Although not demonstrated in this work, this assumption can be weakened to include implicitly defined functions as long as they can be obtained efficiently; this allows, for example, modeling more general convex running costs.

We note that when the control satisfies (2.6), the dynamics in (2.1) is equal to

$$\begin{cases} \mathrm{d}\boldsymbol{z}^*_{t,\boldsymbol{x}}(s) = \nabla_{\boldsymbol{p}}\mathcal{H}\big(s, \boldsymbol{z}^*_{t,\boldsymbol{x}}(s), \boldsymbol{p}_{t,\boldsymbol{x}}(s), \boldsymbol{M}_{t,\boldsymbol{x}}(s), \boldsymbol{u}^*_{t,\boldsymbol{x}}(s)\big)\mathrm{d}s + \sigma(s, \boldsymbol{z}^*_{t,\boldsymbol{x}}(s))\mathrm{d}W(s), \\ \ \boldsymbol{z}^*_{t,\boldsymbol{x}}(t) = \boldsymbol{x}. \end{cases} \tag{2.7}$$

The system of equations (2.5)–(2.7) is called the stochastic Hamiltonian system, where the maximum condition (2.6) corresponds to the variational inequality for the control.

Finding a tuple $(\boldsymbol{z}^*_{t,\boldsymbol{x}}, \boldsymbol{u}^*_{t,\boldsymbol{x}}, \boldsymbol{p}_{t,\boldsymbol{x}}, \boldsymbol{M}_{t,\boldsymbol{x}})$ that satisfies the PMP can be extremely difficult. However, when the value function Φ is differentiable, $(\boldsymbol{p}_{t,\boldsymbol{x}}, \boldsymbol{M}_{t,\boldsymbol{x}})$ satisfying (2.5) can be obtained from Φ; this is formalized in the following theorem that, with weaker assumptions, can be found in [40, Chapter 5].

Theorem 2.2. *[40, Chapter 5] Assume that $\boldsymbol{u}^*_{t,\boldsymbol{x}}$ is an optimal control and $\Phi \in C^{1,3}([t,T]\times\mathbb{R}^d)$. Then*

$$\boldsymbol{p}_{t,\boldsymbol{x}}(s) = -\nabla_{\boldsymbol{z}}\Phi\big(s,\boldsymbol{z}^*_{t,\boldsymbol{x}}(s)\big) \quad \text{and} \quad \boldsymbol{M}_{t,\boldsymbol{x}}(s)) = -\sigma(s,\boldsymbol{z}^*_{t,\boldsymbol{x}}(s))^\top\nabla^2_{\boldsymbol{z}}\Phi\left(s,\boldsymbol{z}^*_{t,\boldsymbol{x}}(s)\right) \tag{2.8}$$

solve (2.5).

Theorem 2.2 along with (2.6) collectively serve to express the optimal control as

$$\boldsymbol{u}^*_{t,\boldsymbol{x}}(s) = \boldsymbol{u}^*_{t,\boldsymbol{x}}\left(s,\boldsymbol{z}^*_{t,\boldsymbol{x}}(s), -\nabla_{\boldsymbol{z}}\Phi\big(s,\boldsymbol{z}^*_{t,\boldsymbol{x}}(s)\big)\right). \tag{2.9}$$

This relation along with (2.7) is one of the key ingredients of our numerical solution approach. Equation (2.9) characterizes optimal control in a feedback or closed-loop form, which is of utmost importance in many real-life applications. This is because optimal control can be quickly computed at any given point in time and space when the value function is known and its gradient is readily available. Thereby, avoiding re-computation of optimal controls for cases when multiple evaluations are needed for different times or states.

2.3. Hamilton-Jacobi-Bellman Equation.

To help approximate the value function Φ, we also use the fact that Φ satisfies the Hamilton-Jacobi-Bellman (HJB) equation, which is a result of the Dynamic Programming (DP) method or Bellman's principle. We state the following result taken from [40] under suitable assumptions, see also [33, Remark 3.4.4, Theorem 3.5.2].

Theorem 2.3. *[40, Propositon 3.5, Chapter 4] Assume that the value function $\Phi \in C^{1,2}([t,T]\times\mathbb{R}^d)$. Then Φ satisfies the HJB equation*

$$\begin{aligned}&-\partial_s\Phi(s,\boldsymbol{z}) + H\big(s,\boldsymbol{x},-\nabla_{\boldsymbol{z}}\Phi(s,\boldsymbol{z}),-\sigma(s,\boldsymbol{z})^\top\nabla^2_{\boldsymbol{z}}\Phi(s,\boldsymbol{z})\big) = 0,\ \forall(s,\boldsymbol{z})\in[t,T)\times\mathbb{R}^d,\\ &\Phi(T,\boldsymbol{z}) = G(\boldsymbol{z}).\end{aligned} \tag{2.10}$$

The smoothness of Φ can be relaxed to continuity in the weaker sense of viscosity solutions [40, Section 5, Chapter 4].

Using the definition of the Hamiltonian in (2.10) we get the HJB equation as the following second-order parabolic PDE:

$$\begin{cases}-\partial_s\Phi(s,\boldsymbol{z}) - \dfrac{1}{2}\mathrm{tr}(\sigma(s,\boldsymbol{z})\sigma(s,\boldsymbol{z})^\top\nabla^2_{\boldsymbol{z}}\Phi(s,\boldsymbol{z})) - \nabla_{\boldsymbol{z}}\Phi(s,\boldsymbol{z})\cdot f(s,\boldsymbol{z},\boldsymbol{u}^*)\\ \qquad\qquad\qquad - L(s,\boldsymbol{z},\boldsymbol{u}^*) = 0,\ \forall\,(s,\boldsymbol{z})\in[t,T)\times\mathbb{R}^d,\\ \Phi(T,\boldsymbol{z}) = G(\boldsymbol{z}(T)).\end{cases} \tag{2.11}$$

In addition, by the envelope theorem, it follows that $\nabla_{\boldsymbol{p}}\mathcal{H} = \nabla_{\boldsymbol{p}}H$ and $\nabla_{\boldsymbol{M}}\mathcal{H} = \nabla_{\boldsymbol{M}}H$. This simplifies the computation of optimal trajectories, which can now be expressed via the value function as:

$$\begin{cases}\mathrm{d}\boldsymbol{z}^*_{t,\boldsymbol{x}}(s) = \nabla_{\boldsymbol{p}}H\big(s,\boldsymbol{z}^*_{t,\boldsymbol{x}}(s),-\nabla_{\boldsymbol{z}}\Phi\big(s,\boldsymbol{z}^*_{t,\boldsymbol{x}}(s)\big),-\sigma(s,\boldsymbol{z}^*_{t,\boldsymbol{x}}(s))^\top\nabla^2_{\boldsymbol{z}}\Phi\big(s,\boldsymbol{z}^*_{t,\boldsymbol{x}}(s)\big)\big)\mathrm{d}s\\ \qquad\qquad\qquad + \sigma(s,\boldsymbol{z}^*_{t,\boldsymbol{x}}(s))\mathrm{d}W(s),\\ \boldsymbol{z}^*_{t,\boldsymbol{x}}(t) = \boldsymbol{x}.\end{cases} \tag{2.12}$$

These modified dynamics do not explicitly involve the control, which reduces the problem solely to the state variables. Note that in (2.12) the dependency of H on the Hessian of the value function could be omitted because the volatility term does not depend on the control. This idea was proposed for deterministic optimal control problems in [30].

2.4. FBSDE Formulation.

One way to avoid the need for a spatial discretization of the HJB equation (2.11) is to use a non-linear version of Feynman-Kac formula and obtain an equivalent system of stochastic differential equations. This idea has been applied to a variety of nonlinear parabolic/elliptic PDEs; see, for example, [1, 27, 6, 31, 33].

Our FBSDE system uses (2.12) as the forward system to sample trajectories. Along those trajectories, we note that the solution to the HJB equation (2.11) must satisfy the backward SDE following the Feynman-Kac formulae, see [40, Chapter 7]

$$\begin{cases} \mathrm{d}\Phi(s,\boldsymbol{z}(s)) & = \nabla_{\boldsymbol{z}}\Phi(s,\boldsymbol{z}(s))^\top \sigma(s,\boldsymbol{z}(s))\,\mathrm{d}W(s) - L(s,\boldsymbol{z}(s),\boldsymbol{u}^*(s))\mathrm{d}s, \\ \Phi(T,\boldsymbol{z}(T)) & = G(\boldsymbol{z}(T)). \end{cases} \tag{2.13}$$

It is important to stress that our choice of the forward system (2.12) is the key difference from other existing works. For example, [13] and [36] use the standard Brownian motion. While both of these choices lead to a valid FBSDE system for (2.11), we advocate for including the control in the dynamics as motivated by stochastic PMP (2.7). As our numerical experiments demonstrate, focusing the sampling along optimal trajectories can lead to more accurate and efficient value function approximations.

3. Neural Network Approach.

In this section, we present a neural network framework for approximating the value function of the stochastic optimal control problem defined by the objective functional (2.2) and dynamics (2.1). The theoretical foundation of our framework is given by the PMP, FBSDE, and Dynamic Programming as presented in the previous section. The key idea is to approximate the value function Φ in (2.3) by a neural network and compute the control using the feedback form (2.9). What distinguishes our framework from similar approaches such as [13, 36] is the use of the feedback form to guide the sampling during training. Thereby we seek to learn to explore the relevant part of the state space. We also derive and experiment with various loss functions that are based on the control objective (2.2), the BSDE (2.13), and the HJB (2.11).

3.1. Neural Network Approximation.

The first building block of our framework is to parameterize the value function using a neural network. Since finding an effective network architecture for any learning task is both crucial and an open research topic, we treat this as a modular component. Our framework can be used with any scalar-valued neural network that takes inputs in $\mathbb{R}^{d+1}$ as long as it is twice differentiable with respect to its last d inputs; this is to allow computations of $\nabla\Phi$.

Among the networks we use in our numerical experiments is the multi-layer perceptron (MLP) model used in [36]. As an alternative, which also satisfies the regularity needed, we propose the residual network also used for deterministic control in [30]. The network is given by

$$\Phi(\boldsymbol{y};\boldsymbol{\theta}) = \boldsymbol{w}^\top \mathcal{N}\mathcal{N}(\boldsymbol{y};\boldsymbol{\theta}_{\mathcal{N}\mathcal{N}}) + \frac{1}{2}\boldsymbol{y}^\top(\boldsymbol{A}^\top\boldsymbol{A})\boldsymbol{y} + \boldsymbol{b}^\top\boldsymbol{y} + c, \tag{3.1}$$

with trainable weights $\boldsymbol{\theta} = (\boldsymbol{w}, \boldsymbol{\theta}_{\mathcal{NN}}, \boldsymbol{A}, \boldsymbol{b}, c)$. Here the inputs $\boldsymbol{y} = (s, \boldsymbol{z}) \in \mathbb{R}^{d+1}$ correspond to time-space, $\mathcal{NN}(\boldsymbol{y}; \boldsymbol{\theta}_{\mathcal{NN}})\colon \mathbb{R}^{d+1} \to \mathbb{R}^m$ is a neural network, and $\boldsymbol{\theta}$ contains the trainable weights: $\boldsymbol{w} \in \mathbb{R}^m$, $\boldsymbol{\theta}_{\mathcal{NN}} \in \mathbb{R}^p$, $\boldsymbol{A} \in \mathbb{R}^{\gamma\times(d+1)}$, $\boldsymbol{b} \in \mathbb{R}^{d+1}$, $c \in \mathbb{R}$, where rank $\gamma = \min(10, d+1)$ limits the number of parameters in $\boldsymbol{A}^\top \boldsymbol{A}$. Here, $\boldsymbol{A}$, $\boldsymbol{b}$, and c model quadratic potentials, that is, linear dynamics; $\mathcal{NN}$ models nonlinear dynamics. For certain experiments, we may choose to omit the quadratic potential terms $\boldsymbol{A}$, $\boldsymbol{b}$ and c for comparison or simplicity reasons.

In our experiments, for $\mathcal{NN}$, we either use a MLP [12]

$$\begin{aligned} \boldsymbol{a}_0 &= \mathrm{act}(\boldsymbol{K}_0 \boldsymbol{y} + \boldsymbol{b}_0) \\ \boldsymbol{a}_{i+1} &= \mathrm{act}(\boldsymbol{K}_{\mathrm{i+1}} \boldsymbol{a}_{\mathrm{i}} + \boldsymbol{b}_{\mathrm{i+1}}), \quad 0 \le \mathrm{i} \le \mathrm{M}-2 \\ \mathcal{NN}(\boldsymbol{y}; \boldsymbol{\theta}_{\mathcal{NN}}) &= \mathrm{act}(\boldsymbol{K}_{\mathrm{M}} \boldsymbol{a}_{\mathrm{M}-1} + \boldsymbol{b}_{\mathrm{M}}), \end{aligned} \tag{3.2}$$

or a residual neural network (ResNet) [15]

$$\begin{aligned} \boldsymbol{a}_0 &= \mathrm{act}(\boldsymbol{K}_0 \boldsymbol{y} + \boldsymbol{b}_0) \\ \boldsymbol{a}_{i+1} &= \boldsymbol{a}_i + \mathrm{act}(\boldsymbol{K}_{\mathrm{i+1}} \boldsymbol{a}_{\mathrm{i}} + \boldsymbol{b}_{\mathrm{i+1}}), \quad 0 \le \mathrm{i} \le \mathrm{M}-2 \\ \mathcal{NN}(\boldsymbol{y}; \boldsymbol{\theta}_{\mathcal{NN}}) &= \boldsymbol{a}_{M-1} + \mathrm{act}(\boldsymbol{K}_{\mathrm{M}} \boldsymbol{a}_{\mathrm{M}-1} + \boldsymbol{b}_{\mathrm{M}}), \end{aligned} \tag{3.3}$$

with neural network weights $\boldsymbol{\theta}_{\mathcal{NN}} = (\boldsymbol{K}_0, \dots, \boldsymbol{K}_M, \boldsymbol{b}_0, \dots, \boldsymbol{b}_M)$ where $\boldsymbol{b}_i \in \mathbb{R}^m \ \forall i$, $\boldsymbol{K}_0 \in \mathbb{R}^{m\times(d+1)}$, and $\{\boldsymbol{K}_1, \dots, \boldsymbol{K}_M\} \in \mathbb{R}^{m\times m}$ with M being the depth of the network. The choice of the element-wise nonlinearity $\mathrm{act}(\cdot)$ is discussed in the respective experiments.

3.2. Training Problem. Ideally, we would choose $\boldsymbol{\theta}$ such that $\Phi(s, \boldsymbol{z}; \boldsymbol{\theta})$ is equal to the value function of the control problem globally, that is, for all $(s, \boldsymbol{z}) \in [t, T] \times \mathbb{R}^d$. Since this is known to be cursed by the dimensionality for reasonable problem sizes, we resort to a semi-global approach, which enforces this property at randomly sampled points in the space-time domain.

To generate samples, we first obtain initial states $\boldsymbol{x} \sim \rho$ from some (possibly Dirac) distribution ρ and then use an Euler Maruyama scheme with $N+1$ equidistant time points $s_0, \dots, s_N$ and step size $\mathrm{d}s = (T-t)/N$. This yields a state trajectory starting at $\boldsymbol{z}_0 = \boldsymbol{x}$ via

$$\boldsymbol{z}_{i+1} = \boldsymbol{z}_i + f(s_i, \boldsymbol{z}_i, \boldsymbol{u}_i)\mathrm{d}s + \sigma(s_i, \boldsymbol{z}_i)\mathrm{d}\boldsymbol{W}_i, \quad i = 0, \dots, N-1 \tag{3.4}$$

where $\mathrm{d}\boldsymbol{W}_i \sim \mathcal{N}(\boldsymbol{0}, \mathrm{d}s \cdot \boldsymbol{I}_d)$, and $\boldsymbol{u}_i = \boldsymbol{u}^*_{t,x}(s_i, \boldsymbol{z}_i)$ is the optimal control obtained from the feedback, that is, form (2.6)

$$\boldsymbol{u}_i^* \in \operatorname*{arg\,max}_{\boldsymbol{u}\in U} \mathcal{H}\left(s_i, \boldsymbol{z}_i, -\nabla\Phi(s_i, \boldsymbol{z}_i; \boldsymbol{\theta}), -\sigma(s_i, \boldsymbol{z}_i)^\top \nabla^2\Phi(s_i, \boldsymbol{z}_i; \boldsymbol{\theta}), \boldsymbol{u}\right).$$

A few comments are in place. First, it is important to note that due to the feedback form, the sampled trajectories depend on the parameters of the value function. Second, the addition of this drift term, motivated by control theory, is the key difference to neural network solvers for the more general class of semi-linear elliptic PDEs [13, 36]. Third, the drift term can also be motivated by the fact that for $\sigma \to 0$, the trajectories above approximate the characteristic curves of the non-viscous HJB equation; thereby our SOC approach coincides with that for deterministic OC in [30].

To further simplify the notations, we omit the subscript $\boldsymbol{z}$ in $\nabla_{\boldsymbol{z}}\Phi$ and $\nabla^2_{\boldsymbol{z}}\Phi$ for the rest of the paper. Furthermore, we collect the states, control, and noise along the discrete trajectories in (3.4) column-wise in the matrices

$$\boldsymbol{Z} \in \mathbb{R}^{d\times N}, \quad \boldsymbol{U} \in \mathbb{R}^{k\times N}, \quad \mathbf{dW} \in \mathbb{R}^{d\times N}.$$

To learn the parameters of the neural networks in an unsupervised way (that is, assuming neither analytic values of Φ nor optimal control trajectories), we approximately solve the minimization problem

$$\begin{aligned}\min_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x}\sim\rho} \big\{ \mathbb{E}_{\boldsymbol{Z},\boldsymbol{U},\mathbf{dW}|\boldsymbol{x}} \{ & \beta_1 P_{\mathrm{BSDE}}^p(\boldsymbol{Z},\boldsymbol{U},\mathbf{dW}) + \beta_2 P_{\mathrm{HJB}}^p(\boldsymbol{Z}) + \beta_3 J(\boldsymbol{Z},\boldsymbol{U}) \\ & + \beta_4 |G(\boldsymbol{z}_N) - \Phi(s_N,\boldsymbol{z}_N;\boldsymbol{\theta})|^p + \beta_5 |\nabla G(\boldsymbol{z}_N) - \nabla\Phi(s_N,\boldsymbol{z}_N;\boldsymbol{\theta})|^p \} \big\},\end{aligned} \tag{3.5}$$

where the terms in the objective function consist of penalty functions for violations of the BSDE system and the HJB PDE, the control objective, and penalty terms for the terminal condition, respectively, and are defined below. The exponent $p \in \{1,2\}$ allows one to choose between different norms for the loss function. In our numerical examples, we use $p = 1$ as it gives much faster convergence. The relative influence of each term is controlled by the components of $\beta \in \mathbb{R}_+^5$. Different choices of β allow us to experiment with different learning approaches; for example, setting $\beta_1 = \beta_4 = \beta_5 = 1$ and $\beta_2 = \beta_3 = 0$ provides the same loss function as in [36] while $\beta_1 = 0$ and $\beta_i > 0$, $i \in \{2,3,4,5\}$ gives the loss function used for deterministic OC problems in [30].

We penalize the violation of the BSDE (2.13) via

$$P_{\mathrm{BSDE}}(\boldsymbol{Z},\boldsymbol{U},\mathbf{dW}) = \sum_{i=0}^{N-1} |\Phi_{i+1}(\boldsymbol{\theta}) - \Phi_i(\boldsymbol{\theta}) + L(s_i,\boldsymbol{z}_i,\boldsymbol{u}_i)\mathrm{d}s - \nabla\Phi_i(\boldsymbol{\theta})^\top \sigma(s_i,\boldsymbol{z}_i)\mathrm{d}\boldsymbol{W}_i| \tag{3.6}$$

where we use the abbreviations $\Phi_i(\boldsymbol{\theta}) := \Phi(s_i,\boldsymbol{z}_i;\boldsymbol{\theta})$ and $\nabla\Phi_i(\boldsymbol{\theta}) := \nabla\Phi(s_i,\boldsymbol{z}_i;\boldsymbol{\theta})$. Similarly, the HJB penalty term reads

$$P_{\mathrm{HJB}}(\boldsymbol{Z}) = \mathrm{d}s \sum_{i=1}^{N} |H(s_i,\boldsymbol{z}_i,-\nabla\Phi_i(\theta),-\sigma(s_i,\boldsymbol{z}_i)^\top\nabla^2\Phi_i(\theta)) - \partial_s\Phi_i(\boldsymbol{\theta})|, \tag{3.7}$$

where $\nabla^2\Phi_i(\theta) := \nabla^2\Phi(s_i,\boldsymbol{z}_i;\theta)$, $\partial_s\Phi_i(\theta) := \partial_s\Phi(s_i,\boldsymbol{z}_i;\theta)$. Finally, we approximate the objective functional via

$$J(\boldsymbol{Z},\boldsymbol{U}) = G(\boldsymbol{z}_N) + \mathrm{d}s \sum_{i=1}^{N} L(s_i,\boldsymbol{z}_i,\boldsymbol{u}_i).$$

In principle, any stochastic approximation approach can be used to approximately solve the above optimization problem. Here, we use Adam [19] and sample a minibatch of trajectories originating in i.i.d. samples from ρ.

3.3. Implementation. We implement and test our proposed approach in two software environments.

To obtain a direct comparison with [36] we modify the FBSNN code accompanying the paper. To this end, we created a publicly available fork at https://github.com/EmoryMLIP/FBSNNs. Our two main modifications are adding the proposed drift to the forward dynamics and adding the control objective in the training loss. Other parameters, including the choice of neural network model, are kept unchanged.

In order to further simplify the experimentation, we also implement our own PyTorch code available at https://github.com/EmoryMLIP/NeuralSOC. Our implementation contains all loss terms in (3.5). We implement both sampling techniques: pure random walk and the proposed one informed by PMP. This facilitates comparisons of our approach with other available methods and simplifies developing new examples.

We tested most of our examples using either Intel Xeon E5-4627 CPU or Nvidia P100 GPU.

4. Numerical Experiments.

We test the efficacy of our proposed algorithm on several different SOC problems. In subsection 4.1, we introduce a two-dimensional trajectory planning problem to visualize the difference between purely random exploration and our proposed sampling scheme. To illustrate the accuracy of the learned value function, we compare it with the value function obtained by solving the corresponding HJB PDE using a finite element method (FEM). The goal of this experiment is to compare the accuracy of the neural network and FEM approximation. In subsection 4.2, we compare our approach to those in [8, 13] using a 100-dimensional benchmark problem. For the original version of this problem, our method shows faster initial convergence and time-to-solution with comparable accuracy. We modify the terminal cost of this problem to further highlight the importance of the feedback form in the sampling (see subsection 4.2.3). Lastly in subsection 4.3 we also test our method on a 12-dimensional problem with nonlinear dynamics, showing that our method generates relatively accurate solutions under complex dynamics.

4.1. 2D Trajectory Planning Problem.

To visualize the behavior of our PMP-based sampling approach, we consider a two-dimensional test problem.

The problem consists of planning an optimal trajectory from the initial state $\boldsymbol{x} \sim \rho = \mathcal{N}((-1.5,-1.5)^\top, 0.4 \cdot \boldsymbol{I}_2)$ to the target $\boldsymbol{x}_{\text{target}} = (1.5, 1.5)^\top$. To make the problem interesting, a hill is placed at the origin, denoted by $Q(\boldsymbol{z})$, which adds height-dependent cost for traveling around that region. In our experiments, $Q(\boldsymbol{z})$ is defined by a two-dimensional Gaussian density with mean zero and covariance of $0.4 \cdot \boldsymbol{I}_2$ scaled by a factor of 50.

The dynamics for the problem read

$$f(s, \boldsymbol{z}, \boldsymbol{u}) = \boldsymbol{u} \quad \text{and} \quad \sigma = \begin{bmatrix} 0.2 & -0.4 \\ -0.4 & 0.2 \end{bmatrix}. \tag{4.1}$$

The choice of non-scalar σ adds to the complexity of the problem by changing the behavior of the standard Brownian motion, see Figure 1.

The running cost and terminal cost of the problem are given, respectively, by

$$L(s, \boldsymbol{z}, \boldsymbol{u}) = \frac{1}{2}\|\boldsymbol{u}\|^2 + Q(\boldsymbol{z}) \quad \text{and} \quad G(\boldsymbol{z}) = 50 \cdot \|\boldsymbol{z} - \boldsymbol{x}_{\text{target}}\|^2. \tag{4.2}$$

The corresponding HJB equation is

$$\partial_s \Phi(s, \boldsymbol{z}) + \frac{1}{2}\operatorname{tr}(\sigma\sigma^\top \nabla^2 \Phi(s, \boldsymbol{z})) - \frac{1}{2}\|\nabla \Phi(s, \boldsymbol{z})\|^2 + Q(\boldsymbol{z}) = 0. \tag{4.3a}$$

with terminal condition

$$\Phi(T, \boldsymbol{z}) = G(\boldsymbol{z}). \tag{4.3b}$$

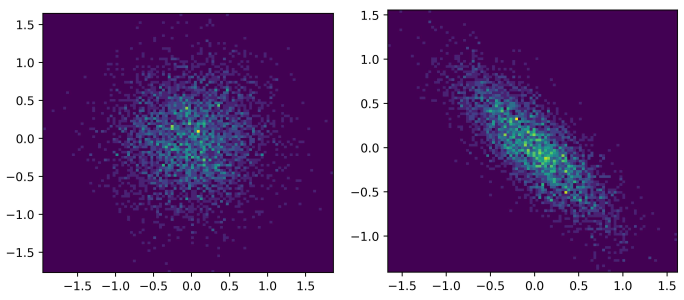


Fig. 1: Action of σ in (4.1) on standard Gaussian distribution (Left) warps it diagonally (Right). This would affect the solution of the problem.

4.1.1. Finite Element Method. Since it is not obvious how to solve the HJB equation (4.3) analytically, we approximately solve it using a finite element method (FEM) to obtain a baseline for this problem.

We approximate the value function by solving the HJB PDE (4.3) on the domain $\Omega = [-3, 3] \times [-3, 3]$ with homogeneous Neumann boundary conditions,

$$\frac{\partial \Phi}{\partial \hat{\boldsymbol{n}}}(s, \boldsymbol{z}) = 0, \text{ on } \partial\Omega, \ \forall s < T,$$

where $\hat{\boldsymbol{n}}$ denotes the unit normal vector. Since the diffusion coefficient σ is independent of time and space, $\operatorname{tr}(\sigma\sigma^\top \nabla^2 \Phi(s, \boldsymbol{z})) = \operatorname{div}(\sigma\sigma^\top \nabla \Phi(s, \boldsymbol{z}))$, which we use to derive a weak form of the PDE. Using the implicit Euler discretization in time on a partition of $[0, T]$ into N sub-intervals with uniform step size, ds, yields

$$\frac{\Phi_{n+1} - \Phi_n}{\mathrm{d}s} + \frac{1}{2}\operatorname{div}(\sigma\sigma^\top \nabla \Phi_n) - \frac{1}{2}\|\nabla \Phi_n\|^2 + Q = 0, \quad n = N, N-1, \dots, 0,$$

where Φ_n denotes the approximated solution $\Phi(t_n, \cdot)$, at $t_n = n \cdot \mathrm{d}s$ and $\Phi_{N+1} = G(\cdot)$. Then, using Green's formula, the weak problem at the n-th time step consists of finding $\Phi_n \in H^1(\Omega)$ such that

$$\int_\Omega (\Phi_{n+1} - \Phi_n) v \mathrm{d}\boldsymbol{z} - \mathrm{d}s \frac{1}{2} \int_\Omega \sigma\sigma^\top \nabla \Phi_n \cdot \nabla v \mathrm{d}\boldsymbol{z} + \mathrm{d}s \int_\Omega \left(Q - \frac{1}{2}\|\nabla \Phi_n)\|^2 \right) v \mathrm{d}\boldsymbol{z} = 0,$$

for all test functions $v \in H^1(\Omega)$. Here, $H^1(\Omega)$ denotes the Hilbert Sobolev space defined by $H^1(\Omega) = \{v \in L^2(\Omega) | \nabla v \in L^2(\Omega)\}$.

To solve the weak problem we use FEniCS [25], we create a triangular mesh for Ω and use $\mathcal{P}_1$ Lagrange finite elements to discretize Φ in space. We discretize Ω using 150 mesh points in each dimension, summing up to a total of 22,500 degrees of freedom, and use the step-size of d$s = 0.001$ in time. At each time step, we use Newton's method to solve for Φ_n, with relative error and absolute error tolerance for the solver set to 10^{-6} and 10^{-10}, respectively. We denote the FEM solution by Φ_{FEM}.

In Figure 2 we plot the solution Φ_{FEM} as well as the optimal control policy at initial time $s = 0$ obtained via the feedback form. We also present trajectories originating from some randomly chosen initial states following the optimal policy. As expected, the trajectories travel from the initial points to the target while avoiding the obstacle in the center of the domain.

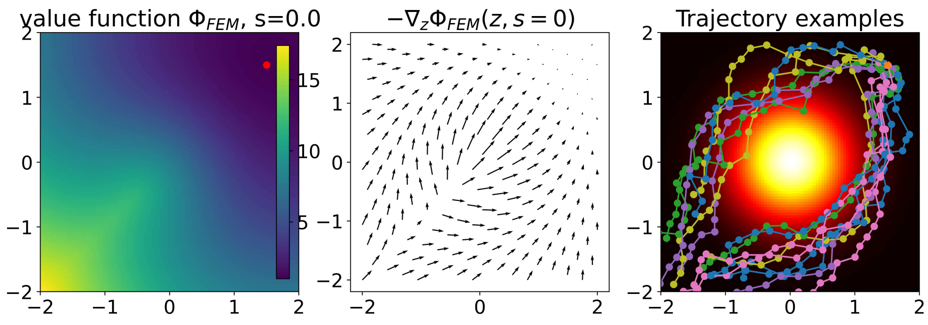


Fig. 2: Results of the two-dimensional test problem. Left: Value function approximation $\Phi_{\mathrm{FEM}}(0,\cdot)$. Middle: Quiver plot of optimal controls at $s = 0$. Right: Trajectories generated from randomly chosen initial states.

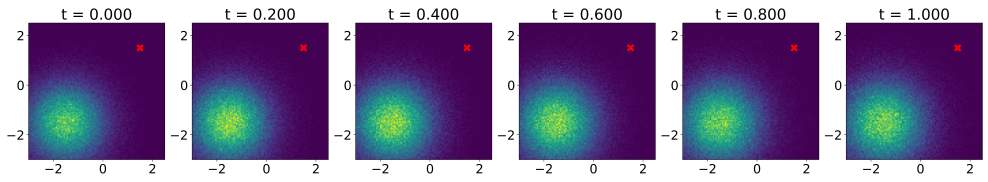
In Table 2, we evaluate the control objective, J, for some fixed initial state. We notice that the estimated value matches with $\Phi_{\mathrm{FEM}}(0)$, suggesting that the FEM solution is an accurate approximation of the true Φ. It is also worth pointing out that FEM is sufficient for the 2D parabolic equation we have here since a variational form is explicitly available. However for problems without a easily accessible variational form, one may want to resort to methods in [5, 23] for true solutions.

4.1.2. Neural Network Approach. For the problem defined in (4.1) and (4.2), the forward SDE (3.4) simplifies to

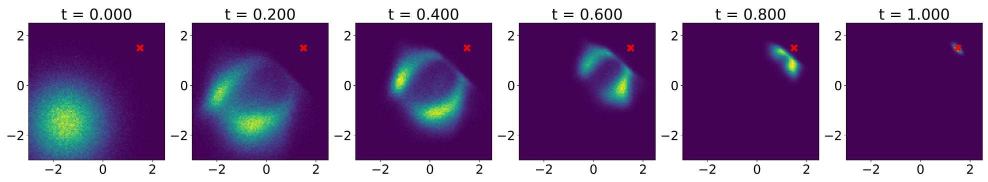
$$\boldsymbol{z}_{i+1} = \boldsymbol{z}_i - \nabla\Phi(s_i, \boldsymbol{z}_i)\mathrm{d}s + \sigma \mathrm{d}\boldsymbol{W}_i. \tag{4.4}$$

Following our proposed method in (3.1), we approximate the value function using a three-layer residual neural network with 32 neurons per layer. We do not include the quadratic terms in the network for this experiment since the simpler structure was already sufficient for solving this problem. Overall, the model consists of 1,217 trainable parameters. We choose tanh as the activation function for all but the final layer of the network, the final layer does not have an activation function. We use the penalty parameters $\beta = (1.0.1.0, 1.0, 1.0, 0.0)$, that is, we enable both the penalty terms, P_{BSDE} and P_{HJB} in (3.5). To approximately solve (3.5) we use a total of 6,000 steps of the Adam optimizer with a batch size of 64. We start with a learning rate of 0.01 and divide it by 10 every 1800 iterations. The average cost per iteration is about 0.22s using an NVIDIA P100 GPU. Note that for the chosen σ in (4.1) full Hessian information of the value function, $\nabla^2\Phi$, is required to calculate P_{HJB}. To this end, we use the efficient implementation in the package hessQuik [29]. We refer to the neural network approximated solution as Φ_{NN} in the following sections. We also notice that sampling the initial states from a slightly larger area during training often helps the robustness of the learned model. Given the stochastic nature of the problem and the random initialization of neural network weights, each training sequence can produce a slightly different model. To account for this, we repeat the training ten times and obtain neural network approximations of the value functions $\Phi_{\mathrm{NN}}^{(j)}$, where $1 \leq j \leq 10$. We compare the resulting models to the FEM solution in the next section.

To gain more insight into the sampling, we store all states visited during training and plot them as two-dimensional histograms for different time points (left to right)



(a) Training samples with pure random walk as FSDE as also used in [13] and [36].



(b) Training samples with PMP-based drift term.

Fig. 3: We visualize training samples of a pure random walk sampler (top row) and our proposed PMP-based sampler (bottom row) for the two-dimensional test problem. At six time points (left to right), we visualize the sampled states as two-dimensional histograms. As expected, the pure random walk explores the area around the initial state in all (even suboptimal) directions, while the proposed approach learns to sample around approximately optimal trajectories.

in Figure 3. We compare the proposed PMP-based sampling (Figure 3b) to the purely noisy dynamics (Figure 3a), that is, without drift, as used in [13, 36]. As expected, the use of purely noisy dynamics leads to the sampling of points only around the initial states in all (even sub-optimal) directions with almost no samples close to the target. On the other hand, with the use of drift term, the sampled states visit the paths between the initial and target states.

Another way to interpret the histogram plots in Figure 3 is by observing the semi-global nature of our neural network approach for SOC problems. Since the loss function in (3.5) penalizes the HJB and BSDE losses in a neighborhood of points sampled using the forward SDE, one would expect the trained model to be more reliable in regions that are frequently visited.

4.1.3. Comparison. In this subsection, we compare the neural network models $\Phi_{\mathrm{NN}}^{(1)}, \ldots, \Phi_{\mathrm{NN}}^{(10)}$ and Φ_{FEM} along the approximately optimal trajectories. Specifically, we randomly sample initial states from ρ and simulate the trajectories using the trained models. We believe that this approach enables a meaningful comparison since the training procedure focuses on those parts of the state space visited by the trajectories and hence the neural networks approximate the value function semi-globally.

For each trained model, we record all the sampled states visited at times $s \in \{0, 0.5, 0.9\}$ while following the corresponding learned policy. We then compare the learned value functions Φ_{NN}^{j} with the reference solution Φ_{FEM} at all these points. In Figure 4, we plot the comparison for times $s \in \{0, 0.5, 0.9\}$ along the rows. The first, second, and fourth columns represent $\Phi_{\mathrm{NN}}^{(1)}$, $\Phi_{\mathrm{NN}}^{(2}$), and Φ_{FEM} at the sampled points, respectively. We observe that value function estimates look similar. The third column shows the average of the ten learned value functions obtained from the ten training

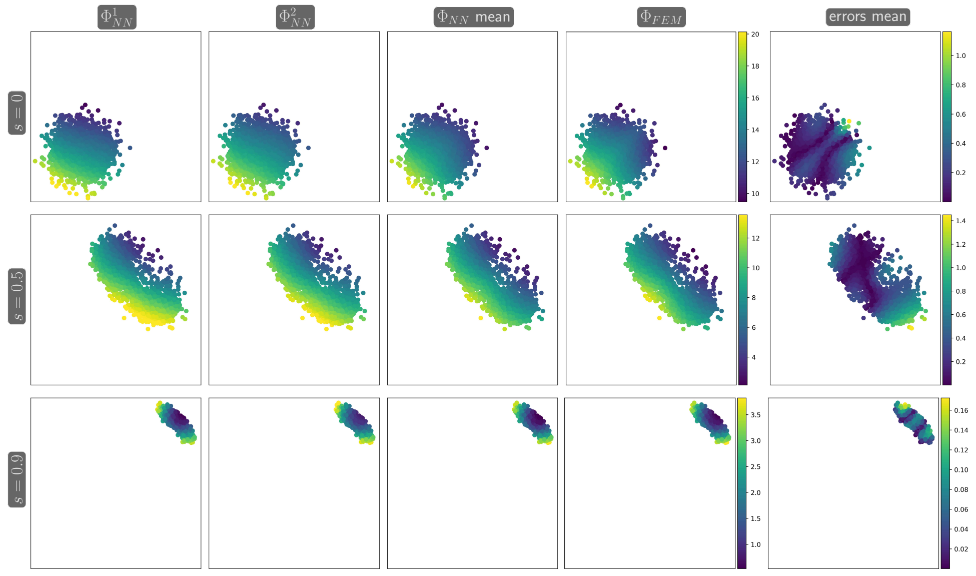


Fig. 4: Comparison between learned value function (First 3 columns, including both individual model and model average) and the FEM solution (fourth column) at different time shots $s = 0, 0.5$ and 0.9. From the errors (fifth column), the neural network solution matches the FEM solution closely over the sampled region.

sequences. Lastly, the last column displays the average absolute mean errors between the learned value functions and Φ_{FEM}.

In Table 1, we compare mean of the absolute and relative errors between Φ_{FEM} and $\Phi^{(j)}_{\text{NN}}$, $1 \leq j \leq 10$, across the sampled points shown in Figure 4 for all ten trained models, computed via

$$\text{AE}(s) = \frac{1}{n_{\text{samples}} \times n_{\text{models}}} \sum_{i=1}^{n_{\text{samples}}} \sum_{j=1}^{n_{\text{models}}} \left| \Phi^{(j)}_{\text{NN}}(s, \boldsymbol{z}_i) - \Phi_{\text{FEM}}(s, \boldsymbol{z}_i) \right| \tag{4.5}$$

and

$$\text{RE}(s) = \frac{1}{n_{\text{samples}} \times n_{\text{models}}} \sum_{i=1}^{n_{\text{samples}}} \sum_{j=1}^{n_{\text{models}}} \frac{\left| \Phi^{(j)}_{\text{NN}}(s, \boldsymbol{z}_i) - \Phi_{\text{FEM}}(s, \boldsymbol{z}_i) \right|}{|\Phi_{\text{FEM}}(s, \boldsymbol{z}_i)|}. \tag{4.6}$$

Our observations indicate that the relative error is smallest at the initial time and increases over time, while the average absolute error remains fairly constant across all states and time intervals. One possible explanation is that we include the control objective, J, in the training loss function. Furthermore, the errors across all the trained models display a relatively low standard deviation, indicating that our proposed training scheme is robust to random initialization.

In Table 2, we compare the value function approximation for one of the trained models, $\Phi^{(1)}_{\text{NN}}$, to the value of the control objective, $J(-\nabla\Phi_{\text{NN}})$, at the initial state $\boldsymbol{x} = (-1.5, -1.5)^\top$ and time $t = 0$. Since the system dynamics are stochastic, we generate $12,000$ trajectories starting from $\boldsymbol{x}$ using the learned feedback control to

Time snapshot	$s = 0$	$s = 0.5$	$s = 0.9$
AE (mean $\pm$ std)	0.31 ± 0.17	0.47 ± 0.44	0.17 ± 0.18
RE (mean $\pm$ std)	0.02 ± 0.01	0.06 ± 0.06	0.15 ± 0.17

Table 1: Average absolute and relative error between $\Phi_{\text{NN}}^{(1)}, \dots, \Phi_{\text{NN}}^{(10)}$ and Φ_{FEM} across all sampled points at different time steps.

Initial state	$\Phi_{\text{FEM}}(0)$	$\Phi_{\text{NN}}(0)$	J_{FEM}	J_{NN}
$\boldsymbol{x}_{\text{init}} = (-1.5, -1.5)^\top$	14.67	14.48	14.68	15.33

Table 2: Discrepancy between the value function Φ and the control objective J at some initial state.

calculate the control objective J for each trajectory. We then use the sample average as a proxy for the expected value. We use a finer step size of $\mathrm{d}s = 0.005$ than the one used in training to get an accurate approximation. We observe that the discrepancy between the value estimate and the actual cost is almost negligible for the FEM solution. For the neural network approximation, the value estimate is about 4% smaller than the actual control objective, which indicates that the value estimates can be overly optimistic.

On the hardware used for our experiments, both approaches showed comparable time-to-solution. The neural network training took approximately 20 minutes using the GPU, while the FEM solution was obtained in roughly one hour using the CPU. However, the FEM approach requires a computational mesh, making it infeasible for $d > 4$, which is the primary use case for our proposed method.

4.2. 100-dimensional example. We consider the 100-dimensional benchmark SOC problem also used in [8, 13] with initial state $\boldsymbol{x} = (0, 0, \dots, 0)^\top \in \mathbb{R}^{100}$ corresponding to time $t = 0$. The drift and diffusion of the system are given by

$$f(s, \boldsymbol{z}, \boldsymbol{u}) = 2\boldsymbol{u} \quad \text{and} \quad \sigma = \sqrt{2},$$

respectively. The terminal and Lagrangian cost are

$$G(\boldsymbol{z}) = \ln\left(\frac{1 + \|\boldsymbol{z}\|^2}{2}\right), \quad \text{and} \quad L(s, \boldsymbol{z}, \boldsymbol{u}) = \|\boldsymbol{u}\|^2, \tag{4.7}$$

respectively. We compute the Hamiltonian (2.4) as

$$\begin{aligned} H(s, \boldsymbol{z}, \boldsymbol{p}, \boldsymbol{M}) &= \sup_{\boldsymbol{u} \in U} \left\{ \frac{\sigma}{2} \operatorname{tr}(\boldsymbol{M}) + \boldsymbol{p} \cdot f(s, \boldsymbol{z}, \boldsymbol{u}) - L(s, \boldsymbol{z}, \boldsymbol{u}) \right\} \\ &= \sup_{\boldsymbol{u} \in U} \left\{ \frac{1}{\sqrt{2}} \operatorname{tr}(\boldsymbol{M}) + \boldsymbol{p} \cdot 2\boldsymbol{u} - \|\boldsymbol{u}\|^2 \right\} \end{aligned}$$

Using the first-order necessary condition we get

$$0 = 2\boldsymbol{u} - 2\boldsymbol{p} \implies \boldsymbol{u} = \boldsymbol{p},$$

and using this closed form for $\boldsymbol{u}$, the Hamiltonian is given by

$$H(s,\boldsymbol{z},\boldsymbol{p},\boldsymbol{M}) = \frac{1}{\sqrt{2}}\operatorname{tr}(\boldsymbol{M}) + \|\boldsymbol{p}\|^2.$$

Hence, the HJB equation satisfied by the value function, $\Phi(\cdot,\cdot)$, reads

$$\frac{\partial}{\partial s}\Phi(s,\boldsymbol{z}) + \Delta\Phi(s,\boldsymbol{z}) - \|\nabla\Phi(s,\boldsymbol{z})\|^2 = 0, \quad \Phi(T,\boldsymbol{z}) = G(\boldsymbol{z}). \tag{4.8}$$

and its solution is given by

$$\Phi(s,\boldsymbol{z}) = -\ln\left(\mathbb{E}\left(\exp\left(-G\left(\boldsymbol{z}+\sqrt{2}\,\mathrm{d}W(T-s)\right)\right)\right)\right), \tag{4.9}$$

which we use to test the performance of our method.

Finally, we note that the forward SDE (3.4) we propose to use for sampling the state space simplifies to

$$\boldsymbol{z}_{i+1} = \boldsymbol{z}_i - 2\,\nabla_{\boldsymbol{z}}\Phi(s_i,\boldsymbol{z}_i)\mathrm{d}s + \sqrt{2}\mathrm{d}\boldsymbol{W}_i. \tag{4.10}$$

4.2.1. The importance of sampling. To demonstrate the impact of using the feedback form to sample the state space, we use the same neural network model as in [36], which is given by a five-layer feed-forward neural network with 256 neurons per hidden layer to approximate the solution $\Phi(s,\boldsymbol{z})$. We partition the time interval $[0,1]$ using 50 uniformly spaced points. We use the same penalty parameters as in the original code, that is, $\beta = (1,0,20,1,1)$. We use the Adam optimizer [19] to update the parameters of the network with a batch size of 64 using 50,000 iterations. The average cost per 100 iterations was 27s using the CPU. For the following experiments, we use (3.5) excluding P_{HJB} penalty and compare our method with FBSNNs in [36].

Method	20k iterations		50k iterations	
	RE	RE_0	RE	RE_0
FBSNN	0.54%	0.12%	0.39%	0.045%
Ours	0.48%	0.0083%	0.39%	0.012%

Table 3: Relative errors for (4.8) obtained using our method and method in [36]

In Figure 5 we plot the exact solution (black-dashed line) (4.9), the learned solution using our approach (blue-solid line) and the solution learned using FBSNNs (red-solid line) along five random trajectories. In the top row, we present the results obtained after training the networks for 20,000 iterations with a learning rate of 10^{-3} and the bottom row presents the results after training the networks for 20K and 30K iterations with learning rates 10^{-3} and 10^{-4}, respectively. These results suggest that our approach approximates the value function better, especially in early iterations, as compared to the FBSNNs.

In Table 3, we also compare the learned solutions to the exact solution Φ in (4.9) by computing the average relative errors,

$$RE = \frac{\|\Phi(\cdot,\cdot;\boldsymbol{\theta}) - \Phi(\cdot,\cdot)\|_2}{\|\Phi\|_2}, \quad RE_0 = \frac{|\Phi(0,\boldsymbol{z}(0);\boldsymbol{\theta}) - \Phi(0,\boldsymbol{z}(0))|}{|\Phi(0,\boldsymbol{z}(0))|},$$

for ten random trajectories. Our method attains lower errors, especially for the initial values and at the earlier iterations.

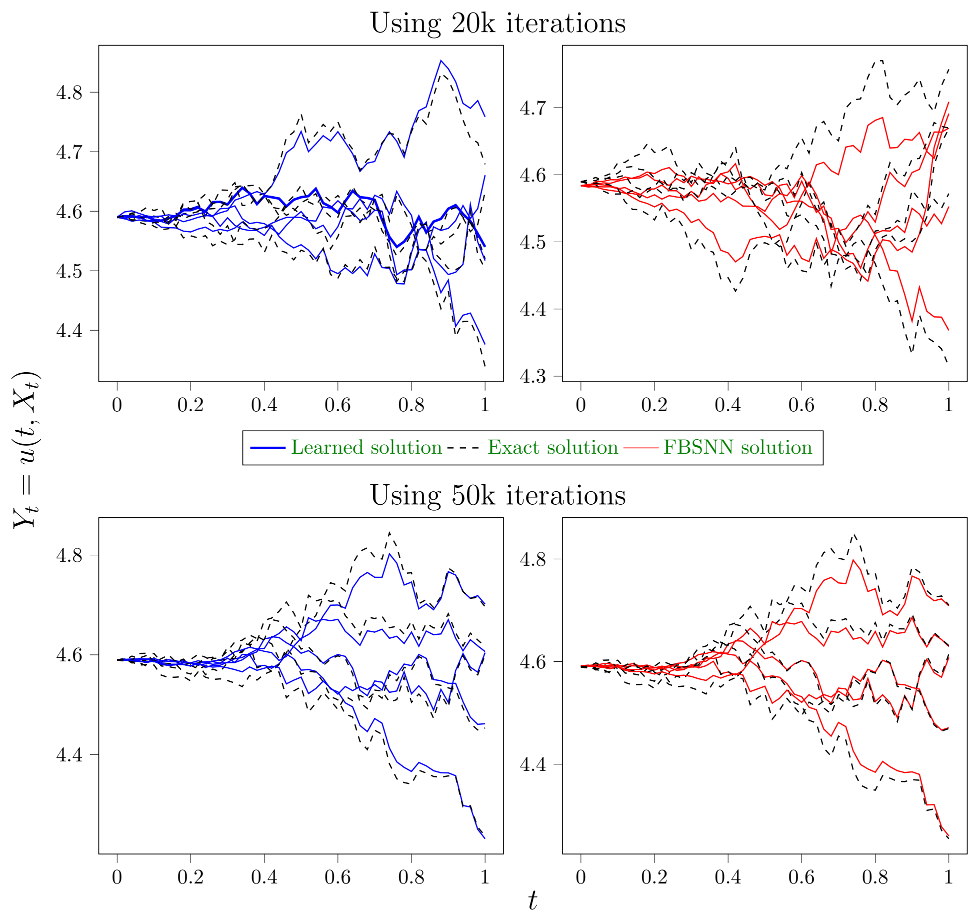


Fig. 5: Solution to (4.8) obtained using our method (left column) and the method in [36] (right column)

4.2.2. Initial states from a distribution. We demonstrate the versatility of our method beyond fixed initial states, especially in addressing input states following a given distribution. Specifically, we sample $\boldsymbol{x}$ from a distribution $\rho = \mathcal{N}(\mathbf{0}, 0.5 \cdot \boldsymbol{I}_{100})$. We repeat the training process with 20k iterations, maintaining the same hyperparameters, but increasing the batch size to 512 from 64 and choosing $\beta_3 = 50$. In Figure 6, we present the mean and variance of the relative errors of the errors relative to (4.9) in the learned value function for ten random trajectories. As expected to the higher complexity of the problem, the maximum relative error over the time interval increased to 1.5%, which is slightly larger than in the original problem.

4.2.3. Shifted target. In the example above, the minimizer of the terminal function coincides with the initial state $\boldsymbol{x} = (0, 0, \ldots, 0)^\top$. Therefore, even a random walk without drift (as used in [36, 13]) will sample around the optimal terminal state, which is critical to accurately approximate the value function. This also means that after training using our approach, the drift term in the sampler is relatively small and that the above experiment does not fully show the advantages of our method.

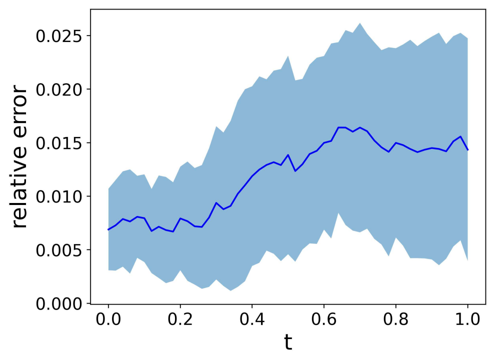


Fig. 6: Mean and variance of the errors relative to (4.9) in the learned value function for ten random trajectories obtained by sampling initial states from a distribution using our method after 20k iteration.

To shed more light on the importance of sampling, we modify the terminal cost to

$$G(z) = 1000 \ln \left(\frac{1 + \|\boldsymbol{z} - \boldsymbol{z}_{\text{target}}\|^2}{2} \right),$$

with $\boldsymbol{z}_{\text{target}} = (3, 3, \ldots, 3)^T$, so that the target for the state variable $\boldsymbol{z}$ at final time T no longer coincides with the initial state. Similar to the two-dimensional test problem in subsection 4.1, solving the modified problem now requires sampling around the target and we expect to benefit from the added drift term.

We compare our method to FBSNNs on the modified problem keeping the same network structure and hyper-parameters. We use a smaller $\sigma = \frac{2\sqrt{2}}{5}$ to improve training speed. We evaluate the performance of the methods using the objective functional J defined in (2.2) at the control obtained from the feedback form via the respective value function approximations. For this experiment, we use a GPU to train and the results of this comparison are shown in Figure 7.

To reduce the effect of the Brownian motion, we run the experiments for each method on the same problem five times and plot the average values corresponding to training iterations. Furthermore, since the primary goal for this example is to explore the difference between sampling strategies, we select much higher weights for the control objective such that we have faster initial convergence for the control variable.

As can be seen in Figure 7 (left), our method not only yields faster initial convergence but also achieves a considerably lower control objective. This indicates that the controls obtained from our approach are more effective, that is, they are closer to optimal. It is also worth pointing out that due to the high terminal cost we assigned when designing the problem, it takes very few iterations to locate the correct state-time region that the optimal solution resides in. Since FBSNNs use a Brownian motion with no drift, the sampling is unlikely to discover the target. Consequently, the generated trajectories in Figure 7 (right) from our method approximately reach the target, while the trajectories obtained from the FBSNN method stay closer to the initial state. Do note additional hyperparameter tuning and training will be needed if one aims to solve the underlying HJB equation accurately as well.

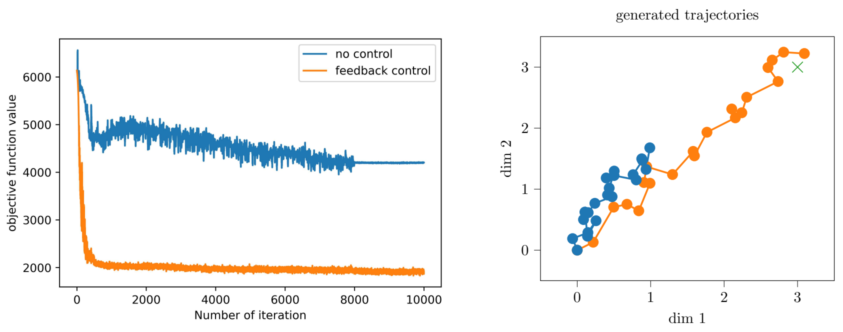


Fig. 7: Computational results for the modified 100-dimensional benchmark problem in subsection 4.2.3. Left: Control objective for both methods given the same initial state, the blue line represents results using FBSNNs in [36], and the orange line denotes our method. Right: Trajectory examples generated using learned value functions on two randomly selected dimensions. The orange line represents our method and blue line FBSNNs.

4.3. Quadcopter Problem with Nonlinear Dynamics. We test our proposed method's ability to deal with nonlinear dynamics using the stochastic version of the quadcopter trajectory planning problem also considered in [26, 30]. We sample initial states from a Gaussian distribution centered at $\boldsymbol{x} = [-1.5, -1.5, -1.5, 0, \ldots, 0]^\top$ and set $\boldsymbol{x}_{\text{target}} = [2, 2, 2, 0, \ldots, 0]^\top$. Here $d = 12$, given the state variable $\boldsymbol{z} = [z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8, z_9, z_{10}, z_{11}, z_{12}]^\top$ the dynamics read

$$f(s, \boldsymbol{z}, \boldsymbol{u}) = \begin{cases} z_7 \\ z_8 \\ z_9 \\ z_{10} \\ z_{11} \\ z_{12} \\ \dfrac{u_1}{m} f_7(z_4, z_5, z_6) = \dfrac{u_1}{m}(\sin(z_4)\sin(z_6) + \cos(z_4)\sin(z_5)\cos(z_6)) \\ \dfrac{u_1}{m} f_8(z_4, z_5, z_6) = \dfrac{u_1}{m}(-\cos(z_4)\sin(z_6) + \sin(z_4)\sin(z_5)\cos(z_6)) \\ \dfrac{u_1}{m} f_9(z_5, z_6) - g = \dfrac{u_1}{m}(\cos(z_5)\cos(z_6)) - g \\ u_2 \\ u_3 \\ u_4 \end{cases}$$

The controls for the problem are $\boldsymbol{u} = [u_1, u_2, u_3, u_4]^\top \in \mathbb{R}^4$. We assume that both the mass m and gravity g are given. The control objective encompasses $L(\boldsymbol{u}(s, \boldsymbol{z})) = 2 + \|\boldsymbol{u}(s, \boldsymbol{z})\|^2$, and $G(\boldsymbol{z}(T)) = 2500 \cdot \|\boldsymbol{z}(T) - \boldsymbol{x}_{\text{target}}\|^2$. The feedback form with

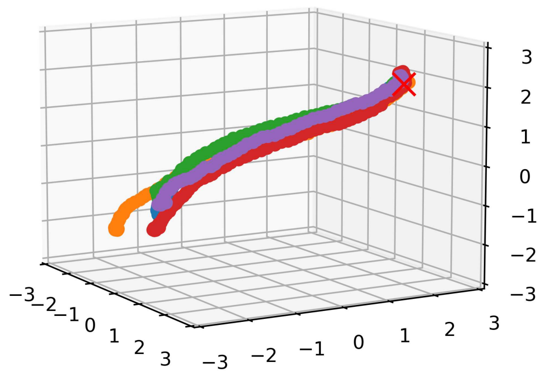


Fig. 8: Flight path examples using the learned controller. The target is depicted by a red cross.

J at $\boldsymbol{x}$	evaluated at $\sigma = 0.2$	evaluated at $\sigma = 0$
Deterministic Model	9.33×10^3	2.18×10^3
Our Model	3.34×10^3	-
Accurate J (deterministic)	-	2.18×10^3

Table 4: Approximated control objective J for initial state $\boldsymbol{x} = [-1.5, -1.5, -1.5, 0, \dots, 0]^\top$. Note the deterministic solution is trained with $\sigma = 0$ while ours with $\sigma = 0.2$. Value for the accurate solution comes from [30].

respect to Φ for this problem takes the form:

$$u_1 = \frac{-1}{2m}\left(f_7 \frac{\partial \Phi}{\partial z_7} + f_8 \frac{\partial \Phi}{\partial z_8} + f_9 \frac{\partial \Phi}{\partial z_9}\right),$$
$$u_2 = -\frac{1}{2}\frac{\partial \Phi}{\partial z_{10}}, \quad u_3 = -\frac{1}{2}\frac{\partial \Phi}{\partial z_{11}}, \quad u_4 = -\frac{1}{2}\frac{\partial \Phi}{\partial z_{12}}.$$

The HJB equation and BSDE can be derived using the feedback form accordingly under section 2. We choose $\sigma = 0.2$ for the problem.

We use the network in (3.1) featuring two layers and 128 neurons per layer for the ResNet. The penalty term is $\beta = (0.1, 0.1, 1.0, 0.1, 0.1)$. We train the network using 6000 iterations of Adam with a batch size of 128. The learning rate initiates at 0.01 and is halved every 1600 iterations. Since the dynamics in this example is more complex, we discretize the SDE with 100 equidistant steps between $t = 0$ and $T = 1$ to enhance accuracy. On average, every training iteration took around 2 seconds on the GPU.

The visualization of the trained policy in Figure 8 shows that the flight trajectories reach the given target from various starting points. While we are not aware of an analytical solution to the problem, we compare the performance of our policy to the pre-trained policy from [30], which is trained for the deterministic problem instance. For each policy, we compute the average value of the control objective over 15,000 randomly chosen trajectories, each using 200 time steps and report the results in Table 4. As to be expected, while the deterministic solution works well for $\sigma = 0$, its performance drops notably when the objective is evaluated with $\sigma = 0.2$. Since the stochasticity of the dynamics is taken into account during training, our model performs better in this case.

5. Discussion.

We propose a neural network approach for approximately solving Hamilton-Jacobi-Bellman PDEs arising in high-dimensional stochastic optimal control. Similar to existing approaches [36, 13], we parameterize the value function with a neural network and experiment with different losses to train the network weights. What sets our work apart from these works is the use of feedback form given by the stochastic Pontryagin maximum principle to design the forward SDE used to explore the state space during training.

Using an intuitive two-dimensional test problem, we visualize that the improved sampling strategy allows us to effectively learn the value function and determine the relevant regions of the state space; see subsection 4.1. Based on this insight, we modify the 100-dimensional test problem also used in [36, 13] by shifting the minimizer of the terminal costs; see subsection 4.2.3. Thereby, we demonstrate that our proposed method dramatically improves the quality of the obtained control. Using a 12-dimensional quadcopter example whose dynamic is nonlinear in the states, we also demonstrate that our model can handle complicated dynamics; see subsection 4.3.

Hyperparameter tuning is crucial in training any neural network model. In our case, choosing different weights for different penalties in the loss function can result in varying outcomes. In our experiments, we found that including the control objective, J, in the training loss is crucial to obtaining accurate results. Without this term, we encountered examples where the optimal paths can not be recovered correctly despite having a lower P_{HJB} loss. One possible cause is that the sampling trajectories, which are also used to compute P_{HJB}, do not effectively identify the relevant regions of the state space. One heuristic we found effective was to use a relatively large weight for the control objective J, especially at the beginning of the training. Once the control objective is sufficiently small, we suggest experimenting with the weights corresponding to P_{BSDE} and P_{HJB}.

Our numerical experiments also show that the modified forward SDE and the control objective can lead to faster initial convergence compared to the approaches in [36, 13] (see subsection 4.2).

We refer to our method as a semi-global method for solving the HJB equation since we do not aim at estimating the value function accurately globally. Instead, we seek to approximate the value function well for states likely to be visited by optimal trajectories of the SOC problem. In theory, any point in the state space has a positive probability of being visited due to the stochasticity in the dynamics. However, the histogram plots in Figure 3 suggest that the density of the optimal trajectory is concentrated in a small subset of the state space. Therefore, focusing the exploration on these subsets may have practical advantages over FSDEs that are pure random walks.

Another benefit of our proposed forward SDE compared to purely random exploration is that it coincides with the characteristic curves of the HJB equation as the stochasticity of the system is reduced. Therefore, our work can be seen as an extension of the neural network approaches for deterministic control problems in [30].

Compared to neural network approaches for semi-linear elliptic/parabolic PDEs such as [36, 13] it is important to highlight that our approach is limited to HJB equations arising in stochastic optimal control. Since our forward SDE is derived from optimality principles, extending it to other high-dimensional PDEs (for example, Black Scholes and Allen Cahan equations) is not obvious and may be impossible.

In future work, we will apply our approach to problems with non-constant and non-scalar diffusion coefficients. Although the theoretical framework supports this, we are not aware of any practical algorithms for this case. Since some SOC problems lead to non-smooth value functions and may include noise terms that depend on the control, testing our scheme on these problems is also a possible extension of our work. To overcome the challenge of efficiently computing Hessians of the value function needed for more general HJB penalties, one can use the hessQuik package [29].

REFERENCES

[1] F. Antonelli, *Backward-forward stochastic differential equations*, Ann. Appl. Probab., 3 (1993), pp. 777–793, http://links.jstor.org/sici?sici=1050-5164(199308)3:3⟨777:BSDE⟩2.0.CO;2-5&origin=MSN.

[2] R. Bellman, *Dynamic programming and stochastic control processes*, Information and Control, 1 (1958), pp. 228–239.

[3] C. Bender and R. Denk, *A forward scheme for backward SDEs*, Stochastic Process. Appl., 117 (2007), pp. 1793–1812, https://doi.org/10.1016/j.spa.2007.03.005, https://doi.org/10.1016/j.spa.2007.03.005.

[4] D. Bertsekas and J. Tsitsiklis, *Neuro-dynamic programming: an overview*, in Proceedings of 1995 34th IEEE Conference on Decision and Control, vol. 1, 1995, pp. 560–564 vol.1, https://doi.org/10.1109/CDC.1995.478953.

[5] F. Camilli and M. Falcone, *An approximation scheme for the optimal control of diffusion processes*, M2AN - Modélisation mathématique et analyse numérique, 29 (1995), pp. 97–122, http://www.numdam.org/item/M2AN_1995__29_1_97_0/.

[6] P. Cheridito, H. M. Soner, N. Touzi, and N. Victoir, *Second-order backward stochastic differential equations and fully nonlinear parabolic PDEs*, Comm. Pure Appl. Math., 60 (2007), pp. 1081–1110, https://doi.org/10.1002/cpa.20168, https://doi.org/10.1002/cpa.20168.

[7] H. Dong and N. V. Krylov, *The rate of convergence of finite-difference approximations for parabolic Bellman equations with Lipschitz coefficients in cylindrical domains*, Appl. Math. Optim., 56 (2007), pp. 37–66, https://doi.org/10.1007/s00245-007-0879-4, https://doi.org/10.1007/s00245-007-0879-4.

[8] W. E, J. Han, and A. Jentzen, *Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations*, Commun. Math. Stat., 5 (2017), pp. 349–380, https://doi.org/10.1007/s40304-017-0117-6, https://doi.org/10.1007/s40304-017-0117-6.

[9] I. Exarchos, E. Theodorou, and P. Tsiotras, *Stochastic differential games: a sampling approach via FBSDEs*, Dyn. Games Appl., 9 (2019), pp. 486–505, https://doi.org/10.1007/s13235-018-0268-4, https://doi.org/10.1007/s13235-018-0268-4.

[10] I. Exarchos and E. A. Theodorou, *Stochastic optimal control via forward and backward stochastic differential equations and importance sampling*, Automatica J. IFAC, 87 (2018), pp. 159–165, https://doi.org/10.1016/j.automatica.2017.09.004, https://doi.org/10.1016/j.automatica.2017.09.004.

[11] W. H. Fleming and H. M. Soner, *Controlled Markov Processes and Viscosity Solutions*, vol. 25 of Stochastic Modelling and Applied Probability, Springer, New York, second ed., 2006.

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016. http://www.deeplearningbook.org.

[13] J. Han, A. Jentzen, and W. E, *Solving high-dimensional partial differential equations using deep learning*, Proc. Natl. Acad. Sci. USA, 115 (2018), pp. 8505–8510, https://doi.org/10.1073/pnas.1718942115, https://doi.org/10.1073/pnas.1718942115.

[14] K. P. Hawkins, A. Pakniyat, E. Theodorou, and P. Tsiotras, *Forward-backward rapidly-exploring random trees for stochastic optimal control*, 2020, https://doi.org/10.48550/ARXIV.2006.12444, https://arxiv.org/abs/2006.12444.

[15] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016.

[16] C. Huré, H. Pham, and X. Warin, *Deep backward schemes for high-dimensional nonlinear PDEs*, Math. Comp., 89 (2020), pp. 1547–1579, https://doi.org/10.1090/mcom/3514, https://doi.org/10.1090/mcom/3514.

[17] E. R. Jakobsen, *On the rate of convergence of approximation schemes for Bellman equations associated with optimal stopping time problems*, Math. Models Methods Appl. Sci., 13 (2003), pp. 613–644, https://doi.org/10.1142/S0218202503002660, https://doi.org/10.1142/S0218202503002660.

[18] S. Ji, S. Peng, Y. Peng, and X. Zhang, *Three algorithms for solving high-dimensional fully coupled fbsdes through deep learning*, IEEE Intelligent Systems, 35 (2020), pp. 71–84, https://doi.org/10.1109/MIS.2020.2971597.

[19] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, arXiv preprint arXiv:1412.6980, (2014).

[20] P. E. Kloeden and E. Platen, *Numerical solution of stochastic differential equations*, vol. 23 of Applications of Mathematics (New York), Springer-Verlag, Berlin, 1992, https://doi.org/10.1007/978-3-662-12616-5, https://doi.org/10.1007/978-3-662-12616-5.

[21] N. V. Krylov, *The rate of convergence of finite-difference approximations for Bellman equations with Lipschitz coefficients*, Appl. Math. Optim., 52 (2005), pp. 365–399, https://doi.org/10.1007/s00245-005-0832-3, https://doi.org/10.1007/s00245-005-0832-3.

[22] K. Kunisch and D. Walter, *Semiglobal optimal feedback stabilization of autonomous systems via deep neural network approximation*, ESAIM Control Optim. Calc. Var., 27 (2021), pp. Paper No. 16, 59, https://doi.org/10.1051/cocv/2021009, https://doi.org/10.1051/cocv/2021009.

[23] H. J. Kushner, *Numerical methods for stochastic control problems in continuous time*, SIAM J. Control Optim., 28 (1990), pp. 999–1048, https://doi.org/10.1137/0328056, https://doi.org/10.1137/0328056.

[24] H. J. Kushner and P. Dupuis, *Numerical methods for stochastic control problems in continuous time*, vol. 24 of Applications of Mathematics (New York), Springer-Verlag, New York, second ed., 2001, https://doi.org/10.1007/978-1-4613-0007-6, https://doi.org/10.1007/978-1-4613-0007-6. Stochastic Modelling and Applied Probability.

[25] H. P. Langtangen and A. Logg, *Solving PDEs in Python*, Springer, 2017, https://doi.org/10.1007/978-3-319-52462-7.

[26] A. T. Lin, Y. T. Chow, and S. Osher, *A splitting method for overcoming the curse of dimensionality in hamilton-jacobi equations arising from nonlinear optimal control and differential games with applications to trajectory generation*, 2018, https://arxiv.org/abs/1803.01215.

[27] J. Ma, P. Protter, and J. M. Yong, *Solving forward-backward stochastic differential equations explicitly—a four step scheme*, Probab. Theory Related Fields, 98 (1994), pp. 339–359, https://doi.org/10.1007/BF01192258, https://doi.org/10.1007/BF01192258.

[28] J. Ma and J. Yong, *Forward-backward stochastic differential equations and their applications*, vol. 1702 of Lecture Notes in Mathematics, Springer-Verlag, Berlin, 1999.

[29] E. Newman and L. Ruthotto, *'hessquik': Fast hessian computation of composite functions*, Journal of Open Source Software, 7 (2022), p. 4171, https://doi.org/10.21105/joss.04171, https://doi.org/10.21105/joss.04171.

[30] D. Onken, L. Nurbekyan, X. Li, S. W. Fung, S. Osher, and L. Ruthotto, *A neural network approach for high-dimensional optimal control*, 2021, https://arxiv.org/abs/2104.03270.

[31] E. Pardoux and S. Tang, *Forward-backward stochastic differential equations and quasilinear parabolic PDEs*, Probab. Theory Related Fields, 114 (1999), pp. 123–150, https://doi.org/10.1007/s004409970001, https://doi.org/10.1007/s004409970001.

[32] M. Pereira, Z. Wang, T. Chen, E. Reed, and E. Theodorou, *Feynman-kac neural network architectures for stochastic control using second-order fbsde theory*, in Proceedings of the 2nd Conference on Learning for Dynamics and Control, A. M. Bayen, A. Jadbabaie, G. Pappas, P. A. Parrilo, B. Recht, C. Tomlin, and M. Zeilinger, eds., vol. 120 of Proceedings of Machine Learning Research, PMLR, 10–11 Jun 2020, pp. 728–738, https://proceedings.mlr.press/v120/pereira20a.html.

[33] H. Pham, *Continuous-time stochastic control and optimization with financial applications*, vol. 61 of Stochastic Modelling and Applied Probability, Springer-Verlag, Berlin, 2009, https://doi.org/10.1007/978-3-540-89500-8, https://doi.org/10.1007/978-3-540-89500-8.

[34] L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishchenko, *The Mathematical Theory of Optimal Processes*, Translated by K. N. Trirogoff; edited by L. W. Neustadt, Interscience Publishers John Wiley & Sons, Inc. New York-London, 1962.

[35] W. B. Powell, *Approximate dynamic programming*, Wiley Series in Probability and Statistics, Wiley-Interscience [John Wiley & Sons], Hoboken, NJ, 2007, https://doi.org/10.1002/9780470182963, https://doi.org/10.1002/9780470182963. Solving the curses of dimensionality.

[36] M. Raissi, *Forward-backward stochastic neural networks: Deep learning of high-dimensional*

partial differential equations, arXiv preprint arXiv:1804.07010, (2018).

[37] R. F. Stengel, *Optimal control and estimation*, Dover Publications, Inc., New York, 1994. Corrected reprint of the 1986 original.

[38] J. Wang and P. A. Forsyth, *Maximal use of central differencing for Hamilton-Jacobi-Bellman PDEs in finance*, SIAM J. Numer. Anal., 46 (2008), pp. 1580–1601, https://doi.org/10.1137/060675186, https://doi.org/10.1137/060675186.

[39] S. Yensiri and R. J. Skulkhu, *An investigation of radial basis function-finite difference (rbf-fd) method for numerical solution of elliptic partial differential equations*, Mathematics, 5 (2017), https://doi.org/10.3390/math5040054, https://www.mdpi.com/2227-7390/5/4/54.

[40] J. Yong and X. Y. Zhou, *Stochastic controls*, vol. 43 of Applications of Mathematics (New York), Springer-Verlag, New York, 1999, https://doi.org/10.1007/978-1-4612-1466-3, https://doi.org/10.1007/978-1-4612-1466-3. Hamiltonian systems and HJB equations.