ACTOR-CRITIC METHOD FOR HIGH DIMENSIONAL STATIC HAMILTON–JACOBI–BELLMAN PARTIAL DIFFERENTIAL EQUATIONS BASED ON NEURAL NETWORKS*

MO ZHOU†, JIEQUN HAN‡, AND JIANFENG LU§

Abstract. We propose a novel numerical method for high dimensional Hamilton–Jacobi–Bellman (HJB) type elliptic partial differential equations (PDEs). The HJB PDEs, reformulated as optimal control problems, are tackled by the actor-critic framework inspired by reinforcement learning, based on neural network parametrization of the value and control functions. Within the actor-critic framework, we employ a policy gradient approach to improve the control, while for the value function, we derive a variance reduced least-squares temporal difference method using stochastic calculus. To numerically discretize the stochastic control problem, we employ an adaptive step size scheme to improve the accuracy near the domain boundary. Numerical examples up to 20 spatial dimensions including the linear quadratic regulators, the stochastic Van der Pol oscillators, the diffusive Eikonal equations, and fully nonlinear elliptic PDEs derived from a regulator problem are presented to validate the effectiveness of our proposed method.

Key words. Hamilton–Jacobi–Bellman equations, high dimensional partial differential equations, stochastic control, actor-critic methods

AMS subject classifications. 65N75, 65C30, 49L12, 93E20

DOI. 10.1137/21M1402303

1. Introduction. The Hamilton–Jacobi–Bellman (HJB) equation is an important family of partial differential equations (PDEs), given its connection with optimal control problems that lead to a wide range of applications. The unknown in the HJB equation can be viewed as the total expected value function for optimal control problems. The equation can be derived from the dynamic programming principle pioneered by Bellman [8], which gives a necessary and sufficient condition of the optimality. Theoretical results for the existence and uniqueness of the HJB equations are well established; see, e.g., [68]. From the viewpoint of stochastic control, the relationship between the viscosity solution of the HJB equations and the backward stochastic differential equations (BSDEs) is introduced in [14, 52, 53, 54, 55].

The wide applications of HJB equations call for efficient numerical algorithms. Various numerical approaches have been developed in the literature, including the monotone approximation scheme [3, 21], the finite volume method [60, 65], and the Galerkin method [4, 5]. In [50], nonoscillatory schemes are developed to solve the HJB equations exploring the connection with hyperbolic conservation laws. The HJB equations related to reachability problems are studied in [42, 45, 46]. A general survey for classical methods to solve the optimal control problem numerically can be found, e.g., in [59]. While these conventional approaches have been quite successful, they fall

†Department of Mathematics, Duke University, Durham, NC 27708 USA (mo.zhou366@duke.edu).

‡Department of Mathematics, Princeton University, Princeton, NJ 08544 USA (frankhan91@gmail.com).

§Department of Mathematics, Department of Physics, and Department of Chemistry, Duke University, Durham, NC 27708 USA (jianfeng@math.duke.edu).

short for solving HJB equations in high dimensions due to the curse of dimensionality [8]: the computational cost goes up exponentially with the dimensionality. Many works attempt to mitigate this fundamental difficulty by leveraging dimension reduction techniques such as proper orthogonal decomposition, sparse grid, pseudospectral collocation, and tensor decomposition (see, e.g., [16, 35, 36, 41, 51]). The performance of these algorithms heavily depends on how well the low dimensional representation matches the solutions, and is typically problem dependent and thus with limited applicability.

To better address the challenge of high dimensionality, a promising direction is to consider the artificial neural network as a more flexible and efficient function approximation tool. This topic has received a considerable amount of attention and been a rapidly developing field in recent years. Several numerical approaches for high dimensional PDEs based on neural network parametrization have been proposed; see, e.g., the reviews [6, 20] and references therein.

For HJB type equations and related optimal control problems, the most tightly connected approach to our work is the deep BSDE method [19, 27], which reformulates parabolic PDEs as control problems using BSDEs, and uses deep neural network parametrization for the solution and control to solve this problem. Theoretical results for convergence of this method are studied in [28]. The deep BSDE method and its variants have been applied to solve HJB type equations, stochastic control problems, and differential games (see, e.g., [13, 19, 24, 26, 27, 32, 34, 40, 49, 56, 58]). Numerical algorithms for solving high dimensional deterministic and stochastic control problems based on other forms combined with deep learning approximation have also been investigated in [7, 18, 25, 48].

While some methods mentioned above have been successful in solving PDEs in high dimensions, there have been two issues that remain to be addressed. On the one hand, most of these works concern parabolic PDEs of finite time horizon (often of order one), while only a few works investigate the static elliptic HJB equations corresponding to control problems with infinite time horizon. On the other hand, most existing works consider equations where the optimal controls are explicitly known given the value function or without controls, while there are many important HJB type equations for which the optimal control is cast through an optimization problem and hence implicit. Recently, an algorithm for a high dimensional finite-time horizon stochastic control problem with implicit optimal control is considered in [33], based on the deep BSDE formulation associated with the stochastic maximum principle. In this paper, we take a different approach and focus on solving the static elliptic type HJB equation with implicit control, in which the above two challenges are compounded.

Our proposed numerical method is heavily inspired by the literature on reinforcement learning (RL) [63], which is of course closely related to control problems. Our motivation for borrowing techniques from RL is due to the impressive revolution and great success in recent years in deep RL by utilizing neural network parametrization [17, 47, 61]. In the RL context, the control problem is usually formulated as a Markov decision process (MDP) on discrete time and state space. If the model is given, finding the optimal policy can be viewed as solving a discrete HJB equation. It is then natural to ask whether algorithms developed in the RL context can be generalized to the context of solving high dimensional HJB equations.

In this paper, we reformulate the HJB type fully nonlinear elliptic PDEs into stochastic control problems and leverage the actor-critic framework in conjunction with a neural network approximation to solve the equations. The actor-critic methods are a class of algorithms in RL [63]. These algorithms iteratively evaluate and improve

the current policies (i.e., controls) until final convergence. The critic refers to the value function of a given policy. The process of estimating the critic is called policy evaluation. The most common algorithms for policy evaluation are temporal difference (TD) methods [10, 39, 64], or their variants, such as the TD(λ) [15] and the least-squares TD (LSTD) [11, 43, 57]. The actor refers to the policy function, and we need to make policy improvement based on a given value function. In this case, the most popular method is policy gradient [2, 9, 15, 39, 66] and their variants, such as natural policy gradient [10, 57]. In this work, we propose a variance reduced version of the LSTD method for policy evaluation derived using stochastic calculus. We also adapt the policy gradient method for policy improvement to the continuous-time stochastic control problem.

The rest of this paper is organized as follows. In section 2, we provide a theoretical background for the optimal control problems and formulate the actor-critic framework for continuous-time stochastic control. In section 3, we introduce the numerical algorithm to solve the optimal control problem. Numerical examples are presented in section 4. We conclude in section 5 with an outlook for future works.

2. Theoretical background of the actor-critic framework.

2.1. Control formulation of elliptic equations.
Consider the following fully nonlinear elliptic PDE

$$(2.1) \quad \inf_{u \in U} \left[\frac{1}{2} \operatorname{Tr} \left(\sigma \sigma^\top \operatorname{Hess}(V) \right)(x, u) + b(x, u)^\top \nabla V(x) + f(x, u) \right] - \gamma V(x) = 0 \quad \text{in } \Omega$$

with boundary condition $V(x) = g(x)$ on $\partial \Omega$. Here the state space Ω is an open, connected set in $\mathbb{R}^d$ with piecewise smooth boundary, and the control space U is a convex closed domain in $\mathbb{R}^{d_u}$. We assume that $V(x) \in C^2(\overline{\Omega})$, $f(x, u) \in C(\overline{\Omega} \times U)$, $b(x, u) \in C(\overline{\Omega} \times U; \mathbb{R}^d)$, $\sigma(x, u) \in C(\overline{\Omega} \times U; \mathbb{R}^{d \times d_w})$ with $\sigma(x, u)\sigma^\top(x, u)$ being uniformly elliptic and bounded, and $\gamma \geq 0$ is a constant. Here and in the following, we use ∇ and Hess to denote the gradient and Hessian operators.

As a starting point of our approach, we reformulate the above elliptic equation as an optimal control problem. Let $(\widetilde{\Omega}, \mathcal{F}, \{\mathcal{F}_t\}_{t \geq 0}, \mathbf{P})$ be a filtered probability space. Consider the following stochastic differential equation (SDE)

$$(2.2) \quad \mathrm{d}X_t = b(X_t, u_t)\, \mathrm{d}t + \sigma(X_t, u_t)\, \mathrm{d}W_t$$

with initial condition $X_0 = x \in \Omega$, where $u_t \in U \subset \mathbb{R}^{d_u}$ is an $\mathcal{F}_t$-adapted control field and W_t is a d_w-dimensional $\mathcal{F}_t$-standard Brownian motion. As we solve the equation in the domain Ω, we define a stopping time

$$(2.3) \quad \tau = \inf\{t : X_t \notin \Omega\}.$$

It is a standard result that $\tau < \infty$ a.s.; see, for example, [38].

We then consider an optimal control problem to minimize the following cost functional

$$(2.4) \quad J^u(x) = \mathbb{E}\Big[\int_0^\tau f(X_s, u_s)e^{-\gamma s}\, \mathrm{d}s + e^{-\gamma \tau} g(X_\tau) \mid X_0^u = x \Big].$$

In this cost functional, f can be interpreted as running cost, g is the terminal cost when the SDE hits the boundary $\partial \Omega$, and γ is the discount rate.

The control u is chosen over the set of stochastic processes that have values in U and are adapted to the filtration $\mathcal{F}_t$. Define

$$(2.5) \qquad V(x) = \inf_u J^u(x)$$

as the optimal value function (i.e., optimal cost-to-go function). According to standard results in stochastic control theory [68], V satisfies the time-independent HJB equation

$$(2.6) \qquad \inf_u \big\{ \mathcal{L}^u V(x,u) + f(x,u) - \gamma V(x) \big\} = 0$$

in Ω with boundary condition $V(x) = g(x)$ on $\partial\Omega$, where

$$\mathcal{L}^u V(x) = \frac{1}{2}\operatorname{Tr}\big(\sigma\sigma^\top \operatorname{Hess}(V)\big)(x,u) + b(x,u)^\top \nabla V(x)$$

is the generator of the controlled SDE (2.2). Note that the HJB equation (2.6) coincides with the original PDE (2.1) and, hence, we can solve the PDE (2.1) by solving the optimal control problem to obtain the optimal value function.

2.2. Actor-critic method in stochastic optimal control problem. Our approach for solving the optimal control problem is based on the actor-critic framework. In such methods, one solves for both the value function and control field. The control (i.e., policy in the RL terminology) is known as the *actor*, while the value function corresponding to the control is known as the *critic* since it is used to evaluate the optimality of the control. Accordingly, the actor-critic algorithms consist of two parts: policy evaluation for the critic and policy improvement for the actor. While many approaches have been developed under the actor-critic framework [2, 10, 15, 39, 57, 64, 66], we will focus on simple and perhaps the most popular algorithms: TD learning for the value function given a policy and policy gradient for improving the control.

2.2.1. TD for discrete Markov decision processes. To better convey the idea, let us first briefly recall the algorithms for the discrete-time MDP with finite state and action space; more details can be found in, e.g., [63]. The MDP starts with some initial state S_0 in the state space $\mathcal{S}$, possibly sampled according to a distribution. At time $t \in \mathbb{N}$, given the current state S_t, the agent picks an action A_t in the action set $\mathcal{A}$ according to a policy. We assume that the policy is deterministic, i.e., the policy is a map π from the state space $\mathcal{S}$ to the action space $\mathcal{A}$:

$$(2.7) \qquad A_t = \pi(S_t).$$

After the action A_t is chosen, the system state will transit to S_{t+1}, according to a probability transition function

$$(2.8) \qquad \mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a) = p(s' \mid s, a).$$

The action also incurs a cost R_{t+1}, which we assume to be given by a deterministic function of the previous state S_t, action A_t, and the current state S_{t+1}:

$$(2.9) \qquad R_{t+1} = f(S_t, A_t, S_{t+1}).$$

The goal of the MDP problem is to choose the best policy to minimize the expected total discounted cost

$$(2.10) \qquad \mathbb{E}_{S_0 \sim \mu, \pi}\Big[\sum_{t=1}^{\infty} \beta^{t-1} R_t\Big],$$

where $\beta \in (0, 1)$ is a discount factor, μ is the distribution of the initial state S_0, and we have used $\mathbb{E}_\pi$ to indicate the dependence on the transition dynamics on the choice of the policy π.

To solve the MDP problem, it is convenient to introduce the (state) value function w.r.t. a policy π as the expected cost starting at states s under that policy:

$$(2.11) \qquad V^\pi(s) = \mathbb{E}_\pi \Big[\sum_{t=1}^\infty \beta^{t-1} R_t \mid S_0 = s \Big].$$

By the dynamic programming principle [8], for any given policy π, the value function satisfies

$$(2.12) \qquad V^\pi(s) = \mathbb{E}_\pi \Big[\sum_{t=1}^n \beta^{t-1} R_t + \beta^n V^\pi(S_n) \mid S_0 = s \Big]$$

for any $n \geq 1$. In order to minimize the total cost (2.10), we search for an optimal policy π^* that satisfies for all π,

$$(2.13) \qquad V^{\pi^*}(s) \leq V^\pi(s) \qquad \forall\, s \in \mathcal{S}.$$

Specifically, by the optimality principle, we have
$$(2.14)$$
$$V^{\pi^*}(s) = \min_{\{a_t\} \subset \mathcal{A}} \mathbb{E} \Big[\sum_{t=1}^n \beta^{t-1} R_t + \beta^n V^{\pi^*}(S_n) \mid S_0 = s, A_t = a_t, t = 0, 1, \ldots, n-1 \Big].$$

Note that while in (2.12) and (2.14) the right-hand side starts at time 0, we can start at any time and run the process for n steps due to stationarity.

Let us make a couple of remarks for the setup of the discrete MDP used here. First, in RL, reward is usually used instead of cost and, hence, one maximizes the total reward instead of minimizing the cost; evidently, the two viewpoints are equivalent up to a change of sign. We use cost, which is more in line with the control literature and also our problem in the continuous setting. Second, the cost R_t is not necessarily a deterministic function as in (2.9), but may follow some probability distribution together with the next state:

$$(2.15) \qquad \mathbb{P}(S_{t+1} = s', R_{t+1} = r \mid S_t = s, A_t = a) = p(s', r \mid s, a).$$

Moreover, the policy can also be probabilistic rather than deterministic as assumed in (2.7). We choose the simplified setting for the cost and policy to make it consistent with our continuous optimal control setting. Finally, we use an MDP without stopping time and thus without terminal cost for simplicity. The adaptation to our PDE setup will be discussed below in sections 2.2.3 and 2.2.4.

TD learning is a class of algorithms that evaluate a given control (i.e., policy) by updating the value function, combining a Monte Carlo estimate of the running cost over a time period and the dynamic programming principle for the future cost-to-go. For a policy π to be evaluated, with a given trajectory $\{S_t, t \geq 0\}$, we update the value function at each t by

$$(2.16) \qquad \widehat{V}^\pi(S_t) \longleftarrow \widehat{V}^\pi(S_t) + \alpha \left(\sum_{k=1}^n \beta^{k-1} R_{t+k} + \beta^n \widehat{V}^\pi(S_{t+n}) - \widehat{V}^\pi(S_t) \right),$$

where α is the learning rate and $\widehat{V}^{\pi}$ on the right-hand side is the current estimate of the value function. In (2.16), we only update the value function at the state S_t and the value at other states remain unchanged. In practice, this update of the value function is usually done for multiple trajectories.

In the above updating rule, we have used the n-step TD $\mathrm{TD}_n^{\pi}(S_t)$, defined as

$$(2.17) \qquad \mathrm{TD}_n^{\pi}(S_t) = \sum_{k=1}^{n} \beta^{k-1} R_{t+k} + \beta^n \widehat{V}^{\pi}(S_{t+n}) - \widehat{V}^{\pi}(S_t),$$

which depends on the trajectory of length $n+1$ (t to $t+n$) from the starting state S_t. TD_n^{π} can be understood as an indicator of the inconsistency between the current estimate of the value function with a sampled value using n-steps of the MDP, since according to (2.12), $\mathbb{E}_{\pi}\mathrm{TD}_n^{\pi}$ vanishes if $\widehat{V}^{\pi}$ agrees with the true value function V^{π}. Hence, TD learning can be viewed as a stochastic fixed point iteration for the value function.

When function approximation is used for the value function, in particular nonlinear approximations such as neural networks, an alternative approach, the LSTD is often used to overcome potential divergence problems of TD learning [11, 57]. Instead of the stochastic fixed point updating formula as (2.16), in the LSTD method, the parameters are optimized to minimize the squares of the TD error as a loss function. More specifically, if the value function is parametrized as $V^{\pi}(\cdot; \theta_V)$, we solve for θ_V by

$$(2.18) \qquad \min_{\theta_V} \mathbb{E}_{S_0 \sim \mu, \pi}\left[\left(\sum_{t=1}^{n} \beta^{t-1} R_t + \beta^n V^{\pi}(S_n; \theta_V) - V^{\pi}(S_0; \theta_V)\right)^2 \mid S_0\right],$$

where μ is some initial distribution for the state S_0. In practice, (2.18) is often solved using the stochastic gradient descent method. Such a method has been proved successful in, e.g., [17, 47].

2.2.2. Policy gradient for discrete MDPs. Policy gradient is a class of methods to learn parametrized policies through gradient based algorithms. Assume we consider a class of (deterministic) policy parametrized as

$$(2.19) \qquad\qquad A_t(S_t) = \pi(S_t; \theta_{\pi}),$$

where θ_{π} denotes a collection of parameters and $\pi(\cdot; \theta)$ is a chosen nonlinear parametrization.

To find an optimal policy, we aim to minimize the objective function (cf. (2.14))

$$(2.20) \qquad\qquad J(\theta_{\pi}) = \mathbb{E}_{S_0 \sim \mu, \pi(\cdot; \theta_{\pi})}\left[\sum_{k=1}^{n} \beta^{k-1} R_k + \beta^n \widehat{V}^{\pi}(S_n)\right]$$

w.r.t. the collective parameter θ_{π}. Note that (2.20) explicitly takes into account the cost of the first n steps, while using an (approximate) value function $\widehat{V}^{\pi}$ for

the future cost after n steps, coming from, e.g., the TD learning algorithm. The objective function (2.20) can be optimized using a stochastic gradient method. Using a stochastic estimate of the gradient $\widehat{\nabla J} \approx \nabla_{\theta_\pi} J$, so that the parameter is updated as

$$(2.21) \qquad \theta_\pi \leftarrow \theta_\pi - \alpha \widehat{\nabla J}$$

with suitable learning rate α. Note that in principle, one needs to differentiate all terms involved in (2.20) w.r.t. θ_π; however, in practice, in the actor-critic framework, one typically leaves out the derivative of $\widehat{V}^\pi$ w.r.t. π, as it is impractical to compute since $\widehat{V}^\pi$ is obtained using, e.g., TD learning. Nevertheless, we would still need to differentiate $\widehat{V}^\pi(S_n)$ w.r.t. S_n, as the state S_n is affected by the choice of the policy, thus

$$\frac{\partial \widehat{V}^\pi(S_n)}{\partial \theta_\pi} \doteq \frac{\partial \widehat{V}^\pi}{\partial S_n} \frac{\partial S_n}{\partial \theta_\pi},$$

where $\doteq$ indicates that the (functional) derivative $\frac{\delta \widehat{V}^\pi}{\delta \pi}$ is omitted. Dropping this term is often applied in actor-critic algorithms. Some justifications can be found in [15]: Under certain conditions, the approximated gradient is still in the direction of improving the performance and the set of critical points of the objective function coincides with the set of zero approximated gradients.

Since we consider deterministic policies, the policy gradient approach discussed above is in the same spirit as the *deterministic policy gradient algorithm* proposed in [62]. One difference is that we use the state value function $V(s)$ while [62] uses the state-action value function (Q-function) $Q(s,a)$. Another difference is that the objective function used in [62] is based on the stationary distribution of a state-action pair given the policy, while we roll out a trajectory for the cost function (combined with using an estimated value function for future cost), which is more suitable to an actor-critic framework. Our approach is also easier to generalize to the continuous setting, which will be discussed in section 2.2.4.

2.2.3. TD for continuous optimal control problems. We now introduce how to adapt the above algorithmic ideas to the continuous setting.

Given a control function $u(x) \in C(\overline{\Omega})$ (which corresponds to π in the discrete setting), the corresponding value function is given by

$$(2.22) \qquad V^u(x) = \mathbb{E}_u \left[\int_0^\tau f(X_s, u(X_s)) e^{-\gamma s} \, \mathrm{d}s + e^{-\gamma \tau} g(X_\tau) \mid X_0 = x \right].$$

Here $\mathbb{E}_u$ indicates expectation w.r.t. the trajectory (with a fixed policy u). This is just the cost functional in (2.4) with a specific control policy. In the continuous setting, the dynamical programming principle indicates that the value function V^u satisfies the PDE (see, e.g., [68])
(2.23)
$$\frac{1}{2} \operatorname{Tr} \left(\sigma \sigma^\top \operatorname{Hess}(V^u) \right)(x, u(x)) + b(x, u(x))^\top \nabla V^u(x) + f(x, u(x)) - \gamma V^u(x) = 0 \text{ in } \Omega$$

with boundary condition $V^u(x) = g(x)$ on $\partial\Omega$.

To better convey the idea, we first consider a fixed time interval $[0, T]$ with $T > 0$ and neglect the stopping time and also the domain boundary. Necessary modifications

MO ZHOU, JIEQUN HAN, AND JIANFENG LU

regarding the stopping time and boundary will be explained below. Applying Itô's formula to $e^{-\gamma t}V^u(X_t)$, we get

$$(2.24) \quad e^{-\gamma T}V^u(X_T) = V^u(X_0) + \int_0^T e^{-\gamma s}\Big[\frac{1}{2}\operatorname{Tr}\big(\sigma\sigma^\top \operatorname{Hess}(V^u)\big)(X_s, u(X_s))$$
$$+ b(X_s, u(X_s))^\top \nabla V^u(X_s) - \gamma V^u(X_s)\Big]\,\mathrm{d}s$$
$$+ \int_0^T e^{-\gamma s}\nabla V^u(X_s)^\top \sigma(X_s, u(X_s))\,\mathrm{d}W_s.$$

Combined with the PDE (2.23), (2.24) gives

$$(2.25) \quad V^u(X_0) = \int_0^T e^{-\gamma s}f(X_s, u(X_s))\,\mathrm{d}s$$
$$- \int_0^T e^{-\gamma s}\nabla V^u(X_s)^\top \sigma(X_s, u(X_s))\,\mathrm{d}W_s + e^{-\gamma T}V^u(X_T).$$

The term $\int_0^T e^{-\gamma s}\nabla V^u(X_s)^\top \sigma(X_s, u(X_s))\,\mathrm{d}W_s$ is a martingale w.r.t. T because ∇V and σ are bounded according to our assumptions [38]. Therefore, taking the expectation, we arrive at

$$(2.26) \qquad V^u(X_0) = \mathbb{E}_u\left[\int_0^T e^{-\gamma s}f(X_s, u(X_s))\,\mathrm{d}s + e^{-\gamma T}V^u(X_T) \mid X_0\right].$$

We observe that this is the analog of (2.12) in the continuous setting, where the unit time discount $e^{-\gamma}$ is the analog of the discount factor β in the discrete setting. Compared with the discrete time setting, besides (2.26), we have in addition (2.25) before taking expectation, thanks to Itô's lemma. Exploiting the two identities, analogously to the discrete case, we define two versions of TD in the continuous setting as

$$(2.27) \qquad \mathrm{TD}_1^u = \int_0^T e^{-\gamma s}f(X_s, u(X_s))\,\mathrm{d}s - \int_0^T e^{-\gamma s}\nabla V(X_s)^\top \sigma(X_s, u(X_s))\,\mathrm{d}W_s$$
$$+ e^{-\gamma T}V(X_T) - V(X_0),$$
$$(2.28) \qquad \mathrm{TD}_2^u = \int_0^T e^{-\gamma s}f(X_s, u(X_s))\,\mathrm{d}s + e^{-\gamma T}V(X_T) - V(X_0).$$

Note that both TD_1 and TD_2 depend on the trajectory X_t; in particular, they should be viewed as random variables, while we suppress such dependence in the notation. From (2.25) and (2.26), if V is the exact value function corresponding to the control u, we have

$$(2.29) \qquad\qquad\qquad \mathrm{TD}_1^u = 0, \quad \mathbb{P}\text{-a.s.}$$
$$(2.30) \qquad\qquad\qquad \mathbb{E}_u\,\mathrm{TD}_2^u = 0.$$

Note in particular that TD_1^u vanishes without taking the expectation for the exact value function while $\operatorname{Var}(\mathrm{TD}_2^u) = \mathbb{E}_u[\int_0^T e^{-2\gamma s}|\nabla V(X_s)^\top \sigma(X_s, u(X_s))|^2\,\mathrm{d}s] > 0$ if $\nabla V^\top(x)\sigma(x, u) \not\equiv 0$. Moreover, as the difference between TD_1 and TD_2 is given by a martingale term, for any approximate value function, we have

$$\mathbb{E}_u \mathrm{TD}_1^u = \mathbb{E}_u \mathrm{TD}_2^u.$$

Now let us introduce two loss functionals for the critic in the spirit of LSTD:

(2.31)

$$
\begin{aligned}
L_1(V) &= \mathbb{E}_{X_0 \sim \mu, u} \left(\mathrm{TD}_1^u \right)^2 \\
&= \mathbb{E}_{X_0 \sim \mu, u} \left[\left(\int_0^{T \wedge \tau} e^{-\gamma s} f(X_s, u(X_s)) \, \mathrm{d}s \right. \right. \\
&\qquad \left. \left. - \int_0^{T \wedge \tau} e^{-\gamma s} \nabla V(X_s)^\top \sigma(X_s, u(X_s)) \, \mathrm{d}W_s + e^{-\gamma(T \wedge \tau)} V(X_{T \wedge \tau}) - V(X_0) \right)^2 \right],
\end{aligned}
$$

(2.32)

$$
\begin{aligned}
L_2(V) &= \mathbb{E}_{X_0 \sim \mu, u} \left(\mathrm{TD}_2^u \right)^2 \\
&= \mathbb{E}_{X_0 \sim \mu, u} \left[\left(\int_0^{T \wedge \tau} e^{-\gamma s} f(X_s, u(X_s)) \, \mathrm{d}s + e^{-\gamma(T \wedge \tau)} V(X_{T \wedge \tau}) - V(X_0) \right)^2 \right],
\end{aligned}
$$

where μ is some initial distribution for X_0 and we have also taken into account the stopping time τ when the process hits the domain boundary. Here the two losses are viewed as functionals of the value function V, the finite-dimensional function approximation will be discussed in the next section.

The stochastic gradient method is used to minimize the loss function in LSTD to find the best approximation of the value function. Written in terms of functional variations, this amounts to approximating

(2.33)
$$
\mathbb{E}_{X_0 \sim \mu, u} \frac{\delta \left(\mathrm{TD}_1^u \right)^2}{\delta V} \approx \frac{\delta \left(\mathrm{TD}_1^u \right)^2}{\delta V} (X_t) = 2 \mathrm{TD}_1^u (X_t) \frac{\delta \mathrm{TD}_1^u}{\delta V} (X_t),
$$

(2.34)
$$
\mathbb{E}_{X_0 \sim \mu, u} \frac{\delta \left(\mathrm{TD}_2^u \right)^2}{\delta V} \approx \frac{\delta \left(\mathrm{TD}_2^u \right)^2}{\delta V} (X_t) = 2 \mathrm{TD}_2^u (X_t) \frac{\delta \mathrm{TD}_2^u}{\delta V} (X_t),
$$

where we evaluate the right-hand side term on a single realization of the trajectory to ease the notation. In our numerical implementation, we use multiple trajectories to further improve the computation efficiency. As we remark above, since (2.25) holds true without taking the expectation, the right-hand side of (2.33) thus vanishes for the exact value function for any realization of X_t, in particular, the variance of the stochastic gradient is 0. In comparison, while the stochastic estimate of (2.34) has the expectation 0 for the exact value function, for each trajectory, the right-hand side is not 0. This means that the stochastic gradient estimate (2.34) has a larger variance than the estimate (2.33). Let us remark that the vanishing variance property of (2.33) is similar to quantum Monte Carlo [22], for which the variance of the local energy estimate also vanishes at the ground state.

In the following, to distinguish the two loss functions, we call the method based on L_1 (2.31) the variance reduced LSTD (VR-LSTD), while that corresponding to L_2 (2.32) is named the LSTD. We will demonstrate in our numerical experiments that VR-LSTD gives better results than LSTD.

2.2.4. Policy gradient for continuous optimal control problems. For the actor part, we use policy gradient to improve the policy. According to the dynamical programming principle [68], for the optimal value function V, we have

(2.35)
$$
V(X_0) = \inf_u \mathbb{E}_u \left[\int_0^T f(X_s, u(X_s)) e^{-\gamma s} \, \mathrm{d}s + e^{-\gamma T} V(X_T) \mid X_0 \right],
$$

where u is minimized over the set of admissible controls. In other words, the control u should minimize the functional on the right-hand side. Therefore, we can use the following loss function for the actor, for which we also incorporate the stopping time:

$$(2.36) \qquad J(u) = \mathbb{E}_{X_0 \sim \mu, u} \left[\int_0^{T \wedge \tau} f(X_s, u(X_s)) e^{-\gamma s} \, \mathrm{d}s + \widehat{V}(X_{T \wedge \tau}) e^{-\gamma(T \wedge \tau)} \right],$$

where $\widehat{V}$ is the current estimate of the value function (via TD learning in the critic part). Observe that this loss function is a continuous analog of (2.20).

In the numerical algorithm, the control, as a high dimensional function, will be parametrized as a neural network $u(\cdot; \theta_u)$, where θ_u denotes collectively the parameters. The parameters are optimized using a stochastic approximation to gradients of $J(u)$. Similarly to our discussion of policy gradient for the discrete case in section 2.2.2, when differentiating the loss function (2.36) w.r.t. the parameters of the control θ_u, several terms would contribute to the derivative, including the control $u(\cdot)$ itself, the SDE trajectory $X_.$, the stopping time τ, and also the estimated value function $\widehat{V}(\cdot)$. Similarly to the discrete case, we will drop the functional derivative of $\widehat{V}$ w.r.t. u, i.e., the derivative $\frac{\delta \widehat{V}}{\delta u} \frac{\partial u}{\partial \theta_u}$, since the dependence of $\widehat{V}$ on u is through the algorithm for the critic, e.g., the TD learning, which is impractical to track. Furthermore, if $\hat{V}$ is the optimal value function, treating it as a fixed function and optimizing u in (2.36) gives the optimal policy function. Therefore, we approximate the functional derivative as

$$(2.37) \quad \frac{\delta J}{\delta u} \doteq \mathbb{E}_{X_0 \sim \mu, u} \left[\int_0^{T \wedge \tau} \frac{\delta f(X_s, u(X_s))}{\delta u} e^{-\gamma s} \, \mathrm{d}s + \mathbb{1}_{\{\tau < T\}} f(X_\tau, u(X_\tau)) e^{-\gamma \tau} \frac{\delta \tau}{\delta u} \right.$$
$$\left. + \nabla \widehat{V}(X_{T \wedge \tau}) e^{-\gamma(T \wedge \tau)} \frac{\delta X_s}{\delta u} \bigg|_{s = T \wedge \tau} + \mathbb{1}_{\{\tau < T\}} (\mathcal{L}^u - \gamma) \widehat{V}(X_\tau) e^{-\gamma \tau} \frac{\delta \tau}{\delta u} \right],$$

where $\doteq$ indicates that we leave out the contribution from the functional derivative of $\widehat{V}$ w.r.t. u and we have

$$(2.38) \qquad \frac{\delta f(X_s, u(X_s))}{\delta u} = \frac{\partial f}{\partial x} \frac{\delta X_s}{\delta u} + \frac{\partial f}{\partial u} \left(\mathrm{Id} + \nabla u(X_s) \frac{\delta X_s}{\delta u} \right).$$

To obtain the formula, we have used Itô's lemma to rewrite

$$(2.39) \quad \mathbb{E}_{X_0 \sim \mu, u}[\widehat{V}(X_{T \wedge \tau}) e^{-\gamma(T \wedge \tau)}] = \mathbb{E}_{X_0 \sim \mu, u} \left[\widehat{V}(X_0) + \int_0^{T \wedge \tau} (\mathcal{L}^u - \gamma) \widehat{V}(X_s) \, \mathrm{d}s \right],$$

and taken the derivative of the right-hand side w.r.t. u.

3. Numerical algorithm. In this section, we present our numerical algorithm for solving high dimensional HJB type elliptic PDEs based on the actor-critic framework discussed in the previous section.

3.1. Function approximation. In order to numerically deal with the high dimensional functions V and u, we use two neural networks to parametrize the value function $V(\cdot; \theta_V)$ and the control $u(\cdot; \theta_u)$, the parameters of which are denoted collectively by θ_V and θ_u, respectively. We apply the structure of the residual neural network [31] in pursuit of better optimization performance. A neural network $\phi(x; \theta)$ with l hidden layers is represented by

$$(3.1) \qquad \phi(x; \theta) = F_l \circ \sigma_l \circ F_{l-1} \circ \sigma_{l-1} \circ \cdots \circ F_1 \circ \sigma_1 \circ F_0(x),$$

where F_i are linear transforms with dimensions depending on the width of hidden layers and the dimensions of inputs and outputs, and σ_i are elementwise activate functions with skip connection: $\sigma_i(x) = x + \text{ReLU}(x)$.

Moreover, note that the VR-LSTD loss function L_1 (2.31) requires the gradient of the value function. Since we are using a neural network parametrization $V = V(\cdot\,;\theta_V)$, a direct approach is to use autodifferentiation of $V(x;\theta_V)$ w.r.t. x to calculate the gradient. We find that a better approach in practice is to use another neural network to represent ∇V, which is consistent with the observations in [27, 29]. Thus, for VR-LSTD, the gradient of the value function is represented by a separate neural network $G(\cdot\,;\theta_G)$ with collective parameters θ_G.

To summarize, w.r.t. the collective parameters, the loss functions for the critic corresponding to (2.31) and (2.32) are

$$
(3.2) \quad L_1(\theta_V, \theta_G) = \mathbb{E}_{X_0 \sim \mu, u} \left[\left(\int_0^{T \wedge \tau} e^{-\gamma s} f(X_s, u(X_s)) \, \mathrm{d}s \right. \right.
$$
$$
- \int_0^{T \wedge \tau} e^{-\gamma s} G(X_s; \theta_G)^\top \sigma(X_s, u(X_s)) \, \mathrm{d}W_s
$$
$$
\left. \left. + e^{-\gamma(T \wedge \tau)} V(X_{T \wedge \tau}; \theta_V) - V(X_0; \theta_V) \right)^2 \right],
$$

$$
(3.3) \quad L_2(\theta_V) = \mathbb{E}_{X_0 \sim \mu, u} \left[\left(\int_0^{T \wedge \tau} e^{-\gamma s} f(X_s, u(X_s)) \, \mathrm{d}s + e^{-\gamma(T \wedge \tau)} V(X_{T \wedge \tau}; \theta_V) \right. \right.
$$
$$
\left. \left. - V(X_0; \theta_V) \right)^2 \right].
$$

We remark that there is no need to add penalty terms in L_1 to ensure the consistency between $V(x, \theta_V)$ and $G(x; \theta_G)$, because if we replace $V^u(\cdot)$ by $V(\cdot, \theta_V)$ in (2.24) and plug it in (3.2), we have

(3.4)

$$
L_1(\theta_V, \theta_G) = \mathbb{E}_{X_0 \sim \mu, u} \left[\left(\int_0^{T \wedge \tau} e^{-\gamma s} [(\mathcal{L}^u V - \gamma V)(X_s; \theta_V) + f(X_s, u(X_s))] \, \mathrm{d}s \right. \right.
$$
$$
\left. \left. - \int_0^{T \wedge \tau} e^{-\gamma s} \left(\nabla_x V(X_s; \theta_V) - G(X_s; \theta_G) \right)^\top \sigma(X_s, u(X_s)) \, \mathrm{d}W_s \right)^2 \right]
$$

$$
(3.5) \quad = \mathbb{E}_{X_0 \sim \mu, u} \left[\left(\int_0^{T \wedge \tau} e^{-\gamma s} [(\mathcal{L}^u V - \gamma V)(X_s; \theta_V) + f(X_s, u(X_s))] \, \mathrm{d}s \right)^2 \right]
$$

$$
(3.6) \quad + \mathbb{E}_{X_0 \sim \mu, u} \left[\int_0^{T \wedge \tau} e^{-2\gamma s} \left| \sigma^\top(X_s, u(X_s)) \left(\nabla_x V(X_s; \theta_V) - G(X_s; \theta_G) \right) \right|^2 \, \mathrm{d}s \right],
$$

where $\mathcal{L}^u$ is the generator of the SDE and we have used Itô's isometry in the second step. Note that the first (3.5) and second (3.6) terms in (3.4) simultaneously enforce V to be the value function and its gradient to be consistent with G.

For a neural network parametrization of V, it is not easy to directly impose the Dirichlet boundary condition $V = g$ on $\partial \Omega$ in the parametrization. Thus, instead, we add a penalty term to the loss functions (3.2) or (3.3) for the critic to help enforce the boundary condition

$$
(3.7) \qquad \eta \, \mathbb{E}_{X \sim \text{Unif}(\partial \Omega)} \left[(V(X; \theta_V) - g(X))^2 \right],
$$

where η is a penalty hyperparameter and $\text{Unif}(\partial \Omega)$ denotes the uniform distribution on $\partial \Omega$.

3.2. Discretization of SDEs and stochastic integrals. In the implementation, we need to simulate numerically, based on a discretization of the diffusion process with approximating stopping time and exit point. The solution to the PDE problem crucially depends on the boundary condition, and thus in control formulation, the exit time and position of the SDE at the boundary. Several schemes have been developed in the literature to deal with the stopping time and exit point of the SDEs in related scenarios. Perhaps the most natural idea is to stop at the last step of the numerical SDE before exiting the domain, which has been tested in the context of using neural networks for solving PDEs in [40]. The error of such boundary treatment has been analyzed in [23]. Moreover, several schemes have been proposed to improve the accuracy around the boundary. In [30], the authors approximate the exit position by the intersection of the domain boundary and the line segment between the consecutive two steps before and after exiting the domain. It has also been considered to reduce step size when the discretized trajectory approaches the boundary [12]. Some bias reduction schemes with the bubble-wrap or max-sampling exit condition are proposed in [44]. After studying and testing several approaches for numerical discretization in our algorithms, we present two choices of discretization and give some remarks on the other schemes.

Let us start with a naïve approach. We can discretize the SDE (2.2) by the Euler–Maruyama scheme with a given partition of interval $[0,T]$: $0 = t_0 < t_1 < \cdots < t_N = T$, where a constant step size $\Delta t = \frac{T}{N}$ is used, so $t_n = n\Delta t$. The SDE is discretized as

$$(3.8) \qquad \mathcal{X}_0 = X_0, \qquad \mathcal{X}_{t_{n+1}} = \mathcal{X}_{t_n} + b(\mathcal{X}_{t_n}, u_n)\Delta t + \sigma(\mathcal{X}_{t_n}, u_n)\xi_n\sqrt{\Delta t},$$

where $u_n = u(\mathcal{X}_{t_n}; \theta_u)$ and $\xi_n \sim N(0, I_{d_w})$ follows the standard normal distribution. Here, we use $\mathcal{X}_{t_n}$ to denote the discretized stochastic process, to distinguish from X_t, the continuous process. Given a numerical trajectory $\mathcal{X}_{t_n}, n = 0, \ldots, N$, we define

$$(3.9) \qquad \bar{n} = \max\{n \in \{0, \ldots, N\} \mid \mathcal{X}_{t_i} \in \Omega, \ i = 0, 1, \ldots, n\}.$$

Thus, if $\bar{n} < N$, $\mathcal{X}_{t_{\bar{n}+1}}$ exits the domain as $\mathcal{X}_{t_{\bar{n}+1}} \notin \Omega$, while if $\bar{n} = N$ the trajectory $\mathcal{X}_{t_n}$ remains in the domain for $n = 0, 1, \ldots, N$.

Perhaps the most direct and intuitive approach for the boundary treatment is to view $t = \bar{n}\Delta t$ as the stopping time, even though $\mathcal{X}_{t_{\bar{n}}}$ is still inside Ω. This scheme will be referred to as the "naïve scheme" in the following. The stochastic integrations in (2.31), (2.32), and (2.36) are correspondingly approximated by

$$(3.10)$$
$$\int_0^{T \wedge \tau} e^{-\gamma s} f(X_s, u_s)\, ds \approx \sum_{n=0}^{\bar{n}-1} e^{-\gamma n \Delta t} f(\mathcal{X}_{t_n}, u_n)\Delta t,$$

$$\int_0^{T \wedge \tau} e^{-\gamma s} \nabla V(X_s)^\top \sigma(X_s, u_s)\, dW_s \approx \sum_{n=0}^{\bar{n}-1} e^{-\gamma n \Delta t} G(\mathcal{X}_{t_n}; \theta_G)^\top \sigma(\mathcal{X}_{t_n}, u_n)\xi_n\sqrt{\Delta t},$$

where $\xi_n\sqrt{\Delta t}$ is the same realization of Brownian increments as in (3.8). We remark that this discretization scheme is similar to the one used in [40] for solving degenerate semilinear elliptic equations, in particular, both algorithms approximate the stopping time by $\bar{n}\Delta t$. However, we aim to solve the value function in the whole domain, while the method developed in [40] only aims at the value at a specific point; thus the overall framework of the algorithm is quite different.

After discretization, the loss functions (2.31), (2.32), and (2.36) are further approximated by Monte Carlo samples: for each iteration, we draw K independent sample trajectories (K is known as the batch size) by drawing initial point X_0 from the distribution μ and independent increments of the Brownian motion. At each iteration, we also draw K independent Monte Carlo samples uniformly from the boundary to approximate the expectation in (3.7). To update the parameters of the neural networks, we employ the Adam optimizer [37].

To apply the policy gradient method to the loss functional (2.36), we need to differentiate the discretized functional w.r.t. the control, similar to the functional derivative setting considered above in (2.37). While the first and third terms in (2.37), which involve derivatives of J through its dependence on u and the trajectory, can be easily dealt with on the discretized level using autodifferentiation, the second and fourth terms in (2.37) become tricky to deal with on the discrete level, since the stopping time is approximated by $\bar{n}\Delta t$, which is discrete so $\delta\bar{n}/\delta u$ is not really well defined. In our implementation, such terms are omitted in the policy gradient w.r.t. u; we leave a better numerical treatment of such terms to future works.

The pseudocode for our actor-critic method for solving high dimensional PDEs is summarized in Algorithm 3.1.

Algorithm 3.1 Neural network based actor-critic solver for fully nonlinear PDEs.

---

input : A fully nonlinear PDE (2.1), terminal time T, number of time intervals N, loss weights η, neural network structures, number of iterations, learning rate, batch size K, the choice of TD

output: Value function $V(\,\cdot\,;\theta_V)$, its gradient $G(x;\theta_G)$ if we choose VR-LSTD, and the control $u(\,\cdot\,;\theta_u)$

initialization: θ_{value} ($\theta_{\text{value}} = (\theta_V, \theta_G)$ for VR-LSTD and $\theta_{\text{value}} = \theta_V$ for LSTD) and θ_u **for** $\ell = 1$ **to** *the number of iterations* **do**

> /* critic steps */
> Sample K independent trajectories $\mathcal{X}_{t_n}^k, k = 1, 2, \ldots, K$
> Sample K points on the boundary $\partial\Omega$ to enforce the boundary condition
> Estimate the gradient of the chosen critic loss ((3.2) + (3.7) or (3.3) + (3.7)) w.r.t.
> θ_{value} using the K trajectories and K boundary points
> Update parameters θ_{value} using the Adam optimizer
> /* actor steps */
> Sample K independent trajectories $\mathcal{X}_{t_n}^k, k = 1, 2, \ldots, K$
> Estimate the gradient of the actor loss (2.36) w.r.t. θ_u using the K trajectories
> Update parameters θ_u using the Adam optimizer

end

3.3. The adaptive step size scheme. It turns out in our numerical experiments that while the above naïve scheme is able to get reasonably accurate value functions, the approximation to control results in large errors, especially near the boundary (see section 4 for more details). To improve the accuracy near the boundary, we adaptively shrink the step size when the trajectory approaches the boundary $\partial\Omega$, instead of using the uniform time step size as in the naïve scheme. More specifically, we use the following scheme at the boundary, which is motivated by the integration scheme used in [12] for the Feynman–Kac representation of boundary value problems of the Poisson equation.

The idea is to reduce the time step size adaptively when $\mathcal{X}_t$ is close to the boundary, and thus to improve the accuracy of the trajectory. We consider the Euler–Maruyama scheme with varying step size given by

$$(3.11) \qquad \mathcal{X}_{t_{n+1}} = \mathcal{X}_{t_n} + b(\mathcal{X}_{t_n}, u_n) h(\mathcal{X}_{t_n}) + \sigma(\mathcal{X}_{t_n}, u_n) \sqrt{h(\mathcal{X}_{t_n})}\, \xi_n,$$

where the step size $h(\mathcal{X}_{t_n})$ depends on the current position of the trajectory. For the choice of step size, we define a subset near the boundary of Ω as

$$(3.12) \qquad \Gamma = \big\{ x \in \overline{\Omega} \mid \operatorname{dist}(x, \partial\Omega) \leq \varsigma \sqrt{3d\Delta t} \big\},$$

where $\varsigma = \sup_{x \in \Omega, u \in U} \|\sigma(x, u)\|$ is the supremum of the operator norm of σ. The adaptive choice of the step size is specified as follows:

1. When $\mathcal{X}_{t_n} \in \Omega \backslash \Gamma$, it would be considered in the "interior" of Ω, as it is very unlikely that after one time step with step size Δt that the trajectory will exit the domain. Thus, we will use the basic constant step size $h(\mathcal{X}_{t_n}) = \Delta t$.

2. When $\mathcal{X}_{t_n} \in \Gamma$, we decrease the step size according to the distance of the trajectory to the boundary, with the minimum step size set as $\frac{1}{10^4}\Delta t$:

$$h(\mathcal{X}_{t_n}) = \max\Big\{ \frac{1}{3d\varsigma^2} \operatorname{dist}(\mathcal{X}_{t_n}, \partial\Omega)^2, \frac{1}{10^4}\Delta t \Big\}.$$

This reduced step size, together with the width of Γ defined in (3.12), are decided such that the probability that $\mathcal{X}_{t_n} \in \Omega \backslash \Gamma$ goes out of Ω in the next step is small. Note that when the step size is small, the diffusion dominates the drift term, and thus it suffices to incorporate the diffusion part in the choice. Note that we have used the supremum of $\|\sigma\|$ for simplicity, one could also choose the criteria more locally if σ varies a lot across the domain. The minimum step size is set to balance the accuracy and computational cost as, otherwise, the scheme might spend an unnecessarily long time resolving the trajectory near the domain boundary.

In summary, we choose the adaptive step size $h = h(\mathcal{X}_{t_n})$ as

$$(3.13) \qquad h(\mathcal{X}_{t_n}) = \begin{cases} \Delta t, & \mathcal{X}_{t_n} \in \Omega \backslash \Gamma, \\ \max\{ \frac{1}{3d\varsigma^2} \operatorname{dist}(\mathcal{X}_{t_n}, \partial\Omega)^2, \frac{1}{10^4}\Delta t \}, & \mathcal{X}_{t_n} \in \Gamma. \end{cases}$$

It should be noted that, as a result of the adaptive step size, different trajectories may have different discretized time steps. The integrals are similarly discretized as in (3.10) with the step size changed to $h(\mathcal{X}_{t_n})$:

$$(3.14)$$

$$\int_0^{T \wedge \tau} e^{-\gamma s} f(X_s, u)\, \mathrm{d}s \approx \sum_{n=0}^{\bar{n}-1} e^{-\gamma \sum_{k=0}^{n-1} h(\mathcal{X}_{t_k})} f(\mathcal{X}_{t_n}, u_n) h(\mathcal{X}_{t_n}),$$

$$\int_0^{T \wedge \tau} e^{-\gamma s} \nabla V(X_s)^\top \sigma(X_s)\, \mathrm{d}W_s$$

$$\approx \sum_{n=0}^{\bar{n}-1} e^{-\gamma \sum_{k=0}^{n-1} h(\mathcal{X}_{t_k})} G(\mathcal{X}_{t_n}; \theta_G)^\top \sigma(\mathcal{X}_{t_n}, u_n) \xi_n \sqrt{h(\mathcal{X}_{t_n})}.$$

For the policy gradient, similarly to our numerical treatment in the case of the naïve scheme, we use autodifferentiation generated by the computational graph in practice, instead of directly numerically approximating the functional derivative de-

fined in (2.37). One reason is that the adaptive step size scheme further complicates the dependence of the trajectory and exit time on the control, compared with the naïve scheme and, hence, makes the direct numerical discretization of (2.37) even more difficult. In practice, the result from using autodifferentiation for the policy gradient seems to be quite accurate, as will be further discussed in the next section.

Remark 3.1. In addition to the adaptive step size, in our numerical experiments, we have also tested the bounded sample of Brownian increments proposed in [12] to further avoid the potentially large error of the trajectory near the boundary due to tail events of the normal sample. We do not find, however, a significant difference in the result between using bounded samples versus the usual normal samples for Brownian increments. Therefore, we will stick to the normal samples for simplicity.

Remark 3.2. Moreover, besides adaptively shrinking the step size near the boundary, we have tested two approaches using constant step size, but try to improve the estimate of the exit time and exit point of the naïve scheme instead. They do not yield satisfactory numerical results, so we will only briefly sketch the ideas without going into details or presenting numerical results.

One scheme is adapted from [30], which tries to determine the exit point on $\partial\Omega$ more accurately. In this scheme, the exit position $\mathcal{X}_\tau$ on $\partial\Omega$ is numerically approximated by the intersection of $\partial\Omega$ and the line segment between $\mathcal{X}_{t_{\bar{n}}}$ and $\mathcal{X}_{t_{\bar{n}+1}}$; the stopping time is correspondingly adjusted. The numerical result from the scheme is still not accurate enough for the control near the boundary.

The linear interpolation above gives an error of order $\sqrt{\Delta t}$ due to the diffusion term; we can further improve the accuracy using a method proposed in [23]. Instead of linear interpolation, we seek for a coefficient $\rho \in (0, 1]$ such that $\mathcal{X}_\tau$, defined by

$$\mathcal{X}_\tau = \mathcal{X}_{t_{\bar{n}}} + b(\mathcal{X}_{t_{\bar{n}}}, u_{\bar{n}})\rho\Delta t + \sigma(\mathcal{X}_{t_{\bar{n}}}, u_{\bar{n}})\sqrt{\rho\Delta t}\,\xi_{\bar{n}},$$

is on $\partial\Omega$. In practice, we observe that numerically solving the coefficient ρ makes the training unstable.

4. Numerical examples. In this section, we present the numerical results for the proposed method. We test it on several examples: the linear quadratic regulator problems, the stochastic Van der Pol oscillator problems, the diffusive Eikonal equations, and fully nonlinear elliptic PDEs derived from a regulator problem. To test the performance of our algorithm, we do not assume knowledge of the true solution or the explicit formula for the control given the value function. The considered dimensions in all four examples are as large as 20. The algorithm is implemented in Python with the deep learning library TensorFlow 2.0 [1]. In all the examples, the weight parameter η associated with the boundary condition (cf. (3.7)) is set to 1 and the terminal time is $T = 0.2$. The numbers of time intervals are $N = 50$ for problems in 4 dimensions (4d) and five dimensions (5d), and $N = 100$ in 10 dimenions (10d) and 20 dimensions (20d). As for the architecture of the neural networks, the width of the hidden layers is set to 200 in all problems, while the numbers of hidden layers are 2 for problems in 4d and 5d, and 3 in 10d and 20d. During the training, we use piecewise constant learning rates of $1e-3$, $1e-4$, and $1e-5$ consecutively in order to achieve high accuracy. The numbers of steps with learning rate $1e-3$ are 20000 for problems in 4d, 5d, and 10d, and 30000 in 20d. The numbers of steps with learning rate $1e-4$ and $1e-5$ are both 10000 in the four examples. The batch sizes are $K = 1024$ for problems in 4d and 5d, and $K = 2048$ in 10d and 20d. The parameters in the numerical examples are determined empirically. In order to illustrate the effect

of some parameters such as T and the basic step size Δt, we also compare the results with different parameters in the first example.

During the training, we sample a validation set $\{X^k\}_{k=1}^K$ uniformly in Ω, independent of the training, to evaluate the errors of the value function and the control. Note that the validation size K is the same as the batch size. We find that such sizes are enough to estimate the error accurately with a small variance. The relative L^2 errors are computed by

$$(4.1) \qquad \mathrm{err}_V^2 = \sum_{k=1}^K (V(X^k) - V(X^k; \theta_V))^2 / \sum_{k=1}^K V(X^k)^2$$

and

$$(4.2) \qquad \mathrm{err}_u^2 = \sum_{k=1}^K |u(X^k) - u(X^k; \theta_u)|^2 / \sum_{k=1}^K |u(X^k)|^2,$$

where $V(\cdot)$ and $u(\cdot)$ are the true value and control functions, respectively (we will choose test examples such that these true solutions are known). In addition to the errors above, we also visualize the density of the true value function and compare that with its neural network approximation, considering the difficulty of visualizing functions in high dimensions directly. Here, the density of a function V is defined as the probability density function of $V(X)$, where X is uniformly distributed in Ω. In our numerical experiments, the density is estimated by Monte Carlo sampling.

Our numerical results indicate that in all the examples, the value functions are approximated accurately, and the associated densities match well with that of the true solution. Furthermore, the numerical results show that, for the critic, the VR-LSTD performs better than LSTD, as expected. The adaptive step size scheme also significantly improves the accuracy, in particular, for the control. The details can be found in the following subsections. The code developed to solve these numerical examples is made publicly available on GitHub [69].

4.1. Linear quadratic regulator. In this subsection we consider the PDE arising from the linear-quadratic regulator (LQR) problem, given by

$$(4.3) \quad \Delta V(x) + \inf_{u \in \mathbb{R}^d} \left(\beta u^\top \nabla V(x) + p|x|^2 + q|u|^2 - 2kd \right) - \gamma V(x) = 0 \quad \text{in } B_R \subset \mathbb{R}^d$$

with boundary condition $V(x) = kR^2$ on ∂B_R, where $B_R = \{x \in \mathbb{R}^d : |x| < R\}$. Here p, q, β, k are positive constants such that

$$(4.4) \qquad k = \frac{\sqrt{q^2\gamma^2 + 4pq\beta^2} - \gamma q}{2\beta^2}.$$

This is the HJB equation corresponding to the controlled stochastic process

$$(4.5) \qquad \mathrm{d}X_t = \beta u \,\mathrm{d}t + \sqrt{2}\,\mathrm{d}W_t$$

with cost functional

$$(4.6) \qquad J^u(x) = \mathbb{E}\Big[\int_0^\tau (p|X_s|^2 + q|u(X_s)|^2 - 2kd)e^{-\gamma s}\,\mathrm{d}s + e^{-\gamma \tau}kR^2 \Big],$$

where τ is the exit time of the domain B_R. The PDE has the exact solution as a quadratic function, $V(x) = k|x|^2$, and the optimal control is also explicitly given as $u^*(x) = \frac{-\beta}{2q}\nabla V(x) = \frac{-k\beta}{q}x$.

TABLE 1
Errors for different discretization schemes and TDs in 5d LQR.

discretization	TD variant	error of value function	error of control
adaptive step size	VR-LSTD	**1.02e − 2**	**9.19e − 3**
adaptive step size	LSTD	1.58e − 1	1.17e − 1
naïve	VR-LSTD	1.29e − 2	1.24e − 1
naïve	LSTD	1.41e − 1	8.55e − 2
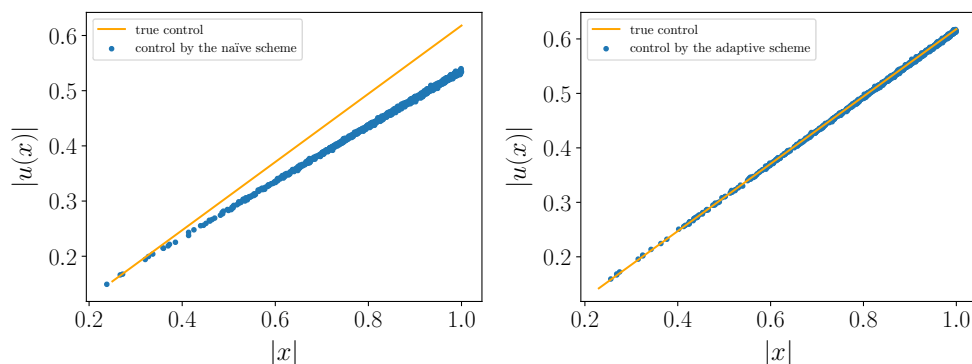
FIG. 1. *The comparison of two discretization schemes in the 5d LQR example. Left: the true optimal control and approximated control by the naïve scheme; right: the true optimal control and approximated control by the adaptive step size scheme. x-axis: the norm of x; y-axis: the norm of control u.*

We choose the model parameters $p = q = R = \beta = \gamma = 1$ and $k = (\sqrt{5} - 1)/2$. The numerical results for our two versions of TDs with different discretization schemes in 5d are shown in Table 1. The results of VR-LSTD have smaller errors due to the smaller asymptotic variance, as we discussed above. Moreover, the adaptive step size scheme is able to compute a more accurate control function, compared with the naïve scheme. One possible reason is that the adaptive step size scheme samples more points $\mathcal{X}_{t_n}$ near the boundary, which helps to improve the accuracy of the control function near the boundary. To further illustrate the idea, let us compare the results for the naïve scheme and the adaptive step size scheme in 5d, both with critics optimized by VR-LSTD. The plot of the norm of the control $|u(x)|$ w.r.t. the norm of the variable $|x|$ is shown in Figure 1. The error of the control for the naïve scheme is significantly larger near the boundary. The adaptive step size scheme achieves a uniform accuracy of the control in the whole domain.

Therefore, for the rest of the numerical experiments, we will stick to the adaptive step size scheme and VR-LSTD loss function for the critic. Figure 2 shows the density and error curves for the LQR problem when $d = 5, 10, 20$. The sharp drop of the errors at steps 20000 and 30000 is due to the reduced learning rates at those steps. The final errors of the value functions and controls are $1.02e − 2$ and $9.19e − 3$ in 5d; $1.40e − 2$ and $1.95e − 2$ in 10d; $1.96e − 2$ and $4.78e − 2$ in 20d.

Considering that T and the basic step size Δt are two hyperparameters in our algorithm, we test different choices of their values in the 5d LQR example and provide the errors in Table 2. The results show that our algorithm is not sensitive to the choice of T and Δt.
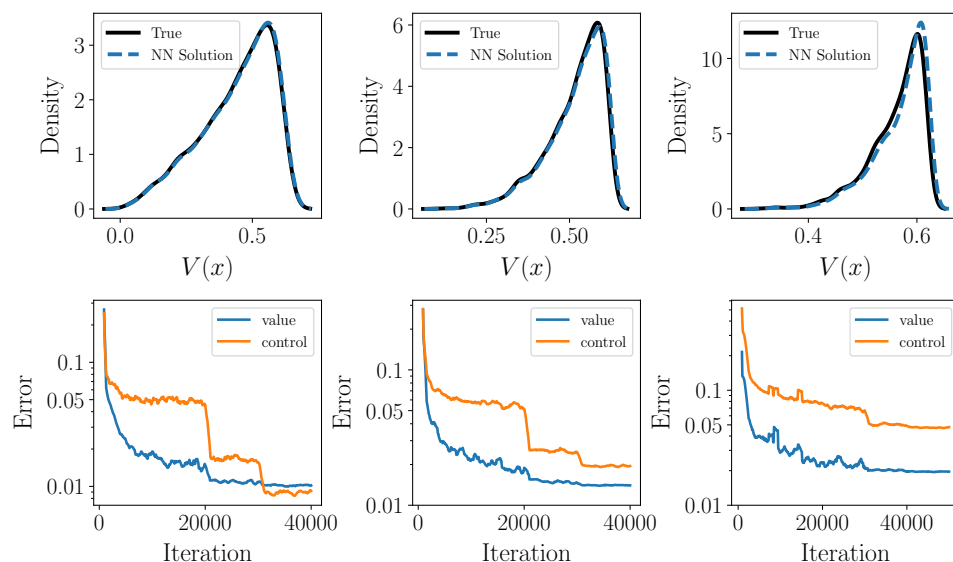
FIG. 2. *Top: density of V for the LQR problem with $d = 5$ (left), $d = 10$ (middle), and $d = 20$ (right). Bottom: associated error curves in the training process with $d = 5$ (left), $d = 10$ (middle), and $d = 20$ (right).*

TABLE 2
Errors of value and control functions with different parameters in the 5d LQR example using adaptive step sizes and VR-LSTD. The first two rows denote different T's with the same number of time intervals $N = 50$. The last two rows denote different T's with the same basic step size $\Delta t = 0.004$. The relationship $T = N\Delta t$ always holds.

T		$T = 0.04$	$T = 0.1$	$T = 0.2$	$T = 0.4$	$T = 0.8$
50 time intervals	value	$8.40e-3$	$9.46e-3$	$1.04e-2$	$1.03e-2$	$9.97e-3$
	control	$1.15e-2$	$9.87e-3$	$1.01e-2$	$8.99e-3$	$8.33e-3$
step size 0.004	value	$3.17e-2$	$1.40e-2$	$9.96e-3$	$9.16e-3$	$8.76e-3$
	control	$3.51e-2$	$1.19e-2$	$9.39e-3$	$1.03e-2$	$1.07e-2$

4.2. Stochastic Van der Pol oscillator. The Van der Pol oscillator is a popular example in the study of dynamical systems because of its chaotic behavior. The stochastic Van der Pol oscillator has been studied in [67], in which some internal or external noise is considered. In this subsection, we consider the generalized stochastic Van der Pol oscillator in high dimensional cases and solve the PDE

$$(4.7) \qquad \Delta V(x) + \inf_{u \in \mathbb{R}^{d/2}} \left[b(x,u)^\top \nabla V(x) + f(x,u) \right] - \gamma V(x) = 0 \quad \text{in } B_R \subset \mathbb{R}^d,$$

where $d = 2n$ is even. The boundary condition is given by (with convention $x_0 = x_n$ and $x_{2n+1} = x_{n+1}$)

$$(4.8) \qquad g(x) = a \sum_{i=1}^{2n} (x_i)^2 - \epsilon \left(\sum_{i=1}^{n} x_{i-1} x_i + \sum_{i=n+1}^{2n} x_i x_{i+1} \right).$$

Here a and ϵ are positive constants. The drift field is given by

$$(4.9) \qquad b_i(x, u) = \begin{cases} x_{i+n} & (1 \le i \le n), \\ (1 - x_{i-n}^2)x_i - x_{i-n} + u_{i-n} & (n+1 \le i \le 2n). \end{cases}$$

We choose the running cost as
(4.10)

$$\begin{aligned} f(x, u) = q|u|^2 &+ \gamma\Big[\sum_{i=1}^{n}(ax_i^2 - \epsilon x_i x_{i-1}) + \sum_{i=n+1}^{2n}(ax_i^2 - \epsilon x_i x_{i+1})\Big] \\ &+ \frac{1}{4q}\Big[(2ax_{n+1} - \epsilon x_{2n} - \epsilon x_{n+2})^2 + \sum_{i=n+2}^{2n}(2ax_i - \epsilon x_{i-1} - \epsilon x_{i+1})^2\Big] - 4na \\ &- 2a\sum_{i=1}^{n}x_{n+i}x_i + \epsilon\sum_{i=1}^{n}x_{n+i}x_{i-1} + \epsilon\sum_{i=1}^{n-1}x_{n+i}x_{i+1} + \epsilon x_{2n}x_1 \\ &- (x_{n+1} - x_1 - x_1^2 x_{n+1})(2ax_{n+1} - \epsilon x_{2n} - \epsilon x_{n+2}) \\ &- \sum_{i=2}^{n}(x_{i+n} - x_i - x_i^2 x_{i+n})(2ax_{i+n} - \epsilon x_{i+n-1} - \epsilon x_{i+n+1}), \end{aligned}$$

so that the true value function has an explicit formula:

$$(4.11) \qquad V(x) = a\sum_{i=1}^{2n}(x_i)^2 - \epsilon\Big(\sum_{i=1}^{n}x_{i-1}x_i + \sum_{i=n+1}^{2n}x_i x_{i+1}\Big).$$

The corresponding optimal control is given by $u_1^*(x) = -\frac{1}{2q}\partial_{n+1}V(x) = 2ax_{n+1} - \epsilon x_{2n} - \epsilon x_{n+2}$ and $u_i^*(x) = -\frac{1}{2q}\partial_{i+n}V(x) = 2ax_{i+n} - \epsilon x_{i+n-1} - \epsilon x_{i+n+1}$ for $i = 2, 3, \ldots, n$.

The PDE can be reformulated as a stochastic control problem with the controlled SDE given by

$$(4.12) \qquad \mathrm{d}X_t = b(X_t, u)\,\mathrm{d}t + \sqrt{2}\,\mathrm{d}W_t$$

with objective function

$$(4.13) \qquad J^u(x) = \mathbb{E}\Big[\int_0^\tau f(X_s, u)e^{-\gamma s}\,\mathrm{d}s + e^{-\gamma\tau}g(X_\tau)\Big].$$

In the numerical experiments, we take $a = q = R = \gamma = 1$ and $\epsilon = 0.1$. Figure 3 shows the density and error curves when $d = 4, 10, 20$. The algorithm learns reasonably nice shapes of the value functions. The final errors of the value functions and controls are $1.01\mathrm{e} - 2$ and $5.12\mathrm{e} - 3$ in 4d; $1.19\mathrm{e} - 2$ and $9.77\mathrm{e} - 3$ in 10d; $1.81\mathrm{e} - 2$ and $2.50\mathrm{e} - 2$ in 20d.

4.3. Diffusive Eikonal equation. The Eikonal equation corresponds to the shortest-path problems with a given metric. In our experiments, we add a small diffusion term to regularize the equation (otherwise, the solution has kinks, which creates difficulty for the neural networks to approximate well in high dimensions). The diffusive Eikonal equation is given by

$$(4.14) \qquad \begin{cases} \epsilon\Delta V(x) + \inf_{u \in B_1}\big(c(x)u^\top \nabla V(x)\big) + 1 = 0 & \text{in } B_R, \\ V(x) = a_3 - a_2 & \text{on } \partial B_R, \end{cases}$$
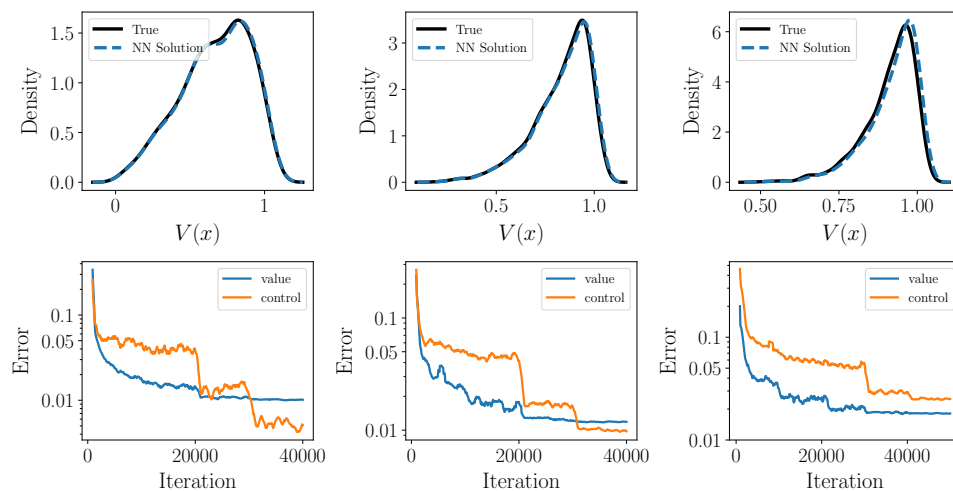
FIG. 3. *Top: density of V for the Van der Pol problem with $d = 4$ (left), $d = 10$ (middle), and $d = 20$ (right), Bottom: associated error curves in the training process with $d = 4$ (left), $d = 10$ (middle), and $d = 20$ (right),*

where

$$c(x) = \frac{3(d+1)a_3}{2da_2(2a_2 - 3a_3|x|)} > 0 \tag{4.15}$$

is a real valued function. Here a_2 and a_3 are positive constants such that $2a_2 - 3a_3R > 0$ and $\epsilon = 1/(2da_2)$. We choose the form of c so that the true solution of the PDE is explicitly given by

$$V(x) = a_3|x|^3 - a_2|x|^2 \tag{4.16}$$

and the optimal control is $u^*(x) = x/|x|$. In the numerical test, we take $a_2 = 1.2$, $a_3 = 0.2$, and $R = 1$.

Unlike the previous two examples, the constraint on the control in this example poses a new challenge to the numerical algorithm. In order to ensure that the control u is in the unit ball, we construct a specific structure of the neural network for the control. Instead of outputting the control directly, the neural network gives a $d + 1$ dimensional vector $(u_{\text{len}}, u_{\text{dir}}) \in \mathbb{R}^{d+1}$. The control is represented by

$$u = \frac{u_{\text{dir}}}{\delta + \text{ReLU}(u_{\text{len}}) + |u_{\text{dir}}|}, \tag{4.17}$$

where $\text{ReLU}(x) = \max(0, x)$ and $\delta = 10^{-15}$. This δ is to ensure that the denominator in (4.17) is not 0 to prevent numerical singularity. Figure 4 shows the density and error curves for the Eikonal equation when $d = 5, 10, 20$. We also tried the straightforward parametrization of the control function as before, with an additional penalty term $\eta'\mathbb{E}_{X\sim\text{Unif}(\Omega)}[\text{ReLU}(|u(X)| - 1)]$ in the loss for the actor. However, the numerical performances indicate that implementing the constraints of control directly like (4.17) is better than the penalty method. The final errors of the value functions and controls are $6.97\mathrm{e}-3$ and $6.03\mathrm{e}-3$ in 5d; $1.02\mathrm{e}-2$ and $1.76\mathrm{e}-2$ in 10d; $1.82\mathrm{e}-2$ and $4.14\mathrm{e}-2$ in 20d.
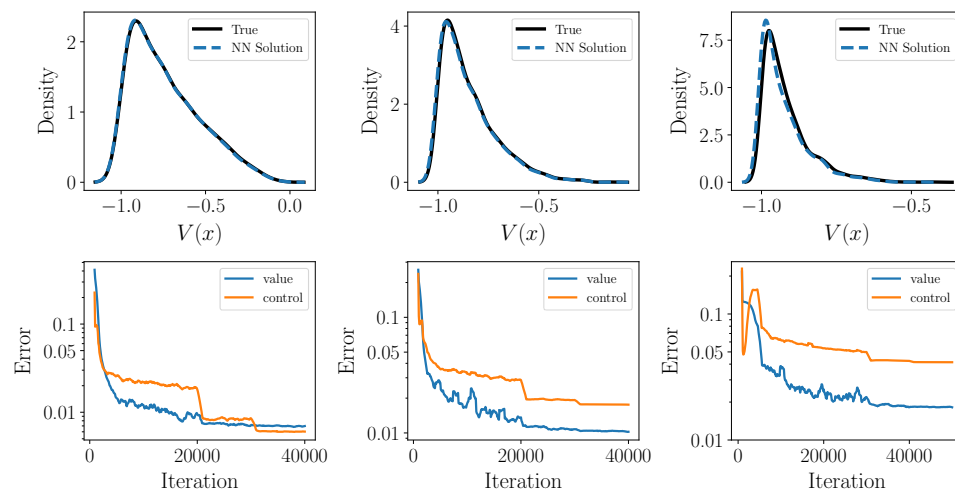
FIG. 4. *Top: density of V for the Eikonal equation with $d = 5$ (left), $d = 10$ (middle), and $d = 20$ (right). Bottom: associated error curves in the training process with $d = 5$ (left), $d = 10$ (middle), and $d = 20$ (right).*

4.4. LQR with a nonconstant diffusion coefficient. In this subsection, we consider a variant of the LQR in which the diffusion coefficient σ is a function of both x and u. Consider the HJB equation

(4.18)
$$\inf_{u \in \mathbb{R}^d} \left[\sum_{i=1}^{d} \left(\partial_i^2 V(x)(1 + \epsilon x_i u_i)^2 + \beta \partial_i V(x) u_i \right) + q|u|^2 + \widetilde{f}(x) \right] - \gamma V(x) = 0 \quad \text{in } B_R \subset \mathbb{R}^d,$$

where

(4.19)
$$\widetilde{f}(x) = \gamma k|x|^2 + \sum_{i=1}^{d} \frac{k^2(\beta + 2\epsilon)^2 x_i^2}{q + 2k\epsilon^2 x_i^2} - 2kd.$$

In contrast to the previous three examples, this is a fully nonlinear PDE. The corresponding SDE is

(4.20)
$$\mathrm{d}X_t = \beta u_t \, \mathrm{d}t + \sigma(X_t, u_t) \, \mathrm{d}W_t,$$

where $\sigma(x, u)$ is a diagonal matrix with ith diagonal element $\sqrt{2}(1 + \epsilon x_i u_i)$, $i = 1, \ldots, d$. The running cost is $f(x, u) = q|u|^2 + \widetilde{f}(x)$. The true value function is $V(x) = k|x|^2$ and the optimal control is

(4.21)
$$u_i^*(x) = -\frac{\beta \partial_i V(x) + 2\epsilon x_i \partial_i^2 V(x)}{2q + 2\epsilon^2 x_i^2 \partial_i^2 V(x)} = -\frac{(\beta + 2\epsilon)x_i}{q/k + 2\epsilon^2 x_i^2}.$$

Note that this example coincides with the first example when $\epsilon = 0$. In the numerical experiments, we set the parameters $q = R = \beta = \gamma = 1, k = (\sqrt{5} - 1)/2$ the same as the first example, and $\epsilon = -1$. The final errors of the value functions and controls are $9.98\mathrm{e} - 3$ and $1.63\mathrm{e} - 2$ in 5d; $1.50\mathrm{e} - 2$ and $4.95\mathrm{e} - 2$ in 10d; $1.96\mathrm{e} - 2$ and $5.25\mathrm{e} - 2$ in 20d. This example showcases that our algorithm is able to solve fully nonlinear elliptic PDEs in high dimensions accurately.

5. Conclusion and future directions. In this paper, we propose and study numerical methods for high dimensional static HJB equations based on neural network parametrization and the actor-critic framework. There are several promising directions for future research. First, the scalability of the methods shall be further tested by problems of higher dimensions. Second, it would be interesting to extend our methods to other types of boundary conditions like natural boundary conditions or broader types of equations, such as the porous medium equation. In both cases, the corresponding control formulation is not so clear. Third, one might explore the better numerical treatment of discretization of the functional derivative (2.37), rather than relying on autodifferentiation. Finally, as an outstanding challenge in the field of deep learning, theoretical analysis for convergence and error analysis of the proposed numerical methods would be of great interest.

REFERENCES

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, *Tensorflow: A system for large-scale machine learning*, in Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), USENIK, 2016, pp. 265–283.

[2] M. S. Abdulla and S. Bhatnagar, *Parametrized actor-critic algorithms for finite-horizon MDPs*, in Proceedings of the 2007 American Control Conference, IEEE, Piscataway, NJ, 2007, pp. 534–539.

[3] G. Barles and E. R. Jakobsen, *On the convergence rate of approximation schemes for Hamilton–Jacobi–Bellman equations*, ESAIM Math. Model. Numer. Anal., 36 (2002), pp. 33–54.

[4] R. W. Beard, G. N. Saridis, and J. T. Wen, *Galerkin approximations of the generalized Hamilton–Jacobi–Bellman equation*, Automatica J. IFAC, 33 (1997), pp. 2159–2177.

[5] R. W. Beard, G. N. Saridis, and J. T. Wen, *Approximate solutions to the time-invariant Hamilton–Jacobi–Bellman equation*, J. Optim. Theory Appl., 96 (1998), pp. 589–626.

[6] C. Beck, M. Hutzenthaler, A. Jentzen, and B. Kuckuck, *An Overview on Deep Learning-Based Approximation Methods for Partial Differential Equations*, preprint, arXiv:2012.12348, 2020.

[7] S. Becker, P. Cheridito, and A. Jentzen, *Deep optimal stopping*, J. Mach. Learn. Res., 20 (2019), 74.

[8] R. Bellman, *Dynamic programming*, Science, 153 (1966), pp. 34–37.

[9] S. Bhatnagar and M. S. Abdulla, *A reinforcement learning based algorithm for finite horizon Markov decision processes*, in Proceedings of the 45th IEEE Conference on Decision and Control, IEEE, 2006, Piscataway, NJ, pp. 5519–5524.

[10] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee, *Natural actor-critic algorithms*, Automatica J. IFAC, 45 (2009), pp. 2471–2482.

[11] J. A. Boyan, *Least-squares temporal difference learning*, in ICML, Morgan Kaufmann, San Francisco, 1999, pp. 49–56.

[12] F. Buchmann and W. Petersen, *Solving Dirichlet problems numerically using the Feynman–Kac representation*, BIT, 43 (2003), pp. 519–540.

[13] Q. Chan-Wai-Nam, J. Mikael, and X. Warin, *Machine learning for semi linear PDEs*, J. Sci. Comput., 79 (2019), pp. 1667–1712.

[14] M. G. Crandall, H. Ishii, and P.-L. Lions, *User's guide to viscosity solutions of second order partial differential equations*, Bull. Amer. Math. Soc. (N.S.), 27 (1992), pp. 1–67.

[15] T. Degris, M. White, and R. Sutton, *Off-policy actor-critic*, in International Conference on Machine Learning, International Machine Learning Society, Madison, WI, 2012.

[16] S. Dolgov, D. Kalise, and K. K. Kunisch, *Tensor decompositions methods for high-dimensional Hamilton–Jacobi–Bellman equations*, SIAM J. Sci. Comput., 43 (2021), pp. A1625–A1650.

[17] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, *Benchmarking deep reinforcement learning for continuous control*, in International Conference on Machine Learning, International Machine Learning Society, Stroudsburg, PA, 2016, pp. 1329–1338.

[18] W. E AND J. HAN, *Deep Learning Approximation for Stochastic Control Problems*, preprint, arXiv:1611.07422, 2016.

[19] 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.

[20] W. E, J. HAN, AND A. JENTZEN, *Algorithms for solving high dimensional PDEs: from nonlinear Monte Carlo to machine learning*, Nonlinearity, 35 (2022), pp. 278–310.

[21] P. A. FORSYTH AND G. LABAHN, *Numerical methods for controlled Hamilton–Jacobi–Bellman PDEs in finance*, J. Comput. Finance, 11 (2007), pp. 1–43.

[22] W. FOULKES, L. MITAS, R. NEEDS, AND G. RAJAGOPAL, *Quantum Monte Carlo simulations of solids*, Rev. Mod. Phys., 73 (2001), pp. 33–84.

[23] E. GOBET, *Weak approximation of killed diffusion using Euler schemes*, Stochastic Process. Appl., 87 (2000), pp. 167–197.

[24] J. HAN AND R. HU, *Deep fictitious play for finding Markovian Nash equilibrium in multi-agent games*, in Proc. Mach. Learn. Res. (PMLR), 107 (2020), pp. 221–245.

[25] J. HAN AND R. HU, *Recurrent neural networks for stochastic control problems with delay*, Math. Control Signals Systems, 33 (2021), pp. 775–795.

[26] J. HAN, R. HU, AND J. LONG, *Convergence of Deep Fictitious Play for Stochastic Differential Games*, preprint, arXiv:2008.05519, 2020.

[27] 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.

[28] J. HAN AND J. LONG, *Convergence of the deep BSDE method for coupled FBSDEs*, Probab. Uncertain. Quant. Risk, 5 (2020), pp. 1–33.

[29] J. HAN, J. LU, AND M. ZHOU, *Solving high-dimensional eigenvalue problems using deep neural networks: A diffusion Monte Carlo like approach*, J. Comput. Phys., 423 (2020), 109792.

[30] J. HAN, M. NICA, AND A. R. STINCHCOMBE, *A derivative-free method for solving elliptic partial differential equations with deep neural networks*, J. Comput. Phys., 419 (2020), 109672.

[31] 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, IEEE, Piscataway, NJ, 2016, pp. 770–778.

[32] P. HENRY-LABORDERE, *Deep Primal-Dual Algorithm for BSDEs: Applications of Machine Learning to CVA and IM*, preprint SSRN 3071506, 2017.

[33] S. JI, S. PENG, Y. PENG, AND X. ZHANG, *Deep Learning Method for Solving Stochastic Optimal Control Problem via Stochastic Maximum Principle*, preprint, arXiv:2007.02227, 2020.

[34] S. JI, S. PENG, Y. PENG, AND X. ZHANG, *Three algorithms for solving high-dimensional fully coupled FBSDEs through deep learning*, IEEE Intell. Syst., 35 (2020), pp. 71–84.

[35] D. KALISE AND K. KUNISCH, *Polynomial approximation of high-dimensional Hamilton–Jacobi–Bellman equations and applications to feedback control of semilinear parabolic PDEs*, SIAM J. Sci. Comput., 40 (2018), pp. A629–A652.

[36] W. KANG AND L. C. WILCOX, *Mitigating the curse of dimensionality: Sparse grid characteristics method for optimal feedback control and HJB equations*, Comput. Optim. Appl., 68 (2017), pp. 289–315.

[37] D. P. KINGMA AND J. BA, *Adam: A method for stochastic optimization*, in Proceedings of the 3rd International Conference on Learning Representations (ICLR), San Diego, CA, 2015.

[38] F. C. KLEBANER, *Introduction to Stochastic Calculus with Applications*, World Scientific, London, 2005.

[39] V. R. KONDA AND J. N. TSITSIKLIS, *Actor-critic algorithms*, in Advances in Neural Information Processing Systems, MIT Press, Cambridge, MA, 1999, pp. 1008–1014.

[40] S. KREMSNER, A. STEINICKE, AND M. SZÖLGYENYI, *A deep neural network algorithm for semilinear elliptic PDEs with applications in insurance mathematics*, Risks, 8 (2020), 136.

[41] K. KUNISCH, S. VOLKWEIN, AND L. XIE, *HJB-POD-based feedback design for the optimal control of evolution problems*, SIAM J. Appl. Dyn. Syst., 3 (2004), pp. 701–722.

[42] J. LYGEROS, *On reachability and minimum cost optimal control*, Automatica J. IFAC, 40 (2004), pp. 917–927.

[43] H. R. MAEI, C. SZEPESVÁRI, S. BHATNAGAR, AND R. S. SUTTON, *Toward off-policy learning control with function approximation*, in Proceedings of the 27th International Conference on Machine Learning (ICML-10), Inernational Machine Learning Society, Madison, WI, 2010, pp. 719–726.

[44] C. MARTIN, H. ZHANG, J. COSTACURTA, M. NICA, AND A. R. STINCHCOMBE, *Solving elliptic equations with Brownian motion: bias reduction and temporal difference learning*, Methodol. Comput. Appl. Probab., (2021), pp. 1–24.

[45] I. M. MITCHELL, A. M. BAYEN, AND C. J. TOMLIN, *A time-dependent Hamilton–Jacobi formulation of reachable sets for continuous dynamic games*, IEEE Trans. Automat. Control, 50 (2005), pp. 947–957.

[46] I. M. MITCHELL AND C. J. TOMLIN, *Overapproximating reachable sets by Hamilton–Jacobi projections*, J. Sci. Comput., 19 (2003), pp. 323–346.

[47] V. MNIH, K. KAVUKCUOGLU, D. SILVER, A. GRAVES, I. ANTONOGLOU, D. WIERSTRA, AND M. RIEDMILLER, *Playing Atari with Deep Reinforcement Learning*, preprint, arXiv:1312.5602, 2013.

[48] T. NAKAMURA-ZIMMERER, Q. GONG, AND W. KANG, *Adaptive deep learning for high-dimensional Hamilton–Jacobi–Bellman equations*, SIAM J. Sci. Comput., 43 (2021), pp. A1221–A1247.

[49] N. NÜSKEN AND L. RICHTER, *Solving high-dimensional Hamilton–Jacobi–Bellman PDEs using neural networks: Perspectives from the theory of controlled diffusions and measures on path space*, Partial Differ. Equ. Appl., 2 (2021), 48.

[50] S. OSHER AND J. A. SETHIAN, *Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton–Jacobi formulations*, J. Comput. Phys., 79 (1988), pp. 12–49.

[51] M. OSTER, L. SALLANDT, AND R. SCHNEIDER, *Approximating the Stationary Hamilton–Jacobi–Bellman Equation by Hierarchical Tensor Products*, preprint, arXiv:1911.00279, 2019.

[52] É. PARDOUX, *Backward stochastic differential equations and viscosity solutions of systems of semilinear parabolic and elliptic PDEs of second order*, in Stochastic Analysis and Related Topics VI, Springer, Boston, 1998, pp. 79–127.

[53] E. PARDOUX AND S. PENG, *Adapted solution of a backward stochastic differential equation*, Systems Control Lett., 14 (1990), pp. 55–61.

[54] E. PARDOUX AND S. PENG, *Backward stochastic differential equations and quasilinear parabolic partial differential equations*, in Stochastic Partial Differential Equations and Their Applications, Springer, Berlin, 1992, pp. 200–217.

[55] S. PENG, *Probabilistic interpretation for systems of quasilinear parabolic partial differential equations*, Stoch. Stoch. Rep., 37 (1991), pp. 61–74.

[56] M. A. PEREIRA, Z. WANG, I. EXARCHOS, AND E. A. THEODOROU, *Learning Deep Stochastic Optimal Control Policies Using Forward-backward SDEs*, in Robotics: Science and Systems, MIT Press, Cambridge, MA, 2019.

[57] J. PETERS AND S. SCHAAL, *Natural actor-critic*, Neurocomputing, 71 (2008), pp. 1180–1190.

[58] H. PHAM, X. WARIN, AND M. GERMAIN, *Neural Networks-Based Backward Scheme for Fully Nonlinear PDEs*, preprint, arXiv:1908.00412, 2019.

[59] A. V. RAO, *A survey of numerical methods for optimal control*, Adv. Astronau. Sci., 135 (2009), pp. 497–528.

[60] S. RICHARDSON AND S. WANG, *Numerical solution of Hamilton–Jacobi–Bellman equations by an exponentially fitted finite volume method*, Optimization, 55 (2006), pp. 121–140.

[61] D. SILVER, A. HUANG, C. J. MADDISON, A. GUEZ, L. SIFRE, G. VAN DEN DRIESSCHE, J. SCHRITTWIESER, I. ANTONOGLOU, V. PANNEERSHELVAM, AND M. LANCTOT, *Mastering the game of Go with deep neural networks and tree search*, Nature, 529 (2016), pp. 484–489.

[62] D. SILVER, G. LEVER, N. HEESS, T. DEGRIS, D. WIERSTRA, AND M. RIEDMILLER, *Deterministic policy gradient algorithms*, in International Conference on Machine Learning, International Machine Learning Society, Stroudsburg, PA, 2014, pp. 387–395.

[63] R. S. SUTTON AND A. G. BARTO, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, 2018.

[64] K. G. VAMVOUDAKIS AND F. L. LEWIS, *Online actor-critic algorithm to solve the continuous-time infinite horizon optimal control problem*, Automatica J. IFAC, 46 (2010), pp. 878–888.

[65] S. WANG, L. S. JENNINGS, AND K. L. TEO, *Numerical solution of Hamilton–Jacobi–Bellman equations by an upwind finite volume method*, J. Global Optim., 27 (2003), pp. 177–192.

[66] Z. WANG, V. BAPST, N. HEESS, V. MNIH, R. MUNOS, K. KAVUKCUOGLU, AND N. DE FREITAS, *Sample Efficient Actor-Critic with Experience Replay*, preprint, arXiv:1611.01224, 2016.

[67] Y. XU, R. GU, H. ZHANG, W. XU, AND J. DUAN, *Stochastic bifurcations in a bistable Duffing–Van der Pol oscillator with colored noise*, Phys. Rev. E(3), 83 (2011), 056215.

[68] J. YONG AND X. ZHOU, *Stochastic Controls: Hamiltonian Systems and HJB Equations*, Appl. Math. 43, Springer, New York, 1999.

[69] M. ZHOU, J. HAN, AND J. LU, Code for "*Actor-critic Method for High Dimensional Static Hamilton–Jacobi–Bellman Partial Differential Equations Based on Neural Network*", https://github.com/MoZhou1995/DeepPDE_ActorCritic.