DEEPMARTNET - A MARTINGALE BASED DEEP NEURAL NETWORK LEARNING METHOD FOR DIRICHLET BVPS AND EIGENVALUE PROBLEMS OF ELLIPTIC PDES IN R^d *

WEI CAI†, ANDREW HE‡, AND DANIEL MARGOLIS§

Abstract. In this paper, we propose DeepMartNet - a Martingale based deep neural network learning method for solving Dirichlet boundary value problems (BVPs) and eigenvalue problems for elliptic partial differential equations (PDEs) in high dimensions or domains with complex geometries. The method is based on Varadhan's Martingale problem formulation for the BVPs/eigenvalue problems where a loss function enforcing the Martingale property for the PDE solution is used for an efficient optimization by sampling the stochastic processes associated with corresponding elliptic operators. High dimensional numerical results for BVPs of the linear and nonlinear Poisson-Boltzmann equation and eigenvalue problems of the Laplace equation and a Fokker-Planck equation demonstrate the capability of the proposed DeepMartNet learning method in solving high dimensional PDE problems.

Key words. Martingale problem, Deep neural network, boundary value problems, Eigenvalue problems.

AMS subject classifications. 35Q68, 65N35, 65T99,65K10

1. Introduction.

Computing eigenvalues and/or eigenfunctions for elliptic operators or solving boundary value problem of partial differential equations (PDEs) in high dimensions and optimal stochastic control problems are among the key tasks for many scientific computing applications, e.g., ground states and band structure calculation in quantum systems, complex biochemical systems, communications and manufacturing productions, etc. Deep neural network (DNN) has been utilized to solve high dimensional PDEs and stochastic control problems due to its capability of approximating high dimensional functions. The first attempt to use DNN to solve high dimensional quasi-linear parabolic PDE was carried out in [7] using Pardoux-Peng theory [19] of nonlinear Feynman-Kac formulas, which connect the solution of the parabolic PDEs with that of backward stochastic differential equations (SDEs) [14]. The loss function for the training of the DeepBSDE uses the terminal condition of the PDEs. The DeepBSDE framework was also applied to solve stochastic control problems [8]. The DeepBSDE started a new approach of SDE based DNN approximations to high dimensional PDEs and stochastic control problems. The work in [21] [24] extended this idea to use a loss function based on the pathwise comparison of two stochastic processes, one from the BSDE in the Pardoux-Peng theory and one from the PDEs solution, this variant of SDE based DNNs has the potential to find the solution in the whole domain compared with one point solution in the original DeepBSDE. Recently, a diffusion Monte Carlo DNN method was developed [9] using the connection between stochastic processes and the solution of elliptic equations and the backward Kolmogorov equation to build a loss function for eigen-value calculations. The capability of SDE paths exploring high dimensional spaces make them a good candidate to be used for DNN learning. It should be mentioned that other approaches to solving high dimensional PDEs include the Feynmann-Kac formula based Picard iteration [13] [6] and stochastic dimension gradient descent method [12].

*December 20, 2023, first version appeared in arxiv preprint arXiv:2311.09456. 2023 Nov 15.

†Corresponding author, Department of Mathematics, Southern Methodist University, Dallas, TX 75275 (cai@smu.edu)

‡Department of Mathematics, Southern Methodist University, Dallas, TX 75275.

§Department of Mathematics, Southern Methodist University, Dallas, TX 75275.

In addition to the Pardoux-Peng BSDE approach by linking stochastic processes and the solution of PDEs, another powerful probabilistic method is the Varadhan's Martingale problem approach [23, 15], which were used to derive a probabilistic weak form for the PDE's solution with a specific Martingale related to the PDE solution through the Ito formulas [17]. The equivalence of the classic weak solution of boundary value problems (BVPs) of elliptic PDEs using bilinear forms and the probabilistic one were established for the Schrodinger equation for the Neumann BVP [10, 11], and then, for the Robin BVP [18]. This Martingale weak form for the PDEs solution and in fact also for a wide class of stochastic control problem [4, 3] provides a new venue to tackle high dimensional PDE and stochastic control problems [2], and this paper focuses on the case of high dimensional PDEs only. The Martingale formulation of the PDE solution is a result of Ito formula and the Martingale nature of Ito integrals. As a simple conclusion from the broader Martingale property, the Feynman-Kac formula provides one point solution of the PDE using expectation of the underlying SDE paths originating from that point. The Martingale based DNN to be studied in this paper, termed DeepMartNet [2], is trained based on a loss function enforcing the conditional expectation definition of the Martingale. It will be shown with extensive numerical tests that using the same set of SDE paths originating from one single point, the DeepMartNet can in fact provide approximation to solutions of the BVPs and eigenvalue problems of elliptic PDEs over the whole solution doamin in high dimensions. In this sense, the DeepMartNet is able to extract more information for the PDE solutions from the SDE paths originating from one point than the (pre-machine learning) traditional use of the one point solution Feynman-Kac formula.

The rest of this paper is organized as follows. In section 2, a brief review of existing SDE-based DNN methods for solving PDEs is given and Section 3 will present the Martingale problem formulation for the BVP problem of a general elliptic PDE for the case of third kind Robin boundary condition, which includes both Dirichlet and Neumann BVPs as special limiting cases. Section 4 will present the Martingale based DeepMartNet for solving high dimensional PDE problems. Numerical results for the Dirichlet BVPs and eigenvalue problems will be present in Section 5. The implementation of the DeepMartNet for the Neumann and Robin boundary conditions will be addressed in a follow-up paper, which will involve reflecting diffusion processes in finite domains and the computation of local times of the processes. Section 6 will present conclusions and future work.

2. A review of SDE based DNNs for solving PDEs. To set the background for the Martingale based DNNs, we will first briefly review some existing DNN based on diffusion paths from SDEs.

Let us first consider a terminal value problem for quasi-linear elliptic PDEs

$$\partial_t u + \mathcal{L}u = \phi, \quad \mathbf{x} \in R^d, \tag{2.1}$$

with a terminal condition $u(T, \mathbf{x}) = g(\mathbf{x})$, where the differential operator $\mathcal{L}$ is given as

$$\mathcal{L} = \mu^\top \nabla + \frac{1}{2} Tr(\sigma\sigma^\top \nabla\nabla^\top) = \mu^\top \nabla + \frac{1}{2} Tr(A\nabla\nabla^\top), \tag{2.2}$$

and $\mu = \mu(t, \mathbf{x}, u, \nabla u), \sigma = \sigma(t, \mathbf{x}, u, \nabla u)$ and the diffusion coefficient matrix

$$A = (a_{ij})_{d\times d} = \sigma\sigma^\top. \tag{2.3}$$

The aim is to find the solution at $\mathbf{x}, t = 0$, $u(0, \mathbf{x})$, and the solution of (2.1) is

related to a coupled FBSDE [19]

$$
\begin{aligned}
d\mathbf{X}_t &= \mu(t, \mathbf{X}_t, Y_t, \mathbf{Z}_t)dt + \sigma(t, \mathbf{X}_t, Y_t)d\mathbf{B}_t, \\
\mathbf{X}_0 &= \xi,
\end{aligned} \tag{2.4}
$$

$$
\begin{aligned}
dY_t &= \phi(t, \mathbf{X}_t, Y_t, \mathbf{Z}_t)dt + \mathbf{Z}_t^T \sigma(t, \mathbf{X}_t, Y_t)d\mathbf{B}_t, \\
Y_T &= g(\mathbf{X}_T),
\end{aligned} \tag{2.5}
$$

where $\mathbf{X}_t$, Y_t and $\mathbf{Z}_t$ are d, 1 and d-dimensional stochastic processes, respectively, that are adapted to $\{\mathcal{F}_t : 0 \le t \le T\}$ - the natural filtration from the d-dimensional Brownian motion $\mathbf{B}_t$. Specifically, we have the following relations,

$$
Y_t = u(t, \mathbf{X}_t), \quad \mathbf{Z}_t = \nabla u(t, \mathbf{X}_t). \tag{2.6}
$$

DeepBSDE. As the first work of using SDEs to train DNN, the Deep BSDE [7] trains the network with input $\mathbf{X}_0 = \mathbf{x}$ and output $\mathbf{Y}_0 = u(0, \mathbf{x})$. Applying the Euler–Maruyama scheme (EM) to the FBSDE (2.4) and (2.5), respectively, we have

$$
\mathbf{X}_{n+1} \approx \mathbf{X}_n + \mu(t_n, \mathbf{X}_n, \mathbf{Y}_n, \mathbf{Z}_n)\Delta t_n + \sigma(t_n, \mathbf{X}_n, \mathbf{Y}_n)\Delta \mathbf{B}_n, \tag{2.7}
$$

$$
Y_{n+1} \approx Y_n + \phi(t_n, \mathbf{X}_n, Y_n, \mathbf{Z}_n)\Delta t_n + Z_n^T \sigma(t_n, \mathbf{X}_n, Y_n)\Delta \mathbf{B}_n. \tag{2.8}
$$

The missing $\mathbf{Z}_{n+1}$ at t_{n+1} will be then approximated by a neural network (NN) with parameters θ_n

$$
\nabla u(t_n, \mathbf{X}_n|\theta_n) \sim \mathbf{Z}_n = \nabla u(t_n, \mathbf{X}_n). \tag{2.9}
$$

Loss function: with an ensemble average approximation, the loss function is defined as

$$
Loss_{bsde}(Y_0, \theta) = E \left\| u(T, \mathbf{X}_T) - g(\mathbf{X}_T) \right\|^2, \tag{2.10}
$$

where

$$
u(T, \mathbf{X}_T) = Y_N.
$$

Trainable parameters are $\{Y_0, \theta_n, n = 1, \cdots, N\}$.

FBSNN. A forward backward neural network (FBSNNs) proposed in [21] uses the mismatch between two stochastic processes to build the loss function for the DNN, which aims to train a DNN $u_\theta(x, t)$ in the whole domain. The following is an improved version of the approach in [24].

- Markov chain one. Starting with $\mathbf{X}_0 = \mathbf{x}$, $Y_0 = u_\theta(\mathbf{x}, 0)$,

$$
\begin{aligned}
\mathbf{X}_{n+1} &= \mathbf{X}_n + \mu(t_n, \mathbf{X}_n, Y_n, Z_n)\Delta t_n + \sigma(t_n, \mathbf{X}_n, Y_n)\Delta \mathbf{B}_n, \\
Y_{n+1} &= Y_n + \phi(t_n, \mathbf{X}_n, Y_n, \mathbf{Z}_n)\Delta t_n + \mathbf{Z}_n^T \sigma(t_n, \mathbf{X}_n, Y_n)\Delta \mathbf{B}_n, \\
\mathbf{Z}_{n+1} &= \nabla u(t_{n+1}, \mathbf{X}_{n+1}).
\end{aligned} \tag{2.11}
$$

- Markov chain two

$$
Y_{n+1}^\star = u(t_{n+1}, \mathbf{X}_{n+1}). \tag{2.12}
$$

- The loss function is a Monte Carlo approximation of

$$
E\left[\frac{1}{N} \sum_{n=1}^{N} \left\| Y_n - Y_n^\star \right\|^2 + 0.02 \left\| Y_N^\star - g(\mathbf{X}_N) \right\|^2 + 0.02 \left\| \mathbf{Z}_N - \nabla g(\mathbf{X}_N) \right\|^2 \right]. \tag{2.13}
$$

Numerical results of half-order convergence of $u_\theta(\mathbf{x}, t)$, similar to the order of the underlying Euler-Maruyama scheme, is observed.

Diffusion Monte Carlo DNN eigensolver. In another approach similar to the power iteration method, a diffusion-Monte Carlo method was proposed [9] through a fixed point of semi-group formulation for the eigenvalue problem of the following linear elliptic operator (i.e. $\mu = \mu(\mathbf{x}), \sigma = \sigma(\mathbf{x})$),

$$\mathcal{L}\Psi = \lambda\Psi. \tag{2.14}$$

Equation (2.14) can be reformulated as a virtual time dependent backward parabolic PDE with the sought-after eigenfunction as the terminal condition, i.e.,

$$\partial_t u(t, \mathbf{x}) + \mathcal{L}u(t, \mathbf{x}) - \lambda u(t, \mathbf{x}) = 0, \quad u(T, \mathbf{x}) = \Psi(\mathbf{x}). \tag{2.15}$$

Thus, the following fix-point property holds,

$$u(T - t, \cdot) = P_t^\lambda \Psi, \quad P_T^\lambda \Psi = \Psi, \tag{2.16}$$

where the semi-group for the evolutionary system is formally defined by

$$P_t^\lambda = e^{-(T-t)\mathcal{L}}. \tag{2.17}$$

The discretized backward in time evolution (with time step Δt of the backward parabolic equation) mimics the power iteration of the semi-group operator $e^{-n\Delta t\mathcal{L}}$, which will converge to the lowest eigenfunction for $\mathcal{L}$ as in a power method. The loss function is set to be $||P_T^\lambda \Psi - \Psi||^2$, while the evolution of PDE solution is done by two SDEs, instead of solving the parabolic equation directly,

$$\begin{aligned} \mathbf{X}_{n+1} =& \mathbf{X}_n + \sigma\Delta\mathbf{B}_n, \\ u_{n+1} =& u_n + (\lambda\Psi_\theta - \mu^T\nabla\Psi_\theta)(\mathbf{X}_n)\Delta t + \nabla\Psi_\theta(\mathbf{X}_n)\Delta\mathbf{B}_n. \end{aligned} \tag{2.18}$$

The original algorithm in [9] uses a separate DNN to approximate the gradient of $\Psi(\mathbf{x})$. And the loss function for the DNN $\Psi_\theta(\mathbf{x})$ approximating the eigen-function and the eigenvalue is then defined by

$$Loss_{semigroup}(\theta, \lambda) = E_{X_0\sim\pi_0}[|u_N - \Psi_\theta(\mathbf{X}_N)|^2]. \tag{2.19}$$

3. Martingale problem formulation of elliptic PDEs. In this section, we will present the Martingale problem formulation for the BVPs and eigenvalue problems of elliptic PDEs, and for this purpose, let us consider a general PDE for the linear elliptic operator $\mathcal{L}$ with $\mu = \mu(\mathbf{x}), \sigma = \sigma(\mathbf{x})$,

$$\mathcal{L}u + V(\mathbf{x}, u, \nabla u) = f(\mathbf{x}, u), \quad \mathbf{x} \in D \subset R^d, \tag{3.1}$$

$$\text{or} \quad \mathcal{L}u = f(\mathbf{x}, u) - V(\mathbf{x}, u, \nabla u), \tag{3.2}$$

$$\Gamma(u) = g, \quad \mathbf{x} \in \partial D,$$

where $f(\mathbf{x}, u) = \lambda u, g = 0$ for the case of an eigenvalue problem with an eigenvalue λ and eigenfunction $u(x)$, and the boundary operator Γ could be Dirichlet, Neumann, or Robin type or a decay condition will be given at ∞ if $D = R^d$. The following shorthand will be used in the rest of the paper

$$v(\mathbf{x}) = V(\mathbf{x}, u(\mathbf{x}), \nabla u(\mathbf{x})). \tag{3.3}$$

The vector $\mu = \mu(\mathbf{x})$, $\sigma_{d\times d} = \sigma_{d\times d}(\mathbf{x})$ can be associated with the variable drift and diffusion, respectively, of the following stochastic Ito process $\mathbf{X}_t(\omega) \in R^d$, $\omega \in \Omega$ (random sample space) with $\mathcal{L}$ as its generator,

$$d\mathbf{X}_t = \mu(\mathbf{X}_t)dt + \sigma(\mathbf{X}_t)\cdot d\mathbf{B}_t \qquad \mathbf{X}_t = \mathbf{x}_0 \in D, \tag{3.4}$$

where $\mathbf{B}_t = (B_t^1, \cdots, B_t^d)^\top \in R^d$ is the Brownian motion in R^d.

The transition probability $P(\mathbf{y}, t; \mathbf{x}, s)$ for the process $\mathbf{X}_t$ will satisfy the following Fokker-Planck equation

$$\frac{\partial P(\mathbf{y}, t; \mathbf{x}, s)}{\partial t} = \mathcal{L}_y^* P(\mathbf{y}, t; \mathbf{x}, s), \tag{3.5}$$

where the adjoint operator

$$\mathcal{L}_y^* = -\nabla_y^\top \mu + \frac{1}{2} Tr(\nabla_y \nabla_y^\top A). \tag{3.6}$$

(**Robin Problem**) Let us consider the BVP of (3.1) with a Robin type boundary condition

$$\Gamma(u) = \gamma^\top \cdot \nabla u + cu = g, \tag{3.7}$$

where the vector

$$\gamma(x) = \frac{1}{2} A \cdot \mathbf{n}, \tag{3.8}$$

and $\mathbf{n}$ is the outward normal at $\mathbf{x} \in \partial D$. The Dirichlet BC $u = f$ can be considered as a limiting case of $c \to N, g \to Ng, N \to \infty$ and the Neumann BC as the case $c = 0$.

With the Martingale problem approach [23], the Martingale problem for the BVP with the third kind boundary condition (3.7) can formulated using a reflecting diffusion process $\mathbf{X}^{ref}$ based on the process $\mathbf{X}$ (3.4) through the following Skorohod problem.

(**Skorohod problem**): Assume D is a bounded domain in R^d with a C^2 boundary. Let $\mathbf{X}(t)$ be a (continuous) path of (3.4) in R^d with $\mathbf{X}(0) \in \bar{D}$. A pair $(\mathbf{X}^{ref}(t), L(t))$ is a solution to the Skorohod problem $S(\mathbf{X}; D)$ if the following conditions are satisfied:

1. $\mathbf{X}^{ref}$ is a path in $\bar{D}$;
2. (**local time**) $L(t)$ is a non-decreasing function which increases only when $X^{ref} \in \partial D$, namely,

$$L(t) = \int_0^t I_{\partial D}(\mathbf{X}^{ref}(s)) L(ds), \tag{3.9}$$

3. The Skorohod equation holds:

$$S(\mathbf{X}; D): \qquad \mathbf{X}^{ref}(t) = X(t) - \int_0^t \gamma(\mathbf{X}^{ref}(s)) L(ds). \tag{3.10}$$

Here, $L(t)$ is the local time of the reflecting diffusion process, where an oblique reflection with respect to the direction γ at the boundary ∂D occurs once the process $\mathbf{X}(t)$ hits the boundary [22]. The sampling of reflecting process and the computation of local time $L(t)$ can be found in [5].

As we will only use the reflecting diffusion $\mathbf{X}^{ref}(t)$ for the rest of our discussion, we will keep the same notation

$$\mathbf{X}(t) \leftarrow \mathbf{X}^{ref}(t) \tag{3.11}$$

with the understanding that it now stands for a reflecting diffusion process within the closed domain $\bar{D}$. Using the Ito formula [17] for the semi-martingale $\mathbf{X}(t)$ (namely, $\mathbf{X}^{ref}(t)$) [10] [18],

$$du(\mathbf{X}(t)) = \sum_{i=1}^{d} \frac{\partial u}{\partial x_i}(\mathbf{X}(t))dX_i(t) + \frac{1}{2}\sum_{i=1}^{d}\sum_{j=1}^{d} a_{ij}(\mathbf{X}(t))\frac{\partial^2 u}{\partial x_i \partial x_j}(\mathbf{X}(t))dt,$$

with the notation of the generator $\mathcal{L}$, we have for the solution $u(\mathbf{x})$ of (3.1) the following differential

$$\begin{aligned}
&du(\mathbf{X}(t)) = \\
&\mathcal{L}u(\mathbf{X}(t))dt - \gamma^{\intercal}\cdot\nabla u(u(\mathbf{X}_t))L(dt) + \sum_{i=1}^{d}\sum_{j=1}^{d}\sigma_{ij}\frac{\partial u}{\partial x_i}(\mathbf{X}(t))dB_i(t) \\
&= \left[f(\mathbf{X}(t), u(\mathbf{X}(t)) - V(\mathbf{X}(t), u(\mathbf{X}(t)), \nabla u(\mathbf{X}(t)))\right]dt - \left[g(\mathbf{X}(t)) - cu(\mathbf{X}(t))\right]L(dt) \\
&+ \sum_{i=1}^{d}\sum_{j=1}^{d}\sigma_{ij}\frac{\partial u}{\partial x_i}(\mathbf{X}(t))dB_i(t),
\end{aligned}$$

which in turns gives a Martingale M_t^u defined by

$$M_t^u \doteq \tag{3.12}$$

$$\begin{aligned}
&u(\mathbf{X}_t) - u(\mathbf{X}_0) - \int_0^t \left[f(\mathbf{X}_s, u(\mathbf{X}_s)) - V(\mathbf{X}_s, u(\mathbf{X}_s), \nabla u(\mathbf{X}_s))\right]ds \\
&+ \int_0^t \left[g(\mathbf{X}_s) - cu(\mathbf{X}_s)\right]L(ds) = \int_0^t \sum_{i=1}^{d}\sum_{j=1}^{d}\sigma_{ij}\frac{\partial u}{\partial x_i}(\mathbf{X}_s)dB_i(s),
\end{aligned} \tag{3.13}$$

due to the Martingale nature of the Ito integral at the end of the equation above.

(**Dirichlet Problem**) For Dirichlet problem of (3.1) with a boundary condition

$$\Gamma[u] = u = g, \quad \mathbf{x} \in \partial D, \tag{3.14}$$

the underlying diffusion process is the original diffusion process (3.1), but killed at the boundary at the first exit time

$$\tau_D = \inf\{t, \mathbf{X}_t \in \partial D\}, \tag{3.15}$$

and it can be shown that in fact

$$\tau_D = \inf\{t > 0, L(t) > 0\}, \tag{3.16}$$

and also that $M^u_{t\wedge\tau_D}$ will be still a Martingale [17], which will not involve the integral with respect to local time $L(t)$, i.e.

(3.17)
$$M^u_{t\wedge\tau_D} = u(\mathbf{X}_{t\wedge\tau_D}) - u(\mathbf{X}_0) - \int_0^{t\wedge\tau_D} [f(\mathbf{X}_s, u(\mathbf{X}_s)) - V(\mathbf{X}_s, u(\mathbf{X}_s), \nabla u(\mathbf{X}_s))]\, ds.$$

For the case of a linear PDE, i.e. $f(\mathbf{x}, u) = f(\mathbf{x}), V = 0$, by taking expectation of (3.17) and letting $t \to \infty$, we will have

$$\begin{aligned} 0 = \quad & E[M^u_0] = lim_{t\to\infty} E[M^u_{t\wedge\tau_D}] = E[M^u_{\tau_D}] \\ = \quad & E[u(\mathbf{X}_{\tau_D}) - u(\mathbf{X}_0)] - \textstyle\int_0^{\tau_D} f(\mathbf{X}_s) ds \\ = \quad & E[g(\mathbf{X}_{\tau_D})] - u(\mathbf{x}) - E[\textstyle\int_0^{\tau_D} f(\mathbf{X}_s) ds], \end{aligned} \tag{3.18}$$

resulting in the well-known Feynman-Kac formuula for the Dirichlet boundary value problem

$$u(\mathbf{x}) = E[g(\mathbf{X}_{\tau_D})] - E[\int_0^{\tau_D} f(\mathbf{X}_s) ds], \quad \mathbf{x} \in D, \tag{3.19}$$

where the diffusion process $\mathbf{X}_t, \mathbf{X}_0 = \mathbf{x}$ is defined by (3.4).

The Martingale problem of the BVPs states the equivalence of M^u_t being a Martingale (i.e. a probabilistic weak form of the BVPs) and the classic weak form: For every test function $\phi(\mathbf{x}) \in C^2_{\partial D} = \{\phi : \phi \in C^2(D) \cap C^1(\overline{D}), (\gamma' \cdot \nabla - \mu^\intercal \mathbf{n})\, \phi = 0\}$, $\gamma' = \gamma - \alpha$, $\alpha_j = \sum_{i=1}^{3} \frac{\partial a_{ij}}{\partial x_i}$, we have

$$\begin{aligned} \int_D u(\mathbf{x}) \mathcal{L}^* \phi d\mathbf{x} = & \int_D [f(\mathbf{x}, u(\mathbf{x})) - V(\mathbf{x}, u(\mathbf{x}), \nabla u(\mathbf{x}))]\, \phi(\mathbf{x}) d\mathbf{x} \\ & + \int_{\partial D} \phi(\mathbf{x})[g(\mathbf{x}) - cu(\mathbf{x})] ds_x, \end{aligned} \tag{3.20}$$

where

$$\mathcal{L}^* \phi = \frac{1}{2} Tr(\nabla \nabla^\intercal A)\phi - div(\mu\phi). \tag{3.21}$$

This equivalence has been proven for the Schrodinger operator $\mathcal{L}u = \frac{1}{2}\Delta u + qu$ for the Neumann problem [10] and the Robin problem [18].

4. DeepMartNet - a Martingale based neural network. In this section, we will propose a DNN method for solving the BVPs and eigenvalue problems for elliptic PDEs using the equivalence between its Martingale problem formulation and classic weak form of the PDEs.

For simplicity of our discussion, let us assume that $s \le t \le \tau_D$, by the Martingale property of $M_t \equiv M^u_t$ of (3.17), we have

$$E[M_t | \mathcal{F}_s] = M_s, \tag{4.1}$$

which implies for any measurable set $A \in \mathcal{F}_s$,

$$E[M_t | A] = M_s = E[M_s | A], \tag{4.2}$$

thus,

$$E[(M_t - M_s)\,|A] = 0, \tag{4.3}$$

i,e,

$$\int_A (M_t - M_s)\, P(d\omega) = 0, \tag{4.4}$$

where

$$\begin{aligned} M_t - M_s =& u(\mathbf{X}_t) - u(\mathbf{X}_s) - \int_s^t \mathcal{L}u(\mathbf{X}_z)dz \\ =& u(\mathbf{X}_t) - u(\mathbf{X}_s) - \int_s^t (f(z, u(\mathbf{X}_z)) - v(\mathbf{X}_z))dz. \end{aligned} \tag{4.5}$$

In particular, if we take $A = \Omega \in \mathcal{F}_s$ in (4.3), we have

$$E[M_t - M_s] = 0, \tag{4.6}$$

i.e. the Martingale M_t has a constant expectation. However, it should be noted that constant expectation by itself does not mean a Martingale, for this we have the following lemma [3].

LEMMA 4.1. *If $E[M_S] = E[M_T]$ holds for any two stopping time $S \leq T$, then $M_t, t \geq 0$ is a Martingale, i.e. $E[M_t|\mathcal{F}_s] = M_s$ for $s \leq t$.*

For a given time interval $[0, T]$, we define a partition

$$0 = t_0 < t_1 < \cdots < t_i < t_{i+1} < \cdots < t_N = T, \tag{4.7}$$

and the increment of the M_t over $[t_i, t_{i+k}]$ can be approximated by using a trapezoidal rule for the integral term

$$\begin{aligned} M_{t_{i+k}} - M_{t_i} =& u(\mathbf{X}_{i+k}) - u(\mathbf{X}_i) - \int_{t_i}^{t_{i+k}} \mathcal{L}u(\mathbf{X}_z)dz \\ \doteq& u(\mathbf{X}_{i+k}) - u(\mathbf{X}_i) - \Delta t \sum_{l=0}^{k} \omega_l \mathcal{L}u(\mathbf{X}_{i+l}) \\ =& u(\mathbf{X}_{i+k}) - u(\mathbf{X}_i) - \Delta t \sum_{l=0}^{k} \omega_l (f(\mathbf{X}_{i+l}, u(\mathbf{X}_{i+l})) - v(\mathbf{X}_{i+l})), \end{aligned} \tag{4.8}$$

where for $k \geq 1$ $\omega_0 = \omega_k = \frac{1}{2}$, $\quad \omega_l = 1, \quad 2 \leq l \leq k-1$ and for $k = 0$, $\omega_0 = 1$.

Adding back the exit time τ_D, we note that

$$M_{t_{i+k}\wedge\tau_D} - M_{t_i\wedge\tau_D} = u(\mathbf{X}_{t_{i+k}\wedge\tau_D}) - u(\mathbf{X}_{t_i\wedge\tau_D}) - \int_{t_i\wedge\tau_D}^{t_{i+k}\wedge\tau_D} \mathcal{L}u(\mathbf{X}_z)dz = 0$$

if both $t_{i+k}, t_i \geq \tau_D$.

Remark 4.2. We could define a different generator $\mathcal{L}$ by not including $\mu^\top \nabla$ in (2.2), then the Martingale in (4.9) will be changed to

$$M_t^* = u(\mathbf{X}_t) - u(\mathbf{x}_0) - \int_0^t (\lambda u(\mathbf{X}_s) - \mu^\top(\mathbf{X}_s)\nabla u(\mathbf{X}_s) - v(\mathbf{X}_s))ds, \tag{4.9}$$

where the process $\mathbf{X}_t$ is given simply by $d\mathbf{X}_t = \sigma \cdot d\mathbf{B}_t$, instead.

- **DeepMartNet for Dirichlet BVPs**

Let $u_\theta(\mathbf{x})$ be a neural network approximating the BVP solution with θ denoting all the weight and bias parameters of a DNN. For a given time interval $[0, T]$, we define a partition

$$0 = t_0 < t_1 < \cdots < t_i < t_{i+1} < \cdots < t_N = T,$$

and M-discrete realizations

$$\Omega' = \{\omega_m\}_{m=1}^M \subset \Omega \tag{4.10}$$

of the Ito process using the Euler-Maruyama scheme with M-realizations of the Brownian motions $\mathbf{B}_i^{(m)}$, $0 \le m \le M$,

$$\mathbf{X}_i^{(m)}(\omega_m) \sim X(t_i, \omega_m), 0 \le i \le N,$$

where

$$\mathbf{X}_{i+1}^{(m)} = \mathbf{X}_i^{(m)} + \mu(\mathbf{X}_i^{(m)})\Delta t_i + \sigma(\mathbf{X}_{\mathbf{i}}^{(\mathbf{m})}) \cdot \Delta \mathbf{B}_i^{(m)}, \tag{4.11}$$

$$\mathbf{X}_0^{(m)} = \mathbf{x}_0 \tag{4.12}$$

where $\Delta t_i = t_{i+1} - t_i$,

$$\Delta \mathbf{B}_i^{(m)} = \mathbf{B}_{i+1}^{(m)} - \mathbf{B}_i^{(m)}.$$

We will build the loss function $Loss(\theta)$ for neural network $u_\theta(\mathbf{x})$ approximation of the BVP solution using the Martingale property (4.4) and the M-realization of the Ito diffusion (3.4).

For each t_i, we randomly take a subset of $A_i \subset \Omega'$ with a uniform sampling (without replacement), corresponding to the mini-batch in computing the stochastic gradient for the empirical training loss. Assuming that the mini-batch in each A_i is large enough such that $\{\mathbf{X}_{i+1}^{(m)}\}$ and $\{\mathbf{X}_i^{(m)}\}$, $\omega_m \in A_i$ sample the distribution $P(t_{i+1}, ., t_i, A), P(t_i, ., 0, x_0)$ well, respectively, then equation (4.22) gives the following approximate identity for the solution $u_\theta(\mathbf{X}_t)$ using the A_i-ensemble average,

As $E[M_{t_{i+k}}^{u_\theta} - M_{t_i}^{u_\theta}] \approx 0$, for a randomly selected $A_i \in \Omega' = \mathcal{F}_{t_i}$ (mini-batches), we can set

$$\begin{aligned} Loss_{mart}(\theta) &= \frac{1}{N}\sum_{i=0}^{N-1}(M_{t_{i+k}\wedge\tau_D}^{u_\theta} - M_{t_i\wedge\tau_D}^{u_\theta})^2 \\ &= \frac{1}{N}\sum_{i=0}^{N-1}\frac{I(t_i \le \tau_D)}{|A_i|^2}\left(\sum_{m=1}^{|A_i|} u_\theta(\mathbf{X}_{i+k}^{(m)}) - u_\theta(\mathbf{X}_i^{(m)}) - \right. \\ &\left. \Delta t\sum_{l=0}^{k}\omega_l(f(\mathbf{X}_{i+l}^{(m)}, u_\theta(\mathbf{X}_{i+l}^{(m)})) - v_\theta(\mathbf{X}_{i+l}^{(m)}))\right)^2, \end{aligned} \tag{4.13}$$

where $v_\theta(\mathbf{x})$ is defined similarly as in (3.3). Refer to Remark 4.4 for the discussion of the size of the mini-batch $|A_i|$.

Now, we define the total loss for the boundary value problem as

$$Loss_{total-bvp}(\theta) = Loss_{mart}(\theta) + \alpha_{bdry} Loss_{bdry}(\theta), \tag{4.14}$$

and α_{bdry} is a hyper-parameter and $Loss_{bdry}(\theta) = ||u_\theta(\mathbf{x}) - g||_2^2$, which can be approximated by evaluations at the boundary.

DeepMartNet solution- $u_{\theta^*}(x)$, here

$$\theta^* = argmin Loss_{total-bvp}(\theta). \tag{4.15}$$

- **DeepMartNet for Dirichlet eigenvalue problems**

For the eigenvalue problem

$$\begin{aligned} \mathcal{L}u + V(\mathbf{x}, u, \nabla u) = \lambda u, \quad \mathbf{x} \in D \subset R^d, \\ \Gamma(u) = u = 0, \quad \mathbf{x} \in \partial D, \end{aligned} \tag{4.16}$$

and the Martingale loss becomes

$$\begin{aligned} Loss_{mart}(\lambda, \theta) = \frac{1}{N} \sum_{i=0}^{N-1} \frac{1}{|A_i|^2} \Bigg(\sum_{m=1}^{|A_i|} u_\theta(\mathbf{X}_{i+k}^{(m)}) - u_\theta(\mathbf{X}_i^{(m)}) - \\ \Delta t \sum_{l=0}^{k} \omega_l (\lambda u_\theta(\mathbf{X}_{i+l}^{(m)}) - v_\theta(\mathbf{X}_{i+l}^{(m)})) \Bigg)^2 \end{aligned} \tag{4.17}$$

and in the case of a bounded domain, the boundary loss $Loss_{bdry}(\theta)$ will be added for the homogeneous boundary condition $g = 0$, i.e., $Loss_{bdry}(\theta) = ||u_\theta(\mathbf{x})||_2^2$. For the decay condition at infinite, a molifier will be used to enforce explicitly the decay condition there (see (5.22)).

Also, in order to prevent the DNN eigenfunction going to a zero solution, we introduce a simple normalization term using l_p (p=1, 2) norm of the solution at some randomly selected location

$$Loss_{normal}(\theta) = \left(\frac{1}{m} \sum_{i=1}^{m} |u_\theta(\mathbf{x}_i)|^p - c \right)^2, \tag{4.18}$$

where $\mathbf{x}_i$ are m arbitrarily selected fixed points and c is a nonzero constant.

Finally, we have the total loss for the eigenvalue problem as

$$Loss_{total-eig}(\lambda, \theta) = Loss_{mart}(\lambda, \theta) + \alpha_{bdry} Loss_{bdry}(\theta) + \alpha_{normal} Loss_{normal}(\theta), \tag{4.19}$$

where α_{bdry} and α_{normal} are hyper-parameters.

DeepMartNet eigen-problem solution - $(\lambda^*, u_{\theta^*}(x))$, here

$$(\lambda^*, \theta^*) = argmin Loss_{total-eig}(\lambda, \theta). \tag{4.20}$$

Remark 4.3. (**Mini-batch in SGD training and Martingale property**) Due to the equivalence between (4.4) and (4.1), the loss function defined above ensures that $M_t^{u_\theta}$ of (3.17) for $u_\theta(\mathbf{x})$ will be a Martingale approximately if the mini-batch A_i explores all subsets of the sample space Ω' during the SGD optimization process of the training, and the sample size $M = |\Omega'| \to \infty$, the time step $\max |\Delta t_i| \to 0$, and the training converges (see Fig. 4.1).

Also, if we take $A_i = \Omega'$ for all i, there will be no stochasticity in the gradient calculation for the loss function, we will have a traditional full gradient descent method and the full Martingale property for $u_\theta(\mathbf{x})$ is not enforced either. Therefore, the mini-batch procedure in DNN SGD optimization corresponds perfectly with the Martingale definition (4.1).

In summary, the Martingale property implies that for any measurable set $A \in \mathcal{F}_s$, we require that

$$E[M_t|A] = M_s, \tag{4.21}$$

which then provides a native mechanism for the mini-batch in the SGD. Therefore, the Martingale based DNN is an ideal fit for deep learning of high-dimensional PDEs.

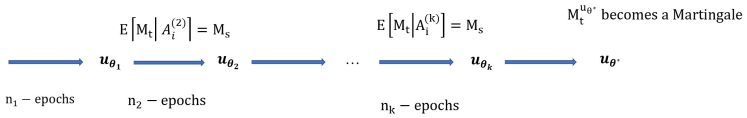

Fig. 4.1: DeepMartNet training and Martingale property

Remark 4.4. (**Size of mini-batch** A_i) The loss function of the DeepMartNet is based on the fact that $\int_{A_i} (M_t^u - M_s^u)\, P(d\omega) = 0$ for the exact solution $u(\mathbf{x})$, where the expectation will be computed by an ensemble average of selected paths from the M-paths. In theory, using the transitional probability, using left end point quadrature for the integral over [s,t] in (4.5) with $|t-s| << 1$, (4.4) can be rewritten as

$$\begin{aligned}
&0 = E[(M_t^u - M_s^u)\,|A_i] \\
&\doteq \int_{\mathbf{z}\in B}\int_{\mathbf{y}\in R^d} \big[u(\mathbf{y}) - u(\mathbf{z}) - (f(\mathbf{z},u(\mathbf{z})) - v(\mathbf{z}))(t-s)\big] P(t,\mathbf{y},s,\mathbf{z}) p(s,\mathbf{z},0,\mathbf{x}_0) dz dy \\
&= \int_{\mathbf{y}\in R^d} u(\mathbf{y}) P(t,\mathbf{y},s,A_i) dy - \int_{\mathbf{z}\in B} \big[u(\mathbf{z}) + (f(\mathbf{z},u(\mathbf{z})) - v(\mathbf{z}))(t-s)\big] P(s,\mathbf{z},0,\mathbf{x}_0) dz \\
&\sim \frac{1}{|A_i|}\sum_{m=1}^{|A_i|} \Big(u(\mathbf{X}_t^{(m)}) - \big[u(\mathbf{X}_s^{(m)}) + (f(\mathbf{X}_s^{(m)}, u(\mathbf{X}_s^{(m)})) - v(\mathbf{X}_s^{(m)}))(t-s)\big] \Big),
\end{aligned} \tag{4.22}$$

where $B = \mathbf{X}_s^{-1}(A_i)$ and $\mathbf{X}_t^{(m)}, 1 \le m \le M$ are the M-sample paths of the diffusion process $\mathbf{X}_t$.

Therefore, the size of mini-batch $|A_i|$ should be large enough to give an accurate sampling of the continuous distribution $P(t,\mathbf{y},s,A), y \in R^d$ and $P(s,\mathbf{z},0,\mathbf{x}_0), z \in B$, so in our simulation we select a sufficient large M and set the size of mini-batch A_i in the following range,

$$M/m_1 \le |A_i| \le M/m_2, \tag{4.23}$$

where M is the total number of paths used and $m_1 > m_2$ are hyper-parameters of the training.

The set A_i can be the same for each $0 \le i \le N-1$ per epoch or can be randomly selected as the subset of Ω' depending how much stochasticity is put into the calculation of the stochastic gradient in the SGD optimizations.

For a low memory implementation of the DeepMartNet, at any time we can just generate enough $|A_i|$ number of paths to be used for some epochs of training, and then regenerate them again without first generating a large number of paths upfront.

5. Numerical Results. The numerical parameters for the DeepMartNet consist of

- M - number of total paths
- T - terminal time of the paths
- N - number of time steps over [0, t]
- $\Delta t = T/N$
- $M_b = |A_i|$ size of mini-batches of paths selected according to (4.23).
- Size of networks

The training is carried out on a Nvidia Superpod one GPU node A100. The Optmizer is Adamax [16].

5.1. Dirichet BVPs of the Poisson-Boltzmann equation . We will first apply the DeepMartNet to solve the Dirichlet BVP of the Poisson-Boltzmann equation (PBE) arising from solvation of biomolecules in inoic solvents [1].

$$\begin{cases} \Delta u(\mathbf{x}) + cu(\mathbf{x}) = f(\mathbf{x}), & \mathbf{x} \in D \\ u(\mathbf{x}) = g(\mathbf{x}), & \mathbf{x} \in \partial D \end{cases} \tag{5.1}$$

where $c < 0$ ($c = -1$ in the numerical tests) with an high-dimensional solution given by

$$u(\mathbf{x}) = \sum_{i=1}^{d} \cos(\omega x_i), \quad \omega = 2. \tag{5.2}$$

In this case, the generator for the stochastic process is $\mathcal{L} = \frac{1}{2}\Delta$, so the corresponding diffusion is simply the Brownian motion $\mathbf{B}(t)$. For the $M-$ Brownian paths $\mathbf{B}^{(j)}, j = 1, \cdots, M$ originating from x_0, the Martingale loss (4.13) becomes

$$\begin{aligned} Loss_{\text{mart}}(\theta) :=& \frac{1}{N} \sum_{i=0}^{N-1} \frac{1}{|A_i|^2} \left(\sum_{j=1}^{|A_i|} u_\theta(\mathbf{B}_{t_{i+1}}^{(j)}) - u_\theta(\mathbf{B}_{t_i}^{(j)}) \right. \\ & \left. - \frac{1}{2} \left(f(\mathbf{B}_{t_i}^{(j)}) - cu(\mathbf{B}_{t_i}^{(j)}) \right) \mathbb{I}(t_i \le \tau_D^{(j)}) \Delta t \right)^2. \end{aligned} \tag{5.3}$$

Meanwhile, we could use the Feynman-Kac formula [17] to compute the solution at the point $\mathbf{x}_0$ by

$$u(\mathbf{x}_0) \approx \frac{1}{M} \sum_{j=1}^{M} \left(g\left(\mathbf{B}_{\tau_D}^{(j)}\right) \mathrm{e}^{\frac{c\tau_D}{2}} + \frac{1}{2} \sum_{i=0}^{N-1} f\left(\mathbf{B}_{t_i}^{(j)}\right) \mathrm{e}^{\frac{ct_i}{2}} \mathbb{I}\left(t_i \le \tau_D^{(j)}\right) \Delta t \right), \tag{5.4}$$

and also define a point solution loss (or an integral identity for the solution for PDE with quasi-linear term V and f in (3.1)), termed Feynman-Kac loss, which is added to the total loss (4.14)

$$Loss_{\text{F-K}}(\theta) = (u_\theta(\mathbf{x}_0) - u(\mathbf{x}_0))^2, \tag{5.5}$$

and a boundary loss is approximated as

$$Loss_{bdry}(\theta) = ||u_\theta - g||_2^2 \approx \frac{1}{N_{bdry}} \sum_{k=1}^{N_{bdry}} |u_\theta(\mathbf{x}_k) - g(\mathbf{x}_k)|^2, \tag{5.6}$$

where uniformly sampled boundary points $\mathbf{x}_k, 1 \le k \le N_{bd}$ are used to compute the boundary integral. Therefore, the total loss for the boundary value problem of the PB equation is

$$Loss_{bvp}(\theta) = Loss_{\text{mart}}(\theta) + \alpha_{\text{F-K}} Loss_{\text{F-K}}(\theta) + \alpha_{bdry} Loss_{bdry}(\theta), \tag{5.7}$$

where the penalty parameter $\alpha_{\text{F-K}}$ ranges from 10 to 1000 and α_{bdry} ranges from 1000 to 10,000. An Adamax optimizer with learning rate 0.05 is applied for training and $\alpha_{\text{F-K}} = 10, \alpha_{\text{bdry}} = 10^3$ are taken for the following numerical tests.

- **Test 1: PBE in a d=20 dimensional cube** $[-1,1]^d$**.** In this test, we solve the PBE in a 20 dimensional cube with the Dirichlet boundary condition given by the exact solution (5.2). The total number of paths is $M = 100,000$ starting from origin, and mini-batch size $M_b = |A_i| = 1000; \Delta t = 0.01$ and T=9. A fully connected network with layers (20, 64, 32, 1) with first hidden layer Tanh and second hidden layer GeLU activation function is used. A mini-batch of $N_{\text{bdry}} = 2000$ points $\mathbf{x}_k$ are uniformly sampled points on the boundary for every epoch of training.
  Fig. 5.1 show the learned solution along diagonal of the cube (top left) as well as along the first coordinate (top right) and the history of the loss (bottom left) and relative error L2 (computed by Monte Carlo sample) (bottom right). The training takes about less than 5 minutes.

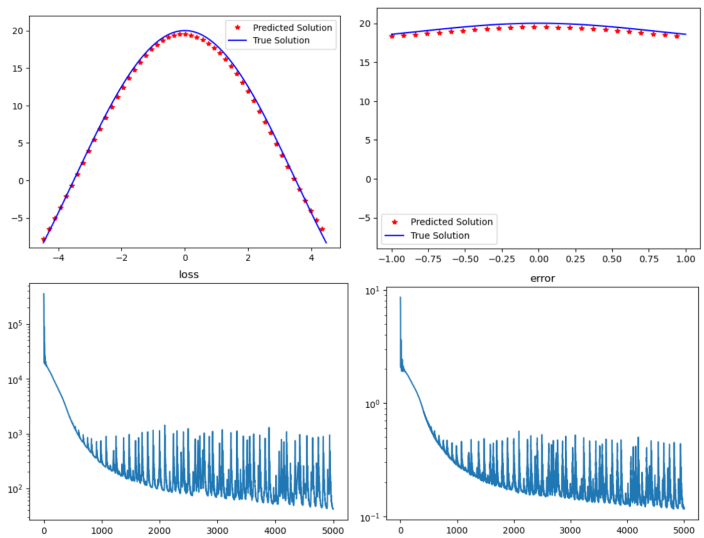

Fig. 5.1: DeepMartNet solution of PBE in $D = [-1,1]^d, d = 20$. (Upper left): true and predicted value of u along the diagonal of the unite cube; (Upper right): true and predicted value of u along the first coordinate axis of $\mathbb{R}^d$. (Lower left): The loss L history; (lower right): The history of relative error L^2 over the cube.

- **Test 2. Effect of path starting point $\mathbf{x}_0$.** In previous test, the DeepMartNet uses all diffusion paths originating from a fixed point $\mathbf{x}_0$ to explore the solution domain. In this test, we consider paths starting from different $\mathbf{x}_0 = (l, 0, \cdots, 0), l = 0.1, 0.3, 0.7$ to investigate the effect of different starting point $\mathbf{x}_0$ on the accuracy of the DeepMartNet. A fully connected network with layers (20, 64, 32, 1) with first hidden layer Tanh and second hidden layer GeLU activation function will be used as in Test 1. The total number of paths $M = 100,000$, and for each epoch, we randomly choose $M_b = 1000$ from the paths; the time step of the paths is $\Delta t = 0.01$.
Fig. 5.2 shows that the three different choices of $\mathbf{x}_0$ produce similar numerical results with the same numerical parameters as in Test 1.

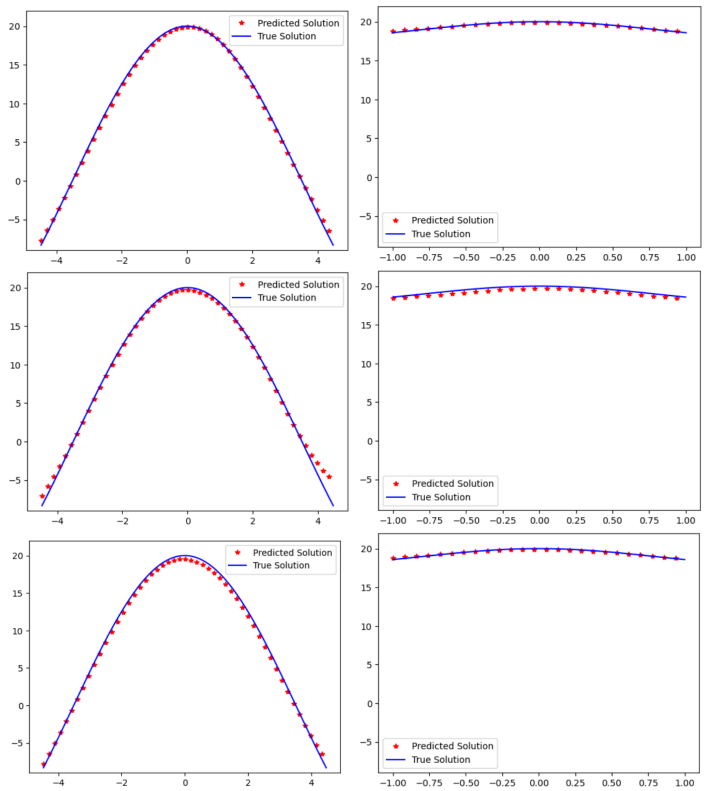

Fig. 5.2: DeepMartNet for PBE in $D = [-1, 1]^d, d = 20$ with 3 different starting points for the paths. From up to down: $\mathbf{x}_0 = (l, 0, \cdots, 0), l = 0.1, 0.3, 0.7$. (left):$u(x\boldsymbol{e}_1)$ where $\boldsymbol{e}_1 = d^{-1/2}(1, 1 \cdots, 1)$,(Right): $u(x\boldsymbol{e})$ where $\boldsymbol{e} = (1, 0 \cdots, 0)$.

Moreover, the DeepMartNet can use diffusion paths starting from different initial position $\mathbf{x}_0$ in training the DNN as long as the mini-batch of the paths A_i for a given epoch corresponds to paths originating from a common initial point $\mathbf{x}_0$. In Fig. 5.3, we compare the numerical result using 120,000 total paths starting from $\mathbf{x}_0 = (0.3, 0, \cdots, 0, 0)$ with that with three choices of $\mathbf{x}_0$ as in Fig. 5.2 with 40,000 paths for each $\mathbf{x}_0$. In both cases, for each epoch, we randomly choose mini-batch size $M_b = 1000$ from the paths; the time step of the paths is $\Delta t = 0.01$. An Adamax optimizer with learning rate 0.05 is

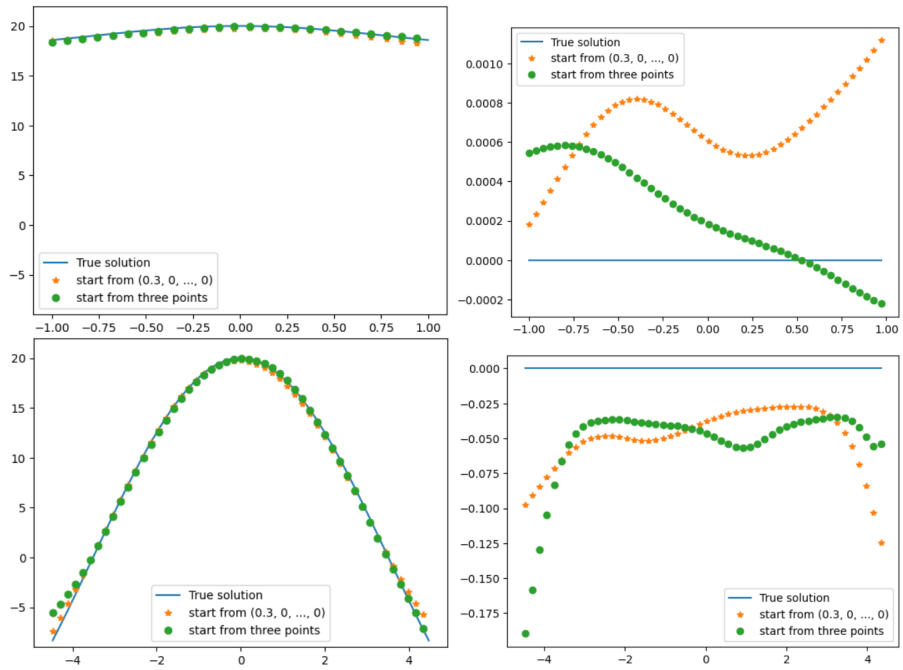

Fig. 5.3: Comparison of DeepMartNet with paths from one starting point and from 3 starting points with same number of total paths in both cases. Upper left: $u(x\boldsymbol{e})$ where $\boldsymbol{e} = (1, 0 \cdots, 0)$; Upper right: the relative error $|u - u_{\text{true}}|/\|u_{\text{true}}\|_\infty$ for the upper left plot; Lower left: $u(x\boldsymbol{e}_1)$ where $\boldsymbol{e}_1 = d^{-1/2}(1, 1 \cdots, 1)$; Lower right: the error $\|u_{\text{true}}\|_\infty$ for the lower left plot.

applied for training. $\alpha_{\text{F-K}} = 10, \alpha_{\text{bdry}} = 10^3$. The DeepMartNet produce similar results for these two cases with same other numerical parameters as in Test 1.

- **Test 3: PBE in a d=100 dimensional unit ball.** In this test, we solve the PBE in a 100 dimensional space, We set $T = 0.25$ and the time step of the paths is $\Delta t = 0.005$, and the total number of paths is $M = 100,000$, and for each epoch, we randomly choose $M_b = 1000$ from the paths. A fully connected network with layers (100, 128, 32, 1) with first hidden layer Tanh and second hidden layer GeLU activation function is used.
  Fig. 5.4 show the learned solution (Upper left): true and predicted value of u at the diagonal of $\mathbb{R}^d$, i.e. x versus $u(x\boldsymbol{e})$ where $\boldsymbol{e} = d^{-1/2}(1, 1 \cdots, 1)$; (Upper right): true and predicted value of u at the first coordinate axis of $\mathbb{R}^d$, i.e. x versus $u(x\boldsymbol{e_1})$ where $\boldsymbol{e} = d^{-1/2}(1, 0 \cdots, 0)$. (Middle left): $u(x\boldsymbol{e}')$ where $\boldsymbol{e}' = 2^{-1/2}(1, 1, 0, 0, \cdots, 0)$ (Middle right): $u(x\boldsymbol{e}'')$ where $\boldsymbol{e}'' = 10^{-1/2}(\underbrace{1, \cdots, 1}_{ten 1's}, 0, 0, \cdots, 0)$ (Lower left): The loss L versus the number of epoch; (lower right): The relative error L^2 error versus the number of epoch. The training takes about less than 30 minutes.
- **Test 4: nonlinear PBE in a d=10 dimensional unit ball.** For a 1:1 symmetric two species ionic solvent, the electrostatic potential based on the Debye-Huckel theory [1] is given by a nonlinear PBE before linearization, and

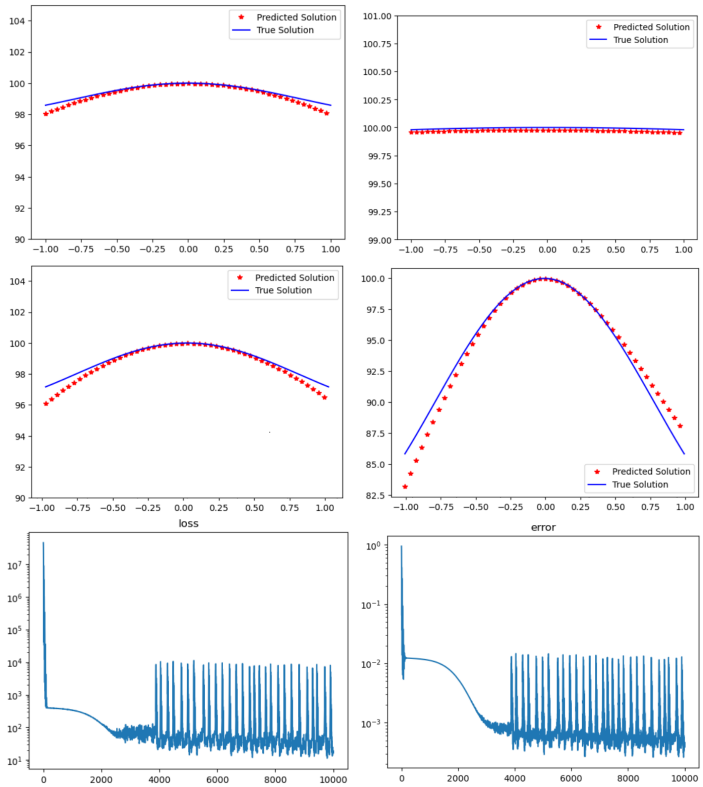

Fig. 5.4: DeepMartNet solution of the PBE in $D = \{\mathbf{x} \in \mathbb{R}^d, |\mathbf{x}| < 1\}$, d=100. (Upper left): true and predicted value of u at the diagonal of $\mathbb{R}^d$, i.e. x versus $u(x\boldsymbol{e})$ where $\boldsymbol{e} = d^{-1/2}(1, 1\cdots, 1)$; (Upper right): true and predicted value of u at the first coordinate axis of $\mathbb{R}^d$, i.e. x versus $u(x\boldsymbol{e_1})$ where $\boldsymbol{e} = d^{-1/2}(1, 0\cdots, 0)$. (Middle left): $u(x\boldsymbol{e'})$ where $\boldsymbol{e'} = 2^{-1/2}(1, 1, 0, 0, \cdots, 0)$ (Middle right): $u(x\boldsymbol{e''})$ where $\boldsymbol{e''} = 10^{-1/2}(\underbrace{1, \cdots, 1}_{ten1's}, 0, 0, \cdots, 0)$ (Lower left): The loss L versus the number of epoch; (lower right): The relative error L^2 error versus the number of epoch.

we will consider the following model nonlinear PBE to test the capability of the DeepMartNet for solving nonlinear PDEs,

$$\begin{cases} -\triangle u + \sinh u = f & x \in D, \\ u = g & x \in \partial D, \end{cases} \tag{5.8}$$

where

$$D = \{x \in \mathbb{R}^d : \|\mathbf{x}\|_2 \leq L\}, L = 1. \tag{5.9}$$

We consider the case of a true solution as

$$u = \alpha \sum_{i=1}^{d} x_i^2, \quad \alpha = 2, \tag{5.10}$$

which gives the boundary data and right hand side of the PBE as

$$g \equiv \alpha L^2,$$

and

$$f = -2\alpha d + \sinh\left(\alpha \sum_{i=1}^{d} x_i^2\right).$$

This time, we use the loss as

$$Loss(\theta) = Loss_{\text{mart}}(\theta) + \alpha_{\text{bdry}} Loss_{\text{bdry}}(\theta), \tag{5.11}$$

where $\alpha_{\text{bdry}} = 10^{-4}$, and the Martingale loss (4.13) is

(5.12)

$$Loss_{\text{mart}}(\theta) = \frac{1}{N}\sum_{i=0}^{N-1} \frac{\mathbb{I}\left(t_i \le \tau_{\partial D}^{(j)}\right)}{M_b^2} \cdot \left(\sum_{j=1}^{M_b} u_\theta\left(W_{t_{i+1}}^{(j)}\right) - u_\theta\left(W_{t_i}^{(j)}\right) - \frac{\Delta t}{2}\left(\sinh u(W_{t_i}^{(j)}) - f\left(W_{t_i}^{(j)}\right)\right)\right)^2,$$

and the boundary condition loss is again

$$Loss_{\text{bdry}}(\theta) = \frac{1}{N_{\text{bdry}}}\sum_{k=0}^{N_{\text{bdry}}} (u_\theta(\mathbf{x}_k) - g(\mathbf{x}_k))^2, \tag{5.13}$$

and a mini-batch of $N_{\text{bdry}} = 2000$ points $\mathbf{x}_k$ are uniformly sampled points on the boundary for every epoch of training. And in this case, we do not have a Feynman-Kac loss at the starting point $\mathbf{x}_0$.
The numerical results in Fig. 5.5 is done with the DeepMartNet with the fully connected NN $(10, 10, 10, 1)$. The activation function for the first hidden layer is Tanh, and the one for the second hidden layer is GELU with Tanh approximation. $M_{\text{tot}} = 10^6$ paths are used and the size of mini-batch $M_b = 4000$ paths are sampled in each epoch. The total number of epoch of the training is 3000. The terminal time for the paths is $T = 0.25$ and time step is $t = 0.01$. The learning rate starts at 0.01, and decreased by a factor of 0.99 every 100 epochs. The training takes less than 20 minutes.

5.2. Eigenvalue problem for elliptic equations. In this section, we will apply the DeepMartNet to solve elliptic eigenvalue problems in bounded and unbounded domains for both self-adjoint and non-self adjoint operators.

5.2.1. Eigenvalue problem of the Laplace equation in a cube in R^{10}. First, we consider a self-adjoint eigenvalue problem

$$-\triangle u = \lambda u, \quad \mathbf{x} \in D = [-L, L]^d, \quad d = 10, \quad L = 1 \tag{5.14}$$

$$u|_{\partial D} = 0, \tag{5.15}$$

with the following eigenfunction

$$u(x) = sin(\frac{\pi x_1}{L}) \cdots sin(\frac{\pi x_d}{L}), \tag{5.16}$$

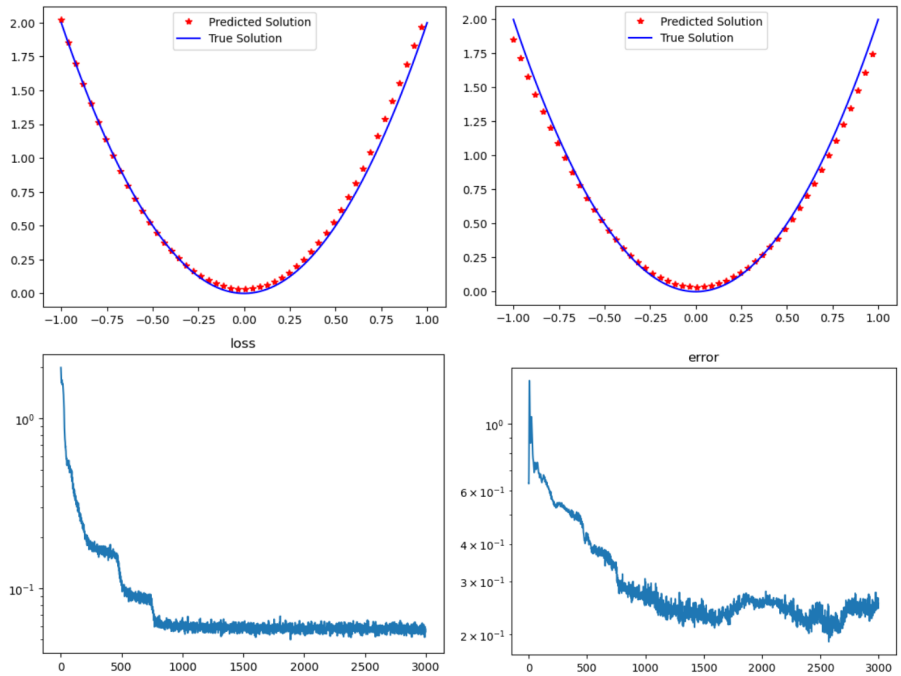

Fig. 5.5: DeepMartNet solution of nonlinear PBE (5.8) in a 10-dimensional unit ball. Upper left: $u(x\boldsymbol{e_1})$ where $\boldsymbol{e_1} = (1, 0, \cdots, 0)$; Upper right: $u(x\boldsymbol{e})$ where $\boldsymbol{e} = (1, 1, \cdots, 1)$; Lower left: loss vs. epoch; right: relative L^2 error vs epoch.

for the lowest eigenvalue

$$\lambda_1 = d(\frac{\pi}{L})^2. \tag{5.17}$$

A DeepMartNet with a fully connected structure $(10, 20, 10, 1)$ and GELU activation function are used. The total number of paths is $M = 100,000$, and for each epoch, we randomly choose $M_b = 1000$ from the paths; the time step of the paths is $\Delta t = 0.05$. A mini-batch of $N_{\text{bdry}} = 2000$ points $\mathbf{x}_k$ are uniformly sampled points on the boundary for every epoch of training. An Adamax optimizer with learning rate 0.05 is applied for training. $\alpha_{\text{bdry}} = 10^3$ in (4.19). The constant $\alpha_{normal} = 10, p = 1, m = 1, \mathbf{x}_1 = 0, c = 1$ in (4.18).

To accelerate the convergence of the eigenvalue, We included an extra term into the loss function (4.19)

$$\alpha_{\text{eig}}|\lambda^{(i)} - \lambda^{(i-100)}|^2, \tag{5.18}$$

where i is the index of the current epoch, which corresponds to the residual of the equation $\frac{d\lambda}{dt} = 0$ as the training time $t \to \infty$, and ensures that the eigenvalue λ will converge and the penalty constant α_{eig} is chosen as 2.5×10^{-8} in this case. The terminal time for the paths is $T = 0.6$ and time step is $t = 0.01$. The learning rate starts at 0.02, and decreased by a factor of 0.995 every 100 epochs. The training takes less than 20 minutes.

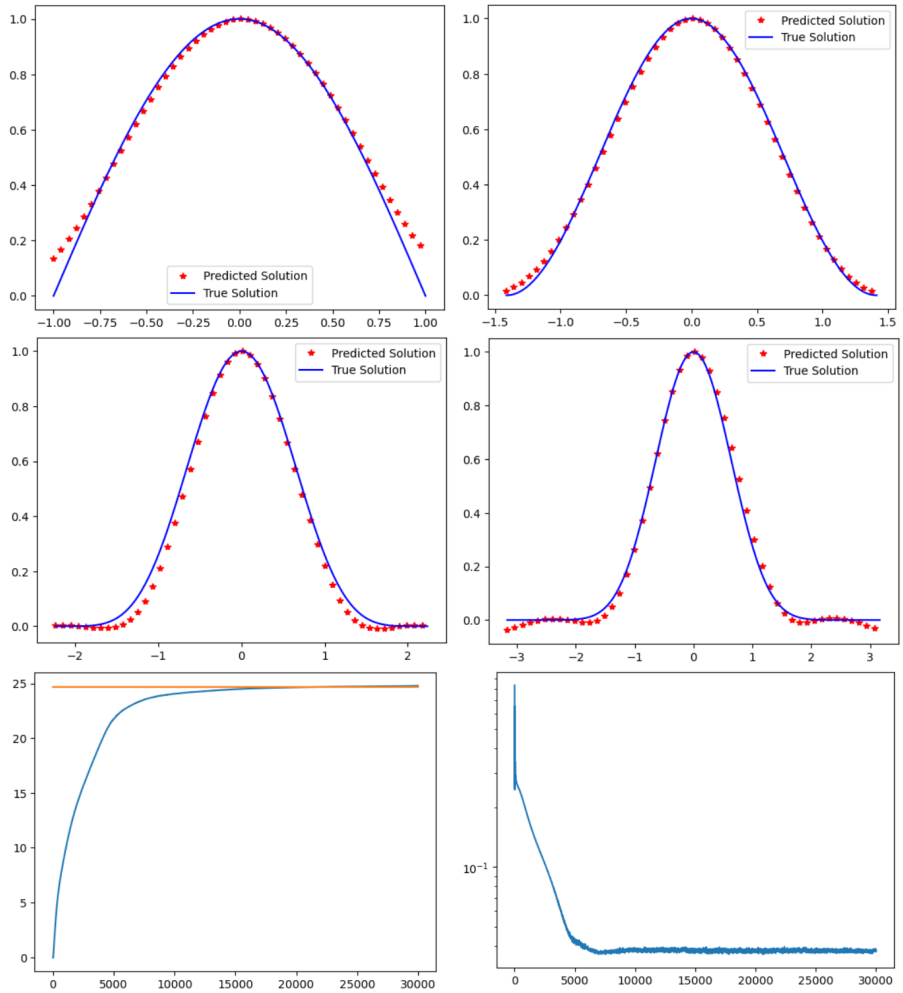

Fig. 5.6: A numerical result for eigenvalue problem of Laplace equation in a cube $[-L, L]^{10}$ in R^{10}. Upper left to middle right: true and predicted value of u at $x\boldsymbol{e}$ where $\boldsymbol{e} = k^{-1/2}(1, \underset{k1's}{\cdots}, 1, 0, 0, \cdots, 0)$ for $k = 1, 2, 5, 10$ Lower left: The predicted eigenvalue λ versus the number of epoch, (orange) horizontal line shows the true eigenvalue; lower right: The relative error L^2 error versus the number of epoch.

5.2.2. A non-self adjoint eigenvalue problem for the Fokker-Planck equations. Here we consider the following a non-self adjoint eigenvalue problem of the Fokker-Planck equation

$$-\Delta\psi - \nabla\cdot(\psi\nabla W) + c\psi = -\Delta\psi - \nabla W\cdot\nabla\psi - \Delta W\psi + c\psi = \lambda\psi, \quad \mathbf{x}\in R^d \tag{5.19}$$

where the eigenfunction for the eigenvalue $\lambda = c$ with a zero decay condition at ∞ is

$$\psi(\mathbf{x}) = e^{-W(\mathbf{x})}. \tag{5.20}$$

Here, we will consider a quadratic potential for our numerical tests

$$W(\mathbf{x}) = ||\mathbf{x}||^2, \quad \mathbf{x}\in R^d.$$

Equation (5.19) can re-written as

$$\mathcal{L}\psi = \frac{1}{2}\Delta\psi + \frac{1}{2}\nabla W \cdot \nabla\psi = -\left(\frac{1}{2}\Delta W - \frac{1}{2}c + \frac{1}{2}\lambda\right)\psi. \tag{5.21}$$

The generator for the SDE $\mathcal{L}$ will have a drift and a diffusion as

$$\mu = \frac{1}{2}\nabla W \quad \text{and} \quad \sigma = I_{d\times d}.$$

In order to enforce explicitly the decay condition of the eigenfunction at the infinite, a mollifier of the following form will be used as a pre-factor for the DNN solution $u_\theta(\mathbf{x}) = \rho(\mathbf{x})\tilde{u}_\theta(\mathbf{x})$ to the eigenfunction„

$$\rho(\mathbf{x}) = \frac{1}{1 + (\frac{||\mathbf{x}||}{\alpha})^2}, \tag{5.22}$$

where α is a constant and $\alpha = \frac{5}{11}$.

The Martingale loss of (4.17) for this case will be

$$\begin{aligned} Loss_{mart} =& \frac{1}{\Delta t}\frac{1}{N}\sum_{i=0}^{N-1}\frac{1}{|A_i|^2}\left(\sum_{m=1}^{|A_i|} u_\theta(\mathbf{x}_\mathbf{i}^{(m)} + \mathbf{1}) - u_\theta(\mathbf{x}_\mathbf{i}^{(m)})\right. \\ &\left. +(\frac{1}{2}\Delta W(\mathbf{x}_\mathbf{i}^{(m)}) - \frac{1}{2}c + \frac{1}{2}\lambda)u_\theta(\mathbf{x}_\mathbf{i}^{(m)})\Delta t\right)^2. \end{aligned} \tag{5.23}$$

For the mini-batches in this case, we take the size of each A_i to be between $\frac{M}{200}$ and $\frac{M}{25}$ as a random assortment of the M total trajectories.

To speed up the convergence of the eigenvalue and eigenfunctions, we found out that by taking fractional powers of each individual loss term is helpful, namely, the total loss is now modified as

$$Loss_{total} = \left((Loss_{mart})^p + \alpha_{normal}(Loss_{normal})^q\right)^r. \tag{5.24}$$

In our numerical tests, a typical choice is $p = 3/8, q = 1, r = 3/4$, and for the normalization $Loss_{normal}$, we set $\alpha_{normal} = 50$ and $c = 30$, $m = 2, \mathbf{x}_1 = 0, \mathbf{x}_2 = (1, 0, \cdots, 0)$ in (4.18).

Figs. 5.7 and 5.8 show the learned eigenvalues ($\lambda = 5$ for $d = 5$ and $\lambda = 200$ for $d = 200$) and eigenfunctions, respectively. The $k = 3$-trapezoidal rule is used in the Martingale loss $Loss_{mart}$ and the other numerical parameters used are listed as follows

- The total number of paths starting from the origin $M = 9,000, 24,000$ for $d = 5, 200$, respectively.
- The number of time steps $N = 1350, 1300$ and the terminal time $T = 9$ for $d = 5, 200$, respectively.
- the learning rate is $1/150$ and is halved every 500 epochs starting at epoch 500 and halved twice at epoch 7500 for stabilization.
- A In the following numerical tests, a fully connected network with layers (d, 6d, 3d, 1) with a Tanh activation function is used for the eigenfunction while a fully connected network (1, d, 1) with a $ReLu^9$ activation function with a constant value input is used to represent the eigenvalue.
- An Adamax optimizer is applied for training.

The final relative error in the eigenvalues after 10,000 epochs of training is $1.3 \times 10^{-2}, 6.7 \times 10^{-3}$ for $d = 5, 200$, respectively. And the relative L^2 error of the eigenfunction calculated along the diagonal of the domain is $2.6 \times 10^{-2}, 2.9 \times 10^{-2}$ for $d = 5, 200$, respectively. The training takes 25 minutes for the case of $d = 200$.

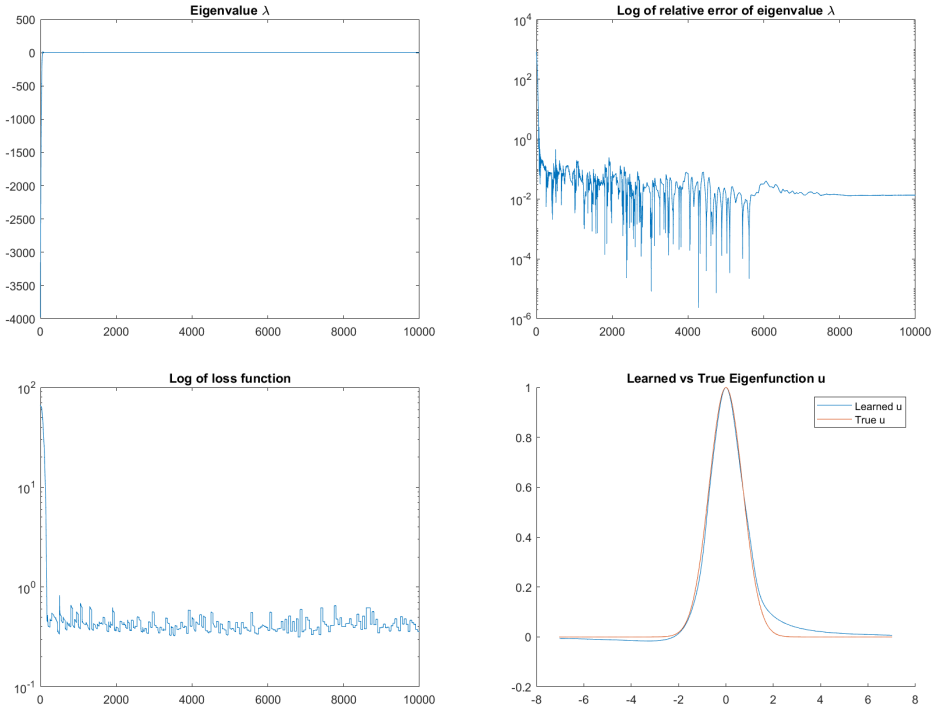

Fig. 5.7: Eigen-value problem of Fokker-Planck equation in R^d $d = 5$ for eigenvalue $\lambda = 5$. (Top left) Convergence of eigenvalue, (top right) history of eigenvalue error, (bottom left) history of loss function, (bottom right) Learn and exact eigenfunction along the diagonal of the domain.

6. Conclusion. In this paper, we introduced a Martingale based neural network, DeepMartNet, for solving the BVPs and eigenvalue problems of elliptic operators with Dirichlet boundary conditions. The DeepMartNet enforces the Martingale property for the PDE solutions through the stochastic gradient descent (SGD) optimization of the Martingale property loss functions. The connection between the Martingale definition and the mini-batches used in stochastic gradient computation shows a natural fit between the SGD optimization in DNN training and the Martingale problem formulation of the PDE solutions. Numerical results in high dimensions for both BVPs and eigenvalue problems show the promising potential in approximating high dimensional PDE solutions. The numerical results show that the DeepMartNet extracts more information for the PDEs solution over the whole solution domain than the traditional one point solution Feynman-Kac formula from the same set of diffusion paths originating from just one point.

Future work on DeepMartNet will include PDEs with Neumann and Robin boundary conditions where the underlying diffusion will be a reflecting one and the local time of the reflecting diffusion will be computed and included in the definition of the Martingale loss function. Another area of application is in optimal stochastic control where the Martingale optimality condition for the control and the backward SDE for the value function can be used to construct the loss function for the control as well as

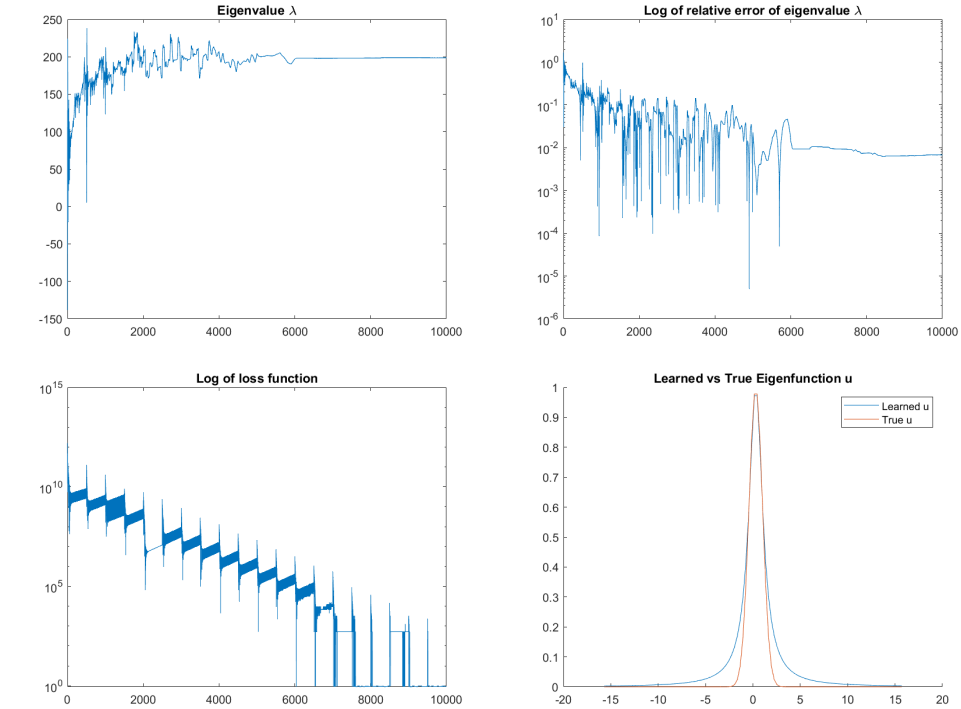

Fig. 5.8: Eigenvalue problem of Fokker-Planck equation in R^d $d = 200$ for eigenvalue $\lambda = 200$. (Top left) Convergence of eigenvalue, (top right) history of eigenvalue error, (bottom left) history of loss function, (bottom right) Learn and exact eigenfunction along the diagonal of the domain.

the value function [2]. Another important area for the application of the DeepMartNet is to solve low-dimension PDEs in complex geometries as the method uses diffusion paths to explore the domain and therefore can handle highly complex geometries such as the interconnects in microchip designs and nano-particles in material sciences and molecules in biology. Finally, The convergence analysis of the DeepMartNet, especially to a given eigenvalue, and the choices of mini-batches of diffusion paths and various hyper-parameters, which can affect the convergence of the learning strongly, in the loss functions are important issues to be addressed.

Acknowledgement. W. C. would like to thank Elton Hsu and V. Papanicolaou for the helpful discussion about their work on probabilistic solutions of Neumann and Robin problems.

REFERENCES

[1] Cai W. Computational Methods for Electromagnetic Phenomena: electrostatics in solvation, scattering, and electron transport. Cambridge University Press; 2013 Jan 3.

[2] Cai W. DeepMartNet – A Martingale based Deep Neural Network Learning Algorithm for Eigenvalue/BVP Problems and Optimal Stochastic Controls. arXiv preprint arXiv:2307.11942. 2023 Jul 21.

[3] Cohen SN, Elliott RJ. Stochastic calculus and applications. New York: Birkhäuser; 2015 Nov

18.
[4] Davis MH. Martingale methods in stochastic control. InStochastic Control Theory and Stochastic Differential Systems: Proceedings of a Workshop of the Sonderforschungsbereich 72 der Deutschen Forschungsgemeinschaft an der Universität Bonn “which took place in January 1979 at Bad Honnef 2005 Oct 6 (pp. 85-117). Berlin, Heidelberg: Springer Berlin Heidelberg.
[5] Ding CY, Zhou, YJ, Cai W, Zeng X, and Yan CH. A Path Integral Monte Carlo (PIMC) Method based on Feynman-Kac Formula for Electrical Impedance Tomography, Journal of Computational Physics., 476 (2023) 111862. 121.
[6] E W, Han J, Jentzen A. Algorithms for solving high dimensional PDEs: from nonlinear Monte Carlo to machine learning. Nonlinearity. 2021 Dec 9;35(1):278.
[7] Han J, Jentzen A, E W. Solving high-dimensional partial differential equations using deep learning. Proceedings of the National Academy of Sciences. 2018 Aug 21;115(34):8505-10.
[8] Han J, E W. Deep learning approximation for stochastic control problems, Deep Reinforcement Learning Workshop, NIPS. arXiv preprint arXiv:1611.07422. 2016.
[9] Han J, Lu J, Zhou M. Solving high-dimensional eigenvalue problems using deep neural networks: A diffusion Monte Carlo like approach. Journal of Computational Physics. 2020 Dec 15;423:109792.
[10] Hsu, P. Reflecting Brownian Motion, Boundary Local Time and the Neumann Problem, Dissertation Abstracts International Part B: Science and Engineering [DISS. ABST. INT. PT. B-SCI. ENG.], 45(6), 1984.
[11] Hsu P. Probabilistic approach to the Neumann problem. Communications on pure and applied mathematics. 1985 Jul;38(4):445-72.
[12] Hu Z, Shukla K, Karniadakis GE, Kawaguchi K. Tackling the curse of dimensionality with physics-informed neural networks. arXiv preprint arXiv:2307.12306. 2023 Jul 23.
[13] Hutzenthaler M, Jentzen A, Kruse, T, Nguyen TA and von Wurstemberger, P. Overcoming the curse of dimensionality in the numerical approximation of semilinear parabolic partial differential equations. Proceedings of the Royal Society A, 476(2244):20190630, 2020.
[14] Ji S, Peng, S, Peng Y, Zhang X. Three algorithms for solving high-dimensional fully coupled FBSDEsthrough deep learning, IEEE Intell. Syst. 35(3) (Feb 2020) 71–84.
[15] Karatzas I, Shreve S. Brownian motion and stochastic calculus. Springer Science & Business Media; 2012 Dec 6.
[16] Kingma DP, Ba J. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980. 2014 Dec 22.
[17] Klebaner FC. Introduction to stochastic calculus with applications. World Scientific Publishing Company; 2012 Mar 21.
[18] Papanicolaou VG. The probabilistic solution of the third boundary value problem for second order elliptic equations, Probab. Theory Relat. Fields 87 (1990) 27-77.
[19] Pardoux E, Peng S. Backward stochastic differential equations and quasilinear parabolic partial differential equations. In Stochastic partial differential equations and their applications 1992 (pp. 200-217). Springer, Berlin, Heidelberg
[20] Pfau D, Spencer JS, Matthews AG, Foulkes WM. Ab initio solution of the many-electron SchrÃ¶dinger equation with deep neural networks. Physical Review Research. 2020 Sep 16;2(3):033429.
[21] Raissi M. Forward-backward stochastic neural networks: Deep learning of high-dimensional partial differential equations. arXiv preprint arXiv:1804.07010. 2018 Apr 19.
[22] Schuss Z. Brownian dynamics at boundaries and interfaces. Springer-Verlag New York; 2015.
[23] Stroock DW, Varadhan SRS.Diffusion processes with boundary conditions. Commun. Pure Appl. Math. 24, 147-225 (1971).]
[24] Zhang W, Cai W. FBSDE based neural network algorithms for high-dimensional quasilinear parabolic PDEs. Journal of Computational Physics. 2022 Dec 1;470:111557.