INTERNATIONAL JOURNAL OF
NUMERICAL ANALYSIS AND MODELING
Volume 18, Number 4, Pages 427–441

A DEEP LEARNING GALERKIN METHOD FOR THE SECOND-ORDER LINEAR ELLIPTIC EQUATIONS

JIAN LI*, WEN ZHANG, AND JING YUE

Abstract. In this paper we propose a Deep Learning Galerkin Method (DGM) based on the deep neural network learning algorithm to approximate the general second-order linear elliptic problem. This method is a combination of Galerkin Method and machine learning. The DGM uses the deep neural network instead of the linear combination of basis functions. Our algorithm is meshfree and we train the neural network by randomly sampling the space points and using the gradient descent algorithm to satisfy the differential operators and boundary conditions. Moreover, the approximate ability of a neural networks' solution to the exact solution is proved by the convergence of the loss function and the convergence of the neural network to the exact solution in L^2 norm under certain conditions. Finally, some numerical experiments reflect the approximation ability of the neural networks intuitively.

Key words. Deep learning Galerkin method, deep neural network, second-order linear elliptic equations, convergence, numerical experiments.

1. Introduction

Mathematical models in many fields can be described by partial differential equations (PDEs). As early as the establishment of the calculus theory, PDEs were used to describe various natural phenomena and were applied to various scientific or engineering technologies. The high-dimensional partial differential equations are applied to physics, engineering, aerospace and other aspects. The well-known examples include the Schrödinger equation in quantum many-body problem, the nonlinear Black-Scholes equation for pricing financial derivatives, the Hamilton-Jacobi-Bellman equation in dynamic programming and so on. Unfortunately, the solutions of most PDEs cannot be expressed in the form of analytical solutions, so their numerical solutions are particularly important. Though many numerical methods have been developed so far for solving PDEs, such as finite difference method, finite element method and finite volume method, these methods still have certain limitations. As for the higher dimensional problems, the computational cost of the surge in grid points goes up exponentially with the dimensionality. As a result, solving numerical solutions has been a longstanding challenge.

With the explosive growth of available data and computing resources, the deep neural network model has shown remarkable success in artificial intelligence[1, 2, 3, 4, 5]. Recently, [6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17] have proposed that it can use the neural networks to solve PDEs. The deep neural networks with many layers seem to do a surprisingly good job in modeling complicated datasets. Besides, many effective algorithms are proposed to solve some high-dimensional PDEs in [18, 27, 28, 30, 31, 33, 34, 35, 36, 37]. In these papers, most of them only give numerical computations and illustrate the validity of numerical solutions through

Received by the editors April 1, 2020 and, in revised form, March 15, 2021.
2000 *Mathematics Subject Classification.* 35J15, 65N12, 74S30.
*Corresponding author.

various pictures. But there are no strict proofs about the existence and uniqueness of the exact solutions as well as the convergence of the error between the numerical solutions and the exact solutions. Here, based on the framework of [6], we directly apply the DGM for solving the second-order PDEs without using Monte Carlo Method. This method is the merger of the Galerkin Method and machine learning, which is different from the traditional Galerkin Method. The DGM uses the deep neural network instead of the linear combination of basis functions. We train the neural network by randomly sampling the space points and using the gradient descent algorithm to satisfy the differential operators and boundary conditions. We don't need to form a grid in this process. This is also an important reason why the DGM can solve the high-dimensional PDEs. Obviously, the method presented is much simpler but more effective. Moreover, we obtain the convergence of the loss function and the neural network, respectively. Finally, the performance of the method is demonstrated by some numerical experiments.

The rest of paper is organized as follows. In the next section, we will introduce a method that solves high dimensional PDEs with meshfree deep learning algorithm. In section 3, the theorem to illustrate the convergence of the loss function is proved. We also give the proof of another theorem to guarantee the convergence of neural network's solution in section 4. Finally, a series of numerical experiments are given in section 5.

2. Methodology

Let $\Omega \subset \mathbb{R}^d$, $(d = 2, 3)$ be a bounded set with a sufficiently smooth boundary $\partial\Omega$. We consider a class of the second-order linear elliptic equations in combination with Dirichlet boundary conditions:

$$\alpha u(x) - \Delta u(x) = f(x), \quad in\ \Omega, \tag{1}$$

$$u(x) = g(x), \quad on\ \partial\Omega, \tag{2}$$

where $\alpha > 0$ is a positive constant.

In the following, recall the classical Sobolev spaces:

$$H^k(\Omega) = \Big\{\nu | \nu \in L^2(\Omega), D_w^k(\nu) \in L^2(\Omega), \forall \alpha : |\alpha| \le k\Big\},$$
$$H_0^k(\Omega) = \Big\{\nu \in H^k(\Omega) : \nu|_{\partial\Omega} = 0\Big\}.$$

Especially, $L^2(\Omega) = \Big\{\nu(x) | \Big(\int_\Omega |\nu(x)|^2 dx\Big)^{\frac{1}{2}} < \infty\Big\}$ is sometimes defined by $H^0(\Omega)$. Here, $D_w^k(\nu)$ is the generalized k-order derivative of ν and its classical norm is equipped with the norm $\|\nu\|_k$.

For simplicity of notations, let us denote

$$\mathcal{G}[u](x) = \alpha u(x) - \Delta u(x) - f(x).$$

By the Lax-Milgram theorem, the second-order linear elliptic problem (1)-(2) has a unique solution

$$u \in H^2(\Omega) \text{ such that } \|u\|_2 \le C(\|f\|_0 + \|g\|_{3/2,\partial\Omega}), \tag{3}$$

where $H^{3/2}(\partial\Omega)$ is equipped with the trace norm

$$\|\phi\|_{3/2,\partial\Omega} = \inf\big\{\|u\|_2 \big| u \in H^2(\Omega)\ and\ u\big|_{\partial\Omega} = \phi\big\}.$$

In this section, we apply the multilayer feed forward networks, which consists of an input layer, one or more hidden layers and one output layer, to approximate solution of the equations (1)-(2). The approximation capabilities of neural network architectures have recently been investigated by many authors. Especially, in [28], they have ascertained that the standard multilayer feedforward networks with activation function ψ can approximate any continuous function well on arbitrary compact subsets X of Ω, whenever activation function ψ is continuous, bounded and nonconstant. For convenience, we will specifically contrive our results only for the instance where there is only one hidden layer, each hidden unit has the same activation function ψ_i and one output unit. The output unit without threshold and activation function, it is only the results of the hidden layer weighted summation. The simplified diagram of this neural network is shown in Figure 1.

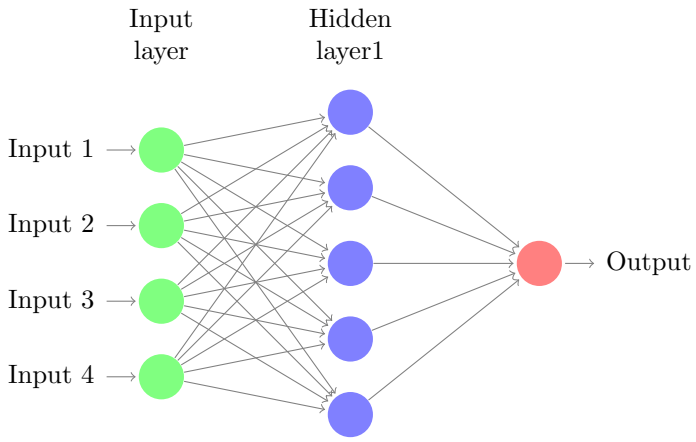


FIGURE 1. The neural networks with one hidden layer and one output unit.

Then the set of all functions carried out by such a network with n hidden units is

$$\mathfrak{C}^n(\psi) = \left\{ \zeta(x) : \mathbb{R}^m \to \mathbb{R} \middle| \zeta(x) = \sum_{i=1}^{n} \beta_i \psi_i \Big(\sum_{j=1}^{m} \alpha_{j,i} x_j + \gamma_i \Big) \right\}. \tag{4}$$

As for $\mathfrak{C}^n(\psi)$, we make the following notes:

- $\mathfrak{C}^n$: the class of neural networks consisting of a single hidden layer with n hidden units and m random points $x_j,\ j = 1, 2, \cdots, m$ and one output unit;
- ψ_i: the activation function of sigmoid type, of each hidden unit;
- γ_i: the threshold of the i_{th} hidden unit;
- $\alpha_{j,i}$: the connection weight of the i_{th} hidden unit and the j_{th} input unit;
- β_i: the connection weight of the i_{th} hidden unit and the output unit.

Denote $\theta = (\beta_1, \cdots, \beta_n, \gamma_1, \cdots, \gamma_n, \alpha_{1,1}, \cdots, \alpha_{m,n}) \in \mathbb{R}^{(2+m)n}$ are the elements of the parameter space. We assume that $U(x;\theta)$ is the neural network solution to equations (1)-(2). Then the loss function of deep neural network is defined by

$$J(U) = \|\mathcal{G}[U](x;\theta)\|^2_{0,\Omega,\nu_1} + \|U(x;\theta) - g(x)\|^2_{0,\partial\Omega,\nu_2}.$$

For the loss function, we make the following points:

- $\|f(y)\|^2_{0,\mathcal{Y},\nu} = \int_{\mathcal{Y}} |f(y)|^2 \nu(y) dy$, where ν is a positive probability density on $y \in \mathcal{Y}$;
- θ are the parameters of the neural network, ν_1 and ν_2 are the positive probability densities of Ω and $\partial\Omega$, respectively;
- $J(U)$ measures how well the approximation solution satisfies differential operators and boundary conditions.

The whole deep learning process is to find appropriate parameters θ such that $J(U)$ is as close to 0 as possible. If $J(U) = 0$, then $U(x)$ is the solution to equations (1)-(2), $U(x)$ naturally meets differential operators and boundary conditions. If we directly integration on Ω to minimize the loss function to find θ, the computation becomes larger and larger as the dimension d increases. To avoid above situation, we apply the DGM instead of the traditional numerical methods without forming mesh. The DGM is provided as follows:

Step 1. Generate sample points $x_n \in \Omega$ and $z_n \in \partial\Omega$ based on respective probability densities ν_1 and ν_2 randomly.

Step 2. At the random points $\tau_n = \{x_n, z_n\}$, we compute the square error $G(\theta_n; \tau_n)$ by

$$G(\theta_n; \tau_n) = \Big(\mathcal{G}[U](x_n; \theta_n)\Big)^2 + \Big(U(z_n; \theta_n) - g(z_n)\Big)^2.$$

Step 3. Apply the stochastic gradient descent step at τ_n by

$$\theta_{n+1} = \theta_n - \eta_n \nabla_\theta G(\theta_n, \tau_n).$$

Here, the learning rate η_n decreases as n increases.

Step 4. Repeat step 3 until $G(\theta_n; \tau_n)$ meets the convergence condition

$$\lim_{n\to\infty} \nabla_\theta G(\theta_n, \tau_n) = 0,$$

where $\mathbb{E}\Big[\nabla_\theta G(\theta_n, \tau_n)|_{\theta_n}\Big] = \nabla_\theta J(\theta_n, \tau_n)$, i.e. $\nabla_\theta G(\theta_n, \tau_n)|_{\theta_n}$ is the unbiased estimation of $\nabla_\theta J(\theta_n, \tau_n)$.

In fact,

$$\begin{aligned}
&\mathbb{E}\Big[\nabla_\theta G(\theta_n, \tau_n)|_{\theta_n}\Big] \\
=&\mathbb{E}\Big[\nabla_\theta \big((\mathcal{G}[U](x_n; \theta_n)\big)^2 + \big(U(z_n; \theta_n) - g(z_n)\big)^2\Big] \\
=&\nabla_\theta \Big[\int_\Omega \big(\mathcal{G}[U](x_n; \theta_n)\big)^2 \nu_1(x_n) dx_n + \int_{\partial\Omega} \big(U(z_n; \theta_n) - g(z_n)\big)^2 \nu_2(z_n) dz_n\Big] \\
=&\nabla_\theta \Big[\int_\Omega \Big|\mathcal{G}[U](x_n; \theta_n)\Big|^2 \nu_1(x_n) dx_n + \int_{\partial\Omega} \Big|U(z_n; \theta_n) - g(z_n)\Big|^2 \nu_2(z_n) dz_n\Big] \\
=&\nabla_\theta \Big[\big\|\mathcal{G}[U](x_n; \theta_n)\big\|^2_{0,\Omega,\nu_1} + \big\|(U(z_n; \theta_n) - g(z_n)\big\|^2_{0,\partial\Omega,\nu_2}\Big] \\
=&\nabla_\theta J(\theta_n, \tau_n).
\end{aligned}$$

Thus, we can use $\nabla_\theta G(\theta_n, \tau_n)$ instead of $\nabla_\theta J(\theta_n, \tau_n)$.

In order to describe the DGM more vividly, we use following photograph in Figure 2 as follows.

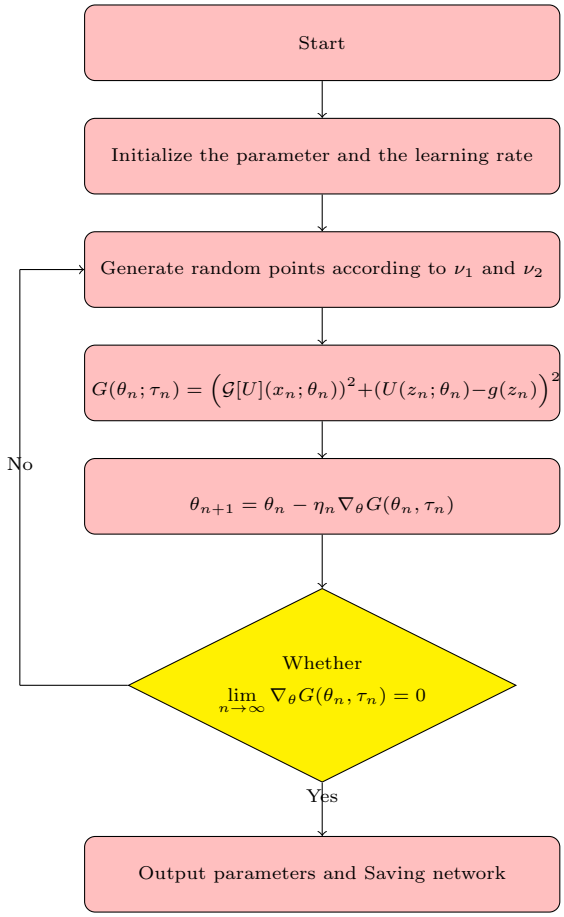


FIGURE 2. Flowchart of the DGM.

3. Convergence of the loss function

In this section, we propose a theorem to ensure the existence of multilayer feed forward networks U which makes the loss function $J(U)$ arbitrarily small. First, we give the following hypotheses:

H1. Let Ω be a bounded, compact and open subset of $\mathbb{R}^d$ and let parameters ν_1 and ν_2 be the measures of the Ω and $\partial\Omega$, respectively.

H2. Let $\mathfrak{C}^n(\psi)$ be the generalization of (4), in which ψ is a common function of hidden units in $\mathcal{C}^2(\Omega)$, bounded and non-constant. Moreover $\mathfrak{C}(\psi) = \{\mathfrak{C}^n(\psi)\}_{n=1}^{\infty}$ is the set implemented by $\mathfrak{C}^n(\psi)$, where n is an arbitrarily large number of multiple hidden layer units.

H3. Let the terms $\alpha u(x)$ and $\Delta u(x)$ be locally *Lipschitz* in $(u, \nabla u)$ with *Lipschitz* constant which has at most polynomial growth on u, uniformly with respect to x. This means that

$$|\Delta U - \Delta u| \leq \left(|\nabla U|^{q_1/2} + |\nabla u|^{q_2/2}\right)|\nabla U - \nabla u|,\ 0 \leq q_i \leq \infty,\ i = 1, 2.$$

$$
\begin{aligned}
|\Delta U-\Delta u|^2 &\le \Big(|\nabla U|^{q_1/2}+|\nabla u|^{q_2/2}\Big)^2|\nabla U-\nabla u|^2\\
&\le \Big(|\nabla U|^{q_1}+|\nabla u|^{q_2}+2|\nabla U|^{q_1/2}\cdot|\nabla u|^{q_2/2}\Big)|\nabla U-\nabla u|^2\\
&\le 2\Big(|\nabla U|^{q_1}+|\nabla u|^{q_2}\Big)|\nabla U-\nabla u|^2
\end{aligned}
$$

Theorem 1. Under the assumptions of *Hypotheses* (***H1***)-(***H3***), there exists a positive constant $C>0$ and a function $U\in\mathfrak{C}(\psi)$ such that

$$J(U)\le C\epsilon,$$

for $\forall\epsilon>0$, where C may depend on α, ν_1, ν_2, f and Ω.

Proof. Let u be the solution to (1)-(2), then $\mathcal{G}[u](x)=0$ naturally. Thus, we have

$$
\begin{aligned}
J(U) &= \|\mathcal{G}[U](x;\theta)\|^2_{0,\Omega,\nu_1}+\|U(x;\theta)-g(x)\|^2_{0,\partial\Omega,\nu_2}\\
&= \|\mathcal{G}[U](x;\theta)-\mathcal{G}[u](x)\|^2_{0,\Omega,\nu_1}+\|U(x;\theta)-g(x)\|^2_{0,\partial\Omega,\nu_2}\\
&\le \int_\Omega|\Delta U-\Delta u|^2d\nu_1(x)+\alpha\int_\Omega|U-u|^2d\nu_1(x)+\int_{\partial\Omega}|U-u|^2d\nu_2(x) \qquad (5)
\end{aligned}
$$

since

$$
\begin{aligned}
&\|\mathcal{G}[U](x;\theta)-\mathcal{G}[u](x)\|^2_{0,\Omega,\nu_1}\\
&=\int_\Omega\Big[(\Delta u-\Delta U)+\alpha(U-u)\Big]^2d\nu_1(x)\\
&=\int_\Omega\Big[-(\Delta U-\Delta u)+\alpha(U-u)\Big]^2d\nu_1(x)\\
&\le\int_\Omega\Big(-(\Delta U-\Delta u)\Big)^2d\nu_1(x)+\int_\Omega\Big(\alpha(U-u)\Big)^2d\nu_1(x).
\end{aligned}
$$

From [28], we can get that there exists a function $U\in\mathfrak{C}(\psi)$, which is uniformly 2-dense on compact sets of $H^2(\Omega)$. The meaning is that for $u\in H^2(\Omega)$ and $\epsilon>0$, there exists $U\in\mathfrak{C}(\psi)$ such that

$$\max_{|\alpha|\le2}\sup_{x\in\bar\Omega}|\partial_x^{(\alpha)}u(x)-\partial_x^{(\alpha)}U(x;\theta)|<\epsilon\ . \qquad (6)$$

Using *Hypothesis* (***H3***) and applying the Hölder inequality with indices p and q satisfying $\frac{1}{p}+\frac{1}{q}=1$, we have

$$
\begin{aligned}
&\int_\Omega|\Delta U-\Delta u|^2d\nu_1(x)\\
\le&\int_\Omega\Big(|\nabla U|^{q_1}+|\nabla u|^{q_2}\Big)|\nabla U-\nabla u|^2d\nu_1(x)\\
\le&\Big[\int_\Omega(|\nabla U|^{q_1}+|\nabla u|^{q_2})^pd\nu_1(x)\Big]^{1/p}\Big(\int_\Omega|\nabla U-\nabla u|^{2q}d\nu_1(x)v\Big)^{1/q}\\
\le&\Big[\int_\Omega(|\nabla(U-u)|^{q_1}+|\nabla u|^{q_1\vee q_2})^pd\nu_1(x)\Big]^{1/p}\Big(\int_\Omega|\nabla U-\nabla u|^{2q}d\nu_1(x)\Big)^{1/q}\\
\le&C(\epsilon^{q_1}+\sup|\nabla u|^{q_1\vee q_2})\epsilon^2. \qquad (7)
\end{aligned}
$$

The last step holds by (6).

Putting together (5)-(7), we obtain

$$J(U)=\|\mathcal{G}[U](x;\theta)\|^2_{0,\Omega,\nu_1}+\|U(x;\theta)-g(x)\|^2_{0,\partial\Omega,\nu_2}\le C\epsilon^2,$$

where $C < \infty$ may change from line to line and for two numbers $q_1 \vee q_2 = \max\{q_1, q_2\}$. ■

4. Convergence of the neural network to the unique solution

Section 4 contains the convergence results of the neural networks U^n to the solution u, as $n \to \infty$, of the equations (8)-(9),

$$\alpha u - \Delta u(x) = f(x), \quad in\ \Omega, \tag{8}$$

$$u(x) = 0, \quad on\ \partial\Omega. \tag{9}$$

At present, the loss function is

$$J(U) = \|\mathcal{G}[U]\|^2_{0,\Omega} + \|U\|^2_{0,\partial\Omega}.$$

Recollect that the norms above are L^2 norms in the space Ω and $\partial\Omega$, respectively. *Theorem* 1 implies that

$$J(U^n) \to 0\ as\ n \to \infty.$$

Each neural network U^n satisfies a PDE with a source term $h^n(x)$,

$$\mathcal{G}[U^n](x) = h^n(x), \quad x\ in\ \Omega, \tag{10}$$

$$U^n(x) = g^n(x), \quad x\ on\ \partial\Omega, \tag{11}$$

for some h^n and g^n such that

$$\|h^n\|^2_{0,\Omega} + \|g^n\|^2_{0,\partial\Omega} \to 0\ as\ n \to \infty. \tag{12}$$

For convenience, we only pay attention to the problem (8)-(9) since the corresponding problem with inhomogeneous boundary conditions can be easily settled by some classical homogeneous methods. For every $n \in \mathbb{N}$, $U^n \in L^2(\Omega)$, for convenience, we still use the same notation for the subsequence of U^n. Next we provide another important theorem.

Theorem 2. Under the *Hypotheses* (***H1***)-(***H3***) and (12). There exists a unique bounded solution $u \in H^2_0(\Omega)$ to (8)-(9). Moreover, when the sequence U^n is uniformly bounded and equicontinuous, the neural networks U^n converges strongly to u in $L^2(\Omega)$. Furthermore, U^n uniformly converges to u in Ω.

Proof. From Theorem 2.1 of [19] combined with Theorems 6.3-6.5 of Chapter V.6 in [20], we can obtain the existence, regularity and uniqueness for (8)-(9). Also, the boundedness established from Theorem 2.1 in [19] and V.2 in [20]. Consider equations (10)-(11) under the situation $g^n(x) = 0$ and the solution to this problem denoted by $\hat{U}^n(x)$. We need to state that $\hat{U}^n(x)$ is uniformly bounded and satisfies the following equations:

$$\mathcal{G}[\hat{U}^n](x) = h^n(x), \quad x\ in\ \Omega, \tag{13}$$

$$\hat{U}^n(x) = 0, \quad x\ on\ \partial\Omega. \tag{14}$$

Multiply both sides of the equation (13) by an arbitrary function $v \in \mathfrak{C}(\psi)$, then integrate on Ω to obtain the following equation

$$\alpha(\hat{U}^n, v) - (\nabla\hat{U}^n, \nabla v) = (h^n, v). \tag{15}$$

In particular, let $v = \hat{U}^n$ in (15)

$$\alpha\|\hat{U}^n\|_{0,\Omega}^2 + \|\nabla\hat{U}^n\|_{0,\Omega}^2 \leq \|h^n\|_{0,\Omega}\|\hat{U}^n\|_{0,\Omega}^2,$$

which implies together with the Poincaré inequality

$$\|\hat{U}^n\|_{1,\Omega} \leq C\|h^n\|_{0,\Omega}.$$

Obviously, sequence $\hat{U}^n(x)$ is uniformly bounded, in addition, there exists a weakly subsequence $\hat{U}^n$ (for simplicity, we still use the same notation for the subsequence) such that

$$\hat{U}^n \rightharpoonup u \ in \ \mathfrak{C}, \ n \to \infty.$$

Due to the compactness of the embedding $H_0^1(\Omega) \hookrightarrow L^2(\Omega)$, we can obtain

$$\hat{U}^n \to u \ in \ L^2(\Omega), \ n \to \infty.$$

This implies that $\hat{U}^n$ converges strongly to u in $L^2(\Omega)$ and $\lim_{n\to\infty} \|\hat{U}^n - u\|_{0,\Omega} = 0$. Up to subsequences, $\hat{U}^n$ converges almost everywhere to u in Ω. From the Theorem 3.3 of [23], we gain that

$$\nabla\hat{U}^n \to \nabla u \ almost \ everywhere \ in \ \Omega.$$

From mow on, we have been ready for passing to the limit as $n \to \infty$ in the weak formulation, therefore the weak formulation of the equation (10) with $g^n = 0$ reads as follows,

$$\alpha(\hat{U}^n, \phi) - (\nabla\hat{U}^n, \phi) - (h^n, \phi) = 0, \ for \ \forall \ \phi \in C_0^\infty(\Omega).$$

Using the above results of the convergence, we then obtain the weak formulation of the equations (8)-(9):

$$\alpha(u, \phi) - (\nabla u, \phi) - (f, \phi) = 0, \ for \ \forall \ \phi \in C_0^\infty(\Omega).$$

Now, $\lim_{n\to\infty} \|\hat{U}^n - u\|_{0,\Omega} = 0$ has been proved. We still need to state $\lim_{n\to\infty} \|U^n - \hat{U}^n\|_{0,\Omega} = 0$. Let us have a recollection. Here, U^n fulfilling equations (10)-(11) is the neural networks approximation solution, besides $\hat{U}^n$ makes (11) hold with $g^n = 0$. Furthermore, sequence U^n is uniformly bounded in $L^2(\Omega)$. Then, we can conclude that the subsequence of U^n converges at least weakly in $L^2(\Omega)$. By formulation (12), we find that g^n converges strongly in $L^2(\Omega)$. What's more, U^n is defined by g^n on the boundary $\partial\Omega$, so U^n will converge to 0 at least along a subsequence on the boundary. Then in (10)-(11), U^n will be same as $\hat{U}^n$ almost everywhere when $g^n = 0$.

Set $F_n = |U^n - \hat{U}^n|^2$. Thus, function sequence $\{F_n\}$ is uniformly bounded in the $L^2(\Omega)$ because of the boundeness of U^n and $\hat{U}^n$ above. $\{F_n\}$ is integrable on area $\bar{\Omega}$ naturally. Moreover, $\{F_n\}$ converges to 0 almost everywhere. Furthermore, using the definition of F_n together with the uniform boundedness as well as equicontinuity

of $U^n(x)$ and $\hat{U}^n(x)$, for $\forall \epsilon_0 > 0$, $\exists \delta > 0$, $|x-y| < \delta,\ x,\ y \in \bar{\Omega}$ and $n \geq 1$, there is

$$\begin{aligned}
&\Big|F_n(x) - F_n(y)\Big| \\
=&\Big||U^n(x) - \hat{U}^n(x)|^2 - |U^n(y) - \hat{U}^n(y)|^2\Big| \\
=&\Big||U^n(x) - \hat{U}^n(x)| + |U^n(y) - \hat{U}^n(y)|\Big| \times \Big||U^n(x) - \hat{U}^n(x)| - |U^n(y) - \hat{U}^n(y)|\Big| \\
\leq&\Big||U^n(x) - \hat{U}^n(x)| + |U^n(y) - \hat{U}^n(y)|\Big| \times \Big|(U^n(x) - U^n(y)) + (\hat{U}^n(y) - \hat{U}^n(x))|\Big| \\
\leq&\Big||U^n(x) - \hat{U}^n(x)| + |U^n(y) - \hat{U}^n(y)|\Big| \times \Big||U^n(x) - U^n(y)| + |\hat{U}^n(x) - \hat{U}^n(y)|\Big| \\
\leq& C\epsilon_0,
\end{aligned}$$

which implies that function sequence $\{F_n(x)\}$ is equicontinuous. Thus, using Vitali's theorem, we infer that

$$\lim_{n\to\infty} \|U^n - \hat{U}^n\|_{0,\Omega}^2 = 0,$$

which shows that $U^n - \hat{U}^n$ converges to 0 strongly in $L^2(\Omega)$. Using a triangle inequality, we can gain the following inequality

$$\begin{aligned}
&\lim_{n\to\infty} \|U^n - u\|_{0,\Omega} \\
=& \lim_{n\to\infty} \|U^n - \hat{U}^n + \hat{U}^n - u\|_{0,\Omega} \\
\leq& \lim_{n\to\infty} \|U^n - \hat{U}^n\|_{0,\Omega} + \lim_{n\to\infty} \|\hat{U}^n - u\|_{0,\Omega} \\
=&0.
\end{aligned}$$

This result shows that U^n converges strongly to u in $L^2(\Omega)$. Furthermore, based on the equicontinuity and uniform boundedness of U^n, by Arzelà-Ascoli theorem, we obtain that U^n uniformly converges to u in Ω. ■

5. Numerical results

In this section, we present a series of numerical results to illustrate the theoretical analysis of the algorithm proposed in this paper. We test the DGM for the second order linear elliptic problem (1)-(2) in the domain $\Omega = (0,1) \times (0,1)$ with manufactured problem $u(x) = \sin(\pi x_1)\sin(\pi x_2)$ and $\alpha = 2$. Then, $f(x) = 2\sin(\pi x_1) sin(\pi x_2) + 2\pi^2 \sin(\pi x_1) sin(\pi x_2)$ can be determined by (1).

In order to observe the influence of the random weight initialization on the loss function, we train the ten neural networks on dataset 4000. These neural networks have one hidden layer and random even seeds from 32 to 50. Also we test the same case with homogeneous boundary (i.e. $g(x) = 0$) under the same condition. The values of the loss function $J(U)$ can be found in Tables 1-2. Moreover, an interesting thing can be found that the loss function does not decrease as the seeds increase.

For convenience, we denote the discrete L^1-norm and L^2-norm between U_i and u_i by

$$errorL^1 = \frac{1}{m}\sum_{i=1}^{m} |U_i - u_i|,$$

$$errorL^2 = \frac{1}{m}\sum_{i=1}^{m} (U_i - u_i)^2,$$

TABLE 1. The neural network with different seeds: $g(x) \neq 0$.

hidden layer	1	1	1	1	1
random seed	32	34	36	38	40
$J(U)$	7.9×10^{-5}	6.7×10^{-5}	2.6×10^{-5}	3.0×10^{-5}	7.6×10^{-5}
random seed	42	44	46	48	50
$J(U)$	1.9×10^{-5}	8.4×10^{-5}	4.8×10^{-5}	8.1×10^{-5}	5.0×10^{-5}

TABLE 2. The neural network with different seeds: $g(x) = 0$.

hidden layer	1	1	1	1	1
random seed	32	34	36	38	40
$J(U)$	4.3×10^{-5}	8.3×10^{-5}	9.5×10^{-5}	6.0×10^{-5}	3.7×10^{-5}
random seed	42	44	46	48	50
$J(U)$	3.5×10^{-5}	2.3×10^{-5}	3.8×10^{-5}	7.7×10^{-5}	4.2×10^{-5}

TABLE 3. Numerical results of neural networks.

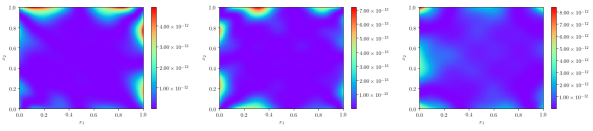
Random seed	42	42	42	42	42
Dataset	1000	2000	4000	8000	16000
Hidden layers	1	1	1	1	1
$errorL^2$	3.53×10^{-13}	4.48×10^{-14}	5.77×10^{-13}	5.95×10^{-14}	9.42×10^{-14}
$errorL^1$	4.03×10^{-7}	1.46×10^{-7}	6.60×10^{-7}	1.88×10^{-7}	2.41×10^{-7}
iterations	20000	20000	20000	14266	20000
CPU(s)	84.30	137.07	282.38	494.72	1004.25
Dataset	1000	2000	4000	8000	16000
Hidden layers	2	2	2	2	2
$errorL^2$	5.71×10^{-12}	2.06×10^{-12}	2.21×10^{-12}	8.48×10^{-12}	1.16×10^{-12}
$errorL^1$	1.42×10^{-6}	9.10×10^{-7}	9.57×10^{-7}	1.74×10^{-6}	6.61×10^{-7}
iterations	19574	19574	19574	19574	19574
CPU (s)	243.37	175.83	477.48	656.65	1461.66

respectively. Here, $errorL^1$ is the mean absolute error (MAE) and the $errorL^2$ is the mean square error (MSE) in the deep learning.

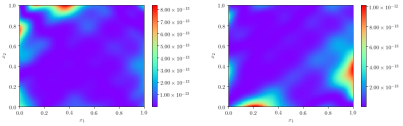
Based on this results, we implement the neural networks with one or two hidden layers and the random seeds of 42 or 44 to solve equations (1)-(2) with homogeneous boundary. The networks have 16 units on each hidden layer and are trained on five datasets with 1000, 2000, 4000, 8000 and 16000 interior and boundary sample points, respectively.

Observed from Tables 3-4 and Figures 3-10, we can see that the neural networks have obtained good results and can approximate the exact solution very well except for a larger error on the boundaries and the form of the L^1 norm with 1 hidden layer, see Figures 5 and 9. The performance shows that $J(U) \to 0$ does not necessarily imply that $U \to u$. On the other hand, the approximation ability of the deep neural networks is obviously better than that of the neural networks with one hidden layer.

Numerical results show that the deep learning is a valuable approach for solving a class of second-order linear elliptic equations. The PDEs' solution can be approximated by a deep neural network which is trained to satisfy the differential

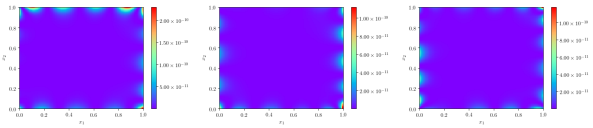


(a) Dataset = 1000. (b) Dataset = 2000. (c) Dataset = 4000.

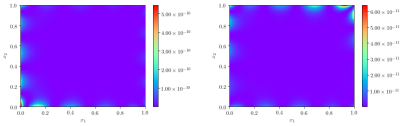


(d) Dataset = 8000. (e) Dataset = 16000.

FIGURE 3. $errorL^2$ with one hidden layer.

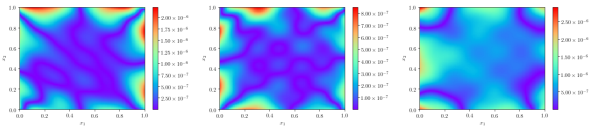


(a) Dataset = 1000. (b) Dataset = 2000. (c) Dataset = 4000.

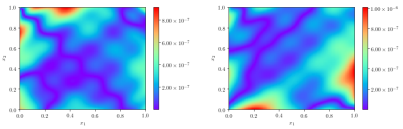


(d) Dataset = 8000. (e) Dataset = 16000.

FIGURE 4. $errorL^2$ with two hidden layers.



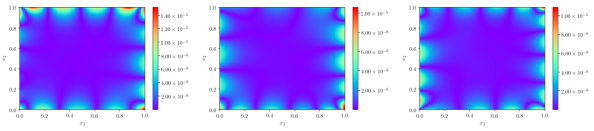
(a) Dataset = 1000. (b) Dataset = 2000. (c) Dataset = 4000.



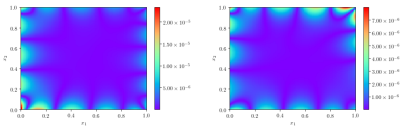
(d) Dataset = 8000. (e) Dataset = 16000.

FIGURE 5. $errorL^1$ with one hidden layer.

operators and boundary conditions. Moreover, the DGM converges to the solution of the PDEs as the number of hidden units goes to infinity.



(a) Dataset = 1000. (b) Dataset = 2000. (c) Dataset = 4000.

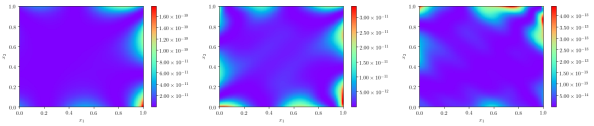


(d) Dataset = 8000. (e) Dataset = 16000.

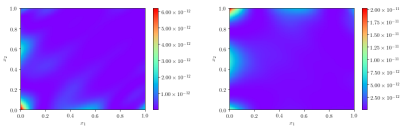
FIGURE 6. $errorL^1$ with two hidden layers.

TABLE 4. Numerical results of neural networks: $g(x) = 0$.

seed	44	44	44	44	44
Batch size	1000	2000	4000	8000	16000
Hidden layers	1	1	1	1	1
$errorL^2$	7.80×10^{-12}	3.06×10^{-12}	3.18×10^{-14}	1.89×10^{-13}	7.71×10^{-13}
$errorL^1$	1.98×10^{-6}	1.47×10^{-6}	1.35×10^{-7}	3.28×10^{-7}	6.86×10^{-7}
iterations	16548	20000	20000	20000	20000
elapsed time $(s)(s)$	95.84	134.19	262.74	358.21	1015.88
Batch size	1000	2000	4000	8000	16000
Hidden layers	2	2	2	2	2
$errorL^2$	1.74×10^{-12}	3.77×10^{-12}	1.26×10^{-11}	2.36×10^{-11}	7.65×10^{-12}
$errorL^1$	7.46×10^{-7}	1.14×10^{-6}	2.07×10^{-6}	2.83×10^{-6}	1.69×10^{-6}
iterations	13953	10314	10386	10940	11002
elapsed time (s)	209.23	297.79	629.17	953.70	1374.60



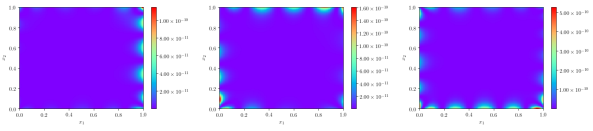
(a) 1000. (b) 2000. (c) 4000.



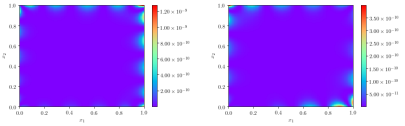
(d) 8000. (e) 16000.

FIGURE 7. $errorL^2$ with one hidden layer: $g(x) = 0$.

Furthermore, we also believe that it will be useful for solving the high-dimensional PDEs, since the DGM is meshfree, which is key because meshes become infeasible in higher dimensions, instead of forming a mesh.

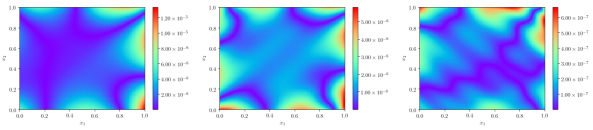


(a) 1000. (b) 2000. (c) 4000.

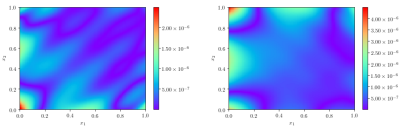


(d) 8000. (e) 16000.

FIGURE 8. $errorL^2$ with two hidden layers: $g(x) = 0$.

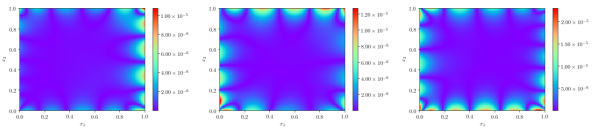


(a) 1000. (b) 2000. (c) 4000.

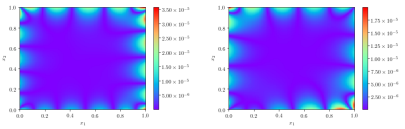


(d) 8000. (e) 16000.

FIGURE 9. $errorL^1$ with one hidden layer: $g(x) = 0$.



(a) 1000. (b) 2000. (c) 4000.



(d) 8000. (e) 16000.

FIGURE 10. $errorL^1$ with two hidden layers: $g(x) = 0$.

Acknowledgments

This research was partly supported by the NSF of China (No. 11771259), Shaanxi special support plan for regional development of talents and innovation

team on computationally efficient numerical methods based on new energy problems in Shaanxi province.

References

[1] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning MIT Press, 2016.
[2] L. Yann, Y. Bengio and G. Hinton, em Deep Learning, MIT Press, 2015, 521(7553): 436-444.
[3] A. Krizhevsky, I. Sutskever and G. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, Advances in Neural Information Processing Systems, 2012, 25(2): 84-90.
[4] G. Hinton, L. Deng, D. Yu, et al., Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups, IEEE Signal Processing Magazine, 2012, 29(6): 82-97.
[5] D. Silver, A. Huang, C. J. Maddison, et al., Mastering the Game of Go with Deep Neural Networks and Tree Search, Nature, 2016, 529(7587): 484-489.
[6] J. Sirignano, K. Spiliopoulos, DGM: A Deep Learning Algorithm for Solving Partial Differential Equations, Journal of Computational Physics, 2018, 375: 1339-1364.
[7] M. Raissi, P. Perdikaris and G. E. Karniadakis, Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations, arXiv:1711.10561, 2017.
[8] M. Raissi, P. Perdikaris and G. E. Karniadakis, Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations, arXiv:1711.10566, 2017.
[9] J. Berg, N. Kaj, A Unified Deep Artificial Neural Network Approach to Partial Differential Equations in Complex Geometries, Neurocomputing, 2017.
[10] I. E. Lagaris, A. Likas and D. I. Fotiadis, Artificial Neural Networks for Solving Ordinary and Partial Differential Equations, IEEE Transactions on Neural Networks, 1998, 9(5): 987-1000.
[11] I. E. Lagaris, A. Likas and D. G. Papageorgiou, Neural-network Methods for Boundary Value Problems with Irregular Boundaries, IEEE Transactions on Neural Networks, 2000, 11(5): 1041-9.
[12] H. Lee, I. S. Kang, Neural Algorithm for Solving Differential Equations, Journal of Computational Physics, 1990, 91(1): 110-131.
[13] A. Malek, B. R. Shekari, Numerical Solution for High Order Differential Equations Using A Hybrid Neural Network-optimization Method, Elsevier Science Inc. 2006, 183: 260-271.
[14] K. Rudd Solving Partial Differential Equations Using Artificial Neural Networks, PhD Thesis, Duke University, 2013.
[15] J. Ling, A. Kurzawski and J. Templeton, Reynolds Averaged Turbulence Modelling Using Deep Neural Networks with Embedded Invariance, Journal of Fluid Mechanics, 2016, 807: 155-166.
[16] J. Tompson, K. Schlachter, P. Sprechmann, et al., Accelerating Eulerian Fluid Simulation With Convolutional Networks, Proceeding of Machine Learning Research, 2017, 70: 3424-3433.
[17] C. Pratik, O. Adam and O. Stanley, et al., Deep Relaxation: Partial Differential Equations for Optimizing Deep Neural Networks, Research in the Mathematical Sciences, 2018, 5(3): 30-.
[18] J. Han, J. Arnulf and W. E, Solving High-Dimensional Partial Differential Equations Using Deep Learning, Proceedings of the National Academy of Sciences, arXiv:1707,20568, 2017.
[19] M. Porzio, Existence of Solutions for Some Noncoercive Parabolic Equations, Discrete and Continuous Dynamical Systems - Series A, 1999, 5(3): 553-568.
[20] O. A. Ladyzhenskaya, V. A. Solonnikov and N.N. Uraltseva, Linear and Quasilinear Equations of Parabolic Type, 1968, 23.
[21] S. Asmussen, P. W. Glynn, Stochastic Simulation: Algorithms and Analysis, Interfaces, 2007, 38(6): 487-488.
[22] D. P. Bertsekas, J. N. Tsitsiklis, Gradient Convergence in Gradient Methods with Errors, SIAM Journal on Optimization, 2000, 10(3): 627-642.
[23] L. Boccardo, A. Dallaglio, et al., Nonlinear Parabolic Equations with Measure Data, Journal of Functional Analysis, 1997, 147(1): 237-258.
[24] P. Chandra, Y. Singh, Feedforward Sigmoidal Networks-Equicontinuity and Fault-Tolerance Properties, IEEE Press, 2004, (15): 1350-66.
[25] G. Cybenko, Approximation by Superpositions of a Sigmoidal Function, Mathematics of Control, Signals, and Systems (MCSS), 1989, 2(4): 303-314.

[26] J. Dean, G. S. Corrado, R. Monga, et al., Large Scale Distributed Deep Networks, Advances in Neural Information Processing Systems, 2012.
[27] W. E, J. Han and A. Jentzen, Deep Learning-Based Numerical Methods for High-Dimensional Parabolic Partial Differential Equations and Backward Stochastic Differential Equations, Communications in Mathematics and Statistics, 2017, 5.
[28] K. Hornik, Approximation Capabilities of Multilayer Feedforward Networks, Neural Networks, 1991, 4(2): 251-257.
[29] B. Piero, R. Alan and Elcrat., Elliptic Partial Differential Equations of Second Order, Springer US, 1997: 213-267.
[30] M. Raissi, Forward-Backward Stochastic Neural Networks: Deep Learning of High-dimensional Partial Differential Equations, arXiv:1804.07010, 2018.
[31] M. Raissi, Deep Hidden Physics Models: Deep Learning of Nonlinear Partial Differential Equations, Journal of Machine Learning Research, arXiv:1801.06637, 2018.
[32] M. Raissi, G. E. Karniadakis, Hidden Physics Models: Machine Learning of Nonlinear Partial Differential Equations, Journal of Computational Physics, arXiv:1708.00588, 2017.
[33] J. Darbon, S. Osher, Algorithms for Overcoming the Curse of Dimensionality for Certain Hamilton-Jacobi Equations Arising in Control Theory and Elsewhere, Research in the Mathematical Sciences, 2016, 3.
[34] R. Fuentes, A. Poznyak, I. Chairez, et al., Partial Differential Equations Numerical Modeling Using Dynamic Neural Networks, International Conference on Artificial Neural Networks. Springer-Verlag, 2009, 5769: 552-562.
[35] A. Alharbi, An Artificial Neural Networks Method for Solving Partial Differential Equations, 2010, 1281: 1425-1428
[36] A. A. S. Almarashi, Approximation Solution of Fractional Partial Differential Equations by Neural Networks, Advances in Numerical Analysis, 2012, 2012: 19-28.
[37] J. Li, J. Yue, W. Zhang and W. S. Duan, The Deep Learning Galerkin Method for the General Stokes Equations, submitted.

Department of Mathematics, Shaanxi University of Science and Technology, Xi'an, 710021, China

E-mail: jianli@sust.edu.cn

Department of Mathematics, Shaanxi University of Science and Technology, Xi'an, 710021, China

E-mail: ZW15290229577@163.com and yjing1995@163.com.