Euro. Jnl of Applied Mathematics: page 1 of 15
doi:10.1017/S0956792520000182

Solving parametric PDE problems with artificial neural networks

YUEHAW KHOO[1], JIANFENG LU[2] and LEXING YING[3]

[1]*Department of Statistics, University of Chicago, IL 60615, USA*
email: ykhoo@uchicago.edu
[2]*Department of Mathematics, Department of Chemistry and Department of Physics, Duke University, Durham, NC 27708, USA*
email: jianfeng@math.duke.edu
[3]*Department of Mathematics and ICME, Stanford University, Stanford, CA 94305, USA*
email: lexing@stanford.edu

(*Received 30 August 2019*; *revised 21 April 2020*; *accepted 27 May 2020*)

The curse of dimensionality is commonly encountered in numerical partial differential equations (PDE), especially when uncertainties have to be modelled into the equations as random coefficients. However, very often the variability of physical quantities derived from PDE can be captured by a few features on the space of the coefficient fields. Based on such observation, we propose using neural network to parameterise the physical quantity of interest as a function of input coefficients. The representability of such quantity using a neural network can be justified by viewing the neural network as performing time evolution to find the solutions to the PDE. We further demonstrate the simplicity and accuracy of the approach through notable examples of PDEs in engineering and physics.

Key words: Neural-network, parametric PDE, uncertainty quantification

2020 Mathematics Subject Classification: 65Nxx

1 Introduction

Uncertainty quantifications in physical and engineering applications often involve the study of partial differential equations (PDE) with random fields of coefficients. To understand the behaviour of a system in the presence of uncertainties, one can extract PDE-derived physical quantities as functionals of the coefficient fields. This can potentially require solving the PDE exponential number of times numerically even with suitable discretisation of the PDE domain and of the range of random variables. Fortunately in most PDE applications, often these functionals depend only on a few characteristic 'features' of the coefficient fields, allowing them to be determined from solving the PDE a limited number of times.

A commonly used approach for uncertainty quantifications is Monte-Carlo sampling. An ensemble of solutions is built by repeatedly solving the PDE with different realisations. Then physical quantities of interest, for example, the mean of the solution at a given location, can be computed from the ensemble of solutions. Although being applicable in many situations, the

computed quantity is inherently noisy. Moreover, it lacks the ability to obtain new solutions if they are not sampled previously. Other approaches exploit the low underlying dimensionality assumption in a more direct manner. For example, the stochastic Galerkin method [13, 16] expands the random solution using certain prefixed basis functions (i.e. polynomial chaos [18, 19]) on the space of random variables, thereby reducing the high-dimensional problem to a few deterministic PDEs. Such type of methods requires careful treatment of the uncertainty distributions, and since the basis used is problem independent, the method could be expensive when the dimensionality of the random variables is high. There are data-driven approaches for basis learning such as applying Karhunen–Loève expansion to PDE solutions from different realisations of the PDE [3]. Similar to the related principal component analysis, such linear dimension-reduction techniques may not fully exploit the nonlinear interplay between the random variables. At the end of day, the problem of uncertainty quantification is one of characterising the low-dimensional structure of the coefficient field that gives the observed quantities.

On the other hand, the problem of dimensionality reduction has been central to the fields of statistics and machine learning. The fundamental task of regression seeks to find a function h_θ parameterised by a parameter vector $\theta \in \mathbb{R}^p$ such that

$$f(a) \approx h_\theta(a), \ a \in \mathbb{R}^n. \tag{1.1}$$

However, choosing a sufficiently large class of approximation functions without the issue of over-fitting remains a delicate business. As an example, in linear regression, the standard procedure is to fix a set of basis (or feature maps) $\{\phi_k(a)\}$ such that

$$f(a) = \sum_k \beta_k \phi_k(a) \tag{1.2}$$

and determine the parameter β_k's from sampled data. The choice of basis is important to the quality of regression, just as in the case of studying PDEs with random coefficients. Recently, deep neural networks have demonstrated unprecedented success in solving a variety of difficult regression problems related to pattern recognitions [8, 11, 15]. A key advantage of neural network is that it bypasses the traditional need to handcraft basis for spanning $f(a)$, but instead learns the optimal basis that satisfies (1.1) directly from data. The performance of neural network in machine learning applications, and more recently in physical applications such as representing quantum many-body states (e.g. [17, 2]), prompts us to study its use in the context of solving PDE with random coefficients. More precisely, we want to learn $f(a)$ that maps the coefficient vectors a in a PDE to some physical quantity described by the PDE.

Our approach to solve for quantities arise from PDE with random coefficients consists of the following simple steps:

- Sample the random coefficients (a in (1.1)) of the PDE from a user-specified distribution. For each set of coefficients, solve the deterministic PDE to obtain the physical quantity of interest ($f(a)$ in (1.1)).
- Use a neural network as the surrogate model $h_\theta(a)$ in (1.1) and train it using the previously obtained samples.
- Validate the surrogate forward model with more samples. The neural network is now ready for applications.

Though being a simple method, to the best of our knowledge, dimension reduction based on neural network representation has not been adapted to solving PDE with uncertainties. We consider two simple but representative parametric PDE tasks in this work: elliptic homogenisation and nonlinear Schrödinger eigenvalue problem. The main contributions of our work are as follows:

- We provide theoretical guarantees on neural network representation of $f(a)$ through explicit construction for the parametric PDE problems under study;
- We show that even a rather simple neural network architecture can learn a good representation of $f(a)$ through training.

We note that our work is different from [10, 14, 12, 9], which solve deterministic PDE numerically using a neural network. The goal of these works is to parameterise the solution of a deterministic PDE using neural network and replace Galerkin-type methods when performing model reduction. It is also different from [6] where a deterministic PDE is solved as a stochastic control problem using neural network. In this paper, the function that we want to parameterise is over the coefficient field of the PDE.

The advantages of having an explicitly parameterised approximation to $f(a)$ are numerous, which we will only list a couple here. First, the neural network parameterised function can serve as a surrogate forward model for generating samples cheaply for statistical analysis. Second, the task of optimising some function of the physical quantity with respect to the PDE coefficients can be done with the help of a gradient calculated from the neural network. To summarise, obtaining a neural network parametrisation could limit the use of expensive PDE solvers in applications.

We demonstrate the success of neural network in two PDE applications. In particular, we consider solving for the effective conductance in inhomogeneous media and the ground state energy of a nonlinear Schrödinger equation (NLSE) having inhomogeneous potential. These are important physical models with wide applications in physics and engineering.

In Section 2, we provide background on the two PDEs of interest. In Section 3, the theoretical justification of using neural network (NN) to represent the physical quantities derived from the PDEs introduced in Section 2 is provided. In Section 4, we describe the neural network architecture for handling these PDE problems and report the numerical results. We finally conclude in Section 5.

2 Two examples of parametric PDE problems

This section introduces the two PDE models – the linear elliptic equation and the NLSE – we want to solve for. We focus on the map from the coefficient field of these equations to certain physical quantities of interest. In both cases, the boundary condition is taken to be periodic for simplicity.

2.1 Effective coefficients for inhomogeneous elliptic equation

Our first example will be finding the effective conductance in a non-homogeneous media. For this, we consider the elliptic equation

$$\nabla \cdot a(x)(\nabla u(x) + \xi) = 0, \quad x \in [0,1]^d \tag{2.1}$$

with periodic boundary condition where $\xi \in \mathbb{R}^d$, $\|\xi\|_2^2 = 1$ ($\|\cdot\|_2$ is the Euclidean norm). To ensure ellipticity, we consider the class of coefficient functions

$$\mathcal{A} = \{a \in L^\infty([0,1]^d) \mid \lambda_1 \geq a \geq \lambda_0 > 0\}. \tag{2.2}$$

For a given ξ, we want to obtain the effective conductance functional $A_{\text{eff}} : \mathcal{A} \to \mathbb{R}$ defined by

$$A_{\text{eff}}(a) = \int_{[0,1]^d} a(x)\|\nabla u_a(x) + \xi\|_2^2 \,\mathrm{d}x = -\int_{[0,1]^d} u_a(x)\nabla \cdot a(x)(\nabla u_a(x) + 2\xi) - a(x)\,\mathrm{d}x, \tag{2.3}$$

where u_a satisfies (2.1) (the subscript 'a' in u_a is used to denote the dependence of the solution of (2.1) on the coefficient field a). The second equality follows from integration by parts.

In practice, to parameterise A_{eff} as a function of the coefficient field $a(x)$, we discretise the domain using a uniform grid with step size h and grid points $x_i = ih$, where the multi-index $i \in \{(i_1, \ldots, i_d)\}$, $i_1, \ldots, i_d = 1, \ldots, n$ with $n = 1/h$. In this way, we can think about the coefficient field $a(x)$ and the solution $u(x)$ evaluated on the grid points both as vectors with length n^d. More precisely, let the action of Laplace operator on u (the term $\nabla \cdot a(x)\nabla u(x)$) be discretised using central difference as

$$\sum_{k=1}^{d} \frac{\frac{1}{2}(a_{i+e_k} + a_i)(u_{i+e_k} - u_i) + \frac{1}{2}(a_{i-e_k} + a_i)(u_{i-e_k} - u_i)}{h^2} \tag{2.4}$$

for each i, where $\{e_k\}_{k=1}^d$ denotes the canonical basis in $\mathbb{R}^d$. Here $a_i := a(i/n)$ and $u_i := u(i/n)$. Then the discrete version of (2.1) is obtained as

$$\begin{aligned}(L_a u + b_{\xi,a})_i := &\\ &\sum_{k=1}^{d} \frac{(a_{i+e_k} + a_i)u_{i+e_k} + (a_i + a_{i-e_k})u_{i-e_k} - (a_{i-e_k} + 2a_i + a_{i+e_k})u_i}{2h^2} \\ &+ \sum_{k=1}^{d} \frac{\xi_k(a_{i+e_k} - a_{i-e_k})}{2h} = 0, \qquad \forall i,\end{aligned} \tag{2.5}$$

where the first equality gives the definitions of L_a and $b_{\xi,a}$. The discrete version of effective conductance is obtained as $2\mathcal{E}(u_a; a)$, where

$$\mathcal{E}(u; a) = -\frac{1}{2}u^\top L_a u - u^\top b_{\xi,a} + \frac{1}{2n^d}a^\top \mathbf{1}, \tag{2.6}$$

and $\mathbf{1}$ is the all-one vector.

2.2 NLSE with random potential

For the second example, we want to find the ground state energy E_0 of a NLSE with potential $a(x)$:

$$-\Delta u(x) + a(x)u(x) + \sigma u(x)^3 = E_0 u(x), \quad x \in [0,1]^d \tag{2.7}$$

subject to the normalisation constraint

$$\int_{[0,1]^d} u(x)^2 \,\mathrm{d}x = 1. \tag{2.8}$$

We take $\sigma = 2$ in this work and thus consider a defocusing cubic Schrödinger equation, which can be understood as a model for soliton in nonlinear photonics or Bose–Einstein condensate with inhomogeneous media. Similar to (2.5), we solve the discretised version of the NLSE

$$-Lu + a_i u_i + \sigma u_i^3 = E_0 u_i \ \ \forall i, \quad \frac{1}{n^d}\sum_{i=1}^{n^d} u_i^2 = 1, \tag{2.9}$$

where

$$L := \sum_{k=1}^{d} \frac{u_{i+e_k} + u_{i-e_k} - 2u_i}{h^2}. \tag{2.10}$$

Due to the nonlinear cubic term, it is more difficult to solve for the NLSE numerically compared to (2.1). Therefore in this case, the value of having a surrogate model of E_0 as a function of a is more significant. We note that the solution u to (2.9) (and thus E_0) can also be obtained from a variational problem

$$\min_{u:\, \|u\|_2^2 = n^d} -u^\top L u + u^\top \mathrm{diag}(a) u + \frac{\sigma}{2}\sum_i u_i^4, \tag{2.11}$$

where the diag$(\cdot)$ operator forms a diagonal matrix given a vector.

3 Theoretical justification of deep neural network representation

The physical quantities introduced in Section 2 are determined through the solution of the PDEs given the coefficient field. Rather than solving the PDE, we will prove that the map from coefficient field to such quantities can be represented using convolutional NNs. The main idea is to view the solution u of the PDE as being obtained via time evolution, where each layer of the NN corresponds to the solution at discrete time step. We focus here the case of solving elliptic equation with inhomogeneous coefficients. Similar line of reasoning can be used to demonstrate the representability of the ground state-energy E_0 as a function of a using an NN.

Theorem 1 *Fix an error tolerance $\epsilon > 0$, there exists a neural network $h_\theta(\cdot)$ with $O(n^d)$ hidden nodes per-layer and $O(n^2/\epsilon)$ layers such that for any $a \in \mathcal{A} = \{a \in L^\infty([0,1]^d) \mid a(x) \in [\lambda_0, \lambda_1],\ \forall x,\ \lambda_0 > 0\}$, we have*

$$|h_\theta(a) - A_{eff}(a)| \le \epsilon \lambda_1. \tag{3.1}$$

The proof of Theorem 1 is given in Appendix A. Note that due to the ellipticity assumption $a \in \mathcal{A}$, the effective conductivity is bounded from below by $A_{\mathrm{eff}}(a) \ge \lambda_0 > 0$. Therefore the theorem immediately implies a relative error bound

$$\frac{|h_\theta(a) - A_{\mathrm{eff}}(a)|}{A_{\mathrm{eff}}(a)} \le \epsilon \frac{\lambda_1}{\lambda_0}. \tag{3.2}$$

We illustrate the main idea of the proof in the rest of the section, and the technical details of the proof are deferred to the supplementary materials.

First observe that the effective coefficient obeys a variational principle

$$A_{\mathrm{eff}}(a) = 2 \min_u \mathcal{E}(u; a), \tag{3.3}$$

where $L_a, b_{\xi,a}$ are defined in (2.5). Therefore, to get $A_{\mathrm{eff}}(a)$, we may minimise $\mathcal{E}(u; a)$ over the solution space u, using e.g. steepest descent:

$$\begin{aligned} u^{m+1} &= u^m - \Delta t \frac{\partial \mathcal{E}(u^m; a)}{\partial u} \\ &= u^m + \Delta t\left(L_a u^m + b_{\xi,a}\right), \end{aligned} \tag{3.4}$$

where Δt is a step size chosen sufficiently small to ensure descent of the energy. Note that the optimisation problem is convex due to the ellipticity assumption (2.2) of the coefficient field a (which ensures $-u^\top L_a u > 0$ except for $u = \mathbf{1}$) with Lipschitz continuous gradient; therefore, the iterative scheme converges to the minimiser with proper choice of step size for any initial condition. Thus we can choose $u^0 = 0$.

Now we can identify the iteration scheme with a convolutional NN architecture by viewing m as an index of the NN layers. The input of the NN is the vector $a \in \mathbb{R}^{n^d}$, and the hidden layers are used to map between the consecutive pairs of $(d+1)$-tensors $U^m_{i_0 i_1 \dots i_d}$ and $U^{m+1}_{i_0 i_1 \dots i_d}$. The zeroth dimension for each tensor U^m is the channel dimension and the last d dimensions are the spatial dimensions. If we let the channels in each U^m be consisted of a copy of a and a copy of u^m, e.g. let

$$U^m_{0 i_1 \dots i_d} = a_{(i_1,\dots,i_d)}, \quad U^m_{1 i_1 \dots i_d} = u^m_{(i_1,\dots,i_d)}, \tag{3.5}$$

in light of (3.4) and (2.5), one simply needs to perform local convolution (to aggregate a locally) and nonlinearity (to approximate quadratic form of a and u^m) to get from $U^m_{1 i_1 \dots i_d} = u^m_{(i_1,\dots,i_d)}$ to $U^{m+1}_{1 i_1 \dots i_d} = u^{m+1}_{(i_1,\dots,i_d)}$; while the 0-channel is simply copied to carry along the information of a. Stopping at $m = M$ layer and letting u^M be the approximate minimiser of $\mathcal{E}(u; a)$, based on (3.3), we obtain an approximation to $A_{\text{eff}}(a)$. Note that the architecture of NN used in the proof resembles a deep ResNet [7], as the coefficient field a is passed from the first to the last layer. The detailed estimates of the approximation error and the number of parameters will be deferred to the supplementary materials.

Let us point out that if we take the continuum time limit of the steepest descent dynamics, we obtain a system of ODE

$$\partial_t u = L_a u + b_{\xi,a}, \tag{3.6}$$

which can be viewed as a spatial discretisation of a PDE. Thus our construction of the neural network in the proof is also related to the work [12] where multiple layers of convolutional NN are used to learn and solve evolutionary PDEs. However, the goal of the neural network here is to approximate the physical quantity of interest as a functional of the (high-dimensional) coefficient field, which is quite different from the view point of [12].

We also remark that the number of layers of the NN required by Theorem 1 is rather large. This is due to the choice of the (unconditioned) steepest descent algorithm as the engine of optimisation to generate the neural network architecture used in the proof. With a better preconditioner such as the algebraic multigrid [20], we can effectively reduce the number of layers to $O(1)$ and thus achieves an optimal count of parameters involved in the NN. In practice, as shown in the next section by actual training of parametric PDEs, the neural network architecture can be much simplified while maintaining good approximation to the quantity of interest.

4 Proposed network architecture and numerical results

In this section, based on the discussion in Section 3, we propose using convolutional NN to approximate the physical quantities given by the PDE with a periodic boundary condition. The architecture of the neural network is described in Section 4.1, and then the implementation details and numerical results are provided in Sections 4.2 and 4.3 respectively.

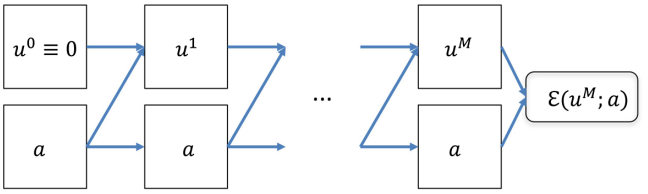

FIGURE 1. Construction of the NN in the proof of Theorem 1. The NN takes the coefficient field a as an input and the convolutional layers are used to map from u^m, a to u^{m+1}, a, $m = 0, M-1$. At u_M, local convolutions and nonlinearity are used to obtain $\mathcal{E}(u^M; a)$.

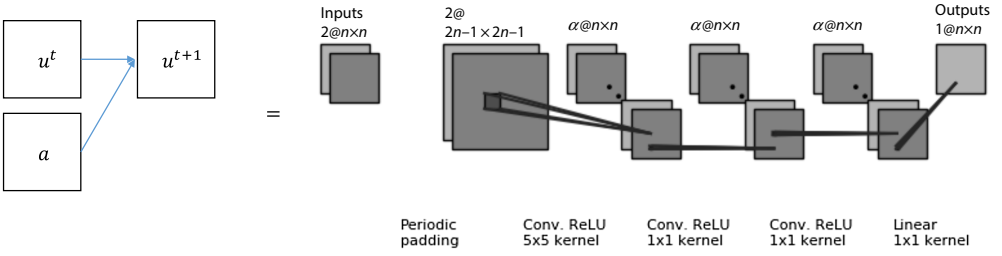

FIGURE 2. The map between (u^t, a) and u^{t+1}.

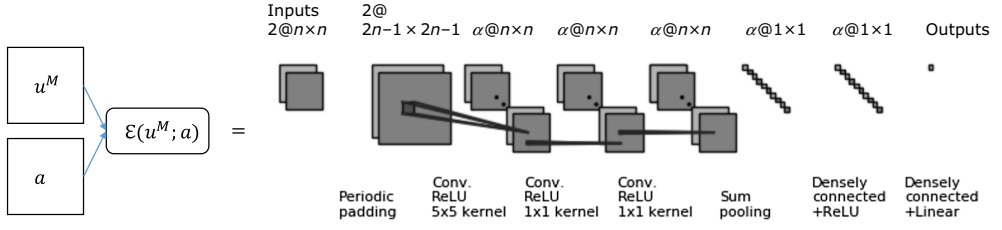

FIGURE 3. The map between (u^M, a) and $\mathcal{E}$.

4.1 Architectures

In this subsection, we present two different architectures. Since we have shown theoretically that the NN proposed in Figure 1 can approximate the effective conductance function, the NN in Figure 1 will serve as a basis for the first architecture. In the second architecture, we simply use an NN that incorporates translational symmetry. For most of the numerical tests in subsequent sections, we report results using the second architecture since its simplicity demonstrates the effectiveness of an NN. However, results for the first NN architecture are also given in order to provide numerical evidence for Theorem 1 for the case of determining the effective conductance.

The first architecture is based on a ResNet [7] where the construction of NN is illustrated in Figure 1. In Figures 2 and 3, the explicit maps between (u^t, a) and u^{t+1}, and between (u^M, a) and $\mathcal{E}$ are detailed respectively. For the sake of illustration, we assume a 2D unit square domain, though it can be generalised to solving PDEs in any dimensions. The input to the NN is a matrix $a \in \mathbb{R}^{n\times n}$ representing the coefficient field on grid points, and the output of the network gives physical quantity of interest from the PDE. Based on (3.4) and (2.5), the cubic function that

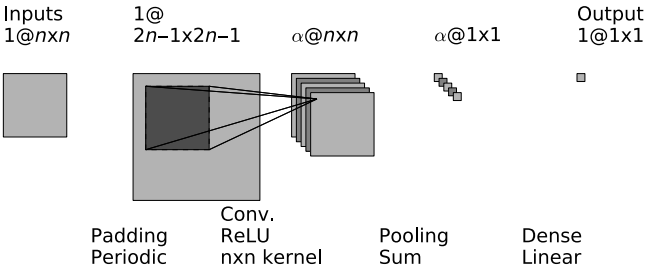

FIGURE 4. Single convolutional layer neural network for representing translational invariant function.

maps from (u^t, a) to u^{t+1} is realised via a few convolutional layers as in Figure 2. Since the function $\mathcal{E}(u^M; a)$ has translational symmetry in the sense that

$$\mathcal{E}(u^M_{(i+\tau_1)(j+\tau_2)}; a_{(i+\tau_1)(j+\tau_2)}) = \mathcal{E}(u^M; a) \tag{4.1}$$

where the additions are done on $\mathbb{Z}_n$, a sum-pooling is used in Figure 3 to preserve the translational symmetry of the function $\mathcal{E}$.

Figure 4 shows the second architecture for the 2D case. When designing this architecture, we forgo the PDE knowledge used to prove Theorem 1, and simply used the fact that

$$f(a) := \mathcal{E}(u_a; a) \tag{4.2}$$

has translational symmetry in terms of the input a. Again, the main part of the network is convolutional layers with ReLU being the nonlinearity. This extracts the relevant features of the coefficient field around each grid point that contribute to the final output. The use of a sum-pooling followed by a linear map to obtain the final output is again based on the translational symmetry of the function f to be represented. More precisely, let $a^{\tau_1\tau_2}_{ij} := a_{(i+\tau_1)(j+\tau_2)}$, where the additions are done on $\mathbb{Z}_n$. The output of the convolutional layer gives basis functions that satisfy

$$\tilde{\phi}_{kij}(a^{\tau_1\tau_2}) = \tilde{\phi}_{k(i-\tau_1)(j-\tau_2)}(a), \quad \forall(\tau_1, \tau_2) \in \{1, \ldots, n\}^2. \tag{4.3}$$

When using the architecture in Figure 4, for any $\tau_1\tau_2$,

$$f(a^{\tau_1\tau_2}) = \sum_{k=1}^{\alpha} \beta_k \sum_{i=1}^{n}\sum_{j=1}^{n} \left(\tilde{\phi}_{k(i-\tau_1)(j-\tau_2)}(a)\right) = \sum_{k=1}^{\alpha} \beta_k \phi_k(a), \quad \phi_k := \sum_{i=1}^{n}\sum_{j=1}^{n} \tilde{\phi}_{kij}, \tag{4.4}$$

where β_k's are the weights of the last densely connected layer. Therefore the translational symmetry of f is preserved.

We note that all operations in Figures 3 and 4 are standard except the padding operation. Typically, zero-padding is used to enlarge the size of the input in image classification task, whereas we extend the input periodically due to the assumed periodic boundary condition.

4.2 Implementation

The neural network is implemented using Keras [4], an application programming interface running on top of TensorFlow [1] (a library of toolboxes for training neural network). A mean-squared-error loss function is used and the optimisation is done using the NAdam optimiser [5].

Table 1. *Error in approximating the effective conductance function $A_{eff}(a)$ in 2D using the architecture in Figure 1. The definition of α is given in Figure 1. We test the NN with data model 1 and 2, where $a_1, \ldots, a_{n^2}$ are generated from independent and correlated random field respectively. The mean and standard deviation of the effective conductance are computed from the samples in order to show the variability. The sample sizes for training and validation are the same.*

Data model	n	α	Training error	Validation error	Average A_{eff}	No. of samples	No. of param.
1	8	10	2.0e-3	1.8e-3	1.58 ± 0.10	1.2e + 4	4416
1	16	10	1.5e-3	1.4e-3	1.58 ± 0.052	2.4e + 4	4416
2	8	10	1.2-3	1.2e-3	4.87 ± 1.00	1.2e + 4	4416
2	16	10	6.8-4	6.7e-4	4.97 ± 1.01	2.4e + 4	4416

The hyper-parameter we tune is the learning rate, which we lower if the training error fluctuates too much. The weights are initialised randomly from the normal distribution. The input to the neural network is whitened to have unit variance and zero-mean on each dimension. The mini-batch size is always set between 50 and 200.

4.3 Numerical examples

4.3.1 *Effective conductance*

For the case of effective conductance, we assume the following distributions for the input data:

1. The independent and identical random variables $a_i, i = 1, \ldots, n^d$ are distributed according to $\mathcal{U}[0.3, 3]$ where $\mathcal{U}[\lambda_0, \lambda_1]$ denotes the uniform distribution on the interval $[\lambda_0, \lambda_1]$.
2. Correlated random variables $a_i, i = 1, \ldots, n^d$ with covariance matrix $PP^T \in \mathbb{R}^{n^d \times n^d}$:

$$a = Pb \in \mathbb{R}^{n^d}, \tag{4.5}$$

$b_1, \ldots, b_{n^d}$ are independently and identically distributed as $\mathcal{U}[0.3, 5]$, and the covariance matrix is defined by

$$[PP^T]_{ij} = \exp\left(-\frac{\|x_i - x_j\|^2}{(2h)^2}\right), \quad i, j = 1, \ldots, n^d. \tag{4.6}$$

The results of learning the effective conductance function are presented in Table 1. We use the same number of samples for training and validation. Both the training and validation error are measured by

$$\sqrt{\frac{\sum_k (h_\theta(a^k) - A_{\text{eff}}(a^k))^2}{\sum_k A_{\text{eff}}(a^k)^2}}, \tag{4.7}$$

where a^k's can be either the training or validation samples sampled from the same distribution and h_θ is the neural network-parameterised approximation function. For this experiment in $d = 2$, we report the results using two different architectures. In Table 1, the results for the architecture in Figure 1 with $M = 5$ provide numerical evidence for Theorem 1. In Table 2, the more economical NN in Figure 5 is used to demonstrate that by simply considering the symmetry in the

Table 2. *Error in approximating the effective conductance function $A_{eff}(a)$ in 2D using the architecture in Figure 5. The definition of α is given in Figure 5. The sample sizes for training and validation are the same.*

Data model	n	α	Training error	Validation error	Average A_{eff}	No. of samples	No. of param.
1	8	16	1.5e-3	1.4e-3	1.58 ± 0.10	1.2e + 4	1057
1	16	16	2.2e-3	1.8e-3	1.58 ± 0.052	2.4e + 4	4129
2	8	16	1.0-3	1.0e-3	4.87 ± 1.00	1.2e + 4	1057
2	16	16	2.5-3	2.5e-3	4.97 ± 1.01	2.4e + 4	4129

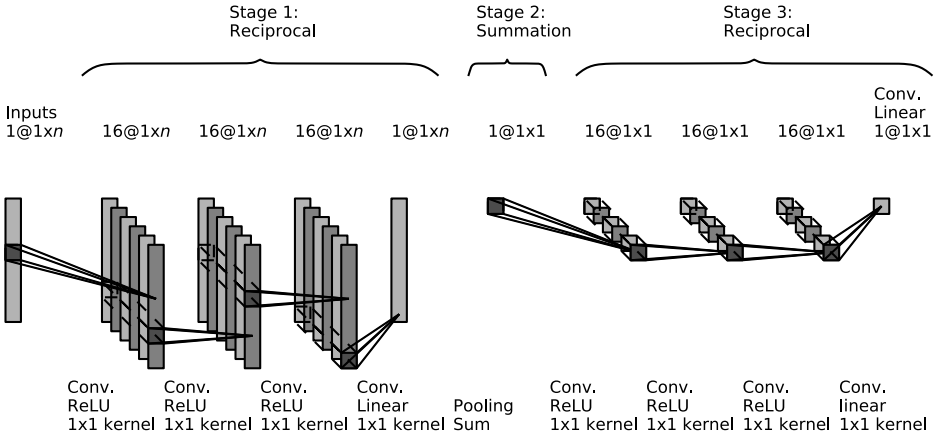

FIGURE 5. Neural network architecture for approximating $A_{\text{eff}}(a)$ in the 1D case. Although the layers in third stage are essentially densely connected layers, we still identify them as convolution layers to reflect the symmetry between the first and third stages.

function to be approximated (without using PDE knowledge), results with relatively good accuracy can already be obtained. The simple NN in Figure 5 gives similar results as the deep NN in Figure 1, indicating we might be able to further reduce the complexity of the NN in Theorem 1 via a more careful construct.

Before concluding this subsection, we use the exercise of determining the effective conductance in 1D to provide another motivation for the usage of a neural network. In 1D the effective conductance can be expressed analytically as the harmonic mean of a_i's:

$$A_{\text{eff}}(a) = \left(\frac{1}{n} \sum_{i=1}^{n} \frac{1}{a_i} \right)^{-1}. \tag{4.8}$$

This function indeed approximately corresponds to the deep neural network shown in Figure 5. The neural network is separated into three stages. In the first stage, the approximation to function $1/a_i$ is constructed for each a_i by applying a few convolution layers with size 1 kernel window. In this stage, the channel size for these convolution layers is chosen to be 16 except the last layer since the output of the first stage should be a vector of size n. In the second stage, a layer of sum-pooling with size n window is used to perform the summation in (4.8), giving a scalar output.

Table 3. *Error in approximating the lowest energy level $E_0(V)$ for $n = 8, 16$ discretisation using the architecture in Figure 5.*

n	α	Training error	Validation error	Average E_0	No. of samples	No. of param.
8	5	4.9×10^{-4}	5.0×10^{-4}	10.48 ± 0.51	4800	331
16	5	1.5×10^{-4}	1.5×10^{-4}	10.46 ± 0.27	1.05×10^4	1291

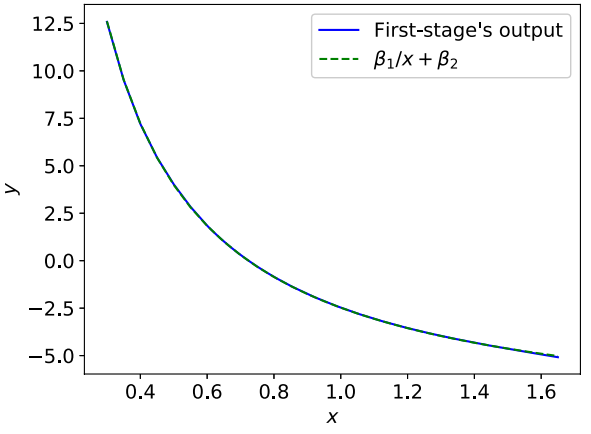

FIGURE 6. The first stage's output of the neural network in Figure 5 fitted by $\beta_1/x + \beta_2$.

The third and first stages have the exact same architecture except the input to the third stage is a scalar. For training 2560 samples are used and another 2560 samples are used for validation. We let $a_i \sim \mathcal{U}[0.3, 1.5]$, giving an effective conductance of 0.77 ± 0.13 for $n = 8$. A validation error of 4.9×10^{-4} is obtained with the neural network in Figure 5, while with the network in Figure 4 the accuracy is 5.5×10^{-3} with $\alpha = 16$. As a sanity check, Figure 6 shows that the output from the first stage is well fitted by the reciprocal function.

We remark that although incorporating domain knowledge in PDE to build a sophisticated neural network architecture would likely boost the approximation quality, such as what we do in the constructive proof for Theorem 1, even a simple network as in Figure 4 can already give decent results.

4.3.2 *Ground state energy of NLSE*

We next focus on the 2D case in the NLSE example. The goal here is to obtain a neural network parametrisation for $E_0(a)$, with input now being $a \in \mathbb{R}^{n^2}$ with i.i.d. entries distributed according to $\mathcal{U}[1, 16]$. In order to generate training samples, for each realisation of a, the nonlinear eigenvalue problem (2.9) subject to the normalisation constraint (2.8) is solved by a homotopy method. First, the case $\sigma = 0$ is solved as a standard eigenvalue problem. Then σ is changed from 0 to 2 with a step size equal to 0.4. For each σ, Newton's method is used to solve the NLSE for $u_a(x)$ and

$E_0(a)$, using $u_a(x)$ and $E_0(a)$ corresponding to the previous σ value as initialisation. The results are presented in Table 3.

5 Conclusion

In this note, we present a method based on deep neural network to solve PDE with inhomogeneous coefficient fields. Physical quantities of interest are learned as a function of the coefficient field. Based on the time-evolution technique for solving PDE, we provide theoretical motivation to represent these quantities using an NN. The numerical experiments on diffusion equation and NLSE show the effectiveness of simple convolutional neural network in parametrising such function to 10^{-3} accuracy. We remark that while many questions should be asked, such as what is the best network architecture and what situations can this approach handle, the goal of this short note is simply to suggest neural network as a promising tool for model reduction when solving PDE with uncertainties.

Conflict of interest

None.

References

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, J., Levenberg, M., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. & Zheng, X. (2016) Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467.

[2] Carleo, G. & Troyer, M. (2017) Solving the quantum many-body problem with artificial neural networks. *Science* **355**(6325), 602–606.

[3] Cheng, M., Hou, T. Y., Yan, M. & Zhang, Z. (2013) A data-driven stochastic method for elliptic PDEs with random coefficients. *SIAM/ASA J. Uncertainty Quant.* **1**(1), 452–493.

[4] Chollet, F. (2017) Keras (2015). http://keras.io.

[5] Dozat, T. (2016) Incorporating Nesterov momentum into ADAM. In: *Proceedings of the ICLR Workshop*.

[6] Han, J., Jentzen, A. & Weinan E. (2017) Overcoming the curse of dimensionality: solving high-dimensional partial differential equations using deep learning. arXiv preprint arXiv:1707.02568.

[7] He, K., Zhang, X., Ren, S. & Sun, J. (2016) Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778.

[8] Hinton, G. E. & Salakhutdinov, R. R. (2006) Reducing the dimensionality of data with neural networks. *Science* **313**(5786), 504–507.

[9] Khoo, Y., Lu, J. & Ying, L. (2018) Solving for high dimensional committor functions using artificial neural networks. arXiv preprint arXiv:1802.10275.

[10] Lagaris, I. E., Likas, A. & Fotiadis, D. I. (1998) Artificial neural networks for solving ordinary and partial differential equations. *IEEE Trans. Neural Networks* **9**(5), 987–1000.

[11] LeCun, Y., Bengio, Y. & Hinton, G. (2015) Deep learning. *Nature* **521**(7553), 436–444.

[12] Long, Z., Lu, Y., Ma, X. & Dong, B. (2017) PDE-net: learning PDEs from data. arXiv preprint arXiv:1710.09668.

[13] MATTHIES, H. G. & KEESE, A. (2005) Galerkin methods for linear and nonlinear elliptic stochastic partial differential equations. *Comput. Methods Appl. Mecha. Eng.* **194**(12), 1295–1331.
[14] RUDD, K. & FERRARI, S. (2015) A constrained integration (CINT) approach to solving partial differential equations using artificial neural networks. *Neurocomputing* **155**, 277–285.
[15] SCHMIDHUBER, J. (2015) Deep learning in neural networks: an overview. *Neural Networks* **61**, 85–117.
[16] STEFANOU, G. (2009) The stochastic finite element method: past, present and future. *Comput. Methods Appl. Mecha. Eng.* **198**(9), 1031–1051.
[17] TORLAI, G. & MELKO, R. G. (2016) Learning thermodynamics with Boltzmann machines. *Phys. Rev. B* **94**(16), 165134.
[18] WIENER, N. (1938) The homogeneous chaos. *Am. J. Math.* **60**(4), 897–936.
[19] XIU, D. & KARNIADAKIS, G. E. (2002) The Wiener–Askey polynomial chaos for stochastic differential equations. *SIAM J. Sci. Comput.* **24**(2), 619–644.
[20] XU, J. & ZIKATANOV, L. (2017) Algebraic multigrid methods. *Acta Numerica* **26**, 591–721.

A Appendix

A.1 Proof of representability of effective conductance by NN

As mentioned previously in Section 3, the first step of constructing an NN to represent the effective conductance is to perform time-evolution iterations in the form of (3.4). However, since at each step we need to approximate the map from u^m to u^{m+1} in (3.4) using NN, the process of time-evolution is similar to applying noisy gradient descent on $\mathcal{E}(u;a)$. More precisely, after performing a step of gradient descent update, the NN approximation incurs noise to the update, i.e.

$$v^0 = u^0 = 0, \quad u^{m+1} = v^m - \Delta t \nabla \mathcal{E}(v^m), \quad v^{m+1} = u^{m+1} + \Delta t \varepsilon^{m+1}. \tag{A.1}$$

Here $\mathcal{E}(u;a)$ is abbreviated as $\mathcal{E}(u)$, and ε^{m+1} is the error for each layer of the NN in approximating each exact time-evolution iteration u^{m+1}.

Now let the spectral norm of L_a and $L_a^\dagger$ satisfy

$$\|L_a\|_2 \le \lambda_a, \quad \|L_a^\dagger\|_2 \le 1/\mu_a, \tag{A.2}$$

and for the case considered, $\lambda_a = O(\lambda_1 n^2)$ and $\mu_a = \Omega(\lambda_0)$. Assuming

$$\|\varepsilon^{m+1}\|_2 \le c\|\nabla \mathcal{E}(v^m)\|_2, \quad \mathbf{1}^\top \varepsilon^{m+1} = 0, \quad m = 0, \dots M-1, \tag{A.3}$$

the following lemma can be obtained.

Lemma 1 *The iterations in* (3.4) *satisfy*

$$\mathcal{E}(v^{m+1}) - \mathcal{E}(v^m) \le -\frac{\Delta t}{2}\|\nabla \mathcal{E}(v^m)\|_2^2, \tag{A.4}$$

if $\Delta t \le \delta, \delta = \left(1 - \frac{1}{2(1-c)}\right)\frac{2}{\lambda_a'}, \lambda_a' = (1 + \frac{c^2}{1-c})\lambda_a$. *Furthermore,*

$$\frac{\Delta t}{2}\sum_{m=0}^{M-1}\|\nabla \mathcal{E}(v^{m+1})\|_2^2 \le \mathcal{E}(v^0) - \mathcal{E}(v^M) \le \mathcal{E}(v^0). \tag{A.5}$$

Proof From Lipschitz property of $\nabla\mathcal{E}(u)$ (A.2),

$$\begin{aligned}
\mathcal{E}(v^{m+1}) - \mathcal{E}(v^m) &\le \langle \nabla\mathcal{E}(v^m), v^{m+1} - v^m\rangle + \frac{\lambda_a}{2}\|v^{m+1} - v^m\|_2^2 \\
&= \langle \nabla\mathcal{E}(v^m), v^m - \Delta t(\nabla\mathcal{E}(v^m) + \varepsilon^{m+1}) - v^m\rangle \\
&\quad + \frac{\lambda_a}{2}\|v^m - \Delta t(\nabla\mathcal{E}(v^m) + \varepsilon^{m+1}) - v^m\|_2^2 \\
&= -\Delta t(1 - \frac{\Delta t\lambda_a}{2})\|\nabla\mathcal{E}(v^m)\|_2^2 \\
&\quad + \Delta t(1 - \frac{\Delta t\lambda_a}{2})\langle \varepsilon^{m+1}, \nabla\mathcal{E}(v^m)\rangle + \frac{\lambda_a\Delta t^2}{2}\|\varepsilon^m\|_2^2 \\
&\le -\Delta t(1 - \frac{\Delta t\lambda_a}{2})\|\nabla\mathcal{E}(v^m)\|_2^2 \\
&\quad + c\Delta t(1 - \frac{\Delta t\lambda_a}{2} + \frac{c\Delta t\lambda_a}{2})\|\nabla\mathcal{E}(v^m)\|_2^2 \\
&= -\Delta t\big((1-c) - (1 - c + c^2)\frac{\Delta t\lambda_a}{2}\big)\|\nabla\mathcal{E}(v^m)\|_2^2 \\
&= -\Delta t(1-c)\big(1 - \frac{1 - c + c^2}{1-c}\frac{\Delta t\lambda_a}{2}\big)\|\nabla\mathcal{E}(v^m)\|_2^2 \\
&= -\Delta t(1-c)\big(1 - \frac{\Delta t\lambda_a'}{2}\big)\|\nabla\mathcal{E}(v^m)\|_2^2. \qquad \text{(A.6)}
\end{aligned}$$

Letting $\Delta t \le \big(1 - \frac{1}{2(1-c)}\big)\frac{2}{\lambda_a'}$, we get

$$\mathcal{E}(v^{m+1}) - \mathcal{E}(v^m) \le -\frac{\Delta t}{2}\|\nabla\mathcal{E}(v^m)\|_2^2. \qquad \text{(A.7)}$$

Summing the LHS and RHS and using the fact that $\mathcal{E}(u) \ge 0$ give (A.5). This concludes the lemma. □

Theorem 2 *If Δt satisfies the condition in Lemma 1, given any $\epsilon > 0$, $|\mathcal{E}(v^M) - \mathcal{E}(v)| \le \epsilon$ for $M = O((\lambda_1^2 + \frac{\lambda_1^2}{\lambda_0} + \lambda_1)\frac{n^2}{\epsilon})$.*

Proof Since by convexity

$$\mathcal{E}(u^*) - \mathcal{E}(v^m) \ge \langle \nabla\mathcal{E}(v^m), u^* - v^m\rangle, \qquad \text{(A.8)}$$

along with Lemma 1,

$$\begin{aligned}
\mathcal{E}(v^{m+1}) &\le \mathcal{E}(u^*) + \langle \nabla\mathcal{E}(v^m), v^m - u^*\rangle - \frac{\Delta t}{2}\|\nabla\mathcal{E}(v^m)\|_2^2 \\
&= \mathcal{E}(u^*) + \frac{1}{2\Delta t}\big(2\Delta t\langle \nabla\mathcal{E}(v^m), v^m - u^*\rangle - \Delta t^2\|\nabla\mathcal{E}(v^m)\|_2^2 \\
&\quad + \|v^m - u^*\|_2^2 - \|v^m - u^*\|_2^2\big) \\
&= \mathcal{E}(u^*) + \frac{1}{2\Delta t}(\|v^m - u^*\|_2^2 - \|v^m - \Delta t\nabla\mathcal{E}(v^m) - u^*\|_2^2) \\
&= \mathcal{E}(u^*) + \frac{1}{2\Delta t}(\|v^m - u^*\|_2^2 - \|v^{m+1} - \Delta t\varepsilon^{m+1} - u^*\|_2^2) \\
&= \mathcal{E}(u^*) + \frac{1}{2\Delta t}(\|v^m - u^*\|_2^2 - \|v^{m+1} - u^*\|_2^2 \\
&\quad + 2\Delta t\langle \varepsilon^{m+1}, v^{m+1} - u^*\rangle - \Delta t^2\|\varepsilon^{m+1}\|_2^2) \\
&= \mathcal{E}(u^*) + \frac{1}{2\Delta t}\big(\|v^m - u^*\|_2^2 - \|v^{m+1} - u^*\|_2^2 + \Delta t^2\|\varepsilon^{m+1}\|_2^2
\end{aligned}$$

$$
\begin{aligned}
&+ 2\Delta t\langle \varepsilon^{m+1}, v^m - u^*\rangle - 2\Delta t\langle \varepsilon^{m+1}, \nabla\mathcal{E}(v^m)\rangle\big) \\
&\le \mathcal{E}(u^*) + \frac{1}{2\Delta t}\big(\|v^m - u^*\|_2^2 - \|v^{m+1} - u^*\|_2^2 + \Delta t^2\|\varepsilon^{m+1}\|_2^2 \\
&\quad + 2\Delta t\|\varepsilon^{m+1}\|_2(\|v^m - u^*\|_2 + \|\nabla\mathcal{E}(v^m)\|_2)\big) \\
&\le \mathcal{E}(u^*) + \frac{1}{2\Delta t}\big(\|v^m - u^*\|_2^2 - \|v^{m+1} - u^*\|_2^2 + \Delta t^2\|\varepsilon^{m+1}\|_2^2 \\
&\quad + 2\Delta t(1 + 2/\mu_a)\|\varepsilon^{m+1}\|_2\|\nabla\mathcal{E}(v^m)\|_2\big) \\
&\le \mathcal{E}(u^*) + \frac{1}{2\Delta t}\big(\|v^m - u^*\|_2^2 - \|v^{m+1} - u^*\|_2^2 + c^2\Delta t^2\|\nabla\mathcal{E}(v^m)\|_2^2 \\
&\quad + 2c(1 + 2/\mu_a)\Delta t\|\nabla\mathcal{E}(v^m)\|_2^2\big). \qquad \text{(A.9)}
\end{aligned}
$$

The last second inequality follows from (A.2), which implies $\|L_a u\|_2 \ge \mu_a\|u\|_2$ if $u^\top\mathbf{1} = 0$. More precisely, the fact that $v^0 = 0, \nabla\mathcal{E}(u)^\top\mathbf{1} = 0$ (follows from the form of L_a and $b_{\xi,a}$ defined in (2.5)), and $\varepsilon^{m\top}\mathbf{1} = 0\ \forall m$ (due to the assumption in (A.3)) implies $v^{m\top}\mathbf{1} = 0$, hence $\frac{\mu_a}{2}\|v^m - u^*\|_2 \le \|\nabla\mathcal{E}(v^m) - \nabla\mathcal{E}(u^*)\|_2 = \|\nabla\mathcal{E}(v^m)\|_2$. Reorganising (A.9) we get

$$
\mathcal{E}(v^{m+1}) - \mathcal{E}(u^*) \le \frac{1}{2\Delta t}\left(\|v^m - u^*\|_2^2 - \|v^{m+1} - u^*\|_2^2 + c\Delta t\big(c\Delta t + 2(1 + \frac{2}{\mu_a})\big)\|\nabla\mathcal{E}(v^m)\|_2^2\right). \qquad \text{(A.10)}
$$

Summing both left- and right-hand sides results in

$$
\begin{aligned}
\mathcal{E}(v^M) - \mathcal{E}(u^*) &\le \frac{1}{M}\sum_{m=0}^{M-1}\mathcal{E}(v^{m+1}) - \mathcal{E}(u^*) \\
&\le \frac{1}{M}\left[\left(\frac{\|v^0 - u^*\|_2^2}{2\Delta t}\right) + \frac{2c}{\Delta t}\left(c\Delta t + 2(1 + \frac{2}{\mu_a})\right)\mathcal{E}(v^0)\right], \qquad \text{(A.11)}
\end{aligned}
$$

where the second inequality follows from (A.5). In order to derive a bound for $\|v^0 - u^*\|_2^2$, we appeal to strong convexity property of $\mathcal{E}(u)$:

$$
\mathcal{E}(v^0) - \mathcal{E}(u^*) \ge \langle\nabla\mathcal{E}(u^*), v^m - u^*\rangle + \frac{\mu_a}{2}\|v^0 - u^*\|_2^2 = \frac{\mu_a}{2}\|v^0 - u^*\|_2^2 \qquad \text{(A.12)}
$$

for $\langle\mathbf{1}, v^0 - u^*\rangle = 0$. Then

$$
\mathcal{E}(v^M) - \mathcal{E}(u^*) \le \frac{1}{M}\left[\left(\frac{1}{\mu_a\Delta t}\right) + \frac{2c}{\Delta t}\left(c\Delta t + 2(1 + \frac{2}{\mu_a})\right)\right]\mathcal{E}(v^0). \qquad \text{(A.13)}
$$

Since $\mathcal{E}(v^0) = \frac{a^\top\mathbf{1}}{2n^d} = O(\lambda_1)$, along with $\lambda_a = O(\lambda_1 n^2)$, $\mu_a = \Omega(\lambda_0)$, we establish the claim. □