SELECTNET: SELF-PACED LEARNING FOR HIGH-DIMENSIONAL PARTIAL DIFFERENTIAL EQUATIONS

YIQI GU

DEPARTMENT OF MATHEMATICS, NATIONAL UNIVERSITY OF SINGAPORE, 10 LOWER KENT RIDGE ROAD, SINGAPORE, 119076 (MATGUY@NUS.EDU.SG)

HAIZHAO YANG*

DEPARTMENT OF MATHEMATICS, PURDUE UNIVERSITY, WEST LAFAYETTE, IN 47907, USA (HAIZHAO@PURDUE.EDU)

CHAO ZHOU

DEPARTMENT OF MATHEMATICS, NATIONAL UNIVERSITY OF SINGAPORE, 10 LOWER KENT RIDGE ROAD, SINGAPORE, 119076 (MATZC@NUS.EDU.SG)

Abstract. The least squares method with deep neural networks as function parametrization has been applied to solve certain high-dimensional partial differential equations (PDEs) successfully; however, its convergence is slow and might not be guaranteed even within a simple class of PDEs. To improve the convergence of the network-based least squares model, we introduce a novel self-paced learning framework, SelectNet, which quantifies the difficulty of training samples, treats samples equally in the early stage of training , and slowly explores more challenging samples, e.g., samples with larger residual errors, mimicking the human cognitive process for more efficient learning. In particular, a selection network and the PDE solution network are trained simultaneously; the selection network adaptively weighting the training samples of the solution network achieving the goal of self-paced learning. Numerical examples indicate that the proposed SelectNet model outperforms existing models on the convergence speed and the convergence robustness, especially for low-regularity solutions.

Key words. High-Dimensional PDEs; Deep Neural Networks; Self-Paced Learning; Selected Sampling; Least Square Method; Convergence.

AMS subject classifications. 65M75; 65N75; 62M45;

1. Introduction.

High-dimensional partial differential equations (PDEs) are important tools in physical, financial, and biological models [41, 20, 66, 22, 63]. However, developing numerical methods for high-dimensional PDEs has been challenging due to the curse of dimensionality in the discretization of the problem. For example, in traditional methods such as finite difference methods and finite element methods, $O(N^d)$ degree of freedom is required for a d-dimensional problem if we set N grid points or basis functions in each direction to achieve $O(\frac{1}{N})$ accuracy. Even if d becomes moderately large, the exponential growth N^d in the dimension d makes traditional methods immediately computationally intractable.

Recent research of the approximation theory of deep neural networks (DNNs) shows that deep network approximation is a powerful tool for mesh-free function parametrization. The research on the approximation theory of neural networks traces back to the pioneering work [9, 26, 1] on the universal approximation of shallow networks with sigmoid activation functions. The recent research focus was on the approximation rate of DNNs for various function spaces in terms of the number of network parameters showing that deep networks are more powerful than shallow networks in approximation efficiency. For example, smooth functions [46, 44, 64, 18, 49, 62, 16, 15, 17], piecewise smooth functions [53], band-limited functions [51], continuous functions [65, 57, 56]. The reader is referred to [56] for the explicit characterization of the approximation error for networks with an arbitrary width and depth.

*CORRESPONDING AUTHOR.

In particular, deep network approximation can lessen or overcome the curse of dimensionality under certain circumstances, making it an attractive tool for solving high-dimensional problems. For functions admitting an integral representation with a one-dimensional integral kernel, no curse of dimensionality in the approximation rate can be shown via establishing the connection of network approximation with the Monte Carlo sampling or equivalently the law of large numbers [1, 16, 15, 17, 51]. Based on the Kolmogorov-Arnold superposition theorem, for general continuous functions, [47, 24] showed that three-layer neural networks with advanced activation functions can avoid the curse of dimensionality and the total number of parameters required is only $O(d)$; [50] proves that deep ReLU network approximation can lessen the curse of dimensionality, if target functions are restricted to a space related to the constructive proof of the Kolmogorov-Arnold superposition theorem in [4]. If the approximation error is only concerned on a low-dimensional manifold, there is no curse of dimensionality for deep network approximation in terms of the approximation error [7, 5, 56]. Finally, there is also extensive research showing that deep network approximation can overcome the curse of dimensionality when they are applied to approximation certain PDE solutions, e.g. [27, 29].

As an efficient function parametrization tool, neural networks have been applied to solve PDEs via various approaches. Early work in [40] applies neural networks to approximate PDE solutions defined on grid points. Later in [11, 38], DNNs are employed to approximate solutions in the whole domain, and PDEs are solved by minimizing the discrete residual error in the L^2-norm at prescribed collocation points. DNNs coupled with boundary governing terms by design can satisfy boundary conditions [48]. Nevertheless, designing boundary governing terms is usually difficult for complex geometry. Another approach to enforcing boundary conditions is to add boundary errors to the loss function as a penalized term and minimize it as well as the PDE residual error [23, 39]. The second technique is in the same spirit of least squares methods in finite element methods and is more convenient in implementation. Therefore, it has been widely utilized for PDEs with complex domains. However, network computation was usually expensive, limiting the applications of network-based PDE solvers. Thanks to the development of GPU-based parallel computing over the last two decades, which greatly boosts the network computation, network-based PDE solvers were revisited recently and have become a popular tool, especially for high-dimensional problems [13, 19, 25, 34, 60, 3, 67, 42, 2, 30, 28, 6, 55, 43]. Nevertheless, most network-based PDE solvers suffer from robustness issues: their convergence is slow and might not be guaranteed even within a simple class of PDEs.

To ease the issue above, we introduce a novel self-paced learning framework, SelectNet, to adaptively choose training samples in the least squares model. Self-paced learning [36] is a recently raised learning technique that can choose a part of the training samples for actual training over time. Specifically, for a training data set with n samplings, self-paced learning uses a vector $v \in \{0,1\}^n$ to indicate whether each training sample should be included in the current training stage. The philosophy of self-paced learning is to simulate human beings' learning style, which tends to learn easier aspects of a learning task first and deal with more complicated samples later. Based on self-paced learning, a novel technique for selected sampling is put forward, which uses a selection neural network instead of the 0-1 selection vector v. Hence, it learns to avoid redundant training information and speeds up the convergence of learning outcomes. This idea is further improved in [31] by introducing a DNN to select training data for image classification. Among similar works, a state-of-the-art algorithm named SelectNet is proposed in [45] for image classification, especially for imbalanced data problems. Based on the observation that samples near the singularity of the PDE solution are rare compared to samples from the regular part, we extend the SelectNet [45] to network-based least squares models, especially for PDE solutions with certain irregularity. As we shall see later, numerical results show that the proposed model is competitive

with the traditional (basic) least squares model for analytic solutions, and it outperforms others for low-regularity solutions, in the aspect of the convergence speed. It is worth noting that our proposed SelectNet model is essentially tuning the weights of training points to realize the adaptive sampling. Another approach is to change the distribution of training points, such as the residual-based adaptive refinement method [33].

The organization of this paper is as follows. In Section 2, we introduce the least squares methods and formulate the corresponding optimization model. In Section 3, we present the SelectNet model in detail. In Section 4, we put forward the error estimates of the basic and SelectNet models. In Section 5, we discuss the network implementation in the proposed model. In Section 6, we present ample numerical experiments for various equations to validate our model. We conclude with some remarks in the final section.

2. Least Squares Methods for PDEs. In this work, we aim at solving the following (initial) boundary value problems, giving a bounded domain $\Omega \subset \mathbb{R}^d$:

- elliptic equations

$$\begin{aligned} &\mathcal{D}_x u(x) = f(x), \text{ in } \Omega, \\ &\mathcal{B}_x u(x) = g_0(x), \text{ on } \partial\Omega; \end{aligned} \tag{2.1}$$

- parabolic equations

$$\begin{aligned} &\frac{\partial u(x,t)}{\partial t} - \mathcal{D}_x u(x,t) = f(x,t), \text{ in } \Omega \times (0,T), \\ &\mathcal{B}_x u(x,t) = g_0(x,t), \text{ on } \partial\Omega \times (0,T), \\ &u(x,0) = h_0(x), \text{ in } \Omega; \end{aligned} \tag{2.2}$$

- hyperbolic equations

$$\begin{aligned} &\frac{\partial^2 u(x,t)}{\partial t^2} - \mathcal{D}_x u(x,t) = f(x,t), \text{ in } \Omega \times (0,T), \\ &\mathcal{B}_x u(x,t) = g_0(x,t), \text{ on } \partial\Omega \times (0,T), \\ &u(x,0) = h_0(x), \quad \frac{\partial u(x,0)}{\partial t} = h_1(x) \text{ in } \Omega; \end{aligned} \tag{2.3}$$

where u is the solution function; f, g_0, h_0, h_1 are given data functions; $\mathcal{D}_x$ is a spatial differential operator concerning the derivatives of x; $\mathcal{B}_x$ is a boundary operator specifying a Dirichlet, Neumann or Robin boundary condition.

In this method, the temporal variable t will be regarded as an extra spatial coordinate, and it will not be dealt with differently from x. For simplicity, the PDEs in (2.1)-(2.3) are unified in the following form

$$\begin{aligned} &\mathcal{D}u(\boldsymbol{x}) = f(\boldsymbol{x}), \text{ in } Q, \\ &\mathcal{B}u(\boldsymbol{x}) = g(\boldsymbol{x}), \text{ in } \Gamma, \end{aligned} \tag{2.4}$$

where $\boldsymbol{x}$ includes the spatial variable x and possibly the temporal variable t; $\mathcal{D}u = f$ represents a generic PDE; $\mathcal{B}u = g$ represents the governing conditions including the boundary condition and possibly the initial condition; Q and Γ are the corresponding domains of the equations.

Now we seek a neural network $u(\boldsymbol{x};\theta)$ approximating the solution $u(\boldsymbol{x})$ of the PDE (2.4). Note the residual errors for the PDE and the governing conditions can be written by

$$\mathcal{R}_Q(u(\boldsymbol{x};\theta)) := \mathcal{D}u(\boldsymbol{x};\theta) - f(\boldsymbol{x}), \quad \mathcal{R}_\Gamma(u(\boldsymbol{x};\theta)) := \mathcal{B}u(\boldsymbol{x};\theta) - g(\boldsymbol{x}). \tag{2.5}$$

One can solve the PDE by searching for the optimal parameters of the network that minimize these residual errors, i.e.

$$\min_{\theta} \|\mathcal{R}_Q(u(\boldsymbol{x};\theta))\|_Q^2 + \lambda\|\mathcal{R}_\Gamma(u(\boldsymbol{x};\theta))\|_\Gamma^2, \tag{2.6}$$

where $\|\cdot\|_*$ is usually the L^2-norm and λ is a parameter for weighting the sum, e.g.,

$$\min_{\theta} \mathbb{E}_{\boldsymbol{x}\in Q}\left[|\mathcal{D}u(\boldsymbol{x};\theta) - f(\boldsymbol{x})|^2\right] + \lambda\mathbb{E}_{\boldsymbol{x}\in\Gamma}\left[|\mathcal{B}u(\boldsymbol{x};\theta) - g(\boldsymbol{x})|^2\right]. \tag{2.7}$$

3. SelectNet Model. The network-based least squares model has been applied to solve certain high-dimensional PDEs successfully. However, its convergence is slow and might not be guaranteed. To ease this issue, we introduce a novel self-paced learning framework, SelectNet, to adaptively choose training samples in the least squares model. The basic philosophy is to mimic the human cognitive process for more efficient learning: learning first from easier examples and slowly exploring more complicated ones. The proposed model is related to selected sampling [8, 32], an important tool of deep learning for computer science applications. Nevertheless, the effectiveness of selected sampling in scientific computing has not been fully explored yet.

In particular, a selection network $\phi_{\rm s}(\boldsymbol{x};\theta_{\rm s})$ (subscript s for "selection") and the PDE solution network $u(\boldsymbol{x};\theta)$ are trained simultaneously; the selection network adaptively weighting the training samples of the solution network achieving the goal of self-paced learning. $\phi_{\rm s}(\boldsymbol{x};\theta_{\rm s})$ is a "mentor" helping to decide whether a sample $\boldsymbol{x}$ is important enough to train the "student" network $u(\boldsymbol{x};\theta)$. The "mentor" could avoid redundant training information and help to speed up the convergence. This idea is originally from self-paced learning [37] and is further improved in [31] by introducing a DNN to select training data for image classification. Among similar works, a state-of-the-art algorithm named SelectNet was proposed in [45] for image classification, especially for imbalanced data problem. Based on the observation that samples near the singularity of the PDE solution are rare compared to samples from the regular part, we extend the SelectNet [45] to network-based least squares models, especially for PDE solutions with certain irregularity.

Originally in image classification, for a training data set $\mathcal{D} = \{(\boldsymbol{x}_i, y_i))\}_{i=1}^n$, self-paced learning uses a vector $\boldsymbol{v} \in \{0,1\}^n$ to indicate whether or not each training sample should be included in the current training stage ($v_i = 1$ if the ith sample is included in the current iteration). The overall target function including $\boldsymbol{v}$ is

$$\min_{\theta,\boldsymbol{v}\in\{0,1\}^n} \sum_{i=1}^n v_i\mathcal{L}(y_i, \phi(\boldsymbol{x}_i;\theta)) - \lambda\sum_{i=1}^n v_i, \tag{3.1}$$

where $\mathcal{L}(y_i, \phi(\boldsymbol{x}_i;\theta))$ denotes the loss function of a DNN $\phi(\boldsymbol{x}_i;\theta)$ for classifying a sample $\boldsymbol{x_i}$ to y_i. When this model is relaxed to $\boldsymbol{v} \in [0,1]^n$ and the alternative convex search is applied to solve the relaxed optimization, a straightforward derivation easily reveals a rule for the optimal value for each entry $v_i^{(t)}$ in the t-th iteration as

$$v_i^{(t)} = 1, \text{ if } \mathcal{L}(y_i, \phi(\boldsymbol{x}_i;\theta^{(t)})) < \lambda, \qquad \text{and} \qquad v_i^{(t)} = 0, \text{ otherwise.} \tag{3.2}$$

A sample with a smaller loss than the threshold λ is treated as an "easy" sample and will be selected in training. Let us assume that the variables $\boldsymbol{v}$ and θ are trained alternatively. When computing $\theta^{(t+1)}$ with a fixed $\boldsymbol{v}^{(t)}$, the classifier is trained only on the selected "easy" samples. When computing $\boldsymbol{v}^{(t+1)}$ with a fixed $\theta^{(t+1)}$, the vector $\boldsymbol{v}$ help to adjust the training samples to be used in computing $\theta^{(t+2)}$. It was shown by extensive numerical experiments that this mechanism helps to reduce the generalization error for image classification when the training data distribution is usually different

from the test data distribution [37]. In [31, 45], a selection network $\phi_s(\boldsymbol{x};\theta_s) \in [0,1]$ is trained to select training samples instead of using the binary vector $\boldsymbol{v}$ with the following loss function:

$$\min_{\theta,\theta_s} \sum_{i=1}^{n} \phi_s(\boldsymbol{x}_i;\theta_s)\mathcal{L}(y_i,\phi(\boldsymbol{x}_i;\theta)) - \lambda \sum_{i=1}^{n} \phi_s(\boldsymbol{x}_i;\theta_s). \tag{3.3}$$

The introduction of the selection network has mainly three advantages. First, it changes the discrete optimization problem in (3.1) to a continuous optimization problem in (3.3) that is much easier to solve. Besides, the selection network with values in $[0,1]$ can more adaptive adjust the weights to each sample. Finally, the number of parameters in the selection network can be much smaller than the size of $\boldsymbol{v}$, since usually a small selection network is good enough to decide weights roughly.

The self-paced idea can also be applied to the preceding least squares model for solving PDEs. One naive way is to rewrite the optimization (2.7) as

$$\min_{\theta} \frac{1}{N_1}\sum_{i=1}^{N_1} v_i'|\mathcal{D}u(\boldsymbol{x}_i^1;\theta) - f(\boldsymbol{x}_i^1)|^2 + \frac{\lambda}{N_2}\sum_{i=1}^{N_2} v_j''|\mathcal{B}u(\boldsymbol{x}_i^2;\theta) - g(\boldsymbol{x}_i^2)|^2, \tag{3.4}$$

where $\{\boldsymbol{x}_i^1\}_{i=1}^{N_1} \subset \Omega$ and $\{\boldsymbol{x}_i^2\}_{i=1}^{N_2} \subset \partial\Omega$ are random samples; v_i' and v_i'' are adaptive binary weights denoting if the samples are selected or not in the loss. Similar adaptive sampling techniques can be found in [52, 14]. Solving PDEs using deep learning is different from conventional supervised learning, where sample data are fixed without the flexibility to be arbitrary in the problem domain. The training and testing data distributions are the same, and there is no limitation for sampling when we solve PDEs. Therefore, appropriately selecting training data and assigning weights $\boldsymbol{v}'$ and $\boldsymbol{v}''$ in each optimization iteration can better facilitate the convergence of deep learning to the true PDE solution.

Intuitively, a good strategy is to first choose "easy" samples to quickly identify a rough PDE solution and then use more "difficult" samples with large residual errors to refine the PDE solution. For example, in the early stage of the training, random samples are uniformly drawn in the PDE domain; in the latter stage of the training, we can select samples with almost the highest residual errors for training. However, this naive selection strategy might be too greedy: large residual errors usually occur where the PDE solution is irregular (e.g., near low regularity points), resulting in selected training samples gathering around these "difficult" points with few samples in other regions. Note that deep neural networks are functions globally supported in the PDE domain. Training with samples restricted in a small area may lead to large test errors in other areas. In our experiments, we observe that this naive selection strategy applied to (3.4) even works worse than the basic model (2.7) (See the numerical example in Section 6.1.2).

Borrowing the idea in [31, 45], we introduce two neural networks, $\phi_{\mathrm{s}}'(\boldsymbol{x};\theta_{\mathrm{s}}')$ and $\phi_{\mathrm{s}}''(\boldsymbol{x};\theta_{\mathrm{s}}'')$, named as the selection network for the PDE residual error and the boundary condition error, respectively, to replace $\boldsymbol{v}'$ and $\boldsymbol{v}''$ in (3.4). The introduction of selection networks admits three main advantages over the naive binary weights, as discussed previously for the models in (3.1) and (3.3). According to the discussion in the last paragraph, the selection networks $\phi_{\mathrm{s}}'(\boldsymbol{x};\theta_{\mathrm{s}}')$ and $\phi_{\mathrm{s}}''(\boldsymbol{x};\theta_{\mathrm{s}}'')$ should satisfy the following requirements. 1) As weight functions, they are required to be non-negative and bounded. 2) They should not have a strong bias for weighting samples in the early stage of training. 3) They prefer higher weights for samples with larger point-wise residual errors in the latter stage of training.

For the first requirement, $\phi_{\mathrm{s}}'(\boldsymbol{x};\theta_{\mathrm{s}}')$ and $\phi_{\mathrm{s}}''(\boldsymbol{x};\theta_{\mathrm{s}}'')$ are enforced to satisfy

$$m_0 < \phi_{\mathrm{s}}'(\boldsymbol{x};\theta_{\mathrm{s}}') < M_0, \quad \forall \boldsymbol{x} \in Q \text{ and } \forall \theta_{\mathrm{s}}', \tag{3.5}$$

$$m_0 < \phi_{\mathrm{s}}''(\boldsymbol{x};\theta_{\mathrm{s}}'') < M_0, \quad \forall \boldsymbol{x} \in \Gamma \text{ and } \forall \theta_{\mathrm{s}}'', \tag{3.6}$$

where $M_0 > 1 > m_0 \geq 0$ are prescribed constants. Note the conditions (3.5)-(3.6) hold automatically if the last layer of activation functions of $\phi'_{\rm s}(\boldsymbol{x};\theta'_{\rm s})$ and $\phi''_{\rm s}(\boldsymbol{x};\theta''_{\rm s})$ is bounded (e.g., using a tanh or sigmoid activation function) and the network output is properly re-scaled and shifted as we shall discuss later in the next section. Therefore, the corresponding weighted least squares method is formulated by

$$\mathbb{E}_{x\in Q}\left[\phi'_{\rm s}(\boldsymbol{x};\theta'_{\rm s})|\mathcal{D}u(\boldsymbol{x};\theta)-f(\boldsymbol{x})|^2\right]+\lambda\mathbb{E}_{x\in\Gamma}\left[\phi''_{\rm s}(\boldsymbol{x};\theta''_{\rm s})|\mathcal{B}u(\boldsymbol{x};\theta)-g(\boldsymbol{x})|^2\right]. \tag{3.7}$$

For the second requirement, when the selection networks are randomly initialized with zero bias and random weights with a zero mean and a small variance, the selection networks are random functions close to a constant. Therefore, the selection networks have no bias in weighting samples in the early stage of training.

The third requirement can also be satisfied. Based on the principle that higher weights should be added to samples with larger point-wise residual errors, we can train $\phi'_{\rm s}(\boldsymbol{x};\theta'_{\rm s})$ and $\phi''_{\rm s}(\boldsymbol{x};\theta''_{\rm s})$ via

$$\max_{\theta'_{\rm s},\theta''_{\rm s}}\mathbb{E}_{x\in Q}\left[\phi'_{\rm s}(\boldsymbol{x};\theta'_{\rm s})|\mathcal{D}u(\boldsymbol{x};\theta)-f(\boldsymbol{x})|^2\right]+\lambda\mathbb{E}_{x\in\Gamma}\left[\phi''_{\rm s}(\boldsymbol{x};\theta''_{\rm s})|\mathcal{B}u(\boldsymbol{x};\theta)-g(\boldsymbol{x})|^2\right] \tag{3.8}$$

subject to the normalization conditions,

$$\frac{1}{|Q|}\int_Q\phi'_{\rm s}(\boldsymbol{x};\theta'_{\rm s})d\boldsymbol{x}=1,\quad \frac{1}{|\Gamma|}\int_\Gamma\phi''_{\rm s}(\boldsymbol{x};\theta''_{\rm s})d\boldsymbol{x}=1. \tag{3.9}$$

Note in (3.8), to achieve the maximum of the loss function, $\phi'_{\rm s}(\boldsymbol{x};\theta'_{\rm s})$ tends to take larger values where $|\mathcal{D}u(\boldsymbol{x};\theta)-f(\boldsymbol{x})|$ is larger, and take smaller values elsewhere. Also, $\phi'_{\rm s}(\boldsymbol{x};\theta'_{\rm s})$ will not take large values everywhere since it is normalized by (3.9). The same mechanism is also true for $\phi''_{\rm s}(\boldsymbol{x};\theta''_{\rm s})$. In the latter stage of training, $\phi'_{\rm s}(\boldsymbol{x};\theta'_{\rm s})$ and $\phi''_{\rm s}(\boldsymbol{x};\theta''_{\rm s})$ have been optimized by the maximization problem above to choose "difficult" samples and, hence, the third requirement above is satisfied.

For simplicity, we can combine (3.8) and (3.9) as the following penalized optimization

$$\begin{aligned}\max_{\theta'_{\rm s},\theta''_{\rm s}}\ &\mathbb{E}_{x\in Q}\left[\phi'_{\rm s}(\boldsymbol{x};\theta'_{\rm s})|\mathcal{D}u(\boldsymbol{x};\theta)-f(\boldsymbol{x})|^2\right]+\lambda\mathbb{E}_{x\in\Gamma}\left[\phi''_{\rm s}(\boldsymbol{x};\theta''_{\rm s})|\mathcal{B}u(\boldsymbol{x};\theta)-g(\boldsymbol{x})|^2\right]\\ &-\varepsilon^{-1}\left[\left(\frac{1}{|Q|}\int_Q\phi'_{\rm s}(\boldsymbol{x};\theta'_{\rm s})d\boldsymbol{x}-1\right)^2+\left(\frac{1}{|\Gamma|}\int_\Gamma\phi''_{\rm s}(\boldsymbol{x};\theta''_{\rm s})d\boldsymbol{x}-1\right)^2\right],\end{aligned} \tag{3.10}$$

where $\varepsilon>0$ is a small penalty constant. When $\phi'_{\rm s}(\boldsymbol{x};\theta'_{\rm s})$ and $\phi''_{\rm s}(\boldsymbol{x};\theta''_{\rm s})$ are fixed, we can train the solution network $u(\boldsymbol{x};\theta)$ by minimizing (3.10), i.e.,

$$\begin{aligned}\min_{\theta}\max_{\theta'_{\rm s},\theta''_{\rm s}}\ &\mathbb{E}_{x\in Q}\left[\phi'_{\rm s}(\boldsymbol{x};\theta'_{\rm s})|\mathcal{D}u(\boldsymbol{x};\theta)-f(\boldsymbol{x})|^2\right]\\ &+\lambda\mathbb{E}_{x\in\Gamma}\left[\phi''_{\rm s}(\boldsymbol{x};\theta''_{\rm s})|\mathcal{B}u(\boldsymbol{x};\theta)-g(\boldsymbol{x})|^2\right]\\ &-\varepsilon^{-1}\left[\left(\frac{1}{|Q|}\int_Q\phi'_{\rm s}(\boldsymbol{x};\theta'_{\rm s})d\boldsymbol{x}-1\right)^2+\left(\frac{1}{|\Gamma|}\int_\Gamma\phi''_{\rm s}(\boldsymbol{x};\theta''_{\rm s})d\boldsymbol{x}-1\right)^2\right],\end{aligned} \tag{3.11}$$

which is the final model in the SelectNet method.

REMARK 3.1. *An alternate way to penalize the selection networks is to divide the residual terms in* (3.8) *by the norms of the selection networks. Namely, we solve*

$$\begin{aligned}\min_{\theta}\max_{\theta'_s,\theta''_s}\ &\|\phi'_s\|_\Omega^{-1}\mathbb{E}_{x\in Q}\left[\phi'_s(\boldsymbol{x};\theta'_s)|\mathcal{D}u(\boldsymbol{x};\theta)-f(\boldsymbol{x})|^2\right]\\ &+\lambda\|\phi''_s\|_\Gamma^{-1}\mathbb{E}_{x\in\Gamma}\left[\phi''_s(\boldsymbol{x};\theta''_s)|\mathcal{B}u(\boldsymbol{x};\theta)-g(\boldsymbol{x})|^2\right].\end{aligned} \tag{3.12}$$

However, in practice, the results of (3.12) *are sensitive to the types of norms and hyperparameters; hence* (3.12) *is more challenging to obtain good numerical results than the formulation* (3.11).

Although the introduction of SelectNet is motivated by self-paced learning in image classification, surprisingly, SelectNet can also be understood via conventional mathematical analysis. The square root of the non-negative selection networks can also be understood as the test function in the weak form of conventional PDE solvers. In the SelectNet, we apply the idea of test functions to both the PDE and the boundary condition, e.g., hoping to identify $u(\boldsymbol{x};\theta)$ ensuring the following two equalities for all non-negative test functions:

$$\left(\sqrt{\phi_{\rm s}'(\boldsymbol{x};\theta_{\rm s}')}, \mathcal{D}u(\boldsymbol{x};\theta)\right)_Q = \left(\sqrt{\phi_{\rm s}'(\boldsymbol{x};\theta_{\rm s}')}, f(\boldsymbol{x})\right)_Q$$

with $(\cdot,\cdot)_Q$ as the inner product of $L^2(Q)$ and

$$\left(\sqrt{\phi_{\rm s}''(\boldsymbol{x};\theta_{\rm s}'')}, \mathcal{B}u(\boldsymbol{x};\theta)\right)_\Gamma = \left(\sqrt{\phi_{\rm s}''(\boldsymbol{x};\theta_{\rm s}'')}, g(\boldsymbol{x})\right)_\Gamma$$

with $(\cdot,\cdot)_\Gamma$ as the inner product of $L^2(\Gamma)$. Conventional methods apply test functions for the PDE only and the test functions are not necessarily non-negative. In the SelectNet, the integration by part is not applied so as to let the test function play a role of weighting, while conventional methods use the integration by part to weaken the regularity requirement of the PDE solution. Only a single test function is used in SelectNet with a maximum requirement to guarantee that the solution of the min-max problem is the solution of the original problem (see Theorem 4.1 later), while conventional methods use sufficiently many test functions that can form a set of basis functions in the discrete test function space. The idea of using test functions in deep learning was also used in [67], where the test function was used in a weak form with integration by part. The idea of using a min-max optimization problem instead of the minimization problem to solve PDEs has been studied for many decades, e.g. [21]. Maximizing over all possible test functions can obtain the best test function that amplifies the residual error the most, which can better help the minimization problem to identify the PDE solution. When an optimization algorithm is applied to solve the min-max problem, the optimization dynamic consists of a solution dynamic that converges to the PDE solution and a test dynamic that provide a sequence of test functions to characterize the error of the numerical solution at each iteration. The training dynamic of the selection network in SelectNet approximates the test function dynamic, and the training dynamic of the solution network in SelectNet approximate the solution dynamic.

4. Error estimates. In this section, theoretical analysis are presented to show the solution errors of the basic and SelectNet models are bounded by the loss function (mean square of the residual). Specifically, we will take the elliptic PDE with Neumann boundary condition as an example. The conclusion can be generalized for other well-posed PDEs by similar argument. Consider

$$\begin{cases} -\Delta u + cu = f, \text{ in } \Omega, \\ \frac{\partial u}{\partial \rm n} = g, \text{ on } \partial\Omega, \end{cases} \tag{4.1}$$

where Ω is an open subset of $\mathbb{R}^d$ whose boundary $\partial\Omega$ is C^1 smooth; $f \in L^2(\Omega)$, $g \in L^2(\partial\Omega)$, $c(x) \geq \sigma > 0$ is a given function in $L^2(\Omega)$.

THEOREM 4.1. *Suppose the problem* (4.1) *admits a unique solution* u_* *in* $C^1(\overline{\Omega})$. *Also, suppose the variational optimization problem*

$$\min_{u\in\mathcal{N}} J(u) := \min_{u\in\mathcal{N}} \int_\Omega |-\Delta u + cu - f|^2 dx + \lambda \int_{\partial\Omega} |\frac{\partial u}{\partial n} - g|^2 dx, \tag{4.2}$$

has an admissible set $\mathcal{N} \subset C^2(\overline{\Omega})$ containing a feasible solution $u_b \in \mathcal{N}$ satisfying

$$J(u_b) < \delta, \tag{4.3}$$

then

$$\|u_b - u_*\|_{H^1(\Omega)} \leq c \max(1, \sigma^{-1}) \max(1, \lambda^{-\frac{1}{2}}) \delta^{\frac{1}{2}}, \tag{4.4}$$

where $c > 0$ is a constant only depending on d and Ω. Furthermore, let $\mathcal{S}'$ be a subset of $\{\phi \in C(\Omega) : \phi > 0\}$ which contains $\phi(x) \equiv 1$ for all $x \in \Omega$; let $\mathcal{S}''$ be a subset of $\{\phi \in C(\partial\Omega) : \phi > 0\}$ which contains $\phi(x) \equiv 1$ for all $x \in \partial\Omega$. Suppose the variational optimization problem

$$\begin{aligned} \min_{u \in \mathcal{N}} J_{\mathcal{S}',\mathcal{S}''}(u) := \min_{u \in \mathcal{N}} \max_{\phi' \in \mathcal{S}', \phi'' \in \mathcal{S}''} & \int_\Omega \phi' | -\Delta u + cu - f|^2 dx + \lambda \int_{\partial\Omega} \phi'' | \frac{\partial u}{\partial n} - g|^2 dx \\ & - \varepsilon^{-1} \left[\left(\frac{1}{|\Omega|} \int_\Omega \phi' dx - 1 \right)^2 + \left(\frac{1}{|\partial\Omega|} \int_{\partial\Omega} \phi'' dx - 1 \right)^2 \right], \end{aligned} \tag{4.5}$$

has a feasible solution $u_s \in \mathcal{N}$ satisfying

$$J_{\mathcal{S}',\mathcal{S}''}(u_s) < \delta, \tag{4.6}$$

then

$$\|u_s - u_*\|_{H^1(\Omega)} \leq c \max(1, \sigma^{-1}) \max(1, \lambda^{-\frac{1}{2}}) \delta^{\frac{1}{2}}. \tag{4.7}$$

Proof. Let $v_\mathrm{b} := u_\mathrm{b} - u_*$. Starting from the identity

$$-\Delta v_\mathrm{b} + c v_\mathrm{b} = -\Delta u_\mathrm{b} + c u_\mathrm{b} - f, \tag{4.8}$$

we multiply v_b to both sides of (4.8) and integrate over Ω. Since $v_\mathrm{b} \in C^1(\overline{\Omega})$, by integration by parts it follows

$$\|\nabla v_\mathrm{b}\|^2_{L^2(\Omega)} + \sigma \|v_\mathrm{b}\|^2_{L^2(\Omega)} \leq \int_\Omega (-\Delta u_\mathrm{b} + c u_\mathrm{b} - f) v_\mathrm{b} dx + \int_{\partial\Omega} v_\mathrm{b} \frac{\partial v_\mathrm{b}}{\partial \mathrm{n}} dx. \tag{4.9}$$

Hence, by the Cauchy-Schwarz inequality,

$$\begin{aligned} \min(1, \sigma) \|v_\mathrm{b}\|^2_{H^1(\Omega)} \leq & \| -\Delta u_\mathrm{b} + c u_\mathrm{b} - f\|_{L^2(\Omega)} \cdot \|v_\mathrm{b}\|_{L^2(\Omega)} \\ & + \|v_\mathrm{b}\|_{L^2(\partial\Omega)} \cdot \| \frac{\partial u_\mathrm{b}}{\partial \mathrm{n}} - g\|_{L^2(\partial\Omega)}. \end{aligned} \tag{4.10}$$

By the trace theorem, $\|v_\mathrm{b}\|_{L^2(\partial\Omega)} \leq c' \|v_\mathrm{b}\|_{H^1(\Omega)}$ for some $c' > 0$ only depending on d and Ω. Then we have

$$\begin{aligned} & \min(1, \sigma) \|v_\mathrm{b}\|^2_{H^1(\Omega)} \\ & \quad \leq \|v_\mathrm{b}\|_{H^1(\Omega)} \left(\| -\Delta u_\mathrm{b} + c u_\mathrm{b} - f\|_{L^2(\Omega)} + c' \| \frac{\partial u_\mathrm{b}}{\partial \mathrm{n}} - g\|_{L^2(\partial\Omega)} \right) \\ & \quad \leq c'' \|v_\mathrm{b}\|_{H^1(\Omega)} \left(\| -\Delta u_\mathrm{b} + c u_\mathrm{b} - f\|^2_{L^2(\Omega)} + \| \frac{\partial u_\mathrm{b}}{\partial \mathrm{n}} - g\|^2_{L^2(\partial\Omega)} \right)^{\frac{1}{2}}, \end{aligned} \tag{4.11}$$

with $c'' = \sqrt{2}\max(1, c')$. Finally, by the hypothesis (4.3), (4.4) directly follows from (4.11).

Moreover, by taking $\phi' \equiv 1$, $\phi'' \equiv 1$ we directly have

$$\int_\Omega |-\Delta u + cu - f|^2 dx + \lambda \int_{\partial\Omega} |\frac{\partial u}{\partial \mathrm{n}} - g|^2 dx \le J_{\mathcal{S}',\mathcal{S}''}(u_\mathrm{s}) < \delta. \tag{4.12}$$

The same estimate for $\|u_\mathrm{s} - u_*\|_{H^1(\Omega)}$ can be obtained by similar argument. ∎

By using the triangle inequality, we can conclude the solutions of the basic and SelectNet models are equivalent as long as the loss functions are minimized sufficiently. As an immediate result, we have the following corollary.

COROLLARY 4.2. *Under the hypothesis of Theorem 4.1, we have*

$$\|u_b - u_s\|_{H^1(\Omega)} \le c\max(1, \sigma^{-1})\max(1, \lambda^{-\frac{1}{2}})\delta^{\frac{1}{2}}. \tag{4.13}$$

5. Network Implementation.

5.1. Network Architecture. The proposed framework is independent of the choice of DNNs. Advanced network design may improve the accuracy and convergence of the proposed framework, which would be interesting for future work.

In this paper, feedforward neural networks will be repeatedly applied. Let $\phi(\boldsymbol{x};\theta)$ denote such a network with an input $\boldsymbol{x}$ and parameters θ, then it is defined recursively as follows:

$$\begin{aligned} &\boldsymbol{x}^0 = \boldsymbol{x}, \\ &\boldsymbol{x}^{l+1} = \sigma(\boldsymbol{W}^l\boldsymbol{x}^l + \boldsymbol{b}^l), \quad l = 0, 1, \cdots, L-1, \\ &\phi(\boldsymbol{x};\theta) = \boldsymbol{W}^L\boldsymbol{x}^L + \boldsymbol{b}^L, \end{aligned} \tag{5.1}$$

where σ is an application-dependent nonlinear activation function, and θ consists of all the weights and biases $\{\boldsymbol{W}^l, \boldsymbol{b}^l\}_{l=0}^L$ satisfying

$$\begin{aligned} &\boldsymbol{W}^0 \in \mathbb{R}^{m\times d}, \quad \boldsymbol{W}^L \in \mathbb{R}^{1\times m}, \quad \boldsymbol{b}^L \in \mathbb{R}, \\ &\boldsymbol{W}^l \in \mathbb{R}^{m\times m}, \quad \text{for } l = 1, \cdots, L-1, \\ &\boldsymbol{b}^l \in \mathbb{R}^{m\times 1}, \quad \text{for } l = 0, \cdots, L-1. \end{aligned} \tag{5.2}$$

The number m is called the width of the network and L is called the depth.

For simplicity, we deploy the feedforward neural network with the activation function $\sigma(\boldsymbol{x}) = \sin(\boldsymbol{x})$ as the solution network that approximates the solution of the PDE. As for the selection network introduced in Section 3, since it is required to be bounded in $[m_0, M_0]$, it can be defined via

$$\phi_\mathrm{s}(\boldsymbol{x};\theta) = (M_0 - m_0)\sigma_\mathrm{s}(\hat{\phi}(\boldsymbol{x};\theta)) + m_0, \tag{5.3}$$

where $\sigma_\mathrm{s}(x) = 1/(1+\exp(-x))$ is the sigmoidal function, and $\hat{\phi}$ is a generic network, e.g. a feedforward neural network with the ReLU activation $\sigma(\boldsymbol{x}) = \max\{0, \boldsymbol{x}\}$.

5.2. Special Network for Dirichlet Boundary Conditions. In the case of homogeneous Dirichlet boundary conditions, it is worth mentioning a special network design that satisfies the boundary condition automatically as discussed in [38, 3].

Let us focus on the boundary value problem to introduce this special network structure. It is straightforward to generalize this idea to the case of an initial boundary value problem and we omit this discussion. Assume a homogeneous Dirichlet boundary condition

$$u(\boldsymbol{x}) = 0, \quad \text{on } \partial\Omega, \tag{5.4}$$

then a solution network automatically satisfying the condition above can be constructed by

$$u(\boldsymbol{x};\theta) = h(\boldsymbol{x})\hat{u}(\boldsymbol{x};\theta), \tag{5.5}$$

where $\hat{u}$ is a generic network as in (5.1), and h is a specifically chosen function such as $h = 0$ on Γ.

For example, if Ω is a d-dimensional unit ball, then $u(\boldsymbol{x};\theta)$ can take the form

$$u(\boldsymbol{x};\theta) = (|\boldsymbol{x}|^2 - 1)\hat{u}(\boldsymbol{x};\theta). \tag{5.6}$$

For another example, if Ω is the d-dimensional cube $[-1,1]^d$, then $u(x;\theta)$ can take the form

$$u(\boldsymbol{x};\theta) = \prod_{i=1}^{d}(x_i^2 - 1)\hat{u}(\boldsymbol{x};\theta). \tag{5.7}$$

Since the boundary condition $\mathcal{B}u = 0$ is always fulfilled, it suffices to solve the min-max problem

$$\min_{\theta}\max_{\theta'_{\mathrm{s}}} \mathbb{E}_{x\in Q}\left[\phi'_{\mathrm{s}}(\boldsymbol{x};\theta'_{\mathrm{s}})|\mathcal{D}u(\boldsymbol{x};\theta) - f(\boldsymbol{x})|^2\right] - \varepsilon^{-1}\left(\frac{1}{|Q|}\int_Q \phi'_{\mathrm{s}}(\boldsymbol{x};\theta'_{\mathrm{s}})d\boldsymbol{x} - 1\right)^2 \tag{5.8}$$

to identify the best solution network $u(\boldsymbol{x};\theta)$.

5.3. Derivatives of Networks. Note that the evaluation of the optimization problem in (3.11) involves the derivative of the network $u(\boldsymbol{x};\theta)$ in terms of $\boldsymbol{x}$. When the activation function of the network is differentiable, the network is differentiable and the derivative in terms of $\boldsymbol{x}$ can be evaluated efficiently via the back-propagation algorithm. Note that the network we adopt in this paper is not differentiable. Hence, finite difference method will be utilized to estimate the derivative of networks. For example, for the elliptic operator $\mathcal{D}u := \nabla\cdot(a(x)\nabla u)$, $\mathcal{D}u(\boldsymbol{x};\theta)$ can be estimated by the second-order central difference formula

$$\begin{aligned}\mathcal{D}u(\boldsymbol{x};\theta) \approx \frac{1}{h^2}\sum_{i=1}^{d} & a(\boldsymbol{x}+\frac{1}{2}h\boldsymbol{e}_i)(u(\boldsymbol{x}+h\boldsymbol{e}_i,\theta) - u(\boldsymbol{x};\theta)) \\ & - a(\boldsymbol{x}-\frac{1}{2}h\boldsymbol{e}_i)(u(\boldsymbol{x};\theta) - u(\boldsymbol{x}-h\boldsymbol{e}_i,\theta)),\end{aligned} \tag{5.9}$$

up to an error of $O(dh^2)$. In the experiments (Section 6), we take $h = 10^{-4}$ for all examples with d up to 100. Hence the truncation errors are up to $O(10^{-6})$, which are overwhelmed by the final errors (at least $O(10^{-4})$). This implies the truncation errors from finite difference can be ignored in practice.

Indeed, one can also use the automatic differentiation in TensorFlow or Pytorch based on the explicit formula of networks to evaluate the derivatives in the practical implementation, which brings no truncation errors. However, the computational cost of this approach is high when a second order (or higher) derivative is computed. Hence we choose finite difference method for derivative computation in this paper.

5.4. Network Training. Once networks have been set up, the rest is to train the networks to solve the min-max problem in (3.11). The stochastic gradient descent (SGD) method or its variants (e.g., Adam [35]) is an efficient tool to solve this problem numerically. Although the convergence of SGD for the min-max problem is still an active research topic [54, 10, 61], empirical success shows that SGD can provide a good approximate solution.

Before completing the algorithm description of SelectNet, let us introduce the key setup of SGD and summarize it in Algorithm 1 below. In each training iteration, we first set uniformly distributed training points $\{\boldsymbol{x}_i^1\}_{i=1}^{N_1} \subset Q$ and $\{\boldsymbol{x}_i^2\}_{i=1}^{N_2} \subset \Gamma$, and define the empirical loss of these training points as

$$
\begin{aligned}
J(\theta, \theta_{\mathrm{s}}) = \frac{1}{N_1}\sum_{i=1}^{N_1} \phi_{\mathrm{s}}'(\boldsymbol{x}_i^1; \theta_{\mathrm{s}}')|\mathcal{D}u(\boldsymbol{x}_i^1, \theta) - f(\boldsymbol{x}_i^1)|^2 \\
+ \frac{\lambda}{N_2}\sum_{i=1}^{N_2} \phi_{\mathrm{s}}''(\boldsymbol{x}_i^2; \theta_{\mathrm{s}}'')|\mathcal{B}u(\boldsymbol{x}_i^2, \theta) - g(\boldsymbol{x}_i^2)|^2 \\
- \varepsilon^{-1}\left[\left(\frac{1}{N_1}\sum_{i=1}^{N_1} \phi_{\mathrm{s}}'(\boldsymbol{x}_i^1; \theta_{\mathrm{s}}') - 1\right)^2 + \left(\frac{1}{N_2}\sum_{i=1}^{N_2} \phi_{\mathrm{s}}''(\boldsymbol{x}_i^2; \theta_{\mathrm{s}}'') - 1\right)^2\right],
\end{aligned} \tag{5.10}
$$

where $\theta_{\mathrm{s}} := [\theta_{\mathrm{s}}', \theta_{\mathrm{s}}'']$. Next, θ_{s} can be updated by the gradient ascent via

$$
\theta_{\mathrm{s}} \leftarrow \theta_{\mathrm{s}} + \tau_{\mathrm{s}} \nabla_{\theta_{\mathrm{s}}} J, \tag{5.11}
$$

and θ can be updated by the gradient descent via

$$
\theta \leftarrow \theta - \tau \nabla_{\theta} J, \tag{5.12}
$$

with step sizes τ_{s} and τ. Note that training points are randomly renewed in each iteration. In fact, for the same set of training points in each iteration, the updates (5.11) and (5.12) can be performed n_1 and n_2 times, respectively.

6. Numerical Experiments. In this section, the proposed SelectNet model is tested on several PDE examples, including elliptic/parabolic and linear/nonlinear high-dimensional problems. Other network-based methods are also implemented for comparison. For all methods, we choose the feedforward architecture with activation $\sigma(x) = \max(x^3, 0)$ for the solution network. Additionally, for SelectNet, we choose feedforward architecture with ReLU activation for the selection network. AdamGrad [12] is employed to solve the optimization problems, with learning rates

$$
\tau_{\mathrm{s}}^{(k)} = 10^{-4}, \tag{6.1}
$$

for the selection network, and

$$
\tau^{(k)} = 10^{-3-3j/1000}, \text{ if } n^{(j)} < k \le n^{(j+1)}, \quad \forall j = 0, \cdots, 1000, \tag{6.2}
$$

for the solution network, where $0 = n^{(0)} < \cdots < n^{(1000)} = n$ are equidistant segments of total iterations. Other parameters used in the model and algorithm are listed in Table 6.1. Unless otherwise specified, in all examples, we set $N_1 = 10000$, $N_2 = 10000$, $n = 20000$, $n_1 = 1$, $\lambda = 1$, $m = 100$, $L = 3$ for all methods and set $n_2 = 1$, $\varepsilon = 0.001$, $m_{\mathrm{s}} = 20$, $L_{\mathrm{s}} = 3$, $m_0 = 0.8$, $M_0 = 5$ especially for SelectNet.

Algorithm 1 The Least Squares Model with SelectNet

Require: the PDE (2.4)
Ensure: the parameters θ in the solution network $u(\boldsymbol{x};\theta)$
  Set parameters n, n_1, n_2 for iterations and parameters N_1, N_2 for sample sizes
  Initialize $u(\boldsymbol{x};\theta^{0,0})$ and $\phi_{\rm s}(\boldsymbol{x};\theta_{\rm s}^{0,0})$
  for $k = 1, \cdots, n$ **do**
    Generate uniformly distributed sampling points
    $\{\boldsymbol{x}_i^1\}_{i=1}^{N_1} \subset Q$ and $\{\boldsymbol{x}_i^2\}_{i=1}^{N_2} \subset \Gamma$
    for $j = 1, \cdots, n_1$ **do**
      Update $\theta_{\rm s}^{k-1,j} \leftarrow \theta_{\rm s}^{k-1,j-1} + \tau_{\rm s}^{(k)} \nabla_{\theta_{\rm s}} J(\theta_{\rm s}^{k-1,j-1}, \theta^{k-1,0})$
    end for
    $\theta_{\rm s}^{k,0} \leftarrow \theta_{\rm s}^{k-1,n_1}$
    for $j = 1, \cdots, n_2$ **do**
      Update $\theta^{k-1,j} \leftarrow \theta^{k-1,j-1} - \tau^{(k)} \nabla_{\theta} J(\theta_{\rm s}^{k,0}, \theta^{k-1,j-1})$
    end for
    $\theta^{k,0} \leftarrow \theta^{k-1,n_2}$
    if Stopping criteria is satisfied **then**
      Return $\theta = \theta^{k,0}$
    end if
  end for

d	the dimension of the problem
m	the width of each layer in the solution network
$m_{\rm s}$	the width of each layer in the selection network
L	the depth of the solution network
$L_{\rm s}$	the depth of the selection network
M_0	the upper bound of the selection network
m_0	the lower bound of the selection network
n	number of iterations in the optimization
n_1	number of updates of the selection network in each iteration
n_2	number of updates of the solution network in each iteration
N_1	number of training points inside the domain in each iteration
N_2	number of training points on the domain boundary in each iteration
ε	penalty parameter to uniform the selection network
λ	summation weight of the boundary least squares

Table 6.1: *Parameters in the model and algorithm.*

We take the (relative) ℓ^2 error at uniformly distributed testing points $\{\boldsymbol{x}_i\} \subset \tilde{Q}$ as the metric to evaluate the accuracy, which is formulated by

$$e_{\ell^2}(\theta) := \left(\frac{\sum_i |u(\boldsymbol{x}_i;\theta) - u(\boldsymbol{x}_i)|^2}{\sum_i |u(\boldsymbol{x}_i)|^2} \right)^{\frac{1}{2}}. \tag{6.3}$$

Here $\tilde{Q} \subset Q$ is the domain for error evaluation. In all examples, we choose 10000 testing points for error evaluation.

	Basic	SelectNet	DRM	WAN
Mean of Errors μ	7.588×10^{-3}	3.288×10^{-4}	8.681×10^{-4}	2.177×10^{-3}
Standard Deviation σ	1.080×10^{-3}	7.821×10^{-5}	1.072×10^{-4}	8.002×10^{-4}
Coefficient of Variation σ/μ	14.2%	23.8%	12.4%	36.8%

Table 6.2: *Means and standard deviations of the ℓ^2 errors obtained within 600 seconds by various methods in the comparative example. (totally 50 trials for each method)*

6.1. Comparative Experiment. In the first experiment, we compare the proposed SelectNet model with other network-based methods on the following 2-D Poisson equation,

$$\begin{aligned} -\Delta u &= 1, \quad \text{in } \Omega := (-1,1)\times(-1,1), \\ u &= 0, \quad \text{on } \partial\Omega, \end{aligned} \tag{6.4}$$

with a solution expressed by the series

$$u(x_1,x_2) = -\frac{64}{\pi^4} \sum_{\substack{n,m=1 \\ n,m \text{ odd}}}^{\infty} (-1)^{\frac{n+m}{2}} \frac{cos(\frac{n\pi x_1}{2})cos(\frac{m\pi x_2}{2})}{nm(n^2+m^2)}. \tag{6.5}$$

As a classic testing example for PDE methods, the problem (6.4) is well-known for the low-regularity of its solution at the four corners of Ω. In this experiment, both the interior training points and testing points are chosen uniformly in the domain, and the boundary training points are chosen uniformly on the boundary. Since the numerical results are influenced by the randomness of the network initialization and the stochastic training process, we implement each method for 50 times with different seeds and compute the mean and standard deviation of the final errors.

6.1.1. Comparison with Recent Methods. We implement the basic least squares model, SelectNet model, and recently raised methods: deep Ritz method (DRM) [19] and weak adversarial networks (WAN) [67] under the same setting, and compare their convergence speed. All methods are implemented for 600 seconds, with learning rates given in (6.2) for the first 10000 iterations and 10^{-6} for the subsequent iterations. The means and standard deviations of the final ℓ^2 errors of 50 trials are listed in Table 6.2. For each method. We select 10 of the 50 trials to present their error curves with respect to the computing time in Figure 6.1. It is observed in the first 50 seconds SelectNet has the fastest error decay, and in the end, SelectNet obtains the smallest errors. We also note that for each method, the error deviations are much smaller than the error means, showing the numerical stability with respect to the stochasticity of algorithms.

Across different trials, the selection networks of the SelectNet model evolve in a nearly identical manner. From all trials, we take one to show the surfaces of the selection network at the initial stage and the 2000th, 5000th, 10000th iterations (see Figure 6.2). We can clearly find that high peaks appear at the four corners over time where the solution is less regular, while other region preserves to be low and constant. This distribution will improve the convergence at the corners that are "difficult" to deal with.

6.1.2. Comparison with Binary Weighting. To verify that SelectNet is advantageous over other weighting strategies, we also implement the binary weighting method. Namely, in the basic least squares method, we select p $(0 < p < 1)$ training points having larger residuals to be weighted with $w_{\mathrm{L}} > 1$, and let the other points be weighted with $w_{\mathrm{S}} < 1$. Specifically, we solve the

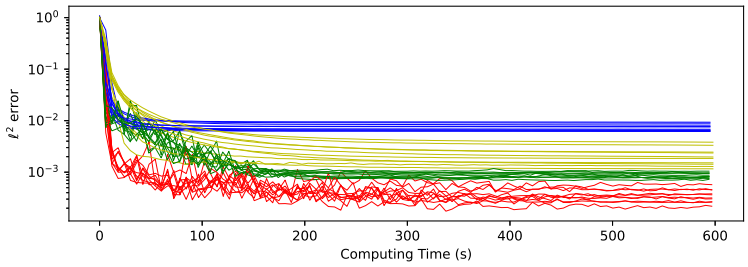


Fig. 6.1: *ℓ^2 errors v.s. computing time in the comparative example (Red: SelectNet model; Blue: the basic model; green: DRM; yellow: WAN. 10 selected curves for each method).*

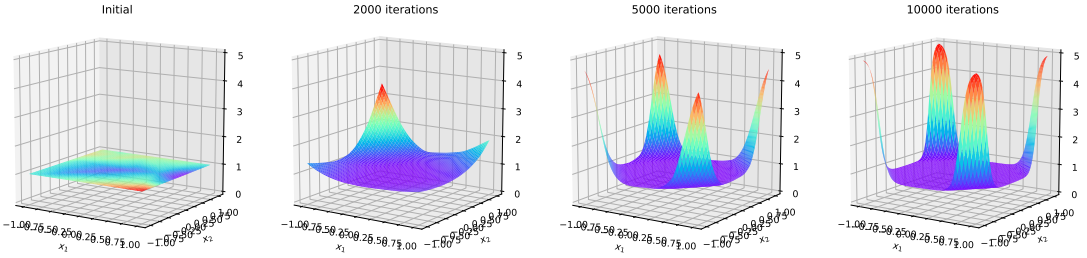


Fig. 6.2: *The evolution of the selection network over time in the comparative example.*

problem (6.4) by

$$\min_{\theta} \frac{1}{N_1}\left(\sum_{\boldsymbol{x}\in\mathcal{L}_1} w_{\mathrm{L}}|-\Delta u(\boldsymbol{x};\theta)-1|^2+\sum_{\boldsymbol{x}\in\mathcal{S}_1} w_{\mathrm{S}}|-\Delta u(\boldsymbol{x};\theta)-1|^2\right) + \frac{\lambda}{N_2}\left(\sum_{\boldsymbol{x}\in\mathcal{L}_2} w_{\mathrm{L}}|u(\boldsymbol{x};\theta)|^2+\sum_{\boldsymbol{x}\in\mathcal{S}_2} w_{\mathrm{S}}|u(\boldsymbol{x};\theta)|^2\right), \tag{6.6}$$

where $\{\mathcal{L}_1,\mathcal{S}_1\}$ is a partition of $\{\boldsymbol{x}_i^1\}_{i=1}^{N_1}$ satisfying $|\mathcal{L}_1|=pN_1$, $|\mathcal{S}_1|=(1-p)N_1$, $|-\Delta u(\boldsymbol{x}';\theta)-1|\geq|-\Delta u(\boldsymbol{x}'';\theta)-1|$ for any $\boldsymbol{x}'\in\mathcal{L}_1$ and $\boldsymbol{x}''\in\mathcal{S}_1$; $\{\mathcal{L}_2,\mathcal{S}_2\}$ is a partition of $\{\boldsymbol{x}_i^2\}_{i=1}^{N_2}$ satisfying $|\mathcal{L}_2|=pN_2$, $|\mathcal{S}_2|=(1-p)N_2$, $|u(\boldsymbol{x}';\theta)|\geq|u(\boldsymbol{x}'';\theta)|$ for any $\boldsymbol{x}'\in\mathcal{L}_2$ and $\boldsymbol{x}''\in\mathcal{S}_2$. The binary weights are chosen subject to the following normalization condition

$$w_{\mathrm{L}}p+w_{\mathrm{S}}(1-p)=1,\quad w_{\mathrm{L}}p+w_{\mathrm{S}}(1-p)=1. \tag{6.7}$$

As with the preceding tests, we implement the weighting model (6.6) with various combinations of parameters for 600 seconds. The means and deviations of the final ℓ^2 errors are listed in Table 6.3. It shows the best combination obtains the mean error 7.375×10^{-3}, which is slightly better than the original basic model and much worse than the SelectNet model.

6.2. High-dimensional Examples. In the second experiment, we will implement the basic and SelectNet models in a series of high-dimensional examples ($d\geq10$) to reflect the advantage of using SelectNet. Note from the preceding comparative experiment that SelectNet can obtain much smaller error means than the basic model, which overwhelms the error deviations. Therefore,

$p = 20\%$			
$w_{\mathrm{L}}/w_{\mathrm{S}}$	2	4	8
Mean of Errors μ	8.104×10^{-3}	8.649×10^{-3}	9.260×10^{-3}
Standard Deviation σ	8.384×10^{-4}	1.131×10^{-3}	1.205×10^{-3}
Coefficient of Variation σ/μ	10.3%	13.1%	13.0%
$p = 50\%$			
$w_{\mathrm{L}}/w_{\mathrm{S}}$	2	4	8
Mean of Errors μ	7.395×10^{-3}	7.612×10^{-3}	7.506×10^{-3}
Standard Deviation σ	1.080×10^{-3}	1.168×10^{-3}	1.113×10^{-3}
Coefficient of Variation σ/μ	14.6%	15.3%	14.8%
$p = 80\%$			
$w_{\mathrm{L}}/w_{\mathrm{S}}$	2	4	8
Mean of Errors μ	7.426×10^{-3}	$\mathbf{7.375 \times 10^{-3}}$	7.512×10^{-3}
Standard Deviation σ	9.502×10^{-4}	1.023×10^{-3}	1.077×10^{-3}
Coefficient of Variation σ/μ	12.8%	13.9%	14.3%

Table 6.3: *Means and standard deviations of the ℓ^2 errors obtained within 600 seconds by binary weighting in the comparative example. (totally 50 trials for each combination)*

considering the long time spent in high-dimensional problems, we only implement both models for once in each case to present the results in the paper.

Since in high-dimensional cases, most of the random points following a uniform distribution are near the boundary, we take an annularly uniform strategy instead of uniform sampling. Specifically, for a high-dimensional unit circle, we divide it into N_a annuli $\{k/N_a < |\boldsymbol{x}| < (k+1)/N_a\}_{k=0}^{N_a-1}$ and generates N_1/N_a samples uniformly in each annulus. In the following experiments, we choose $N_a = 10$. This sampling strategy is applied in generating interior training points and testing points. For generating boundary training points, we still use uniform sampling.

6.2.1. Elliptic Equations with Low-Regularity Solutions. First, let us consider the nonlinear elliptic equation inside a bounded domain

$$\begin{aligned} -\nabla \cdot (a(x)\nabla u) + |\nabla u|^2 &= f(x), \quad \text{in } \Omega := \{x : |x| < 1\}, \\ u &= g(x), \quad \text{on } \partial\Omega, \end{aligned} \tag{6.8}$$

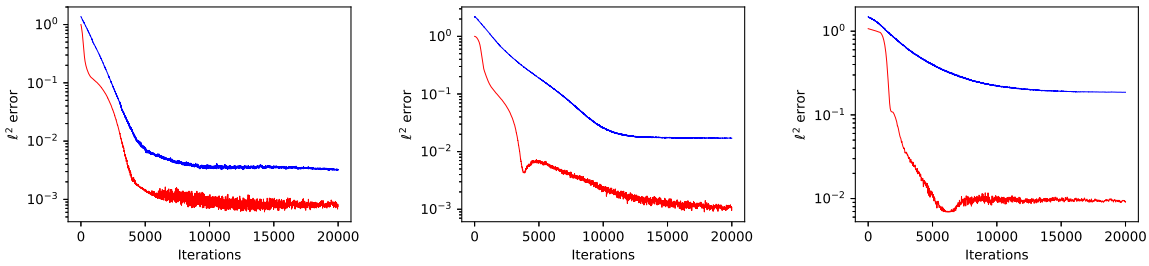
with $a(x) = 1 + \frac{1}{2}|x|^2$. In this case, we specify the exact solution by

$$u(x) = \sin(\frac{\pi}{2}(1 - |x|)^{2.5}), \tag{6.9}$$

whose first derivative is singular at the origin and the third derivative is singular on the boundary. Note the problem is nonlinear if $\mu \neq 0$. We solve the high-dimensional nonlinear problem for $d = 10$, 20 and 100. The errors obtained by the basic and SelectNet models with 20000 iterations are listed in Table 6.4. Since the basic model costs less time for one iteration, we also list the errors obtained by SelectNet with the same computing time as the basic model for comparison. The curves of error decay versus iterations are shown in Fig. 6.3. From these results, it is observed both models are effective on the nonlinear elliptic problem of all dimensions, but SelectNet has a clearly better performance than the basic model: its accuracy is one-digit better than the basic model. Besides, we present in Fig. 6.4 the following surfaces at (x_1, x_2)-slice

Dimension	SelectNet	SelectNet*	Basic
$d = 10$	7.944×10^{-4}	8.089×10^{-4}	3.193×10^{-3}
$d = 20$	9.584×10^{-4}	1.241×10^{-3}	1.707×10^{-2}
$d = 100$	9.257×10^{-3}	1.004×10^{-2}	1.862×10^{-1}

Table 6.4: *ℓ^2 errors obtained by various models in the nonlinear elliptic example. ("SelectNet" and "Basic" denote the final errors obtained by SelectNet and basic models with* 20000 *iterations; "SelectNet*" denotes the error obtained by SelectNet model with the same computing time as 20000 iterations of basic model, the same below)*



(a) $d = 10$ (b) $d = 20$ (c) $d = 100$

Fig. 6.3: *ℓ^2 errors v.s. iterations in the nonlinear elliptic example (Red: SelectNet model; Blue: the basic model).*

- the numerical solution: $u(x_1, x_2, 0, \cdots, 0; \theta)$
- the modulus of the numerical residual error: $|\mathcal{D}u(x_1, x_2, 0, \cdots, 0; \theta) - f(x_1, x_2, 0, \cdots, 0)|$
- the selection network: $\phi'_{\mathrm{s}}(x_1, x_2, 0, \cdots, 0; \theta'_{\mathrm{s}})$

for the 20-dimensional case. It shows that the residual error accumulates near the origin due to its low regularity. On the other hand, the selection network attains its peak at the origin, implying that training points are highly weighted near the origin where the residual error is mainly distributed. Note that the selection network is not supported locally near the low-regularity point, which means that the selection network will not make the training of the solution network focus on the low-regularity point only.

6.2.2. Linear Parabolic Equations. In this example, SelectNet is tested on the following initial boundary value problem of the linear parabolic equation

$$\begin{aligned} \partial_t u(x,t) - \nabla_x \cdot (a(x)\nabla_x u(x,t)) &= f(x,t), \quad \text{in } Q := \Omega \times (0,1), \\ u(x,t) &= g(x), \quad \text{on } \partial\Omega \times (0,1), \\ u(x,0) &= h(x), \quad \text{in } \Omega, \end{aligned} \tag{6.10}$$

where $a(x) = 1 + \frac{1}{2}|x|$ and $\Omega := \{x : |x| < 1\}$. The exact solution is set by

$$u(x,t) = \exp(|x|\sqrt{1-t}). \tag{6.11}$$

Note u is at most C^0 smooth at $t = 1$ and $|x| = 0$. In the SelectNet model, time-discretization schemes are not utilized. Instead, we regard t as an extra spatial variable of the problem. Hence the problem domain $\Omega \times (0, 1)$ is an analog of a hypercylinder, and the "boundary" value is specified

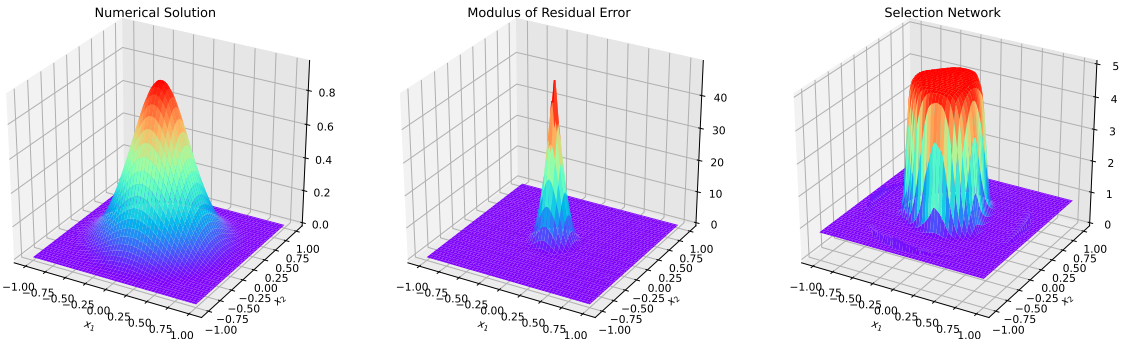


Fig. 6.4: *The (x_1, x_2)-surfaces of the numerical solution, the modulus of residual errors and selection network by SelectNet (d=20) in the nonlinear elliptic example.*

Dimension	SelectNet	SelectNet*	Basic
$d = 10$	1.490×10^{-2}	1.502×10^{-2}	3.531×10^{-2}
$d = 20$	2.990×10^{-2}	3.000×10^{-2}	8.748×10^{-2}
$d = 100$	6.302×10^{-2}	6.268×10^{-2}	1.357×10^{-1}

Table 6.5: *ℓ^2 errors obtained by various models in the linear parabolic example.*

in the bottom $\Omega \times \{t = 0\}$ and the side $\partial\Omega \times (0, 1)$. This example is tested for $d = 10$, 20 and 100, by evaluating the relative ℓ^2 error in $\Omega \times (0, 1)$. The errors of the basic and SelectNet models are listed in Table 6.5. It is clearly shown SelectNet still obtains smaller errors than the basic model with the same number of iterations or computing time. In Fig. 6.5 the curves of error decay are presented, and in Fig. 6.6 the (t, x_1)-surfaces of the numerical solution, the modulus of the residual errors and selection network for $d = 20$ are displayed, from that we can observe the residual error is mainly distributed near the singular point $\boldsymbol{x} = 0$ and the terminal slice $t = 1$. Accordingly, the selection network takes its maximum in this region.

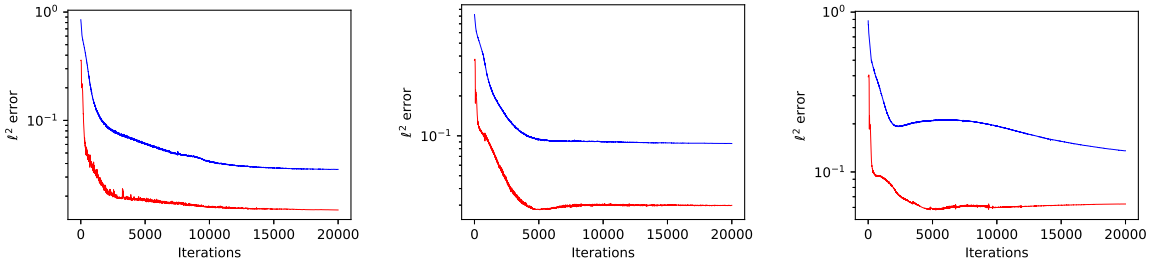
6.2.3. Allen-Cahn Equation. In this example, we test SelectNet model on the following 100-dimensional Allen-Cahn equation

$$\begin{aligned} \partial_t u(x,t) - \Delta_x u(x,t) - u(x,t) + u^3(x,t) &= f(x,t), \quad \text{in } Q := \Omega \times (0,1), \\ u(x,t) &= g(x), \quad \text{on } \partial\Omega \times (0,1), \\ u(x,0) &= h(x), \quad \text{in } \Omega, \end{aligned} \tag{6.12}$$

where $a(x) = 1 + \frac{1}{2}|x|$ and $\Omega := \{x : |x| < 1\}$. Note the Allen-Cahn equation is a nonlinear parabolic equation. The exact solution is set as

$$u(x,t) = e^{-t} \sin(\frac{\pi}{2}(1 - |x|)^{2.5}). \tag{6.13}$$

The errors obtained by SelectNet model and the basic model with 20000 iterations are 6.358×10^{-3} and 3.347×10^{-2}, respectively. And the SelectNet error obtained with the same computing time as the basic model is 6.218×10^{-3}. The error curves versus iterations are shown in Fig. 6.7. It can be seen in the figure the error curve of the SelectNet decays faster to lower levels than the basic model. Moreover, the (t, x_1)-surface of the numerical solution, the modulus of residual errors and selection network are shown in Fig. 6.8, from that we can observe the selection network takes its



(a) $d = 10$ (b) $d = 20$ (c) $d = 100$

Fig. 6.5: *ℓ^2 errors v.s. iterations in the linear parabolic example (Red: SelectNet model; Blue: the basic model).*

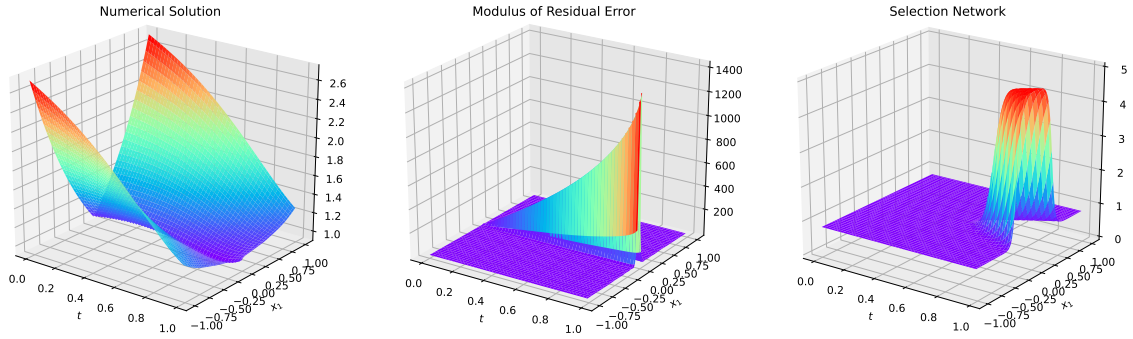


Fig. 6.6: *The (t, x_1)-surfaces of the numerical solution, the modulus of residual errors and selection network by SelectNet (d=20) in the linear parabolic example.*

maximum near the singular point $\boldsymbol{x} = 0$ and the initial slice $t = 0$, where the highest residual error is located.

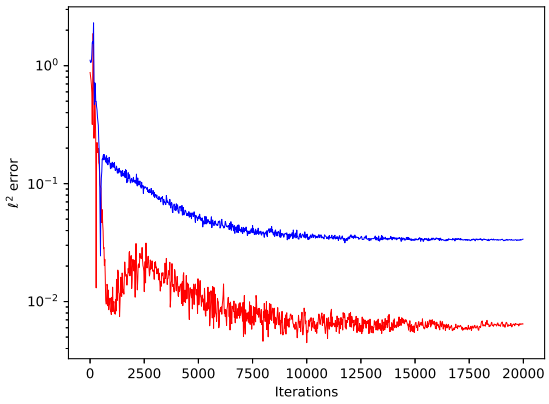
6.2.4. Hyperbolic Equations. In the last example, we test SelectNet by solving the initial boundary value problem of the hyperbolic (wave) equation, which is given by

$$\begin{aligned} &\frac{\partial^2 u(x,t)}{\partial t^2} - \Delta_x u(x,t) = f(x,t), \text{ in } \Omega \times (0,1), \\ &u(x,t) = g_0(x,t), \text{ on } \partial\Omega \times (0,1), \\ &u(x,0) = h_0(x), \quad \frac{\partial u(x,0)}{\partial t} = h_1(x) \text{ in } \Omega, \end{aligned} \tag{6.14}$$

with $\Omega := \{x : |x| < 1\}$ and exact solution is set by

$$u(x,t) = \big(\exp(t^2) - 1\big)\sin(\frac{\pi}{2}(1 - |x|)^{2.5}). \tag{6.15}$$

Same as in preceding examples, we solve the problem of $d = 10$, 20 and 100 and compute the relative ℓ^2 errors of the basic and SelectNet models. The obtained errors are listed in Table 6.6, which demonstrates the SelectNet still converges faster than the basic model (especially when d is higher), obtaining smaller errors. Also, we display the curves of error decay in Fig. 6.9, and the



(a) $d = 100$

Fig. 6.7: *ℓ^2 errors v.s. iterations in the Allen-Cahn example (Red: SelectNet model; Blue: the basic model).*

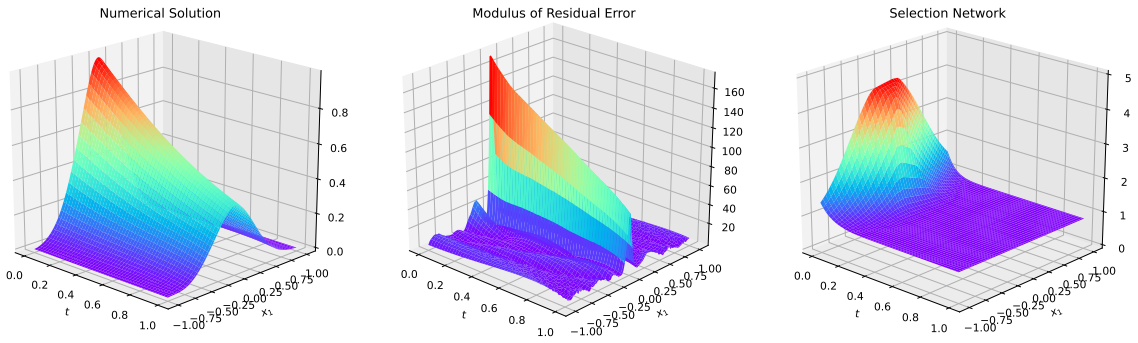


Fig. 6.8: *The (t, x_1)-surfaces of the numerical solution, the modulus of residual errors and selection network by SelectNet in the Allen-Cahn example.*

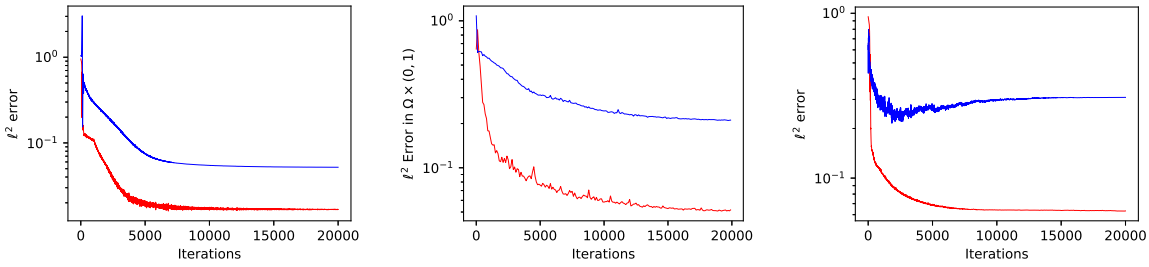
(t, x_1)-surfaces of the numerical results when $d = 20$ in Fig. 6.10. The results in the examples of parabolic and hyperbolic equations imply our proposed model works successfully for time-dependent problems.

7. Conclusion. In this work, we improve the network-based least squares models on generic PDEs by introducing a selection network for selected sampling in the optimization process. The objective is to place higher weights on the sampling points having larger point-wise residual errors, and correspondingly we propose the SelectNet model that is a min-max optimization. In the implementation, both the solution and selection functions are approximated by feedforward neural networks, which are trained alternatively in the algorithm. The proposed SelectNet framework can solve high-dimensional PDEs that are intractable by traditional PDE solvers.

In the numerical examples, it is demonstrated the proposed SelectNet model works effectively for elliptic, parabolic, and hyperbolic equations, even if in the case of nonlinear equations. Furthermore, numerical results show that the proposed model outperforms the basic least squares model. In the problems with low-regularity solutions, SelectNet will focus on the region with larger errors

Dimension	SelectNet	SelectNet*	Basic
$d = 10$	1.671×10^{-2}	1.701×10^{-2}	5.200×10^{-2}
$d = 20$	3.281×10^{-2}	3.292×10^{-2}	9.665×10^{-2}
$d = 100$	6.319×10^{-2}	6.351×10^{-2}	3.089×10^{-1}

Table 6.6: *Final ℓ^2 errors obtained by various models in the hyperbolic example.*



(a) $d = 10$ (b) $d = 20$ (c) $d = 100$

Fig. 6.9: *ℓ^2 errors v.s. iterations in the hyperbolic example (Red: SelectNet model; Blue: the basic model).*

automatically, finally improving the speed of convergence.

In this paper, we apply neural networks with piecewise polynomial functions as activation functions. If the floor, ReLU, Sign, and exponential functions are used as activation functions, [58, 59] showed that deep network approximation has no curse of dimensionality in the approximation error for Hölder continuous functions. But unfortunately, efficient numerical algorithms for these networks are still not available yet. It is interesting to explore the application of these networks to approximate the solutions of high-dimensional PDEs in the weak sense as future work.

Acknowledgments. Y. G. was partially supported by the Ministry of Education in Singapore under the grant MOE2018-T2-2-147 and MOE AcRF R-146-000-271-112. H. Y. was partially supported by the US National Science Foundation under award DMS-1945029. C. Z was partially supported by the Ministry of Education in Singapore under the grant MOE AcRF R-146- 000-271-112 and by NSFC under the grant award 11871364.

REFERENCES

[1] A. R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39(3):930–945, May 1993.

[2] Christian Beck, Sebastian Becker, Patrick Cheridito, Arnulf Jentzen, and Ariel Neufeld. Deep splitting method for parabolic PDEs. *arXiv e-prints*, arXiv:1907.03452, Jul 2019.

[3] Jens Berg and Kaj Nystrm. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, 317:28 – 41, 2018.

[4] J. Braun and M. Griebel. *On a Constructive Proof of Kolmogorov's Superposition Theorem.* Preprint. SFB 611, 2007.

[5] Jian-Feng Cai, Dong Li, Jiaze Sun, and Ke Wang. Enhanced Expressive Power and Fast Training of Neural Networks by Random Projections. *arXiv e-prints*, arXiv:1811.09054, Nov 2018.

[6] Wei Cai and Zhi-Qin John Xu. Multi-scale Deep Neural Networks for Solving High Dimensional PDEs. *arXiv e-prints*, arXiv:1910.11710, Oct 2019.

[7] Charles K. Chui, Shao-Bo Lin, and Ding-Xuan Zhou. Construction of neural networks for realization of localized

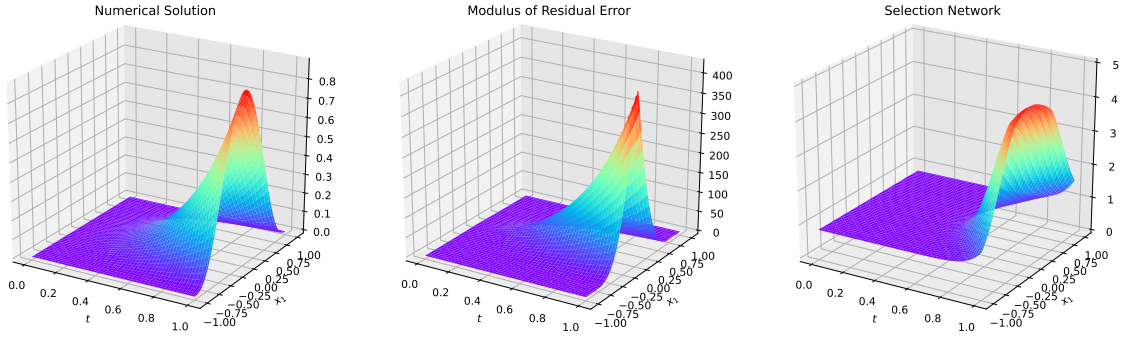


Fig. 6.10: *The (t, x_1)-surfaces of the numerical solution, the modulus of residual errors and selection network by SelectNet (d=20) in the hyperbolic example.*

deep learning. *Frontiers in Applied Mathematics and Statistics*, 4:14, 2018.

[8] Dominik Csiba and Peter Richtárik. Importance sampling for minibatches. *J. Mach. Learn. Res.*, 19(1):962–982, January 2018.

[9] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, Feb 1989.

[10] Constantinos Daskalakis and Ioannis Panageas. The limit points of (optimistic) gradient descent in min-max optimization. In *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, NIPS'18, pages 9256–9266, USA, 2018. Curran Associates Inc.

[11] M. W. M. G. Dissanayake and N. Phan-Thien. Neural-network-based approximations for solving partial differential equations. *Communications in Numerical Methods in Engineering*, 10(3):195–201, 1994.

[12] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011.

[13] Weinan E, Jiequn Han, and Arnulf Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics*, 5(4):349–380, Dec 2017.

[14] Weinan E, Jiequn Han, and Linfeng Zhang. Integrating Machine Learning with Physics-Based Modeling. *arXiv e-prints*, arXiv:2006.02619, 2020.

[15] Weinan E, Chao Ma, and Qingcan Wang. A priori estimates of the population risk for residual networks. *ArXiv*, abs/1903.02154, 2019.

[16] Weinan E, Chao Ma, and Lei Wu. A priori estimates of the population risk for two-layer neural networks. *Communications in Mathematical Sciences*, 17(5):1407 – 1425, 2019.

[17] Weinan E, Chao Ma, and Lei Wu. Barron Spaces and the Compositional Function Spaces for Neural Network Models. *Constructive Approximation*, 2020.

[18] Weinan E and Qingcan Wang. Exponential convergence of the deep neural network approximation for analytic functions. *Sci. China Math.*, 61:1733–1740, 2018.

[19] Weinan E and Bing Yu. The deep ritz method: A deep learning-based numerical algorithm for solving variational problems. *Commun. Math. Stat.*, 6:1–12, 2018.

[20] Matthias Ehrhardt and Ronald E. Mickens. A fast, stable and accurate numerical method for the blackscholes equation of american options. *International Journal of Theoretical and Applied Finance*, 11(05):471–501, 2008.

[21] K. O. Friedrichs. Symmetric positive linear differential equations. *Communications on Pure and Applied Mathematics*, 11(3):333–418, 1958.

[22] Abhijeet Gaikwad and Ioane Muni Toke. Gpu based sparse grid technique for solving multidimensional options pricing pdes. In *Proceedings of the 2Nd Workshop on High Performance Computational Finance*, WHPCF '09, pages 6:1–6:9, New York, NY, USA, 2009. ACM.

[23] D. Gobovic and M. E. Zaghloul. Analog cellular neural network with application to partial differential equations with variable mesh-size. In *Proceedings of IEEE International Symposium on Circuits and Systems - ISCAS '94*, volume 6, pages 359–362 vol.6, May 1994.

[24] Namig J. Guliyev and Vugar E. Ismailov. Approximation capability of two hidden layer feedforward neural networks with fixed weights. *Neurocomputing*, 316:262 – 269, 2018.

[25] Jiequn Han, Arnulf Jentzen, and Weinan E. Solving high-dimensional partial differential equations using deep

learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018.

[26] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359 – 366, 1989.

[27] M. Hutzenthaler, A. Jentzen, Th. Kruse, and T. A. Nguyen. A proof that rectified deep neural networks overcome the curse of dimensionality in the numerical approximation of semilinear heat equations. Technical report, 2020.

[28] Martin Hutzenthaler, Arnulf Jentzen, Thomas Kruse, and Tuan Anh Nguyen. A proof that rectified deep neural networks overcome the curse of dimensionality in the numerical approximation of semilinear heat equations. *SN Partial Differential Equations and Applications*, 1(10), 2020.

[29] Martin Hutzenthaler, Arnulf Jentzen, and von Wurstemberger Wurstemberger. Overcoming the curse of dimensionality in the approximative pricing of financial derivatives with default risks. *Electron. J. Probab.*, 25:73 pp., 2020.

[30] Martin Hutzenthaler, Arnulf Jentzen, and von Wurstemberger Wurstemberger. Overcoming the curse of dimensionality in the approximative pricing of financial derivatives with default risks. *Electron. J. Probab.*, 25:73 pp., 2020.

[31] Lu Jiang, Zhengyuan Zhou, Thomas Leung, Li-Jia Li, and Li Fei-Fei. Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels. *Proceedings of the 35th International Conference on Machine Learning*, 80:2304–2313, 2018.

[32] Angelos Katharopoulos and François Fleuret. Not all samples are created equal: Deep learning with importance sampling. volume 80 of *Proceedings of Machine Learning Research*, pages 2525–2534, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[33] T. Kaufmann, C. Engstrm, and C. Fumeaux. Residual-based adaptive refinement for meshless eigenvalue solvers. In *2010 International Conference on Electromagnetics in Advanced Applications*, pages 244–247, 2010.

[34] Yuehaw Khoo, Jianfeng Lu, and Lexing Ying. Solving parametric pde problems with artificial neural networks. *European Journal of Applied Mathematics*, page 115, 2020.

[35] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[36] M. P. Kumar, Benjamin Packer, and Daphne Koller. Self-paced learning for latent variable models. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 1189–1197. Curran Associates, Inc., 2010.

[37] M. P. Kumar, Benjamin Packer, and Daphne Koller. Self-paced learning for latent variable models. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 1189–1197. Curran Associates, Inc., 2010.

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

[39] I. E. Lagaris, A. C. Likas, and D. G. Papageorgiou. Neural-network methods for boundary value problems with irregular boundaries. *IEEE Transactions on Neural Networks*, 11(5):1041–1049, Sep. 2000.

[40] Hyuk Lee and In Seok Kang. Neural algorithm for solving differential equations. *Journal of Computational Physics*, 91(1):110 – 131, 1990.

[41] T.T. Lee, F.Y. Wang, and R.B. Newell. Robust model-order reduction of complex biological processes. *Journal of Process Control*, 12(7):807 – 821, 2002.

[42] Ke Li, Kejun Tang, Tianfan Wu, and Qifeng Liao. D3M: A Deep Domain Decomposition Method for Partial Differential Equations. *IEEE Access*, 8:5283–5294, 2020.

[43] Qianxiao Li, Bo Lin, and Weiqing Ren. Computing committor functions for the study of rare events using deep learning. *The Journal of Chemical Physics*, 151(5):054112, 2019.

[44] Shiyu Liang and R. Srikant. Why deep neural networks for function approximation? In *ICLR*, 2017.

[45] Yunru Liu, Tingran Gao, and Haizhao Yang. SelectNet: Learning to Sample from the Wild for Imbalanced Data Training. *Proceedings of The First Mathematical and Scientific Machine Learning Conference*, arXiv:1905.09872, 2020.

[46] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6231–6239. Curran Associates, Inc., 2017.

[47] Vitaly Maiorov and Allan Pinkus. Lower bounds for approximation by MLP neural networks. *Neurocomputing*, 25(1):81 – 91, 1999.

[48] A. Malek and R. Shekari Bvolumeokhti. Numerical solution for high order differential equations using a hybrid neural network-optimization method. *Applied Mathematics and Computation*, 183(1):260 – 271, 2006.

[49] Hadrien Montanelli and Qiang Du. New error bounds for deep relu networks using sparse grids. *SIAM Journal*

on Mathematics of Data Science, 1(1), Jan 2019.

[50] Hadrien Montanelli and Haizhao Yang. Error bounds for deep ReLU networks using the KolmogorovArnold superposition theorem. *Neural Networks*, 129:1–6, 2020.

[51] Hadrien Montanelli, Haizhao Yang, and Qiang Du. Deep ReLU networks overcome the curse of dimensionality for bandlimited functions. *Journal of Computational Mathematics*, To appear.

[52] Tenavi Nakamura-Zimmerer, Qi Gong, and Wei Kang. Adaptive Deep Learning for High Dimensional Hamilton-Jacobi-Bellman Equations. *arXiv e-prints*, arXiv:1907.05317, 2019.

[53] Philipp Petersen and Felix Voigtlaender. Optimal approximation of piecewise smooth functions using deep ReLU neural networks. *Neural Networks*, 108:296 – 330, 2018.

[54] Hassan Rafique, Mingrui Liu, Qihang Lin, and Tianbao Yang. Non-convex min-max optimization: Provable algorithms and applications in machine learning. *ArXiv*, abs/1810.02060, 2018.

[55] M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686 – 707, 2019.

[56] Zuowei Shen, Haizhao Yang, and Shijun Zhang. Deep Network Approximation Characterized by Number of Neurons. *arXiv e-prints*, arXiv:1906.05497, Jun 2019.

[57] Zuowei Shen, Haizhao Yang, and Shijun Zhang. Nonlinear approximation via compositions. *Neural Networks*, 119:74 – 84, 2019.

[58] Zuowei Shen, Haizhao Yang, and Shijun Zhang. Deep network with approximation error being reciprocal of width to power of square root of depth. *arXiv:2006.12231*, 2020.

[59] Zuowei Shen, Haizhao Yang, and Shijun Zhang. Neural network approximation: Three hidden layers are enough. 2020.

[60] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339 – 1364, 2018.

[61] Christopher Srinivasa, Inmar Givoni, Siamak Ravanbakhsh, and Brendan J Frey. Min-max propagation. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5565–5573. Curran Associates, Inc., 2017.

[62] Taiji Suzuki. Adaptivity of deep reLU network for learning in besov and mixed smooth besov spaces: optimal rate and curse of dimensionality. In *International Conference on Learning Representations*, 2019.

[63] David J. Wales and Jonathan P. K. Doye. Stationary points and dynamics in high-dimensional systems. *The Journal of Chemical Physics*, 119(23):12409–12416, 2003.

[64] Dmitry Yarotsky. Error bounds for approximations with deep ReLU networks. *Neural Networks*, 94:103 – 114, 2017.

[65] Dmitry Yarotsky. Optimal approximation of continuous functions by very deep ReLU networks. In Sébastien Bubeck, Vianney Perchet, and Philippe Rigollet, editors, *Proceedings of the 31st Conference On Learning Theory*, volume 75 of *Proceedings of Machine Learning Research*, pages 639–649. PMLR, 06–09 Jul 2018.

[66] H. Yserentant. Sparse grid spaces for the numerical solution of the electronic schrödinger equation. *Numerische Mathematik*, 101(2):381–389, Aug 2005.

[67] Yaohua Zang, Gang Bao, Xiaojing Ye, and Haomin Zhou. Weak adversarial networks for high-dimensional partial differential equations. *Journal of Computational Physics*, 411:109409, 2020.