Journal of Computational Physics 475 (2023) 111848

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

Active learning based sampling for high-dimensional nonlinear partial differential equations

Wenhan Gao [a], Chunmei Wang [b,*]

[a] Department of Applied Mathematics and Statistics, Stony Brook University, Stony Brook, NY 11794, USA
[b] Department of Mathematics, University of Florida, Gainesville, FL 32611, USA

ARTICLE INFO

Article history:
Received 28 December 2021
Received in revised form 24 November 2022
Accepted 5 December 2022
Available online 9 December 2022

Keywords:
Deep learning
Active learning
Adaptive sampling
Nonlinear PDEs
High-dimensional

ABSTRACT

The deep-learning-based least squares method has shown successful results in solving high-dimensional and non-linear partial differential equations (PDEs). However, this method usually converges slowly. To speed up the convergence of this approach, an active-learning-based sampling algorithm is proposed in this paper. This algorithm actively chooses the most informative training samples from a probability density function based on residual errors to facilitate error reduction. In particular, points with larger residual errors will have more chances of being selected for training. This algorithm imitates the human learning process: learners are likely to spend more time repeatedly studying mistakes than other tasks they have correctly finished. A series of numerical results are illustrated to demonstrate the effectiveness of our active-learning-based sampling in high dimensions to speed up the convergence of the deep-learning-based least squares method.

1. Introduction

1.1. Problem statement

High-dimensional partial differential equations are widely used in modeling complicated phenomena; e.g., the Hamilton-Jacobi-Bellman (HJB) equation [31] in control theory, the Schrödinger equation [9] in quantum mechanics. There are various traditional numerical methods for solving PDEs, especially for low-dimensional and linear problems. However, when it comes to solving high-dimensional problems, the curse of dimensionality becomes a major computational challenge for many traditional methods such as the finite difference method [42] and the finite element methods where the computational complexity is exponential to the number of dimensions. With that being said, traditional methods oftentimes are computationally intractable in solving high-dimensional problems.

In approximation theory, deep neural networks (DNNs) have been a more effective tool for function approximation than traditional approximation tools such as finite elements and wavelets. In [61,63,49,34,53,18], it has been shown that ReLU-type DNNs can provide more attractive approximation rates than traditional tools to approximate continuous functions and smooth functions with explicit error characterization [49,34,53] in the L^∞-norm [61,63,49,34,53] and in the $W^{s,p}$-norm for $p \in [1,\infty)$ [18]. If target functions lie in the space of functions with special integral representations [2,13,10,55,54, 39], ReLU-type DNNs can lessen the curse of dimensionality in function approximation. Armed with advanced activation functions, DNNs are able to conquer the curse of dimensionality in approximation theory for approximating continuous

* Corresponding author.
E-mail addresses: wenhan.gao@stonybrook.edu (W. Gao), chunmei.wang@ufl.edu (C. Wang).

https://doi.org/10.1016/j.jcp.2022.111848
0021-9991/

functions [50,51,62,52]. In generalization theory, it was shown that DNNs can achieve a dimension-independent error rate for solving PDEs [3,36,33,6]. These theoretical results have justified the application of DNNs to solve high-dimensional PDEs recently in [16,43,44,11,56,19,22]. Two main advantages of deep-learning-based methods presented in these studies can be summarized as follows: firstly, the curse of dimensionality can be weakened or even be overcome in certain classes of PDEs [21,20]; secondly, deep-learning-based PDE solvers are mesh-free without tedious mesh generation for complex domains in traditional solvers. Thus, deep-learning-based methods have shown tremendous potential to surpass other numerical methods especially in solving high-dimensional PDEs in complex domains.

Deep-learning-based PDE solvers have been applied to solve problems in many areas of practical engineering and life sciences [60,45,57]. However, deep-learning-based solvers usually are computationally expensive meaning that the convergence speed is usually slow. To reach a desired precision, the training process can take a very long time. Therefore, it is imperative to speed up the convergence. A recent study [15] proposed a self-paced learning framework, that was inspired by an algorithm named "SelectNet" [32] originally for image classification, to ease the issue of slow convergence. In [15], new neural networks called selection networks were introduced to be trained simultaneously with the residual model based solution network. The selection network learned to provide weights to training points to guide the solution network to learn more from points that have larger residual errors.

In this paper, we aim to address the aforementioned issue of slow convergence by introducing active-learning-based adaptive sampling techniques to sample the most informative training examples for networks to be trained on. This algorithm is motivated by the active learning approach primarily used in supervised learning in computer vision [14,64,41,46], which chooses high-quality data to label so that the network learns faster and less human labeling labor is encountered.

1.2. Related work

1.2.1. Relevant literature review

Mesh-based neural networks are proposed to approximate solution operators for a specific type of differential equations [29,27,37,58,23]. However, these methods become computationally intractable in high dimensions since the degrees of freedom will explode exponentially as the number of dimensions increases. Therefore, the mesh-free model [35,4,12,56,43,24,65,5] is the key to solving high-dimensional PDEs. Mesh-free models use sampling of points and the Monte-Carlo method to approximate the objective loss function to avoid generating meshes that are computationally restrictive in high dimensions. In an early study [27], PDEs are solved via deep neural networks by directly minimizing the degree to which the network approximation violates the PDE at prescribed collocation points, and boundary conditions are treated as a hard constraint by constructing boundary governing functions that satisfy the boundary conditions although designing such functions usually is difficult in complicated PDE problems. In a later study concerning boundary value problems [28], boundary residual errors are also taken into account in a single loss function to enforce boundary conditions. The latter is more commonly used since it does not require auxiliary functions that are problem-dependent and difficult to construct.

Active learning methods are not commonly utilized in deep-learning-based PDE solvers. In [59], active learning is used to select "additional observation locations" to "minimize the variance of the PDE coefficient" and to "minimize the variance of the solution of the PDE". In [35], the residual-based adaptive refinement (RAR) is proposed to adaptively select large residual points. RAR first chooses uniform points and acquires residual errors at these points, and then replicates a certain number of large residual points to the set for training. RAR and our active-learning-based adaptive sampling (to be introduced in Section 2.2) share the same methodology of selecting large residual error points based upon uniform points. The difference is that our adaptive sampling ranks all points. Low residual points will less likely be selected whereas RAR still keeps them. Large residual error points are also ranked, i.e., even if two points both have large residuals, the one with larger residual error is more likely to be selected. In contrast, RAR does not differentiate them and chooses the largest k residual points. In [26], various adaptive strategies have also been introduced. For example, in their adaptive-sampling-in-time approach, collocation points initially come from a small specified time interval and this interval gradually expands. As shown in [26], this approach considerably improves the convergence and overall accuracy. However, the number of training points accumulates in this approach and the computation becomes more expensive. In our adaptive sampling approach, the number of training points does not grow; therefore, in some cases, our adaptive sampling is a better fit although the approaches in [26] are already highly sophisticated.

1.2.2. Deep-learning-based least squares method

Consider the following boundary value problem for simplicity to introduce the deep-learning-based least squares method: find the unknown solution $u(\boldsymbol{x})$ such that

$$\begin{cases} \mathcal{D}u(\boldsymbol{x}) = f(\boldsymbol{x}), & \text{in } \Omega, \\ \mathcal{B}u(\boldsymbol{x}) = g(\boldsymbol{x}), & \text{on } \partial\Omega, \end{cases} \tag{1.1}$$

where $\partial\Omega$ is the boundary of the domain, $\mathcal{D}$ and $\mathcal{B}$ are differential operators in Ω and on $\partial\Omega$, respectively. The goal is to train a neural network, denoted by $\phi(\boldsymbol{x};\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is the set of network parameters, to approximate the ground truth solution $u(\boldsymbol{x})$ of the PDE (1.1). In the least squares method (LSM), the PDE is solved by finding the optimal set of parameters $\boldsymbol{\theta}^*$ that minimizes the loss function; i.e.,

$$\begin{aligned}\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) :&= \arg\min_{\boldsymbol{\theta}} \|\mathcal{D}\phi(\boldsymbol{x};\boldsymbol{\theta}) - f(\boldsymbol{x})\|_2^2 + \lambda\|\mathcal{B}\phi(\boldsymbol{x};\boldsymbol{\theta}) - g(\boldsymbol{x})\|_2^2 \\ &= \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x}\in\Omega}\left[|\mathcal{D}\phi(\boldsymbol{x};\boldsymbol{\theta}) - f(\boldsymbol{x})|^2\right] + \lambda\mathbb{E}_{\boldsymbol{x}\in\partial\Omega}\left[|\mathcal{B}\phi(\boldsymbol{x};\boldsymbol{\theta}) - g(\boldsymbol{x})|^2\right] \\ &\approx \arg\min_{\boldsymbol{\theta}} \frac{1}{N_1}\sum_{i=1}^{N_1} |\mathcal{D}\phi(\boldsymbol{x}_i;\boldsymbol{\theta}) - f(\boldsymbol{x}_i)|^2 + \frac{\lambda}{N_2}\sum_{j=1}^{N_2} \left|\mathcal{B}\phi\left(\boldsymbol{x}_j;\boldsymbol{\theta}\right) - g\left(\boldsymbol{x}_j\right)\right|^2,\end{aligned} \tag{1.2}$$

where λ is a positive hyper-parameter that weights the boundary loss. The last step is a Monte-Carlo approximation with $\boldsymbol{x}_i \in \Omega$ and $\boldsymbol{x}_j \in \partial\Omega$ being N_1 and N_2 allocation points sampled from the respective probability densities that $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ follow. This model will be referred to as the basic model in this paper. For a time-dependent PDE, the temporal coordinate can be regarded as another spatial coordinate to build a similar least squares model.

1.3. Organization

This paper is structured as follows. The active-learning-based adaptive sampling will be introduced in Section 2. The detailed numerical implementation of the proposed sampling method is discussed in Section 3. In Section 4, extensive numerical experiments will be provided to demonstrate the efficiency of the proposed method.

2. Active-learning-based adaptive sampling

2.1. Overview of active learning

Active learning [47,8] is a machine learning method in which the learning algorithm inspects unlabeled data and interactively chooses the most informative data points to learn. Active learning methodology has primarily been applied in computer vision tasks [14], and has shown an extraordinary performance. Active learning for deep learning proceeds in rounds. In each iteration/round, the current model serves to assess the informativeness of training examples. It is usually used for supervised or semi-supervised learning in which there is a large amount of unlabeled data and the algorithm chooses the most informative data points to get labeled for training. There are two most common metrics to measure informativeness: uncertainty [30] and diversity [38]. In the uncertainty sampling, the algorithm tries to find training examples that the current model is most uncertain about, as a proxy of the model output being incorrect. For example, in a classification problem, this uncertainty can be measured by the current model prediction; if for a training example, the model output probability distribution over classes has similar values as opposed to high probability for a single class, then the current model is uncertain about this particular training example. In the diversity sampling [48], the model seeks to identify training examples that most represent the diversity in the data. It is worth mentioning that query by committee (QBC) [66] is also a common active learning approach. Our proposed algorithm will be a diversity sampling approach that focuses on training examples that the neural network model is more uncertain about. It should be pointed out that diversity sampling and uncertainty sampling are not contradictory to each other. One can combine these two approaches to Active Learning.

2.2. Adaptive sampling (AS)

In supervised or semi-supervised learning, the uncertainty sampling algorithm chooses data points that the current model is most uncertain about to get labeled and to be trained on. The least squares method for solving PDEs is an unsupervised learning approach because there is no label in the loss function. However, the idea of uncertainty sampling can still be applied to choose the most informative allocation points to learn. Hence, inspired by the uncertainty sampling, we propose the active-learning-based adaptive sampling in this paper to select high-quality train examples to speed up the convergence of PDE solvers with the basic least squares idea in (1.2).

In our adaptive sampling, the objective is to preferentially choose allocation points with larger absolute residual errors. Intuitively, one can think of large residual error at a point as a proxy of the model being wrong to a greater extent at this particular point. The fundamental methodology of adaptive sampling is to choose from a biased importance distribution that attaches higher priority to important volumes/regions of the domain. For simplicity, the absolute residual error of the network approximation at a point $\boldsymbol{x}$ is defined as follows:

$$\mathcal{R}_{abs}(\boldsymbol{x}) = \begin{cases} |\mathcal{D}\phi(\boldsymbol{x}) - f(\boldsymbol{x})| & \text{if } \boldsymbol{x} \in \Omega, \\ |\mathcal{B}\phi(\boldsymbol{x}) - g(\boldsymbol{x})| & \text{if } \boldsymbol{x} \in \partial\Omega, \end{cases} \tag{2.1}$$

where ϕ is the solution network. The adaptive sampling is thus proposed to choose allocation points following a distribution, denoted by π, given by the probability density function proportional to the distribution of residual errors, that is $q(\boldsymbol{x}) \propto \mathcal{R}_{abs}^p(\boldsymbol{x})$ in Ω and on $\partial\Omega$, respectively. The following generalized density function can be applied to choose allocations in Ω or on $\partial\Omega$, respectively. More precisely, we define

$$q(\boldsymbol{x}) = \frac{\mathcal{R}^p_{abs}(\boldsymbol{x})}{NC} \tag{2.2}$$

where $NC = \int_{\Omega \text{ or } \partial\Omega} \mathcal{R}^p_{abs}(\boldsymbol{x}) d\boldsymbol{x}$ is an unknown normalizing constant, p is a non-negative constant exponent that controls the effect of adaptive sampling. With larger p, larger residual error points are more likely to be sampled; when $p = 0$, there is no effect of adaptive sampling.

As one may have noticed, the unknown normalizing constant is difficult to calculate. Therefore, one may not be able to directly sample points from $q(\boldsymbol{x})$. Two techniques to simulate observations from $q(\boldsymbol{x})$ are reviewed in the following.

2.2.1. Metropolis Hastings sampling

The Metropolis-Hastings (MH) algorithm [7] is a commonly used Markov chain Monte Carlo (MCMC) method [1]. Markov chain comes in because the next sample depends on the current sample. Monte Carlo comes in because it simulates observations from a target distribution that is difficult to directly sample from. The Metropolis-Hastings algorithm will be adopted to select points from the target distribution $q(\boldsymbol{x})$. Note that all algorithms sample points in Ω for training, and the same logic follows for sampling on $\partial\Omega$.

Algorithm 2.1: Metropolis Hastings Sampling.

Result: N_1 points in Ω for training
Require: PDE (1.1); the current solution net $\phi(\boldsymbol{\theta})$
Initialize a random point $x_0 \in \Omega$, $j := 0$ and $m = N_1 + b$, where b is a positive integer ;
while $j < m$ **do**
  $j := j + 1$;
  Draw $x_{candidate} \sim Unif(\Omega)$, and $u \sim Unif(0, 1)$;
  acceptance_rate = $\frac{\mathcal{R}^p_{abs}(x_{candidate})}{\mathcal{R}^p_{abs}(x_{j-1})}$;
  if $u <$ *acceptance_rate* **then**
    $x_j = x_{candidate}$;
  else
    $x_j = x_{j-1}$;
  end
end
return last N_1 points;

In Algorithm 2.1, the first b points will be discarded which are called "burn-ins" where b is a non-negative integer. It is a computational trick to correct the bias of early samples. In practice, b can be 0. However, there is no rigorous justification for burn-ins based on the best of our knowledge. Note that in actual implementation, Algorithm 2.2 in a vectorized version following the coding style in Python will be deployed.

Algorithm 2.2: Vectorized Metropolis Hastings Sampling.

Result: N_1 points in Ω for training
Require: PDE (1.1); the current solution net $\phi(\boldsymbol{x}; \boldsymbol{\theta})$
Generate an array of $N_1 + b$ uniform points $\boldsymbol{X} \subset \Omega$;
RE_array := $\mathcal{R}^p_{abs}(\boldsymbol{X}) = |\mathcal{D}\phi(\boldsymbol{X}; \boldsymbol{\theta}) - f(\boldsymbol{X})|^p$;
Generate an array of $N_1 + b$ uniform points U following a uniform distribution on $(0, 1)$;
for *i in range* $(0, N_1 + b - 1)$ **do**
  if *RE_array[i+1] < RE_array[i]* **then**
    acceptance_rate = $\frac{\text{RE\_array[i+1]}}{\text{RE\_array[i]}}$;
    if *acceptance_rate < U[i+1]* **then**
      $\boldsymbol{X}[i+1] = \boldsymbol{X}[i]$
    end
  end
end
return the last N_1 points in $\boldsymbol{X}$;

2.2.2. Self-normalized sampling

The Metropolis-Hastings Algorithm sometimes can be computationally expensive, and the convergence of this algorithm can be slow. Therefore, an alternative algorithm is proposed to sample training examples. This algorithm is faster than the MH algorithm because this algorithm is entirely parallel in generating points as opposed to the MH algorithm in which each sample point is based on the previous sample; therefore, the MH algorithm is not parallel. Our proposed new sampling technique will be used in the numerical experiment.

This algorithm is named "Self-normalized Sampling" because the discrete probability density function obtained by this algorithm normalizes itself. It is well-known that the marginal distribution of points generated by the Metropolis-Hastings algorithm will converge to the target distribution; the self-normalized sampling algorithm does efficiently generate a set of

Algorithm 2.3: Self-normalized Sampling.

Result: N_1 points in Ω for training
Require: PDE (1.1); the current solution net $\phi(\boldsymbol{x};\boldsymbol{\theta})$
1 Generate N_1 uniformly distributed points $\{\boldsymbol{x}_i\}_{i=1}^{N_1} \subset \Omega$; denote by $\boldsymbol{X}$;
2 RE_array := $\mathcal{R}_{abs}^{p}(\boldsymbol{X}) = |\mathcal{D}\phi(\boldsymbol{X};\boldsymbol{\theta}) - f(\boldsymbol{X})|^p$;
3 SRE = sum(RE_array) ;
4 probability_array = $\frac{\text{RE\_array}}{\text{SRE}}$;
5 Generate N_1 points, $\boldsymbol{X}$_training, following the discrete probability density function $f(\text{RE\_array}[i])$ = probability_array$[i]$;
6 **return** $\boldsymbol{X}$_training ;

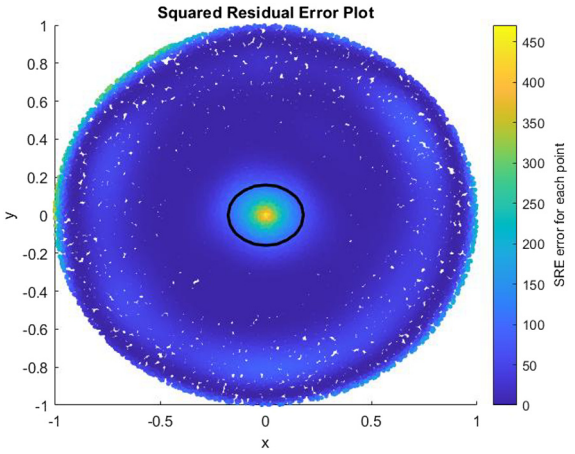


Fig. 2.1. Distribution of Squared Residual Errors. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

Table 1
The Number of Points Inside Ellipse.

Approach	Number of Points
Uniform Annular	61
Metropolis Hastings	210
Self-normalized Sampling	255

points that focuses on large-residual-error volumes in the experiment. Moreover, in Appendix D, we will discuss why this self-normalized algorithm makes sense.

In general, self-normalized sampling is less computationally expensive than the Metropolis-Hastings algorithm. However, in some rare cases, the residual error is only large in some extremely small volumes and relatively low elsewhere; N_1 points generated in step 1 of Algorithm 2.3 can fail to have any point inside these large error volumes. A naive remedy to this is to generate more than N_1 points in step 1 so that it covers a broader range of points. On the other hand, the Metropolis-Hastings algorithm can substantially sample points on these small volumes although the number of burn-ins needs to be very large to reach these volumes.

A demonstration of points generated in 2D by Algorithms 2.2 and 2.3 are presented. Fig. 2.1 is an example of a distribution of squared residual errors. Figs. 2.3 and 2.4 are distributions of 500 points generated by the Metropolis-Hastings algorithm and the self-normalized sampling, respectively, with $p = 1$ and 3500 burn-ins for the Metropolis-Hastings algorithm; the uniform distribution used in Algorithms 2.2 and 2.3 is replaced by uniform annular distribution [15]. Fig. 2.2 is the distribution of 500 points generated by a uniform annular distribution. The number of points inside of the ellipse $\frac{x^2}{0.18^2} + \frac{y^2}{0.16^2} = 1$ generated by each approach are shown in Table 1.

2.3. Adaptive sampling: a statistical viewpoint

Adaptive sampling is motivated by active learning that takes values of residual errors as a proxy of the model being wrong. It is natural to connect adaptive sampling with uncertainty sampling that takes uncertainty as a proxy of the model being wrong. Adaptive sampling can be explained from a statistical viewpoint. The definition of expectation and Monte-Carlo estimate is in Appendix B.

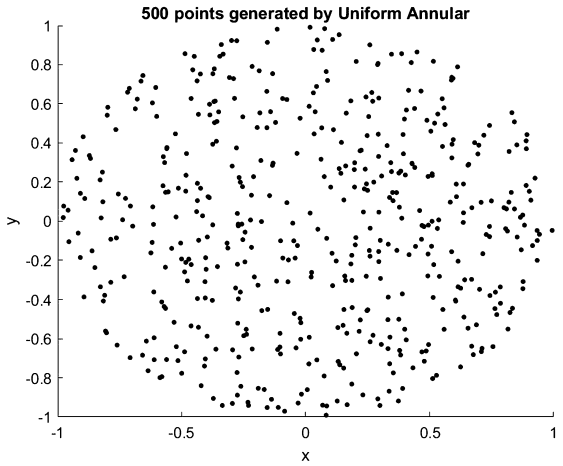


Fig. 2.2. 500 Points Generated by Uniform Annular.

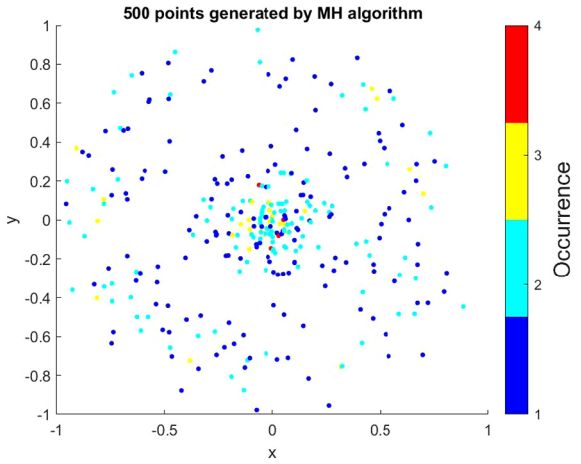


Fig. 2.3. 500 Points Generated by Metropolis.

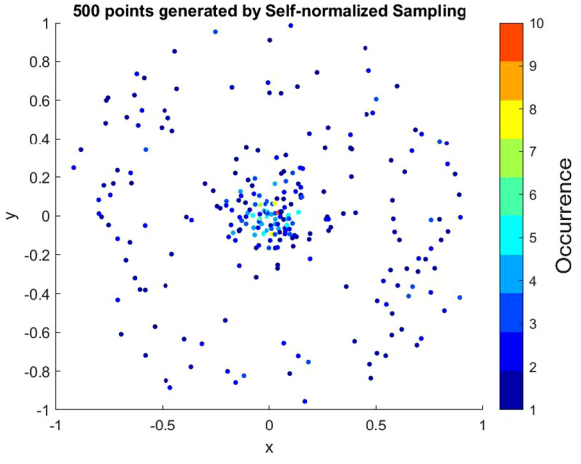


Fig. 2.4. 500 Points Generated by Self-normalized.

Table 2
The Setting of the Solution Network.

Symbol	Meaning	Value
L	Depth of the Network	3
m	Width of the Network	100
σ	Activation Function	$ReLU^3 = max(x^3, 0)$
d	Dimension of Spatial Coordinates	to be specified
τ	Learning Rate for Solution Network ϕ	Specified in (3.4)

Definition 2.1 *(Owen, 2013 [40]).* Let $\boldsymbol{x} \in \Omega \subset \mathbb{R}^d$ be a random variable and $w(\boldsymbol{x})$ be any function defined on Ω with density $p(\boldsymbol{x})$. Let q(x) be another probability density on Ω such that $q(\boldsymbol{x}) = 0$ implies $w(\boldsymbol{x})p(\boldsymbol{x}) = 0$. $q(\boldsymbol{x})$ is also called biasing distribution or important distribution. The importance sampling estimate is defined by

$$\hat{\mu}_q = \frac{|\Omega|}{n} \sum_{i=1}^{n} \frac{w(\boldsymbol{x}_i)\, p(\boldsymbol{x}_i)}{q(\boldsymbol{x}_i)}, \boldsymbol{x}_i \sim q(\boldsymbol{x}).$$

Denote by $Unif(\Omega)$ a uniform distribution on Ω. Note that if $p(\boldsymbol{x}) = Unif(\Omega)$, then $|\Omega|$ and $p(\boldsymbol{x}) = \frac{1}{|\Omega|}$ cancel out each other.

Theorem 2.1 *(Owen, 2013 [40]). The importance sampling estimate $\hat{\mu}_q$ is an unbiased estimate of μ.*

As one may have noticed, Theorem 2.1 can be applied to estimate expectations in Equation (1.2), where $|\mathcal{D}\phi(\boldsymbol{x};\boldsymbol{\theta}) - f(\boldsymbol{x})|^2 = w(\boldsymbol{x})$, $q(\boldsymbol{x})$ is defined in (2.2), and $p(\boldsymbol{x}) = Unif(\Omega)$. The importance sampling estimate of $\mu = \mathbb{E}_{\boldsymbol{x}\in\Omega}\big[|\mathcal{D}\phi(\boldsymbol{x};\boldsymbol{\theta}) - f(\boldsymbol{x})|^2\big] = \mathbb{E}_{\boldsymbol{x}\in\Omega}\big[w(\boldsymbol{x})\big]$ is thus $\hat{\mu}_q = \frac{1}{n}\sum_{i=1}^{n} \frac{w(\boldsymbol{x}_i)}{q(\boldsymbol{x}_i)}$, $\boldsymbol{x}_i \sim q(\boldsymbol{x})$.

One way to justify adaptive sampling is to adopt the notion of active learning. Adaptive sampling can also be understood as a better estimating approach. In (1.2), the plain Monte-Carlo estimate will face the issue of slow convergence if, for example, the distribution of squared residual errors is large at a small volume $\mathcal{V} \subset \Omega$ but is low, or nearly zero, outside of $\mathcal{V}$. The plain Monte-Carlo estimate could fail to have even one point inside the volume $\mathcal{V}$. Faster convergence can be achieved "by sampling from a distribution that overweights the important region, hence the name importance sampling" [40]. This paper is not concerned with finding an unbiased estimate to the expectation but finding the best parameter $\boldsymbol{\theta}$ that minimizes the expectation. Therefore, allocation points will be sampled from $q(\boldsymbol{x})$ without altering the formulation in the last step of (1.2) since only $w(\boldsymbol{x})$ depends on $\boldsymbol{\theta}$.

3. Setup of numerical implementation

3.1. Network architecture

Note that adaptive sampling introduced in Section 2 is an active-learning-based sampling technique that is independent of the choice of network architectures. In the numerical implementation, the solution network $\phi(\boldsymbol{x};\boldsymbol{\theta})$ is a feed-forward neural network in which each layer is fully connected. The network can be expressed as a composition of several activation functions; i.e.,

$$\begin{aligned} &\phi(\boldsymbol{x};\boldsymbol{\theta}) = h_L \circ h_{L-1} \circ \ldots \circ h_1 \circ h_0(\boldsymbol{x}), \\ &h_l(\boldsymbol{x}^l) = \sigma(\boldsymbol{W}^l \boldsymbol{x}^l + b^l), \text{ for } l = 0, 1, 2, \ldots, L-1, \qquad h_L(\boldsymbol{x}^L) = \boldsymbol{W}^L(\boldsymbol{x}^L + b^L), \end{aligned} \tag{3.1}$$

where L is the depth of the network, σ is a nonlinear activation function, $\boldsymbol{W}^0 \in \mathbb{R}^{m\times d}$, $\boldsymbol{W}^L \in \mathbb{R}^{m\times 1}$, $\boldsymbol{W}^l \in \mathbb{R}^{m\times m}$, $l = 1, \cdots, L-1$, $\boldsymbol{b}^L \in \mathbb{R}$, $\boldsymbol{b}^l \in \mathbb{R}^{m\times 1}$, $l = 0, \cdots, L-1$, m is the number of nodes or neurons in a layer and called the width of the network, d is the dimension of the problem and $\boldsymbol{\theta} = \{\boldsymbol{W}^l, \boldsymbol{b}^l\}_{l=0}^{L}$. Network setting parameters used in the numerical implementation are listed in Table 2.

The network is trained to solve the minimization problem defined in (1.2) via Adam [25] (a variant of stochastic gradient descent), where the empirical loss is defined by

$$J(\boldsymbol{\theta}) = \frac{1}{N_1}\sum_{i=1}^{N_1} |\mathcal{D}\phi(\boldsymbol{x}_i;\boldsymbol{\theta}) - f(\boldsymbol{x}_i)|^2 + \frac{\lambda}{N_2}\sum_{j=1}^{N_2} \left|\mathcal{B}\phi\left(\boldsymbol{x}_j;\boldsymbol{\theta}\right) - g\left(\boldsymbol{x}_j\right)\right|^2, \tag{3.2}$$

and $\boldsymbol{\theta}$ is updated by

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \tau \nabla J(\boldsymbol{\theta}), \tag{3.3}$$

where τ is the learning rate of the solution network defined by

$$\begin{aligned}&\tau^{(k)} = 10^{-3-(3j/1000)}, \text{ for } \left\lceil \frac{0.999n}{1000} j \right\rceil \le k < \left\lceil \frac{0.999n}{1000}(j+1) \right\rceil, \ j = 0, 1, 2, ..., 999,\\ &\tau^{(k)} = 10^{-6}, \text{ for } \left\lceil 0.999n \right\rceil \le k < n,\end{aligned} \tag{3.4}$$

where $\tau^{(k)}$ denotes the learning rate at the k-th epoch, and n is the number of total epochs. Index starts from 0, i.e., the learning rate for the first epoch is $\tau^{(0)}$.

3.2. Derivatives of networks

In order to calculate residual errors, $\mathcal{D}\phi(\boldsymbol{x};\boldsymbol{\theta})$ needs to be evaluated. It is well-known that Autograd can perform such differentiation. However, in our experiment, we observe that Autograd does not output precise partial derivatives in high dimensions. Therefore, in our implementation, the numerical differentiation (see Appendix C) with step size $h = 0.0001$ will be used to estimate partial derivatives except in Section 4.5, where Autograd is used since it outputs correct partial derivatives in 2D. For example, let $\phi(\boldsymbol{x};\boldsymbol{\theta})$ be the solution network with $\boldsymbol{x} \in \mathbb{R}^d$ and parameters $\boldsymbol{\theta}$. The numerical differentiation estimate of $\phi(\boldsymbol{x};\boldsymbol{\theta})$, denoted by $\nabla\phi(\boldsymbol{x};\boldsymbol{\theta})$, is defined by

$$\nabla\phi(\boldsymbol{x};\boldsymbol{\theta}) \approx \frac{1}{h}\sum_{i=1}^{d}\left[\phi(\mathbf{a} + h\boldsymbol{e}_i;\boldsymbol{\theta}) - \phi(\mathbf{a};\boldsymbol{\theta})\right],$$

where $\mathbf{a} \in \mathbb{R}^d$, $\boldsymbol{e}_i$ is a vector of all 0's except a 1 in the i-th entry, and $h \in \mathbb{R}$ is the step size. Note that, as found in [15], with a step size of 10^{-4}, the truncation error is up to $O(10^{-6})$, and it can be ignored in practice since the final error is at least $O(10^{-4})$. We will resolve the issue that Autograd does not output precise partial derivatives in our future work. As mentioned above, numerical differentiation only causes negligible errors in this work.

3.3. Training solution networks with active sampling

Algorithm 3.1: Training Solution Networks.

Result: parameters $\boldsymbol{\theta}^*$
Require: PDE (1.1)
Set $n =$ total iterations/epoch, N_1 and N_2 for size of sample points in Ω and on $\partial\Omega$ respectively ;
Initialize $\phi(\boldsymbol{x};\boldsymbol{\theta})$; **while** $k < n$ **do**
  Generate sampling training points by Algorithm 2.2 or 2.3, $\left\{\boldsymbol{x}_i^1\right\}_{i=1}^{N_1} \subset \Omega$ and $\left\{\boldsymbol{x}_j^2\right\}_{j=1}^{N_2} \subset \partial\Omega$;
  Loss: $J(\boldsymbol{\theta}) = \frac{1}{N_1}\sum_{i=1}^{N_1}\left|\mathcal{D}\phi\left(\boldsymbol{x}_i^1;\boldsymbol{\theta}\right) - f\left(\boldsymbol{x}_i^1\right)\right|^2 + \frac{\lambda}{N_2}\sum_{j=1}^{N_2}\left|\mathcal{B}\phi\left(\boldsymbol{x}_j^2;\boldsymbol{\theta}\right) - g\left(\boldsymbol{x}_j^2\right)\right|^2$;
  Update: $\boldsymbol{\theta} := \boldsymbol{\theta} - \tau\nabla J(\boldsymbol{\theta})$;
end
return $\boldsymbol{\theta}^* := \boldsymbol{\theta}$;

3.4. Uniform annular distribution

The uniform annular approach will be employed to sample points in Ω in the first step of Algorithm 2.2 and 2.3. The uniform annular approach divides the interior of the domain Ω 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, where N_1 is the number of training points in Ω in each epoch, and N_a is the number of annuli; in practice N_a should be divided by N_1, i.e., $N_a|N_1$. This distribution is also utilized in [15]. As a side note, this uniform annular distribution approach can also be understood as a biased distribution that covers a broader range of points in Ω. Unlike the adaptive sampling distribution in (2.2), the uniform annular distribution is not necessarily a better distribution for the model to learn for each PDE example. It is worth pointing out that the domain Ω considered in this paper is always a high-dimensional unit ball or unit cube. Therefore, this uniform annular distribution can be employed. In the general case, it is difficult to apply the uniform annular distribution.

3.5. Accuracy assessment

To measure the accuracy of the model, 10000 testing points $\left\{\boldsymbol{x}_i^t\right\}_{i=1}^{10000} \subset \Omega$, which are different from training points, will be sampled from the uniform annular distribution to approximate the overall relative ℓ_2 error and the relative ℓ_1 maximum modulus error at these points. Note that in the implementation, the random seed is fixed which implies that for the same PDE example, all models will be tested on the same set of uniform annular points to measure accuracy. The overall relative ℓ_2 error and the relative ℓ_1 maximum modulus error are, respectively, defined as follows

Table 3
Parameter Setting.

Symbol	Meaning	Value
n	Number of Epochs	20000
N_1	Number of Training Points in Ω in Each Epoch	12000
N_2 for Example (4.1)	Number of Training Points on $\partial\Omega$ in Each Epoch	12000
N_2 for Example (4.2)	Number of Training Points on $\partial\Omega$ in Each Epoch	$12000 + \frac{12000}{d}$
N_2 for Example (4.3)	Number of Training Points on $\partial\Omega$ in Each Epoch	$12000 + \frac{12000}{d}$
λ	Boundary Loss Weighting Term in (4.3)	10

Table 4
Example 4.1 Result; Error Reduction Formula: $1 - \frac{error(AS)}{error(Basic)}$.

Dimension		AS	Basic	RAR	Error Reduced by AS
10	ℓ_2 error	8.784735e-03	2.526952e-02	2.024158e-02	65.24%
	max ℓ_1 error	3.681198e-02	1.336612e-01	9.741917e-02	72.46%
	time in Sec	8179.919345	6155.072389	7423.530141	
20	ℓ_2 error	3.102093e-02	7.198276e-02	6.345444e-02	56.90%
	max ℓ_1 error	1.145958e-01	2.719710e-01	3.157725e-01	57.86%
	time in Sec	12489.657977	7985.964646	9022.221589	
100	ℓ_2 error	1.205655e-01	3.964277e-01	3.423974e-01	69.59%
	max ℓ_1 error	3.280543e-01	1.246953e+00	1.016873e+00	73.69%
	time in Sec	43374.188778	32276.845542	37841.390514	

$$e_{\ell^2}^{overall}(\boldsymbol{\theta}) := \frac{\left(\sum_{i=1}^{10000} \left|\phi\left(\boldsymbol{x}_i^t;\boldsymbol{\theta}\right) - u\left(\boldsymbol{x}_i^t\right)\right|^2\right)^{\frac{1}{2}}}{\left(\sum_{i=1}^{10000} \left|u\left(\boldsymbol{x}_i^t\right)\right|^2\right)^{\frac{1}{2}}}, \qquad e_{modulus}^{max}(\boldsymbol{\theta}) := \frac{max\left(\left|\phi\left(\boldsymbol{x}_i^t;\boldsymbol{\theta}\right) - u\left(\boldsymbol{x}_i^t\right)\right|\right)_{i=1}^{10000}}{max\left(\left|u\left(\boldsymbol{x}_i^t\right)\right|\right)_{i=1}^{10000}}.$$

4. Numerical experiment

In this section, our adaptive sampling is tested on three types of PDE examples: elliptic, parabolic, and hyperbolic equations. Testing includes various dimensions. In this section, we will always use Algorithm 2.3 (Self-normalized Sampling) to sample training points. In our experiments, Algorithm 2.2 (Metropolis Hastings Sampling) and Algorithm 2.3 (Self-normalized Sampling) give similar results. It is worth mentioning that the main focus of this work is to "choose" training examples based on the residual error distribution, and it is independent of the algorithm that is used to simulate the residual error distribution. Therefore, one may use other algorithms, such as Gibbs sampling, to simulate the distribution as well. Adaptive sampling will be compared with RAR in Examples 4.1, 4.2, and 4.3. In the comparison experiments, we only replace AS with RAR (in line 3 of Algorithm 3.1, AS is replaced by RAR), and all other settings are kept the same. The RAR algorithm used in this work (see Algorithm A.1) is slightly different from the algorithm proposed in the paper [35] in order to make the comparison in which the number of training points will still be fixed. m in the algorithm proposed in the paper [35] is chosen to be 2000 for testing. Finally, our adaptive sampling will be applied to some current frameworks to test if adaptive sampling is compatible with these frameworks. The parameter setting for Sections 4.1 and 4.2 is listed in Table 3.

4.1. Elliptic equation

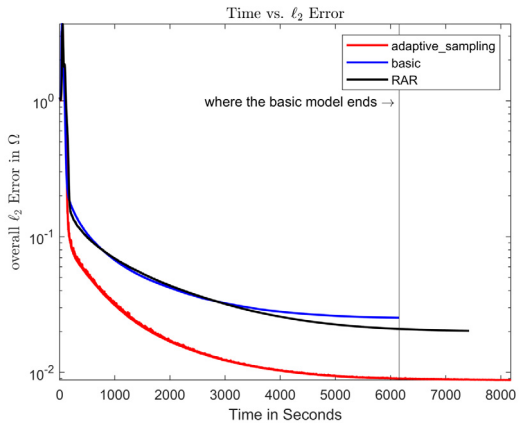
We consider a nonlinear elliptic equation:

$$\begin{aligned} -\nabla\cdot\left((1+\frac{1}{2}|\boldsymbol{x}|^2)\nabla u\right) + (\nabla u)^2 &= f(\boldsymbol{x}), \quad \text{in } \Omega := \{\boldsymbol{x} : |\boldsymbol{x}| < 1\}, \\ u &= g(\boldsymbol{x}), \quad \text{on } \partial\Omega, \end{aligned} \tag{4.1}$$

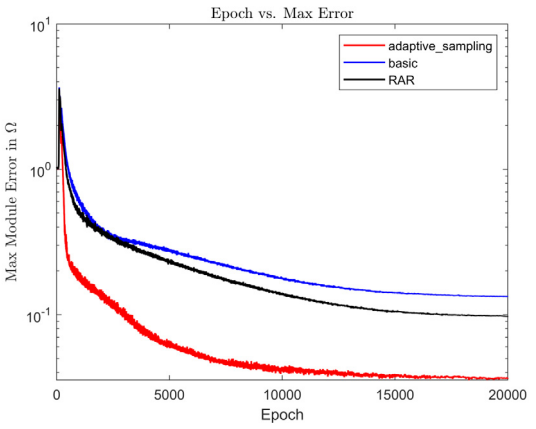
where $g(\boldsymbol{x}) = 0$ and f is given appropriately so that the exact solution is given by $u(\boldsymbol{x}) = sin(\frac{\pi}{2}(1-|\boldsymbol{x}|)^{2.5})$.

Adaptive sampling is tested on this PDE example in various high dimensions: 10, 20, and 100. Adaptive sampling will be compared with the basic least squares method discussed in Section 1.2.2. Since the computational costs of these two models are different, for this particular example, error decay versus time in seconds is also displayed to demonstrate the efficiency of adaptive sampling. Running time is obtained by training on Quadro RTX 8000 Graphics Card. Only training time has been taken into account; time for accuracy assessment and saving data is not included. The overall relative error seems very large in 100 dimensions because the true solution vanishes in most of the volume when the dimension is high.

Table 4 shows the overall relative ℓ_2 error and maximum relative ℓ_1 modulus error by adaptive sampling and the basic residual model at the end of 20000 epochs. Time versus ℓ_2 error decay in 10 dimensions, 20 dimensions, and 100 dimen-

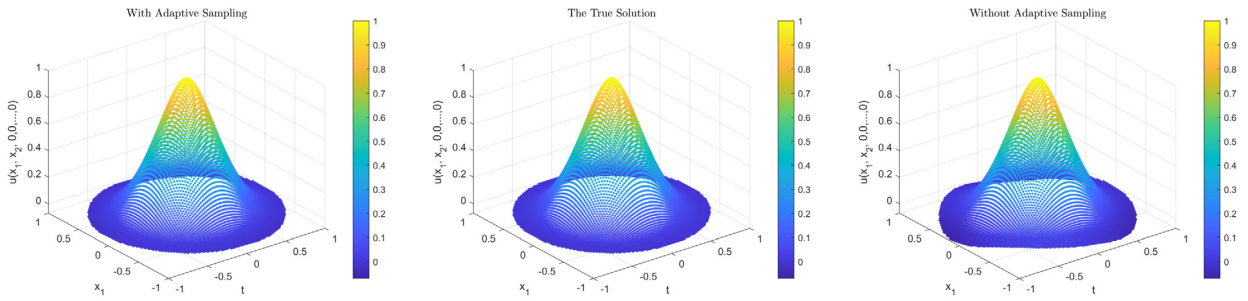


(a) Time vs. Overall ℓ_2 Error in Ω



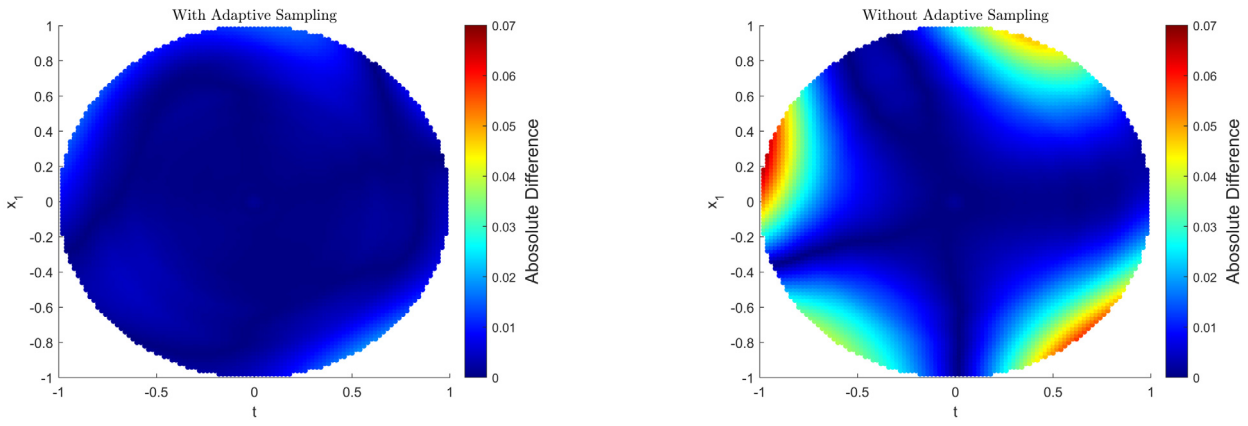
(b) Epoch vs. Max Modulus Error in Ω

Fig. 4.1. Example 4.1 numerical results in 10 dimensions.



(a) With Adaptive Sampling (b) The True Solution (c) Without Adaptive Sampling

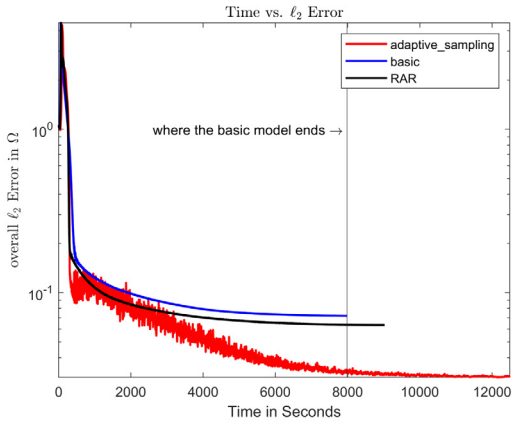
Fig. 4.2. Example 4.1 10 dimensions $(x_1, x_2, 0, 0, ..., 0)$-surface of network solutions and the true solution.



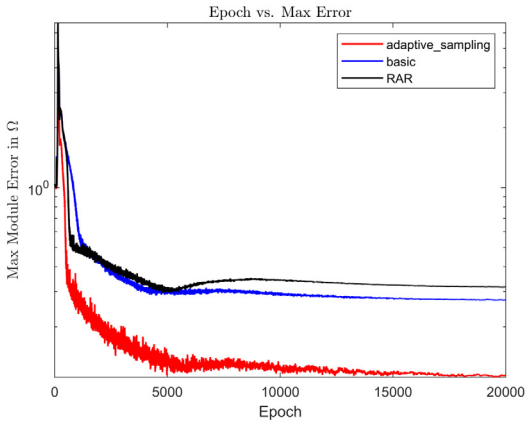
(a) With Adaptive Sampling (b) Without Adaptive Sampling

Fig. 4.3. Example 4.1 10 dimensions $(x_1, x_2, 0, 0, ..., 0)$-surface absolute difference $|u - \phi|$.

sions are demonstrated in Figs. 4.1, 4.4, 4.5, respectively. Fig. 4.2 shows the (x_1, x_2)-surface of the ground-truth solution and network solutions with and without adaptive sampling, where axes are x_1, x_2, and $u(x_1, x_2, 0, 0, ..., 0)$. Fig. 4.3 shows the heat-map of the absolute difference in (x_1, x_2)-surface, where the color bar represents the absolute difference between the ground-truth solution and network solution. These results clearly show that adaptive sampling helps to reduce the error, especially relative l_1 maximum modulus error. More precisely, adaptive sampling helps to significantly reduce the variance

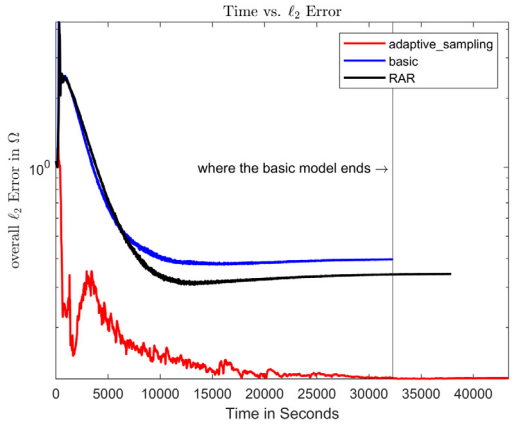


(a) Time vs. Overall ℓ_2 Error in Ω

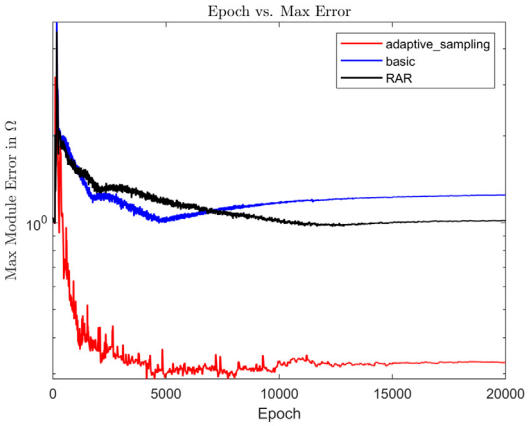


(b) Epoch vs. Max Modulus Error in Ω

Fig. 4.4. Example 4.1 numerical results in 20 dimensions.



(a) Time vs. Overall ℓ_2 Error in Ω



(b) Epoch vs. Max Modulus Error in Ω

Fig. 4.5. Example 4.1 numerical results in 100 dimensions.

in the distribution of error over the domain so that, compared to the results without adaptive sampling, it is less likely to have volumes/areas where the error is relatively much larger.

4.2. Parabolic equation

We consider the following parabolic equation:

$$\begin{aligned} \partial_t u(\boldsymbol{x},t) - \nabla_{\boldsymbol{x}} \cdot \left((1+\frac{1}{2}|\boldsymbol{x}|)\nabla_{\boldsymbol{x}} u(\boldsymbol{x},t) \right) &= f(\boldsymbol{x},t), \quad \text{in } \Omega := \omega \times \mathbb{T}, \\ u(\boldsymbol{x},t) &= g(\boldsymbol{x},t), \quad \text{on } \partial\Omega = \partial\omega \times \mathbb{T}, \\ u(\boldsymbol{x},0) &= h(\boldsymbol{x}), \quad \text{in } \omega, \end{aligned} \tag{4.2}$$

where $\omega := \{\boldsymbol{x} : |\boldsymbol{x}| < 1\}$, $\mathbb{T} = (0,1)$, and

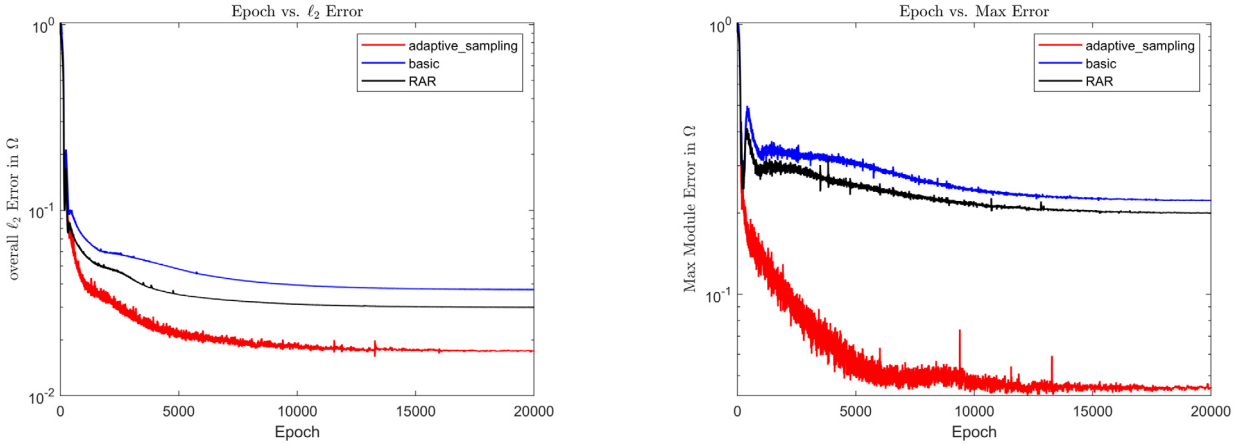
$$g(\boldsymbol{x}) = e^{|\boldsymbol{x}|\sqrt{1-t}},$$
$$h(\boldsymbol{x}) = exp(|\boldsymbol{x}|),$$

f is given appropriately so that the exact solution is given by $u(\boldsymbol{x},t) = e^{|\boldsymbol{x}|\sqrt{1-t}}$.

Table 5 shows the overall relative l_2 error and maximum relative l_1 modulus error by adaptive sampling and the basic residual model at the end of 20000 epochs. The number of epochs versus ℓ_2 error decay in 10 dimensions and 20 dimensions are illustrated in Figs. 4.6 and 4.9 respectively. Fig. 4.7 shows the (t, x_3)-surface of the ground-truth solution and

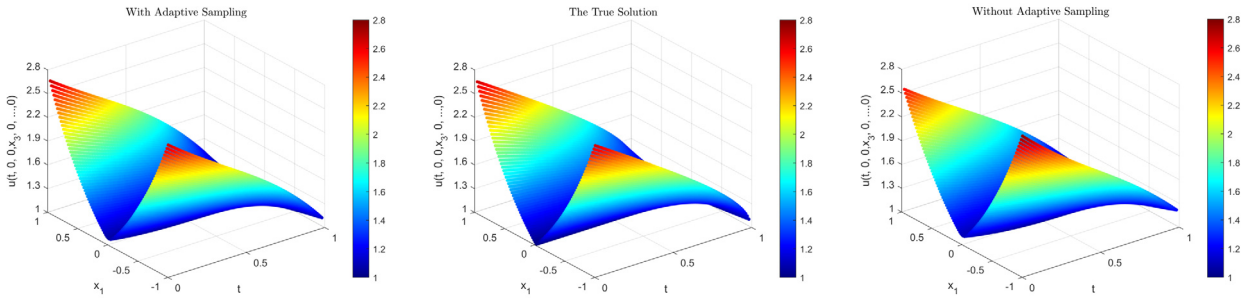
Table 5
Example 4.2 Result.

Dimension		AS	Basic	RAR	Error Reduced by AS
10	ℓ_2 error	1.731916e-02	3.735915e-02	2.997037e-02	53.54%
	Max Modulus Error	4.530897e-02	2.229580e-01	2.000248e-01	79.68%
20	ℓ_2 error	2.688834e-02	5.709440e-02	4.336126e-02	52.91%
	Max Modulus Error	9.229024e-02	2.167297e-01	1.748666e-01	57.42%



(a) Epoch vs. Overall Relative ℓ_2 Error in Ω (b) Epoch vs. Max Relative Modulus Error in Ω

Fig. 4.6. Example 4.2 numerical results in 10 dimensions.



(a) With Adaptive Sampling (b) The True Solution (c) Without Adaptive Sampling

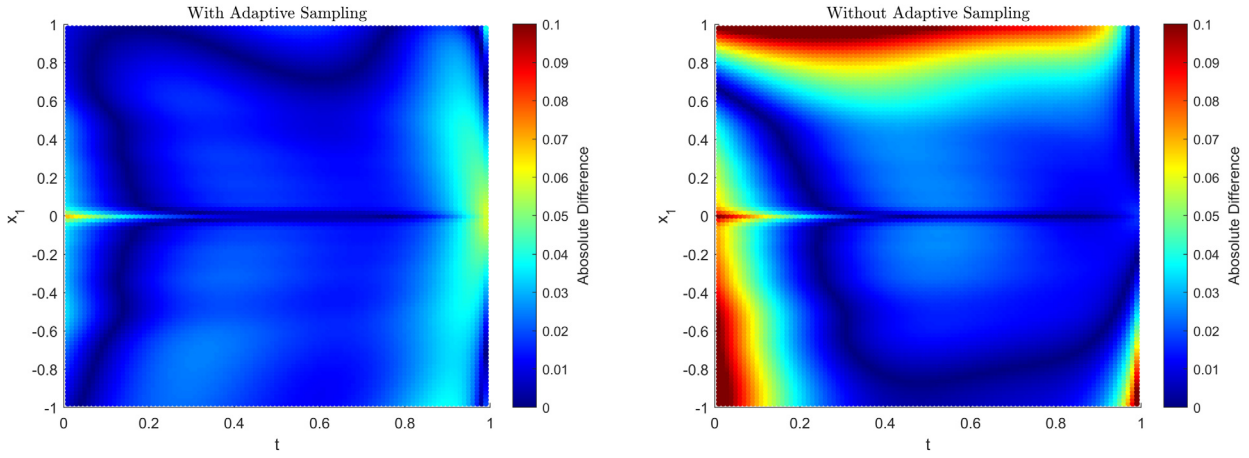
Fig. 4.7. Example 4.2 10 dimensions $(t, 0, 0, x_3, 0, ..., 0)$-surface of network solutions and the true solution.

network solutions with and without adaptive sampling, where axes are t, x_3, and $u(t, 0, 0, x_3, 0, ..., 0)$. Fig. 4.8 shows the heat-map of the absolute difference in (t, x_3)-surface, where the color bar represents the absolute difference between the ground-truth solution and network solution. This example evidently demonstrates the advantage of adaptive sampling in reducing the variance in the distribution of error over the domain. These results clearly show that adaptive sampling helps to reduce the error, especially relative l_1 maximum modulus error.

4.3. Hyperbolic equation

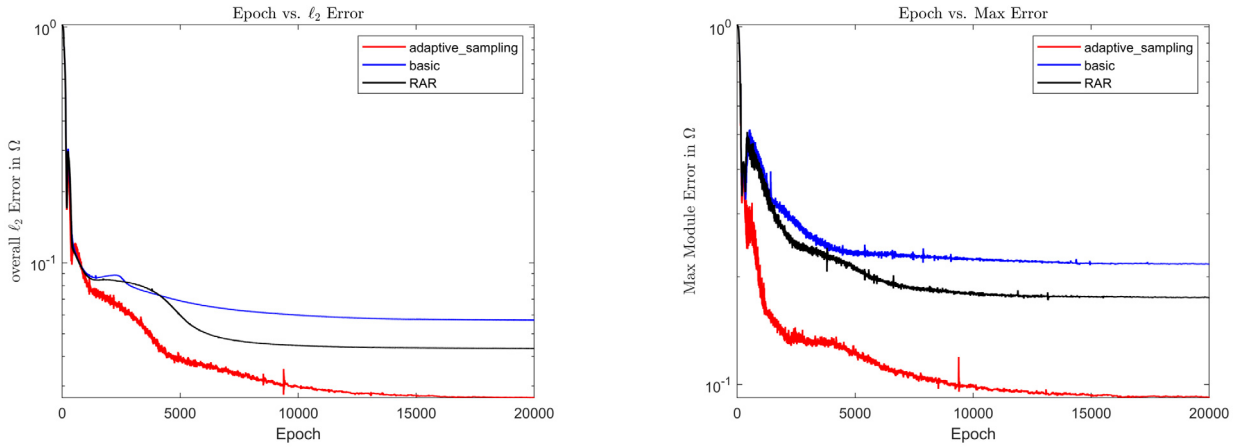
Lastly, we consider the following hyperbolic equation:

$$\begin{aligned} \frac{\partial^2 u(x,t)}{\partial t^2} - \Delta_x u(x,t) &= f(x,t), \quad \text{in } \Omega := \omega \times \mathbb{T}, \\ u(x,t) &= g_0(x,t), \quad \text{on } \partial\Omega = \partial\omega \times \mathbb{T}, \\ u(x,0) = h_0(x) \quad \frac{\partial u(x,0)}{\partial t} &= h_1(x), \quad \text{in } \omega, \end{aligned} \tag{4.3}$$



(a) With Adaptive Sampling (b) Without Adaptive Sampling

Fig. 4.8. Example 4.2 10 dimensions $(t, 0, 0, x_3, 0, ..., 0)$-surface absolute difference $|u - \phi|$.



(a) Epoch vs. Overall Relative ℓ_2 Error in Ω (b) Epoch vs. Max Relative Modulus Error in Ω

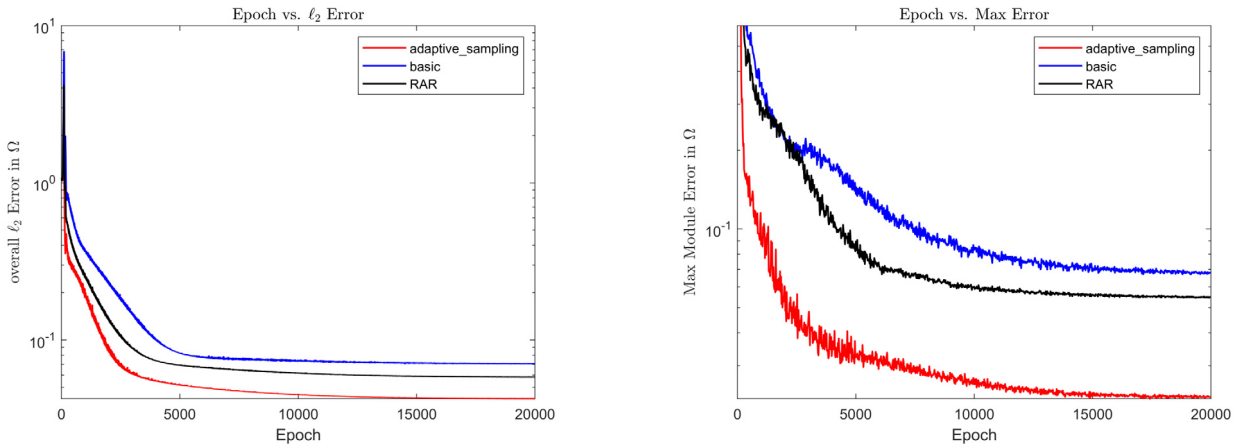
Fig. 4.9. Example 4.2 numerical results in 20 dimensions.

Table 6
Example 4.3 Result.

Dimension		AS	Basic	RAR	Error Reduced by AS
10	ℓ_2 error	4.233210e-02	7.093248e-02	5.827290e-02	40.32%
	Max Modulus Error	2.265792e-02	6.659816e-02	5.484901e-02'	65.98%
20	ℓ_2 error	7.342700e-02	1.299070e-01	9.876399e-02	43.48%
	Max Modulus Error	8.995471e-02	1.484266e-01	1.350940e-01	39.39%

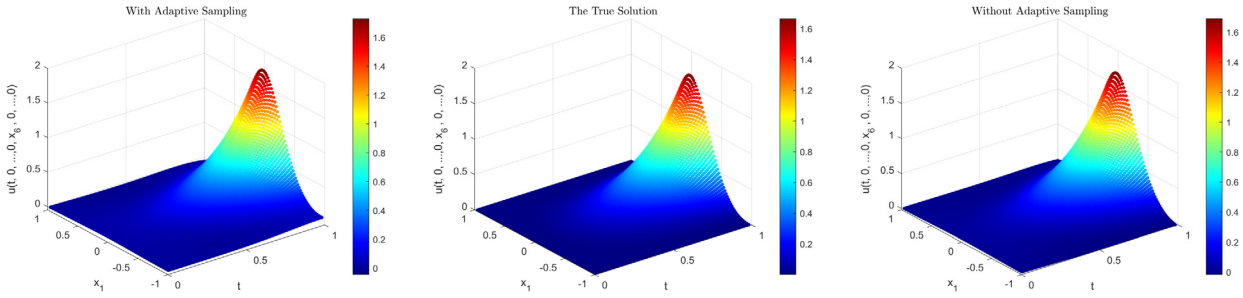
where $\omega := \{x : |x| < 1\}$, $\mathbb{T} = (0, 1)$. $g_0(x, t) = 0$, $h_0(x) = 0$, $h_1(x) = 0$, and $f(x, t)$ is given appropriately so that the exact solution is $u(x, t) = \left(\exp\left(t^2\right) - 1\right) \sin\left(\frac{\pi}{2}(1 - |x|)^{2.5}\right)$. Δ_x denotes the Laplace operator taken in the spatial variable x only.

Table 6 shows the overall relative l_2 error and maximum relative l_1 modulus error by adaptive sampling and the basic model at the end of 20000 epochs. The number of epochs versus ℓ_2 error decay in 10 dimensions and 20 dimensions are presented respectively in Fig. 4.10 and 4.13. Fig. 4.11 shows the (t, x_6)-surface of the ground-truth solution and network solutions with and without adaptive sampling, where axes are t, x_6, and $u(t, 0, ..., 0, x_6, 0, ..., 0)$. Fig. 4.12 shows the heat-map of absolute difference in (t, x_6)-surface, where the color bar represents the absolute difference between the ground-truth solution and network solution. This example shows that the adaptive sampling reduces the variance in the distribution of error over the domain. We can see from these results that adaptive sampling helps to reduce the error, especially relative l_1 maximum modulus error.



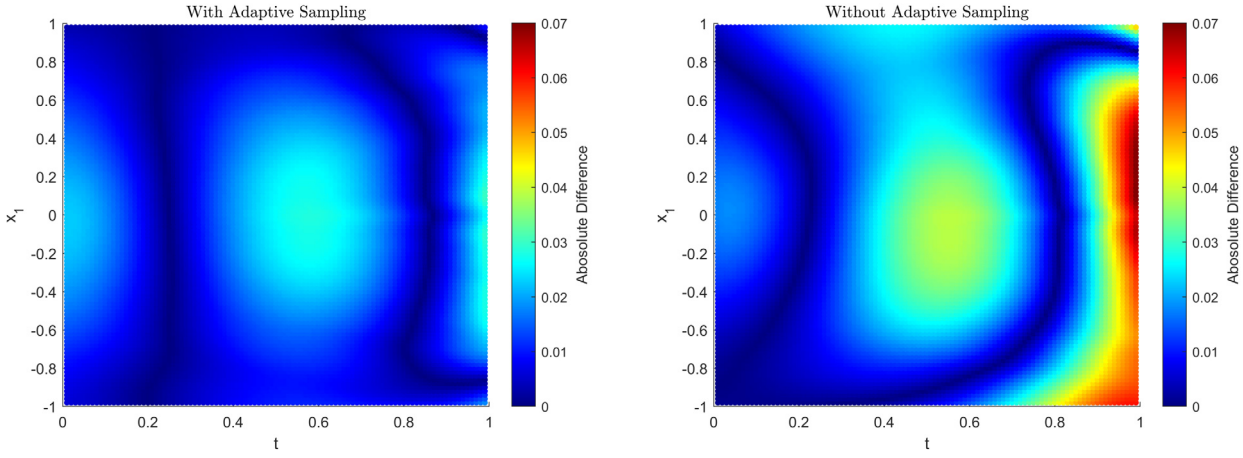
(a) Epoch vs. Overall Relative ℓ_2 Error in Ω (b) Epoch vs. Max Relative Modulus Error in Ω

Fig. 4.10. Example 4.3 numerical results in 10 dimensions.



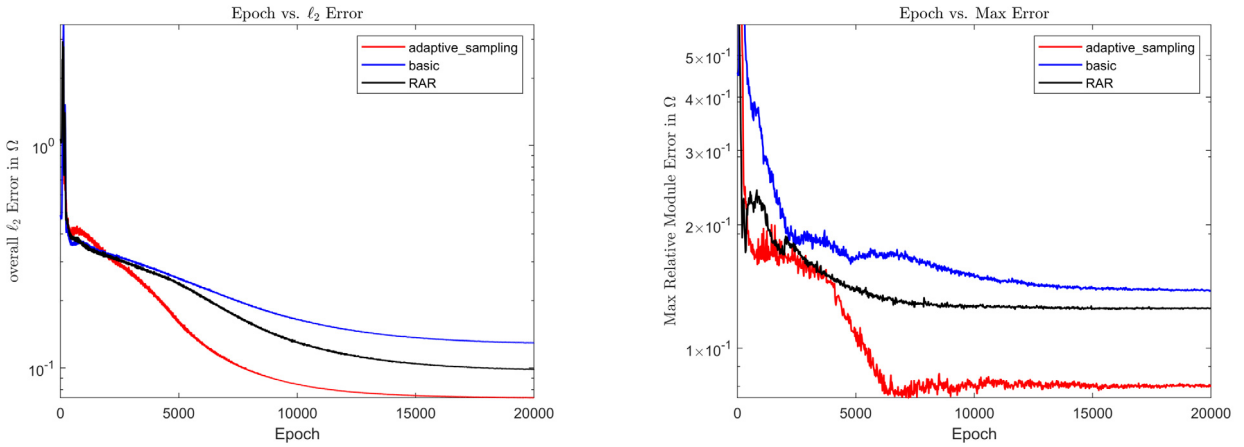
(a) With Adaptive Sampling (b) The True Solution (c) Without Adaptive Sampling

Fig. 4.11. Example 4.3 10 dimensions $(t, 0, ..., 0, x_6, 0, ..., 0)$-surface of network solutions and the true solution.



(a) With Adaptive Sampling (b) Without Adaptive Sampling

Fig. 4.12. Example 4.3 10 dimensions $(t, 0, ..., 0, x_6, 0, ..., 0)$-surface absolute difference $|u - \phi|$.



(a) Epoch vs. Overall Relative ℓ_2 Error in Ω (b) Epoch vs. Max Relative Modulus Error in Ω

Fig. 4.13. Example 4.3 numerical results in 20 dimensions.

4.4. Discussion on comparison with RAR

In all three PDE examples above, AS and RAR both can speed up the convergence and lower the error. In the above results, AS performs better than RAR in high-dimensional settings. Here we provide some intuition behind this out-performance. As we mentioned in Section 1.2.1, AS differentiates all training points and is more prone to select points with higher residual errors. On the other hand, RAR does not differentiate between training points; RAR replicates k points with the largest residuals. In AS, by ranking all training points, the neural network model focuses more on points that the model is more uncertain about. As we can see from the above results, even though AS does a good job in reducing the overall relative ℓ_2 error, AS is even more effective in reducing the max relative modulus error. This is because when the model is producing high residual errors on some volumes, AS will make the model highly focused on these volumes. In comparison, although RAR also focuses on points with high residuals, the effect might not be enough, especially in high-dimensional cases. As mentioned in Section 2.1, there are two main approaches to active learning: uncertainty sampling and diversity sampling. AS is an uncertainty sampling approach. Intuitively, RAR is more diversity-preserving than AS because it still keeps points that are randomly generated. However, AS also preserves diversity throughout the training process. AS will tackle points with higher residuals first, but after some iterations, the residual error of these points will be reduced, and other points that were previously not considered "high-residual" points now will be considered "high-residual" points by AS. Therefore, over the course of the whole training process, the AS algorithm will balance itself in selecting points that not only focuses on the uncertainty of the neural network model but also preserves diversity in some sense. In conclusion, AS outperforms RAR because AS focuses more on "high-residual error" areas/volumes as demonstrated in the massive reduction of max modulus error. AS also balances itself in selecting training points over the course of the whole training process so that AS will still preserve diversity to some extent.

4.5. Poisson's equation

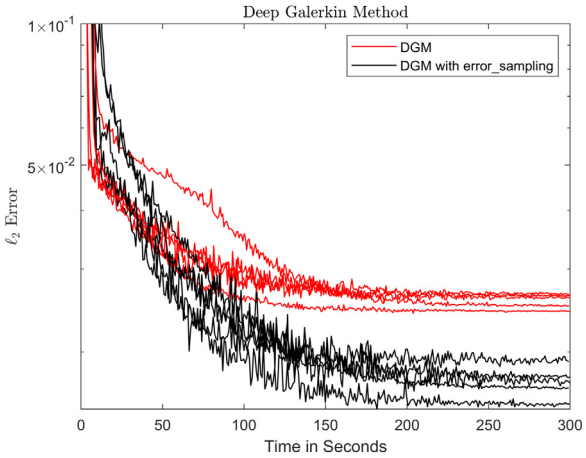
In this example, we will show that adaptive sampling is compatible with some recent frameworks: the Deep Ritz Method (DRM) [11], the Deep Galerkin Method (DGM) [57], and the Weak Adversarial Networks (WAN) [65]. We still use residual errors to define the sampling distribution and use Algorithm 2.3 to sample training points for all frameworks. To this end, we consider a 2D Poisson's equation: find $u(x, y)$ such that

$$\begin{aligned} -\Delta u(x, y) &= f(x, y), \qquad \text{in } \Omega := (0, 1) \times (0, 1), \\ u(x, y) &= g(x, y), \qquad \text{on } \partial\Omega. \end{aligned}$$

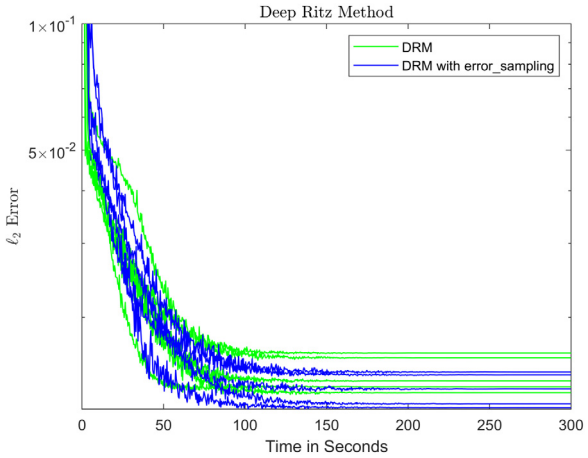
The exact solution is given by $u(x, y) = \min\{x^2, (1 - x)^2\}$. All models are trained for 300 seconds in NVIDIA Quadro P1000 Graphics Card. Each framework without and with adaptive sampling both have 30 trials to approximate the PDE. The error obtained with adaptive sampling is compared with the error obtained without adaptive sampling to test if adaptive sampling is compatible with various frameworks to lower the error. Note that, in different trials, different random seeds are used. Moreover, in the k-th trial, the same torch random seed is used with and without adaptive sampling, which implies that the networks are initialized with the same parameters. In this test, ResNet [17] with 4 residual blocks is used with Tanh as activation functions. Each epoch has 1024 uniform points on the boundary and in the domain respectively for training. Moreover, instead of numerical differentiation, Pytorch Autograd is utilized to evaluate derivatives.

Table 7
Compatibility tests; * denotes training equipped with adaptive sampling.

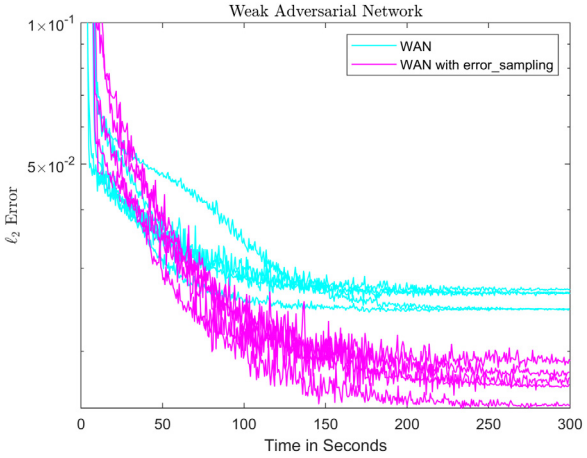
Framework	Mean	Standard Deviation	Minimum Value	Coefficient of Variation
DGM	0.0333	0.0064	0.0244	19.2%
DGM*	0.0280	0.0079	0.0155	29.2%
DRM	0.0273	0.0097	0.0132	35.5%
DRM*	0.0255	0.0093	0.0122	36.5%
WAN	0.0329	0.0062	0.0245	18.8%
WAN*	0.0282	0.0075	0.0153	26.6%



(a) DGM: ℓ_2 Error vs Time



(b)DRM: ℓ_2 Error vs Time



(c)WAN: ℓ_2 Error vs Time

Fig. 4.14. Compatibility Test: Five Lowest ℓ_2 Error Trials in Each Framework.

Table 7 shows statistics of ℓ_2 errors obtained in 30 trials without and with adaptive sampling. Fig. 4.14 shows the comparison of the five lowest ℓ_2 error curves obtained with and without adaptive sampling in 30 trials versus time in seconds for DGM, DRM, and WAN respectively. These results clearly show that adaptive sampling is compatible with all three frameworks. The errors obtained with adaptive sampling are lower than those without adaptive sampling. Note that, as observed in previous high-dimensional examples, the advantage of adaptive sampling becomes more significant when the dimension is high.

CRediT authorship contribution statement

Wenhan Gao: implemented the numerical test and wrote the manuscript draft.

Chunmei Wang: proposed the idea, supervised Wenhan, revised the draft to be submitted.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Chunmei Wang reports financial support was provided by National Science Foundation. Wenhan Gao reports financial support was provided by National Science Foundation. Chunmei Wang reports a relationship with National Science Foundation that includes: funding grants.

Data availability

Data will be made available on request.

Acknowledgement

W.G. was partially supported by the National Science Foundation under grant DMS-2050133. C.W. was partially supported by the National Science Foundation under awards DMS-2136380 and DMS-2206332. The authors would like to express our deepest appreciation to Dr. Haizhao Yang from University of Maryland College Park for his invaluable insights, unparalleled support, and constructive advice. Thanks should also go to Dr. Yiqi Gu from University of Hong Kong for helpful discussions.

Appendix A. RAR sampling

Algorithm A.1: RAR sampling to choose 12000 training points.

Result: $N = 12000$ points for training
Require: PDE (1.1); the current solution net $\phi(\boldsymbol{\theta})$
1 Generate 10000 uniformly distributed points $\{\boldsymbol{x}_i\}_{i=1}^{10000} \subset \Omega$; denote by $\boldsymbol{X}$;
2 Residual_Error_array = $\mathcal{R}_{abs}^{p}(\boldsymbol{X}) = |\mathcal{D}\phi(\boldsymbol{X}) - f(\boldsymbol{X})|^p$;
3 Add 2000 points in $\boldsymbol{X}$ with the largest residual errors to $\boldsymbol{X}$;
4 **return** $\boldsymbol{X}$ for training ;

The same logic follows for sampling on $\partial\Omega$.

Appendix B. Preliminary definition

Definition B.1. Let $\boldsymbol{x}$ be a real random variable and $w(\boldsymbol{x})$ be any real function. The expectation of $w(\boldsymbol{x})$ is defined by

$$\mu = \mathbb{E}_{\boldsymbol{x}\in\Omega}\left[w(\boldsymbol{x})\right] = \int_{\Omega} w(\boldsymbol{x})\, p(\boldsymbol{x})\, d\boldsymbol{x},$$

where $p(\boldsymbol{x})$ is the probability density function of $\boldsymbol{x}$.

Definition B.2. Let $\{\boldsymbol{x_i}\}_{i=1}^{n} \in \Omega \subset \mathbb{R}^d$ be real random variables that follow the distribution $p(\boldsymbol{x})$ and $w(\boldsymbol{x})$ be any function defined on Ω with density $p(\boldsymbol{x})$. The plain Monte-Carlo estimate of μ, the expectation of $w(\boldsymbol{x})$, is defined by

$$\hat{\mu} = \frac{|\Omega|}{n}\sum_{i=1}^{n} w\left(\boldsymbol{x}_i\right) p\left(\boldsymbol{x}_i\right), \boldsymbol{x_i} \sim p(\boldsymbol{x}),$$

where $|\Omega| = \int_{\Omega} d\boldsymbol{x}$ denotes the volume (length in 1D and area in 2D) of Ω, n is the number of points sampled for estimating μ. Note that $\hat{\mu}$ is an unbiased estimate of μ.

Appendix C. Numerical differentiation

Definition C.1. A real scalar function $\phi(\boldsymbol{x})$ of d variables is a rule that assigns a number $\phi(\boldsymbol{x}) \in \mathbb{R}$ to an array of numbers $\boldsymbol{x} = (x_1, \cdots, x_d) \in \mathbb{R}^d$.

Definition C.2. Let $\phi(\boldsymbol{x})$ be a real scalar function of d variables defined on $\mathbb{R}^d$. The **partial derivative** of $\phi(\boldsymbol{x})$ at a point $a \in \mathbb{R}^d$ with respect to $x_i (i = 1, \cdots, d)$ is given by

$$\frac{\partial \phi}{\partial x_i}(a) = \lim_{h_i \to 0} \frac{\phi(a + h\boldsymbol{e_i}) - \phi(a)}{h},$$

where $h \in \mathbb{R}$, $\boldsymbol{e_i} \in \mathbb{R}^d$ is an array of all zeros except 1 for the i-th element.

Definition C.3. Let $\phi(\boldsymbol{x})$ be a real scalar function of d variables defined on $\mathbb{R}^d$, $a \in \mathbb{R}^d$, and $h \in \mathbb{R}$. The **numerical differentiation** estimate of $\frac{\partial \phi}{\partial x_i}(a)$ with respect to $x_i (i = 1, \cdots, d)$ is given by:

$$\frac{\partial \phi}{\partial x_i}(a) \approx \frac{\phi(a + h\boldsymbol{e_i}) - \phi(a)}{h}.$$

Note that the truncation error is of order $O(h)$.

Appendix D. Self-normalized sampling

Let $q(\boldsymbol{x})$ be the desired distribution of a continuous variable $\boldsymbol{x}$ over the domain Ω. Suppose that we cannot compute $q(\boldsymbol{x})$ directly but have the unnormalized version $f(\boldsymbol{x})$ such that

$$q(\boldsymbol{x}) = \frac{f(\boldsymbol{x})}{NC}$$

where $NC = \int_\Omega f(\boldsymbol{x}) d\boldsymbol{x}$ is the unknown normalizing constant. The self-normalized sampling algorithm is: In the first step,

Algorithm D.2: Self-normalized Sampling.

Result: k points approximately follow the target distribution $q(\boldsymbol{x})$
Require: The unnormalized distribution $f(\boldsymbol{x})$
1 Generate an ordered list of n uniformly distributed points $\{\boldsymbol{x}_i\}_{i=1}^n \subset \Omega$; denote by $\boldsymbol{X}$. ;
2 $f(\boldsymbol{X})$ denotes the list $\{f(\boldsymbol{x}_i)\}_{i=1}^n \subset \Omega$;
3 constant $NC = \sum_{i=1}^n f(\boldsymbol{x}_i)$;
4 A new discrete probability mass function over $\boldsymbol{X}$ given by $p(\boldsymbol{x}_i) = \frac{f(\boldsymbol{x}_i)}{NC}$;
5 Generate k points following the discrete p.m.f. $p(\boldsymbol{x})$;
6 **return** k points generated ;

n should be large even if k is small, and when k is large, n can be equal to or even smaller than k as long as n is still large; in the numerical experiment in this work, 12,000 allocation points are needed, so n and k are both set to be 12,000.

We will show that this algorithm makes sense. Let χ be any subset of Ω.

The probability of $\boldsymbol{x} \in \chi$ in the sense of our target distribution $q(\boldsymbol{x})$ is:

$$P(\boldsymbol{x} \in \chi) = \int_\chi q(\boldsymbol{x}) d\boldsymbol{x} = \int_\chi \frac{f(\boldsymbol{x})}{NC} d\boldsymbol{x} = \frac{\int_\chi f(\boldsymbol{x}) d\boldsymbol{x}}{\int_\Omega f(\boldsymbol{x}) d\boldsymbol{x}} \approx \frac{\int_\chi d\boldsymbol{x} \cdot \frac{1}{m} \sum_{j=1}^m f(\boldsymbol{x}_j)}{\int_\Omega d\boldsymbol{x} \cdot \frac{1}{n} \sum_{i=1}^n f(\boldsymbol{x}_i)} = \frac{n}{m} \cdot \frac{\int_\chi d\boldsymbol{x}}{\int_\Omega d\boldsymbol{x}} \cdot \frac{\sum_{j=1}^m f(\boldsymbol{x}_j)}{\sum_{i=1}^n f(\boldsymbol{x}_i)}. \tag{D.1}$$

Note that, this is the Monte Carlo integration, $\boldsymbol{x}_j$ and $\boldsymbol{x}_i$ are m and n random points in χ and Ω respectively. This approximation is governed by the Law of Large Numbers; therefore, m and n are expected to be large.

Now, suppose out of n points generated in Algorithm D.2, there are m points in χ. When m and n are large, by the Law of Large Numbers, $\frac{m}{n} \approx \frac{\int_\chi d\boldsymbol{x}}{\int_\Omega d\boldsymbol{x}}$. Hence, we can apply these n and m as random points to Equation (D.1), the above probability becomes:

$$P(\boldsymbol{x} \in \chi) \approx \frac{\sum_{j=1}^m f(\boldsymbol{x}_j)}{\sum_{i=1}^n f(\boldsymbol{x}_i)}, \tag{D.2}$$

and this is the probability of $\boldsymbol{x} \in \chi$ in the sense of the discrete p.m.f. $p(\boldsymbol{x})$ in Algorithm D.2.

One advantage of this algorithm is that it generates points in a completely parallel fashion whereas in the Metropolis-Hastings algorithm, each draw is based on the previous draw; thus, the MH algorithm cannot be parallelized.

References

[1] Christophe Andrieu, Nando de Freitas, Arnaud Doucet, Michael I. Jordan, An introduction to MCMC for machine learning, Mach. Learn. (ISSN 0885-6125) 50 (1) (2003) 5–43.
[2] A.R. Barron, Universal approximation bounds for superpositions of a sigmoidal function, IEEE Trans. Inf. Theory (ISSN 0018-9448) 39 (3) (May 1993) 930–945, https://doi.org/10.1109/18.256500.
[3] Julius Berner, Philipp Grohs, Arnulf Jentzen, Analysis of the generalization error: empirical risk minimization over deep artificial neural networks overcomes the curse of dimensionality in the numerical approximation of Black–Scholes partial differential equations, SIAM J. Math. Data Sci. 2 (3) (2020) 631–657, https://doi.org/10.1137/19M125649X.
[4] Giuseppe Carleo, Matthias Troyer, Solving the quantum many-body problem with artificial neural networks, Science (ISSN 1095-9203) 355 (6325) (Feb 2017) 602–606.
[5] Fan Chen, Jianguo Huang, Chunmei Wang, Haizhao Yang, Friedrichs learning: weak solutions of partial differential equations via deep learning, arXiv: 2012.08023, 2021.
[6] Ziang Chen, Jianfeng Lu, Yulong Lu, On the representation of solutions to elliptic PDEs in Barron spaces, arXiv:2106.07539, 2021.

[7] Siddhartha Chib, Edward Greenberg, Understanding the Metropolis-Hastings algorithm, Am. Stat. (ISSN 0003-1305) 49 (4) (1995) 327–335.
[8] David A. Cohn, Zoubin Ghahramani, Michael I. Jordan, Active learning with statistical models, J. Artif. Intell. Res. (ISSN 1076-9757) 4 (1) (March 1996) 129–145.
[9] P.A.M. Dirac, The Principles of Quantum Mechanics, Clarendon Press, Oxford, 1981.
[10] E. Weinan, Stephan Wojtowytsch, Representation formulas and pointwise properties for Barron functions, arXiv e-prints, arXiv:2006.05982, June 2020.
[11] E. Weinan, Bing Yu, The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems, Commun. Math. Stat. (ISSN 2194-6701) 6 (1) (2018).
[12] E. Weinan, Jiequn Han, Arnulf Jentzen, Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations, Commun. Math. Stat. (ISSN 2194-671X) 5 (4) (Nov 2017) 349–380.
[13] E. Weinan, Chao Ma, Lei Wu, The Barron space and the flow-induced function spaces for neural network models, in: Constructive Approximation, 2021.
[14] Yarin Gal, Riashat Islam, Zoubin Ghahramani, Deep Bayesian Active Learning with Image Data, 2017.
[15] Yiqi Gu, Haizhao Yang, Chao Zhou, Selectnet: Self-paced learning for high-dimensional partial differential equations, J. Comput. Phys. (ISSN 0021-9991) 441 (2021) 110444.
[16] Jiequn Han, Arnulf Jentzen, E. Weinan, Solving high-dimensional partial differential equations using deep learning, Proc. Natl. Acad. Sci. (ISSN 0027-8424) 115 (34) (2018) 8505–8510, https://doi.org/10.1073/pnas.1718942115, https://www.pnas.org/content/115/34/8505.
[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Deep Residual Learning for Image Recognition, 2015.
[18] Sean Hon, Haizhao Yang, Simultaneous neural network approximations in Sobolev spaces, arXiv:2109.00161, 2021.
[19] Jianguo Huang, Haoqin Wang, Haizhao Yang, Int-Deep: a deep learning initialized iterative method for nonlinear problems, J. Comput. Phys. (ISSN 0021-9991) 419 (2020) 109675.
[20] Martin Hutzenthaler, Arnulf Jentzen, Thomas Kruse, Tuan Anh Nguyen, Philippe von Wurstemberger, Overcoming the curse of dimensionality in the numerical approximation of semilinear parabolic partial differential equations, Proc. R. Soc. A, Math. Phys. Eng. Sci. 476 (2244) (2020) 20190630, https://doi.org/10.1098/rspa.2019.0630, https://royalsocietypublishing.org/doi/abs/10.1098/rspa.2019.0630.
[21] Martin Hutzenthaler, Arnulf Jentzen, Thomas Kruse, Overcoming the curse of dimensionality in the numerical approximation of parabolic partial differential equations with gradient-dependent nonlinearities, in: Foundations of Computational Mathematics, 2021.
[22] Arnulf Jentzen, Diyora Salimova, Timo Welti, A Proof That Deep Artificial Neural Networks Overcome the Curse of Dimensionality in the Numerical Approximation of Kolmogorov Partial Differential Equations with Constant Diffusion and Nonlinear Drift Coefficients, 2019.
[23] Yuehaw Khoo, Lexing Ying, SwitchNet: a Neural Network Model for Forward and Inverse Scattering Problems, 2018.
[24] Yuehaw Khoo, Jianfeng Lu, Lexing Ying, Solving for High Dimensional Committor Functions Using Artificial Neural Networks, 2018.
[25] Diederik P. Kingma, Jimmy Ba, Adam: A Method for Stochastic Optimization, 2017.
[26] Colby L. Wight, Jia Zhao, Solving Allen-Cahn and Cahn-Hilliard equations using the adaptive physics informed neural networks, Commun. Comput. Phys. (ISSN 1991-7120) 29 (3) (2021) 930–954.
[27] I.E. Lagaris, A. Likas, D.I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Netw. (ISSN 1045-9227) 9 (5) (1998) 987–1000, https://doi.org/10.1109/72.712178.
[28] I.E. Lagaris, A.C. Likas, D.G. Papageorgiou, Neural-network methods for boundary value problems with irregular boundaries, IEEE Trans. Neural Netw. 11 (5) (2000) 1041–1049, https://doi.org/10.1109/72.870037.
[29] Hyuk Lee, In Seok Kang, Neural algorithm for solving differential equations, J. Comput. Phys. (ISSN 0021-9991) 91 (1) (1990) 110–131.
[30] David D. Lewis, William A. Gale, A sequential algorithm for training text classifiers, in: Bruce W. Croft, C.J. van Rijsbergen (Eds.), SIGIR '94, London, Springer, London, 1994, pp. 3–12.
[31] Juan Li, Qingmeng Wei, Optimal control problems of fully coupled fbsdes and viscosity solutions of Hamilton-Jacobi-Bellman equations, SIAM J. Control Optim. 52 (3) (2014) 1622–1662.
[32] Yunru Liu, Tingran Gao, Haizhao Yang, Selectnet: Learning to Sample from the Wild for Imbalanced Data Training, 2019.
[33] Jianfeng Lu, Yulong Lu, Min Wang, A priori generalization analysis of the deep Ritz method for solving high dimensional elliptic equations, arXiv: 2101.01708, 2021.
[34] Jianfeng Lu, Zuowei Shen, Haizhao Yang, Shijun Zhang, Deep network approximation for smooth functions, SIAM J. Math. Anal. 53 (5) (2021) 5465–5506, https://doi.org/10.1137/20M134695X.
[35] Lu Lu, Xuhui Meng, Zhiping Mao, George Em Karniadakis, DeepXDE: a deep learning library for solving differential equations, SIAM Rev. 63 (1) (2021) 208–228, https://doi.org/10.1137/19M1274067.
[36] Tao Luo, Haizhao Yang, Two-layer neural networks for partial differential equations: optimization and generalization theory, arXiv e-prints, arXiv: 2006.15733, 2020.
[37] A. Malek, R. Shekari Beidokhti, Numerical solution for high order differential equations using a hybrid neural network—optimization method, Appl. Math. Comput. (ISSN 0096-3003) 183 (1) (2006) 260–271.
[38] Prem Melville, Raymond J. Mooney, Diverse ensembles for active learning, in: Proceedings of 21st International Conference on Machine Learning (ICML-2004), Banff, Canada, July 2004, pp. 584–591.
[39] Hadrien Montanelli, Haizhao Yang, Qiang Du, Deep ReLU networks overcome the curse of dimensionality for generalized bandlimited functions, J. Comput. Math. (ISSN 1991-7139) 39 (6) (2021) 801–815, https://doi.org/10.4208/jcm.2007-m2019-0239, http://global-sci.org/intro/article_detail/jcm/19912.html.
[40] Art B. Owen, Monte Carlo Theory, Methods and Examples, 2013.
[41] Remus Pop, Patric Fulop, Deep Ensemble Bayesian Active Learning: Addressing the Mode Collapse Issue in Monte Carlo Dropout via Ensembles, 2018.
[42] R. Andrew, D.F. Griffiths, The Finite Difference Method in Partial Differential Equations, Wiley, Chichester, England, 1980.
[43] Maziar Raissi, Paris Perdikaris, George Em Karniadakis, Physics informed deep learning (part I): data-driven solutions of nonlinear partial differential equations, arXiv preprint arXiv:1711.10561, 2017.
[44] Maziar Raissi, Paris Perdikaris, George Em Karniadakis, Physics informed deep learning (part II): data-driven discovery of nonlinear partial differential equations, arXiv preprint arXiv:1711.10566, 2017.
[45] Christoph Reisinger, Gabriel Wittum, Efficient hierarchical approximation of high-dimensional option pricing problems, SIAM J. Sci. Comput. 29 (2007) 440, https://doi.org/10.1137/060649616.
[46] Ozan Sener, Silvio Savarese, Active Learning for Convolutional Neural Networks: A Core-Set Approach, 2018.
[47] Burr Settles, Active learning literature survey, Computer Sciences Technical Report 1648, 2010.
[48] Jingyu Shao, Qing Wang, Fangbing Liu, Learning to sample: an active learning framework, CoRR, arXiv:1909.03585 [abs], 2019.
[49] Zuowei Shen, Haizhao Yang, Shijun Zhang, Deep network approximation characterized by number of neurons, Commun. Comput. Phys. (ISSN 1991-7120) 28 (5) (2020) 1768–1811, https://doi.org/10.4208/cicp.OA-2020-0149.
[50] Zuowei Shen, Haizhao Yang, Shijun Zhang, Deep network with approximation error being reciprocal of width to power of square root of depth, Neural Comput. 33 (4) (03 2021) 1005–1036, https://doi.org/10.1162/neco_a_01364.
[51] Zuowei Shen, Haizhao Yang, Shijun Zhang, Neural network approximation: three hidden layers are enough, Neural Netw. (ISSN 0893-6080) 141 (2021) 160–173, https://doi.org/10.1016/j.neunet.2021.04.011.

[52] Zuowei Shen, Haizhao Yang, Shijun Zhang, Deep network approximation: achieving arbitrary accuracy with fixed number of neurons, arXiv:2107.02397, 2021.
[53] Zuowei Shen, Haizhao Yang, Shijun Zhang, Optimal approximation rate of ReLU networks in terms of width and depth, J. Math. Pures Appl. 157 (January 2022) 101–135, https://doi.org/10.1016/j.matpur.2021.07.009.
[54] Jonathan W. Siegel, Jinchao Xu, Approximation rates for neural networks with general activation functions, Neural Netw. (ISSN 0893-6080) 128 (2020) 313–321, https://doi.org/10.1016/j.neunet.2020.05.019, https://www.sciencedirect.com/science/article/pii/S0893608020301891.
[55] Jonathan W. Siegel, Jinchao Xu, Sharp bounds on the approximation rates, metric entropy, and n-widths of shallow neural networks, arXiv:2101.12365, 2021.
[56] Justin Sirignano, Konstantinos Spiliopoulos, DGM: a deep learning algorithm for solving partial differential equations, J. Comput. Phys. (ISSN 0021-9991) 375 (2018) 1339–1364.
[57] Justin Sirignano, Jonathan F. MacArt, Jonathan B. Freund, DPM: a deep learning PDE augmentation method with application to large-eddy simulation, J. Comput. Phys. (ISSN 0021-9991) 423 (2020) 109811.
[58] Wei Tang, Tao Shan, Xunwang Dang, Maokun Li, Fan Yang, Shenheng Xu, Ji Wu, Study on a Poisson's equation solver based on deep learning technique, in: 2017 IEEE Electrical Design of Advanced Packaging and Systems Symposium (EDAPS), 2017, pp. 1–3.
[59] Ramakrishna Tipireddy, David A. Barajas-Solano, Alexandre M. Tartakovsky, Conditional Karhunen-Loève expansion for uncertainty quantification and active learning in partial differential equation models, J. Comput. Phys. (ISSN 0021-9991) 418 (Oct 2020) 109604.
[60] Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, Ken Perlin, Accelerating Eulerian fluid simulation with convolutional networks, https://arxiv.org/abs/1607.03597, 2016.
[61] Dmitry Yarotsky, Optimal approximation of continuous functions by very deep ReLU networks, in: Sébastien Bubeck, Vianney Perchet, Philippe Rigollet (Eds.), in: Proceedings of the 31st Conference on Learning Theory, in: Proceedings of Machine Learning Research, vol. 75, 06–09 Jul 2018, PMLR, pp. 639–649, http://proceedings.mlr.press/v75/yarotsky18a.html.
[62] Yarotsky Dmitry, Elementary superexpressive activations, arXiv e-prints, arXiv:2102.10911, February 2021.
[63] Dmitry Yarotsky, Anton Zhevnerchuk, The phase diagram of approximation rates for deep neural networks, in: H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems, vol. 33, Curran Associates, Inc., 2020, pp. 13005–13015, https://proceedings.neurips.cc/paper/2020/file/979a3f14bae523dc5101c52120c535e9-Paper.pdf.
[64] Donggeun Yoo, In So Kweon, Learning Loss for Active Learning, 2019.
[65] Yaohua Zang, Gang Bao, Xiaojing Ye, Haomin Zhou, Weak adversarial networks for high-dimensional partial differential equations, J. Comput. Phys. (ISSN 0021-9991) 411 (Jun 2020) 109409.
[66] Yue Zhao, Ciwen Xu, Yongcun Cao, Research on query-by-committee method of active learning and application, in: Xue Li, Osmar R. Zaïane, Zhanhuai Li (Eds.), Advanced Data Mining and Applications, Springer Berlin Heidelberg, Berlin, Heidelberg, ISBN 978-3-540-37026-0, 2006, pp. 985–991.