Journal of Computational Physics 505 (2024) 112911

Contents lists available at ScienceDirect

Journal of Computational Physics

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

Efficient and stable SAV-based methods for gradient flows arising from deep learning

Ziqi Ma [a], Zhiping Mao [a,*], Jie Shen [b,*]

[a] School of Mathematical Sciences, Fujian Provincial Key Laboratory of Mathematical Modeling and High-Performance Scientific Computing, Xiamen University, Xiamen, 361005, China

[b] School of Mathematical Science, Eastern Institute of Technology, Ningbo, Zhejiang, 315200, China

ARTICLE INFO

Keywords:
Deep learning
Gradient flow
SAV method
Energy stable
Adam method
Adaptive learning rate

ABSTRACT

The optimization algorithm plays an important role in deep learning and significantly affects the stability and efficiency of the training process, and consequently the accuracy of the neural network approximation. A suitable (initial) learning rate is crucial for the optimization algorithm in deep learning. However, a small learning rate is usually needed to guarantee the convergence of the optimization algorithm, resulting in a slow training process. We develop in this work efficient and energy stable optimization methods for function approximation problems as well as solving partial differential equations using deep learning. In particular, we consider the gradient flows arising from deep learning from the continuous point of view, and employ the particle method (PM) and the smoothed particle method (SPM) for the space discretization while we adopt SAV-based schemes, combined with the adaptive strategy used in Adam algorithm, for the time discretization. To illustrate the effectiveness of the proposed methods, we present a number of numerical tests to demonstrate that the SAV-based schemes significantly improve the efficiency and stability of the training as well as the accuracy of the neural network approximation. We also show that the SPM approach gives a slightly better accuracy than the PM approach. We further demonstrate the advantage of using adaptive learning rate in dealing with more complex problems.

1. Introduction

Neural network-based machine learning, which is also called deep learning, has been widely used in classical artificial intelligence, for instance, object detection and face recognition using convolutional neural networks [1,2], handwriting learning and natural language processing using recurrent neural networks [3], generating examples for the image dataset and unknown photographs of human faces using generative adversarial networks [4]. In the last decade, it has also been shown that deep learning is a powerful tool in scientific computing. For examples, a PINN framework that can easy handle inverse problems and complex geometries is proposed in [5], DeepONet, which uses neural network to approximate differential operators, is proposed in [6] for solving a class of similar problems, a mesh-free computational framework for solving elliptic PDEs on unknown manifolds based on deep learning theory is proposed in [7], and deep learning has also been used to solve many complex high-dimensional nonlinear PDEs [8,9]. We refer to [10–12] and references therein for more efforts in these directions.

* Corresponding authors.
E-mail addresses: maziqi@stu.xmu.edu.cn (Z. Ma), zpmao@xmu.edu.cn (Z. Mao), jshen@eitech.edu.cn (J. Shen).

https://doi.org/10.1016/j.jcp.2024.112911
Received 2 May 2023; Received in revised form 1 March 2024; Accepted 2 March 2024

Available online 7 March 2024

A K-hidden layer neural network is defined as

$$f(\mathbf{x},\theta) = f_K(...f_2(f_1(\mathbf{x}))),$$

where θ denotes all parameters to be trained, and for the k-th hidden layer

$$f_k(\mathbf{x}) = \frac{1}{m_k}\sum_{i=1}^{m_k} a_{i,k}\sigma(\boldsymbol{\omega}_{i,k}^T\mathbf{x} + b_{i,k}),\ k = 1,2,\dots,K.$$

Here m_k is the number of neurons in the k-th hidden layer, $\sigma(\cdot)$ is a nonlinear activation function, $\theta_{i,k} = (a_{i,k}, b_{i,k}, \boldsymbol{\omega}_{i,k})$, where $a_{i,k}$, $b_{i,k} \in \mathbb{R}$, $\boldsymbol{\omega}_{i,k} \in \mathbb{R}^D$ are the parameters to be trained. We shall drop the subscript k if no confusion arises. The idea of deep learning is to find the best approximation of a target functional using a neural network as the surrogate function. To this end, we need to define the so-called loss function (energy functional) to formulate an optimization problem. For example, in the supervised learning with a given target function f^*, the loss function, also called the population risk, in the L^2 sense is given by

$$\mathcal{I}(\theta) = \int_{\mathcal{R}^D} (f(\mathbf{x},\theta) - f^*(\mathbf{x}))^2\mu(d\mathbf{x}), \tag{1}$$

where μ is a probability distribution; or with a given set of data $(\mathbf{x}_i, y_i) \in \mathbb{R}^D \times \mathbb{R}, i = 1,2,\dots,n$, the loss function, also called the empirical risk, is given by

$$\mathcal{I}(\theta) = \frac{1}{n}\sum_{i=1}^{n} (f(\mathbf{x}_i,\theta) - y_i)^2. \tag{2}$$

A major task in deep learning is to develop efficient optimization algorithms for the training of neural networks. This is also a bottleneck that significantly affects the efficiency and accuracy of the neural network approximation. Popular optimization algorithms used in deep learning are mostly gradient descent-based methods. The gradient descent (GD) method, proposed in [13], takes the form:

$$\theta^{t+1} = \theta^t - \Delta t\nabla_\theta\mathcal{I}(\theta^t),$$

and was first described as an optimization algorithm for deep learning in [14], where Δt denote the learning rate lr, $\nabla_\theta\mathcal{I}(\theta^t)$ is the gradient of the loss function w.r.t. θ at the time step t. However, due to the fact that the optimization problems arising from deep learning are usually nonconvex and high-dimensional, the original GD method often fails as it is more likely trapped in a local minimum. Moreover, it is computationally expensive due to the fact that the GD method uses the gradient calculated from all samples at each epoch. Therefore, the Stochastic Gradient Descent (SGD) method, which randomly chooses one or a batch of training samples from the training set, is proposed to resolve this issue [15]. In particular, the SGD [15] integrates the idea of mini-batch, and updates the parameters using a mini-batch of l training examples at each time step t by the following formula:

$$\theta^{t+1} = \theta^t - \Delta t\nabla_\theta\mathcal{I}(\theta^t; \mathbf{x}_{i:i+l}, y_{i:i+l}),$$

where the gradient is computed by using l data $(\mathbf{x}_{i:i+l}, y_{i:i+l})$ in the ith mini-batch.[1] However, the learning rate Δt for the SGD method is usually set to be a constant, so it does not converge if the learning rate is too large while it converges very slowly if the learning rate is too small. Therefore, several adaptive optimization algorithms are proposed. For instance, Adagrad (Adaptive Gradient) adapts the learning rate by assigning higher learning rates to infrequent features, which ensures that the parameter updates rely less on frequency and more on relevance [16]. An extension of Adagrad is RMSprop, which avoids the learning rate going to zero [17]. It accumulates past squared gradients over window of a fixed size rather than all of them to improve the efficiency. One of the most popular adaptive optimization algorithms in deep learning is the so called Adam algorithm [18], which first computes the estimates of the first moment (the mean) and second moment (the uncentered variance) of the gradient:

$$m^t = \beta_1 m^{t-1} + (1-\beta_1)\nabla_\theta\mathcal{I}(\theta^t), \quad v^t = \beta_2 v^{t-1} + (1-\beta_2)\|\nabla_\theta\mathcal{I}(\theta^t)\|^2, \tag{3}$$

where $\beta_1 = 0.9, \beta_2 = 0.999$, and m^0, v^0 is initialized to 0, then updates parameters with the bias-corrected first and second moment estimates $\hat{m}^t, \hat{v}^t$:

$$\hat{m}^t = \frac{m^t}{1-(\beta_1)^t}, \quad \hat{v}^t = \frac{v^t}{1-(\beta_2)^t}, \quad \theta^{t+1} = \theta^t - \frac{\Delta t}{\sqrt{\hat{v}^t} + \epsilon}\hat{m}^t. \tag{4}$$

More detail can be found in [19] and references therein.

However, the performance, especially the stability and the efficiency, of the aforementioned optimization algorithms strongly depends on the choice of the (initial) learning rate, even for these adaptive-based methods. For instance, in practice, one needs to manually and carefully adjust the learning rate to obtain a more or less "good" learning process. On the one hand, the training may

[1] For the sake of simplicity, we ignore the dependence on $(\mathbf{x}_{i:i+l}, y_{i:i+l})$ in this paper.

fail if the learning rate is too big, and on the other hand, the training process is very slow if the learning rate is too small. Therefore, it is important to develop efficient and stable optimization algorithms for the training process in deep learning.

Recently, Liu et al. considered the optimization problem as finding the steady state solutions of a gradient flow from the continuous point of view and then proposed efficient and energy stable SAV-based algorithms [20]. We note that from the continuous point of view, the training process in deep learning can be viewed as a dynamic system, see [21–27] and references therein. For examples, it has been shown that the 2LNN model [28,29] and the ResNet [30,31] correspond to continuous dynamic systems under certain conditions. In addition, it is shown in [32] that the SGD dynamics for one hidden layer neural networks can be captured by a certain gradient flow governed by a nonlinear partial differential equation.

Motivated by this, we aim to adopt the efficient and stable optimization algorithms proposed in [20] for the training process in deep learning from the continuous point of view, namely, we would like to develop efficient and stable algorithms for the continuous nonlinear PDEs arising from deep learning. In particular, we shall consider the function approximation problem using the L^2 loss function with one hidden layer neural networks approach, and develop efficient SAV-based algorithms for the corresponding gradient flow. Specifically, we consider the empirical risk (2) with the following one hidden layer neural network

$$f(\mathbf{x}) = \frac{1}{m}\sum_{i=1}^{m} a_i \sigma(\boldsymbol{\omega}_i^T \mathbf{x}).$$

Here we use $\mathbf{x} \Rightarrow [\mathbf{x}; 1]$, $\boldsymbol{\omega}_i \Rightarrow [\boldsymbol{\omega}_i; b_i]$ to simplify the notations, m is the number of neurons. By using the mean field analysis [32,33], it follows that as $m \to \infty$, the one hidden layer neural network can be represented by the function in the expectation form from the continuous point of view:

$$f(\mathbf{x}) = \int a\sigma(\boldsymbol{\omega}^T \mathbf{x})\pi(da, d\boldsymbol{\omega}) = \mathbb{E}_{(a,\boldsymbol{\omega})\sim\pi}[a\sigma(\boldsymbol{\omega}^T \mathbf{x})], \tag{5}$$

where π is the probability distribution of $\theta = (a, \boldsymbol{\omega})$, and the gradient flow corresponding to the training process is given by

$$\frac{\partial \pi}{\partial t} = -\nabla \cdot (\pi \nabla v), \quad v = -\frac{\delta \mathcal{I}}{\delta \pi}, \tag{6}$$

where $\mathcal{I}$ is the loss functional given by (1). Therefore, the training process, i.e., the optimization in deep learning, becomes finding the steady solution of the above gradient flow.

The main purpose of this work is to develop stable and efficient numerical schemes to solve (6). In particular, we numerically solve the problem (6) using the particle method (PM) and the smoothed particle method (SPM) for the space discretization while using SAV-based schemes for the time discretization. We point out here that in the present work we test the SPM approach for the space discretization for (6) to see if SPM would give a better accuracy compared with the PM approach since SPM is a nonlocal approach. Our numerical results show that the SPM approach delivers a little better accuracy compared with the PM approach. We would like to develop better space discretizations such as spectral methods for (6) in future. Our main contributions are as follows:

- We formulate and implement the PM and SPM approaches, and show that SPM achieves approach higher accuracy than the PM approach.
- We adopt several efficient and energy stable SAV-based schemes, including the vanilla SAV, the restart SAV and the relaxed SAV schemes to improve the robustness and efficiency of the training processes. Note that the vanilla SAV and the relaxed SAV schemes are all unconditionally energy stable with a modified energy.
- By combining the SAV-based schemes with the Adam adaptive technique, we develop a SAV based adaptive algorithm which is more efficient and results in more reliable training processes, and also improves accuracy.

The rest of the paper is organized as follows. The numerical schemes for both space and time discretization of (6) are presented in Section 2. Then several numerical examples including regression and classification problems are shown to demonstrate the effectiveness of the present algorithms in Section 3. A summary with some discussions is given in Section 4.

2. Numerical methods

We first introduce PM and SPM approximations for the spactial discretization, and then present several efficient and energy stable SAV-based schemes for the time discretization.

2.1. Space discretization for the gradient flow

We introduce the PM and SPM methods for the high-dimensional nonlinear system (6). These two methods have been proposed in [26], but as fas as we know the SPM has not yet been implemented in the framework of deep learning.

2.1.1. Particle method

We approximate the continuous distribution π by

$$\hat{\pi}(\theta) = \frac{1}{m}\sum_{k=1}^{m} \delta(\theta - \theta_k), \tag{7}$$

where $\delta(\cdot)$ is the Dirac delta function, m is the number of particles, and $\{\theta_k\}_{k=1}^m$ are the m particles. Then (5) is approximated by

$$f(x;\hat{\pi}) = \frac{1}{m}\sum_{k=1}^{m} a_k \sigma(\boldsymbol{\omega}_k^T \mathbf{x}).$$

By testing (6) with a function $g(\theta)$, we obtain a weak formulation of (6):

$$\frac{d}{dt}\int g(\theta) d\pi(\theta) = -\int \nabla\cdot(\pi v(\pi,\theta)) g(\theta) d\theta = \int \langle v(\pi,\theta), \nabla g(\theta)\rangle \, d\pi(\theta).$$

Then applying the approximation (7) on both sides of the above equation and using the chain rule yields

$$\frac{d}{dt}\frac{1}{m}\sum_{k=1}^{m} g(\theta_k) = \frac{1}{m}\sum_{k=1}^{m}\left\langle \nabla g(\theta_k), \frac{d\theta_k}{dt}\right\rangle = \frac{1}{m}\sum_{k=1}^{m}\langle \nabla g(\theta_k), v(\hat{\pi},\theta_k)\rangle,$$

which implies

$$\frac{d\theta_k}{dt} = v(\hat{\pi},\theta_k) = -\frac{\delta \mathcal{I}(\theta_1,...,\theta_m)}{\delta\theta_k}, \tag{8}$$

where $\theta = (a,\boldsymbol{\omega})$, and $\mathcal{I}(\theta_1,...,\theta_m)$ is the approximation of the energy functional (1) by using (7). Consequently, we obtain

$$\begin{aligned} \frac{da_k}{dt} &= -\frac{\delta \mathcal{I}}{\delta a_k} = -\mathbb{E}_{\mathbf{x}}[(f(\mathbf{x};\hat{\pi}) - f^*(\mathbf{x}))\sigma(\boldsymbol{\omega}_k^T\mathbf{x})], \\ \frac{d\boldsymbol{\omega}_k}{dt} &= -\frac{\delta \mathcal{I}}{\delta \boldsymbol{\omega}_k} = -\mathbb{E}_{\mathbf{x}}[(f(\mathbf{x};\hat{\pi}) - f^*(\mathbf{x}))a_k\sigma'(\boldsymbol{\omega}_k^T\mathbf{x})\mathbf{x}]. \end{aligned} \tag{9}$$

The expectation with respect to $\mathbf{x}$ here corresponds exactly to the mini-batch in the SGD method. Therefore, for the implementation of PM, we can directly employ the GD (or SGD for the stochastic version) method.

2.1.2. Smoothed particle method

Unlike the PM approach, here we use the following smooth function to approximate the distribution π

$$\tilde{\pi}(\theta) = \frac{1}{m}\sum_{k=1}^{m}\phi_h(\theta-\theta_k), \tag{10}$$

where ϕ_h is a probability density function of $\mathcal{N}(0,h^2 I)$, $h\in(0,1)$ is a constant with a default value 0.0001. In this case, (5) can be approximated by

$$f(\mathbf{x};\tilde{\pi}) = \int a\sigma(\boldsymbol{\omega}^T\mathbf{x})\tilde{\pi}(da,d\boldsymbol{\omega}) = \mathbb{E}_{(a,\boldsymbol{\omega})\sim\tilde{\pi}}[a\sigma(\boldsymbol{\omega}^T\mathbf{x})].$$

Next we introduce a random variable $\xi\sim\mathcal{N}(0,I_{D+1})$, which is equivalent to $h\xi\sim\mathcal{N}(0,h^2 I_{D+1})$. Then the above expectation can be calculated by computing the expectation with respect to the variable ξ. Let us write ξ as $\xi=(\xi_1,\boldsymbol{\xi}_2)$, where $\xi_1\in\mathbb{R}$ is for the parameter a while $\boldsymbol{\xi}_2\in\mathbb{R}^D$ is for the parameter $\boldsymbol{\omega}$. Therefore, the above expectation can be written as

$$f(\mathbf{x};\tilde{\pi}) = \frac{1}{m}\sum_{k=1}^{m}\mathbb{E}_{(\xi_1,\boldsymbol{\xi}_2)}[(a_k+h\xi_1)\sigma((\boldsymbol{\omega}_k+h\boldsymbol{\xi}_2)^T\mathbf{x})].$$

By using a similar argument as that for the PM approach, we obtain the following system

$$\frac{d\theta_k}{dt} = v(\tilde{\pi},\theta_k) = -\frac{\delta \mathcal{I}(\theta_1,...,\theta_m;\xi)}{\delta\theta_k}, \tag{11}$$

where $\mathcal{I}(\theta_1,...,\theta_m;\xi)$ is the approximation of the energy functional (1) by using (10). This gives

$$\begin{aligned} \frac{da_k}{dt} &= -\mathbb{E}_{\xi_2}[\frac{\delta \mathcal{I}(\boldsymbol{\xi}_2)}{\delta a_k}] = -\mathbb{E}_{\boldsymbol{\xi}_2}[\mathbb{E}_{\mathbf{x}}[(f(\mathbf{x};\tilde{\pi}) - f^*(\mathbf{x}))\sigma(\boldsymbol{\omega}_k^T\mathbf{x} + h\boldsymbol{\xi}_2^T\mathbf{x})]], \\ \frac{d\boldsymbol{\omega}_k}{dt} &= -\mathbb{E}_{(\xi_1,\boldsymbol{\xi}_2)}[\frac{\delta \mathcal{I}(\xi_1,\boldsymbol{\xi}_2)}{\delta \boldsymbol{\omega}_k}] \\ &= -\mathbb{E}_{(\xi_1,\boldsymbol{\xi}_2)}[\mathbb{E}_{\mathbf{x}}[(f(\mathbf{x};\tilde{\pi}) - f^*(\mathbf{x}))(a_k+h\xi_1)\sigma'(\boldsymbol{\omega}_k^T\mathbf{x} + h\boldsymbol{\xi}_2^T\mathbf{x})\mathbf{x}]]. \end{aligned} \tag{12}$$

For the implementation, we employ the Monte Carlo method for the numerical integration to approximate the expectation with respect to ξ in the above equation. By generating a series of random variables $\{\xi^i\}_{i=1}^J\sim\mathcal{N}(0,I_{D+1})$, and using them to calculate the expectation J times (in this work, we take $J=10$), we use the obtained average to compute gradients and then update parameters. In summary, the SPM algorithm can be implemented as follows:

Algorithm 1 Smoothed Particle Method.

Input: θ^0
Output: θ^N
for $n = 0, 1, ..., N-1$ **do**
  Generate $\{\xi^j\}_{j=1}^J \sim \mathcal{N}(0, I_{D+1})$
  for $j = 1, 2, ..., J$ **do**
    Calculate $L_j = \mathcal{I}(\mathbf{x}, \theta^n, \xi^j)$
  end for
  $Loss = \frac{1}{J}\sum_{j=1}^J L_j$
  Use the Loss to renew θ^{n+1}
end for
return θ^N

2.2. Time discretization for the gradient flow

We now consider the time discretization for the ODE systems (8) and (11). We write systems (8) and (11) into the following unified form

$$\frac{d\theta}{dt} = -\frac{\delta \mathcal{I}(\theta)}{\delta \theta} := -\mathcal{N}(\theta), \tag{13}$$

where $\theta = [\theta_1, \dots, \theta_m]^T$. Due to the nonlinear activate functions in the neural network approximation, the resulting system is usually a complex nonlinear system. Taking the inner product of (13) with θ_t, we obtain the energy dissipation law

$$\frac{d}{dt}\mathcal{I}(\theta) = \left(\frac{\delta \mathcal{I}(\theta)}{\delta \theta}, \theta_t\right) = -\|\theta_t\|^2 \le 0,$$

where $(\cdot,\cdot)$ and $\|\cdot\|$ denote the standard discrete L^2 inner product and the corresponding norm, respectively. However, the commonly used GD method and its variants usually do not ensure energy diminishing, and may not converge if the time step (i.e., the learning rate) is too large. Thus, inspired by the SAV method for the gradient flow [34], and encouraged by its success for optimization problems [20], we shall employ several SAV-based algorithms which enjoy an energy dissipation law at the discrete level.

2.2.1. SAV-based algorithms

We first recall the vanilla SAV method. The SAV method is a popular numerical method in dealing with complex gradient flow systems proposed in [34] since it is easy to design efficient and energy stable schemes for gradient flows. In order to improve the performance (e.g., the efficiency and the stability) of the SAV method, we introduce a linear semi positive-definite operator $\mathcal{L}$ to the system (13):

$$\frac{d\theta}{dt} + \mathcal{L}(\theta) + \mathcal{N}(\theta) - \mathcal{L}(\theta) = 0. \tag{14}$$

For example, we can take $\mathcal{L}(\theta) = \lambda(-\Delta)^k\theta$, where λ, $k \ge 0$ are given constants. We then introduce a scalar auxiliary variable (SAV)

$$r(t) = \sqrt{\mathcal{I}(\theta) + C}, \tag{15}$$

where C is a constant to ensure that $\mathcal{I}(\theta) + C \ge 0$, and expand the system (14) into the following system:

$$\frac{d\theta}{dt} + \mathcal{L}(\theta) + \frac{r}{\sqrt{\mathcal{I}(\theta)+C}}\mathcal{N}(\theta) - \mathcal{L}(\theta) = 0, \tag{16a}$$

$$\frac{dr}{dt} = \frac{1}{2\sqrt{\mathcal{I}(\theta)+C}}\left(\mathcal{N}(\theta), \frac{d\theta}{dt}\right). \tag{16b}$$

With $r(0) = \sqrt{\mathcal{I}(\theta|_{t=0}) + C}$, it is obvious that the solution of (14) is also a solution of the above system. We then discretize the above system with the following first order scheme:

$$\frac{\theta^{n+1}-\theta^n}{\Delta t} + \mathcal{L}(\theta^{n+1}) + \frac{\mathcal{N}(\theta^n)}{\sqrt{\mathcal{I}(\theta^n)+C}}r^{n+1} - \mathcal{L}(\theta^n) = 0, \tag{17a}$$

$$\frac{r^{n+1}-r^n}{\Delta t} = \frac{1}{2\sqrt{\mathcal{I}(\theta^n)+C}}\left(\mathcal{N}(\theta^n), \frac{\theta^{n+1}-\theta^n}{\Delta t}\right). \tag{17b}$$

Taking the L^2 inner product for (17a) with $\theta^{n+1}-\theta^n$ and multiplying $2\Delta t r^{n+1}$ on both sides of (17b), summing up the results, we can easily prove the following:

Theorem 1. *The scheme (17) is unconditionally energy stable in the following sense with a modified energy: $\forall\ \Delta t > 0$, we have*

$$(r^{n+1})^2 - (r^n)^2 = -(\mathcal{L}(\theta^{n+1}-\theta^n), \theta^{n+1}-\theta^n) - (r^{n+1}-r^n)^2 - \frac{\|\theta^{n+1}-\theta^n\|^2}{\Delta t} \le 0. \tag{18}$$

The scheme (17) can be easily and efficiently implemented as follows: Letting

$$\theta^{n+1} = \theta^{n+1,1} + r^{n+1}\theta^{n+1,2}, \tag{19}$$

we have that (17a) is equivalent to the following two decoupled equations:

$$\frac{\theta^{n+1,1}}{\Delta t} + \mathcal{L}(\theta^{n+1,1}) = \frac{\theta^n}{\Delta t} + \mathcal{L}(\theta^n),$$
$$\frac{\theta^{n+1,2}}{\Delta t} + \mathcal{L}(\theta^{n+1,2}) = -\frac{\mathcal{N}(\theta^n)}{\sqrt{\mathcal{I}(\theta^n)+C}},$$

from which we derive

$$\theta^{n+1,1} = \theta^n,\ \theta^{n+1,2} = -\frac{\Delta t}{\sqrt{\mathcal{I}(\theta^n)+C}}(I+\Delta t\mathcal{L})^{-1}\mathcal{N}(\theta^n).$$

Then we substitute the above equations into (17b) to get

$$r^{n+1} = \frac{r^n}{1+\Delta t\frac{(\mathcal{N}(\theta^n),(I+\Delta t\mathcal{L})^{-1}\mathcal{N}(\theta^n))}{2(\mathcal{I}(\theta^n)+C)}}. \tag{20}$$

Now we obtain θ^{n+1} in view of (19). We summarize the SAV method in Algorithm 2.

Algorithm 2 SAV scheme.

Input: $\theta^0, \Delta t, r^0 = \sqrt{\mathcal{I}(\theta^0)+C}$
Output: θ^N
for $n = 0, 1, ..., N-1$ **do**
  $\theta^{n+1,1} = \theta^n$
  $\theta^{n+1,2} = -\frac{\Delta t}{\sqrt{\mathcal{I}(\theta^n)+C}}(I+\Delta t\mathcal{L})^{-1}\mathcal{N}(\theta^n)$
  $r^{n+1} = \frac{r^n}{1+\Delta t\frac{(\mathcal{N}(\theta^n),(I+\Delta t\mathcal{L})^{-1}\mathcal{N}(\theta^n))}{2(\mathcal{I}(\theta^n)+C)}}$
  $\theta^{n+1} = \theta^{n+1,1} + r^{n+1}\theta^{n+1,2}$
end for
return θ^N

2.2.2. Restart SAV method

Theorem 1 states that the modified energy $(r^n)^2$ is unconditionally stable. However, for the vanilla SAV, the computed r^n may not be a good approximation of $r(t^n)$. For instance, r^n may go to 0 rapidly if Δt or $\frac{(\mathcal{N}(\theta^n),(I+\Delta t\mathcal{L})^{-1}\mathcal{N}(\theta^n))}{2(\mathcal{I}(\theta^n)+C)}$ is too big. Therefore, we may lose accuracy and even obtain a wrong steady state solution. The main reason for this is that r^n is not directly linked to $r(t^n)$. To resolve this issue, a simple idea is to introduce

$$\hat{r}^n := r(\theta^n) = \sqrt{\mathcal{I}(\theta^n)+C}$$

directly according to the definition, and renew r^{n+1} using

$$r^{n+1} = \frac{\hat{r}^n}{1+\Delta t\frac{(\mathcal{N}(\theta^n),(I+\Delta t^n\mathcal{L})^{-1}\mathcal{N}(\theta^n))}{2(\mathcal{I}(\theta^n)+C)}}$$

instead of using (20). We call this algorithm as Restart SAV (ResSAV) and state it in Algorithm 3.

Algorithm 3 Restart SAV scheme.

Input: $\theta^0, \Delta t$
Output: θ^N
for $n = 0, 1, ..., N-1$ **do**
  $\hat{r}^n = \sqrt{\mathcal{I}(\theta^n)+C}$
  $\theta^{n+1,1} = \theta^n$
  $\theta^{n+1,2} = -\frac{\Delta t}{\sqrt{\mathcal{I}(\theta^n)+C}}(I+\Delta t\mathcal{L})^{-1}\mathcal{N}(\theta^n)$
  $r^{n+1} = \frac{\hat{r}^n}{1+\Delta t\frac{(\mathcal{N}(\theta^n),(I+\Delta t\mathcal{L})^{-1}\mathcal{N}(\theta^n))}{2(\mathcal{I}(\theta^n)+C)}}$
  $\theta^{n+1} = \theta^{n+1,1} + r^{n+1}\theta^{n+1,2}$
end for
return θ^N

2.2.3. Relaxed SAV method

The restart SAV usually performs better than the vanilla SAV, however, it no longer guarantees an energy dissipation law. Therefore, we adopt the idea of relaxed SAV proposed in [35] that enjoys the energy dissipation law and, at the same time, links r^n directly to $r(t^n)$. Specifically, we employ the relaxation technique to update r^{n+1} by using a linear combination of the value computed by using (20) (here we denote it by $\tilde{r}^{n+1}$) and the value of $\hat{r}^{n+1}$ given by

$$\hat{r}^{n+1} := \sqrt{I(\theta^{n+1}) + C}, \tag{21}$$

leading to the following relaxed SAV method

$$\frac{\theta^{n+1} - \theta^n}{\Delta t} + \mathcal{L}(\theta^{n+1}) + \frac{\mathcal{N}(\theta^n)}{\sqrt{I(\theta^n) + C}} \tilde{r}^{n+1} - \mathcal{L}(\theta^n) = 0, \tag{22a}$$

$$\frac{\tilde{r}^{n+1} - r^n}{\Delta t} = \frac{1}{2\sqrt{I(\theta^n) + C}} \left(\mathcal{N}(\theta^n), \frac{\theta^{n+1} - \theta^n}{\Delta t} \right), \tag{22b}$$

$$r^{n+1} = \xi_0 \tilde{r}^{n+1} + (1 - \xi_0)\hat{r}^{n+1}, \tag{22c}$$

where ξ_0 is determined by solving the following optimization problem

$$\xi_0 = \min_{\xi \in [0,1]} \xi, \qquad s.t. (r^{n+1})^2 - (\tilde{r}^{n+1})^2 \le \eta \frac{\|\theta^{n+1} - \theta^n\|^2}{\Delta t}. \tag{23}$$

Here $\eta \in [0, 1]$ is an artificial parameter with a default value of 0.99. It has been shown in [35] that the relaxation parameter ξ_0 is given by

$$\xi_0 = \max \left\{ 0, \frac{-b - \sqrt{b^2 - 4ac}}{2a} \right\}, \tag{24}$$

where the coefficients are given by

$$a = (\tilde{r}^{n+1} - \hat{r}^{n+1})^2, \quad b = 2\hat{r}^{n+1}(\tilde{r}^{n+1} - \hat{r}^{n+1}),$$

$$c = (\hat{r}^{n+1})^2 - (\tilde{r}^{n+1})^2 - \eta \frac{\|\theta^{n+1} - \theta^n\|^2}{\Delta t}.$$

By using (18) with r^{n+1} replaced by $\tilde{r}^{n+1}$ and (23), we have immediately the following:

Theorem 2. *The relaxed SAV scheme (22) is unconditionally energy stable in the sense that*

$$\begin{aligned} &(r^{n+1})^2 - (r^n)^2 \\ &= -(\mathcal{L}(\theta^{n+1} - \theta^n), \theta^{n+1} - \theta^n) - (\tilde{r}^{n+1} - r^n)^2 - (1 - \eta) \frac{\|\theta^{n+1} - \theta^n\|^2}{\Delta t} \le 0. \end{aligned}$$

The relaxed SAV scheme can be implemented as follows:

Algorithm 4 Relaxed SAV scheme.

Input: $\theta_k^0, \Delta t, r^0 = \sqrt{I(\theta^0) + C}$
Output: θ^N
for $n = 0, 1, ..., N-1$ **do**
  Obtain θ^{n+1} and $\tilde{r}^{n+1}$ by using Algorithm 2
  $\hat{r}^{n+1} = \sqrt{I(\theta^{n+1}) + C}$
  $a = (\tilde{r}^{n+1} - \hat{r}^{n+1})^2$
  $b = 2\hat{r}^{n+1}(\tilde{r}^{n+1} - \hat{r}^{n+1})$
  $c = (\hat{r}^{n+1})^2 - (\tilde{r}^{n+1})^2 - \eta \frac{\|\theta^{n+1} - \theta^n\|^2}{\Delta t}$
  $\xi_0 = max\{0, \frac{-b - \sqrt{b^2 - 4ac}}{2a}\}$
  $r^{n+1} = \xi_0 \tilde{r}^{n+1} + (1 - \xi_0)\hat{r}^{n+1}$
end for
return θ^N

2.2.4. Adaptive SAV methods

When solving complex nonlinear PDEs, the adaptive time step is usually used to improve the efficiency and accuracy. In particular for optimizing algorithms in deep learning, one of the most commonly used methods is the Adam method [18], which can be seen as a GD-based algorithm that uses a modified learning rate and gradient to update parameters. We now propose an adaptive version

Table 1
Notations of parameters.

# of neurons	m	# of data	M	Dimension of the problem	D
batch size	l	SAV constant	C	Coefficient for the linear operator	λ

for the SAV schemes based on the Adam algorithm. More precisely, we use the Adam algorithm (3)-(4) to update the time step and the nonlinear term, and then substitute them into the corresponding SAV methods to update the parameters, i.e.:

- Step 1: Compute $\hat{\mathcal{N}}(\theta^n)$ and $\widehat{\Delta t}$ using the adaptive strategy of the Adam method;
- Step 2: Use the adaptive corrective terms to update θ^{n+1} using SAV-based schemes.

We denote these adaptive SAV schemes as A-SAV, A-ResSAV and A-RelSAV corresponding to the vanilla SAV, the restart SAV and the relaxed SAV, respectively. The details for the adaptive implementation are given as follows:

Algorithm 5 The adaptive SAV scheme.

Input: $\theta^0, \Delta t, r^0, \epsilon = 1e-8, \beta_1 = 0.9, \beta_2 = 0.999, m^0 = 0, v^0 = 0$
Output: θ^N
for $n = 0, 1, ..., N-1$ **do**
  $m^{n+1} = \beta_1 m^n + (1-\beta_1)\mathcal{N}(\theta^n)$
  $v^{n+1} = \beta_2 v^n + (1-\beta_2)\|\mathcal{N}(\theta^n)\|^2$
  $\hat{m}^{n+1} = \frac{m^{n+1}}{1-(\beta_1)^{n+1}}$
  $\hat{v}^{n+1} = \frac{v^{n+1}}{1-(\beta_2)^{n+1}}$
  $\hat{\mathcal{N}}(\theta^n) = \hat{m}^{n+1}$
  $\widehat{\Delta t} = \frac{\Delta t}{\sqrt{\hat{v}^{n+1}}+\epsilon}$
  update θ^{n+1} by using Algorithm 2, 3 and 4, respectively
end for
return θ^N

3. Numerical results

In this section, we present several numerical examples, including three regression problems as well as a classification problem to illustrate the effectiveness of the present methods. In all numerical examples, we use "ReLU" as the activate function if not mentioned. We use PM-Euler to denote the PM with the forward Euler scheme, and PM-SAV to denote the PM with the SAV scheme, likewise for PM-ResSAV, PM-RelSAV, SPM-Euler, SPM-SAV, SPM-ResSAV, SPM-RelSAV. We also use SPM-A-SAV to denote the adaptive version of SPM-SAV, and similar notations are used for other methods. We point out here that the PM-Euler is actually the GD method or the SGD method if the mini-batch technique is used. Namely, the PM-Euler method is equivalent to the GD method, or the SGD method if the mini-batch technique is used. Numerical results show that the results obtained by using the SPM-Euler method are almost the same as or a little better than the ones obtained by using the PM-Euler method, see the result in Fig. 2. Similarly, the results obtained by using the SPM-A-Euler method are almost same as or a little better than the ones obtained by using the PM-A-Euler method, which is actually the Adam method, one of the most popularly used optimization method in deep learning. For the learning rate, we use either the time step Δt or the notation lr. In all numerical tests, we take the linear semi positive-definite operator to be $\mathcal{L}(\theta) = \lambda\theta$ with a suitable $\lambda \geq 0$.

For the sake of convenience, we also list some other frequently used parameters in Table 1.

3.1. Regression problem

We first consider the regression problem. Note that before calculation, we process data with the Z-score normalization

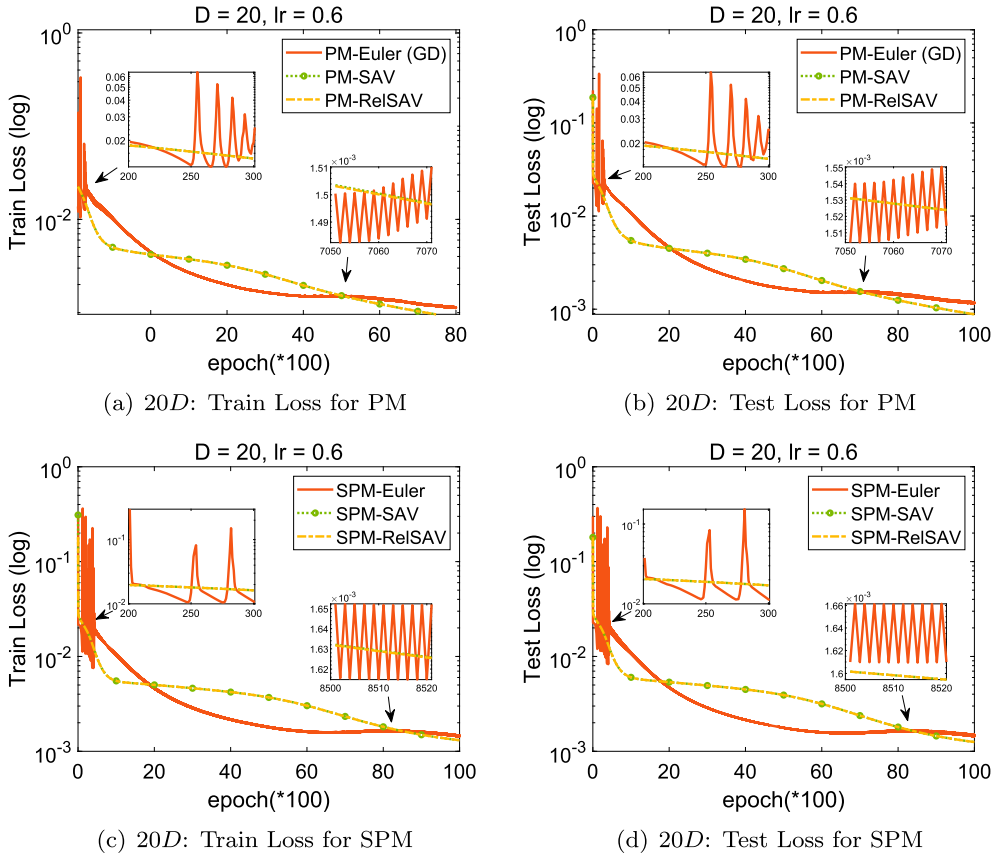
$$\mathbf{x} \Rightarrow \frac{\mathbf{x}-\mu}{\sigma},$$

where μ, σ are the mean and standard deviation of the data, respectively, and then take the relative error as the loss function. For each case, we use 80% of the data for training while use the other 20% of the data for testing.

3.1.1. Example 1

Example 1. Consider the following target function:

$$f^*(x_1, ..., x_D) = \sin\left(\sum_{i=1}^{D} p_i x_i\right) + \cos\left(\sum_{i=1}^{D} q_i x_i\right), \tag{25}$$



(a) $20D$: Train Loss for PM (b) $20D$: Test Loss for PM

(c) $20D$: Train Loss for SPM (d) $20D$: Test Loss for SPM

Fig. 1. Example 1: Train loss (left) and test loss (right) using the full batch of data. Upper: PM method, lower: SPM method. Other parameters are set as $m = 1000$, $M = 10000$, $l = 8000$, $lr = 0.6$, $C = 1$, $\lambda = 10$.

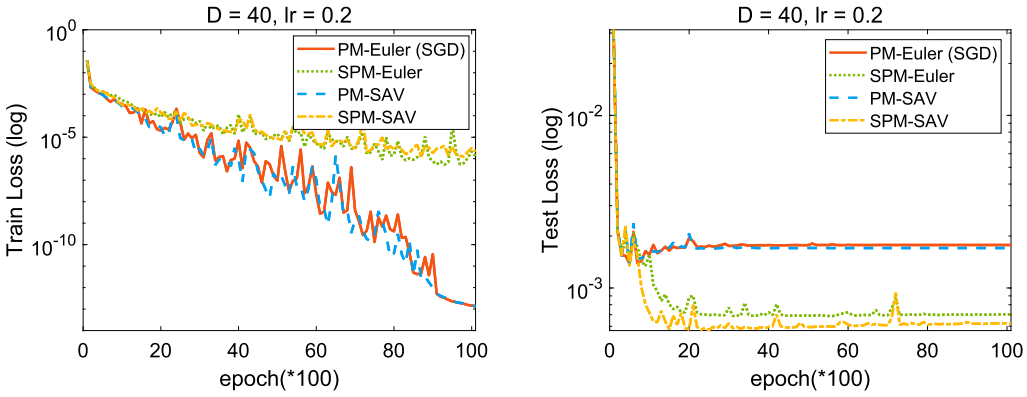
where $\{p_i, q_i\}_{1\le i\le D}$ are given coefficients, D is the dimension. We generate a random data set $\{\mathbf{x}_j, f_j^*\}_{1\le j\le M}$, where $\mathbf{x}_j \in (0,1)^D$.

We begin by verifying the theoretical results given in Theorems 1 and 2, i.e., the discrete energy dissipation law, with full batch of training data. To this end, we employ both PM and SPM methods with or without SAV/Relaxed SAV schemes for a 20D problem using $M = 10000$ data. The learning rate, i.e., the time step, is set to be 0.6, and other parameters for the SAV scheme are $C = 1, \lambda = 10$. If not otherwise specified, the parameters for the smoothed particle method are set to be $h = 0.0001, J = 10$. The results are shown in Fig. 1. We observe that the train and test losses obtained by using the SAV method or the relaxed SAV method decay asymptotically. This is in agreement with Theorem 1 and 2. However, we observe oscillations for the result obtained without using SAV-based methods.

As mentioned before, it is usually computationally expansive to use full data. Therefore, we shall employ the stochastic version of SAV-based methods, i.e., utilize the mini-batch technique as in SGD. Hereafter, if not otherwise specified, we use the stochastic SAV-based methods in all following numerical tests.

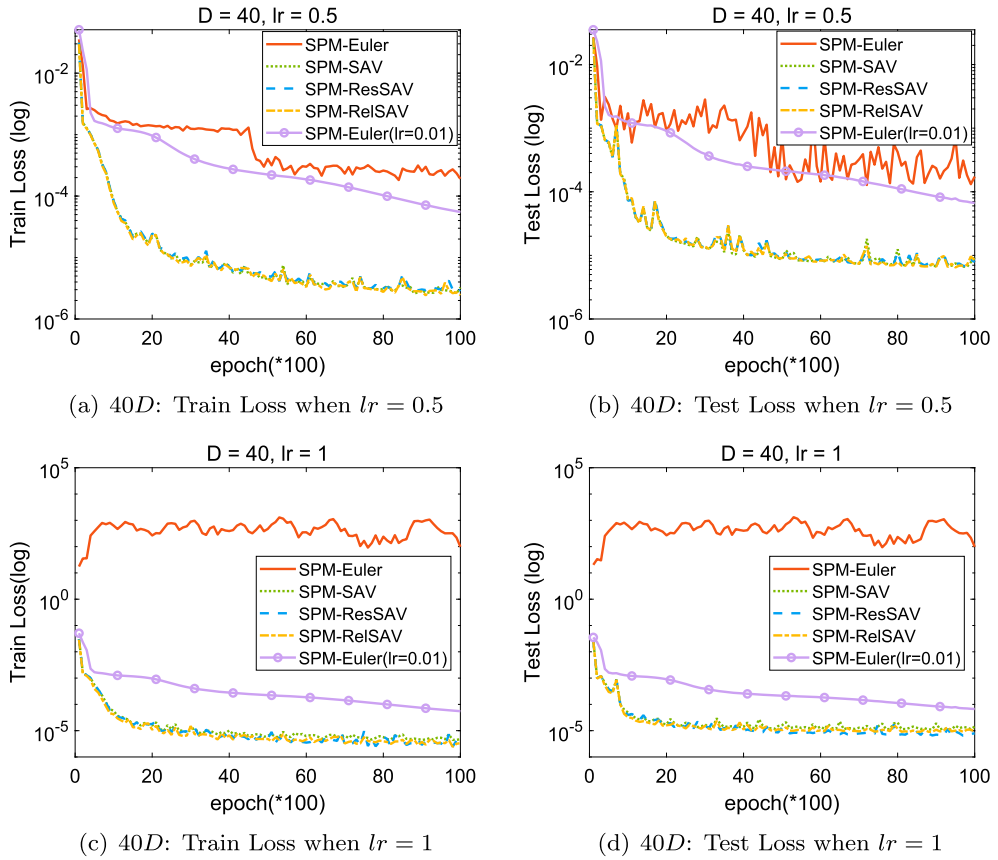
To demonstrate the effectiveness of the SPM method, we show numerical results in Fig. 2 for Example 1 with $D = 40$, $M = 10000$, $m = 1000$ by comparing the accuracy between the results obtained by using PM-based methods and the ones obtained by using SPM-based methods. The batch size used here is $l = 64$, and the other parameters are set as follows: $C = 1$, $\lambda = 4$. Observe that we obtain higher (but not significantly improved) accuracy with SPM-based methods compared with PM-based methods. We also test different sets of data with different numbers and different dimensions showing a similar result. One of our future works is to develop a highly accurate space discretization method.

Next, we show how the SAV-based schemes improve the efficiency and stability of the training process. Consider Example 1 with $D = 40$, $m = 1000$, $M = 100000$. We implement both PM and SPM methods, and the behaviors for both methods are almost the same. Thus, we only show the result with the SPM method. The train and test losses using different types of SAV-based methods with different (fixed) learning rates are shown in Fig. 3. The result obtained by using SGD with a default learning rate 0.01 is presented as the baseline. We observe that all results obtained with the three SAV-based schemes (i.e., the vanilla SAV, the restart SAV and the relaxed SAV) are almost the same. For the case of $lr = 0.5$, it shows that the test loss using SAV-based methods is about 2 orders of magnitude higher than that using the Euler scheme. For the case of $lr = 1$, the test loss using SAV-based schemes is almost the same as the case of $lr = 0.5$ while the solution obtained by using the SPM-Euler method even does not converge. This also indicates that



(a) $40D$: Train Loss (b) $40D$: Test Loss

Fig. 2. Example 1: Train loss (left) and the test loss (right) for $40D$. Other parameters are set as: $m = 1000$, $M = 10000$, $l = 64$, $lr = 0.2$, $C = 1$, $\lambda = 0$. c.



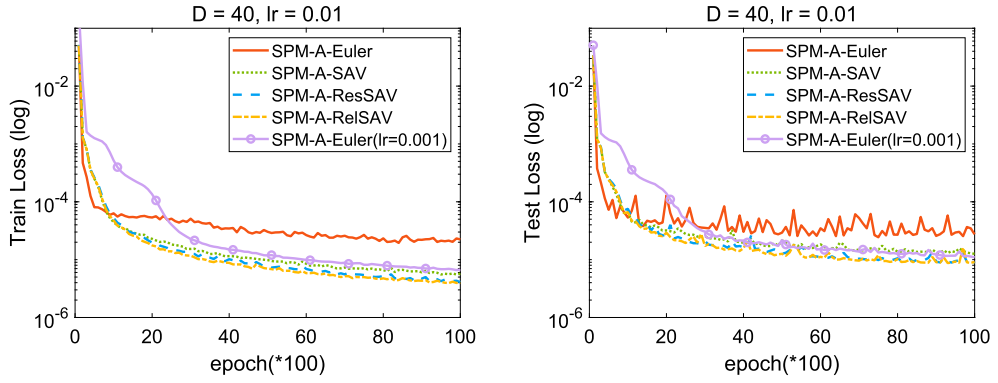
(a) $40D$: Train Loss when $lr = 0.5$ (b) $40D$: Test Loss when $lr = 0.5$

(c) $40D$: Train Loss when $lr = 1$ (d) $40D$: Test Loss when $lr = 1$

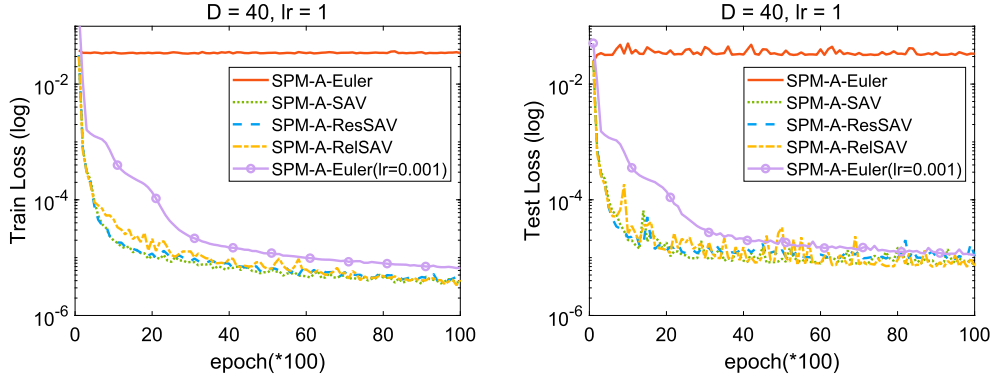
Fig. 3. Example 1: Train loss (left) and the test loss (right) with different fixed learning rate. Upper: $lr = 0.5$, lower: $lr = 1$. Other parameters are set as: $m = 1000$, $M = 100000$, $l = 256$, $C = 100$, $\lambda = 4$.

the training process is stable by using the SAV-based methods with respect to the learning rate. Furthermore, we observe that all obtained results are much better than those obtained by using SPM-Euler with the default learning rate ($lr = 0.01$).

The fixed learning rate for each case is used in the above tests. Next we consider the case of using the adaptive learning rate. More precisely, we solve the above problem by combining the SAV-based schemes and the adaptive strategy used in Adam algorithm, see Algorithm 5 for more details. We show in Fig. 4 the comparison of the results between the SAV-based methods and non-SAV-based method for different initial learning rates ($lr = 0.01$ and $lr = 1$). We also present the result obtained by using the Adam method with the default initial learning rate $lr = 0.001$ as the baseline. Again, the accuracy using SAV-based methods is much better than that using a non-SAV-based scheme with the same (big) initial learning rate, and even slightly better than that using Adam with a small



(a) $40D$: Train Loss when $lr = 0.01$ (b) $40D$: Test Loss when $lr = 0.01$



(c) $40D$: Train Loss when $lr = 1$ (d) $40D$: Test Loss when $lr = 1$

Fig. 4. Example 1: Train loss (left) and the test loss (right) with different initial learning rate for adaptive method. Upper: $lr = 0.01$, lower: $lr = 1$. Other parameters are set as:: $m = 1000, M = 100000, l = 256, C = 100, \lambda = 4$.

initial learning rate (i.e., the default case). It also converges much faster than the default case, i.e., the case of using Adam method with the initial learning rate $lr = 0.001$. Moreover, even with an adaptive learning rate, we cannot obtain a convergent solution with a relatively big initial learning rate (e.g., $lr = 1$) with the non-SAV based schemes. On the other hand, we observe that the accuracy by using SAV-based schemes with an adaptive learning rate is more or less the same as that by using a fixed learning rate, i.e., the advantage of adaptive learning rate is not significant for this example. However, as we will see in the next example that the adaptive learning rate is necessary to improve the efficiency for more complex problems.

We also investigate the influence of the coefficient λ of the linear operator for the SAV-based schemes. As we mentioned previously, the vanilla SAV scheme may give a wrong solution. We first employ the SPM method and SAV scheme with $lr = 0.5$ and different values of $\lambda = 0, 1, 4, 10$. The train and test losses are shown in Fig. 5 showing that $\lambda = 4$ gives the best result for the SAV scheme. We also employ the SPM method with restart and relaxed SAV methods with different values of λ. Numerical results show that the parameter λ just weakly affects the restart and relaxed SAV methods. We further demonstrate this by plotting the value of $1 - q^n$, where

$$q^n = \frac{r^n}{\hat{r}^n},$$

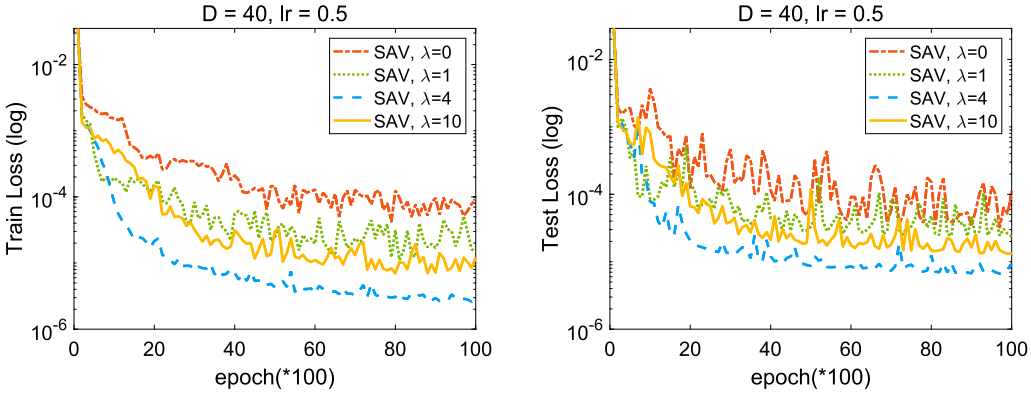
for $\lambda = 0, 4$ and the three SAV-based schemes in Fig. 6. The ratio q^n (or $1 - q^n$) should be an approximation of 1 (or 0). We observe that $1 - q^n \approx 0.6$ for the case of SAV with $\lambda = 0$ while $1 - q^n \approx 0$ for the other cases.

3.1.2. *Example 2*

For Example 1, we obtained similar accuracy when using fixed and adaptive learning rates. Particularly, we obtained high accuracy solutions as well as efficient and stable training processes in both cases. Now we show in the next example the advantage of using the adaptive learning rate.

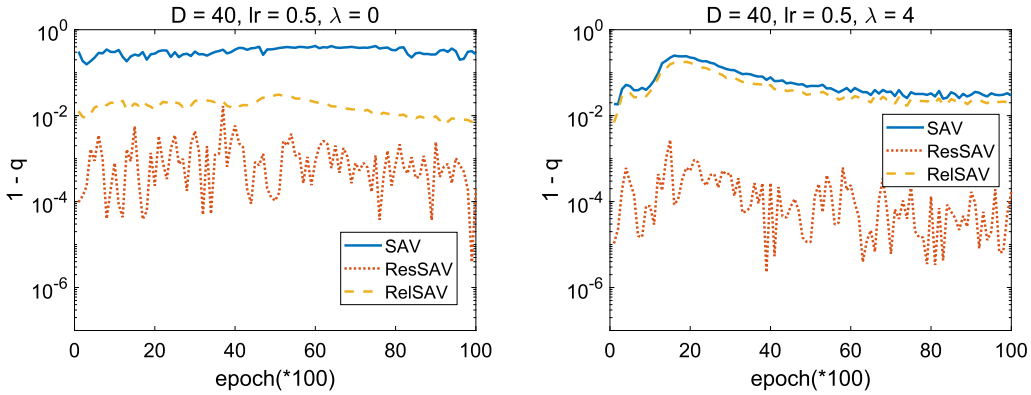
Example 2. Let us consider the following quadratic function:

$$f^*(x_1, ..., x_D) = \sum_{i=1}^{D} c_i x_i^2, \tag{26}$$



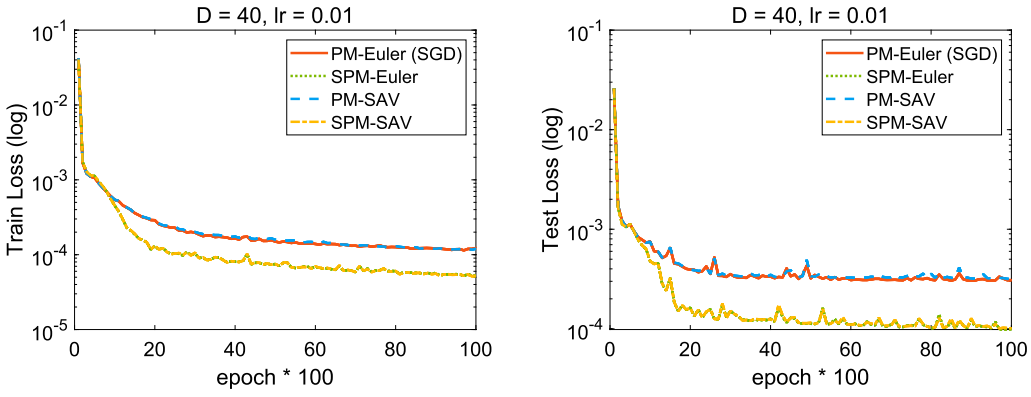
(a) $40D$: Train Loss (b) $40D$: Test Loss

Fig. 5. Example 1: Train loss (left) and test loss (right) with different values of the coefficient λ. Other parameters are $m = 1000, M = 100000, l = 256, lr = 0.5, C = 100$.



(a) $1 - q$ for $\lambda = 0, C = 10$ (b) $1 - q$ for $\lambda = 4, C = 10$

Fig. 6. Example 1: The values of $1 - q$ for SAV, ResSAV and RelSAV methods with different values of the parameter λ. Left: $\lambda = 0$, right: $\lambda = 4$. Other parameters are: $m = 1000, M = 100000, l = 256, lr = 0.5, C = 10$.

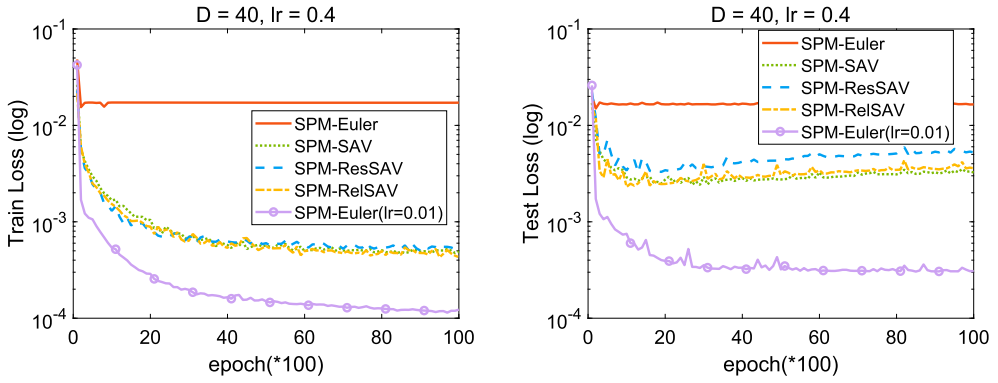


(a) $40D$: Train Loss (b) $40D$: Test Loss

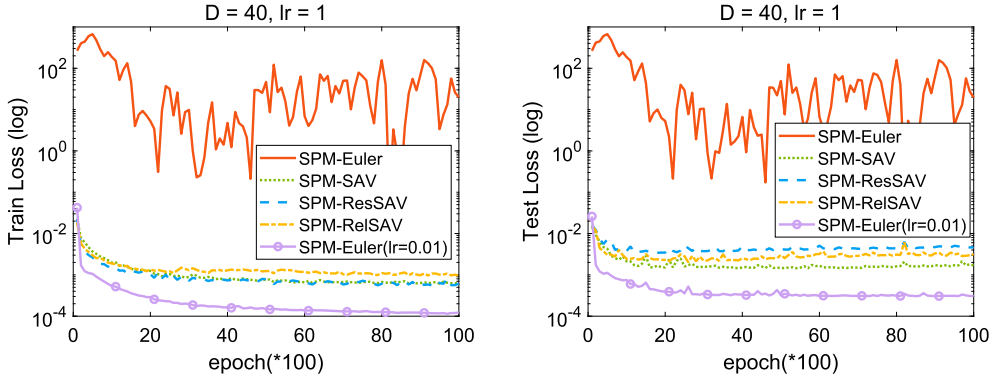
Fig. 7. Example 2: Train loss (left) and the test loss (right) for the 40D. Other parameters are set as: $m = 100, M = 10000, l = 64, lr = 0.01, C = 1, \lambda = 0$.

where $x \in [0,5]^D$, $\{c_i\}_{1 \le i \le D}$ is a set of preset coefficients.

We first compare the accuracy of the solutions between the PM method and the SPM method by considering Example 2 with $D = 40$, $m = 100$, $M = 10000$, $l = 64$, $lr = 0.01$, $C = 1$, $\lambda = 0$. Again, the result obtained by using the SPM method is slightly better than the one obtained by using the PM method (see Fig. 7).



(a) $40D$: Train Loss when $lr = 0.5$ (b) $40D$: Test Loss when $lr = 0.5$



(c) $40D$: Train Loss when $lr = 1$ (d) $40D$: Test Loss when $lr = 1$

Fig. 8. Example 2: Train loss (left) and the test loss (right) with different fixed learning rate: $lr = 0.4$ (upper), 1 (lower). Other parameters are set as: $m = 100, M = 10000, l = 64, C = 1, \lambda = 4$.

We then employ the present SPM method with different time stepping schemes using the fixed learning rate for Example 2 with $D = 40$, $m = 100$ $M = 1000$, $C = 1, \lambda = 4$. The result of train and test losses for different learning rates ($lr = 0.4$, 1) is shown in Fig. 8. The result using the SPM method with the forward Euler scheme with a default learning rate $lr = 0.01$ is used as a benchmark.

For the SPM method with the forward Euler scheme using a large learning rate, the solutions do not converge. However, we always obtain convergent solution by using the SAV-based schemes. Nevertheless, unlike in the case of Example 1, we observe in the current case that we lose about one order of magnitude on accuracy for the solution using the SAV-based schemes with a large fixed learning rate, compared with the solution using $lr = 0.01$. This prevents us to use a large learning rate to obtain high accurate solutions.

To resolve the aforementioned issue, we need to use the adaptive learning rate. We next employ the adaptive algorithms using the Adam strategy and SAV-based schemes, similarly as in the previous example. The results with different initial learning rates ($lr = 0.1, 1$) are presented in Fig. 9. The result using the adaptive SPM-Euler with a default initial learning rate $lr = 0.001$ is also presented as the benchmark. Again, we observe that the accuracy of the solution using SAV-based schemes is better than that using the non-SAV schemes. Moreover, unlike the case with fixed learning rates, here the accuracy of the solution using adaptive SAV-based schemes is almost the same as that in the benchmark (i.e., the case using the Adam method with initial learning rate $lr = 0.001$). Moreover, if we take a look at the left bottom of the plots, we observe that the SAV-based schemes with a big initial learning rate converge faster when compared with the benchmark.

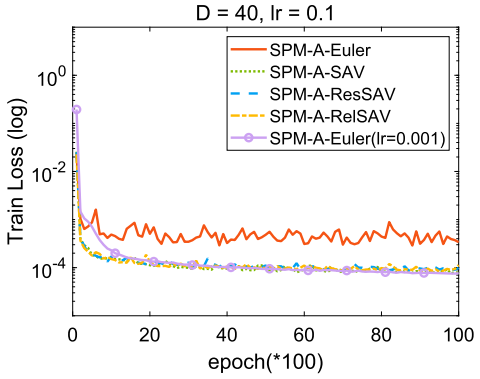
3.1.3. Example 3

We now consider a more difficult regression problem.

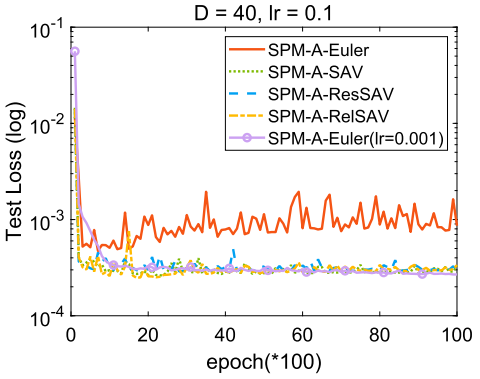
Example 3. Consider the following target function,

$$f^*(\mathbf{x}) = e^{-10\|\mathbf{x}\|_2}, \quad \mathbf{x} = (x_1, ..., x_D), \tag{27}$$

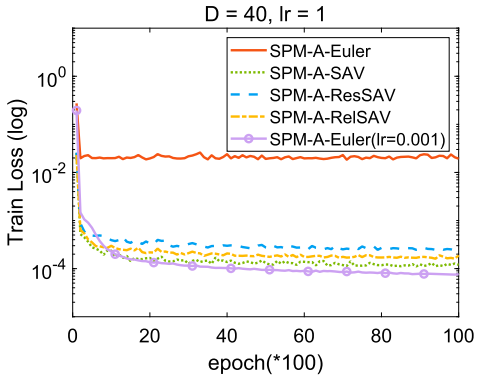
which is a high dimensional Gaussian function with steep gradients near the origin.



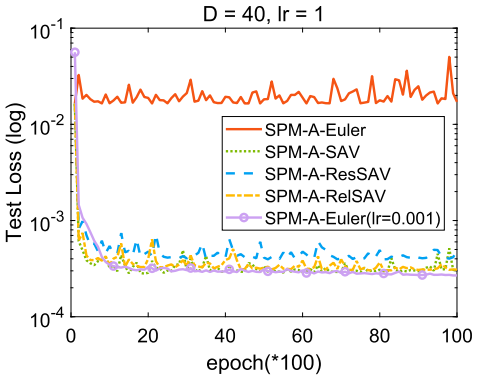
(a) $40D$: Train Loss when $lr = 0.1$



(b) $40D$: Test Loss when $lr = 0.1$



(c) $40D$: Train Loss when $lr = 1$



(d) $40D$: Test Loss when $lr = 1$

Fig. 9. Example 2: Train loss (left) and the test loss (right) with different initial learning rates for adaptive method: $lr = 0.1$ (upper), 0.1 (lower). Other parameters are set as: $m = 100, M = 10000, l = 64, C = 1, \lambda = 4$.

Since the target function is very sharp near the origin, we use a nonuniform distributed data set, namely, we randomly generate 100000 data with $\{\mathbf{x}\} \sim \mathcal{N}(0, 0.2)$. Other parameters are given as follows: $m = 10000$, $h = 0.000001$. The batch size is $l = 256$. We show in Fig. 10 the result using SPM method with the relaxed SAV scheme and the non-SAV scheme, and employ both fixed and adaptive learning rates by setting the (initial) learning rate to be $lr = 1$. Again, the result using adaptive SPM-Euler method, Adagrad, RMSprop with the default learning rate $lr = 0.001$ are presented for comparison. It shows that the relaxed SAV scheme is more efficient and stable than the Euler method with either the fixed learning rate or the adaptive learning rate. In addition, the solution using relaxed SAV with the adaptive learning rate has better accuracy and a much faster convergence than adaptive SPM-Euler with the default initial learning rate. This means that the adaptive relaxed SAV approach is more efficient that the Adam approach.

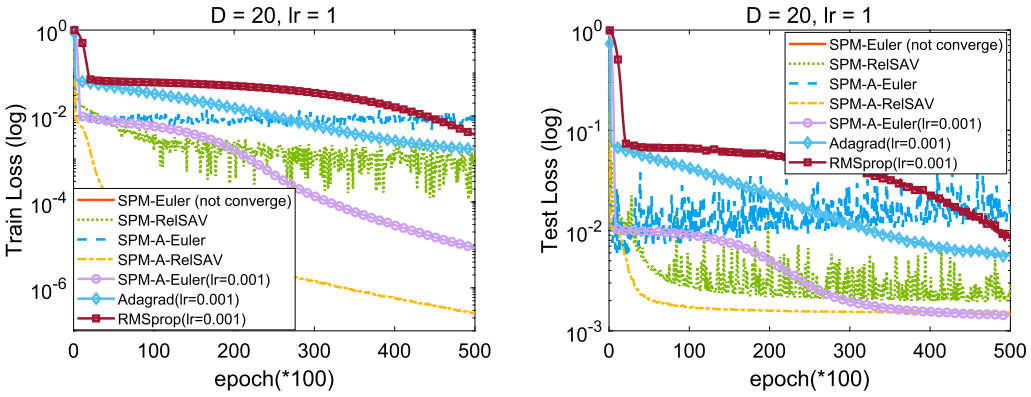
3.2. Classification problem

Example 4. We consider in this subsection the classification problem, i.e., the MNIST (Mixed National Institute of Standards and Technology) problem [36], whose dataset is a large-scale handwritten digit database collected by the National Institute of Standards and Technology, containing a training set of 60000 samples and a test set of 10000 samples.

The results on the accuracy, train loss and test loss using different methods are shown in Fig. 11. We observe that there is no obvious difference between the four algorithms. This is because classification problems do not require as high precision as regression problems, We only need to approximate the label with about 10% of accuracy to get the correct classification. Nevertheless, the accuracy of the SPM method is still slightly better than the PM method. Also, the train loss using relaxed SAV method is slightly better than the one using a non-SAV method.

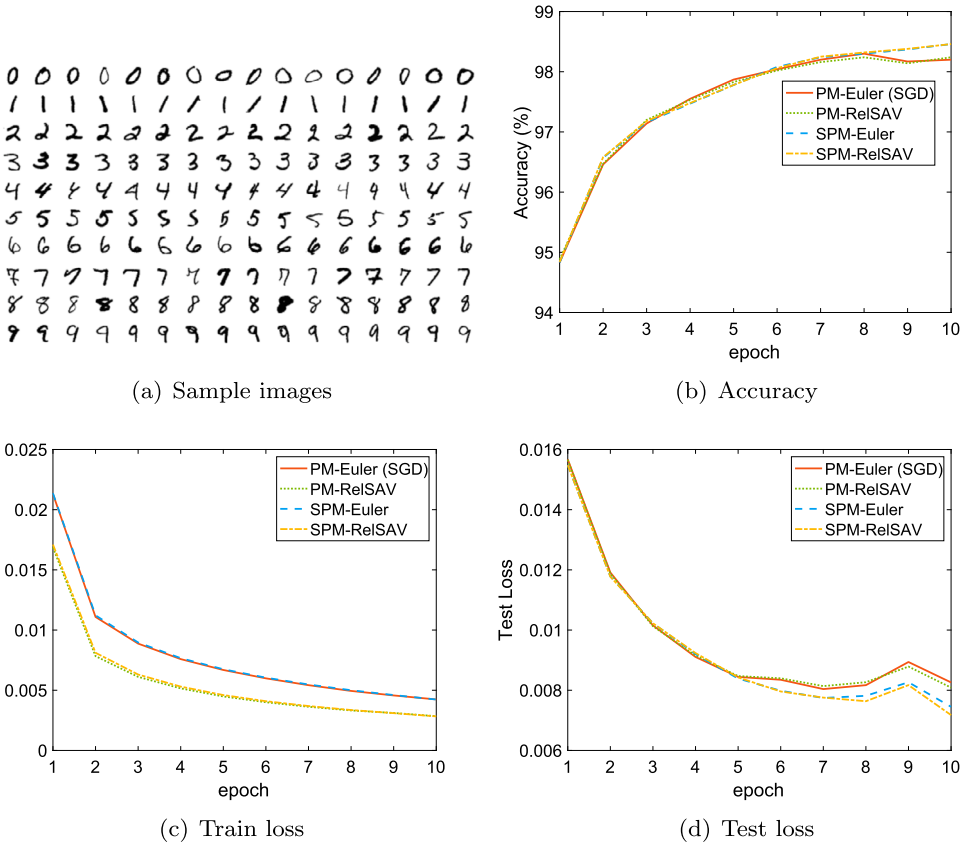
3.3. Solving partial differential equations

Besides the function approximation problems, we also consider in this subsection solving PDEs using the neural network-based method with the present SAV-based optimization methods.



(a) $20D$: Train Loss (b) $20D$: Test Loss

Fig. 10. Example 3: Train loss (left) and the test loss (right) between the results obtained by using Euler, RelSAV, A-Euler and A-RelSAV, as well as two other popularly used adaptive methods (i.e., Adagrad and RMSprop with default learning rate 0.001). Other parameters are set as: $m = 10000$, $M = 100000$, $l = 256$, $C = 100$, $\lambda = 4$.



(a) Sample images (b) Accuracy

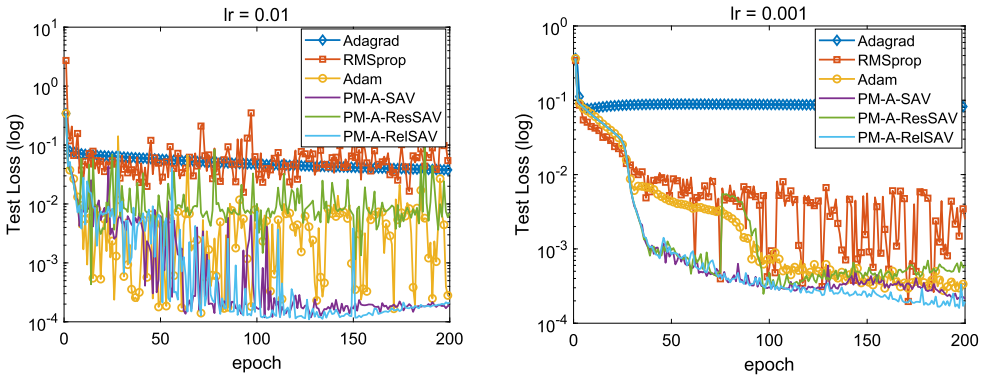
(c) Train loss (d) Test loss

Fig. 11. Example 4 (classification problem): (a) Sample images from MNIST test dataset. (b) Accuracy. (c) Train loss. (d) Test loss.

Example 5. Consider the following burgers equation:

$$\begin{cases} u_t + uu_x = \nu u_{xx}, x \in (-1,1), t \in (0,T], \\ u(x,0) = -\sin \pi x, x \in (-1,1), \\ u(-1,t) = u(1,t) = 0, t \in [0,T], \end{cases} \tag{28}$$

where $\nu = 0.01/\pi$ and $T = 1$. Here we adopt PINNs to solve the above equation and use *tanh* as the activate function. We use 200×60 equally distributed residual points for $x \times t$ domain. Since the expressive ability of the single hidden layer neural network is too weak



(a) Test error, $lr = 0.01$ (b) Test error, $lr = 0.001$

Fig. 12. Example 5 Test losses obtained by different optimizers. (a) $lr = 0.01$. (b) $lr = 0.001$. Other parameters are set as $C = 10000$, $\lambda = 4$.

to solve the above Burgers equation, thus we use a neural network with 5 hidden layers and 32 neurons for each layer. Furthermore, we use PM approach in the case since we do not have a continuous PDE form like (6).

We compare the test errors between the ones obtained by using the present adaptive SAV-based methods and the ones obtained by using several popularly used adaptive optimization methods (i.e., Adagrad, RMSprop and Adam) in Fig. 12 for $lr = 0.01$ and $lr = 0.001$, respectively. Observe that the present adaptive SAV-based methods enjoy superior efficiency and accuracy. Furthermore, for a big learning rate ($lr = 0.01$), the results of Adagrad and RMSprop even do not converge.

4. Conclusion and discussion

We considered in this paper efficient and stable numerical methods for solving gradient flows arising from deep learning. For the space discretization, we implemented both particle method (PM) and smoothed particle method (SPM), and showed that the SPM is slightly better than the PM in terms of accuracy. For the time discretization, we adopted several efficient SAV-based time stepping schemes, and developed their adaptive version by applying the adaptive strategy used in Adam. Numerical results show that the SAV-based schemes significantly improve the efficiency and robustness of the training process and the accuracy of the solution when using large learning rates by comparing with the SGD method or the Adam method. When compared with the Adam method with a small learning rate, we obtain in most cases much more accurate solutions with much higher efficiency by the proposed methods.

In the present work, we mainly considered the function approximation problem by only one-hidden layer neural network. However, the proposed methods can be extended to the multi-layer neural network and different architectures of neural networks, see Example 5. We shall consider extending the proposed methods to different problems, for instance, solving more complex PDEs, with different neural network architectures in a future work.

CRediT authorship contribution statement

Ziqi Ma: Data curation, Investigation, Methodology, Visualization, Writing – original draft, Writing – review & editing. **Zhiping Mao:** Conceptualization, Data curation, Funding acquisition, Investigation, Methodology, Project administration, Supervision, Writing – original draft, Writing – review & editing. **Jie Shen:** Conceptualization, Funding acquisition, Investigation, Methodology, Project administration, Supervision, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgement

This work is supported by the National Key R&D Program of China under Grant No. 2022YFA1004500 and the National Natural Science Foundation of China under Grant No. 12171404 and No. 12371409.

References

[1] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, Commun. ACM 60 (2017) 84–90.
[2] J. Long, E. Shelhamer, T. Darrell, Fully convolutional networks for semantic segmentation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 3431–3440.
[3] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, S. Khudanpur, Recurrent neural network based language model, in: Interspeech, Volume 2, Makuhari, 2010, pp. 1045–1048.
[4] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial networks, Commun. ACM 63 (2020) 139–144.
[5] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.
[6] L. Lu, P. Jin, G. Pang, Z. Zhang, G.E. Karniadakis, Learning nonlinear operators via deeponet based on the universal approximation theorem of operators, Nat. Mach. Intell. 3 (2021) 218–229.
[7] S. Liang, S.W. Jiang, J. Harlim, H. Yang, Solving PDEs on unknown manifolds with machine learning, arXiv:2106.06682 [abs], 2021.
[8] J. Han, A. Jentzen, W. E, Solving high-dimensional partial differential equations using deep learning, Proc. Natl. Acad. Sci. 115 (2018) 8505–8510.
[9] C. Huré, H. Pham, X. Warin, Deep backward schemes for high-dimensional nonlinear PDEs, Math. Comput. 89 (2020) 1547–1579.
[10] L. Lu, X. Meng, Z. Mao, G.E. Karniadakis, DeepXDE: a deep learning library for solving differential equations, SIAM Rev. 63 (2021) 208–228.
[11] T. Muther, A.K. Dahaghi, F.I. Syed, V. Van Pham, Physical laws meet machine intelligence: current developments and future directions, Artif. Intell. Rev. (2022) 1–67.
[12] J. Sirignano, K. Spiliopoulos, DGM: a deep learning algorithm for solving partial differential equations, J. Comput. Phys. 375 (2018) 1339–1364.
[13] A. Cauchy, et al., Méthode générale pour la résolution des systemes d'équations simultanées, C. R. Sci. Paris 25 (1847) 536–538.
[14] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representations by back-propagating errors, Nature 323 (1986) 533–536.
[15] H. Robbins, S. Monro, A stochastic approximation method, Ann. Math. Stat. (1951) 400–407.
[16] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, J. Mach. Learn. Res. 12 (2011).
[17] G. Hinton, N. Srivastava, K. Swersky, Neural networks for machine learning lecture 6a overview of mini-batch gradient descent, Cited on 14 (2012) 2.
[18] D.P. Kingma, J. Ba Adam, A method for stochastic optimization, arXiv preprint arXiv:1412.6980, 2014.
[19] S. Ruder, An overview of gradient descent optimization algorithms, arXiv preprint arXiv:1609.04747, 2016.
[20] X. Liu, J. Shen, X. Zhang, An efficient and robust SAV based algorithm for discrete gradient systems arising from optimizations, arXiv preprint arXiv:2301.02942, 2023.
[21] R.T. Chen, Y. Rubanova, J. Bettencourt, D.K. Duvenaud, Neural ordinary differential equations, Adv. Neural Inf. Process. Syst. 31 (2018).
[22] E. Haber, L. Ruthotto, Stable architectures for deep neural networks, Inverse Probl. 34 (2017) 014004.
[23] J. Han, Q. Li, et al., A mean-field optimal control formulation of deep learning, Res. Math. Sci. 6 (2019) 1–41.
[24] Q. Li, L. Chen, C. Tai, et al., Maximum principle based algorithms for deep learning, J. Mach. Learn. Res. 18 (2018) 1–29.
[25] Y. Lu, A. Zhong, Q. Li, B. Dong, Beyond finite layer neural networks: bridging deep architectures and numerical differential equations, in: International Conference on Machine Learning, PMLR, 2018, pp. 3276–3285.
[26] C. Ma, L. Wu, et al., Machine learning from a continuous viewpoint, I, Sci. China Math. 63 (2020) 2233–2266.
[27] E. Weinan, A proposal on machine learning via dynamical systems, Commun. Math. Stat. 1 (2017) 1–11.
[28] A. Barron, Universal approximation bounds for superpositions of a sigmoidal function, IEEE Trans. Inf. Theory 39 (1993) 930–945.
[29] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Control Signals Syst. 2 (1989) 303–314.
[30] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[31] C. Ma, L. Wu, et al., The Barron space and the flow-induced function spaces for neural network models, Constr. Approx. 55 (2022) 369–406.
[32] S. Mei, A. Montanari, P.-M. Nguyen, A mean field view of the landscape of two-layer neural networks, Proc. Natl. Acad. Sci. 115 (2018) E7665–E7671.
[33] J. Sirignano, K. Spiliopoulos, Mean field analysis of neural networks: a central limit theorem, Stoch. Process. Appl. 130 (2020) 1820–1852.
[34] J. Shen, J. Xu, J. Yang, The scalar auxiliary variable (SAV) approach for gradient flows, J. Comput. Phys. 353 (2018) 407–416.
[35] M. Jiang, Z. Zhang, J. Zhao, Improving the accuracy and consistency of the scalar auxiliary variable (SAV) method with relaxation, J. Comput. Phys. 456 (2022) 110954.
[36] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (1998) 2278–2324.