Int. J. Nonlinear Anal. Appl. In Press, 1–5
ISSN: 2008-6822 (electronic)
http://dx.doi.org/10.22075/ijnaa.2024.34512.5156

Optimization functions for neural network-based approximation of Burger's–Fisher equation: A comparative analysis

Haniye Hajinezhad[a], Abulfazl Yavari[b,*], Seyyed Mohammad Reza Hashemi[c]

[a]Department of Mathematics, Payame Noor University, Tehran, Iran
[b]Department of Computer Engineering and IT, Payame Noor University, Tehran, Iran
[c]Faculty of Mathematics, Statistics, Computer Science, Semnan University, Semnan, Iran

(Communicated by Madjid Eshaghi Gordji)

Abstract

This study explores the effectiveness of different optimization functions for approximating the solution of Burger's–Fisher equation with initial and boundary conditions based on neural networks. It compares and analyzes the performance of nine common optimization functions, emphasizing computational accuracy. Extensive experiments show that the choice of appropriate optimization function significantly influences the performance of neural network-based solvers for approximating the solution of Burger's–Fisher equation with initial and boundary conditions. The findings provide valuable insights and practical recommendations for researchers applying neural networks to solve Berger's equation in fields such as fluid dynamics and heat transfer.

Keywords: Burger's–Fisher equation, Neural networks, Optimization functions
2020 MSC: 92B20, 65K05, 35Q92

1 Introduction

Neural networks are a type of machine learning algorithm that are inspired by the structure and function of the human brain. They are used to learn patterns in data and make predictions based on those patterns. In addition to numerical methods, neural networks can be used to approximate the solutions of equations, but they differ in their approach. Numerical methods rely on mathematical models to approximate solutions, while neural networks learn the solution directly from data.

Neural networks have been used for solving differential equations for recent decades. One of the key papers in this area is the 1998 work of Lagaris et al. [9], in which they presented a technique for solving differential equations with neural networks.

Nonlinear partial differential equations are significant in the realms of physical science and engineering. One of the key advantages of using neural networks for solving differential equations is their ability to handle complex and non-linear systems, which are often challenging for traditional numerical methods [1, 5, 14, 19]. Additionally, neural

*Corresponding author
Email addresses: h.hajinezhad@pnu.ac.ir (Haniye Hajinezhad), a.yavari@pnu.ac.ir (Abulfazl Yavari), smr.hashemi@shahroodut.ac.ir (Seyyed Mohammad Reza Hashemi)

Received: June 2024 *Accepted:* August 2024

networks can learn from data and adapt to new situations, making them particularly well-suited for solving differential equations in real-world applications.

The Burger equation is a highly nonlinear equation that combines convection, diffusion, and reaction mechanisms. Here we review several notable research efforts focused on employing neural networks to solve the Burger equation. Wen and Chaolu [20] introduced a neural network approach based on Lie series within Lie groups of differential equations for solving Burgers–Huxley equations, incorporating initial or boundary value terms in the loss functions. Cheng and Zhang [3] proposed a method that integrates the Physical Informed Neural Network (PINN) with Resnet blocks to address the Navier-Stokes and Burger's equations, incorporating these equations into the loss function of the deep neural network to guide the model. Additionally, the loss function accounts for the inclusion of initial and boundary conditions. Ye et al. [21] presented the modified PINN method for time-fractional Burger-type equations, introducing locally adaptive activation functions and two effective weighting strategies to enhance solution accuracy. Lu et al. [12] introduced a residual-based adaptive refinement method to improve the training efficiency of PINNs for approximating solutions of differential equations, including Burger's equation. This paper emphasizes the significance of optimizer function selection in improving the approximation of Burger's–Fisher equation with initial and boundary conditions.

Adaptive gradient methods play a pivotal role in deep learning. Despite stochastic gradient descent (SGD) being a long-standing favourite, it grapples with challenges related to ill-conditioning and the time needed to train deep neural networks with large datasets. Consequently, more sophisticated algorithms have emerged to address the limitations of SGD. Currently, optimization algorithms for deep learning dynamically adapt their learning rates during training. Adaptive gradient methods essentially fine-tune the learning rate for each parameter, reducing it when gradients of specific parameters are large, and vice versa. Several adaptive methods have been introduced recently, emerging as predominant alternatives to SGD.

In this paper, the authors compare various optimization functions, including SGD [17], Rprop, RMSprop [16], RAdam [10], AdaGrad, Nadam [4], Adam [8], AdamW [11], and Adamax [8], for training neural networks to approximate the solution of the Burger's–Fisher equation with initial and boundary conditions. They evaluate the performance of these optimizers in terms of accuracy.

The paper's structure is as follows: Section 2 introduces the Burger's-Fisher equation with initial and boundary conditions. Section 3 describes the neural network used to approximate the solution of the Burger's-Fisher equation. Section 4 is dedicated to comparing the performance of different optimizer functions. Finally, the conclusion is presented.

2 Description of Burger's–Fisher equation

The generalized Burger's–Fisher equation has been applied in various fields including gas dynamics, number theory, heat conduction, and elasticity [22]. The generalized Burger's–Fisher equation is as follows.

$$u_t = \alpha u^{\delta} u_x - u_{xx} = \beta u(1 - u^{\delta}), \qquad a \leqslant x \leqslant b, \quad t \geq 0. \tag{2.1}$$

By assuming appropriate initial and boundary conditions, the exact solution is

$$u(x,t) = (\frac{1}{2} + \frac{1}{2}\tanh(\theta_1(x - \theta_2)))^{\delta} \tag{2.2}$$

where,

$$\theta_1 = \frac{-\alpha\delta}{2(1+\delta)}, \quad \theta_2 = \frac{\alpha}{1+\delta} + \frac{\beta(1+\delta)}{\alpha}.$$

To test the accuracy of the approximate solution obtained by the neural network, the initial and boundary conditions are determined by the exact solution. Numerous researchers have investigated this equation to approximate its solution with numerical methods [6, 7, 13, 15, 18].

3 Description of used Neural Network

The initial step in solving a differential equation using neural networks (NNs) involves reframing the problem as an optimization task, based on the definition of the loss function. Subsequently, the architecture of the neural network

needs to be designed. This entails determining the number of hidden layers, the number of neurons in each layer, the volume of training data, the selection of activation function, the number of epochs, and the selection of optimization functions and other relevant parameters. Once the architecture is established, the neural network must be trained using an appropriate dataset. During the training process, the loss function is used to measure the disparity between the network's predictions and the actual values. The network's parameters (weights and biases) are iteratively adjusted using optimization algorithms such as gradient descent to minimize the loss function. Upon completion of training, the neural network can approximate the solution to the given differential equation for unseen inputs.

In this paper, the idea of Chen and his colleague [2] is used to formulate the loss function. Our neuron network comprises an input layer with two variables x and t, an output layer with variable u(x,t), and an intermediate structure of five hidden layers, each containing 32 neurons. We employ the GELU activation function. Also, 32 points as x and 16 points as t are used for training. The number of epoches is considered 5000, in other cases, it is mentioned. A suitable optimization function should be determined to update the network parameters based on the gradients calculated through the backpropagation method.

4 Comparing the accuracy of different optimizer functions

In this section, our objective is to identify the ideal optimization function for training the proposed neural network. For this purpose, the infinity norm and one norm of approximation error is defined, respectively as follows.

$$\|E\|_\infty = Max_{1\le i\le N}|u_{ana_i} - u_{net_i}|,$$
$$\|E\|_1 = \frac{\sum_{i=1}^{N}|u_{ana_i} - u_{net_i}|}{N},$$

where u_{ana_i} and u_{net_i} represent the exact solution and neural network's approximate solution at (x_i, t_i) respectively, and the N is the total number of points used for error computation, set to $N = 10000$.

We conducted a series of experiments using the neural network described in the preceding section to explore the efficacy of nine optimization algorithms, namely SGD [17], Rprop, RMSprop [16], RAdam [10], AdaGrad, Nadam [4], Adam [8], AdamW [11], and Adamax [8]. Each algorithm represents a unique case used to approximate the solution of equation 2.1 with specific initial and boundary conditions.

In the first test, to ensure statistical reliability, each of these 9 cases was run 30 times, and the average of $\|E\|_\infty$ and $\|E\|_1$ of the resulting 30 runs were computed. Table 1 presents the average of $\|E\|_\infty$ and $\|E\|_1$ for each optimizer function, by assuming $\alpha = 1$, $\beta = 1$, and $\delta = 2$, with $x \in [-10, 20]$ and $t \in [0, 1]$. The columns representing the infinity norm of the error and the one norm of the error are arranged in ascending order. The table indicates that Adamax, AdamW, and Radam provide more accurate approximations, while RMSprop yields less accurate results.

Table 1: Average of $\|E\|_\infty$ and $\|E\|_1$ for Each Optimizer Function

Optimizer	$\|E\|_\infty$	Optimizer	$\|E\|_1$
Adamax	$1.64708244050343 \times 10^{-6}$	Adamax	$7.85898779760152 \times 10^{-6}$
AdamW	$2.52910249324498 \times 10^{-6}$	Radam	$1.46709472871244 \times 10^{-5}$
Radam	$2.82038610701581 \times 10^{-6}$	AdamW	$1.48394629333827 \times 10^{-5}$
Adagrad	$3.00686953460146 \times 10^{-6}$	Adagrad	$1.57488386856474 \times 10^{-5}$
SGD	$5.90508594510665 \times 10^{-6}$	SGD	$2.58074128027252 \times 10^{-5}$
Adam	$6.33155090038516 \times 10^{-6}$	Adam	$3.55779421131220 \times 10^{-5}$
Rprop	$8.64033769476345 \times 10^{-6}$	Rprop	$4.46275454700967 \times 10^{-5}$
Nadam	$4.44236498029124 \times 10^{-5}$	Nadam	$2.87230686660080 \times 10^{-4}$
RMSprop	$3.19363565259401 \times 10^{-2}$	RMSprop	$2.27806677748135 \times 10^{-1}$

For the second test, Tables 2 displays $\|E\|_1$ of each optimizer function by assuming $\alpha = 1$, $\beta = 1$, and $\delta = 2$, with $x \in [-10, 20]$ and $t \in [0, 1]$ and under different conditions, including 10000 and 20000 epochs. The results consistently show that Adamax, AdamW, and Radam lead to more accurate approximations, while RMSprop produces less accurate results.

The results of another test are presented in Table 3, showcasing $\|E\|_1$ of each optimizer function by assuming $\alpha = 2$, $\beta = 4$, and $\delta = 2$, with $x \in [-10, 20]$ and $t \in [0, 1]$. Once again, this table shows that Adamax, AdamW, and Radam lead to more accurate approximations, while RMSprop produces less accurate results.

Table 2: $\|E\|_1$ for Each Optimizer Function with 10000 and 20000 Epochs by assuming $\alpha = 1$, $\beta = 1$, and $\delta = 2$

Optimizer	$\|E\|_1$ (Epoche=10000)	Optimizer	$\|E\|_1$ (Epoche=20000)
Radam	$5.35030649640497 \times 10^{-6}$	Adamax	$2.35291704558418 \times 10^{-6}$
AdamW	$5.35423608295110 \times 10^{-6}$	AdamW	$2.41199352109394 \times 10^{-6}$
Adamax	$5.46882189079660 \times 10^{-6}$	Radam	$3.04817965468302 \times 10^{-6}$
Adagrad	$8.93723642925836 \times 10^{-6}$	Adagrad	$5.15871498623183 \times 10^{-6}$
Rprop	$9.93681627194604 \times 10^{-6}$	Adam	$5.83348527166089 \times 10^{-6}$
SGD	$9.96818089713675 \times 10^{-6}$	SGD	$2.64993640289539 \times 10^{-5}$
Adam	$1.34757426205124 \times 10^{-5}$	Rprop	$2.65192343763965 \times 10^{-5}$
Nadam	$1.36896577896813 \times 10^{-4}$	Nadam	$3.59266351531136 \times 10^{-5}$
RMSprop	$2.48884376692317 \times 10^{-4}$	RMSprop	$1.72365296041790 \times 10^{-4}$

Table 3: $\|E\|_1$ for Each Optimizer Function by assuming $\alpha = 2$, $\beta = 4$, and $\delta = 2$

Optimizer	$\|E\|_1$
Adamax	$7.57195157904586 \times 10^{-5}$
Radam	$3.40890863500638 \times 10^{-4}$
AdamW	$3.89393208257223 \times 10^{-4}$
Adam	$4.12877173177112 \times 10^{-4}$
Nadam	$4.35333281243569 \times 10^{-4}$
Adagrad	$4.80405060820108 \times 10^{-4}$
SGD	$1.64000379193429 \times 10^{-3}$
Rprop	$1.65260399241475 \times 10^{-3}$
RMSprop	$4.66357137269634 \times 10^{-3}$

The final test results are presented in Table 4, showcasing the $\|E\|_1$ of each optimizer function by assuming $\alpha = 3$, $\beta = 2$, and $\delta = 2$, with $x \in [-10, 20]$ and $t \in [0, 1]$. Once again, findings of table 4 show that Adamax, AdamW, and Radam lead to more accurate approximations, while RMSprop produces less accurate results.

Table 4: $\|E\|_1$ for Each Optimizer Function by assuming $\alpha = 3$, $\beta = 2$, and $\delta = 2$

Optimizer	$\|E\|_1$
Adamax	$2.09975415513011 \times 10^{-5}$
AdamW	$2.96310587707187 \times 10^{-5}$
Radam	$3.36089030061656 \times 10^{-5}$
SGD	$3.96262605974220 \times 10^{-5}$
Rprop	$4.02997607315718 \times 10^{-5}$
Adagrad	$5.41389936737727 \times 10^{-5}$
Nadam	$7.69479337684435 \times 10^{-5}$
Adam	$1.22248478864545 \times 10^{-4}$
RMSprop	$2.29861196186085 \times 10^{-3}$

In summary, the results from Tables 1-4 confirm that Adamax, AdamW, and Radam are the most suitable optimizers for approximating the Berger-Fisher equation, while RMSprop consistently yields less accurate approximations.

5 Conclusions

In conclusion, this study presented a comprehensive analysis of various optimization functions for approximating the solution of the Burger's-Fisher equation with initial and boundary conditions using neural networks. Through extensive experiments and comparisons, it was demonstrated that the choice of optimization function significantly impacts the performance of neural network-based solvers for this complex equation. The findings highlight the effectiveness of Adamax, AdamW, and Radam as the most suitable optimizers for accurately approximating the Burger's-Fisher equation with initial and boundary conditions, while RMSProp consistently yielded less accurate results. These insights provide valuable guidance for researchers applying neural networks to solve challenging equations in fluid dynamics and heat transfer fields. The study contributes to advancing neural network applications in solving differential equations.

References

[1] L.P. Aarts and P. Van der Veer, *Solving nonlinear differential equations by a neural network method*, Int. Conf. Comput. Sci., Springer, 2001.

[2] F. Chen, D. Sondak, P. Protopapas, M. Mattheakis, S. Liu, D. Agarwal, and M. Di Giovanni, *Neurodiffeq: A python package for solving differential equations with neural networks*, J. Open Source Software **5** (2020), no. 46, 1931.

[3] C. Cheng and G.-T. Zhang, *Deep learning method based on physics informed neural network with resnet block for solving fluid flow problems*, Water **13** (2021), no. 4, 423.

[4] T. Dozat, *Incorporating nesterov momentum into adam*, Workshop track-ICLR, 2016.

[5] A. Fallah and M.M. Aghdam, *Physics-informed neural network for solution of nonlinear differential equations*, Nonlinear Approaches in Engineering Application: Automotive Engineering Problems, Cham: Springer Nature Switzerland, 2024, pp. 163–178.

[6] B. Gürbüz and M. Sezer, *A modified laguerre matrix approach for burgers–fisher type nonlinear equations*,Numerical Solutions of Realistic Nonlinear Phenomena, Springer, 2020, pp. 107-123.

[7] H. Hajinezhad, *A numerical approximation for the one-dimensional burger–fisher equation*, Asian Res. J. Math. **18** (2022), no. 5, 22–30.

[8] D.P. Kingma, *Adam: a method for stochastic optimization*, arXiv preprint arXiv:1412.6980 (2014).

[9] I.E. Lagaris, A. Likas, and D.I. Fotiadis, *Artificial neural networks for solving ordinary and partial differential equations*, IEEE Trans. Neural Networks **9** (1998), no. 5, 987–1000.

[10] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han, *On the variance of the adaptive learning rate and beyond*, arXiv preprint arXiv:1908.03265 (2019).

[11] I. Loshchilov and F. Hutter, *Decoupled weight decay regularization*, arXiv preprint arXiv:1711.05101 (2017).

[12] L. Lu, X. Meng, Z. Mao, and G.E. Karniadakis *Deepxde: A deep learning library for solving differential equations*, SIAM Rev. **63** (2021), no. 1, 208–228.

[13] R.K. Mohanty and S. Sharma, *A high-resolution method based on off-step non-polynomial spline approximations for the solution of Burgers-Fisher and coupled nonlinear Burgers' equations*, Engin. Comput. **37** (2020), no. 8, 2785–2818.

[14] K. Parand, A.A. Aghaei, S. Kiani, T.I. Zadeh, and Z. Khosravi, *A neural network approach for solving nonlinear differential equations of Lane–Emden type*, Engin. Comput. **40** (2024), no. 2, 953–969.

[15] M.A. Pirdawood and Y.A. Sabawi, *High-order solution of generalized Burgers–Fisher equation using compact finite difference and dirk methods*, J. Phys.: Conf. Ser. **1999** (2021), no. 1, 012088.

[16] M. Riedmiller and H. Braun, *A direct adaptive method for faster backpropagation learning: The RPROP algorithm*, Proc. IEEE Int. Conf. Neural Networks, 1993, pp. 586—591.

[17] H. Robbins and S. Monro, *A stochastic approximation method*, Ann. Math. Statist. **25** (1951), 400–407.

[18] A. Singh, S. Dahiya, and S. Singh, *A fourth-order b-spline collocation method for nonlinear Burgers–Fisher equation*, Math. Sci. **14** (2020), no. 1, 75–85.

[19] P. Venkatachalapathy and S. Mallikarjunaiah, *A deep learning neural network framework for solving singular nonlinear ordinary differential equations*, Int. J. Appl. Comput. Math. **9** (2023), no. 5, 68.

[20] Y. Wen and T. Chaolu, *Study of Burgers–Huxley equation using neural network method*, Axioms **12** (2023), no. 5, 429.

[21] Y. Ye, X. Liu, Y. Li, H. Fan, and H. Zhang, *Deep neural network method for solving the fractional Burgers-type equations with conformable derivative*, Phys. Scripta **98** (2023), no. 6, 065214.

[22] C.G. Zhu and W.S. Kang, *Numerical solution of Burgers–Fisher equation by cubic b-spline quasi-interpolation*, Appl. Math. Comput. **216** (2010), 2679–2686.