Finding optimal hyperparameters of feedforward neural networks for solving differential equations using a genetic algorithm

C Boonthanawat and C Boonyasiriwat*

Department of Physics, Faculty of Science, Mahidol University, Ratchathewi, Bangkok 10400, Thailand

*E-mail: chaiwoot.boo@mahidol.ac.th

Abstract. In this work, feedforward neural networks are used to solve 2D Laplace equation on rectangular domains. Optimal values of weights and biases of a network are recursively computed during the network training by minimizing a least-squares loss function using data at collocation points to approximate the true solution. The performance of the network largely depends on network architecture and model capability. In this work, an optimal set of hyperparameters is searched on various values of relative error. The genetic algorithm is used to find the optimal activation function, optimization algorithm, and weight initialization. In addition, we also searched for the optimal value of the number of hidden layers for a specific value of total parameters. Numerical results show that we can successfully obtain an optimal set of hyperparameters that is consistent across many values of relative error.

1. Introduction

Many real-world problems are modeled as differential equations. Thus, a solution to a governing differential equation is crucial for a problem of interest. Numerical methods for solving differential equations include, but not limited to, finite difference, finite element, finite volume, spectral method, and neural networks. Neural network method is one of the techniques that has its merits for high dimensional problem and complex domain where most numerical methods become infeasible [1].

Lagaris *et al.* [2] proposed a novel method based on neural networks for solving boundary-value problems in rectangular domains. In their approach, a trial solution must be explicitly constructed to satisfy the boundary condition. Alternatively, Liyao *et al.* [3] proposed a method based on neural networks called mixed residual method. The method rewritten a given PDEs into first-order systems. Neural networks are then used to approximate the solution and its high-order derivatives. The mixed residual method can be used to solve problems on irregular domains with inexact boundary condition using the method proposed by Sirignano and Spiliopoulos [4].

In this work, the problem is divided into two parts. The first part is to find the optimal activation function, optimization algorithm, and weight initialization using a genetic algorithm. Genetic algorithm is a search-based optimization technique based on the principles of genetics and natural selection. It is suitable for our problem because it works with discrete variables and non-differentiable cost function, and it is easy to apply the algorithm to a wide variety of problems. It can also overcome nonlinear problems with many local optima and the results are less dependent from the initial point.

The second part is to find the optimal value of the number of hidden layers when total parameters of the network is fixed. We cannot optimize total parameters because model capability is dependent on total parameters. By increasing the value of total parameters, the network will always perform better.

2. Feedforward neural network

A fully connected feedforward neural network is one of the simplest types of neural network where all neurons in one layer are connected to all neurons in the previous layer except the input layer. From universal approximation theorem, the neural network can approximate any continuous function making it a powerful tool. To train the network, we used a collocation method to discretize the domain Ω into a set of points turning into discrete optimization problems. These sets of points are used as a training dataset. The cost function is usually defined as a quadratic function. We used a gradient based method for optimization. Backpropagation algorithm is employed to recursively calculate the gradient across network layers. The schematic representation of a fully connected feedforward neural network is shown in figure 1.

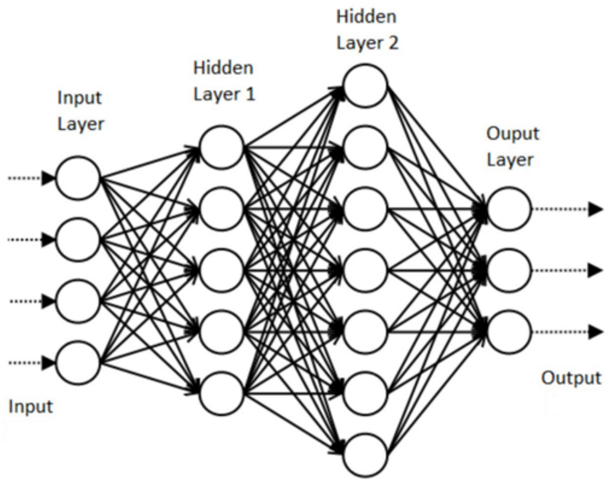

Figure 1. Schematic representation of a fully connected feedforward neural network.

3. Description of the methods

In this work, we interested in 2D Laplace equation,

$$\nabla^2 u(x,y) = 0, \qquad (x,y) \in \Omega \tag{1}$$

$$u(x,y) = g(x,y), \quad (x,y) \in \partial\Omega \tag{2}$$

when $g(x,y)$ is Dirichlet boundary condition.

Liyao *et al.* [3] proposed the mixed residual method. The method first rewritten 2D Laplace equation into first-order systems,

$$p(x,y) = \nabla u(x,y), \quad (x,y) \in \Omega \tag{3}$$

$$\nabla \cdot p(x,y) = 0, \qquad (x,y) \in \Omega \tag{4}$$

$$u(x,y) = g(x,y), \qquad (x,y) \in \partial\Omega \tag{5}$$

The method directly constrain boundary condition in the cost function. The method can be used to solve PDEs in an irregular domain. In their proposed approach, the trial solution $(\hat{u}(x,y), \hat{p}(x,y))$ employs a neural network directly. The weights and biases of the neural network are trained by minimizing a least-squares cost function using data at collocation points, $s_n = \{(x_n, y_n) \in \Omega, (\tau_n, z_n) \in \partial\Omega\}$. The cost function is defined as,

$$J(\theta) = \sum_{(x_n,y_n)} \|\hat{p}(x_n,y_n) - \nabla\hat{u}(x_n,y_n)\|^2 + \sum_{(x_n,y_n)} \|\nabla\cdot\hat{p}(x_n,y_n)\|^2 + \sum_{(\tau_n,z_n)} \|\hat{u}(\tau_n,z_n) - g(\tau_n,z_n)\|^2 \tag{6}$$

3.1. Stopping criteria and performance measurement

The stopping criteria we used is the relative error norm [5] shown in equation (7).

$$E_{norm} = \frac{\sqrt{\sum_{\vec{x}\in\Omega}(u(\vec{x}) - \psi(\vec{x};\theta))^2}}{\sqrt{\sum_{\vec{x}\in\Omega} u(\vec{x})^2}} \tag{7}$$

where $\vec{x}$ is the grid point in the domain Ω, $u(\vec{x})$ is the true solution of the differential equation, and $\psi(\vec{x};\theta)$ is the approximated solution of the neural network.

The relative error produces an aggregate error that accurately reflects relative errors of the true solution for regions of the domain with both large and small values. Once a specific value of relative error is chosen, we measure the performance of the neural network with "number of epochs". An epoch refers to one cycle of training through the full training dataset. The lower the number of epochs is, the faster the network is trained.

The main study of this work is to see how an optimal set of hyperparameters changed when we changed the value of relative error. The relative error for numerical results were chosen to be,

$$E = \{1\%, 0.5\%, 0.1\%, 0.05\%\} \tag{8}$$

3.2. Genetic algorithm

The algorithm begins by initializing a population of individuals randomly. Then, it runs each member of that population through a fitness function. Fitness function will determine how "fit" or how "good" the member is. The members are then selected through a selection algorithm to reproduce using crossover and mutation operator. The algorithm will be repeated until a desired number of iterations have passed. At termination, the algorithm presents all members of the last generation according to the fitness function.

3.2.1. Chromosome and fitness value. To prepare the neural network, we must first encode the network to an appropriate form. In this work, we used parametric representation [6]. The network is specified by a set of parameters. (e.g. activation function, optimization algorithm, and weight initialization). This type of representation is most suitable when we know what type of architectures we are trying to find. Once the network is trained to reach specific value of relative error defined in equation (8), the fitness value of the network is defined by,

$$f = \frac{1}{epochs} \tag{9}$$

3.2.2. Population initialization and selection. The network is initialized randomly to avoid a condition known as premature convergence due to the loss of diversity. Then, we sort the members in the population according to its fitness value (equation (9)) from the best to the worst member. The members are then selected through elitism. Elitism keeps a specific percentage of the best members in the population from the previous generation to the next generation. Next, the members are selected through the roulette wheel selection method in which the fitter member has proportionally higher chance for selection.

3.2.3. Crossover and mutation operator. We used single point crossover. A point between both parents' chromosomes is picked randomly, and designated a crossover point. All parameters beyond the crossover point are swapped between the two parents creating two children. Mutation is used to maintain and introduce diversity in the population. A mutation parameter is represented by percentage. Once the mutation occurs, we randomly choose one of the parameters of the child, and its new value is randomly chosen from the search space.

3.3. Problem setup
In this work, we considered 2D Laplace equation in rectangular domain,

$$\nabla^2 u(x,y) = 0, \quad (x,y) \in [0,1] \times [0,1] \tag{10}$$

We chose the boundary condition such that the analytic solution is,

$$u(x,y) = \sin(\omega y)e^{-\omega x} \tag{11}$$

For the numerical results, we set $\omega = \pi$. We used 100×100 collocation points uniformly distributed on the domain and 4000 collocation points uniformly distributed on the boundary as the training dataset. The test collocation points for calculating the relative error are 1000×1000 uniformly distributed across the domain. The batch size is 1000, and finally we used exponential decay learning rate schedules to adjust the learning rate during training. The maximum learning rate is $1e^{-2}$, the minimum is $1e^{-4}$, step size is 40, and decay parameter is 0.9998.

4. Results and discussion
4.1. The baseline results
We trained 30 neural networks with hidden layers equal to 4, neuron per layer equal to 30, Swish activation function, Adam optimization algorithm, and Glorot uniform for weight initialization. We then calculated the average number of epochs, standard deviation, and coefficient of variation. The coefficient of variation (CV) is a measure of relative variability. It is the ratio of the standard deviation to the average. It is useful when you want to compare the spread of data for different datasets. Different values of relative error are considered to be different datasets because the number of epochs is expected to be greater when the value of relative error decreases. The results are shown in table 1.

The values of CV showed that the spread of data increases when the value of relative error decreases. The high spread of data makes the number of epochs from one neural network less reliable as an indicator for the best hyperparameters. The results are more reliable when we use many data points.

Genetic algorithms can be used to solve the problem. On average, the optimal hyperparameters would have a higher chance to be selected and passed on to the next generation. As a number of generations increases, the number of parameters in the population will also increase. Other hyperparameters that on average don't perform as well will die off. In the last generation, we can show the hyperparameters with the best performance, and also show the hyperparameters that have the highest amount in the population.

Table 1. The baseline performance.

	Error (%)			
	1	0.5	0.1	0.05
Avg. epochs	111.5	200.2	1273.6	3988.2
SD	23.4	58.9	437.2	2095.4
CV (%)	21.0	29.4	34.3	52.5

4.2. Optimal hyperparameters using genetic algorithm

In this section, the hyperparameters of genetic algorithms are 10 generations, 50 populations, elitism percentage = 10%, mutation chance = 20%. The neural network has a hidden layer equal to 4, and neurons per layer equal to 30. The search space for hyperparameters are,

- Activation function : Tanh, Sigmoid, Swish, Softplus, Mish, Gelu, List
- Optimization algorithm : Adamax, Adam, Nadam, Sgd, Rmsprop, Adadelta, Adagrad
- Weight initialization algorithm : He normal, He uniform, Glorot normal, Glorot uniform, Lecun normal, Lecun uniform, Random normal, Random uniform, Identity, Orthogonal

The average and best fitness values of each generation for different values of relative error is plotted in figure 2. The best performance of the 1st, 2nd and 3rd for hyperparameters of the last generation is shown in table 2.

Table 2. Results of the best performance of the 1st, 2nd and 3rd hyperparameters.

		Error (%)			
		1	0.5	0.1	0.05
Activation function	1st	Gelu	Gelu	Gelu	Gelu
	2nd	Gelu	Gelu	List	List
	3rd	Gelu	Gelu	Gelu	Gelu
Optimization algorithm	1st	Adam	Adam	Adam	Adam
	2nd	Adam	Adam	Adam	Adam
	3rd	Adam	Adam	Adam	Adam
Weight initialization	1st	Lecun uniform	Lecun uniform	Lecun uniform	Lecun normal
	2nd	Lecun uniform	Lecun uniform	Lecun uniform	Glorot uniform
	3rd	Lecun uniform	Lecun uniform	Lecun uniform	Lecun uniform
Epochs	1st	39	56	294	608
	2nd	40	57	303	666
	3rd	40	59	305	687

Also from the numerical results, the greatest number of parameters across all values of relative error in the population of the last generation are Gelu activation function, Adam optimization algorithm, and Lecun uniform initialization.

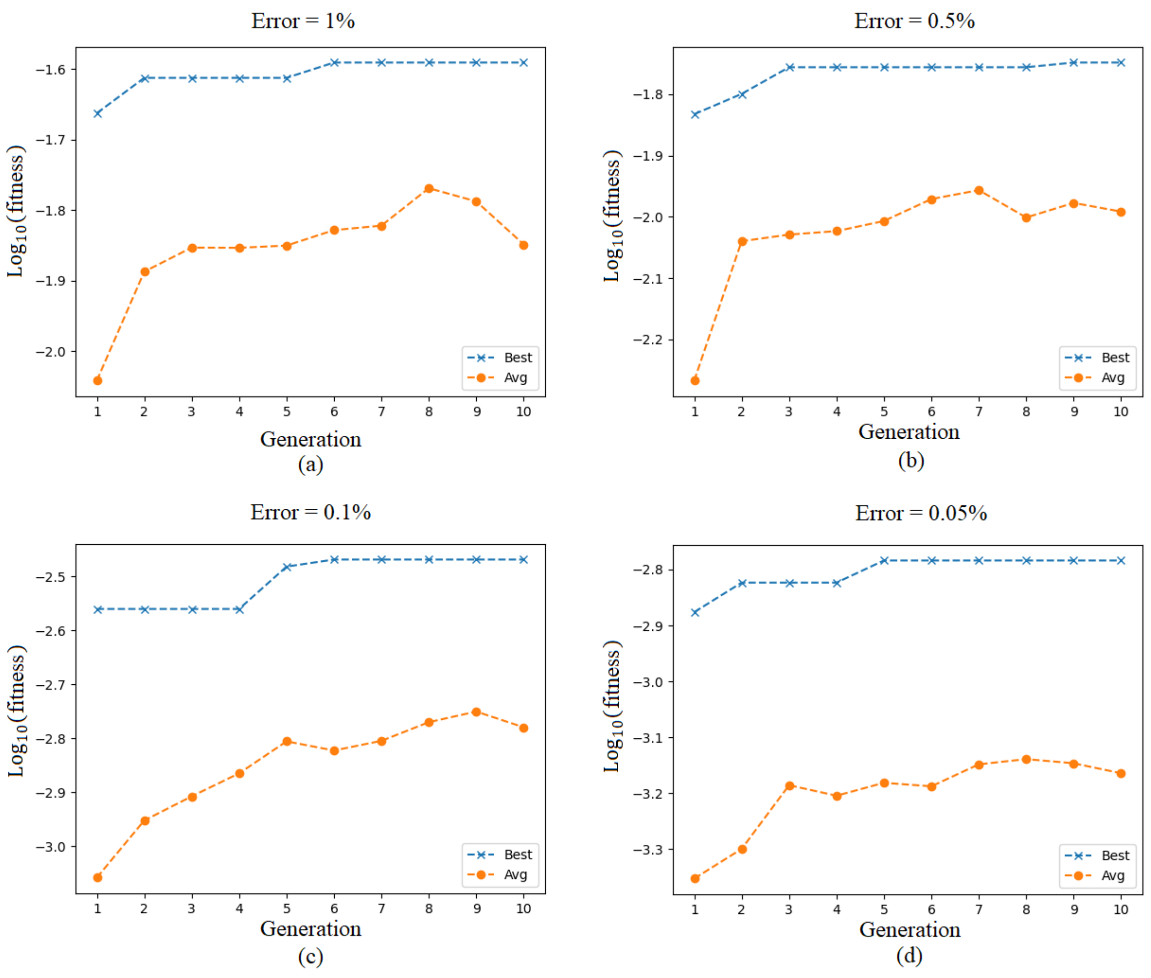

Figure 2. Relationship between fitness value and generation when relative error equal to $1\%, 0.5\%, 0.1\%, 0.05\%$ for (a), (b), (c), and (d), respectively.

Combining the best performance and the greatest number of parameters, the results show conclusive evidence for the best optimization algorithm. Adam optimization algorithm performs the best across different values of relative error.

For activation function, Gelu has the greatest number of parameters of the last generation. Gelu also is the best parameter across values of relative error. We believed the reason for Gelu activation function to be the best is due to its shape. Gelu function's shape resembles the shape of Relu function which was designed to reduce the vanishing gradient problem. One of the advantages of Gelu activation compared to Relu activation is the higher-order differentiability.

For weight initialization, Lecun uniform has the greatest number of parameters of the last generation. Lecun uniform performs consistently across different values of relative error except when relative error $= 0.05\%$. We believed the difference between the 1st, 2nd and 3rd of the weight initialization is due to the high spread of data for low value of relative error.

4.3. Optimal value of hidden layers

In this section, the neural network uses Gelu activation function, Adam optimization algorithm, and Lecun uniform initialization. We trained 30 neural networks of $2-9$ hidden layers with

$51, 36, 29, 25, 22, 20, 18, 16$ neurons in each layer, respectively. We considered total parameters of the network to be comparable, and equal to 2900 parameters. The average number of epochs and standard deviation of the hidden layer for different values of relative error is shown in figure 3. The standard deviation for each value of relative error is shown in table 3, and the bold value is the lowest standard deviation for specific relative error.

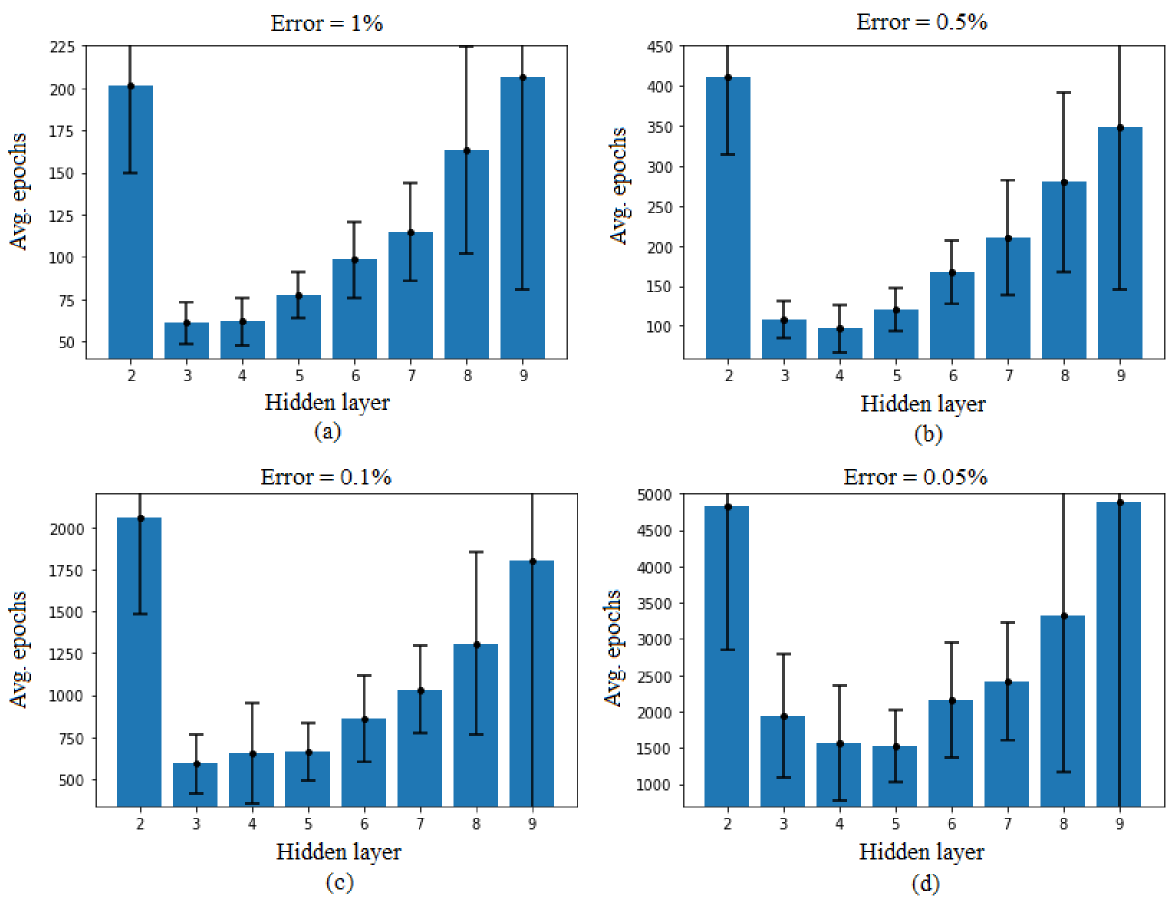

Figure 3. Relationship between average number of epochs and hidden layer when relative error equal to $1\%, 0.5\%, 0.1\%, 0.05\%$ for (a), (b), (c), and (d), respectively.

The results showed that the hidden layer with the lowest number of epochs are 3, 4, 3, 5 for relative error $= 1\%, 0.5\%, 0.1\%, 0.05\%$, respectively. For the standard deviation, the lowest are 3 and 5 hidden layers at $E = 1\%, 0.5\%$ and $E = 0.1\%, 0.05\%$, respectively. Combining the average number of epochs and standard deviation, the best hidden layer is $3 - 5$ layers. At 2 hidden layer, the network is too shallow and cannot learn the data as well as the deep network. Vanish gradient problem starts to occur at 6 hidden layer and beyond. The average number of epochs is increased substantially.

5. Conclusion

In this work, we used the mixed residual method to solve 2D Laplace equation on the rectangular domain. The neural network can be trained until a specific value of relative error is reached. The performance of the network will be determined by the number of epochs. For relative error equal to $1\%, 0.5\%, 0.1\%, 0.05\%$, Adam optimization algorithm shows the best performance

Table 3. The standard deviation for different hidden layers and relative error. The bold value is the lowest standard deviation for specific relative error.

	Hidden layer							
	2	3	4	5	6	7	8	9
SD ($E = 1\%$)	51.5	**12.1**	14.3	13.8	22.6	29.3	61.0	125.8
SD ($E = 0.5\%$)	96.7	**23.2**	29.9	27.1	39.5	72.3	112.6	204.3
SD ($E = 0.1\%$)	577.5	173.4	298.9	**168.5**	256.2	262.2	529.7	1972.1
SD ($E = 0.05\%$)	1972.1	848.0	791.6	**498.9**	796.7	807.9	2161.7	5289.9

across all settings. Gelu is the best activation function we found. We believed the shape of Gelu function and its high-order differentiability is the reason for its high performance. For weight initialization, at relative error $= 1\%, 0.5\%, 0.1\%$, Lecun uniform outperforms all other parameters. At error $= 0.05\%$, Lecun uniform is the 3rd best parameter, but still has the greatest number of parameters in the population. For a fixed total parameters, the optimal value of hidden layers are $3 - 5$ layers. The average number of epochs and standard deviation is the lowest at the range.

Acknowledgments
I would like to thank Mahidol University Center for Scientific Computing (MCSC) for financial support. I also would like to thank my friends and family for advice and encouragement.

References
[1] Berg J and Nyström K 2018 A unified deep artificial neural network approach to partial differential equations in complex geometries *Neurocomputing* **317** 28–41
[2] Lagaris I E, Likas A and Fotiadis D I 1998 Artificial neural networks for solving ordinary and partial differential equations *IEEE Trans. Neural Netw.* **9** 987–95
[3] Lyu L, Zhang Z, Chen, Chen M, and Chen J 2020 MIM: A deep mixed residual method for solving high-order partial differential equations *Preprint* abs/2006.04146
[4] Sirignano J A and Spiliopoulos K 2019 DGM: a deep learning algorithm for solving partial differential equations *J. Comput. Phys.* **375** 1339–64
[5] Mcfall K S and Mahan J R 2009 Artificial neural network method for solution of boundary value problems with exact satisfaction of arbitrary boundary conditions *IEEE Trans. Neural Netw.* **20** 1221–33
[6] Sun Y, Xue B, Zhang M and Yen G 2020 Evolving deep convolutional neural networks for image classification *IEEE Trans. Evol. Comput.* **24** 394–407