Int J Adv Eng Sci Appl Math (2021) 13(2-3):278–297
https://doi.org/10.1007/s12572-021-00306-9

On the choice of hyper-parameters of artificial neural networks for stabilized finite element schemes

Subodh M. Joshi[1] · **Thivin Anandh**[1] · **Bhanu Teja**[1] · **Sashikumaar Ganesan**[1]

Accepted: 30 July 2021 / Published online: 28 August 2021

Abstract This paper provides guidelines for an effective artificial neural networks (ANNs) design to aid stabilized finite element schemes. In particular, ANNs are used to estimate the stabilization parameter of the streamline upwind Petrov–Galerkin (SUPG) stabilization scheme for singularly perturbed problems. The effect of the artificial neural network (ANN) hyper-parameters on the accuracy of ANNs is found by performing a global sensitivity analysis. First, a Gaussian process regression metamodel of the artificial neural networks is obtained. Next, analysis of variance is performed to obtain Sobol' indices. The total-order Sobol' indices identify the hyper-parameters having the maximum effect on the accuracy of the ANNs. Furthermore, the best-performing and the worst-performing networks are identified among the candidate ANNs. Our findings are validated with the help of one-dimensional test cases in the advection-dominated flow regime. This study provides insights into hyper-parameters' effect and consequently aids in building effective ANN models for applications involving nonlinear regression, including estimation of SUPG stabilization parameters.

Keywords Artificial neural networks · SUPG · Analysis of variance · Machine learning · Singularly perturbed problems

Mathematics Subject Classification 65M60 · 62J10 · 68T99

✉ Sashikumaar Ganesan
sashi@iisc.ac.in

Subodh M. Joshi
subodhmadhav@iisc.ac.in

[1] Department of Computational and Data Sciences, Indian Institute of Science, Bangalore 560012, India

1 Introduction

Fluid flows and species concentrations are typically described by partial differential equations (PDEs) such as the Navier-Stokes equations (NS) or the convection–diffusion (CD) equation. An advection-dominated fluid flow makes the governing PDEs singularly perturbed. Solution of singularly perturbed partial differential equations (SPPDEs) requires numerical schemes that are stable and accurate. These requirements on the numerical schemes stem from the fact that the flow of information in the numerical approximation needs to closely follow the actual dynamics of the system, i.e., the 'upwinding' of the information [33]. Moreover, convection-dominated flows may contain regions of high gradients, for example, at the boundary layers or internal layers, which need to be resolved correctly to avoid spurious oscillations [29]. Accurate high-order finite element (FE) methods are being increasingly used for simulation of fluid flows [1, 52]. Such methods, however, require a sophisticated and robust numerical stabilization. There exist several strategies for improving the stability of the finite element methods to convection-dominated problems. In this paper, we focus on the streamline upwind Petrov–Galerkin (SUPG) numerical scheme, which is a popular stabilized finite element method for the solution of SPPDEs [8, 28, 36].

Various researchers have recently proposed using artificial neural networks (ANNs) to stabilize higher-order finite element schemes. For example, Veiga and Abgrall present a method for parameter-free stabilization of higher-order finite element schemes using multilayer perceptrons

[50]. Schwander et al. use ANNs for estimating local solution regularity for the addition of artificial dissipation in the context of Fourier Spectral methods [43]. Discacciati et. al. propose using ANNs for inducing artificial dissipation at the appropriate places in Runge–Kutta discontinuous Galerkin (RKDG) methods [14]. Similarly, ANNs have successfully been used for identifying 'troubled cells' to add the artificial dissipation or local slope limiting [37, 38]. A review of the ANN-based methods in CFD has been presented in [35]. The majority of the studies reviewed here follow a supervised learning approach through offline training of the ANN. It is observed that good results can be obtained by using a simple feedforward neural network with one or more *hidden* layers. However, the exact architecture of the neural networks, as well as the hyper-parameters such as the activation functions used for different layers, are case-specific. Moreover, effective implementation of ANNs for scientific computing require careful selection of ANN hyper-parameters. Although there are some commonly followed guidelines for the same, there seems to be no set prescription, and researchers seem to follow a heuristic approach.

This paper focuses on a systematic analysis of the relative sensitivity of hyper-parameters and network architectures on the learning accuracy of the ANNs as characterized by certain performance metrics. In particular, a Gaussian process regression (GPR) metamodel of the neural networks is used for performing analysis of variance (ANOVA). The second-order (SO) and total-order (TO) Sobol' indices are found out. To the best of our knowledge, no studies exist regarding the global sensitivity analysis of ANN hyper-parameters for application to nonlinear regression though similar studies exist for classification problems. Based on the insights obtained through this analysis, the optimal network architecture for application to learning of the SUPG stability parameter is identified. Lastly, the efficacy of our approach is demonstrated using numerical examples.

The rest of the paper is arranged as follows: Sect. 2 presents a review of the SUPG stabilization method as well as a one-dimensional formulation for the same. Section 3 presents a review of artificial neural networks along with SUPG stabilization using ANNs. We also identify the best and the worst-performing neural networks for nonlinear regression among the candidate ANNs. In Sect. 4, we present a variance-based sensitivity analysis of network hyper-parameters. Numerical results for one-dimensional benchmark problems are presented in Sect. 5. Finally, some observations of our study and conclusions are presented in Sect. 6.

2 Review of stabilized finite element schemes

It is well-known that the standard Galerkin finite element discretization induces spurious oscillations in the numerical solution of the convection-dominated equation. To suppress the spurious oscillations and to enhance the stability of the numerical scheme, the standard Galerkin schemes are augmented with stabilization methods. The stabilization mainly depends on the convection term in the equation. Streamline Upwind Petrov–Galerkin (SUPG) is one of the popular stabilization methods for convection-dominated problems, see for example [3, 15, 44, 51] and references therein. Other popular stabilization methods such as Galerkin least-squares [49], edge stabilization [16], continuous interior penalty [17], local projection stabilization [45], orthogonal subgrid scale [39] have also been proposed in the literature for convection-dominated problems, see [25] for an overview. Application of the SUPG method to time-dependent as well as stationary convection-dominated flows has been well-studied. For example, Bochev et al. show that the SUPG scheme with the implicit time-stepping can be used for convection-dominated flows without any restrictions on the Peclet or Courant numbers [7]. Behzadi et al. present a semi-discrete SUPG method for contaminant transport in shallow-water models [6]. An adaptive SUPG scheme for time-dependent convection–diffusion–reaction (CDR) equations is proposed by DeFrutos et. al. [20]. A variant of the SUPG method in Arbitrary Lagrangian–Eulerian (ALE) framework for the time-dependent convection–diffusion problem on time-dependent domains has been developed by Ganesan and Srivastava [22]. The error analysis of the SUPG scheme with regards to its stability for the time-dependent CDR equation has been presented by John and Novo [30]. Extension of the SUPG method for linear hyperbolic problems has been carried out in [31]. A SUPG reduced order model (ROM) for linear convection–diffusion–reaction equation has been proposed by Giere et al. [23]. Similarly, a Proper Orthogonal Decomposition (POD)-based ROM with SUPG scheme for advection-dominated flows has been proposed by Li et al., [34]. In summary, the SUPG method can be used for accurate numerical simulation of convection-dominated flows. The SUPG or any of the methods mentioned above require a well-tuned stabilization parameter. Although an analytical expression for the same exists for simple cases (e.g., One-dimensional (1D) CD equation), a closed-form expression for the stabilization parameter does not exist for a general case.

The basic idea of the streamline upwind method is to add diffusion (or viscosity), which acts only in the flow direction. Extended to a Petrov–Galerkin formulation, the standard Galerkin weighting functions are modified by

adding a streamlined upwind perturbation, which again acts only in the flow direction. The modified weighting function is applied to all terms in the equation, resulting in a consistent weighted residual formulation [3]. We consider a one-dimensional convection–diffusion equation for the present work. The one-dimensional SUPG formulation for the convection–diffusion equation is given as follows.

2.1 One-dimensional model problem

The one-dimensional convection–diffusion equation on a bounded domain $\Omega \subset \mathbb{R}$ with boundary $\partial\Omega$ is given as follows:

$$-\epsilon\phi'' + b\phi' = f \text{ in } \Omega \; \phi = \phi_b \text{ on } \partial\Omega \tag{1}$$

Here, $\epsilon > 0$ is the constant diffusion coefficient, b, f and ϕ_b are known functions and ϕ is the unknown scalar quantity, e.g., species concentration or temperature.

2.1.1 Weak form of the model problem

Let $H^1(\Omega)$ and $L_2(\Omega)$ be the standard Sobolev spaces, and $(\cdot,\cdot)$ is the inner product in $L_2(\Omega)$. Upon multiplying equation (1) by a function $v \in V := H_0^1(\Omega)$, integrating over Ω, and applying integration by parts to the higher order derivative term, the weak form of the model problem (1) reads:

Find $\phi \in H^1(\Omega)$ such that

$$\epsilon(\phi', v') + (b\phi', v) = (f, v), \quad \forall v \in H_0^1(\Omega). \tag{2}$$

As discussed early, the standard Galerkin discretization (finite dimensional counterpart of weak form) induces spurious oscillations in the finite element solution for $\epsilon << b$, unless the finite element mesh resolves all the scales in the solution.

2.1.2 Discrete form augmented with SUPG stabilization

Let $\mathcal{T}_h$ be an admissible decomposition of the domain Ω and let K be a cell in $\mathcal{T}_h$. Let V_h be a finite dimensional subspace of V, whose trace approximates ϕ_b. The streamlined upwind Petrov–Galerkin stabilized discrete form reads:

$$\begin{aligned} &\text{Find } \phi_h \in V_h \text{ such that } \phi_h - \phi_{bh} \in H_0^1(\Omega) \text{ and} \\ &-\epsilon(\phi_h', v_h') + (b\phi_h', v_h) + (R_h(\phi_h), \tau b v_h') = (f, v_h), \\ &\forall v_h \in H_0^1(\Omega). \end{aligned} \tag{3}$$

Here, $R_h(\phi) = -\epsilon\phi'' + b\phi' - f$ is the residual (defined element-wise) and $\tau \in \mathbb{R}^+$ is a non-negative stabilization parameter on any element $K \in \mathcal{T}_h$. Often the expression

$$\tau|_K = \frac{h_K}{2|b|}\left(\coth(\mathrm{Pe}_K) - \frac{1}{\mathrm{Pe}_K}\right) \text{with } \mathrm{Pe}_K = \frac{|b|h_K}{2\epsilon} \tag{4}$$

is used [25] for 1D problems. Here, h_K is the diameter of the cell K, $|b|$ is the Euclidean norm of b and Pe_K is the local Peclet number. We refer to [25] for more detailed discussion on the choice of this expression. Note that, generally, the parameters h_K, Pe_K, and $\tau|_K$ are functions of $x \in K$. In the present work, we assume that these parameters are independent of h_K.

3 SUPG stabilization using artificial neural networks

3.1 Review of artificial neural networks

The artificial neural network (ANN) technique is set to transform the mainstream scientific computing and is increasingly being used for augmenting the traditional numerical methods to improve their robustness. Artificial neural networks are networks of artificial neurons, which in turn are mathematical functions mimicking signal processing in biological neurons. It is well-established that any measurable function can be approximated using an artificial neural network with one or more hidden layers [4, 12, 26]. Additionally, once trained, the ANNs are very efficient as their application mainly involves a series of matrix-vector products which can be computed efficiently. Hence, ANNs are finding an ever-increasing application in computational science [2, 9, 13]. Although ANNs can have a variety of architectures, in this study, we limit our scope to *Feedforward* ANNs (FF-ANNs).

In FF-ANNs, layers of artificial neurons are arranged progressively such that each neuron of any layer is *connected* to every neuron in adjacent layers. This architecture is also called a multilayer perceptron (MLP). The data are perceived via the *input* layer, and the network output is obtained via the *output* layer. The layers which lie between the input and the output layers are known as *hidden* layers. Networks with more than one hidden layer are called as deep neural networks (DNNs). Figure 1a shows a schematic of an ANN for which the input layer has three neurons, the output layer has two neurons, and the two hidden layers have four neurons each. The connections between the neurons have *weights* attached to them. Thus, the inter-neuron signals can be amplified or subdued by tweaking these weights. A neuron in a separate layer receives signals from all the neurons from the previous parent layer and gets *activated* if the weighted sum of all the input signals exceeds a threshold value.

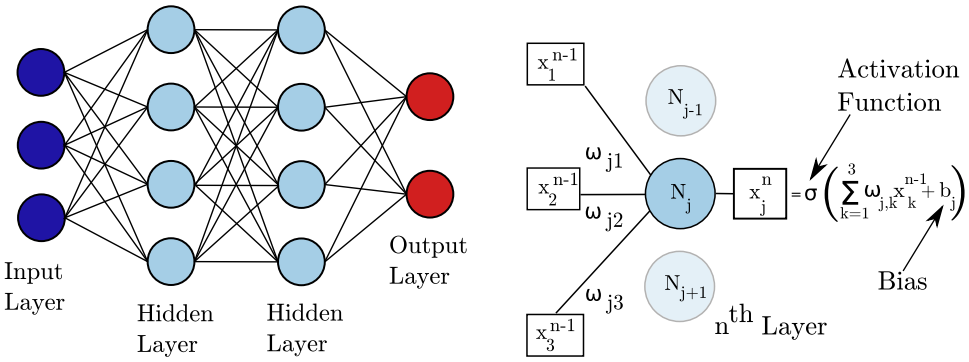


Fig. 1 Representation of an artificial neural network and computations in a single artificial neuron

(a) Schematic of an ANN (b) Computations in a single neuron

Let N_j^n indicate the jth neuron of nth layer in a MLP. The weighted sum of all the input signals from the $(n-1)$th layer is stored in the neuron. This is also called as *propagation function* [37]. Mathematically,

$$f_{prop}(x_1^{n-1}, x_2^{n-1}, \ldots, x_M^{n-1}, \omega_{j,1}^{n-1}, \omega_{j,2}^{n-1}, \ldots, \omega_{j,M}^{n-1}) = \sum_{k=1}^{M} \omega_{j,k}^{n-1} x_k^{n-1}, \tag{5}$$

where M is the total number of neurons in the $(n-1)$th layer. The threshold value is indicated by *bias* or $-b_j$ (often taken as the negative of the threshold [37]). Once the propagation function crosses the bias, ideally the neuron will get triggered, indicating a 'high' output. Else it will give a 'low' output. This ideal process can be modeled using a step function as follows.

$$\sigma(y) = \begin{cases} 1, & \text{if} \quad y > b_j \\ 0, & \text{if} \quad y < b_j \end{cases} \tag{6}$$

In practice, the step function is not used as the *activation function* since it is a discontinuous function [37]. Instead, the following continuous functions are widely used as activation functions.

1. Sigmoid function

$$f(y) = \frac{1}{1+e^{-vy}}, \quad v \in \mathbb{R}$$

2. Rectified Linear Unit (ReLU)

$$f(y) = \max(0, y)$$

3. Leaky-ReLU

$$f(y) = \max(0, y) - v\max(0, -y)$$

4. Hyperbolic tangent or TanH function

$$f(y) = \frac{e^{vy} - e^{-vy}}{e^{vy} + e^{-vy}}, \quad v \in \mathbb{R}$$

In each of these cases, the parameter v controls the *spread* of the activation function. Often the Leaky-ReLU is preferred over the ReLU function because the ReLU activation function may render some of the neurons ineffective during the training process [37]. Figure 2 shows the plots of the activation functions used in this study viz. sigmoid, Leaky-ReLU and TanH functions. Additionally, we consider the *identity* activation function $\sigma_I(x) = x$.The output of a neuron N_j^n is mathematically computed as follows:

$$x_j^n = \sigma\left(\sum_{k=1}^{M} \omega_{j,k}^{n-1} x_k^{n-1} + b_j\right) \tag{7}$$

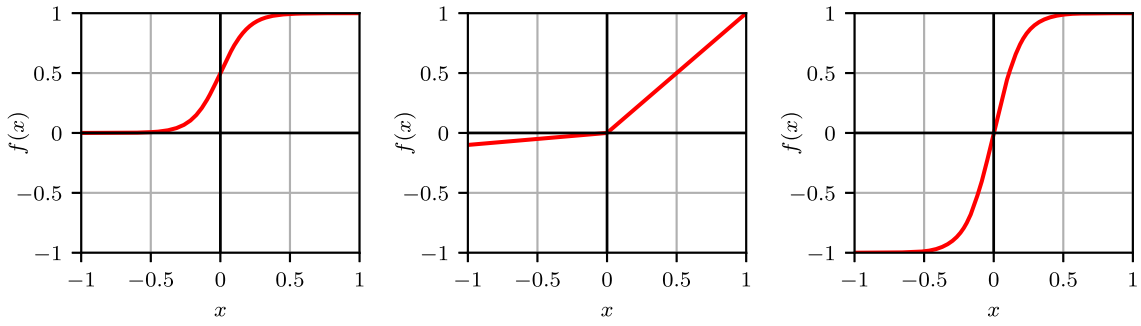
Here, $\sigma(\cdot)$ is a suitable activation function as illustrated in Fig. 1b. Additionally, a *dropout* layer is constructed before the output layer and it acts as a regularizer. Since the tuning of the regularization parameter is quite well-documented, we do not include it as one of the hyper-parameters in the present study. We refer to [10, 11] for the details of the *dropout* layer.

3.1.1 Training of ANNs

Consider a MLP network with an input (IP) layer having the dimension of $IP-D$. Also, consider the output (OP) layer with the dimension $OP-D$. Suppose this network has L number of layers and input data consisting of M samples given by $y^{(i)} \in \mathbb{R}^{IP-D}, i = 1, 2, 3, \ldots, M$. Finally, the matrix $\mathbf{A}^{n-1}$ is a matrix containing the weights $\omega_{i,j}^{n-1}$ for connections between N_j^{n-1} and N_i^n such that $\left(\mathbf{A}^{n-1}\right)_{i,j} = \omega_{i,j}^{n-1}$ and $\mathbf{b}_n$ be a vector containing bias for all the neurons in the n^{th} layer.

Definition 1 (Hypothesis ($h_\Theta(\mathbf{y})$)) The hypothesis is the prediction given by the ANN corresponding to the input data subject to the network parameters.

To train the neural network, we need a metric to indicate the accuracy of the network. *Learning* the stabilization parameter τ in a supervised setting is a *regression* problem. For such problems, a natural choice of the cost function is



(a) Sigmoid Function (b) Leaky-ReLU Function (c) TanH Function

Fig. 2 The used activation functions of ANNs in this study

the *Mean Squared Error (MSE)*. The MSE is defined as follows:

Definition 2 (Cost function) The cost function $J(\Theta)$ mean squared error is defined as:

$$J(\Theta) = \frac{1}{2M}\sum_{i=1}^{M}\sum_{j=1}^{OP-D}\left(h_\Theta(y^{(i)})_j - Y_j^{(i)}\right)^2,$$

where $Y_j^{(i)}$ is the reference value for jth output neuron corresponding to ith training example.

The hypothesis computation from the input data, also known as the *Forward Propagation*, is given in Algorithm 1 presented in Appendix A. The training of FF-ANNs involves computing the cost function followed by the *Backpropagation* of the error as described in [24]. In the *Backpropagation* algorithm, the error is propagated from the output layer up to the first hidden layer. This error is then used for computation of the correction term which is basically the derivative of the cost function with respect to the weights and the biases. This term is in turn used for reduction of the cost function via a gradient descent based optimization algorithm. See Algorithm 2 given in Appendix A for the work flow for training of ANNs.

3.2 Application of an ANN for SUPG stabilization

3.2.1 Training data

The ANN uses three features such as the advection velocity b, the dissipation coefficient ϵ, and the cell size h_k to predict the SUPG stabilization parameter. The distribution of the parameters are given in Table 1. Thus, using the analytical formula for the stabilization parameter for the $1D$ problems, a dataset containing 900 examples is generated uniformly, covering the feature value range. However, the training is performed on a range of training data size (TDS) values $6, 12, 25, 50, 100, 200, 400and800$. The samples in the training data for a corresponding TDS are randomly chosen from the large dataset. When the training and testing are repeated (for the reasons explained in the next section), the training and testing samples are again randomly chosen from the primary dataset. For training the network, the MSE is used as the cost function.

3.2.2 Network architectures

There exist a number of possible ANN configurations for learning the SUPG stabilization parameter. A large number of unique ANNs can be obtained by varying the network hyper-parameters such as the number of hidden layers, the number of neurons in each hidden layer, the activation function on the hidden and the output layers, etc. Table 2 shows the values of the hyper-parameters used for constructing neural networks in this study.

Each artificial neural network is assigned an index (ID). For each one of these networks, the number of neurons in the input layer (i.e., $IP - D$) remains three and the number of neurons in the output layer (i.e., $OP - D$) remains one. The hidden layer (HL) is numbered as 1, 2 and 3 from left to right (i.e., input to output layers). Dimension of n^{th} hidden layer is indicated by $HLn - D$ (e.g., for the second hidden layer, the dimension is given by $HL2 - D$). Let the activation function (AF) for the n^{th} hidden layer be indicated by $HLn - A$ and that for the output layer by $OP - A$. As we can see, the ANN IDs range from 1 to 7536.

3.2.3 Performance metrics

Testing is performed with fifty randomly selected samples from the main dataset. The value of the SUPG stabilization parameter estimated by the ANN is written as τ^{ANN}, whereas the reference value computed using the analytical

Table 1 Distribution of Training dataset parameters

Distribution of training data parameters		
S.No	Parameter	Value
1	Diffusion (ϵ)	1e-11 to 1e0
2	Convection (**b**)	1 to 5
3	Cell Size (h)	1e-5 to 1e-1

formula is written as τ^{ref}. The following four metrics are used for assessing the performance of the ANN.

1. **Relative L_1 Error** computed as

$$L_1\text{Error} := \frac{\sum_{i=1}^{N_s} |\tau_i^{ANN} - \tau_i^{ref}|}{\sum_{i=1}^{N_s} |\tau_i^{ref}|},$$

2. **Mean Squared Error (MS Error)** computed as

$$\text{MS Error} := \frac{1}{N_s}\left(\sum_{i=1}^{N_s}\left(\tau_i^{ANN} - \tau_i^{ref}\right)^2\right)$$

3. **Relative Minimum Error (Min Error)** computed as

$$\text{Min Error} := \min_i\left(\frac{|\tau_i^{ANN} - \tau_i^{ref}|}{|\tau_i^{ref}|}\right), \quad i \in [1, 2, 3, \ldots, N_s]$$

4. **Relative Maximum Error (Max Error)** computed as

$$\text{Max Error} := \max_i\left(\frac{|\tau_i^{ANN} - \tau_i^{ref}|}{|\tau_i^{ref}|}\right), \quad i \in [1, 2, 3, \ldots, N_s]$$

In this study, we use $N_s = 50$.

3.2.4 Implementation details

The in-house code ParMooN [21, 53] is used for computations of the SUPG solutions of the convection–diffusion (CD) equation. The ANNs are implemented in ParMooN using MLPACK, a fast and flexible C++ Machine-Learning (ML) library [10, 11]. Python is used for the data management, plotting, parallelization and automation of numerical experiments.

3.2.5 Accuracy of ANNs

In the context of the training of ANNs, we define *an experiment* as a process of start to finish training of each ANN using a range of training-dataset-sizes (TDS). Algorithm 3 in Appendix A presents the work flow for *an experiment*.

Figure 3 shows the errors corresponding to all 7536 networks on the four performance metrics for TDS 200. On the independent axis, we plot the network IDs (ANN ID), and on the y-axis, the corresponding metric. Each marker on the plot indicates the performance of an ANN with the respective ANN ID. Similar plots can be obtained for the other values of the training data (i.e., 6, 12,..., 800). Each of the sub-plots in the figure also features indicator lines for the 5-percentile ($p5$) corresponding to the smaller value and 95-percentile ($p95$) corresponding to the larger value. The zones containing ANNs with errors less than $p5$ value as well as greater than $p95$ value are shaded. Additionally, the networks featuring one hidden layer are highlighted in red triangles, two hidden layers in green squares, and three hidden layers in blue circles.

It is observed that the random nature of training and testing data causes the results to change when the experiment is repeated. This necessitates achieving the state of statistical invariance of the results to proceed with further analysis. Repeating the experiment, a sufficiently large number of times reduces the error margin [?]. Figure 4 shows the statistical averages of errors for all 7536 networks on the four performance metrics along with the 95% confidence intervals computed using the 50 experiment instances. It can be seen that the confidence intervals are small with respect to the mean value and thus the statistical average represents the true mean reasonably well. For further analysis, we continue using the statistical mean values computed using 50 experiments. It is noted that the errors vary for different networks, and some networks

Table 2 Permissible values of network hyper-parameters for this study

Values of ANN Hyper-parameters		
Number	Parameter Name	Permissible Values
1	Number of hidden layers	1,2,3
2	Neurons per hidden layer	5,10,15
3	Activation functions for the hidden layers as well as the output layer	Sigmoid (AF-1) Identity (AF-2) Leaky-ReLU (AF-3) TanH (AF-4)
Number of possible unique networks with one HL: 48 Number of possible unique networks with two HL: 576 Number of possible unique networks with three HL: 6912 **Total number of possible networks: 7536**		

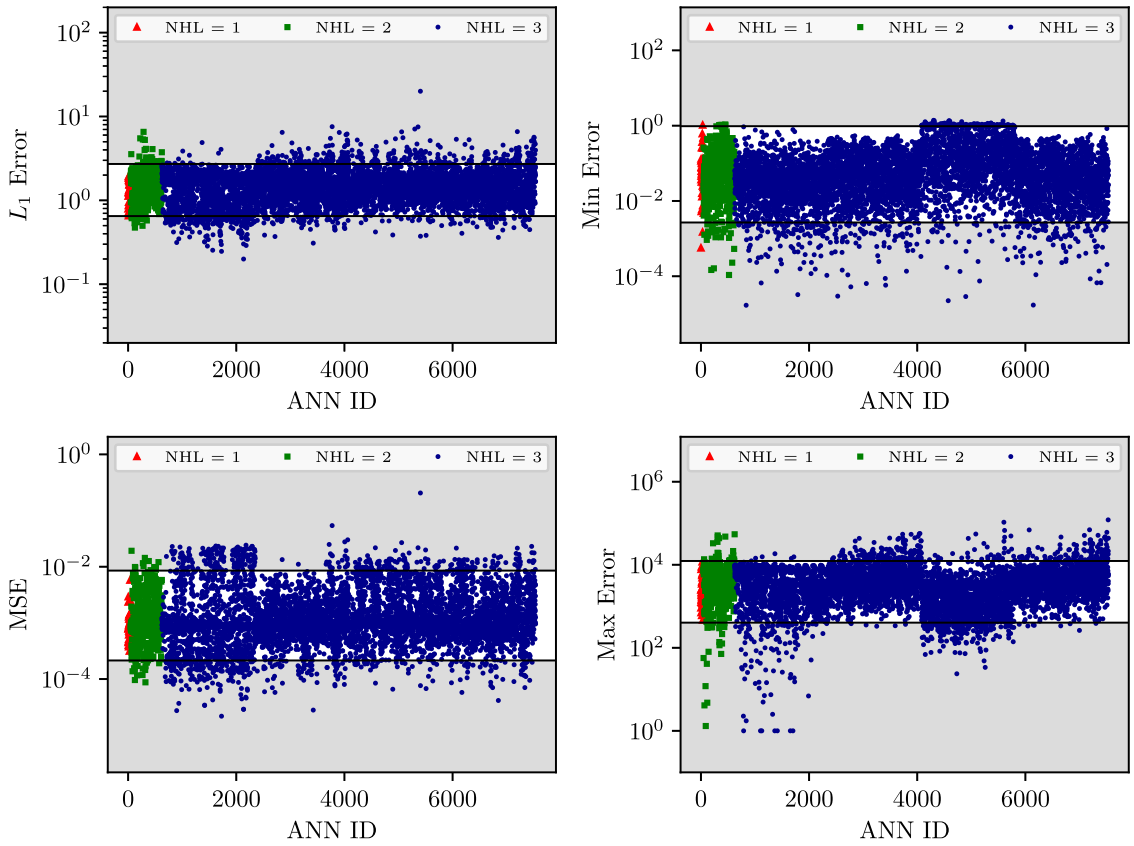


Fig. 3 Results for an experiment for all the candidate networks for the Training Dataset Size 200

consistently perform well across the experiments. On the other hand, some networks consistently result in a large testing error. Also, the range of values for the relative maximum error (Max Error) along with the 95% confidence intervals are relatively wide. This is logical as the maximum error for each ANN ID can be several orders of magnitude larger than the mean error and will change with each experiment owing to the random data used for training and testing. The longer tails of the 95% intervals (lower values) are simply due to the plotting on the log scale.

In order to observe the effect of the TDS on the accuracy of the network, we need metrics that yield a collective measure of all the candidate networks. We refer to these as aggregate measures. Here, we consider the following four aggregate measures:

1. Algebraic mean value for all the networks
2. 5-percentile value denoted by $p5$
3. 95-percentile value denoted by $p95$
4. Standard deviation for all the networks indicating the spread of the data

Moreover, these four aggregate metrics can be defined for all the network performance metrics viz. L_1 Error, Min Error, Max Error, and MS Error. Figure 5 shows the variation of the aggregate measures over a range of TDS values. As expected, the errors reduce with an increase in the training dataset sizes. However, the reduction diminishes after TDS 200. Additionally, it does not come as a surprise that the aggregate MS error is overall the lowest as compared to the other metrics since the *Backpropagation Algorithm* uses MS error as the objective function for the optimization.

To identify better performing networks on the prescribed performance metrics, we select all networks with the mean error value (for a respective metric) below the $p5$ value. Then, we select those networks which feature in the lists of the most accurate networks corresponding to L_1 Error, MS Error, and Min Error simultaneously. Out of the total 7536 ANNs, no network with just one hidden layer appears in this selection. Two networks with two hidden layers each and thirty three networks with three hidden layers each appear in the list, i.e., these networks have L_1 Error, MS Error, and Min Error less than the $p5$ threshold of the respective measure on average over the

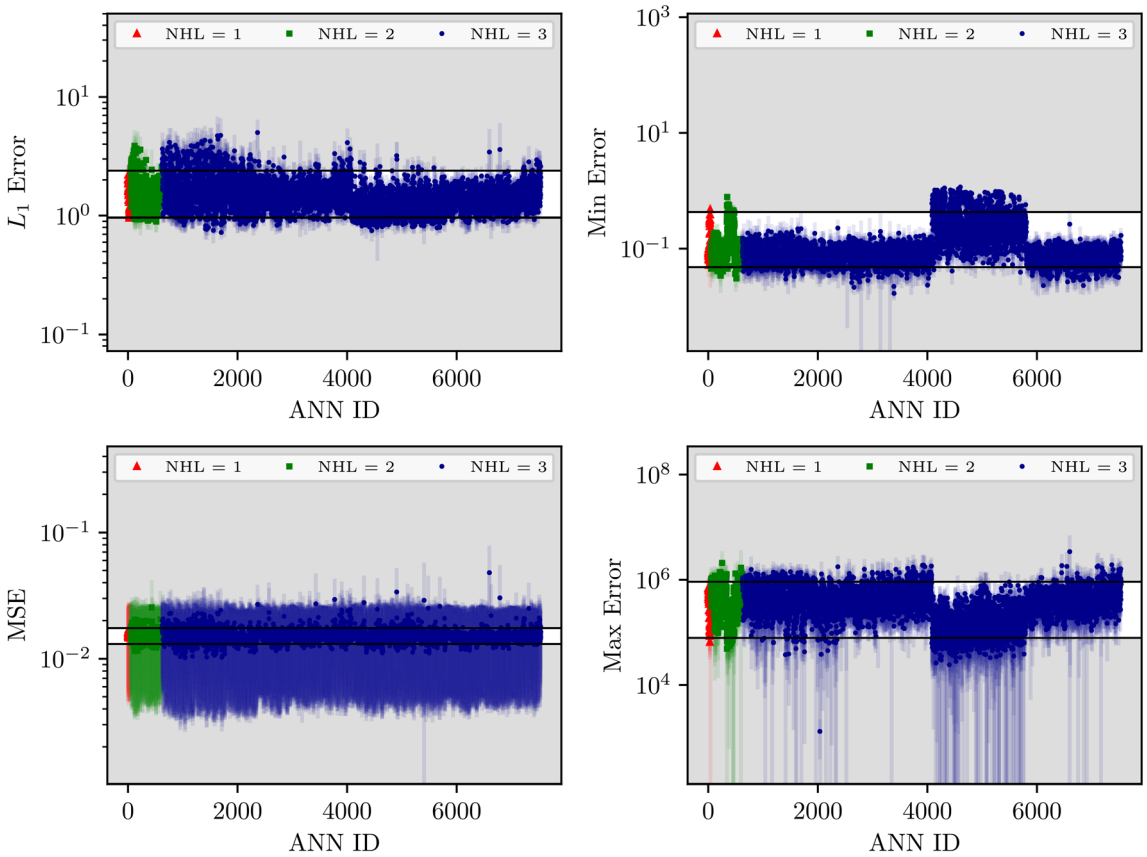


Fig. 4 Statistical average of errors for 50 instances of the experiment with 95% confidence intervals for the Training Dataset Size 200

fifty experiments. Next, we plot a frequency distribution of the top-performing networks with respect to their hyper-parameters. This reveals some interesting findings. Figure 6 shows the frequency distribution. We can summarize our observations as follows.

1. Mainly networks with three hidden layers feature in the group of the best-performing networks.
2. The majority of the top-performing networks have the sigmoid function on the output layer followed by the Leaky-ReLU function for the rest of the networks.
3. The networks seem to follow a trend of having a large central hidden layer and relatively smaller hidden layers closer to the IP and OP layers.
4. The hidden layer closer to the output layer prefers having an identity function or the leaky ReLU function for the activation function.

Table 3 shows the building blocks of the overall best-performing and worst-performing ANNs based on MS Error for TDS 100 and 400. In both cases (TDS 100 and 400), the best-performing ANNs feature the sigmoid activation function at the output layer. The worst-performing network (WPN) for TDS 100 features an identity activation function on all the hidden layers. This is equivalent to having no activation and naturally results in the poor performance. Figure 7 shows the comparison of the reference stability parameter τ^{ref} and its ANN prediction for the overall best-performing and worst-performing ANNs when measured according to MSE metric. Data corresponding to three randomly selected experiments are shown in the figure. The figure also shows correlation coefficients (Corr) between the reference and the predicted value of the stabilization parameter for 50 testing samples. The overall best-performing network shows a statistically significant correlation between the two values indicating that the network has learned to identify the correct stabilization parameter. The correlation coefficient in the case of the other network reflects its poor performance. Moreover, some of the data points are plotted in gray color. These values are actually negative, and only the magnitude is plotted in the figure due to the log-log scale.

To understand the effect of different hyper-parameters on network accuracies, we propose performing a global sensitivity analysis. To perform this analysis, we will use the statistical mean values for all the performance metrics

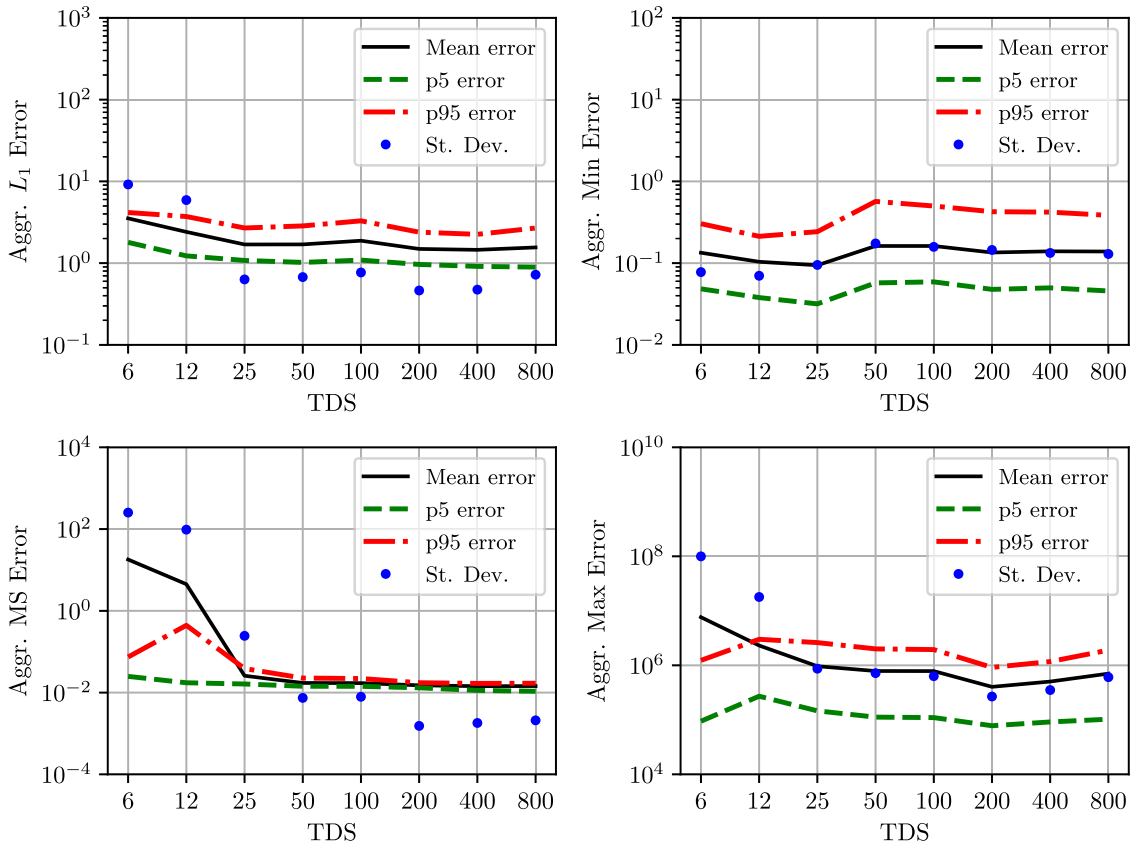


Fig. 5 Variation of the aggregate error with respect to the training dataset size

for all the ANN candidates. However, the available data may not be regular enough to compute the relative sensitivities. To address this issue, we first obtain a metamodel of the ANNs. Subsequently, we perform the analysis of variance (ANOVA) on this metamodel. In the next section, we present the results for variance-based sensitivity analysis using Sobol' indices.

4 Variance-based sensitivity analysis of MLP hyper-parameters

The global sensitivity analysis based on analysis of variance (ANOVA) was first proposed by Sobol' [46, 47]. Saltelli proposed a more economical way to compute the Sobol indices [40]. Later, Sobol et al. proposed an improved, more efficient algorithm for computation of sensitivity indices [48]. We direct the readers to the review of existing global sensitivity analysis methods, including the ANOVA technique, presented by Iooss and Lemaître and the references therein for further details [27].

Several researchers have used the global sensitivity analysis approach including ANOVA for analyzing the uncertainty in ANN surrogate models. Kowalski and Kusy have used the Sobol' indices method for determining the significance of features in a probabilistic network for classification tasks [32]. Schroder et al. have used the global sensitivity analysis based on the Sobol' indices to analyze uncertainty propagation in a neural network-based surrogate model of a wind turbine load [41]. In the current work, we analyze the ANN hyper-parameters by performing analysis of variance (ANOVA) and comparing total-order (TO) Sobol' indices. A similar approach has been followed in [19], where Extended Fourier Amplitude Sensitivity Test (EFAST) is used instead of ANOVA for a classification model. However, EFAST cannot compute the interaction between the input parameters [18]. Thus, Fernández-Navarro et al. [18] have proposed the use of ANOVA instead. An essential difference between these studies and our current work is that we use a feedforward

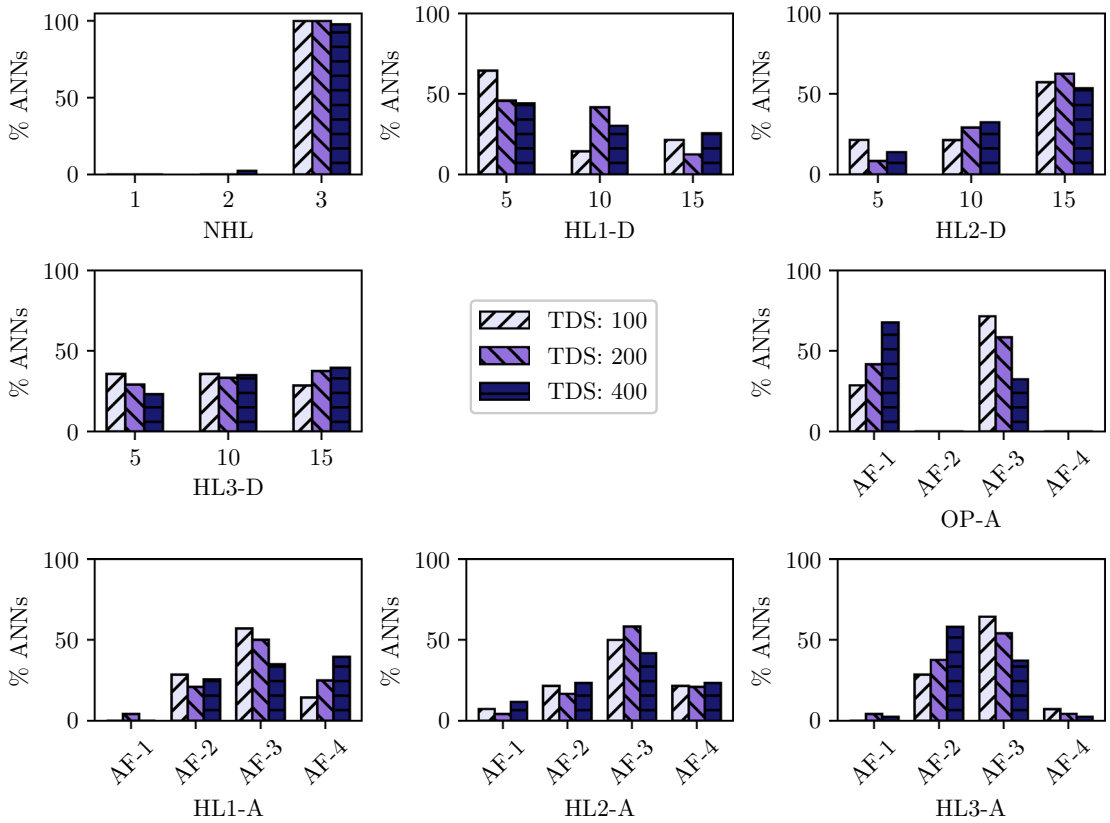


Fig. 6 Frequency distribution of hyper-parameters for top-performing networks in all three metrics for TDS 400

Table 3 Hyper-parameters corresponding to the best-performing and the worst-performing ANNs

Hyper-parameters of the overall Best/Worst ANNs				
	TDS 100		TDS 400	
Performance-	Min MSE	Max MSE	Min MSE	Max MSE
NHL	3	3	3	3
$OP-A$	Sigmoid	TanH	Sigmoid	Leaky-ReLU
$HL1-D$	5	10	10	5
$HL1-A$	Leaky-ReLU	Identity	Identity	Identity
$HL2-D$	15	10	10	10
$HL2-A$	Identity	Identity	TanH	Leaky-ReLU
$HL3-D$	5	15	15	10
$HL3-A$	Identity	Identity	Leaky-ReLU	Identity

ANN for nonlinear regression for the application of SUPG stabilization and not for classification. To our knowledge, no studies exist regarding the global sensitivity analysis of ANN hyper-parameters for application to nonlinear regression.

4.1 Analysis of variance and Sobol' indices

Let $\mathbf{X} \in \Omega \subset \mathbb{R}^d, d \in \mathbb{N}$ be a random variable and let $f(\mathbf{X}) \in L^2(\Omega)$ be a square integrable function on domain $\Omega = [0,1]^d$. ANOVA decomposition of f is given, (see [18, 27, 46] and the references therein), as follows.

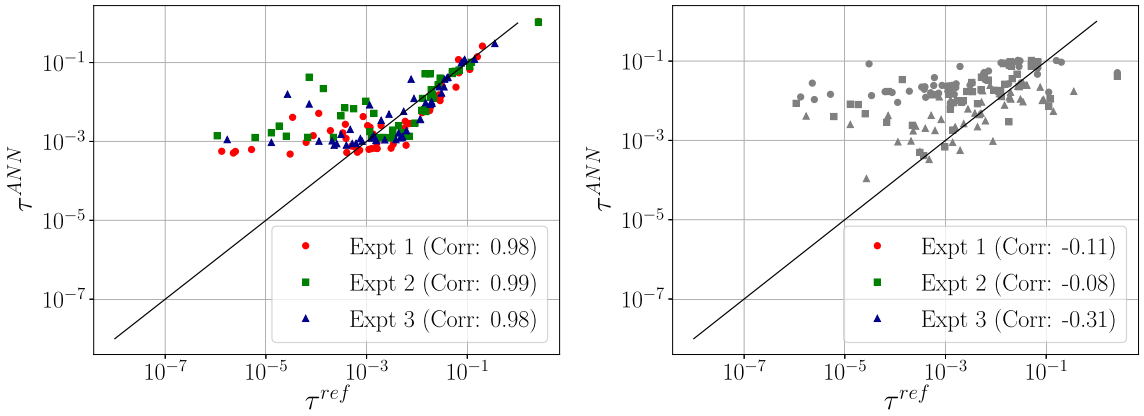


Fig. 7 ANNs showing **a** minimum mean squared error and **b** maximum mean squared error

Definition 3 ANOVA Decomposition

$$f(\mathbf{X}) = f_0 + \sum_{1 \le i \le d} f_i(X_i) + \sum_{1 \le i < j \le d} f_{ij}(X_i, X_j) + \cdots + f_{12\ldots d}(\mathbf{X}) \tag{8}$$

under the condition, (see [46]),

$$\int_0^1 f_{i_1,i_2,\ldots,i_s}(x_{i_1},\ldots,x_{i_s})dx_{i_k} = 0, \quad 1 \le k \le s, \quad \{i_1,\ldots,i_s\} \subseteq \{1,\ldots,d\}.$$

Here, f_0 is the expected value of the function f. The univariate function $f_i(X_i) = \mathbb{E}[Y|X_i] - \mathbb{E}[Y]$ quantifies contribution due to each random variable $Y = f(\mathbf{X})$. The remaining terms quantify higher-order effects. A similar decomposition of the variance of the function $f(\cdot)$ can be written as [18, 27],

$$Var(Y) = \sum_{i=1}^{d} D_i(Y) + \sum_{i<j}^{d} D_{ij}(Y) + \cdots + D_{12\ldots d}(Y), \tag{9}$$

where

$$D_i(Y) = Var[\mathbb{E}(Y|X_i)],$$
$$D_{ij}(Y) = Var[\mathbb{E}(Y|X_i,X_j)] - \big(D_i(Y) + D_j(Y)\big)$$

The Sobol' indices are given as (refer [18, 27, 46]),

Definition 4 Sobol' Indices

$$S_i = \frac{D_i(Y)}{Var(Y)}, \quad S_{ij} = \frac{D_{ij}(Y)}{Var(Y)}, \quad \text{Higher Order Terms} \tag{10}$$

Furthermore, the TO Sobol' indices combine the lower-order and higher-order terms and are given by:

$$S_{T_i} = S_i + \sum_{i<j} S_{ij} + \sum_{j\neq i, k\neq i, j<k} S_{ijk} + \quad \text{Higher Order Terms} \tag{11}$$

We use Sobol's improved algorithm presented in [48] for computation of the sensitivity indices in Eq. (11).

In order to compute the Sobol' indices, we first need a metamodel of the ANNs. We obtain such a metamodel by performing Gaussian process regression (GPR) or Kriging on design space. GPR is a powerful Bayesian approach towards regression problems [42]. In this study, the GPR is performed using the Python-based library OpenTURNS with squared exponential covariance [5]. Figure 8 shows the GPR on the activation functions of the hidden layer for one hidden layer networks with the activation function on the output layer ($OP - A$) as sigmoid function and the number of neurons on the hidden layer ($HL1 - D$) fixed to 15. Similar results are obtained for the rest of the hyper-parameters involved in this work. We refer [42] for additional details regarding the Gaussian process regression.

4.2 Total-order Sobol' indices

Figure 9 shows the total-order Sobol' indices corresponding to the four performance metrics, i.e., relative L_1 Error, MS Error, Max Error, and Min Error for the entire design space. As seen earlier, the training dataset sizes six and twelve result in a larger error. Thus, these two points should not be considered for the sensitivity analysis. We also highlight the median values with respect to TDS by a horizontal dark line. The following observations are made:

1. On average, the activation function on the output layer ($OP - A$) has the maximum value of the Sobol' index

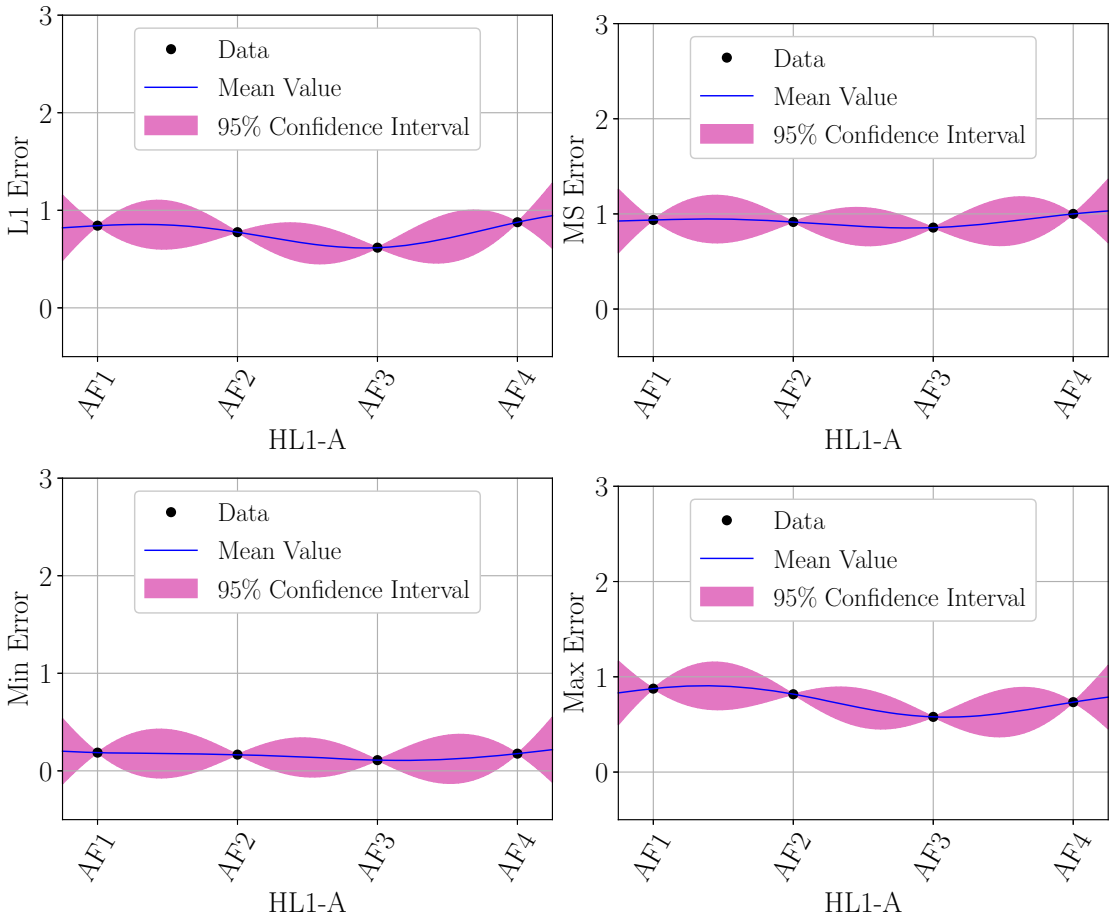


Fig. 8 Gaussian Process Regression on activation functions corresponding to the hidden layers of one hidden layer networks; $HL1 - D = 15$, $OP - A =$ Sigmoid

translating in the maximum sensitivity followed by the number of hidden layers.

2. Between the activation function and the dimension of hidden layers, the dimension has slightly more impact on the output. Thus, changing the dimension of the hidden layer may be more effective than changing the activation function.
3. The relative sensitivities are different depending on the metric under consideration.

Since these data contain ANNs of all types, it is imperative to analyze separately the ANNs with one, two, and three hidden layers as they belong to separate families.

Figure 10 shows the total-order Sobol' indices for the ANNs with only one hidden layer. Here, it is evident that for the L_1 Error, the Max Error and the Min Error, the performance metric is the most sensitive to the activation function on the output layer ($OP - A$). Next to this, it is sensitive to the dimension of the hidden layer ($HL1 - D$) followed by the activation function on the hidden layer ($HL - A$). The MS Error, on the other hand, shows similar (high) sensitivity to the activation function on the output layer and the dimension of the hidden layer.

Figure 11 shows the Sobol' indices for networks with two hidden layers exclusively, and Fig. 12 shows the same for the ANNs with three hidden layers. It is consistently observed that the activation function on the output layer ($OP - A$) has the maximum value of Sobol' indices. Dimension of hidden layers ($HL - D$) has larger sensitivity in comparison to the activation function on the hidden layers ($HL - A$). Since this analysis is performed on networks grouped together according to the number of hidden layers, it gives a more coherent picture than the results shown in Fig. 9. It is interesting to see that the activation function on the central hidden layer has the least role to play on the overall performance. This is logical as the activation function of the hidden layers closer to either end manipulates the signals during the forward and backpropagation steps, thus reducing the effect of the activation on the central hidden layer.

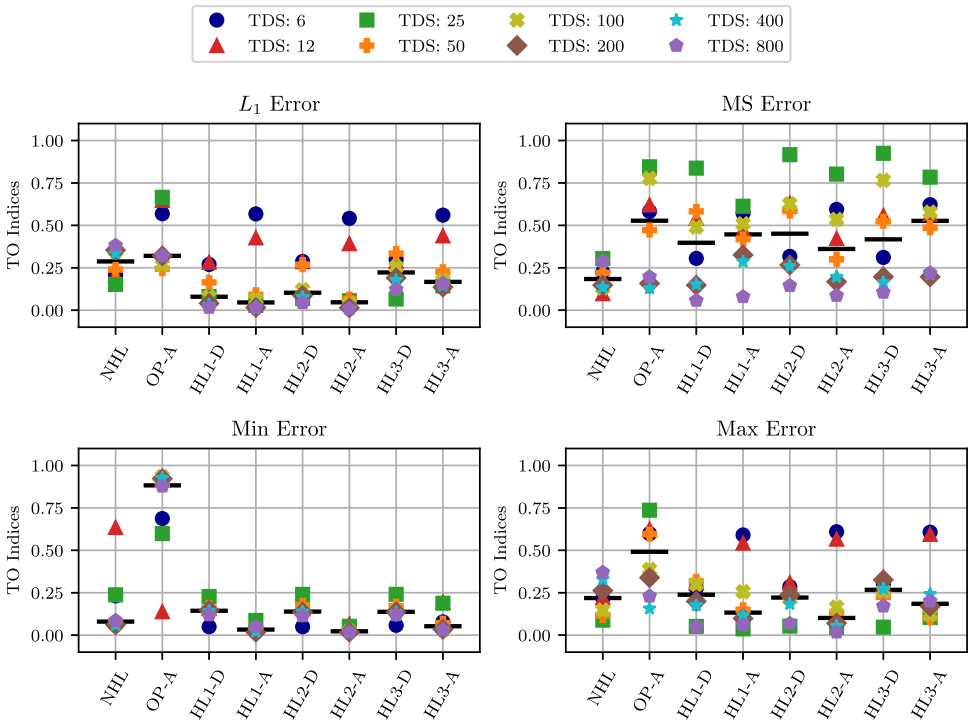


Fig. 9 Total-Order Sobol' indices for ANN hyper-parameters. Median values are indicated by dark horizontal lines

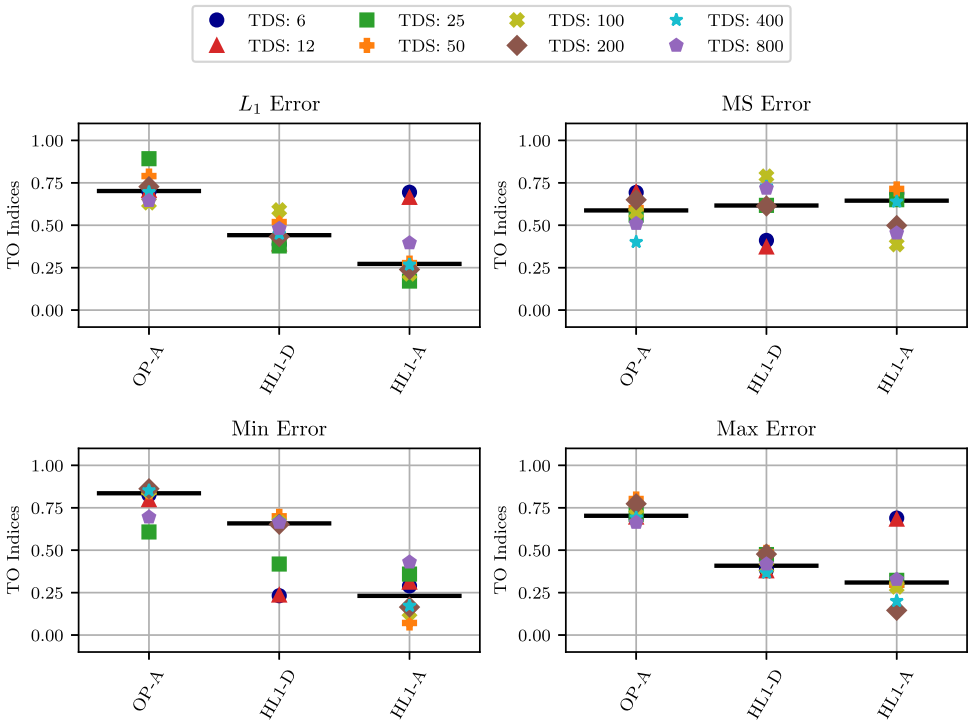


Fig. 10 Total-Order Sobol' indices for ANNs with one hidden layer

4.3 Second-order Sobol' indices

Second-order (SO) Sobol' indices are analogous to the correlation between the input dimensions. Thus, if two (or more) input hyper-parameters are interdependent, their SO Sobol' indices would likely have a large value. In order to analyze the effect of ANN hyper-parameters, it is important that the candidate hyper-parameters have small SO Sobol' index values. Figure 13 shows the values of second-order

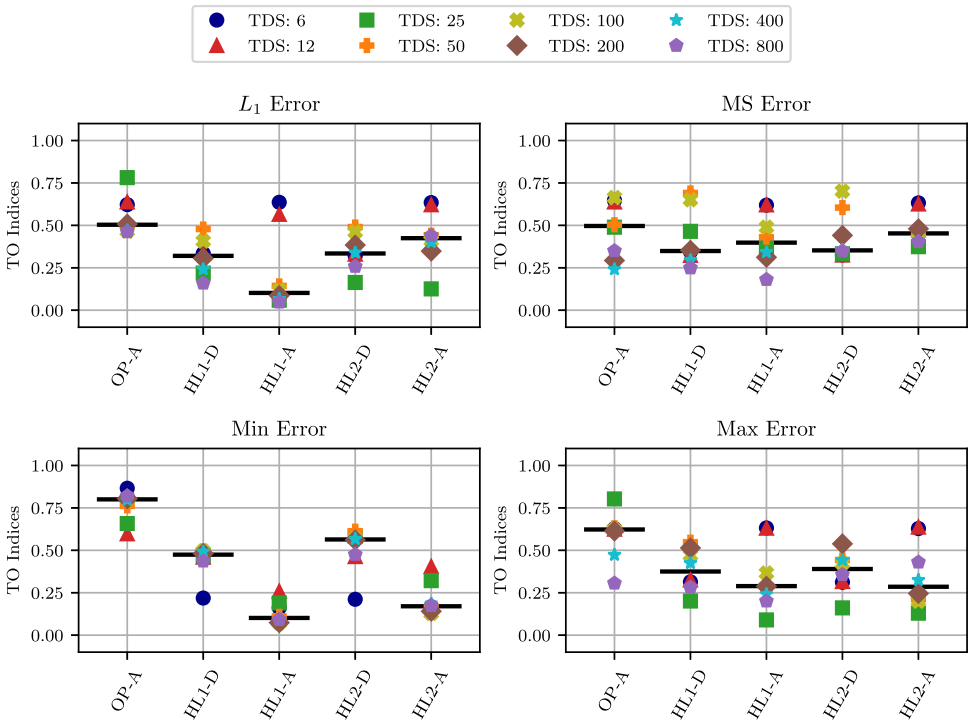


Fig. 11 Total-Order Sobol' indices for ANNs with two hidden layers

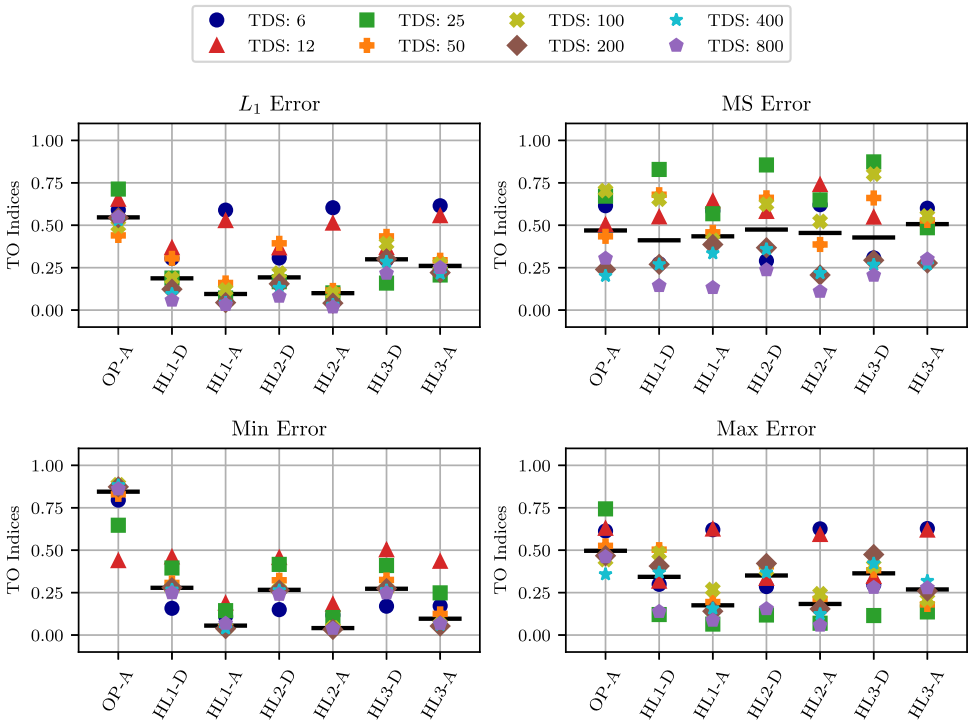


Fig. 12 Total-Order Sobol' indices for ANNs with three hidden layers

Sobol' indices between the ANN hyper-parameters. It can be seen that the cross-parameter SO Sobol' indices have relatively small values indicating near orthogonality of the input dimensions.

5 Validation on 1D test cases

In this section, we validate our analysis with the help of two examples. Based on the analysis in the previous sections, the best-performing ANN is selected to study its performance in predicting an accurate stabilization

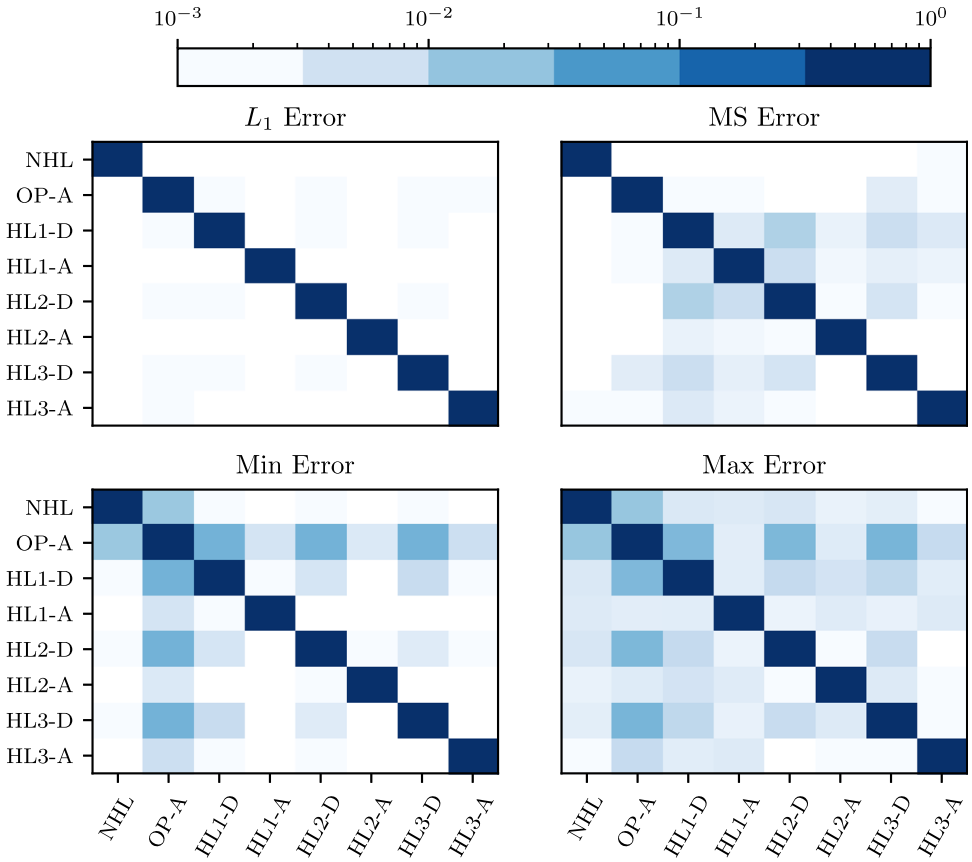


Fig. 13 Second-Order Sobol' indices

parameter. Details of the best-performing ANN are given in Table 3. For both the examples, we consider two representative training dataset sizes, i.e., TDS100 and TDS400. Moreover, an experiment consists of training each network on a randomly chosen dataset of particular size TDS, obtaining the value of τ^{ANN} for each finite element cell, and using it in the SUPG stabilized finite element numerical scheme to obtain the SUPG solution of the convection-dominated equation.

5.1 Example 1

Consider a 1D convection–diffusion boundary value problem

$$-\epsilon\phi'' + b\phi' = 1\text{on}(0,1), \phi(0) = \phi(1) = 0, b = 1. \quad (12)$$

The analytical solution of the problem is as follows:

$$\phi(x) = x - \frac{e^{\left(-\frac{1-x}{\epsilon}\right)} - e^{\left(-\frac{1}{\epsilon}\right)}}{1 - e^{\left(-\frac{1}{\epsilon}\right)}}. \quad (13)$$

Consider the coefficient as $\epsilon = 10^{-5}$ ($\epsilon << b$) which makes the equation convection dominated with a boundary layer at $x = 1$. The analytical solution is plotted in Fig. 14a. The sharp gradient (negative) in the solution at $x = 1$ is evident. Figure 14a also shows the standard Galerkin solution which has spurious oscillations. Figure 14b shows the SUPG stabilized solution with τ^{ref} in which the spurious oscillations are suppressed.

Next, we estimate the τ^{ANN} using the best-performing network with TDS400 and compute the SUPG solution with the obtained τ^{ANN}. It can be seen from Fig. 14b that the SUPG solutions obtained with τ^{ref} and τ^{ANN} are matching closely.

We next study the performance of ANNs for various diffusion parameters in order to test the robustness. Furthermore, the L_2- and $L_\infty -$ *errors* in the discrete solution are computed as

$$\begin{aligned} L_2 - error &:= \sqrt{\int_\Omega (\phi_h - \phi)^2 d\Omega} \\ L_\infty - error &:= \sup_{x\in\Omega} |\phi_h - \phi|, \end{aligned} \quad (14)$$

where ϕ is the analytical solution and ϕ_h is the discrete SUPG solution obtained with τ^{ANN}, which is estimated from various networks. Figure 15 depicts the performance of the best- and worst-performing networks with various diffusion parameters with respect to $L_2 - error$ and $L_\infty - error$. It is interesting to note that ANNs consistently

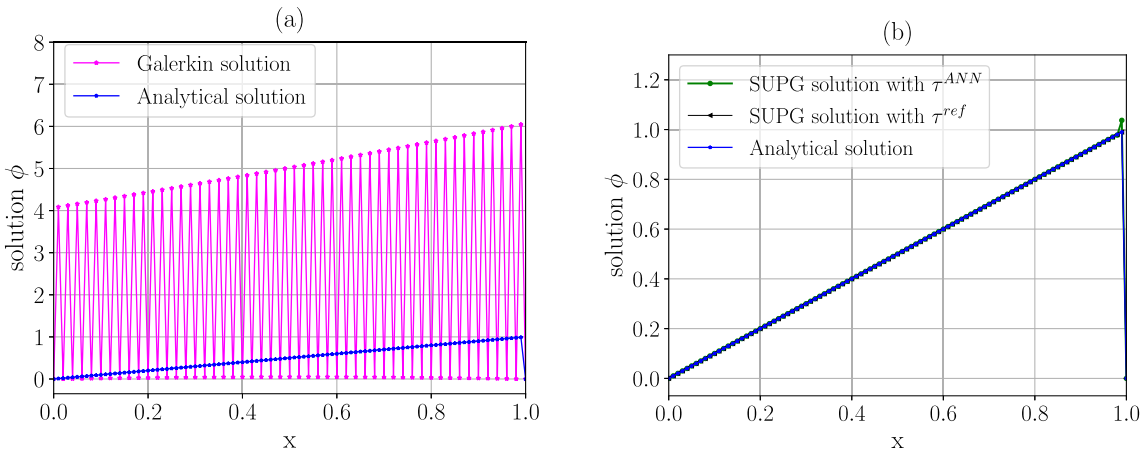


Fig. 14 **a** Analytical, Galerkin, SUPG with τ^{ref} and **b** SUPG with τ^{ANN} solutions of example one with $\epsilon = 10^{-5}$

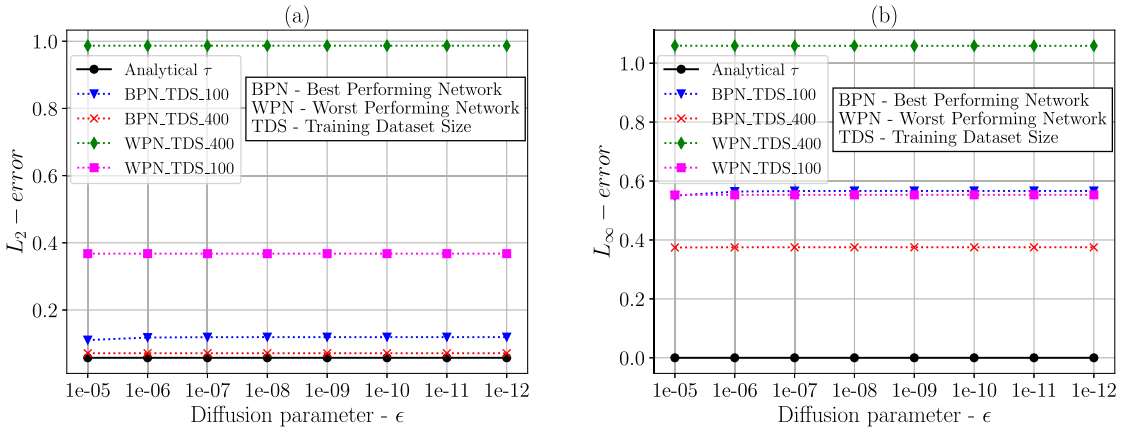


Fig. 15 **a** Absolute L_2 error in SUPG solution with τ^{ANN} for various networks and **b** absolute L_∞ errors in SUPG solution with τ^{ANN} for various networks

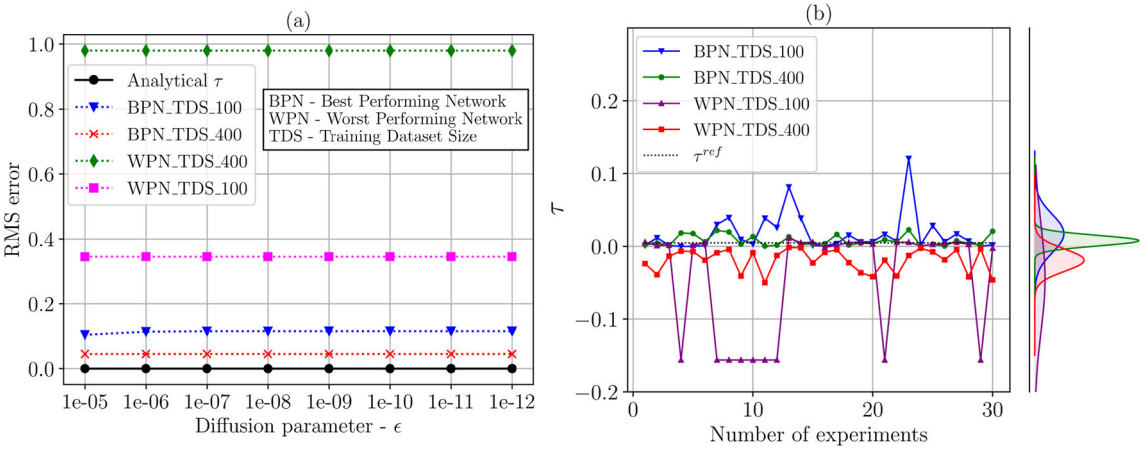


Fig. 16 **a** Absolute RMS error in stabilized solution with τ^{ANN} obtained from various networks and **b** distribution of stabilization parameter obtained from various networks for $\epsilon = 10^{-5}$

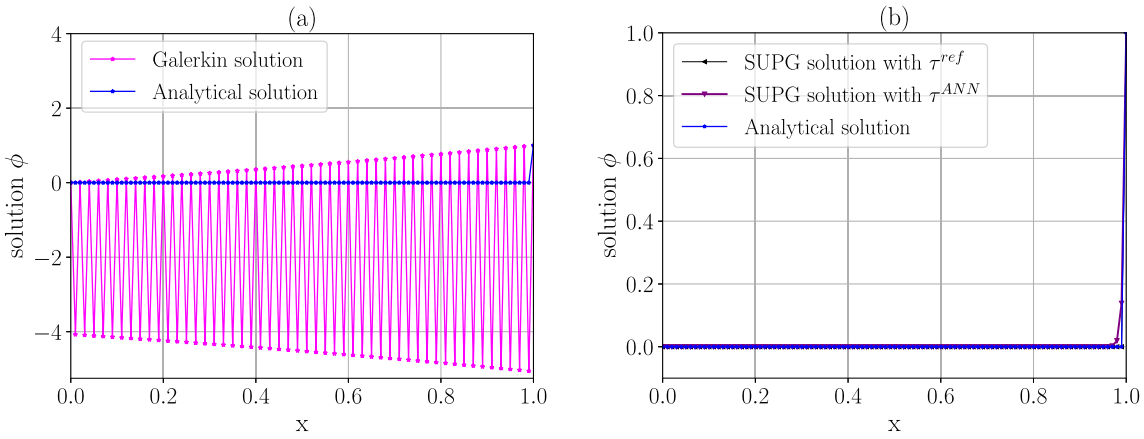


Fig. 17 **a** Analytical, Galerkin, SUPG with τ^{ref} and **b** SUPG with τ^{ANN} solutions of example two with $\epsilon = 10^{-5}$

perform better across all norms when trained with TDS400, and it is also observed in the earlier analysis.

The error comparison shown in Figs. 15a, b and 16a is obtained by taking average over thirty experiments (for each diffusion coefficient ϵ). This averaging process ensures that the error estimates reflect the overall average performance of the models. However, rarely an experiment may generate τ^{ANN} values significantly far from the normal range even for the best-performing network (BPN). One reason for this behavior may be the fact that the training data for that particular experiment could be biased against the considered test case. These eccentric τ^{ANN} values do not stabilize the solution and produce oscillations, which in turn results in high global errors, completely dominating averaging process/value. This can be readily inferred from the solution plots (Figs. 14a and 17a) where the $L_2 - norm$ of oscillating solution is several magnitudes higher than the stabilized solution. Because occurrence of such experiments is rare (approximately one in thirty), they are considered as outliers and are removed from our averaging process.

The procedure to identify these outliers is described next. For each model/network, the norm of the error for different ϵ values is stored in a matrix K (rows indexed with model and columns with ϵ). The Frobenius norm of the matrix K, given as follows, is found out.

$$||K||_F = \left(\sum_{i,j} K_{i,j}^2 \right)^{\frac{1}{2}}.$$

The norm for each of the experiment is evaluated, and the mean μ and standard deviation σ of the distribution of the norm is obtained. A metric called the z-score is obtained by the following relation.

$$\text{z-score} = \left| \frac{||K||_F - \mu}{\sigma} \right|.$$

Only the experiments with z-score less than or equal to two are considered in the averaging process. For each ϵ, the experiment is repeated for each network thirty times with different training datasets of corresponding sizes. It can be seen from Fig. 16b that the BPN with 400 training data size generates a stabilization parameter τ^{ANN} which is very close to the expected value resulting in minimum statistical variance. On the other hand, the BPN with TDS 100 generates τ^{ANN} with high statistical variance (spread of the data). For the WPN with TDS400, the τ^{ANN} is far from the expected value τ^{ref}, however, with low statistical variance. WPN with TDS100 performs the worst among the candidate networks, showing a large statistical variance and the generated τ^{ANN} values far from the expected values τ^{ref}.

5.2 Example 2

Similar set of experiments are carried out for the second example

$$-\epsilon\phi'' + \phi' = 0 \text{on}(0,1), \phi(0) = 0, \phi(1) = 1, \tag{15}$$

for which the analytical solution is

$$\phi(x) = \frac{e^{-(1-x)/\epsilon} - e^{-1/\epsilon}}{1 - e^{-1/\epsilon}}.$$

Figure 17a shows the analytical and standard Galerkin solutions. The analytical solution has a sharp gradient (positive) at $x = 1$. The spurious oscillations in the Galerkin solution are evident. Figure 17b shows SUPG solutions obtained with the parameters τ^{ref} and τ^{ANN}. In the stabilized solution obtained with τ^{ANN}, the oscillations are well-

suppressed, and the solution closely matches with the SUPG solution obtained with τ^{ref}.

6 Conclusion

An artificial neural network-based scheme is proposed to estimate the stabilization parameter in the SUPG finite element scheme. Moreover, the effects of different hyper-parameters on the accuracy of artificial neural networks are studied in detail. In this study, we observed

1. The activation function in the output layer of ANNs has the largest effect on the accuracy of the estimated stabilization parameter, followed by the number of hidden layers.
2. The activation function on the innermost hidden layer has a negligible effect on the accuracy of the ANN models.
3. Compared to the activation functions in the hidden layers, the dimension of the hidden layers has a notable effect.

We attempted to identify a recipe to build consistently well-performing and worst-performing ANNs on more than one performance metric based on these observations. The numerical results show a good correlation between the reference value of the stabilization parameter (τ^{ref}) and the ANN estimated (τ^{ANN}) obtained with the best-performing network (see Table 3 for the details of the best-performing network). Moreover, this study also serves as guidelines for selecting ANN hyper-parameters for nonlinear regression in general and SUPG stabilization in particular.

In this paper, we limited our study to the one-dimensional singularly perturbed convection–diffusion problems. ANN-based stabilization of the SUPG scheme for higher-dimensional problems as well as for other systems of PDEs may pose challenges since the reference stabilization parameter τ^{ref} may not always be available. In such cases, the objective function to be minimized during the training of ANNs needs to be defined differently. The extension of the ANN-based stabilization to higher-dimensional PDEs will be a topic of our future work.

Appendix A

Algorithm 1: Hypothesis computation

```
for i = 1 to M do
    for n = 1 to L do
        if n == 1 then
            z ← y^(i);
        else
            z ← x^(n−1);
        end
        a^(n+1) = σ(A^n z + b_{n+1})
    end
    h_Θ(y^(i)) ← a^L
end
```

Algorithm 2: Training of FF-ANN

```
Initialize weights and biases (i.e., Θ) with random values;
Perform mean normalization and feature scaling on the dataset;
Create training dataset of size M;
Forward propagation to get a hypothesis h_Θ(x);
Compute the loss function J(Θ);
while Loss function ≥ threshold value do
    for i = 1 to M do
        Forward propagation for example i;
        Compute loss function;
        Back propagation of error;
        Add correction term for each neuron;
    end
    Advance a step in optimization algorithm to improve Θ;
    Re-compute the loss function by forward propagation;
end
```

Algorithm 3: An Experiment

```
for i in [6,12,25,50,100,200,400,800] do
    TDS ← i
    for ANN ID = 1 to 7536 do
        Create the training data of size TDS by random selection;
        Create the testing data of size 50 by random selection;
        Train the ANN corresponding to ANN ID until convergence;
        Test on the testing data;
        Calculate the four performance measures, viz. L1, MS, Min, Max Errors;
    end
end
```

Acknowledgements This work is partially supported by Ministry of Education, Government of India through the scheme for transformational and advanced research in science, MoE/STARS-1/388. Furthermore, S. M. Joshi would like to acknowledge the C. V. Raman PostDoc fellowship from IISc, Bangalore.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Abgrall, R., Ricchiuto, M.: High Order Methods for CFD. Wiley, Hoboken (2017)
2. Aimone, J.B., Parekh, O., Severa, W.: Neural computing for scientific computing applications: more than just machine learning. In: Proceedings of the Neuromorphic Computing Symposium, NCS '17. Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3183584.3183618
3. Brooks, A.N.T.: Streamline upwind/petrov-galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier–Stokes equations. Comput. Methods Appl. Mech. Eng. **32**(1–3), 199–259 (1982). https://doi.org/10.1016/0045-7825(82)90071-8
4. Barron, A.R.: Universal approximation bounds for superpositions of a sigmoidal function. IEEE Trans. Inf. Theory **39**(3), 930–945 (1993). https://doi.org/10.1109/18.256500
5. Baudin, M., Dutfoy, A., Iooss, B., Popelin, A.L.: OpenTURNS: an industrial software for uncertainty quantification in simulation, pp. 1–38. Springer, Cham (2016)
6. Behzadi, F., Newman, J.C.: A semi-discrete SUPG method for contaminant transport in shallow water models. Procedia Comput. Sci. **80**, 1313–1323 (2016). https://doi.org/10.1016/j.procs.2016.05.476
7. Bochev, P.B., Gunzburger, M.D., Shadid, J.N.: Stability of the SUPG finite element method for transient advection-diffusion problems. Comput. Methods Appl. Mech. Eng. **193**(23–26), 2301–2323 (2004). https://doi.org/10.1016/j.cma.2004.01.026
8. Brooks, A.N., Hughes, T.J.R.: Streamline upwind/Petrov–Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier–Stokes equations. Comput. Methods Appl. Mech. Eng. **32**, 199–259 (1982)
9. Brunton, S.L., Noack, B.R., Koumoutsakos, P.: Machine learning for fluid mechanics. Ann. Rev. Fluid Mech. **52**(1), 477–508 (2020). https://doi.org/10.1146/annurev-fluid-010719-060214
10. Curtin, R.R., Cline, J.R., Slagle, N.P., March, W.B., Ram, P., Mehta, N.A., Gray, A.G.: MLPACK: a scalable C++ machine learning library. J. Mach. Learn. Res. **14**, 801–805 (2013)
11. Curtin, R.R., Edel, M., Lozhnikov, M., Mentekidis, Y., Ghaisas, S., Zhang, S.: Mlpack 3: a fast, flexible machine learning library. J. Open Source Softw. **3**(726), 10 (2018). https://doi.org/10.21105/joss.00726
12. Cybenko, G.: Approximation by Superpositions of a Sigmoidal Function. Math. Control Signals Syst. **2**, 303–314 (1989)
13. Cybenko, G.: Neural networks in computational science and engineering. IEEE Comput. Sci. Eng. **3**(1), 36–42 (1996). https://doi.org/10.1109/99.486759
14. Discacciati, N., Hesthaven, J.S., Ray, D.: Controlling oscillations in high-order discontinuous Galerkin schemes using artificial viscosity tuned by neural networks. J. Comput. Phys. **409**, 109304 (2020). https://doi.org/10.1016/j.jcp.2020.109304
15. Burman, E.: Consistent supg-method for transient transport problems: stability and convergence. Comput. Methods Appl. Mech. Eng. **199**(17–20), 1114–1123 (2010). https://doi.org/10.1016/j.cma.2009.11.023
16. Burman, E.P.: Edge stabilization for galerkin approximations of convection-diffusion-reaction problems. Comput. Methods Appl. Mech. Eng. **193**(15–16), 1437–1453 (2004). https://doi.org/10.1016/j.cma.2003.12.032
17. Burman, E., Fernandez, M.P.: Continuous interior penalty finite element method for Oseen's equations. SIAM J. Numer. Anal. **44**(3), 1248–1274 (2006). https://doi.org/10.1137/040617686
18. Fernández-Navarro, F., Carbonero-Ruz, M., Alonso, D.B., Torres-Jimenez, M.: Global sensitivity estimates for neural network classifiers. IEEE Trans. Neural Netw. Learn. Syst. **28**(11), 2592–2604 (2017). https://doi.org/10.1109/TNNLS.2016.2598657
19. Fock, E.: Global sensitivity analysis approach for input selection and system identification purposes—a new framework for feedforward neural networks. IEEE Trans. Neural Netw. Learn. Syst. **25**(8), 1484–1495 (2014). https://doi.org/10.1109/TNNLS.2013.2294437
20. de Frutos, J., García-Archilla, B., John, V., Novo, J.: An adaptive SUPG method for evolutionary convection-diffusion equations. Comput. Methods Appl. Mech. Eng. **273**, 219–237 (2014). https://doi.org/10.1016/j.cma.2014.01.022
21. Ganesan, S., John, V., Matthies, G., Meesala, R., Abdus, S., Wilbrandt, U.: An object oriented parallel finite element scheme for computations of pdes: design and implementation. In: 2016 IEEE 23rd International Conference on High Performance Computing Workshops (HiPCW) pp. 2–11 (2016). https://doi.org/10.1109/HiPCW.2016.023
22. Ganesan, S., Srivastava, S.: ALE-SUPG finite element method for convection-diffusion problems in time-dependent domains: conservative form. Appl. Math. Comput. **303**, 128–145 (2017). https://doi.org/10.1016/j.amc.2017.01.032
23. Giere, S., Iliescu, T., John, V., Wells, D.: SUPG reduced order models for convection-dominated convection-diffusion-reaction equations. Comput. Methods Appl. Mech. Eng. **289**, 454–474 (2015). https://doi.org/10.1016/j.cma.2015.01.020

24. Hecht-Nielsen, R.: Theory of the backpropagation neural network. In: International 1989 joint conference on neural networks, pp. 593–605 vol. 1 (1989)
25. Roos, H.G., Stynes, M.L.: Numerical Methods for Singularly Perturbed Differential Equations. Springer, Berlin (2008)
26. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Netw. **2**(5), 359–366 (1989). https://doi.org/10.1016/0893-6080(89)90020-8
27. Iooss, B., Lemaître, P.: A review on global sensitivity analysis methods. Oper. Res./Comput. Sci. Interfaces Ser. **59**, 101–122 (2015)
28. Jeon, Y.: Hybridized SUPG and upwind numerical schemes for convection dominated diffusion problems. J. Comput. Appl. Math. **275**, 91–99 (2015). https://doi.org/10.1016/j.cam.2014.08.005
29. John, V., Knobloch, P.: On spurious oscillations at layers diminishing (SOLD) methods for convection-diffusion equations: part I—a review. Comput. Methods Appl. Mech. Eng. **196**(17–20), 2197–2215 (2007). https://doi.org/10.1016/j.cma.2006.11.013
30. John, V., Novo, J.: Error analysis of the SUPG finite element discretization of evolutionary convection-diffusion-reaction equations. SIAM J. Numer. Anal. **49**(3), 1149–1176 (2011)
31. Johnson, C., Nävert, U., Pitkäranta, J.: Finite element methods for linear hyperbolic problems. Comput. Methods Appl. Mech. Eng. **45**(1–3), 285–312 (1984). https://doi.org/10.1016/0045-7825(84)90158-0
32. Kowalski, P.A., Kusy, M.: Determining the significance of features with the use of sobol method in probabilistic neural network classification tasks. In: Proceedings of the 2017 Federated Conference on Computer Science and Information Systems, FedCSIS 2017 **11**, 39–48 (2017). https://doi.org/10.15439/2017F225
33. LeVeque, R.J.: Numerical Methods for Conservation Laws. Birkhauser-Verlag, Basel (1990)
34. Li, R., Wu, Q., Zhu, S.: Proper orthogonal decomposition with SUPG-stabilized isogeometric analysis for reduced order modelling of unsteady convection-dominated convection-diffusion-reaction problems. J. Comput. Phys. **387**(18), 280–302 (2019). https://doi.org/10.1016/j.jcp.2019.02.051
35. Lye, K.O., Mishra, S., Ray, D.: Deep learning observables in computational fluid dynamics. J. Comput. Phys. **410**, 109339 (2020). https://doi.org/10.1016/j.jcp.2020.109339
36. Nasu, S., Nojima, K., Kawahara, M.: SUPG finite element method for adiabatic flows. Comput. Math. Appl. **66**(3), 250–268 (2013). https://doi.org/10.1016/j.camwa.2013.05.003
37. Ray, D., Hesthaven, J.S.: An artificial neural network as a troubled-cell indicator. J. Comput. Phys. **367**, 166–191 (2018). https://doi.org/10.1016/j.jcp.2018.04.029
38. Ray, D., Hesthaven, J.S.: Detecting troubled-cells on two-dimensional unstructured grids using a neural network. J. Comput. Phys. **397**, 108845 (2019). https://doi.org/10.1016/j.jcp.2019.07.043
39. Codina, R.: Stabilization of incompressibility and convection through orthogonal sub-scales in finite element methods. Comput. Methods Appl. Mech. Eng. **190**(13–14), 1579–1599 (2000). https://doi.org/10.1016/S0045-7825(00)00254-1
40. Saltelli, A.: Making best use of model evaluations to compute sensitivity indices. Comput. Phys. Commun. **145**(2), 280–297 (2002). https://doi.org/10.1016/S0010-4655(02)00280-1
41. Schröder, L., Dimitrov, N.K., Aasted Sorensen, J.: Uncertainty propagation and sensitivity analysis of an artificial neural network used as wind turbine load surrogate model. J. Phys. Conf. Ser. (2020). https://doi.org/10.1088/1742-6596/1618/4/042040
42. Schulz, E., Speekenbrink, M., Krause, A.: A tutorial on Gaussian process regression: modelling, exploring, and exploiting functions. J. Math. Psychol. **85**, 1–16 (2018). https://doi.org/10.1016/j.jmp.2018.03.001
43. Schwander, L., Ray, D., Hesthaven, J.S.: Controlling oscillations in spectral methods by local artificial viscosity governed by neural networks. J. Comput. Phys. **431**, 110144 (2021). https://doi.org/10.1016/j.jcp.2021.110144
44. Ganesan, S.: An operator-splitting galerkin/supg finite element method for population balance equations: stability and convergence. ESAIM Math. Modell. Numer. Anal. **46**(6), 1447–1465 (2012). https://doi.org/10.1051/m2an/2012012
45. Ganesan, S.L.: Stabilization by local projection for convection-diffusion and incompressible flow problems. J. Sci. Comput. **43**(3), 326–342 (2010). https://doi.org/10.1007/s10915-008-9259-8
46. Sobol', I.M.: Sensitivity estimates for non linear mathematical models. Math. Modell. Comput. Exp. **1**, 407–414 (1993)
47. Sobol', I.M.: Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates. Math. Comput. Simul. **55**, 271–280 (2001)
48. Sobol', I.M., Tarantola, S., Gatelli, D., Kucherenko, S.S., Mauntz, W.: Estimating the approximation error when fixing unessential factors in global sensitivity analysis. Reliab. Eng. Syst. Saf. **92**(7), 957–960 (2007). https://doi.org/10.1016/j.ress.2006.07.001
49. Hughes, T.J.R., Franca, L.P.G.: A new finite element formulation for computational fluid dynamics: Viii. The Galerkin/least-squares method. Comput. Methods Appl. Mech. Eng. **73**(2), 173–189 (1989). https://doi.org/10.1016/0045-7825(89)90111-4
50. Veiga, M.H., Abgrall, R.: Towards a general stabilisation method for conservation laws using a multilayer perceptron neural network: 1D scalar and system of equations. In: Proceedings of the 6th European Conference on Computational Mechanics: Solids, Structures and Coupled Problems, ECCM 2018 and 7th European Conference on Computational Fluid Dynamics, ECFD 2018, pp. 2525–2539 (2020)
51. John, V.J.: Error analysis of the supg finite element discretization of evolutionary convection-diffusion-reaction equations. SIAM J. Numer. Anal. **49**(3), 1149–1176 (2011). https://doi.org/10.1137/100789002
52. Wang, Z.J., Fidkowski, K., Abgrall, R., Bassi, F., Caraeni, D., Cary, A., Deconinck, H., Hartmann, R., Hillewaert, K., Huynh, H.T., Kroll, N., May, G., Persson, P.O., van Leer, B., Visbal, M.R.: High-order CFD methods: current status and perspective. International Journal for Numerical Methods in Fluids **72**(Published online 24 January 2013 in Wiley Online Library (wileyonlinelibrary.com/journal/nmf)), 811–845 (2013). https://doi.org/10.1002/fld.3767
53. Wilbrandt, U., Bartsch, C., Ahmed, N., Alia, N., Anker, F., Blank, L., Caiazzo, A., Ganesan, S., Giere, S., Matthies, G., Meesala, R., Shamim, A., Venkatesan, J., John, V.: Parmoon—a modernized program package based on mapped finite elements. Comput. Math. Appl. **74**, 74–88 (2016). https://doi.org/10.1016/j.camwa.2016.12.020
54. Yadav, S., Ganesan, S.: How deep learning performs with singularly perturbed problems? In: Proceedings—IEEE 2nd International Conference on Artificial Intelligence and Knowledge Engineering, AIKE 2019 pp. 293–297 (2019). https://doi.org/10.1109/AIKE.2019.00058