Application of Deep Galerkin Method to Solve Compressible Navier-Stokes Equations*

Masaharu MATSUMOTO1,2†

1Department of Computer Science, The University of Tokyo, Tokyo 113–8656, Japan
2Currently, Information Network Center, Fukushima University, Fukushima, Fukushima 960–1296, Japan

Recently, the application of a deep-learning technique to fluid analysis has been suggested. Additionally, a deep-learning-based method called the Deep Galerkin Method (DGM) has been suggested for solving a partial differential equation. In DGM, a loss function for training a deep neural network is formulated so that differential operators, boundary conditions, and initial conditions of the targeted partial differential equation are satisfied. This study aims to extend and apply DGM to solving compressible Navier-Stokes equations and examine the feasibility of using DGM for fluid analysis. In this paper, DGM is applied to two-dimensional Burgers equations with periodic boundary conditions, one-dimensional Navier-Stokes equations for a shock tube problem, and two-dimensional Navier-Stokes equations for the supersonic flow around a blunt body. The approximate solutions obtained using DGM show generally good agreement with that obtained using a finite difference method.

Key Words: Deep Learning, Compressible Flows, Computational Fluid Dynamics, Shock Waves

Nomenclature

\( E^k, F, G, U \): \( k \)-dimensional vector function
\( L \): loss function
\( p \): pressure
\( Re \): Reynolds number
\( s_n \): coordinate points extracted randomly
\( t \): time
\( T \): upper bound of time domain
\( x, y \): position
\( x \): \( d \)-dimensional position vector
\( u \): flow velocity
\( \gamma \): specific heat ratio \((=5/3)\)
\( \theta \): weights and biases in DNN
\( \mu \): viscosity
\( v \): diffusion coefficient
\( \rho \): density
\( \tau \): viscous stress
\( \phi \): dissipation function
\( \Omega \): bounded domain \((\subset \mathbb{R}^d)\)

1. Introduction

Recently deep learning1) has attracted much attention and is being studied and developed actively all over the world. This is a type of machine learning in which classification, regression, and cluster analysis are performed using a multilayered artificial neural network and a large amount of training data. Deep learning has become popular in the last decade because it has achieved higher performance than other methods, especially in phonetic/image recognition and natural language processing. The research and development for neural network architecture used in deep learning, including optimization algorithms based on a stochastic gradient descent method, libraries/frameworks implementing deep learning, hardware suitable for deep learning (e.g., central processing units (CPUs), graphics processing units (GPUs), and tensor processing units (TPUs)), has been actively conducted in recent years.

The application of deep learning in which the deep neural network (DNN) has excellent function approximation ability in relation to fluid analysis has also been proposed in recent years. Tompson et al. have proposed a method2) that combines a finite difference method (FDM) and a deep-learning technique using a convolutional neural network to solve Euler equations for simulating incompressible fluids in real-time. Additionally, many methods have been proposed, such as a method applied to the particle method3) and a method predicting an approximate solution using only velocity fields.4) These are intended not only for research and development, but also for applications such as computer graphics and video games. Such fluid analysis using the deep-learning method is expanding rapidly worldwide.5) In many other methods using deep learning for fluid analysis, the neural network is often trained by inputting learning data that already exists in advance, such as observed physical values from experiments or approximate solutions from numerical calculations. On the other hand, in this paper, the author proposes a new Navier-Stokes equation solution method that uses a deep-learning technique which does not rely on conventional numerical methods such as FDM or the finite element method (FEM).

Navier-Stokes equations that are simultaneous partial differential equations are often used as a mathematical model describing fluid behavior in various science and engineering fields, including aerospace. Numerical solutions of Navier-
Stokes equations are among the most computation-intensive tasks in scientific and engineering applications. The research and development of numerical algorithms for solving Navier-Stokes equations are widely carried out to improve calculation accuracy or speed. There is no example of using deep learning directly to solve Navier-Stokes equations, and numerical solution methods that do not rely on conventional numerical methods such as FDM or FEM are rare. The proposal of such a new solution method using deep learning is related to artificial intelligence and data science, which has become popular in recent years and is expected to lead to future development.

Attempts to obtain an approximate solution for a differential equation using a neural network of one to several layers have been proposed since the 1990s. At that time, the technique to transform the hidden layer of the neural network into multiple layers was still in its infancy, and the expressive power of the neural network had a limit, so that it seemed to only obtain solutions for comparatively simple differential equations. However, recent technological developments in the realm of deep learning have made it possible to apply more complex differential equations. Sirignano et al. suggested a deep-learning-based method called the Deep Galerkin Method (DGM) solving a partial differential equation (PDE). In DGM, a loss function for training the DNN is for the initial conditions of the targeted PDE are satisfied. The approximate solution can be obtained by training a DNN in which the space-time coordinate points generated using pseudo-random numbers are used as input data. Sirignano et al. showed that DGM could obtain approximate solutions of multi-dimensional Hamilton-Jacobi-Bellman equations and one-dimensional Burgers equations. The computational cost for training using DGM is not low, but once the training is completed, it becomes possible to predict the approximate solution of the targeted PDE using only the basic linear computation of the matrix. However, this method still has only a few application examples, and it is not clear what kind of PDE can be solved. In this study, the author focuses on applying DGM to solve Navier-Stokes equations for compressible fluid analysis. In fluid analysis using conventional methods with deep learning, the fluid behavior is predicted using a methodology that extracts the feature quantity from a learning dataset that already exists in advance, such as observed physical values using experiments or approximate solutions using numerical calculations. On the other hand, in fluid analysis using DGM, the fluid behavior is predicted using a methodology that evaluates, corrects, and memorizes the approximate solution of the Navier-Stokes equations applying the function approximation ability of a neural network.

Based on the above, in this study, a new method of solving Navier-Stokes equations using a deep-learning-based method is proposed. Specifically, this study aims to extend and apply DGM to solving compressible Navier-Stokes equations and examines the feasibility of using DGM for fluid analysis.

2. DGM

DGM has mainly two characteristics. The first is that it is not necessary to prepare a large number of labeled training datasets for the input data used to train the DNN. The second is that it does not require computational grids that are often used in the numerical method to analyze PDEs (i.e., DGM is a mesh-free method).

2.1. DGM algorithm

The DGM algorithm used in this study is described here. It is suggested that original DGM be applied to a scalar parabolic PDE, but the Navier-Stokes equations targeted in this study are simultaneous PDEs. Therefore, the DGM algorithm needs to be extended to apply to simultaneous PDEs. Here, the spatial d-dimensional simultaneous PDEs shown below are targeted in this study,

\[
\frac{\partial U(x,t)}{\partial t} + \sum_{i=1}^{d} \frac{\partial E_i(U(x,t))}{\partial x_i} = 0 \quad x \in \Omega
\]

\[
U(x,t) = G(x,t) \quad x \in \partial \Omega
\]

\[
U(x,t = 0) = U_0(x) \quad x \in \Omega
\]

where \(0 \leq t \leq T\), and \(U, E_i, G\) and \(U_0\) are k-dimensional vector functions (i.e., \(k\) is the number of unknown dependent variables). \(G\) and \(U_0\) indicate boundary and initial conditions, respectively. In DGM, the approximate PDE solution can be obtained according to the following processes (see also Fig. 1).

1. The space-time coordinates \((x_1,t_1)\) are extracted randomly from the computational domain \((\Omega \times [0,T])\), the space-time coordinates \((x_2,t_2),(x_3,t_3)\) are extracted randomly from the region corresponding to the boundary conditions \((\partial \Omega \times [0,T])\), and the space coordinates \((x_3,0)\) are extracted randomly from the region corresponding to the initial condition \((\Omega)\).

2. The loss function \(L\) is calculated using the coordinate points obtained randomly \(s_n = (x_{1n},t_{1n}),(x_{2n},t_{2n}), (x_{3n},0)\) as the input data of DNN. Here, \(n\) indicates the number of iterations,

\[
L = \sum_{i=1}^{k} \frac{\partial E_i(U_{1n})}{\partial \theta_i} + \frac{\partial E_i(U_{2n})}{\partial \theta_i} + \frac{\partial E_i(U_{3n})}{\partial \theta_i}
\]

\[
+ \sum_{j=1}^{d} \frac{\partial E_j(U_{1n})}{\partial x_j} + \frac{\partial E_j(U_{2n})}{\partial x_j} + \frac{\partial E_j(U_{3n})}{\partial x_j}
\]

\[
+ \sum_{j=1}^{d} \frac{\partial E_j(U_{1n})}{\partial x_j} + \frac{\partial E_j(U_{2n})}{\partial x_j} + \frac{\partial E_j(U_{3n})}{\partial x_j}
\]

where, \(L = L(\theta, s_n)\), and \(\theta\) indicates the parameters optimized in DNN (i.e., so-called weights and biases). Additionally, \(F(x,t; \theta)\) indicates the approximate solution of \(U\) predicted (output) by DNN. \(F_i, G_i, U_0\) indicate an element of vector \(F, E_i, G, \text{and } U_0\), respectively. The purpose of DGM is to find the \(\theta\) that minimizes the loss function \(L\).

3. \(\theta_{n+1}\) is updated using an optimization method based on a stochastic gradient descent method.
the functions right-hand side in Eq. (2) are not given explicitly, although together using the number of equations. The second is that the loss function is added to the number of solutions for the simultaneous equations. Therefore, in DGM, unlike deep learning conducted generally, the more overfitted, the smaller the difference from the exact solution. It should also be noted that DGM cannot be used to predict the other solution under a different boundary and initial conditions. In the extended DGM for simultaneous equations, there are mainly two improvements from the original DGM.\(^ {10} \) First, the output of the neural network is directly used (i.e., until the loss function decreases to a specific value or less). This condition depends on the PDE to be solved, but in this study, less than approximately 0.01 is sufficient.

The three terms on the right-hand side in Eq. (2) evaluate the summation of differences from differential operators (first term), boundary conditions (second term), and initial conditions (third term) of the targeted PDEs. If Eq. (2) becomes zero, then \( F \) becomes the approximate solution of Eq. (1) (i.e., DNN becomes the approximate function of \( U \)). That is, DGM performs multiple regression analysis using a huge nonlinear function that is DNN as a regression function while evaluating differential operators. Reducing the loss function has a direct effect on the accuracy of the predicted solution. Therefore, in DGM, unlike deep learning conducted generally, the more overfitted, the smaller the difference from the exact solution. It should also be noted that DGM cannot predict the solution outside the computational domain \( (\Omega \times [0, T]) \), and the obtained approximate solution cannot be used to predict the other solution under a different boundary and initial conditions. In the extended DGM for simultaneous equations, there are mainly two improvements from the original DGM.\(^ {10} \) First, the output of the neural network is increased to the number of solutions for the simultaneous equations. The second is that the loss function is added together using the number of equations.

The time and spatial derivatives of the first term on the right-hand side in Eq. (2) are not given explicitly, although the functions \( G \) and \( U \) that correspond to the boundary and initial conditions of the second and third terms are evident from Eq. (1). However, \( F(x, t; \theta) \) can be obtained when the DNN architecture is determined, so these derivatives can be obtained by automatic differentiation using the chain rule. For example, a time derivative of \( F \) can be obtained using automatic differentiation as follows.

\[
\theta_{n+1} = \theta_n - \alpha_n \nabla_{\theta} L(\theta_n, s_n) \tag{3}
\]

where \( \alpha \) indicates the learning rate.

4. Repeat the above steps 1–3 until the convergence condition is satisfied (i.e., until the loss function \( L \) decreases to a specific value or less). This condition depends on the PDE to be solved, but in this study, less than approximately 0.01 is sufficient. Here, \( w_n \) indicates intermediate computation on \( n \)-th node represented on the neural network architecture. The derivative of \( F \) can be broken down into intermediate computations that vary concerning the input of the neural network.

2.2. DGM network architecture

DGM requires a model \( F(x, t; \theta) \) that can draw a nonlinear profile such that the function \( U(x, t) \) is steeply discontinuous with respect to \( x \). Sirignano et al. have suggested the architecture\(^ {10} \) as a network suitable for DGM referencing the Long Short-Term Memory (LSTM) architecture.\(^ {12} \) Considering the architecture suggested by Sirignano et al. and the number of dependent variables in the targeted simultaneous PDEs, the architecture used in this study is set as follows (see also Fig. 2).

\[
F(x, t; \theta) = WS_{L+1} + b \tag{5}
\]

\( S^l = \sigma(W^lX + b^l), \)
\( Z^l = \sigma(U^{l-1}X + W^{l-1}S^l + b^{l-1}), \quad \ell = 1, \ldots, L \)
\( G^\ell = \sigma(U^{\ell-1}X + W^{\ell-1}S^l + b^{\ell-1}), \quad \ell = 1, \ldots, L \)
\( R^\ell = \sigma(U^{\ell-1}X + W^{\ell-1}S^l + b^{\ell-1}), \quad \ell = 1, \ldots, L \)
\( H^\ell = \sigma(U^{\ell-1}X + W^{\ell-1}R^\ell + b^{\ell+1}S^{\ell+1}) \quad \ell = 1, \ldots, L \)
\( S^{l+1} = (1 - G^\ell) \odot H^\ell + Z^\ell \odot S^l \)
Here, $X = (x, t) \in \mathbb{R}^{d+1}$. $L$ indicates the number of hidden layers, and the symbol $\odot$ indicates the Hadamard product. The number of units per each layer is $M$, and $\sigma: \mathbb{R}^M \to \mathbb{R}^M$ is an element-wise nonlinear transformation as follows,

$$\sigma(z) = (\phi(z_1), \phi(z_2), \ldots, \phi(z_M))$$  \hspace{1cm} (6)$$

where, $\phi: \mathbb{R} \to \mathbb{R}$ is a nonlinear activation function. The weight parameters in $\theta$ have dimensions $\{W^1, U^c, U^b, U^{c,b}, W^h, W^{c,b}, W^{h,c}\} \in \mathbb{R}^{M \times (d+1)}$, $\{W^c, W^b, W^{c,b}, W^{h,c}\} \in \mathbb{R}^{M \times M}$, $\{b^1, b^c, b^{c,b}, b^{h,c}\} \in \mathbb{R}^M$, $W \in \mathbb{R}^{k \times M}$, $b \in \mathbb{R}^k$. The number of hidden layers $L$, the number of units per each layer $M$, and the activation function $\phi$ that needs to be determined are hyperparameters. According to Sirignano et al., $L = 3$, $M = 50$, and $\phi(z) = \tanh(z)$ are useful, so these values are also used in this study.

### 2.3. Implementation of DGM

In this study, the DGM network architecture is implemented using Tensorflow, which is often used as a framework for machine learning. Tensorflow has a reverse-mode automatic differentiation function that can easily calculate the derivatives of a wide range of variables. The backpropagation used for training the DNN and the estimation of derivatives for $i, x$, and $\theta$ are implemented using the Tensorflow function in this study. The optimization using the ADAM algorithm based on a stochastic gradient descent method is applied in the training. In this study, synchronized distributed parallel training is performed using two GPGPUs (NVIDIA Titan RTX). As mentioned in the previous section, the space-time coordinate points extracted randomly are used as the input data required for DNN training. The batch size per each GPU node with 4,096 i.e., input data size is $s_i \times 4,096$ space-time coordinate points is used per one iteration. The total number of iterations is 25,600 for training. The training time for DNN depends on the various working conditions, such as the number of iterations, the number of simultaneous PDEs, the complexity of PDEs, the hyperparameters in the architecture of DNN, the hardware and software used, etc. The computational load for DNN training is by no means small, and the typical training time in this study is on the order of several hours to a day.

### 3. Results and Discussion

In this section, the following three results are discussed: 1) Two-dimensional Burgers equations (i.e., a preliminary study), 2) One-dimensional compressible Navier-Stokes equations (i.e., shock tube problem), and 3) Two-dimensional compressible Navier-Stokes equations (i.e., supersonic flow around a blunt body).

#### 3.1. Two-dimensional Burgers equations — a preliminary study

Compressible Navier-Stokes equations are nonlinear simultaneous PDEs that consist of a mass conservation equation, momentum conservation equation, and total energy conservation equation. The original DGM has been suggested as an approximate solution for a PDE, so it is necessary to examine whether or not the DGM suggested here can be adapted to simultaneous PDEs. Therefore, in this section, an analysis of two-dimensional Burgers equations is carried out as a preliminary study. Burgers equations are nonlinear advection-diffusion equations represented as a form without a pressure gradient term in incompressible Navier-Stokes equations. Multi-dimensional Burgers equations form into simultaneous PDEs. The governing equations targeted here are as follows.

$$\begin{align*}
\frac{\partial u_x}{\partial t} + u_x \frac{\partial u_x}{\partial x} + u_y \frac{\partial u_y}{\partial y} &= \nu \left( \frac{\partial^2 u_x}{\partial x^2} + \frac{\partial^2 u_x}{\partial y^2} \right) \\
\frac{\partial u_y}{\partial t} + u_x \frac{\partial u_y}{\partial x} + u_y \frac{\partial u_y}{\partial y} &= \nu \left( \frac{\partial^2 u_y}{\partial x^2} + \frac{\partial^2 u_y}{\partial y^2} \right)
\end{align*}$$

(7)

The boundary and initial conditions for Burgers equations can be set arbitrarily. The boundary conditions are set to be periodic as shown below.

$$\begin{align*}
u(0, y, t) &= \nu(1, y, t) \\
u(x, 0, t) &= \nu(x, 1, t)
\end{align*}$$

(8)

where $\nu(x, y, t) = (u_x(x, y, t), u_y(x, y, t))$. The initial conditions are set using trigonometric functions as shown below.

$$\begin{align*}
u_x(x, y, 0) &= \sin(4\pi y) \\
u_y(x, y, 0) &= \frac{1}{2} \cos(2\pi x)
\end{align*}$$

(9)

where, the ranges of computational domain are set to $0 \leq x \leq 1$, $0 \leq y \leq 1$ and $0 \leq t \leq 1$, and the coefficient $\nu$ of the diffusion term in Eq. (7) is set to six types: $0$, $1 \times 10^{-4}$, $5 \times 10^{-4}$, $1 \times 10^{-3}$, $5 \times 10^{-3}$, and $1 \times 10^{-2}$. Considering the governing equations are simultaneous PDEs, the loss function is set as follows.

$$L = (L_1 + L_2) + (L_3 + L_4) + (L_5 + L_6)$$

(10)

$L_1$ and $L_2$ are parts of the loss function correspond to the evaluations of differential operators.

$$\begin{align*}
L_1 &= f_1^1 f_2^1 + f_2^1 f_1^2 + f_1^2 f_1^1 - \nu \left( \frac{\partial f_1^1}{\partial x} + \frac{\partial f_1^2}{\partial y} \right) \\
L_2 &= f_1^2 f_2^1 + f_2^1 f_1^2 + f_1^2 f_2^1 - \nu \left( \frac{\partial f_1^2}{\partial x} + \frac{\partial f_1^1}{\partial y} \right)
\end{align*}$$

(11)

Here, $f_1^1$ and $f_1^2$ mean $f(x_1, y_1, t_1)$ and $f(x_1, y_1, t_1)$, respectively. $L_3$ and $L_4$ correspond to the evaluation of boundary conditions.

$$\begin{align*}
L_3 &= f_3(x_1, y_1, t_2) - f_3(x_1, y_1, t_2) (x_2 = 1) \\
f_3(x_1, y_1, t_2) &= f_3(x_1, y_1, t_2) (x_2 = 0)
\end{align*}$$

(12)

$$\begin{align*}
L_4 &= f_3(x_1, y_1, t_2) - f_3(x_1, y_1, t_2) (y_2 = 1) \\
f_3(x_1, y_1, t_2) &= f_3(x_1, y_1, t_2) (y_2 = 0)
\end{align*}$$

Here, $f_3$ and $f_3$ are approximate solutions predicted as an
output of DNN. \(x_1, y_1,\) and \(t_1\) are the space-time coordinate points extracted randomly from the computational domain; \(x_2, y_2,\) and \(t_2\) are the space-time coordinate points extracted randomly from the regions corresponding to the boundary conditions; and \(x_3\) and \(y_3\) are the space coordinate points extracted randomly from the regions corresponding to the initial conditions. The training of DNN is conducted from the case of \(v = 0.01,\) which has the highest value for \(v,\) in descending order of value. As discussed later, it is easier to train for high \(v,\) and the convergence of the training can be accelerated even for low \(v\) using the training results of the weight parameters for high \(v\) as the initial values. The initial values for weight parameters in the case of \(v = 0.01\) are random numbers based on the Xavier initialization. \(^{15}\) Otherwise, the weight parameters from the previous training result are used as the initial values.

Figure 3 shows the predicted solution of \(u_x\) determined using DGM under the conditions of \(v = 5 \times 10^{-4}, t = 1,\) and \(x_1, y_1,\) and \(t_1\) are the space-time coordinate points randomly from the computational domain; \(x_2, y_2,\) and \(t_2\) are the space-time coordinate points randomly from the regions corresponding to the boundary conditions; and \(x_3\) and \(y_3\) are the space coordinate points randomly from the regions corresponding to the initial conditions. The training of DNN is conducted from the case of \(v = 0.01,\) which has the highest value for \(v,\) in descending order of value. As discussed later, it is easier to train for high \(v,\) and the convergence of the training can be accelerated even for low \(v\) using the training results of the weight parameters for high \(v\) as the initial values. The initial values for weight parameters in the case of \(v = 0.01\) are random numbers based on the Xavier initialization. \(^{15}\) Otherwise, the weight parameters from the previous training result are used as the initial values.

Figure 3 shows the predicted solution of \(u_x\) determined using DGM under the conditions of \(v = 5 \times 10^{-4}, t = 1,\) and \(x_1, y_1,\) and \(t_1\) are the space-time coordinate points randomly from the computational domain; \(x_2, y_2,\) and \(t_2\) are the space-time coordinate points randomly from the regions corresponding to the boundary conditions; and \(x_3\) and \(y_3\) are the space coordinate points randomly from the regions corresponding to the initial conditions. The training of DNN is conducted from the case of \(v = 0.01,\) which has the highest value for \(v,\) in descending order of value. As discussed later, it is easier to train for high \(v,\) and the convergence of the training can be accelerated even for low \(v\) using the training results of the weight parameters for high \(v\) as the initial values. The initial values for weight parameters in the case of \(v = 0.01\) are random numbers based on the Xavier initialization. \(^{15}\) Otherwise, the weight parameters from the previous training result are used as the initial values.

Figure 3 shows the predicted solution of \(u_x\) determined using DGM under the conditions of \(v = 5 \times 10^{-4}, t = 1,\) and \(x_1, y_1,\) and \(t_1\) are the space-time coordinate points randomly from the computational domain; \(x_2, y_2,\) and \(t_2\) are the space-time coordinate points randomly from the regions corresponding to the boundary conditions; and \(x_3\) and \(y_3\) are the space coordinate points randomly from the regions corresponding to the initial conditions. The training of DNN is conducted from the case of \(v = 0.01,\) which has the highest value for \(v,\) in descending order of value. As discussed later, it is easier to train for high \(v,\) and the convergence of the training can be accelerated even for low \(v\) using the training results of the weight parameters for high \(v\) as the initial values. The initial values for weight parameters in the case of \(v = 0.01\) are random numbers based on the Xavier initialization. \(^{15}\) Otherwise, the weight parameters from the previous training result are used as the initial values.

Figure 4 shows the approximate solution of \(u_x\) by an upwind finite difference method (hereinafter referred to as FDM). Since two-dimensional Burgers equations are nonlinear, it is not easy to solve these analytically. Therefore, the FDM approximate solution (i.e., number of grid points is \(N_x \times N_y = 101 \times 101\)) is shown for comparison. The predicted solution determined using DGM shown in Fig. 3 is visualized with the same number of grid points used in FDM shown in Fig. 4. Comparing the two results shows good agreement, proving that DGM can be effectively adapted for solving simultaneous PDEs.

Figure 5 shows the dependency of the loss function on iterations under the conditions of various \(v\) in the training using DGM.
ond using a CPU (Intel Xeon E5-2630 v4) implemented by Fortran with multiple threads of OpenMP. Instead, once the training is completed using DGM, the approximate solution at an arbitrary location and time in the computational domain can be predicted. On the other hand, only values on grid points can be obtained using FDM. When using DGM, the solution prediction time for 101 × 101 grid points used in the FDM at a specific time is approximately several seconds.

### 3.2. One-dimensional Navier-Stokes equations for a shock tube problem

This section focuses on a shock tube problem using one-dimensional Navier-Stokes equations. A shock tube problem is often solved to validate a numerical (discretization) method, such as FDM. For that purpose, the Euler equations (i.e., inviscid Navier-Stokes equations) are often used as the governing equations because the influence of artificial viscosity in the numerical calculation needs to be examined. If the Euler equations are set to the governing equations when solving this problem using DGM, a strong discontinuity surface at the shock front is generated due to the zero viscosity. In such a case, the steep gradient representation in DNN may be difficult as mentioned in the previous section. Therefore, in this study, the Navier-Stokes equations that do not ignore the viscous term are set as the governing equations. The governing equations targeted here are formulated as follows.

\[
\begin{aligned}
\frac{\partial \rho}{\partial t} + \frac{\partial \rho u}{\partial x} + \frac{\partial \rho u}{\partial x} = 0 \\
\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} + \frac{\partial p}{\partial x} = \frac{1}{Re} \frac{\partial^2 u}{\partial x^2} \\
\frac{\partial p}{\partial t} + \frac{\partial p}{\partial x} + \gamma \frac{\partial u}{\partial x} = (\gamma - 1) \frac{1}{Re} \left( \frac{\partial u}{\partial x} \right)^2 
\end{aligned}
\]

(14)

Here, Eq. (14) is nondimensionalized (normalized) by the following representative variables: length \(x\), density \(\rho\), velocity \(u\), time \(t_{ri}\), and pressure \(p_{ri} = \rho_{ri}u_{ri}^2\). \(Re = \rho_{ri}u_{ri}\) indicates a Reynolds number that is the ratio of inertial forces to viscous forces in a fluid motion. Additionally, the heat conduction term in the energy conservation equation of Eq. (14) is ignored, and \(Re\) is assumed to be a constant in the whole computational domain for simplicity. Furthermore, Eq. (14), except for the mass conservation equation, is expressed as a non-conservation form, and the total energy conservation equation is converted to an internal energy conservation equation notated by pressure. There are many multiplications of variables differentiated by time and space in a conservation form. But in a non-conservation form, the derivatives are decomposed, and only one variable is differentiated, as shown in Eq. (14). A non-conservation form is suitable for evaluating derivatives when calculating the loss function of DNN using DGM. The influence of the difference between a conservation form and a non-conservation form to the predicted DGM result should be examined in the future. But in this study, the derivatives are estimated under a non-conservation form.

Figure 6 shows the computational domain and working conditions. The computational domain is the range from \(0 \leq x \leq 1\) and \(0 \leq t \leq 1\), and a high-pressure region is located upstream (i.e., left side in Fig. 6), and a low-pressure region is located downstream (i.e., right side in Fig. 6) as the initial conditions. The boundary conditions upstream and downstream are set to the Neumann condition. That is, the gradient for all of the physical quantities is set to zero. Based on the above, the loss function \(L\) is set as follows.

\[
L = \sum_{i=1}^{3} L_i^2 + \sum_{i=4}^{6} L_i^2 + \sum_{i=7}^{9} L_i^2 
\]

(15)

\(L_1, L_2, \) and \(L_3\) that are parts of the loss function correspond to the evaluations of differential operators.

\[
\begin{aligned}
L_1 &= \frac{\partial f_1^i}{\partial t} + f_1^i u \frac{\partial f_1^i}{\partial x} + f_1^i \frac{\partial f_1^i}{\partial x} \\
L_2 &= f_1^i \frac{\partial f_1^i}{\partial t} + f_2^i u \frac{\partial f_1^i}{\partial x} + f_2^i \frac{\partial f_1^i}{\partial x} - \frac{1}{Re} \frac{\partial^2 f_1^i}{\partial x^2} \\
L_3 &= \frac{\partial f_1^i}{\partial t} + f_3^i u \frac{\partial f_1^i}{\partial x} + \gamma \frac{\partial f_1^i}{\partial x} - (\gamma - 1) \frac{1}{Re} \left( \frac{\partial f_1^i}{\partial x} \right)^2
\end{aligned}
\]

Here, \(f_1^i, f_2^i, \) and \(f_3^i\) mean \(f_\rho(x_1, t_1), f_u(x_1, t_1), \) and \(f_p(x_1, t_1),\) respectively. \(L_4, L_5, \) and \(L_6\) correspond to the evaluations of boundary conditions.

\[
\begin{aligned}
L_4 &= f_\rho(x_2, t_2) - \frac{\partial f_\rho(x_2, t_2)}{\partial x} \\
L_5 &= f_u(x_2, t_2) - \frac{\partial f_u(x_2, t_2)}{\partial x} \\
L_6 &= f_p(x_2, t_2) - \frac{\partial f_p(x_2, t_2)}{\partial x}
\end{aligned}
\]

(17)

Additionally, \(L_7, L_8, \) and \(L_9\) correspond to the evaluations of initial conditions.

\[
\begin{aligned}
L_7 &= \begin{cases} 
0 & (0 \leq x_3 \leq 0.5) \\
f_\rho(x_3, 0) - 1.0 & (0.5 < x_3 \leq 1.0)
\end{cases} \\
L_8 &= \begin{cases} 
0.125 & (0 \leq x_3 \leq 0.5) \\
f_u(x_3, 0) - 0.125 & (0.5 < x_3 \leq 1.0)
\end{cases} \\
L_9 &= \begin{cases} 
0 & (0 \leq x_3 \leq 0.5) \\
f_p(x_3, 0) - 0.1 & (0.5 < x_3 \leq 1.0)
\end{cases}
\end{aligned}
\]

(18)

where \(f_\rho, f_u, \) and \(f_p\) indicate the approximate solutions of \(\rho, u, \) and \(p\) that are predicted as the output of DNN. \(x_1 \) and \(t_1\) are the space-time coordinate points extracted randomly from the computational domain, \(x_2 \) and \(t_2\) are the space-time coor-
dinate points extracted randomly from the regions corresponding to the boundary conditions, and $x_3$ is the space coordinate point extracted randomly from the regions corresponding to the initial conditions.

When using DGM, the nondimensionalization (normalization) of governing equations has a vital role in the training results of DNN. In general, if the scales of input/output (IO) data for the DNN are significantly different, the training accuracy and speed may decrease because the weight parameters in the DNN are biased when updated using a stochastic gradient descent method. Physical quantities in Navier-Stokes equations include density, flow velocity, and pressure, etc., but the scales of these values vary greatly under dimensionality. For high-precision training, it is necessary to make the scales of the coordinate values given to the DNN as the input data and the physical values of the solution obtained from the DNN as the output data uniform. This issue can be solved by performing the nondimensionalization of variables in the targeted governing equations. Here, the working conditions (see Fig. 6) are set so that the absolute values of all of the IO data for DNN would be approximately 0 to 1 using nondimensionalization.

The numerical results obtained using FDM (i.e., second-order TVD scheme\cite{16}) are also shown here to compare and validate the predicted results using DGM. When using FDM, the working conditions are the same as those of DGM (see Fig. 6). The number of spatial grid points is set to 1,001. The governing equations are calculated in a conservation form, unlike the DGM case. Figures 7–9 show the distributions of density $\rho$, flow velocity $u$, and pressure $p$ at $t = 0.2$ in the predicted DGM results and the numerical FDM results under the condition of $1/Re = 10^{-3}$. For the shock tube problem, the shock-wave front and the contact discontinuity surface propagate downstream (i.e., right side in Fig. 6), and the expansion wave propagates upstream (i.e., left side in Fig. 6) over time from the initial condition. These figures show that the shock-wave front, contact surface, and expansion wave propagate around $x = 0.9$, $x = 0.7$, and $x = 0.2$–$0.5$, respectively. Comparing the predicted DGM results and the numerical FDM results, they generally show good agreement, although there are slight differences in the region where the distribution changes significantly. To see more details about this error, Fig. 10 shows a $x$–$t$ diagram of the density distribution obtained using DGM, and Fig. 11 shows the $x$–$t$ diagram of the error between DGM and FDM. The error in Fig. 11 is defined as follows,

$$\text{Error} = \frac{|\rho_{DGM} - \rho_{FDM}|}{\rho_{FDM}}$$

where, $\rho_{DGM}$ is the density obtained using DGM and $\rho_{FDM}$ is that obtained using FDM. Comparing Fig. 10 and Fig. 11, the error increases along the blanch of the shock wave and the contact surface. This means that there is a small gap between the propagation of shock wave and the contact surface when using both DGM and FDM. In this study, the optimization of various parameters for DNN training (e.g., the number of DGM architecture layers $L$, the number of units $M$, batch size, the number of iterations, etc.) is insufficient when using DGM. Additionally, the FDM results depend on the number of grid points and the order of accuracy in a scheme.
Hence, it is essential to note that both DGM and FDM are difference from the exact solution. In any case, the difference of results between DGM and FDM may be reduced by optimizing and adjusting the parameters for training when using DGM, and a level of performance that satisfies the required accuracy will be obtained. However, a detailed error analysis of DGM will be an issue of future research.

3.3. Two-dimensional Navier-Stokes equations for supersonic flow around a blunt body

 Needless to say that computational fluid dynamics is especially useful in situations where multi-dimensional fluid behaviour is analyzed. Finally, here, the supersonic flow around a blunt body using two-dimensional Navier-Stokes equations is studied as an example of a more practical case. The governing equations targeted are formulated as follows.

\[
\begin{align*}
\frac{\partial \rho}{\partial t} + \rho \frac{\partial u_x}{\partial x} + \rho u_x \frac{\partial u_x}{\partial y} + \rho \frac{\partial u_y}{\partial y} + \frac{\partial \rho}{\partial y} + \rho u_x \frac{\partial u_y}{\partial y} + \rho \frac{\partial \rho}{\partial y} &= 0 \\
\rho \frac{\partial u_x}{\partial t} + \rho u_x \frac{\partial u_x}{\partial x} + \rho u_x \frac{\partial u_y}{\partial y} + \frac{\partial \rho}{\partial y} + \rho u_x \frac{\partial u_y}{\partial y} &= \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} \\
\rho \frac{\partial u_y}{\partial t} + \rho u_x \frac{\partial u_x}{\partial x} + \rho u_y \frac{\partial u_y}{\partial y} + \frac{\partial \rho}{\partial y} + \rho u_y \frac{\partial u_y}{\partial y} &= \frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \tau_{yy}}{\partial y} \\
\frac{\partial p}{\partial t} + u_x \frac{\partial p}{\partial x} + u_y \frac{\partial p}{\partial y} + \rho \left( \frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} \right) &= (\gamma - 1) \Phi
\end{align*}
\]

Here, a laminar flow is assumed, and the heat conduction term in the energy conservation equation is ignored for simplicity. As mentioned in the previous section, Eq. (20) is nondimensionalized by representative variables and is expressed as a non-conservation form for the same reasons. The viscous stress \( \tau_{ij} \) and dissipation function \( \Phi \) are formulated as follows.

\[
\begin{align*}
\tau_{xx} &= \frac{1}{Re} \left( 2 \frac{\partial u_x}{\partial x} - \frac{2}{3} \left( \frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} \right) \right) \\
\tau_{xy} &= \frac{1}{Re} \left( \frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} \right) \\
\tau_{yy} &= \frac{1}{Re} \left( 2 \frac{\partial u_y}{\partial y} - \frac{2}{3} \left( \frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} \right) \right)
\end{align*}
\]

Here, \( Re \) is assumed to be a constant in the whole computational domain for simplicity.

Figure 12 shows the computational domain and working conditions used in the DGM analysis. The range of computational domain is set to \( 0 \leq x \leq 1, \ 0 \leq y \leq 1, \) and a blunt body such as a circular cylinder with a radius of \( r = 0.125 \) is placed at the center of the computational domain \( x = 0.5, \ y = 0.5 \). The working fluid flows from the inlet boundary of \( x = 0 \) with the supersonic speed of Mach 3. The boundary conditions are indicated by the red lines in Fig. 12. The inlet boundary is set to the Dirichlet condition. At the internal boundary on the object surface, the flow velocity is set to zero, and the gradients of density and pressure for the normal direction to the object surface are set to zero. All of the other boundary conditions are assumed to be free-stream, so no particular conditions are set. The initial conditions are set equal to the inlet boundary conditions (i.e., \( \rho = 0.25, \ u_x = 1.0, \ u_y = 0.0, \) and \( p = \rho u_x^2/(\gamma M^2) = 0.0167 \)). The \( Re \) is set to fixed at 1,000. Based on the above, the loss function is formulated, although it is omitted here.

The numerical results using FDM (i.e., second-order TVD scheme) is also shown here to compare and validate the predicted results using DGM. For FDM, the conservation form of governing equations, Eq. (20), is converted to a generalized coordinate system, and the calculation is performed under the same working conditions (see Fig. 12) as DGM. The number of grid points is set to \( 150 \times 500 \) in \( r-\theta \) coordinates. Figures 13, 14 and 15 show the density, absolute value of velocity, and pressure distributions obtained using DGM and FDM at \( t = 1 \), respectively. The upper half from the dashed line in each figure shows the results using DGM, and the lower half shows the results using FDM. In the supersonic flow around a blunt body, a bow-shock is generated in front of the object (i.e., upstream side), and the density and pressure increase. As shown in the figures, there are slight differences in the position of the shock-wave front (i.e., shock stand-off distance) and the density around the object. The location of the shock stand-off distance and the steepness of the discontinuity surface strongly depend on the com-
putational accuracy of the FDM numerical scheme and the number of grid points. In this paper, a detailed error analysis of DGM and FDM was not sufficiently conducted, but it was proven that the overall flow field is roughly consistent between DGM and FDM, as shown in the figures. From these results, DGM can be effectively used to analyze two-dimensional Navier-Stokes equations, although more optimization and adjustment of the weight parameters in DNN are needed.

4. Conclusion

In order to examine the feasibility of the deep-learning-based method called Deep Galerkin Method (DGM) be used for compressible fluid analysis, DGM was extended and applied to the analysis of compressible Navier-Stokes equations. In DGM, a loss function for training a deep neural network is formulated so that differential operators, boundary conditions, and initial conditions of the targeted partial differential equation are satisfied. In this study, DGM was applied to two-dimensional Burgers equations as a preliminary study, one-dimensional Navier-Stokes equations for a shock tube problem, and two-dimensional Navier-Stokes equations for the supersonic flow around a blunt body. The approximate solutions obtained using DGM showed generally good agreement with the solutions obtained using FDM.

DGM tends to predict the solution effectively if the flow field has a low Reynolds number, such that the solution is sufficiently smooth. On the other hand, under the condition of a high Reynolds number (i.e., the influence of viscosity or diffusion terms is relatively small), remarkably steep discontinuities, such as shock waves, are generated. Therefore, in such a flow field, the training of DNN using DGM tends not to work well due to the difficulty in evaluating the steep discontinuities. It may be possible to solve flow fields under high Reynolds numbers by skillfully adding an artificial viscosity that acts only on steep discontinuities, as is often the case with computational fluid dynamics. Additionally, in this study, the optimization of hyperparameters in DNN was not sufficient. Considering the characteristics of DNN, steep discontinuities may be able to be expressed by increasing the number of hidden layers and units in the DNN. Furthermore, it is necessary to examine the case of three-dimensional space, the case including heat conduction and other terms, and the case of changing parameters such as viscosity coefficient, heat conduction coefficient, and Mach number.

Since the error analysis of DGM solutions is very important in this research, the DGM solutions were compared with FDM solutions in this paper. This is because the exact solutions of the Burgers equations and the Navier-Stokes equations are not available. However, it should be noted that the FDM solutions presented in this study are also only approximate solutions. The accuracy of FDM solutions depends on the precision of the selected numerical scheme and the number of grid points. Therefore, in this paper, it is only shown that the DGM and FDM solutions tend to be in good agreement under specific conditions. A detailed error analysis of the DGM solutions will be conducted in a future study.

References

1) Goodfellow, I., Bengio, Y., and Courville, A.: Deep Learning, The MIT Press, Massachusetts, 2016.
2) Tompson, J., Schlachter, K., Sprechmann, P., and Perlin, K.: Acceler-
ating Eulerian Fluid Simulation with Convolutional Networks, Proc. Mach. Learn. Res., 70 (2017), pp. 3424–3433.

3) Ladický, L., Jeong, S., Solenthaler, B., Pollefeys, M., and Gross, M.: Data-Driven Fluid Simulations Using Regression Forests, ACM Trans. Graph., 34 (2015), Article No. 199.

4) Kim, B., Azevedo, V. C., Thuerey, N., Kim, T., Gross, M., and Solenthaler, B.: Deep Fluids: A Generative Network for Parameterized Fluid Simulations, Comput. Graph. Forum, 38 (2019), pp. 59–70.

5) Brunton, S. L., Noack, B. R., and Kounoutsakos, P.: Machine Learning for Fluid Mechanics, Annu. Rev. Fluid Mech., 52 (2020), pp. 477–508.

6) Lee, H. and Kang, I. S.: Neural Algorithm for Solving Differential Equations, J. Comput. Phys., 91 (1990), pp. 110–131.

7) Dissanayake, M. W. M. G. and Phan-Thien, N.: Neural-Network-Based Approximations for Solving Partial Differential Equations, Comm. Numer. Meth. Eng., 10 (1994), pp. 195–201.

8) Lagaris, I. E., Likas, A., and Fotiadis, D. I.: Artificial Neural Networks for Solving Ordinary and Partial Differential Equations, IEEE Trans. Neural Netw., 9 (1998), pp. 987–1000.

9) Smaoui, N. and Al-Enezi, S.: Modelling the Dynamics of Nonlinear Partial Differential Equations using Neural Networks, J. Comput. Appl. Math., 170 (2004), pp. 27–58.

10) Sirignano, J. and Spiliopoulos, K.: DGM: A Deep Learning Algorithm for Solving Partial Differential Equations, J. Comput. Phys., 375 (2018), pp. 1339–1364.

11) Kiefer, J. and Wollowitz, J.: Stochastic Estimation of the Maximum of a Regression Function, Ann. Math. Statist., 23 (1952), pp. 462–466.

12) Hochreiter, S. and Schmidhuber, J.: Long Short-Term Memory, Neural Comput., 9 (1997), pp. 1735–1780.

13) Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Greg, S., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jozefowicz, R., Jia, Y., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Schuster, M., Monga, R., Moore, S., Murray, D., Olah, C., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. and Zheng, X.: Tensorflow: Large-Scale Machine Learning on Heterogeneous Systems, 2015, https://www.tensorflow.org (accessed 2021/10/18)

14) Kingma, D. P. and Ba, J.: Adam: A Method for Stochastic Optimization, arXiv:1412.6980v9 [cs.LG], 2015, http://arxiv.org/abs/1412.6980v9.

15) Glorot, X. and Bengio, Y.: Understanding the Difficulty of Training Deep Feedforward Neural Networks, Proc. Mach. Learn. Res., 9 (2010), pp. 249–256.

16) Yee, H. C.: A Class of High-Resolution Explicit and Implicit Shock Capturing Methods, NASA TM-101088, 1989.

Keisuke Sawada
Associate Editor