Turbulent Combustion Modeling Assisted by Deep Learning

YAN, WUDI
九州大学大学院総合理工学府総合理工学専攻機械・システム理工学メジャー

Turbulent Combustion Modeling Assisted by Deep Learning

A Master Dissertation

By

YAN WUDI

Department of Mechanical and Systems Engineering

Interdisciplinary Graduate School of Engineering Sciences

Kyushu University

2023

Turbulent Combustion Modeling Assisted by Deep Learning

By

YAN WUDI

Submitted in Partial Fulfilment of the Requirements

For the Degree

Master of Engineering

Supervised by

Professor Hiroaki Watanabe

Department of Mechanical and Systems Engineering

Interdisciplinary Graduate School of Engineering Sciences

Kyushu University

2023

TABLE OF CONTENTS

ABSTRACT

With the rise of computer technology, artificial intelligence technology is also becoming increasingly mature. The neural network fitting technology under deep learning has become a substitute for large scale database due to its high fitting accuracy and small fitting result memory.

We have focused on the reforming of Hot Coke Oven Gas (HCOG), COG is a common by-product of the steel industry and has been produced in each of the last ten years around 75 Mtoe. At present, most of COG is not used effectively and is just emitted directly, so if COG is used effectively, energy savings can be achieved.

Due to the complex composition of crude COG, the size of HCOG database used for simulation is also large. We hope to use neural networks to fit the HCOG database, in order to replace the HCOG database with smaller fitting results for simulation calculations, which can save CPU memory and improve the accuracy of the model.

In Chapter 1, A concise introduction was given to the development history of artificial intelligence technology, especially the application of deep learning as a branch. In addition, the background and necessity of Hot Coke Oven Gas (HCOG) application were introduced. Finally, the application of neural networks in fitting databases was introduced.

In Chapter 2, it mainly introduces the algorithm of dealing with Loss function. Starting with the linear regression problem, the Gradient descent method and the least square method are introduced at first. Then the Nonlinear regression problem is introduced. In order to solve the limitation of the least square method in dealing with nonlinear problems, the Gauss-Newton algorithm is introduced. Finally, the Levenberg–Marquardt (LM) algorithm, which combines the Gradient descent method and the Gauss-Newton algorithm is introduced. This is the core algorithm of this paper.

In Chapter 3, it introduces the computational rules between neurons, Forward Propagation and Backward Propagation in neural networks. Combining visual neural network diagrams has deepened our understanding of the LM algorithm. In addition, based on the order difference of magnitude, 21 output data were divided into three groups of Group A, Group B and Group C, laying the foundation for the analysis of data fitting errors in Chapter 4.

In Chapter 4, the fitting results of six representative output values *T*, *ENTH*, *Y_H2*, *Y_CO*, *Y_C2H2* and *Y_C6H6* from three groups were introduced, and the differences and fitting characteristics of Group A, Group B and Group C data were analyzed.

In Chapter 5, the summary and conclusions are presented.

NOMENCLATURE

Below are descriptions of the symbols used in this thesis.

η	Learning rate
$\nabla_{x_i} f$	Gradient vector of f at x_i
ϵ	Tiny increase
L	Loss function
J	Jacobian matrix
H	Hessian matrix
r	Residual between observed and theoretical value
$\mathcal{Z}$	Mixture fraction
ZV	Variance of mixture fraction
C	Progress variable
h^n	The normalized enthalpy defect
$LAMBDA$	Thermal conductivity
C_P	Specific heat capacity
$DIFF$	Thermal diffusivity
RHO	Density
$VISC$	Viscosity
$WMIX$	Molar mass of mixture
$ENTH$	Specific enthalpy
T	Temperature
Y_H2	Mass fraction of H_2
Y_OH	Mass fraction of OH

Y_H2O Mass fraction of H_2O

Y_O2 Mass fraction of O_2

Y_CO2 Mass fraction of CO_2

Y_CO Mass fraction of CO

Y_CH4 Mass fraction of CH_4

Y_C2H2 Mass fraction of C_2H_2

Y_C6H6 Mass fraction of C_6H_6

Y_C10H8 Mass fraction of $C_{10}H_8$

Y_C9H8 Mass fraction of C_9H_8

Y_C14H10 Mass fraction of $C_{14}H_{10}$

W_C Reaction rate of Progress variable

LIST OF FIGURES

Chapter 3

Chapter 4

LIST OF TABLES

Chapter 1

Introduction

1.1 The Development of Deep Learning

In 1943, Warren S. McCulloch and Walter Pitts built the first neural network based on the structure and working principles of biological neurons and proposed the McCulloch-Pitts (MCP) mathematical model [1]. In 1986, Geoffrey Hinton proposed the Backpropagation (BP) algorithm, which is now the most basic and widely used method for non-linear processing. In 2012, Geoffrey Hinton's group won the ImageNet image recognition competition with the Convolutional Neural Network (CNN) they built. From that time deep learning or machine learning has once again become popular. From web searches to content filtering on social networks to recommendations on e-commerce websites, and it is increasingly present in consumer products such as cameras and smartphones [2]. Examples include image recognition and speech-to-text, which are applications that are already integrated into our daily lives. It is fair to say that we are no strangers to deep learning.

As its name suggests, the most important feature of deep learning is that it enables autonomous learning. In other words, in jobs involving repetitive nature, we can summarize the iterative laws and set up error checking methods, then the computer can automatically perform the operations and reduce the error as expected. For activities where the number of calculations is excessive and the process is tedious, deep learning can guide the computer to perform this act instead of the human.

The difference from mere computer computing is that deep learning gives computers the ability to self-correct errors. The larger the number of samples, the smaller the error in fitting and predicting the data will be. This process eliminates the need for manual error reduction, thus making large-scale, high-volume automated computing possible. It gives the computer a certain level of intelligence, rather than just a computing tool that requires manual commands at every step.

Deep learning belongs to a branch under artificial intelligence, which also has several directions such as image recognition, speech recognition and data fitting. This article only focuses on the part of large-scale database fitting to introduce the application of artificial intelligence. As computer performance continues to improve and more and more deep learning techniques become possible, it is believed that computers will become even smarter in the future.

1.2 Background of HCOG

Energy is a major pillar of human society, but with the emergence of environmental issues, green will be the theme of future energy development. In order to reduce carbon emissions while making the most efficient use of resources, a by-product called Coke oven gas (COG) has come into view. COG is a common by-product of the steel industry and has been produced in each of the last ten years around 75 Mtoe [3]. At present, most of COG is not used effectively and is just emitted directly, so if COG is used effectively, energy savings can be achieved. This is a partial oxidation process, and we hope to convert the Tar, CH_4, and other substances in the crude COG into CO and H_2.

COG is made up of a variety of chemical components, such as hydrocarbons and carbon oxides. This also means that the results of COG data analysis are large and complex. When doing simulations, the calculations are usually done with a workstation server. Firstly, we have to import the database into the CPU before we can proceed to the next step of the calculation. However, as the accuracy requirements of simulation increase, so do the requirements for the dimensionality and number of sampling points of the database. This means that the size of the database can be as small as a few gigabytes to tens or even hundreds of gigabytes. the problem with an overly large database is that the CPU has a limited capacity and when the data is too large the CPU does not have extra capacity for the calculation. the CPU limitation puts a limit on the maximum capacity of the current data. The CPU capacity

becomes a bottleneck when we want to further increase the computational accuracy of the model.

One idea to solve this problem is to use neural network fitting techniques under deep learning to replace the database. In my own experience, a 4-dimensional, 21-output database of 2.29 GB stored in binary format was replaced by a matrix of 1.69 MB rebuilt in Fortran after neural network fitting. the difference in size was a staggering 1300 times greater. This means that for a database of a hundred gigabytes or more, the size of the matrix after fitting would also be under 100 megabytes. With acceptable error accuracy, this provides a new way of thinking for scenarios where it is computationally difficult to use a large capacity database.

1.3 Application of neural network fitting

Neural network fitting is an application branch at the very end of deep learning, which is used to do the fitting of non-linear regression. For linear regression problems, the fitted straight line can be solved quickly and accurately using gradient descent or least squares because of the simple linear relationship between the independent and dependent variables. However, for non-linear regression problems, the relationship between the independent and dependent variables is non-linear, which means that the relationship is more complex and a simple linear model cannot meet the fitting requirements. In addition, non-linear regression still suffers from overfitting, and while the model performs well on the training data, the accuracy drops dramatically when it comes to predictions for future data. This means that the computer is simply memorizing the data, not learning patterns in the data. Over-fitting is generally caused by the fitted model being too weak against noise, which means that the model also should have some ability to filter the noise during the fitting process.

Neural network fitting is a good tool for dealing with non-linear regression problems. For example, the most widely used back propagation algorithm, after forward propagation to calculate the bias and weights, by analyzing the error of the neural network output value and the sample value back in turn can itself modify the network parameters, to achieve the self-modification of the machine. At the same time, neural networks have a high degree of freedom and compatibility. When faced with different research samples, we can combine our experience to set up neural networks with different specifications to meet the needs of non-linear regression problems of varying complexity. Neural networks are robust in the training process and are able to handle situations containing noisy and incomplete data [4]. This also means that there will be better accuracy when predicting data outside of the training data.

Chapter 2

Algorithms for deep learning

There are two general approaches in solving linear regression problems. One is the gradient descent method, which uses iteration to make the calculated values slowly approach the target value. In this process, the learning rate needs to be set to control the convergence rate of the model. Too fast a convergence rate may miss the global optimal solution [5], while too slow a convergence rate increases the training time and may also fall into the local optimal solution, which is the biggest drawback of the gradient descent method.

Another method is the least squares method, which determines the optimal parameters by minimizing the sum of squares of the residuals between the actual values of the observed data and the predicted values of the model, which means that it does not need to do iterations and the computational process is simple, requiring only one computational step to obtain the result. In addition, the least squares method must be able to find the global optimal solution.

However, for nonlinear regression problems, the least squares method cannot be used for calculation and the gradient descent method tends to fall into local optimum solutions in the face of complex functional relationships. I will introduce two new algorithms based on the above two methods:

(i) A backpropagation algorithm (BP) with gradient descent as the core algorithm. This is one of the most common algorithms applied in fitting models to neural networks. It uses the gradient descent method to combine the error of the network output value with the sample value to in turn correct the network parameters in the next iteration cycle, thus optimizing the network and reducing the fitting error.

(ii) In solving linear problems, the least squares method can compute the full optimal solution directly without iterations, which actually presents a good idea. The Levenberg-Marquardt algorithm (LM) is an optimization algorithm for nonlinear least squares problems that combines the gradient descent method and the least squares method. It approximates the nonlinear model by linearizing it in each iteration so that the idea of least squares can be applied to nonlinear regression problems.

2.1 Linear regression solution

The neural network fitting problem is essentially a regression problem. We want to be able to describe the relationship between the independent and dependent variables in images or matrices without knowing the functional relationship between them, so that we can predict future data or test data that are not in the sample.

The linear regression problem is the simplest regression problem, and I will use it as an example to introduce the gradient descent and least squares methods, thus setting the stage for more complex algorithms to be introduced later.

2.1.1 Gradient descent method

The core idea of the gradient descent method is that the function changes fastest along the direction of the gradient, then the direction of the inverse gradient is the direction of the fastest decrease in the value of the function. If we summarize the error function of the calculated value and the sample value, we then find the gradient direction of this error function, and as long as we adjust the parameters along the inverse direction of the gradient, then the calculated value after each iteration will be closer and closer to the sample value,

and after enough iterations of calculation, the calculated value will be wirelessly close to the sample value.

Expressed in the equation as:

$$x_{i+1} = x_i - \eta \nabla_{x_i} f \qquad (\eta > 0) \tag{2.1}$$

The convergence condition is

$$\|f(x_{i+1}) - f(x_i)\| < \epsilon \quad \text{or} \quad \left\|\nabla_{x_i} f\right\| \approx 0 \tag{2.2}$$

Here, $f(x_1, x_2, \dots, x_n)$ is differentiable in the domain of definition, $\nabla_{x_i} f$ is the gradient vector of f at x_i, η is the learning rate, ϵ is a tiny increase in x, here it is used as a threshold to determine the degree of regression.

The derivation process is a first-order Taylor expansion of $f(x)$ at x:

$$where \qquad x = [x_1, x_2, \dots, x_n]^T \tag{2.3}$$

$$\begin{aligned} f(x + \epsilon) &= f(x) + \epsilon^T \nabla_x f + o(\|\epsilon\|) \\ &\approx f(x) + \epsilon^T \nabla_x f \end{aligned} \tag{2.4}$$

$$where \qquad \nabla_x f = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n}\right]^T \tag{2.5}$$

Eq. (2.4) describes that when x increases ϵ, $f(x)$ increases $\epsilon^T \nabla_x f$ which means the inner product of ϵ and $\nabla_x f$. If the modulus length of ϵ is limited to a fixed value (in the actual regression problem, ϵ is the value we need to assign ourselves), then $\epsilon^T \nabla_x f$ will obtain a minimum value (negative value) as long as the direction of ϵ opposite to $\nabla_x f$.

Therefore:

$$\begin{aligned} \epsilon &= -\eta \nabla_x f \qquad (\eta > 0) \\ x_{i+1} &= x_i - \eta \nabla_{x_i} f \qquad (\eta > 0) \end{aligned} \tag{2.6}$$

Where η is a very small scalar that does not affect the direction, it limits the step size in each iteration step and is generally referred to as the learning rate.

Eq. (2.6) describes the direction of the iteration, then makes the whole operation process form a closed loop. Eq. (2.2) lists two commonly used conditions for iterative termination:

(i) When the first-order derivative is zero and the function achieves the extreme value point.

(ii) When the error between the calculated value and the sample value is within an acceptable range.

It should be noted that the learning rate η is a very important parameter that controls the step size of each parameter update, as can be seen from Eq. (2.6). If the learning rate is set too small, the number of iterations and the time spent before the optimal solution is obtained will be longer. If the learning rate is set too large, then the magnitude of each parameter change becomes larger, which means that the optimal solution is likely to be missed. Therefore, the learning rate setting will directly affect the accuracy of the operation. In addition to manually adjusting the learning rate based on computational experience, there are some adaptive learning rate optimization algorithms, such as Adam and AMSGrad [6], which can automatically adjust the learning rate to obtain better convergence performance.

2.1.2 Least squares method

The least squares method is a computational method for solving linear problems. Compared with the gradient descent method, it does not require a tedious iterative process. It determines the parameter values by minimizing the sum of squared residuals (the sum of squared residuals is the loss function in deep learning) to achieve the best fit. I think it is the easiest way to solve linear problems.

The expression of the loss function L is:

$$L = \sum (Observed\ value\ -\ theoretical\ value)^2 \tag{2.7}$$

Gauss developed a theory of error analysis, thus proving that it is indeed the case that the model estimated with the smallest sum of squares of errors is the closest to the real situation. Please see the proof for yourself if you are interested.

The derivation of the least squares method is shown below.

$$h_\theta(x_1, x_2, \dots, x_n) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n \tag{2.8}$$

$$h_\theta(X) = X\theta \tag{2.9}$$

Then the loss function is:

$$L(\theta) = \frac{1}{2}(X\theta - Y)^T(X\theta - Y) \tag{2.10}$$

Where Y is the theoretical value.

The minimum can be solved by taking the first order derivative of the loss function and making it equal to 0:

$$\frac{\partial L(\theta)}{\partial \theta} = 0 \tag{2.11}$$

$$X^T X\theta - X^T Y = 0 \tag{2.12}$$

$$\theta = (X^T X)^{-1} X^T Y \tag{2.13}$$

Eq. (2.13) holds provided that X is an orthogonal matrix. From Eq. (2.13), we know that X and Y are known quantities, bring them into Eq. (2.8) to find the value of θ.

2.2 Non-linear regression solution

The relationship between the independent and dependent variables of a linear regression problem is linear and can be represented by a linear equation. However, for nonlinear regression problems, the relationship between the dependent and independent variables is nonlinear, meaning that the complexity of the model is greatly increased, and it may be necessary to use power functions, logarithmic functions, and other forms to describe the relationship between the variables.

Above we mentioned two basic methods for solving linear problems, the gradient descent method and the least squares method.

Firstly gradient descent method performs well in dealing with linear regression problems, this is because linear regression models are convex optimization problems [7], and it is always possible to find the global optimal solution. However, when dealing with nonlinear regression problems, the gradient descent method will possibly fall into a local optimal solution if the model is non-convex optimized. Moreover, as the complexity of the model increases, the number of iterations will be greatly increased due to the step size limitation, which means the computation time will be greatly increased.

Facing this situation, the gradient descent method can be improved to optimize the model, such as Stochastic Gradient Descent method (SGD) [8] and Batch Gradient Descent method (BGD) [9], and they can all accelerate the convergence speed of the model and improve the model accuracy. In the following, I will introduce the backpropagation method (BP) with gradient descent as the core to illustrate one solution to the nonlinear regression problem.

The least squares method makes it a simple algorithm when dealing with linear regression problems because it does not require iterations and computational calculations in one step. Also, compared with iterative computation, numerical computation can find the global optimal solution without worrying about getting into a local optimal solution. However, it is clear from Eq. (2.8) that the least squares method is essentially a linear method and is not directly applicable to nonlinear regression models.

In this case, we need to change our thinking. Is there any way we can transform the nonlinear problem into a linear regression problem and then solve it by least squares method? Gauss-Newton algorithm is an optimization algorithm applied to nonlinear least squares problems. It mainly optimizes the model parameters step by step by iteration to obtain the optimal solution, and I will introduce it in detail in the following. In addition, there are also optimization algorithms such as the Quasi-Newton Method [10], Trust Region Method [11], etc. Please check them yourself if you are interested.

2.2.1 Jacobian and Hessian matrix

The Gauss-Newton algorithm is a summary of the Gauss method and Newton method for and only for solving nonlinear least squares problems. It uses the Jacobian matrix to approximate the Hessian matrix to achieve fast convergence, approximates the nonlinear problem as a linear problem, performs local linear approximation in each iteration, and gradually optimizes the parameters to reduce the residual sum of squares.

Before the introduction of the Gaussian-Newton algorithm, I will first introduce the Jacobian and Hessian matrices. Jacobian matrix is a matrix in which the first order partial derivatives of the functions are arranged in a certain way.

For a nonlinear function $f(x) = [f_1(x), f_2(x), \dots, f_m(x)]$ with m output variables and n input variables, where $x = [x_1, x_2, \dots, x_n]$ is a vector of input variables and the Jacobian matrix J is an $m \times n$ matrix whose elements consist of the partial derivatives of the functions:

$$\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \tag{2.14}$$

Where $\frac{\partial f_i}{\partial x_j}$ denotes the partial derivative of the function f_i with respect to the variable x_j. From Eq. (2.14), the Jacobian matrix is a transpose of the gradient vector of the multivariate function and represents the rate of change of the function in the direction of each variable. The computation of the gradient vector is the most crucial part of both the gradient descent method and the Gaussian Newton algorithm, which reflects the trend of change, and the change of the first-order derivative, which is related to the trend of change of the function value, which determines the direction of update of the parameters. At the same time, the Jacobian matrix provides a local linear approximation of the function at a point. Near a given point, a nonlinear function can be approximated by a Taylor expansion to a linear function. Although this linear function is accurate only near the given point, it provides an idea of approximating the solution locally, describing the behavior of the function near the given point.

Similar to the Jacobian matrix, the Hessian matrix is a square matrix composed of second order derivatives. By analyzing the eigenvalues of the Hessian matrix, we can determine the curvature of the function at a point and the concavity.

For a nonlinear function $f(x) = [x_1, x_2, \dots, x_n]$, if all second-order partial derivatives of $f(x)$ exist and are continuous in the domain of definition, the Hessian matrix H is an $n \times n$ matrix whose elements consist of the second-order partial derivatives of the functions:

$$\begin{bmatrix} \frac{\partial^2 f}{\partial {x_1}^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial {x_2}^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial {x_n}^2} \end{bmatrix} \tag{2.15}$$

Where $\frac{\partial^2 f}{\partial x_i \partial x_j}$ denotes the second-order partial derivative of the function f with respect to the variable x_i, x_j. The eigenvalues of the Hessian matrix can determine the extreme values of multivariate functions:

(i) If it is a positive definite matrix, there is a local minimum at the critical point.
(ii) If it is a negative definite matrix, there is a local maximum at the critical point.
(iii) If it is an indefinite matrix, it is not an extreme value at the critical point.

From Eq. (2.15), The Hessian matrix provides information about the curvature and second-order derivatives of a function, which is important for finding the extreme value point of a function and for the optimization of an algorithm. However, as the dimensionality of the

data increases, the computational effort increases significantly. Some of the algorithms described below, such as the Gaussian-Newton method, will actively avoid computing the Hessian matrix thus reducing the computational effort.

Jacobian and Hessian matrices they are not algorithms, but simply matrices that contain information about the first and second order partial derivatives of a function. Both Jacobian and Hessian matrices have applications in neural network fitting, where they can be used as a basis for adjusting step size, learning rate, or other grid parameters. They make the matrix calculation and expression of the algorithm easy to understand

2.2.2 Gauss-Newton algorithm

From Eq. (2.8), it is clear that the least squares method cannot be used for the computation of nonlinear problems. Gaussian-Newton method is an improved algorithm of Newton method. Newton method finds the minimal value point by the Hessian matrix of the sum of squared residuals, which is equivalent to an optimization algorithm of least squares, and applies the least squares method that deals with linear regression problems to nonlinear regression problems. However, the Hessian matrix of complex models can be very computationally intensive, which leads to long computation time of iterations. To solve this problem, Gaussian-Newton method was developed based on Newton method, which does not require the computation of Hessian matrix compared to Newton method, though at the cost that it is used for and only for solving nonlinear least squares problems.

It is clear from Eq. (2.7) that the core of the least squares method is to find the minimum value of the sum of squares of the residuals. The sum of squared residuals $r(x_i)$ of the loss function $L(\theta)$ is:

$$L(\theta) = \frac{1}{2}\sum_{i=0}^{m} r(x_i)^2 \tag{2.16}$$

Solving the minimum of $L(\theta)$ using Newton method requires calculating its gradient vector with the Hessian matrix.

$$\nabla_\theta L = \sum r_i \frac{\partial r_i}{\partial \theta}$$

$$= [r(x_1), r(x_2), \dots, r(x_m)] \begin{bmatrix} \nabla_\theta r(x_1)^T \\ \nabla_\theta r(x_2)^T \\ \vdots \\ \nabla_\theta r(x_m)^T \end{bmatrix} \tag{2.17}$$

where the Jacobian matrix is:

$$J_r(\theta) = \begin{bmatrix} \nabla_\theta r(x_1)^T \\ \nabla_\theta r(x_2)^T \\ \vdots \\ \nabla_\theta r(x_m)^T \end{bmatrix} \tag{2.18}$$

Therefore, Eq. (2.17) can be written as:

$$\nabla_\theta L = r^T J_r \tag{2.19}$$

Where,

$$r = [r(x_1), r(x_2), \dots, r(x_m)]^T \tag{2.20}$$

It is clear from Eq. (2.15) that the Hessian matrix is:

$$H = \sum \left[r_i \frac{\partial^2 r_i}{\partial \theta^2} + \left(\frac{\partial r_i}{\partial \theta}\right)\left(\frac{\partial r_i}{\partial \theta}\right)^T \right] \tag{2.21}$$

Since $r_i \approx 0$, Eq. (2.21) can be approximated as:

$$H \approx \sum \left[\left(\frac{\partial r_i}{\partial \theta}\right)\left(\frac{\partial r_i}{\partial \theta}\right)^T \right]$$

$$= J_r^T J_r \tag{2.22}$$

Bringing the gradient vector and the Hessian matrix (approximation) into the Newton method root formula [12], the Gauss-Newton algorithm iteration is obtained as:

$$\theta_i = \theta_{i-1} - (J_r^T J_r)^{-1} J_r^T r \tag{2.23}$$

The essential difference between the Gauss-Newton method and the Newton method can be seen in Eq. (2.22). For the least squares problem, the residuals are very small, so the Gaussian Newton method discards $r_i \frac{\partial^2 r_i}{\partial \theta^2}$, thus replacing the Hessian matrix calculation with the Jacobian matrix. Also, this explains why the Gaussian Newton method is applicable and only applicable to the least-squares nonlinear regression problem.

2.2.3 Levenberg–Marquardt algorithm

As can be seen from the previous section, the gradient descent method and the Gauss-Newton algorithm have their own advantages and disadvantages in solving nonlinear regression problems. The gradient descent method uses the first-order derivative matrix for computation, so it can move quickly in the direction of gradient descent. However, the convergence speed becomes slower when the optimal solution is approached. On the contrary, although the Gauss-Newton algorithm is calculated by replacing the Hessian matrix with the Jacobian matrix approximation, it is still essentially calculated by using the second-order derivative matrix, and the convergence speed of the Gauss-Newton algorithm will be faster when the optimal solution is approached, and approach the optimal solution in a more accurate way. This is because both the gradient descent method and the Gauss-Newton algorithm are derived from Taylor expansions, and the accuracy of the second-order expansion is higher than that of the first-order expansion. Thus, it can be seen that the gradient descent and Gauss-Newton algorithm each have their own advantages in finding the direction of gradient descent and approaching the optimal solution. So, is there an algorithm that takes both of them into account?

The Levenberg-Marquardt (LM) algorithm is an optimization algorithm for nonlinear problems that combines Gauss-Newton algorithm and gradient descent method, as mentioned above. It was published in 1944 by Kenneth Levenberg, and was rediscovered in 1963 by Donald Marquardt. By looking at Eq. (2.6) and Eq. (2.23), we can find some similarity in the form of the iterative formulas of the gradient descent method and the Gauss-Newton algorithm. The expression of LM algorithm is:

$$\theta_i = \theta_{i-1} - (J_r^T J_r + \mu I)^{-1} J_r^T r \tag{2.24}$$

Compared with Eq. (2.23), Eq. (2.24) just has one more term μI, μ is called damping factor and I is the identity matrix. I is introduced to add a non-negative constant value to the eigenvalues of the Hessian matrix (or an approximation of the Jacobian matrix), thus ensuring that the matrix is positive definite (the quadratic form corresponding to the positive definite matrix has convexity). And μ must be greater than zero to ensure that the iterations proceed in the direction of gradient descent. As μ approaches zero, Eq. (2.24) approximates Eq. (2.23), which is close to the Gauss-Newton algorithm. When μ is large, Eq. (2.24) approximates Eq. (2.6), it is close to the gradient descent method. In a general regression problem, a large μ is used in the initial stage so that the optimal solution can be approached quickly by using the

nature of the gradient descent method that advances rapidly in the direction of gradient descent, and the parameters are adjusted in the process, and μ decreases gradually. Near the optimal solution, as μ is close to zero, it is converted to a Gauss-Newton algorithm, which approximates the optimal solution in a small range.

The LM algorithm is the preferred algorithm when facing general fitting problems. This is because it can combine both accuracy and fitting speed. In the fitting examples in this paper, the LM algorithm is used for most of the cases.

Chapter 3

Experimental configuration and computational setup

This chapter focuses on the structure of the target database. For neural network fitting, the data lose the physical meaning behind it and become mere numbers. Therefore, it will become important to analyze the effect of order-of-magnitude differences in the data on the fitting error. Also, this chapter will illustrate how to determine the neural network scale. The main core idea of this chapter is to introduce how to combine theoretical algorithms with experimental operations, a transition from abstract mathematical formulas to intuitive neural networks.

(i) Grouping the data by order of magnitude differences sets the stage for the next chapter that discusses the relationship between order of magnitude and fitting error.

(ii) Analyze the design ideas of neural network fitting size, and provide reference for determining the size of neural network when facing different databases. Some data pre-processing and post-processing methods will also be introduced to improve the accuracy of the fitting results.

(iii) Combine the iterative process with the algorithm description in the previous chapter to analyze the whole neural network fitting process more intuitively.

3.1 Structure of the target database

The need for HCOG applications was mentioned earlier, and we expect to use neural networks to fit the HCOG database, thus building smaller sized neural networks instead of the database itself. All the neural network input and output quantities are represented as matrices. It is important to note that even though the data have their physical meaning, they are pure numbers for neural network, so it must be noted that the data of the same physical quantity are in the same row of the matrix instead of the same column, otherwise the order of the physical quantities will be completely disrupted and the fitting results will be meaningless and a waste of time.

The size of the target database is 2.29 GB, with 4 control quantities (input data), 21 dependent variables (output data), and a total of 2.98 million rows of data, which are present in Table in binary format. Table 3.1 shows the data composition and range of the target database.

Table 3.1 Composition of the target database

Projects	**Minimum**	**Maximum**
Input data		
Z	0.00	1.00
ZV	0.00	0.25
C	0.00	1.00
$\boldsymbol{h^n}$	0.00	1.00
Output data		
$\boldsymbol{LAMBDA}$	2.54E-02	3.69E-01
$\boldsymbol{C\_P}$	9.17E+02	3.12E+03

$\boldsymbol{DIFF}$	2.13E-05	1.81E-03
$\boldsymbol{RHO}$	7.93E-02	1.29
$\boldsymbol{VISC}$	2.07E-05	9.49E-05
$\boldsymbol{WMIX}$	1.41E+01	3.20E+01
$\boldsymbol{ENTH}$	-6.73E+06	1.67E+03
$\boldsymbol{T}$	3.00E+02	2.94E+03
$\boldsymbol{Y\_H2}$	0.00	6.61E-02
$\boldsymbol{Y\_OH}$	0.00	6.75E-02
$\boldsymbol{Y\_H2O}$	0.00	4.73E-01
$\boldsymbol{Y\_O2}$	0.00	1.00
$\boldsymbol{Y\_CO2}$	0.00	2.67E-01
$\boldsymbol{Y\_CO}$	0.00	4.70E-01
$\boldsymbol{Y\_CH4}$	0.00	2.14E-01
$\boldsymbol{Y\_C2H2}$	0.00	2.83E-02
$\boldsymbol{Y\_C6H6}$	0.00	4.15E-02
$\boldsymbol{Y\_C10H8}$	0.00	6.48E-02
$\boldsymbol{Y\_C9H8}$	0.00	1.06E-02
$\boldsymbol{Y\_C14H10}$	0.00	1.26E-02
$\boldsymbol{W\_C}$	-1.49E-02	2.36E+04

From Table 3.1, we can see that the order of magnitude difference of the target database, the smallest is $VISC$, and the variation of its data range is also small. The largest order of

magnitude difference is W_C, and the order of magnitude difference reaches six. It is worth noting that the physical meaning of the mass fraction of each component is its residual amount in the reactor. Even if the reaction is fully carried out, it is not guaranteed that all chemical components are completely reacted and consumed. This means that for the mass fraction of each component, even if the minimum value is zero, there will be a significant amount of data that falls close to zero but with a non-zero value. In the case of the mass fraction of Oxygen (Y_O2), for example, in addition to the values that are zero, the values that are nearly close to zero reach an order of magnitude difference of eight, on the order of minus eight of ten, which is even larger than the order of magnitude difference of W_C. Therefore, the order of magnitude difference of the mass fraction of some components is actually the largest, which undoubtedly makes the neural network fitting much more difficult.

So why are neural network fitting results so sensitive to order-of-magnitude differences in the data? This is because the iterative process is still essentially a matrix computation process, and the arithmetic process then requires a discussion of how many decimal places to retain. At present, in my operation, the accuracy is fully satisfied (within ten percent) if all matrix information is kept within four decimal places in the general case (order of magnitude difference within four). For a magnitude difference of eight like W_C, there is no point in discussing the retention of decimals, and more data preprocessing is needed to reduce the fitting error.

Depending on the order of magnitude of the data, I divided the 21 outputs into three groups, grouped as follows:

(i) Group A: those with an order of magnitude difference of two or less. They are *LAMBDA, C_P, DIFF, RHO, VISC, WMIX, ENTH, T,* and they have the simplest neural network fitting, and also the highest fitting accuracy, and the error can usually be controlled within ten percent.

(ii) Group B: mass fraction of those chemical components with an order of magnitude difference of five or four. As mentioned before, the order of magnitude differences of the mass fractions of the chemical components are not intuitive since the minimum values of the mass fractions are all zero. For the following mass fractions, *Y_H2, Y_OH, Y_H2O, Y_CO2, Y_CO, Y_CH4, Y_C9H8, Y_C14H10,* the results of the neural network fitting alone cannot meet the accuracy requirements. This is because the order of magnitude difference which is at four or five is beyond the neural network resolution. For group B, some pre-processing and post-processing of the data is required to keep the fitting error within ten percent.

(iii) Group C: the order of magnitude difference of the mass fraction of the chemical components is greater than five, they are *Y_O2, Y_C2H2, Y_C6H6, Y_C10H8,*

W_C, for the projects of Group C, the fitting is extremely difficult. Since most of the mass fractions have reached an order of magnitude difference of eight, insignificant fluctuations for large numbers, which are thousands of errors for small numbers. Therefore, the common pre-processing and post-processing means have failed. For Group C's projects, in most cases, specific data processing and neural network fitting plans are developed for the data structure characteristics of a particular project itself.

It is important to emphasize again that for neural network fitting, all of the data have lost their underlying physical meaning. Therefore, the order of magnitude difference of the data, the trend of the data in its value domain is the most important factor that affects the fitting error.

3.2 Computational setup of neural networks

Unlike algorithms, the computation of the neural network fitting process embodies iterative ideas everywhere. Since iteration is involved, it is necessary to understand the basic computational ideas and structure of neural networks, to determine the start and end conditions of iterative computation, and to understand the computational process of an iterative loop process. This section will introduce the structure of neural network, the setting of neural network scale and the principle of data transfer in a more intuitive way combined with the previous algorithm to fully introduce what is a neural network. Two important concepts will also be introduced: Forward Propagation and Backward Propagation, which are the soul of neural network fitting.

3.2.1 Components of Neural Networks

In the previous part, we introduced some basic algorithms, which are abstract and poorly understood, and remain only at the theoretical level. In this subsection, the structure of neural networks will be introduced, and the previously mentioned LM algorithm, for example, will be embedded in the structure itself to make the concept of neural networks more intuitive and easy to understand for the reader. The neural network is inspired by the neural network of the biological brain, and its structure is shown in Fig 3.1.

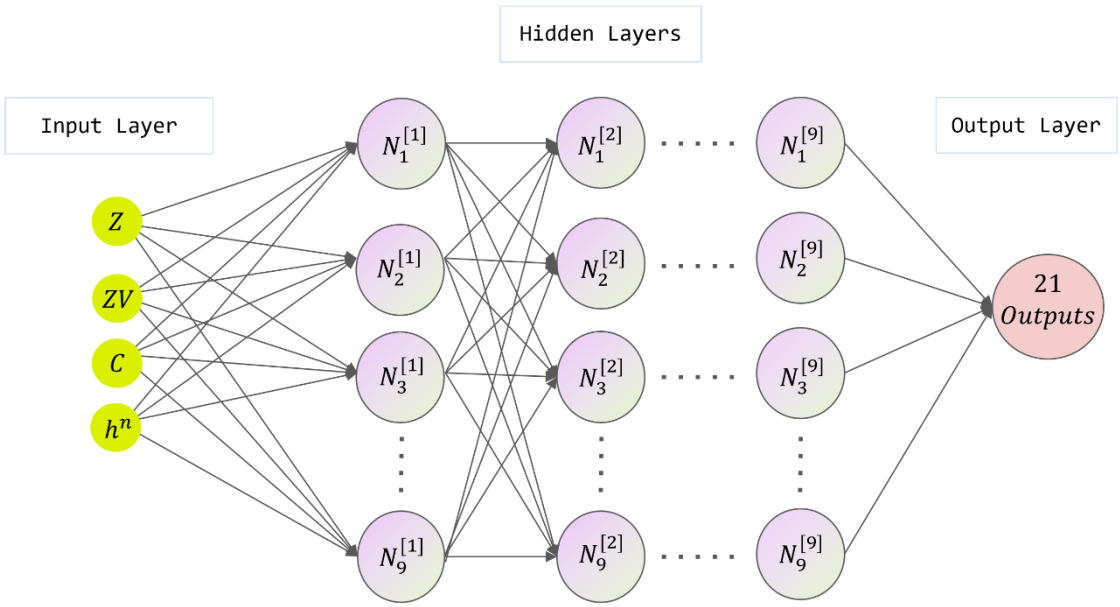

Figure 3.1 Neural network schematic

First of all, neural networks are composed of neural layers and neurons, it is analyzed structurally and divided into three layers: input layer, hidden layer and output layer. The input layer is the input data of the database, each input data is imported into the neural network in the form of a row matrix, and a column matrix composed of different input data constitutes a set of independent variables of the neural network involved in the calculation. Similar to the input layer, the output layer is composed of the output data from the database, and a matrix of columns of different output data forms a set of target values for the neural network.

The case of the hidden layer is a little different, and it is the focus of the neural network. Both the input and output layers serve the hidden layer by providing and receiving data

flowing from the hidden layer. Before introducing the hidden layer in detail, it is necessary to introduce the concept of neuron.

A neuron represents a computational unit, which can be understood as the smallest unit of a neural network. The computational path (in the direction of the arrow in the figure) performs a matrix operation each time it passes through a neuron. Two important parameters are also computed: weights and biases. (they are the purpose of the neural network fitting calculation, and the detailed calculation process will be described below). The essence of a neuron is a set of column vectors, which represent the result of the computation between two layers of the neural network (also understood as the output of the previous layer and the input of the next layer). In the input and output layers, the number of neurons and the number of the types of the input data output data are the same (known as features).

Let's return to the hidden layer, for which the number of neurons is considered set. Determine the number of neurons based on experience and the complexity of the structure of input and output data. There is no empirical formula for the number of neurons to refer to, and appropriate adjustments should be made based on the fitting accuracy results. For the calculation in this article, details are shown in Table 3.2, four input data and twenty-one output data were designed using a hidden layer of nine to thirteen layers and nine to thirteen neurons per layer. Too many neurons and too few neurons can pose some problems. Too many neurons will make the training time too long, and there is also a risk of overfitting, resulting in a poorly fitted network on new data. Too few neurons directly affect the fitting error and are an oversimplification of the neural network. Therefore, before formally doing the fitting analysis, it is necessary to review the relevant literature or conduct experiments with databases of the same size to determine the appropriate neural network size. In general, for complex data structures, prioritizing the appropriate number of hidden layers can improve the fitting accuracy [12].

Table 3.2 The scale of the neural network used in this paper

Projects	**Number**
Input data	4
Output data	21
Input layer	1
Output layer	1

Hidden layer	9 or 13
Neurons per hidden layer	9 or 13

3.2.2 Forward propagation

As mentioned before, the essence of neural network fitting is iterative computation. Then iterative computation is to form a computational cycle, we can set the number of cycles or fitting error as a marker for iterative stopping, so what is the whole process of closed operation of neural network fitting? If we simply compute from the input data to the output data, for the whole neural network, it just reads the input data, combines the bias and weight calculation and then 'spits' the data out again, this computation process is one-way, it does not constitute a cycle. Therefore, if we can use the error between the output data and the target data as a criterion to update the parameters in the neural network, wouldn't this constitute a cycle? Therefore, this part will introduce two important concepts: Forward propagation and Backward propagation, which are the core ideas of neural network fitting that can perform iterative operations.

As the name implies, Backward Propagation is the process by which input data passes through the hidden layer and is computed as output data. Before introducing the data transfer between the neurons, let's look at the following important concepts:

(i) Weights: the weight determines the degree of influence of the corresponding neuron of the previous layer on the neuron of the next layer, which controls the flow of input data between neurons and determines the sensitivity of the neuron to the corresponding data.

(ii) Bias: the bias acts on the activation threshold of the neuron. Since the weights are computed in the form of a product, while the bias is computed in the form of an addition, the calculation of bias is actually a nonlinear operation,, which improves the ability of the neural network to handle nonlinear problems.

(iii) Activation Function: it is essentially a nonlinear function, where the output of a neuron has to pass through an activation function before it can go to the next neuron. The activation function introduces nonlinear relationships so that the neural network can represent the intrinsic relationships of a complex model. Common activation functions are: Sigmoid function (Eq. (3.1)), Tanh function (Eq. (3.2)), ReLU function (Eq. (3.3)), etc.

$$S(x) = \frac{1}{1 + e^{-x}} \tag{3.1}$$

$$Tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3.2}$$

$$ReLU(x) = \max(0, x) \tag{3.3}$$

(iv) Loss Function: it indicates the difference between the output data of the neural network and the target data, which measures the accuracy of the neural network fit. The common loss functions are Mean Squared Error (Eq. (2.16)), Cross-Entropy, etc.

The data propagation schematic is shown in Figure 3.2. This model is a three-layer neural network model with four neurons. The nomenclature of neurons is referenced from *Neural Network and Deep Learning* [14]. Where, a_j^l is the activation value (the output value of the activation function) of the j^{th} neuron in l^{th} layer, b_j^l represents the bias of the j^{th} neuron in l^{th} layer, and w_{jk}^l is the weight of the k^{th} neuron in $(l-1)^{th}$ layer to the j^{th} neuron in l^{th} layer.

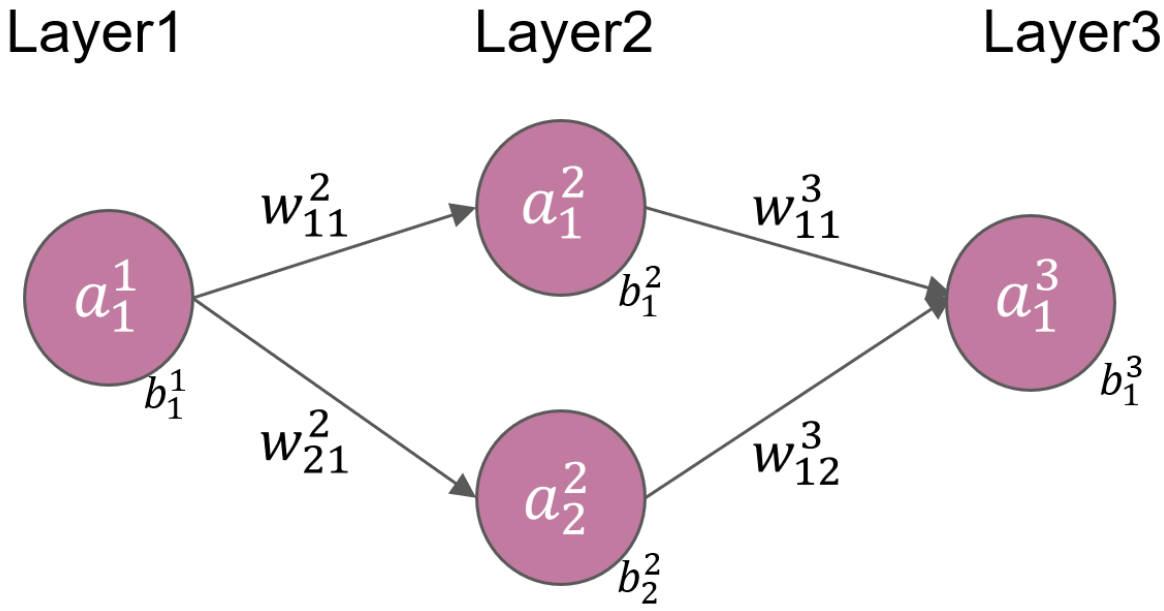

Figure 3.2 Data transfer between neurons

For example, the data transfer between layer 2 neurons to layer 3 neurons is

$$a_1^3 = L(w_{11}^3 a_1^2 + w_{12}^3 a_2^2 + b_1^3) \tag{3.4}$$

$$a_j^l = L\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right) \tag{3.5}$$

Eq. (3.5) is written in matrix form as:

$$a^l = L(w^l a^{l-1} + b^l) \tag{3.6}$$

Where, L is the loss function. According to Eq. (3.6), we write the weight input for each layer as *z*:

$$z^l = (w^l a^{l-1} + b^l) \tag{3.7}$$

$$a^l = L(z^l) \tag{3.8}$$

After understanding the important concepts of weights, biases and inter-neuron algorithms, the concept of Forward Propagation is well understood. Forward Propagation is the process of computing from the input layer to the output layer, and its purpose is to calculate the error between the output data (computed values) of the neural network and the target data.

As mentioned earlier, the nature of neural network fitting is a large-scale iterative operation. Iterative operations imply a complete closed-loop computational process, and the error between the output data and the target data is the evaluation criterion of an iterative cycle, and the quality of an iterative cycle is expressed by the size of the error. The purpose of Forward Propagation is simply to bring the new set of input data into the bias and weights of the previous iterative cycle to calculate the fitting error of the new set, and to prepare the correction of the bias and weights for this cycle.

3.2.3 Backward propagation

The error of the fitting is now known and it will be used as a reference for the correction of the weights and biases during the next iteration. The error is expressed by the loss function, with reference to Eq. (2.16). If we want to correct the

weights and biases using the errors between the output data and the target data, then we need to calculate $\frac{\partial L}{\partial w_{jk}^{l}}$ and $\frac{\partial L}{\partial b_{j}^{l}}$, against a small step in the direction of these two first-order partial derivatives of loss function, thus correcting the weights and biases.

According to the chain rule, the error equation for the output of neural network layer is:

$$\frac{\partial L}{\partial z_j^l} = \frac{\partial L}{\partial a_j^l}\frac{\partial a_j^l}{\partial z_j^l} = \frac{\partial L}{\partial a_j^l}L'(z_j^l) \tag{3.9}$$

Eq. (3.9) is written in matrix form as shown in the following equation:

$$\frac{\partial L}{\partial z_j^l} = \delta^l = \nabla_a L \odot L'(z^l) \tag{3.10}$$

According to Eq. (3.9), $\frac{\partial L}{\partial a_j^l}$ describes the variation of the loss function with the output value of the *j* neuron, while $\frac{\partial a_j^l}{\partial z_j^l}$describes how fast the output value of the loss function (neuron output value) changes with z_j^l. Since the loss function is known, according to Eq. (2.16), then $\frac{\partial L}{\partial a_j^l}$ is:

$$\frac{\partial L}{\partial a_j^l} = a_j^l - y_j \tag{3.11}$$

Where *y* is the target database value. Observing Eq. (3.9), it can be known that z_j^l can be described by the last layer ($(l+1)^{th}$ is now the previous layer of l^{th} in backpropagation). The derivation process is:

$$\delta_j^l = \sum_k \frac{\partial L}{\partial z_k^{l+1}}\frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \delta_k^{l+1}\frac{\partial z_k^{l+1}}{\partial z_j^l} \tag{3.12}$$

According to Eq. (3.5):

$$z_k^{l+1} = \sum_j w_{kj}^{l+1}a_j^l + b_k^{l+1} \tag{3.13}$$

$$z_k^{l+1} = \sum_j w_{kj}^{l+1}L\left(z_j^l\right) + b_k^{l+1} \tag{3.14}$$

So that:

$$\delta_j^l = \sum_k w_{kj}^{l+1}\delta_k^{l+1}\,L'(z_j^l) \tag{3.15}$$

Then, $\frac{\partial L}{\partial w_{jk}^{l}}$ and $\frac{\partial L}{\partial b_{j}^{l}}$ are respectively:

$$\frac{\partial L}{\partial w_{jk}^{l}} = \frac{\partial L}{\partial z_{j}^{l}} \frac{\partial z_{j}^{l}}{\partial w_{jk}^{l}} = \frac{\partial L}{\partial z_{j}^{l}} a_{k}^{l-1} = a_{k}^{l-1} \delta_{j}^{l} \tag{3.16}$$

$$\frac{\partial L}{\partial b_{j}^{l}} = \frac{\partial L}{\partial z_{j}^{l}} \frac{\partial z_{j}^{l}}{\partial b_{j}^{l}} = \frac{\partial L}{\partial z_{j}^{l}} = \delta_{j}^{l} \tag{3.17}$$

It is clear from Eq. (3.15) that we can calculate the error δ_{j}^{l} for l^{th}layers from the error δ_{j}^{l+1} for $(l+1)^{th}$ layers. In the actual calculation process, when we calculate the error between the computed result of the neural network fitting and the target database through Forward Propagation, we can calculate δ^{output} of the output layer. Gradually, the iterative calculation is carried out for the previous layer, and finally δ^{input} input layer is calculated.

This is the process of Backward Propagation, in which the weights and biases of each layer can be corrected through Eq. (3.16) and Eq. (3.17). Thus, after the Backward Propagation is completed once, every weight and bias in the neural network is updated. Forward Propagation and Back Propagation form a complete iterative loop. First a set of input data from the database is imported into the neural network to calculate the error of the set of fitting computed results with respect to the corresponding target database through Forward Propagation. The error is then utilized to iterate through Backward Propagation, updating the weights and biases of each layer and returning from the output layer to the input layer, thus forming a closed-loop loop. An artificially set number of cycles or an error minimum will be used as a flag to stop the iteration. The iterative process is shown schematically in Figure 3.3.

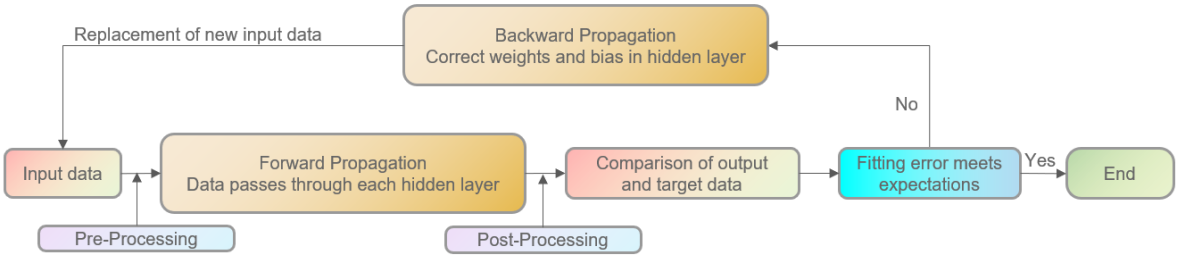

Figure 3.3 Flowchart of the neural network fitting process

Chapter 4

Fitting results and discussions

This chapter discusses fitting results based on the HCOG database. The twenty-one-output data were divided into three groups in Chapter 3, and typical fitting cases for each group will be selected for discussion in this chapter.

As mentioned before, a fitting error of ten percent or less is acceptable, but it is also demanding to require all data to be fitted within this range. In fact, the data distribution of the analog simulation is complex, and the source of the data is generally measured by using sensors in real experiments, which does not exclude the case of measurement error. In addition, since the data are collected in the whole reactor, naturally including the data in the reaction area and non-reaction area, it is obvious that the error on the reaction area is more sensitive, if the data with large fitting error are concentrated in the non-reaction area, while the fitting error of the reaction area turns out to be optimistic, this situation is not unacceptable either.

For orders of magnitude greater than five, neural network fitting is too difficult to obtain satisfactory results, and at present only more satisfactory results can be requested in sensitive areas, such as reaction areas. If we want to keep the fitting error of the whole database within ten percent, more work is needed in the future.

4.1 Analysis of results from Group A

In this section, we will pay attention to *T* and *ENTH* from Group A. They are representative of the output data in this group, and *T* exhibits stability over the entire value domain and has an order of magnitude difference of only two, so the fitting error is perfect and typical of the ideal neural network fitting samples. *ENTH*, although the value domain spans a larger area covering both positive tens of thirds and negative tens of sixth, the fitting results are also within three orders of magnitude difference of the absolute value of the data, so the results are very satisfactory. They are representative data of Group A. In fact, the neural network fitting error of Group A data can be controlled within ten percent in the whole value domain, and they are the ideal fitting samples that need normalization pre-processing only without any data post-processing.

4.1.1 Fitting result comparison of *T*

T is the temperature, which is an important physical quantity in simulation, reflecting the process of the chemical reaction at that place, and is the most intuitive quantity. Temperature can clearly indicate reactive and non-reactive regions, so it makes sense to accurately fit a neural network to the temperature data.

The reactor is composed of four nozzles for oxygen supply and one inlet for HCOG fed, and the four nozzles are mounted in a way to generate a swirling flow field so as to mix with the HCOG jet well. The geometry of the reformer which is also the computational domain in this work is exhibited in Fig. 4.1, and a total of more than 17.5 million unstructured cells have been used.

Figure 4.2 reflects the temperature distribution in the reactor, the red area is where the temperature is higher and we can see the outline of the reaction area, which can be roughly taken as the dividing line between the reactive and non-reactive areas. In general, the reaction area and its surrounding area are sensitive areas, and any data belonging to this area needs to minimize the fitting error. It is factually correct that the high temperature area is near the nozzle and roughly matches the reaction zone profile.

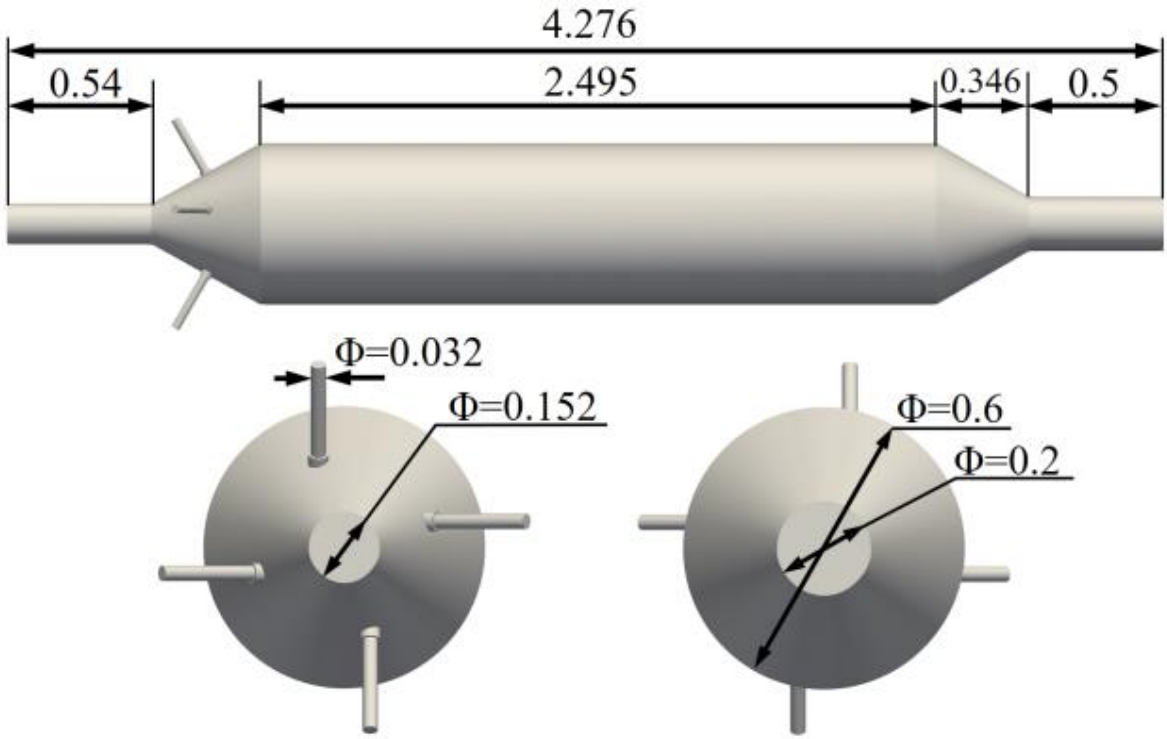

Figure 4.1 Configuration of the computational domain (unit: m)

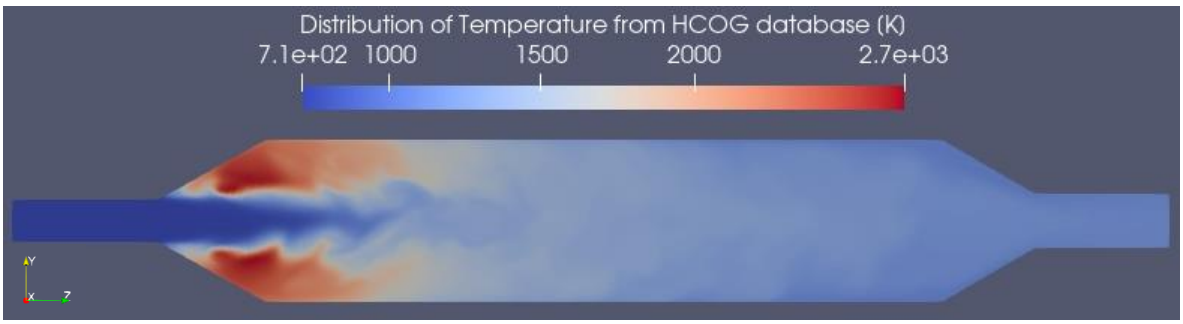

Figure 4.2 Snapshots of temperature distribution in the reactor from HCOG database

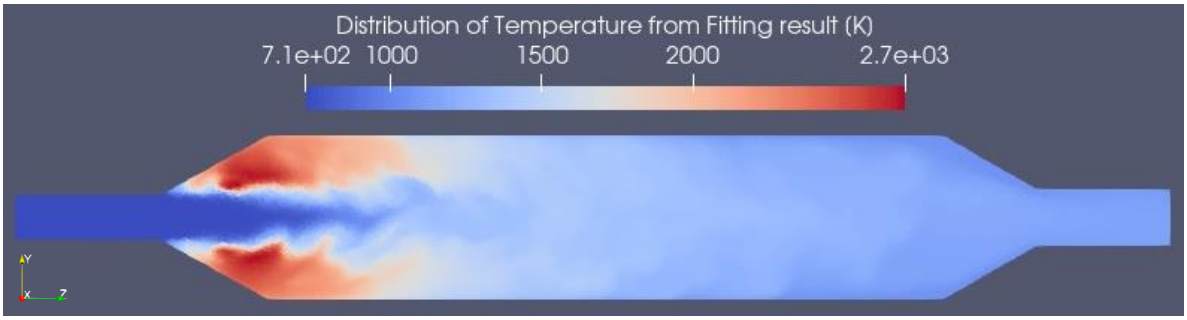

Figure 4.3 Snapshots of temperature distribution in the reactor from Fitting result

When we compare Fig. 4.2 and Fig. 4.3, we can see that both the trend of the temperature distribution and the value domain of the temperature, the fitting results are consistent with the database data, which indicates that the fitting accuracy is very good.

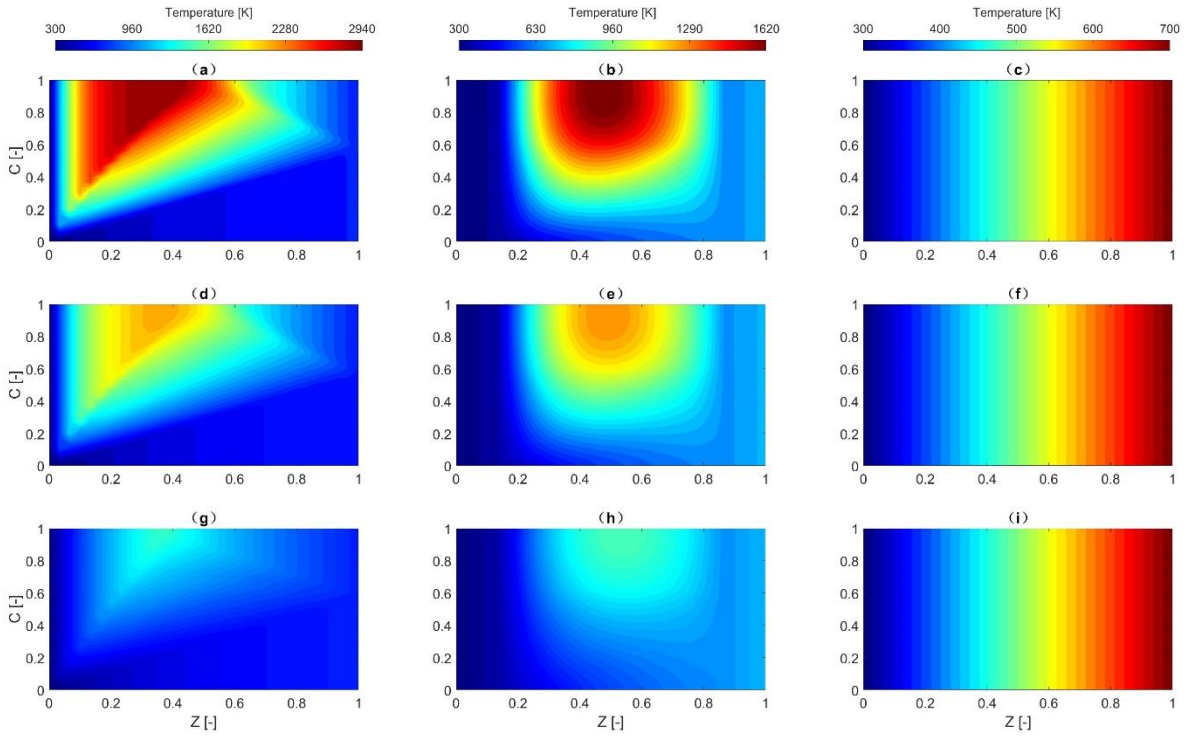

Figure 4.4 Temperature distributions in $Z \times C$ space from HCOG database

Figure 4.4 shows the temperature distributions from HCOG database in Z × C space, the set values of *ZV* and h^n from (a) to (i) are different. The set values for *ZV* and h^n for each case are:

(a) *ZV*=0 [-], h^n=0 [-].
(b) *ZV*=0.125 [-], h^n=0 [-].
(c) *ZV*=0.25 [-], h^n=0 [-].
(d) *ZV*=0 [-], h^n=0.5 [-].
(e) *ZV*=0.125 [-], h^n=0.5 [-].
(f) *ZV*=0.25 [-], h^n=0.5 [-].
(g) *ZV*=0 [-], h^n=1 [-].
(h) *ZV*=0.125 [-], h^n=1 [-].
(i) *ZV*=0.25 [-], h^n=1 [-].

Note that the highest temperature region occurs with *ZV*=0, h^n=0, when *Z* is in the range of 0.3 to 0.4 (near stoichiometric layer) and *C* is in the range of 0.8 to 1.0. This is because *Z* (Mixture fraction) describes the ratio of the mass fractions of the fuel and the oxidizer, when *Z* is 0, it means oxidizer side and when *Z* is 1, it means fuel side, so the reaction is only sufficient in the region (*Z* from 0.3 to 0.4) where the oxidizer and the fuel are sufficiently in contact and mixed, and therefore the region has the highest temperature. *C* (Process variable) describes how the chemical reaction proceeds, a *C* of 0 means that process of the chemical reaction chain has not started at all, an increase in *C* reflects the advancement of process of chemical reaction, so when *C* is near the maximum value, it means that the reaction has proceeded sufficiently and therefore reaches the maximum temperature. *ZV* (Fluctuation of mixture fraction) describes the inhomogeneity of the mixture; the smaller the *ZV*, the more homogeneous the mixing of the fuel and oxidizer. In this case, the reaction zone is usually more stable. When *ZV* reaches its maximum value (0.25), from the third column of Fig. 4.4, it could be seen that the temperature remains essentially unchanged. This is due to the fact that the components of the mixture fluctuate so much that the reaction basically does not start to take place. h^n (Normalized total enthalpy defect) describes the heat loss between the reactor and the outside; when h^n is zero, the heat loss between the reactor and the outside is minimized, and therefore the reaction temperature is higher. And as h^n increases, the heat loss between the reactor and the outside also increases, and the temperature at the corresponding place decreases under the same *Z*, *ZV* and *C*. Therefore, the highest temperature will come from the vicinity of the stoichiometric layer (*Z* from 0.3 to 0.4), where the stable combustibles composition (*ZV*=0) as well as the chemical reactions are fully carried out (*C*=1).

Fig 4.5 shows the temperature distributions from the fitting result in $Z \times C$ space, for each case from (a) to (i), ZV and h^n are set as in Fig 4.3. By comparing with Fig. 4.3, it can be seen that the temperature distribution is similar, which indicates that the error of the fitting results is very satisfactory.

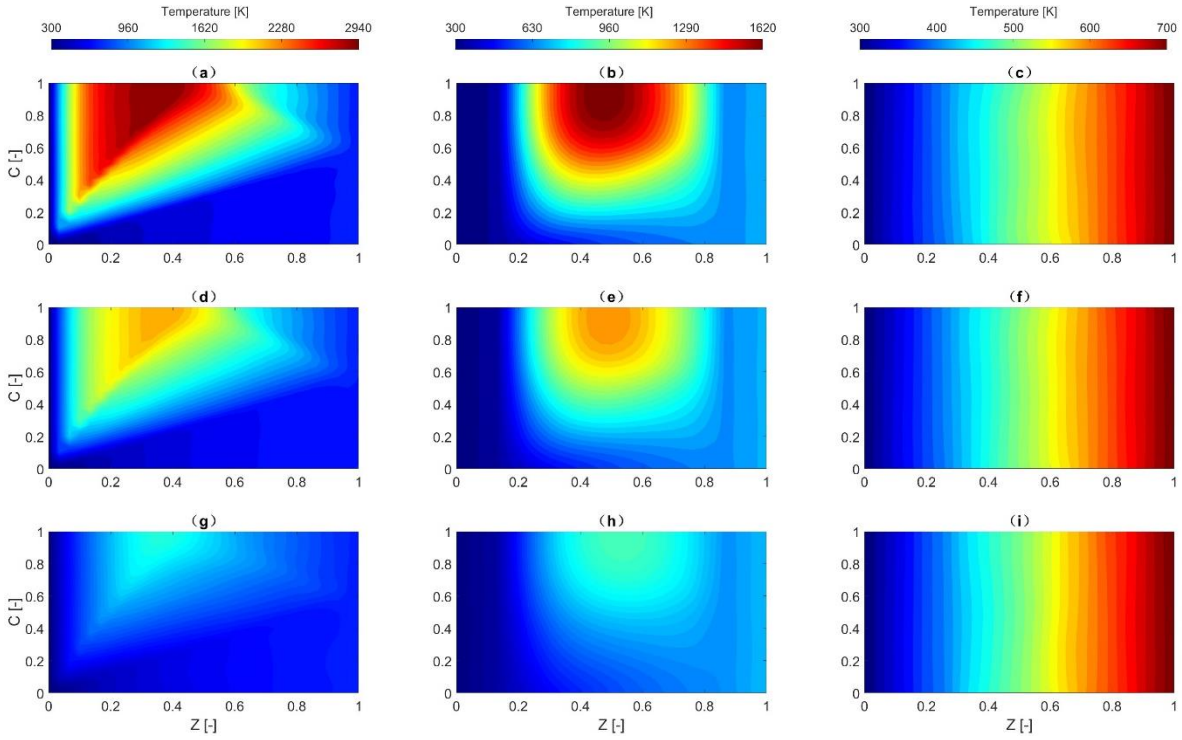

Figure 4.5 Temperature distributions in $Z \times C$ space from Fitting result

Fig 4.6 shows the distribution of the relative error of temperature between the neural network fitting data and the HCOG database data calculated according to Eq. (4.1), over the entire domain.

$$Relative\ error = \frac{fitting\ result - HCOG\ database}{HCOG\ database} \times 100\% \qquad (4.1)$$

As can be seen in Fig 4.6, the maximum value of the fitting error is 2.4%, which occurs near the inlet, and in the whole domain, the fitting error is much less than 10%, which shows that the neural network fitting error for temperature is very successful, both in terms of the distribution of temperatures and the specific values, which are perfectly in line with the expected values.

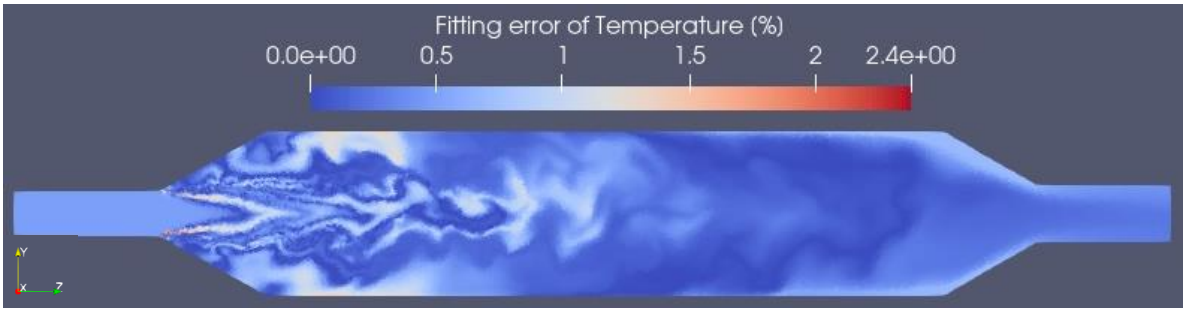

Figure 4.6 Snapshots for the distribution of temperature fitting errors in the reactor

4.1.2 Fitting result comparison of *ENTH*

ENTH is the specific enthalpy [J/(kg·K)], it represents the ability of a substance to absorb and release heat. In general, when *ENTH* is negative, the smaller the value, the more exothermic the substance is. Thus, from Fig 4.9, it can be seen that the fuel side ($Z = 1$) of ENTH is minimized.

At first glance, the fitting results for *ENTH* are very difficult, as can be seen in Fig 4.7, which shows that the domain of values for ENTH is very wide and involves both positive and negative numbers. However, in reality, the order of magnitude difference in the absolute value of *ENTH* is within 3. The fitting of *ENTH* is not very difficult and the fitting results are very satisfactory.

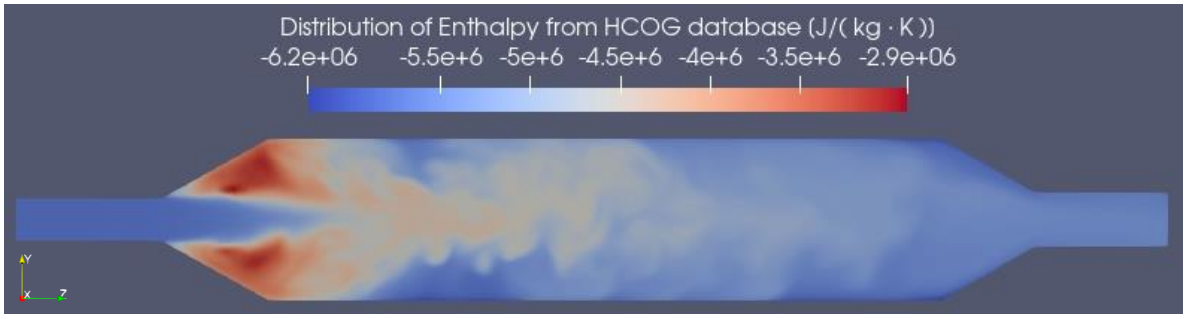

Figure 4.7 Snapshots of *ENTH* distribution in the reactor from HCOG database

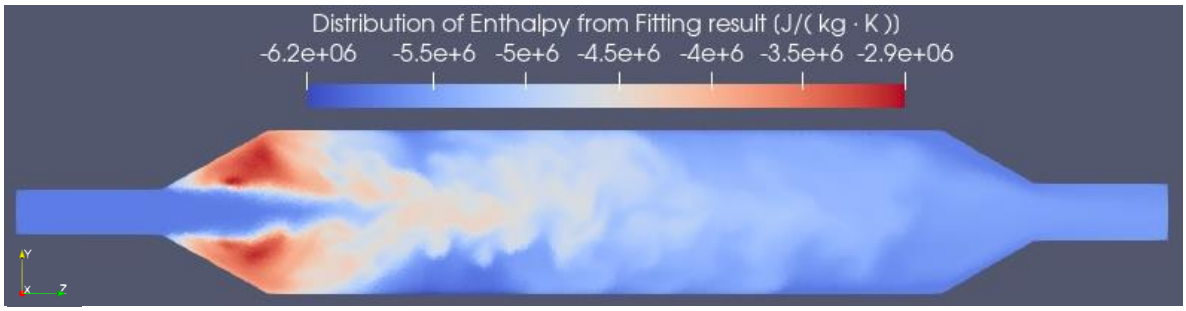

Figure 4.8 Snapshots of *ENTH* distribution in the reactor from Fitting result

As can be seen earlier, *COG* gas is supplied from the left inlet, and the reforming gas containing H_2 and *CO* is obtained at the right outlet, so in the center of the left side of Fig. 4.7, and on the right side near the outlet are the fuels, which have smaller *ENTH* (negative value), which is consistent with the previous projections.

Comparing Fig 4.7 and Fig 4.8, it can be seen that the fitting results, both in terms of the *ENTH* value domain and the distribution in the reactor, are in line with the data in the HCOG database, and the fitting results are very satisfactory. This also shows that the positive and negative sign of the target database and the wide range of values do not increase the fitting difficulty and increase the fitting error, as long as the order of magnitude difference of the absolute values is within a manageable range, then the fitting accuracy will be very good.

At the same time it gives us an inspiration. For neural network fitting, the fitting error for very small numbers with values distributed around 0 is very large. This is because the resolution of the neural network is certain, and for the neural network itself, there is no difference between fitting the 0 of the target database to the $10^{-6}$and -10^{-6}, because the numbers themselves are simply numbers to the neural network, and the neural network doesn't understand the physical meaning of the numbers behind them. If the mass fraction of the substance is being analyzed, a negative fit result is absolutely unacceptable.

When fitting data distributed on both sides of 0 or so, it is perfectly acceptable to fit its absolute value and then manually add a plus or minus sign according to some flags, which will make fitting decimals near 0 less difficult.

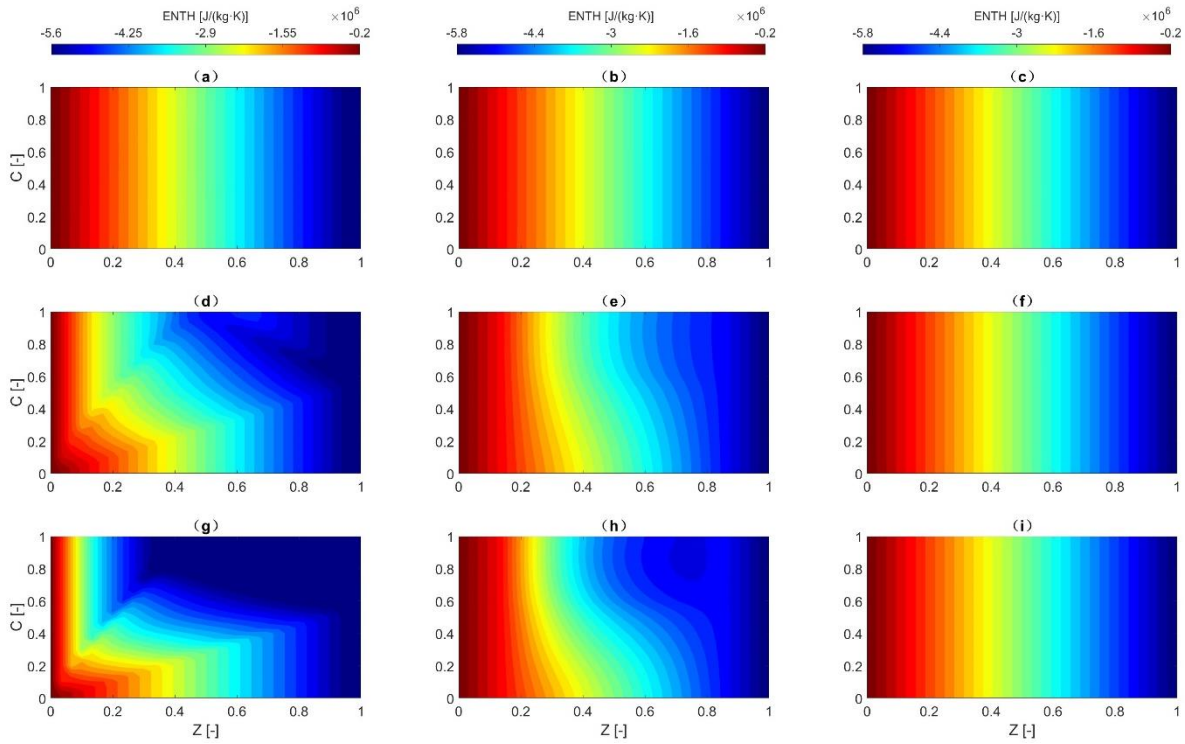

Figure 4.9 *ENTH* distributions in $Z \times C$ space from HCOG database

Figure 4.9 is the distribution of *ENTH* in $Z \times C$ space from HCOG database, the set values of *ZV* and h^n from (a) to (i) are different. The set values for *ZV* and h^n for each case are:

(a) *ZV*=0 [-], h^n=0 [-].
(b) *ZV*=0.125 [-], h^n=0 [-].
(c) *ZV*=0.25 [-], h^n=0 [-].
(d) *ZV*=0 [-], h^n=0.5 [-].
(e) *ZV*=0.125 [-], h^n=0.5 [-].
(f) *ZV*=0.25 [-], h^n=0.5 [-].
(g) *ZV*=0 [-], h^n=1 [-].
(h) *ZV*=0.125 [-], h^n=1 [-].
(i) *ZV*=0.25 [-], h^n=1 [-].

ENTH describes the exothermic state of the system, so *ENTH* is affected by h^n, Comparing each column of Fig 4.9, the area of the blue region increases significantly as h^n increases when *ZV* is certain. Moreover, as *Z* increases, the previously more neatly stepped distribution of *ENTH* ($h^n = 0$, (a) to (c)) is also broken, and the blue region (the region with smaller *ENTH*) extends rapidly in the direction of the smaller *Z*, which indicates an increase

in the heat loss of the system, and a severe loss of heat, which corresponds to (g) of Fig. 4.4, with a significant decrease in temperature. The third column of Figure 4.9 (ZV = 0.25) does not seem to have a significant change in the distribution of *ENTH* with increasing h^n. This is because when ZV = 0.25, the homogeneity of the mixture is so poor that there is no chemical reaction going on at all, and naturally there is no heat loss to speak of, and thus the distribution of *ENTH* shows a consistent.

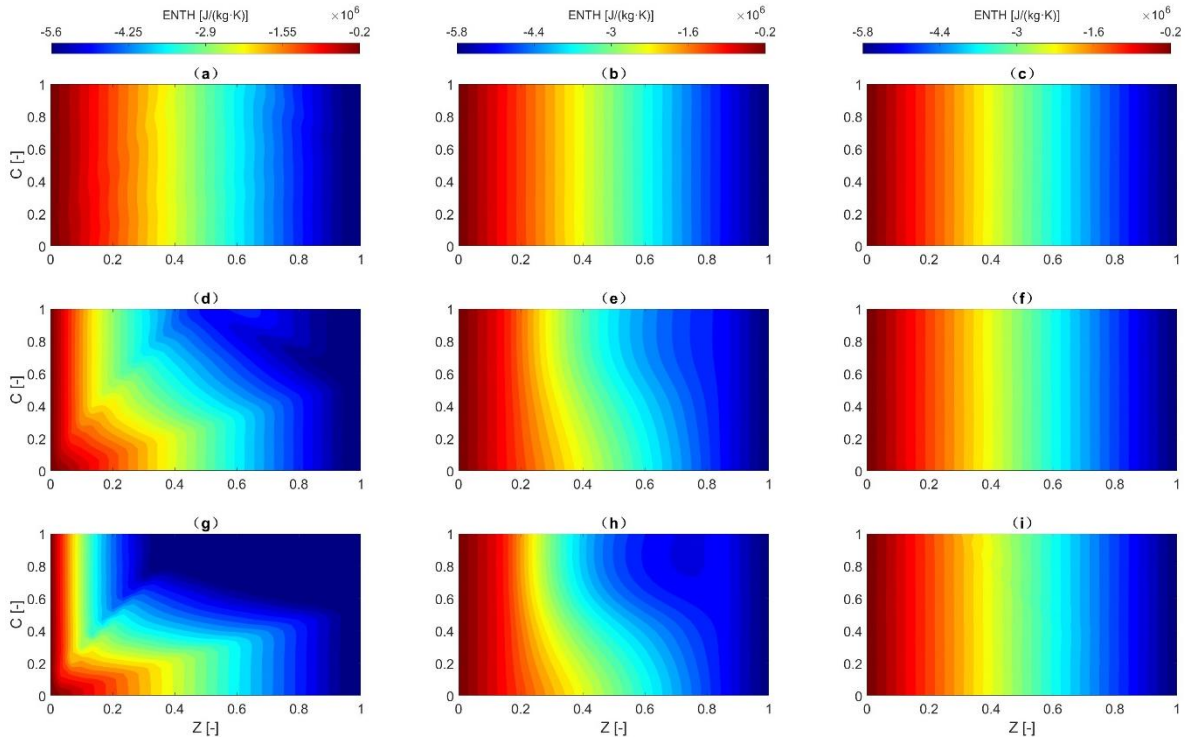

Figure 4.10 *ENTH* distributions in $Z \times C$ space from Fitting result

Comparing Fig 4.9 and Fig. 4.10, it can be seen that the fitting results of *ENTH* are ideal, and both the range of the value domain and the distribution of *ENTH* in $Z \times C$ space are in high agreement with the distribution of *ENTH* from HCOG database. If we looks closely, we can see that the edges of the color columns for different values of *ENTH* in the first row (h^n = 0.25) of Figure 4.10 are not smooth. Not being smooth means that the value of *ENTH* at the boundary of every color column is not uniform and fluctuates. This is because the data from HCOG database at the corresponding location all have the same value, so the critical line of the *ENTH* distribution is straight, while the data from fitting result fluctuates around the critical value of that *ENTH*, but the average value is the same, thus resulting in a distribution with the same color bar but not smooth edges.

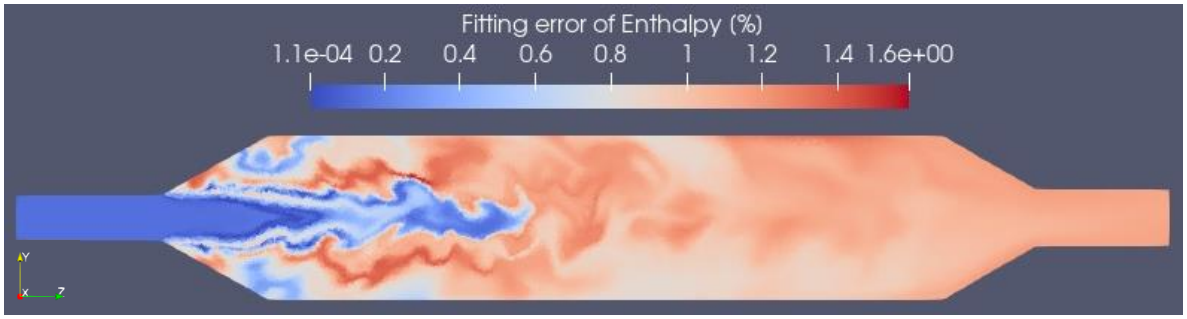

Figure 4.11 Snapshots for the distribution of *ENTH* fitting errors in the reactor

Observation of the *ENTH* fitting error distribution in Figure 4.11 shows that the neural network fitting for *ENTH* is perfect, within 1.6% error, and within 1% error for the fitting result near the reaction zone. Such a fitting can be used directly without any post-processing, the fitting results for both *T* and *ENTH* are perfect, and in fact the errors of the fitting results for the entire Group A are at similar levels, and they are both optimal for the target database.

4.2 Analysis of results from Group B

Fitting mass fractions to chemical components is a very challenging task. The value domain of the mass fraction is between 0 and 1, and it is very easy to get tiny values close to 0, for example, at the level of 10^{-6}, which creates an order-of-magnitude difference of about 6, which leads to very poor fitting results. At the same time, for the minimum value 0 for the mass fraction, the neural network fitting is also highly likely to give very small numbers that are negative but close to 0. This is contrary to the chemical meaning of the mass fraction and requires post-processing of the data. Fortunately, the order-of-magnitude difference in the mass fraction of group B is within 5, and the fitting result is also very favorable.

The purpose of the reorganization of the COG is to obtain H_2 and *CO*, so Group B will be represented by *Y_H2* and *Y_CO*.

4.2.1 Fitting result comparison of *Y_H2*

Y_H2 is the mass fraction of hydrogen, there is H_2 in the initial COG, H_2 is also present in the partially oxidized recombinant gas, so H_2 will be present near the inlet and outlet of the reactor. As shown in Fig 4.12, it can be seen that the maximum value of *Y_H2* occurs at the edge of the reaction zone, which suggests that a large amount of H_2 is generated there, which is in line with our prediction.

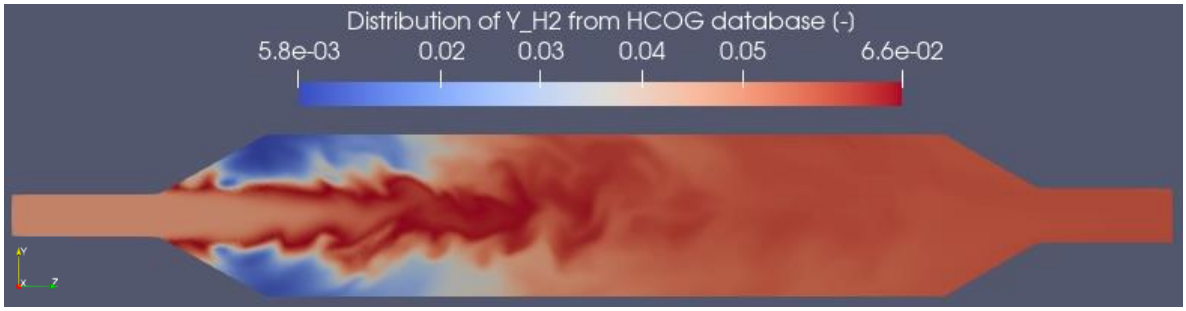

Figure 4.12 Snapshots of *Y_H2* distribution in the reactor from HCOG database

Fig 4.13 describes *Y_H2* distribution from the fitting results. First the distributional trends are quite consistent, but there is a slight difference in the minimum value of the domain, this has a negligible effect on the fitting results. As can be seen from the value domain, the order of magnitude difference in *Y_H2* is not very large.

The minimum value of *Y_H2* is not 0. This is because Fig 4.12 and Fig 4.13 are only snapshots of the center interface of the reactor, which does not cover the full domain of values of the mass fraction throughout the reactor. The reason for choosing a central slice is that it provides a more visual representation of the distribution of physical quantities in the reaction zone; when half a reactor is chosen, the superimposed distribution map will affect the visualization.

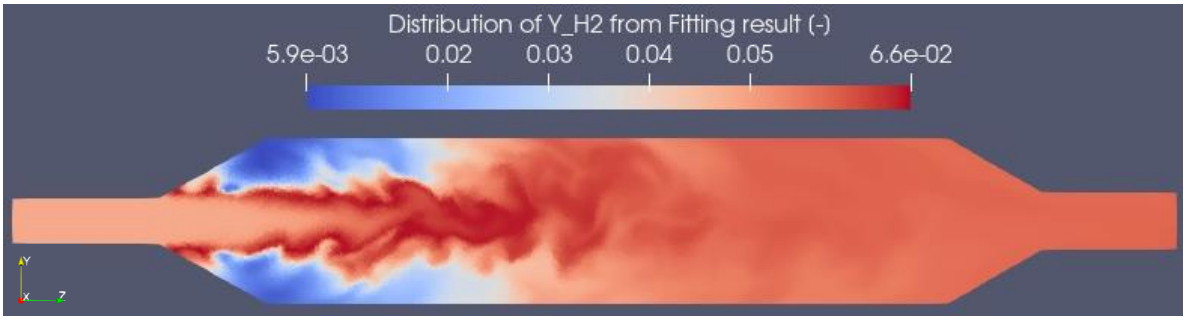

Figure 4.13 Snapshots of *Y_H2* distribution in the reactor from Fitting result

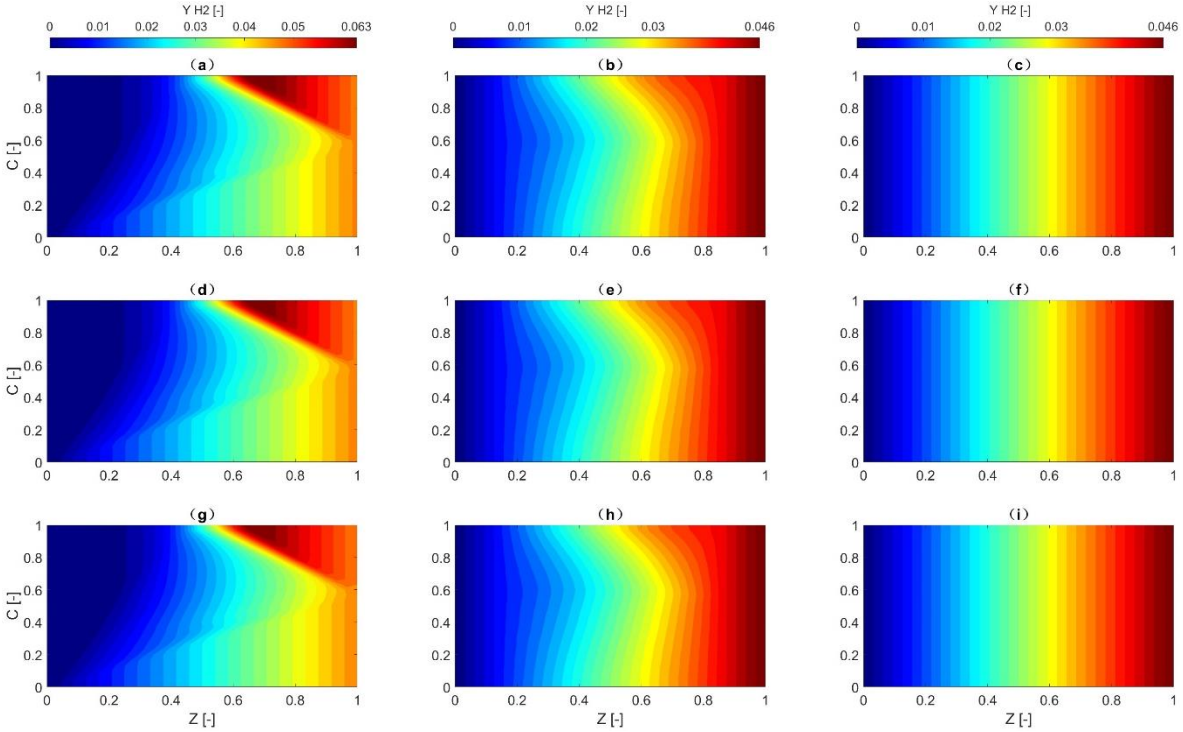

Figure 4.14 *Y_H2* distributions in $Z \times C$ space from HCOG database

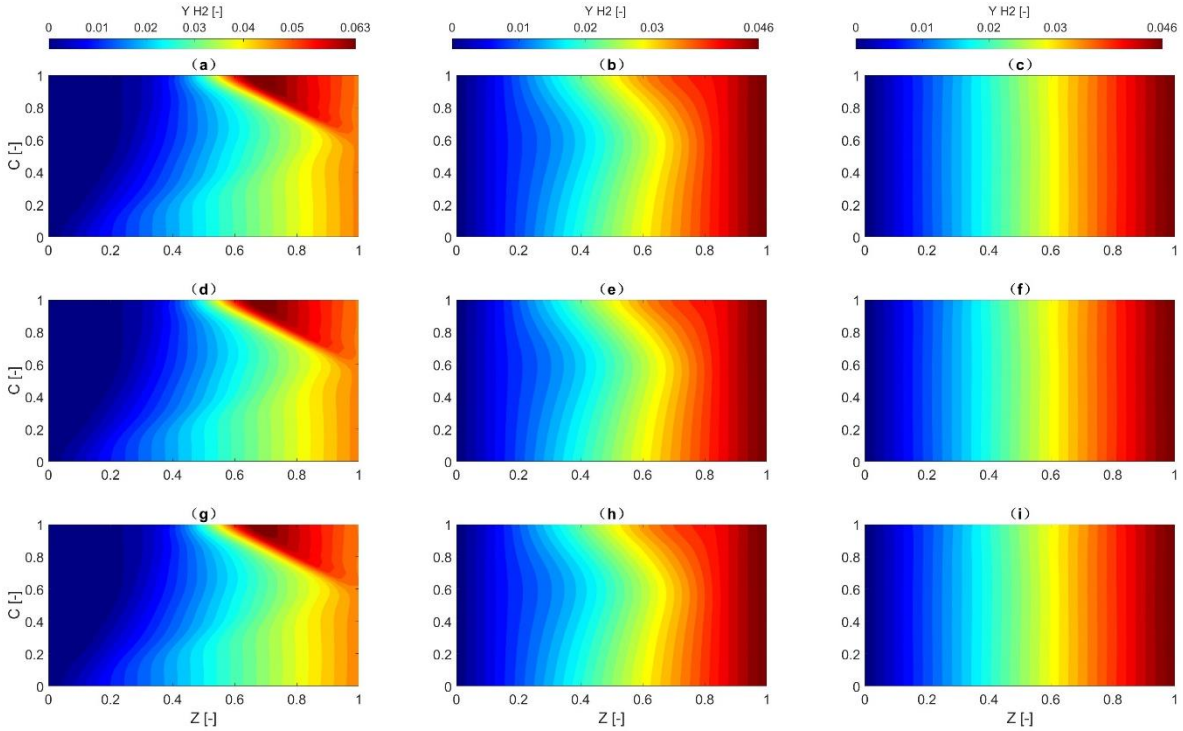

Figure 4.15 *Y_H2* distributions in $Z \times C$ space from Fitting result

Fig 4.14 and Fig 4.15 show the distribution of *Y_H2* in $Z \times C$ space from the HCOG database and the fitting results, respectively. The set values of *ZV* and h^n from (a) to (i) are different. The set values for *ZV* and h^n for each case are:

(a) *ZV*=0 [-], h^n=0 [-].
(b) *ZV*=0.125 [-], h^n=0 [-].
(c) *ZV*=0.25 [-], h^n=0 [-].
(d) *ZV*=0 [-], h^n=0.5 [-].
(e) *ZV*=0.125 [-], h^n=0.5 [-].
(f) *ZV*=0.25 [-], h^n=0.5 [-].
(g) *ZV*=0 [-], h^n=1 [-].
(h) *ZV*=0.125 [-], h^n=1 [-].
(i) *ZV*=0.25 [-], h^n=1 [-].

The increase in H^n has little effect on *Y_H2* produced, this is because the increase in heat dissipation affects the production of the whole partial oxidation, the mass fraction is essentially a percentage and H^n does not affect the compositional proportions of the products. However, the effect of *ZV* is too large for the distribution of *Y_H2*. As *ZV* increases, the distribution of the mixture becomes inhomogeneous, and the chemical reaction becomes more and more difficult to carry out. So that when $ZV = 0.25$, the maximum value of *Y_H2* decreases and the distribution becomes completely gradient, which means that no reaction zone is generated. When the system heat loss is minimized and the mixture is homogeneous and stable ($ZV = 0, H^n = 0$), at this point the temperature reaches its maximum value, *Y_H2* reaches its maximum downstream of the reactor ($Z = 0.7, C = 1$).

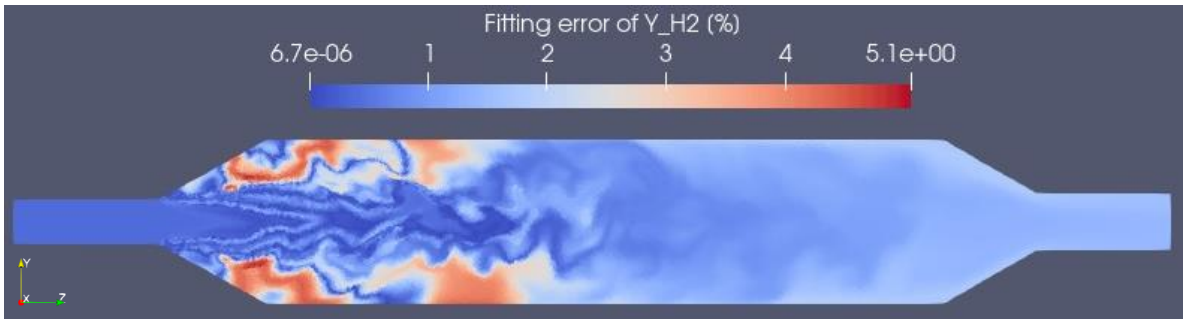

Figure 4.16 Snapshots for the distribution of *Y_H2* fitting errors in the reactor

Fig 4.16 shows the distribution of the *Y_H2* neural network fitting results, from which it can be seen that the maximum error is 5.1%, which is within acceptable limits. However, comparing the fitting results of *T* and *ENTH*, it is frustrating to see that the fitting error of *Y_H2* is still about three times as large as theirs. The maximum fitting error occurs in the outer part of the reaction zone, and the fitting error measured within the reaction zone is within 1%, which is a satisfactory fitting result. It can be said that overall the fitting results of *Y_H2*, although acceptable, have deteriorated compared to the fitting results of the output data in group A. Looking closely at the fitting errors in sensitive areas such as the reaction area, the fitting results are still satisfactory.

4.2.2 Fitting result comparison of *Y_CO*

Y_CO is the mass fraction of carbon monoxide, *CO* is also a major component of recombinant gases and is the main product of the partial oxidation process. Fig 4.17 depicts the *Y_CO* distribution from the HCOG database. Unlike H_2, the crude COG does not contain *CO*, so we can see that *Y_CO* is 0 at the reactor inlet. At the edge of the reaction zone and downstream of the reactor, *Y_CO* reaches a maximum value, implying that there is a large amount of *CO* production. Compared to *Y_H2*, the mass fraction of *CO* downstream of the reactor is much larger than the mass fraction of *H2* because the relative molecular mass of *H2* is too small.

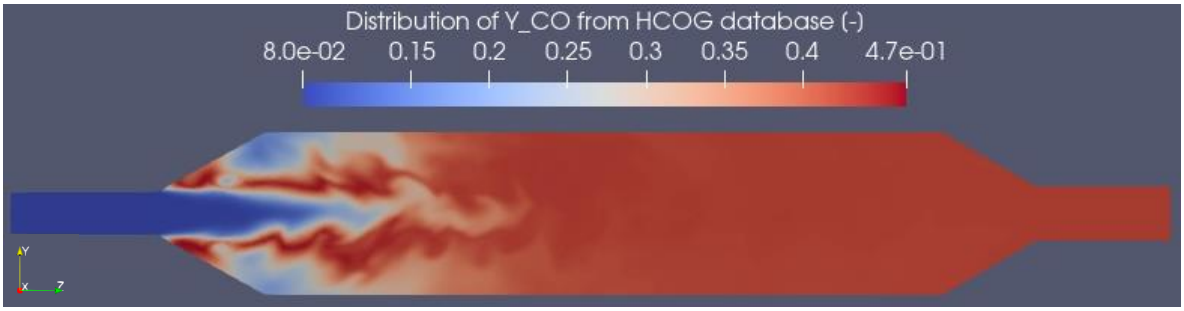

Figure 4.17 Snapshots of *Y_CO* distribution in the reactor from HCOG database

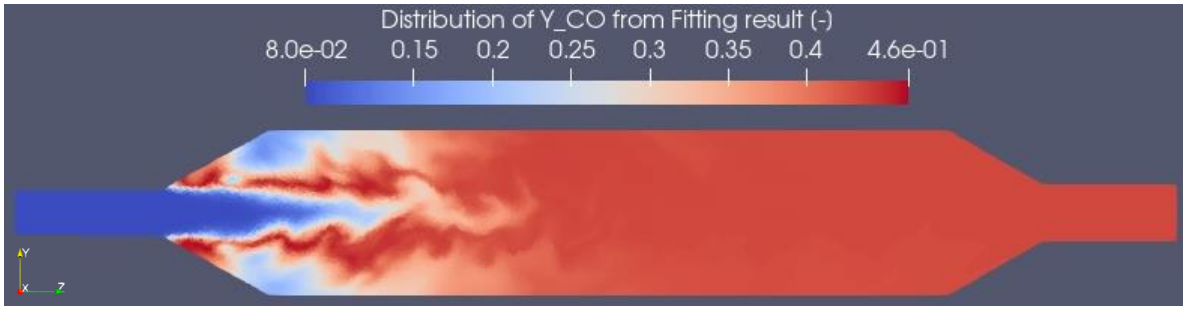

Figure 4.18 Snapshots of *Y_CO* distribution in the reactor from Fitting result

Comparing the *Y_CO* distribution plots in Fig 4.17 and Fig 4.18, it can be seen that *CO* fitting results are still very satisfactory and the *Y_CO* distribution trends are consistent. There is only a slight difference in the maximum values, but this is acceptable. Although the data structure of output data from Group B is too complex and a pessimistic view of their neural network fitting has been reported. However, since the order of magnitude difference of the data in Group B is not very large, the fitting solution is still largely acceptable.

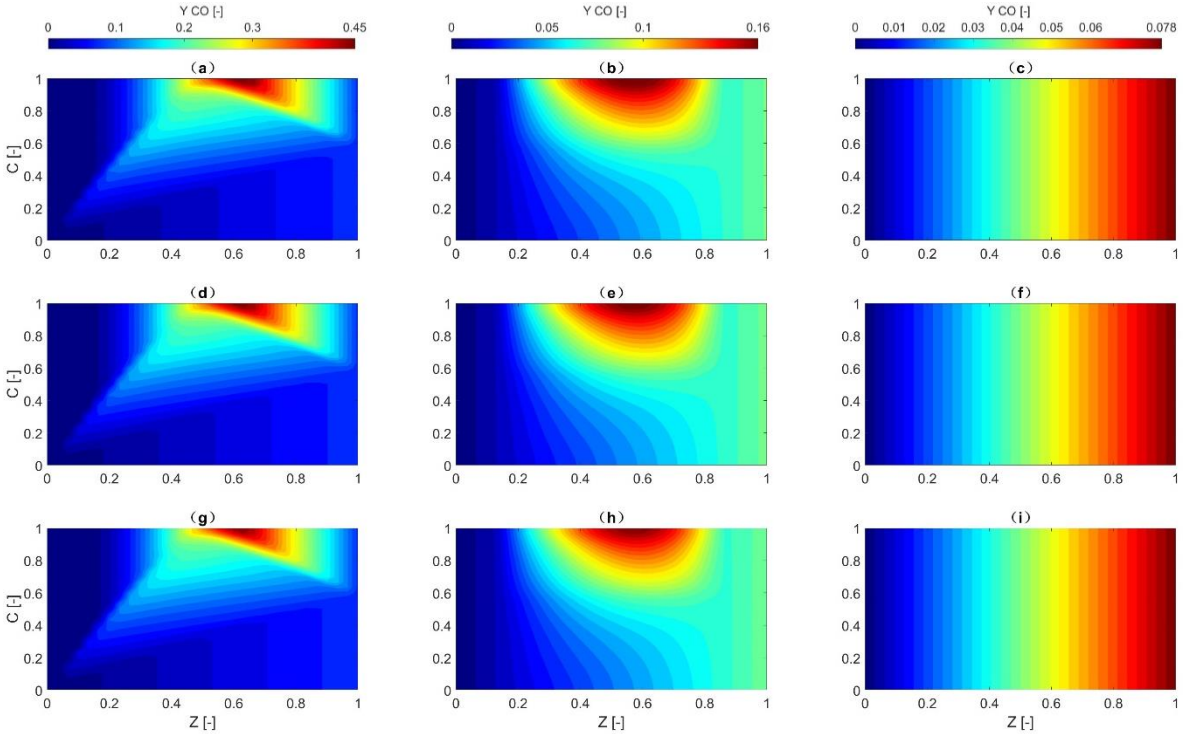

Figure 4.19 *Y_CO* distributions in $Z \times C$ space from HCOG database

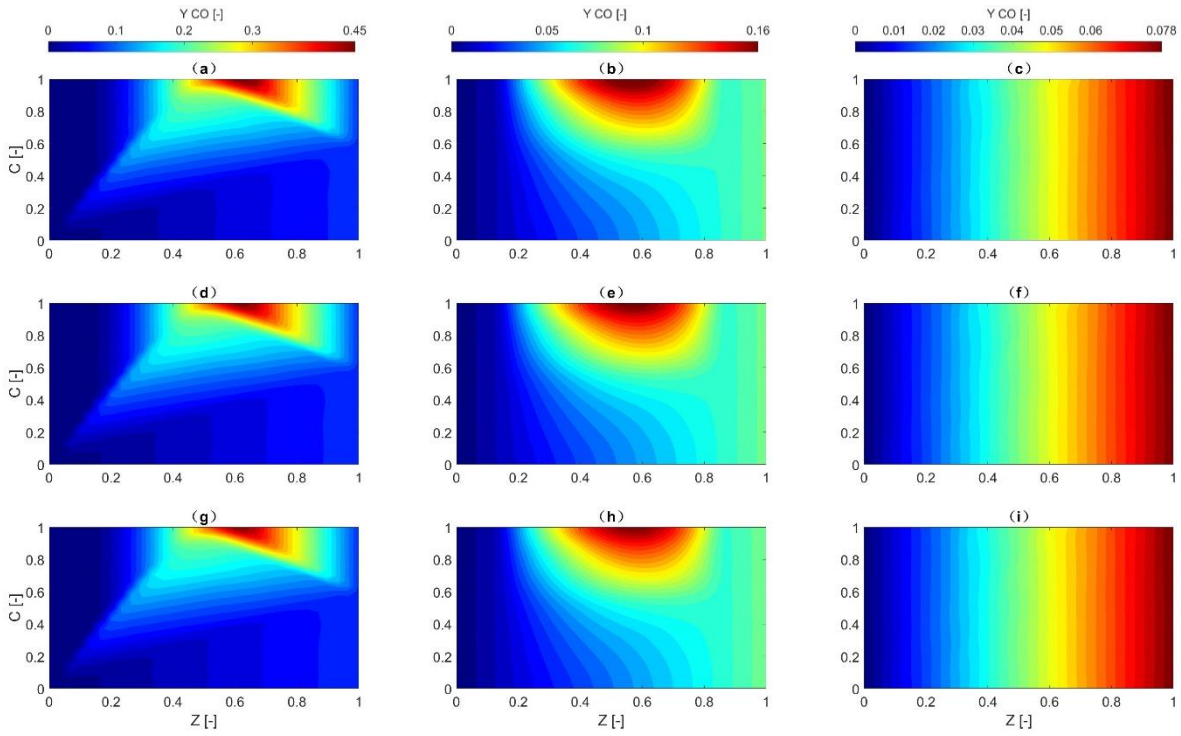

Figure 4.20 *Y_CO* distributions in $Z \times C$ space from Fitting result

Fig 4.19 and Fig 4.20 are the distribution of *Y_CO* in $Z \times C$ space from the HCOG database and the fitting result. The set values of *ZV* and h^n from (a) to (i) are different. The set values for *ZV* and h^n for each case are:

(a) *ZV*=0 [-], h^n=0 [-].
(b) *ZV*=0.125 [-], h^n=0 [-].
(c) *ZV*=0.25 [-], h^n=0 [-].
(d) *ZV*=0 [-], h^n=0.5 [-].
(e) *ZV*=0.125 [-], h^n=0.5 [-].
(f) *ZV*=0.25 [-], h^n=0.5 [-].
(g) *ZV*=0 [-], h^n=1 [-].
(h) *ZV*=0.125 [-], h^n=1 [-].
(i) *ZV*=0.25 [-], h^n=1 [-].

Y_CO is similar to *Y_H2* in that neither of them is much affected by h^n, but very much by *ZV*. As *ZV* increases, the inhomogeneity of the mixture increases, and partial oxidation process becomes difficult. the *CO* changes from being concentrated near the reaction zone at the beginning (first column of Fig. 4.19 and Fig. 4.20) to being distributed in a gradient (third column of Fig. 4.19 and Fig. 4.20), which implies that basically no reaction occurs here.

Comparing the trend of the distribution and the range of values of *Y_CO* in the two figures, the fitting results are acceptable.

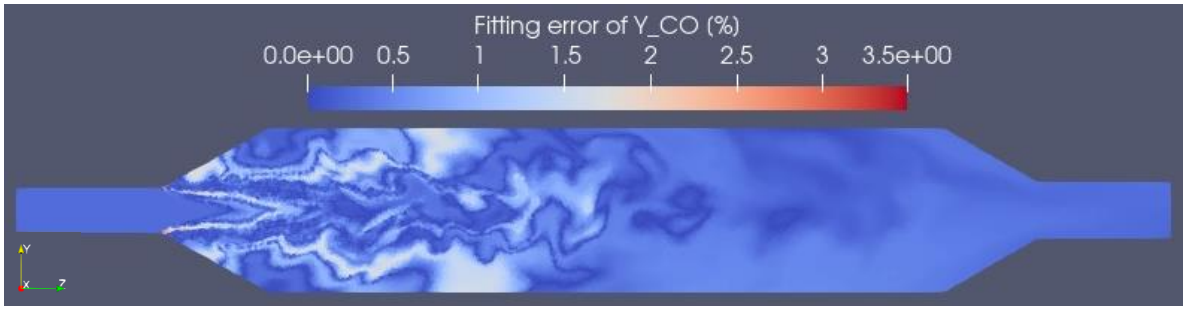

Figure 4.21 Snapshots for the distribution of *Y_CO* fitting errors in the reactor

Figure 4.21 shows the fitting error distribution of *Y_CO*, from the figure it can be seen that the fitting accuracy of *Y_CO* is better than that of *Y_H2*, the vast majority of the area of the reactor has an error of about 1%, which is already comparable to the fitting results of Group A. The very small portion of the error at the inlet of the reactor is 3.5%, and this is also because of the absence of *CO* here, and the *Y_CO* is 0. However, the neural network still gives values close to 0 but not zero.

4.3 Analysis of results from Group C

The neural network fitting of the mass fractions of the chemical components of group C was the most difficult, with order-of-magnitude differences of more than eight for these data, and the fitting accuracy was very bad.

4.3.1 Fitting result comparison of *Y_C2H2*

Y_C2H2 is the mass fraction of acetylene, which is an additional product throughout the partial oxidation process; it is not a constituent of COG, but only a by-product. From Fig 4.18, it can be seen that *Y_C2H2* at the reactor inlet is 0, while Y_C2H2 peaks at the end of the reaction zone and near the outlet, which indicates that there is C_2H_2 generation during the partial oxidation process.

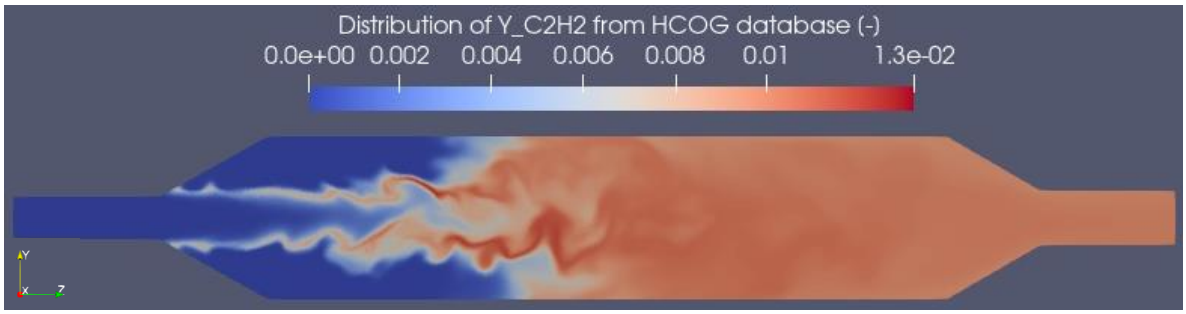

Figure 4.22 Snapshots of *Y_C2H2* distribution in the reactor from HCOG database

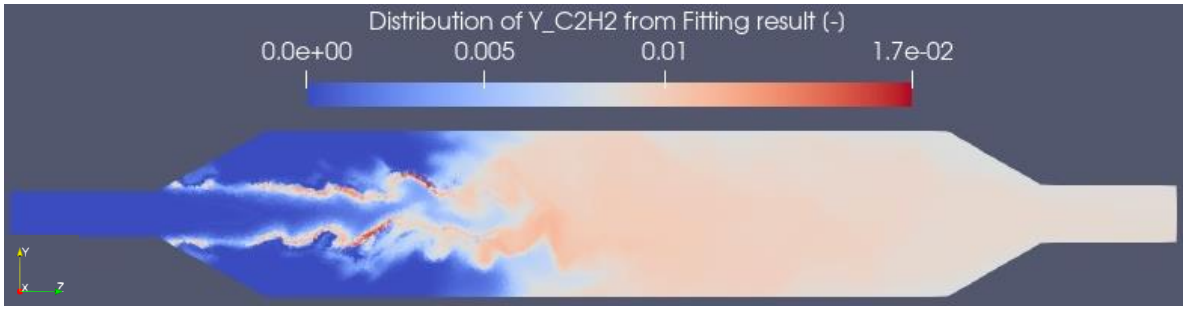

Figure 4.23 Snapshots of *Y_C2H2* distribution in the reactor from Fitting result

Comparison of Fig 4.22 and Fig 4.23 shows that downstream of the reactor (right half), the distribution of the fitting results follows the same trend. However, in the reaction zone close to the inlet of the reactor, *Y_C2H2* of the fitting results is significantly higher than the value of *Y_C2H2* of HCOG database, which leads to the maximum value of the color bar of the fitting results being higher than the HCOG database result (experimental values).

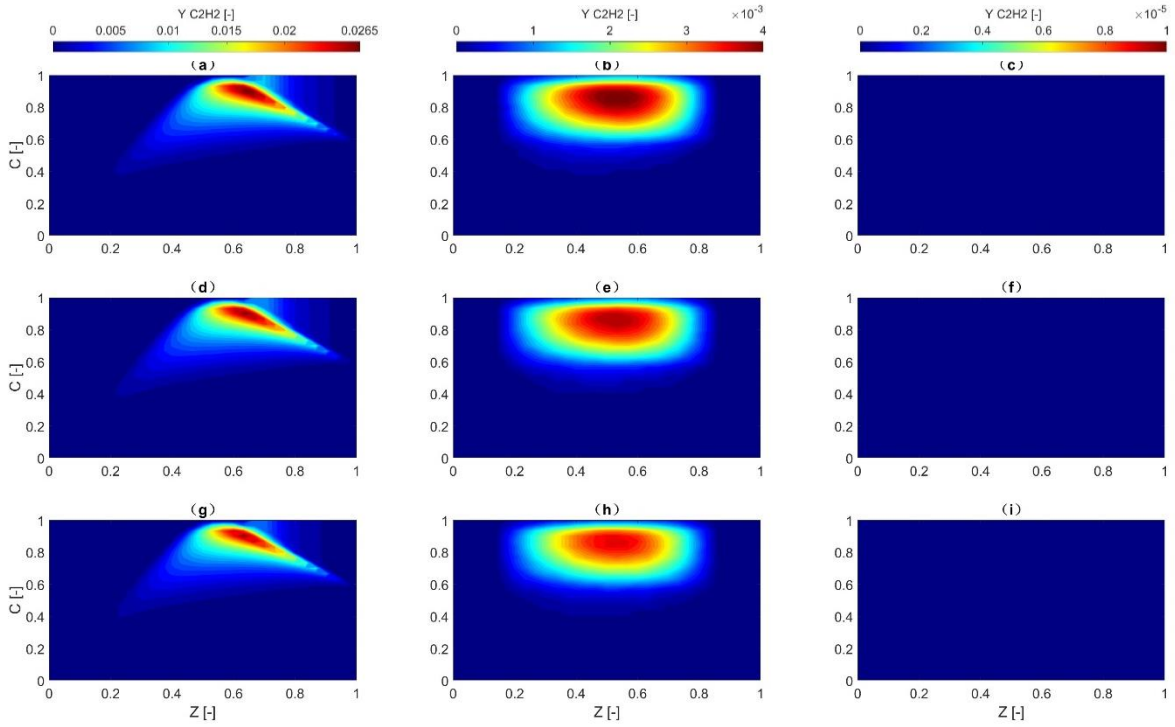

Figure 4.24 *Y_C2H2* distributions in $Z \times C$ space from HCOG database

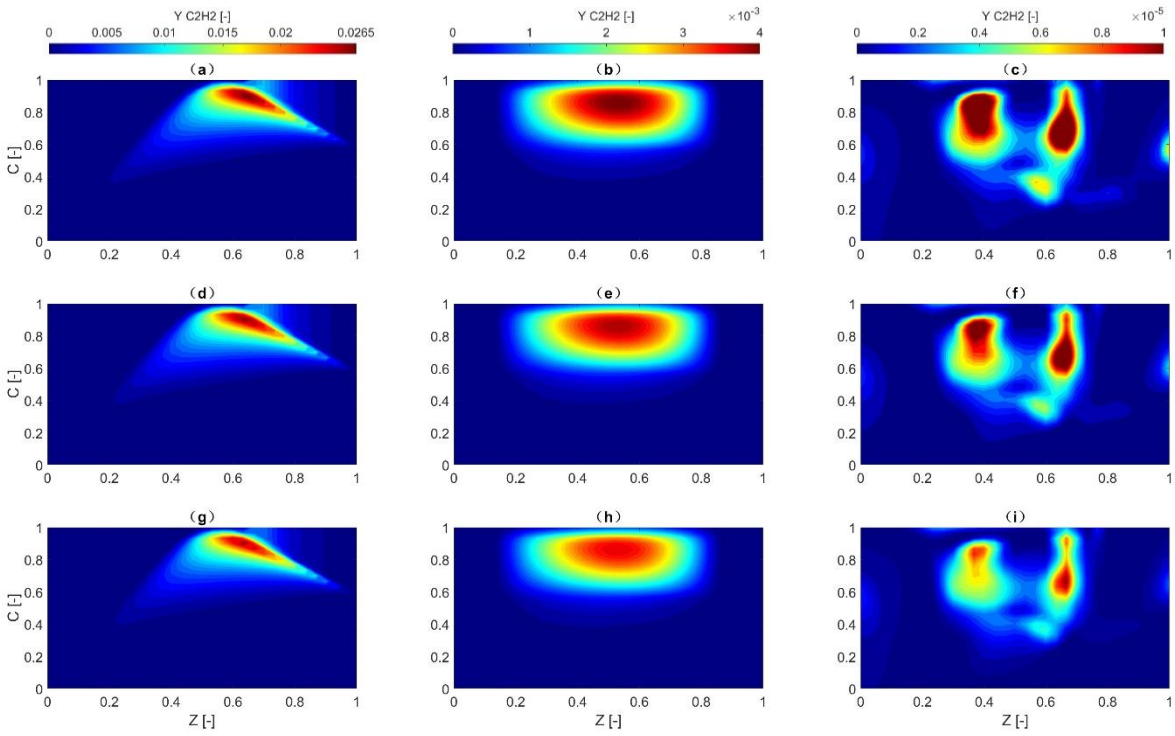

Figure 4.25 *Y_C2H2* distributions in $Z \times C$ space from Fitting result

Figures 4.24 and 4.25 show the distribution of *Y_C2H2* in $Z \times C$ space from the HCOG database and fitting results, respectively. The set values for *ZV* and h^n for each case are:

(a) *ZV*=0 [-], h^n=0 [-].
(b) *ZV*=0.125 [-], h^n=0 [-].
(c) *ZV*=0.25 [-], h^n=0 [-].
(d) *ZV*=0 [-], h^n=0.5 [-].
(e) *ZV*=0.125 [-], h^n=0.5 [-].
(f) *ZV*=0.25 [-], h^n=0.5 [-].
(g) *ZV*=0 [-], h^n=1 [-].
(h) *ZV*=0.125 [-], h^n=1 [-].
(i) *ZV*=0.25 [-], h^n=1 [-].

Comparing the two figures, it can be seen that at a *ZV* of 0 or 0.125, the neural network fitting accuracy is very good regardless of the value of h^n, which perfectly presents the distribution of *Y_C2H2*.

However, as *ZV* increases, the inhomogeneity of the mixture increases, and the reaction becomes unstable. When *ZV* is 0.25, *Y_C2H2* is 0 (third column of Fig 4.24) under such situation because the mixture is so inhomogeneous that not much chemical reaction is taking place at all. Returning to the data itself, and leaving aside the physical meaning behind the numbers, this means that when the target data is 0, the neural network does not give a fitting value of 0. Instead, it gives very small numbers close to zero on the order of 10^{-5}. This is the problem with the resolution of the neural network mentioned before, which is an iterative process that utilizes Forward Propagation and Back Propagation for the overall value domain to reduce the error. If the range of the value domain of the target data is too large (more than 6 orders of magnitude), in order to take into account the error of the large values, when the error of the large values is controlled within 1%, and the small values are processed with the same precision, then the error will not be 1%, but may be 10% or even 100%.

The current solution is either to group the value domains by order of magnitude and perform separate and distinct neural network simulations, or to post-process the data manually. However, these are not generalized solutions and further research is needed in the future.

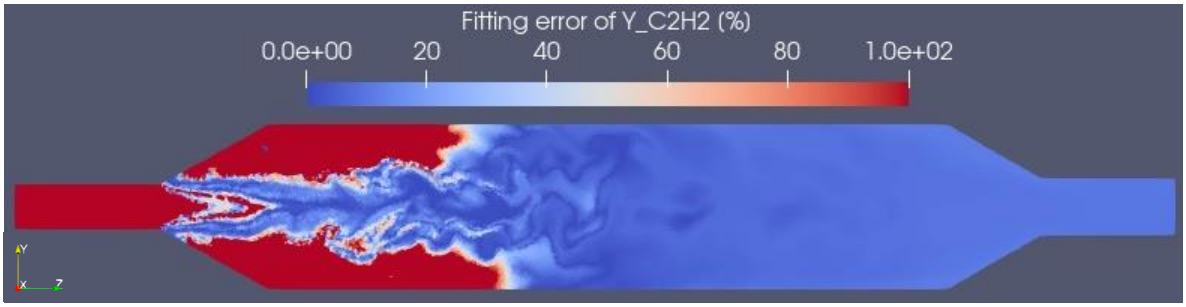

Figure 4.26 Snapshots for the distribution of *Y_C2H2* fitting errors in the reactor

Consistent with expectations, the fitting accuracy for Group C is very poor, and even the maximum value of 10% of acceptable error is only about half area of reactor in Fig 4.26. Comparing this to Fig 4.22, it can be seen that the fitting error increases significantly as *Y_C2H2* decreases. *Y_C2H2* is maximized over most of the region with errors within 10%. This indicates that the fitting accuracy decreases significantly as the order of magnitude difference of target data increases.

4.3.2 Fitting result comparison of *Y_C6H6*

Y_C6H6 is the mass fraction of benzene, benzene is a component of TAR and is contained in the crude COG. There is no C_6H_6 in the final recombinant gas, and we can verify this with the distribution figure below.

Figure 4.27 shows the distribution of *Y_C6H6* in the reactor from the HCOG database. As we expected, the distribution of C_6H_6 is only in the crude COG, which is consumed during partial oxidation, and *Y_C6H6* is 0 outside the reaction zone and downstream of the reactor.

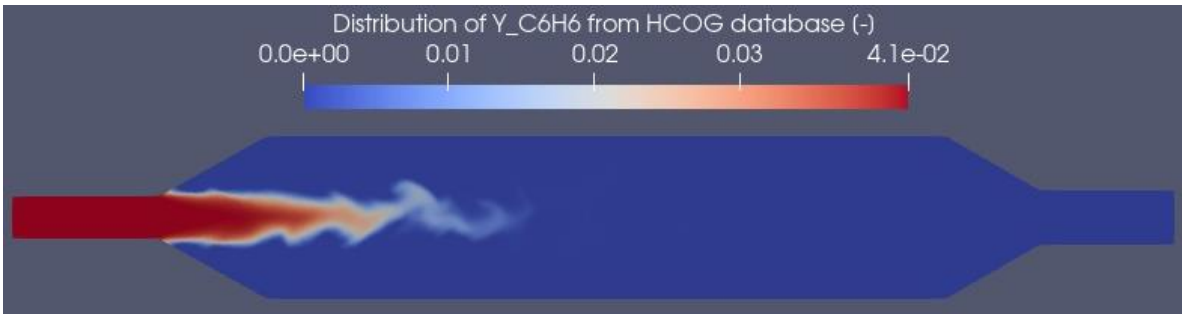

Figure 4.27 Snapshots of *Y_C6H6* distribution in the reactor from HCOG database

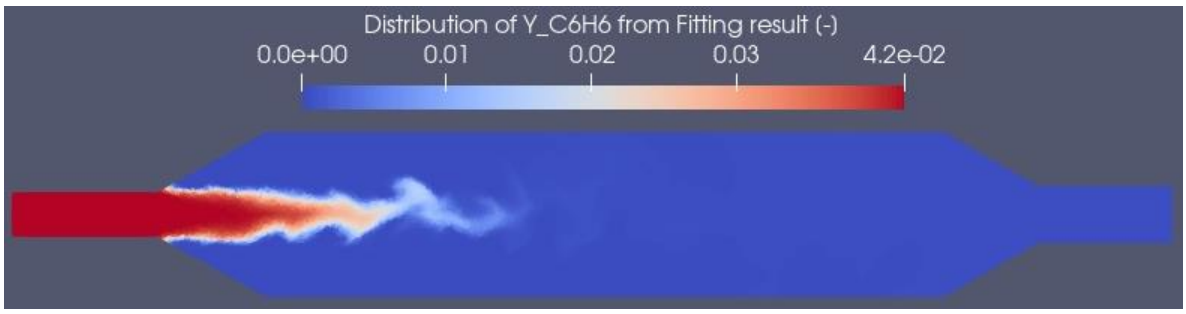

Figure 4.28 Snapshots of *Y_C6H6* distribution in the reactor from Fitting result

Compared to the distribution of the fitting results for *Y_C2H2*, the fitting accuracy of *Y_C6H6* is higher. Both the range of the color bar and the distribution of *Y_C6H6* are the same as that of *Y_C6H6* from the HCOG database.

The dismal situation is that we again have to encounter fitting for 0. As can be seen in Fig 4.27, the region where *Y_C6H6* is 0 is much larger compared to the distribution of *Y_C2H2*, accounting for more than 80% of the entire reactor area, which is catastrophic for neural network fitting. But optimistically, since *Y_C6H6* only has values in a small region, it is very convenient to manually tweak the fitting results.

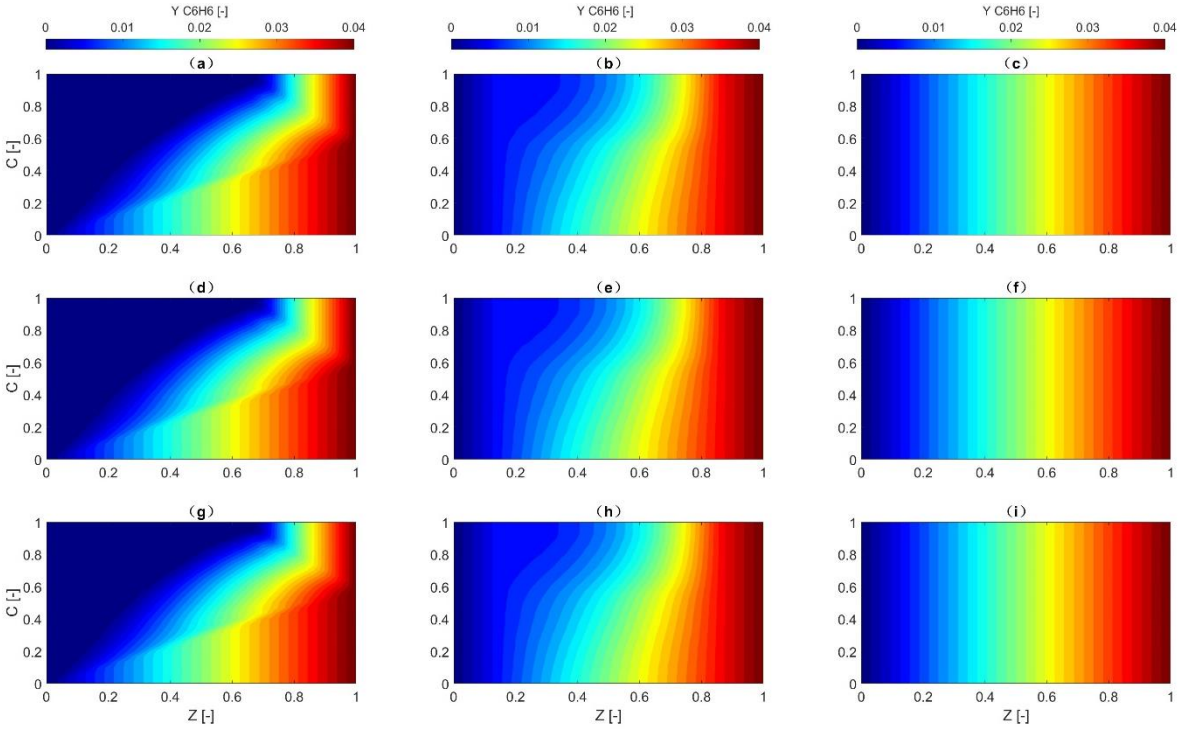

Figure 4.29 *Y_C6H6* distributions in $Z \times C$ space from HCOG database

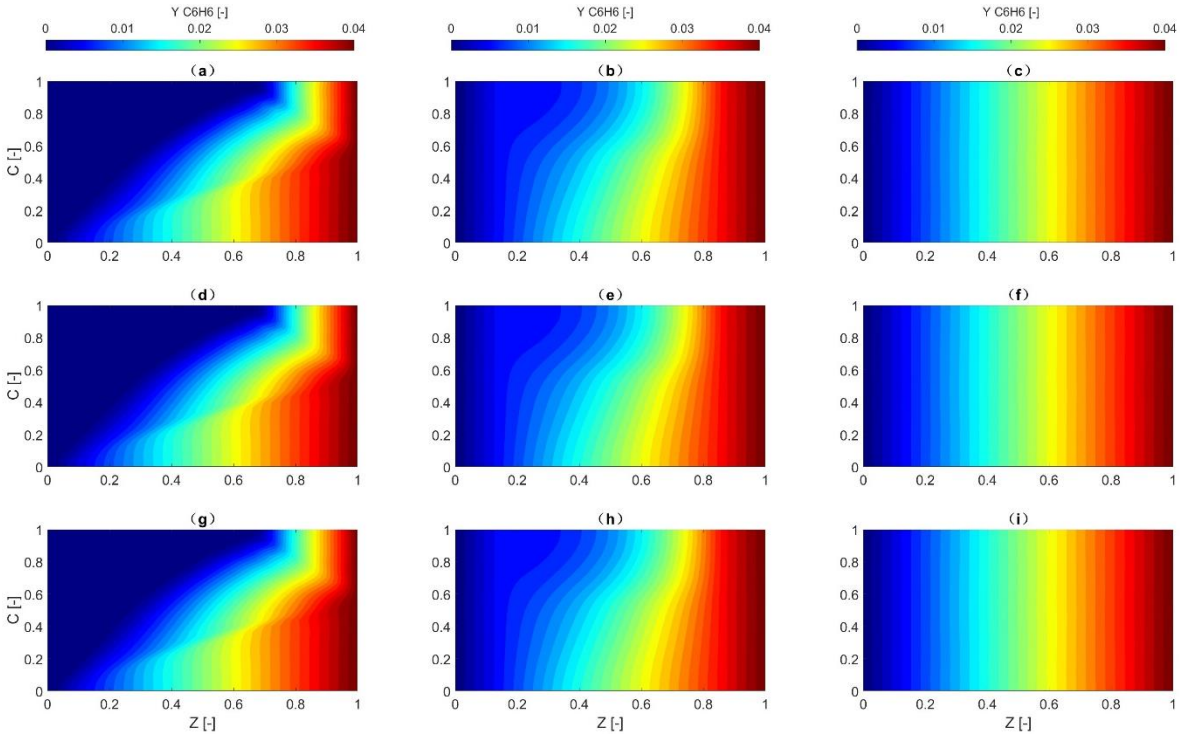

Figure 4.30 *Y_C6H6* distributions in $Z \times C$ space from Fitting result

Fig 4.29 and Fig 4.30 show the distribution of *Y_C6C6* in $Z \times C$ space from the HCOG database and fitting results. The set values for *ZV* and h^n for each case are:

(a) *ZV*=0 [-], h^n=0 [-].
(b) *ZV*=0.125 [-], h^n=0 [-].
(c) *ZV*=0.25 [-], h^n=0 [-].
(d) *ZV*=0 [-], h^n=0.5 [-].
(e) *ZV*=0.125 [-], h^n=0.5 [-].
(f) *ZV*=0.25 [-], h^n=0.5 [-].
(g) *ZV*=0 [-], h^n=1 [-].
(h) *ZV*=0.125 [-], h^n=1 [-].
(i) *ZV*=0.25 [-], h^n=1 [-].

First, with $Z = 1$ for the fuel side and $Z = 0$ for the oxidizer side, it can be seen that the distribution of C_6H_6 is concentrated on the fuel side, while on the oxidizer side *Y_C6H6* is 0, which is in line with the previous prediction. As *C* increases, the chemical reaction process go further, and C_6H_6 is gradually being consumed in this process, so the *Y_C6H6* distribution area rapidly narrows near the fuel side as *C* increases.*Y_C6H6* is not much affected by h^n, and the chemical reaction becomes unstable as *ZV* increases, so *Y_C6H6* becomes a stepwise distribution when *ZV* is 0.25. Comparing Fig 4.29 and Fig 4.30, it can be seen that the distribution of *Y_C6H6* from the fitting result is still consistent, and there is not much difference between these two cases.

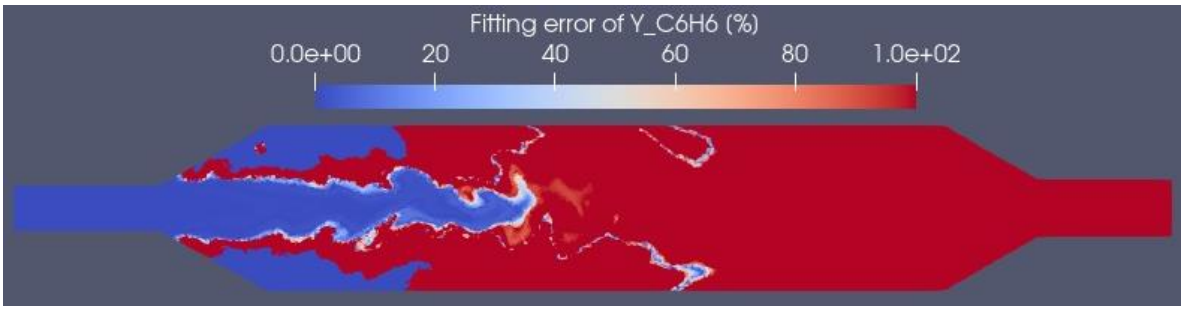

Figure 4.31 Snapshots for the distribution of *Y_C6H6* fitting errors in the reactor

Fig 4.31 shows that the fitting error distribution of *Y_C6H6* is exactly opposite to that of *Y_C2H2*. In the outer part of the reaction zone and downstream of the reactor, the fitting error of *Y_C6H6* is very large, due to the fact that there is no C_6H_6 distribution in these area, but the neural network still gives small values close to 0 but still not 0 in these area. In the

region where *Y_C6H6* is not 0, the fitting error is roughly within 10%, and the fitting result here is barely acceptable.

The fitting results for *Y_C2H2* and *Y_C6H6* enroll that too large a difference in the order of magnitude of the target data has a devastating effect on the accuracy of the fitting, I don't have a general solution to this problem at the moment. Only a specific special solution can be given for a specific problem. For example, artificially modifying the fitting results or constructing multiple neural networks to fit the same target data in groups.

4.4 Reduction of memory size

As you can see from the fitting results, most of the data is fitted with an error of 10% or less, which is within acceptable limits. So it goes back to the original question, what are we putting up with this 10% error for? As mentioned before, the neural network fitting is to solve the problem that the database is too large and occupies too much CPU memory, thus leaving no memory for simulation.

Table 4.1 compares the memory size between HCOG database and the neural network reconstructed in Fortran. Since the next step simulation is based on the Fortran language, I rebuilt the neural network in Fortran based on MATLAB's fitting result, which allows for both reading the HCOG database and importing the data into MATLAB for fitting, and it also contains the weight and bias information for the neural network after MTALAB's fitting, meaning that it can give 21 outputs' fitting result based on the four inputs (*Z*, *ZV*, *C*, h^n) that are entered into the code.

Table 4.1 Memory size comparison between HCOG database and fitted neural network

Projects	**Memory size**
HCOG database	2.29 Gigabytes
Neural network reconstructed in Fortran	1.69 Megabytes

The size of the HCOG database is 1390 times larger than the size of the Fortran file of the fitted neural network, and the use of the neural network instead of the initial HCOG database can result in significant memory savings while meeting the accuracy requirements. Then, in the limited CPU memory, more space can be saved, and the amount of data import can be increased or the database dimension can be increased in improving the accuracy of simulation results.

Chapter 5

Conclusions

In this work, the structure of the database and the fundamentals of neural networks are presented. A detailed derivation of the LM algorithm is presented, as well as the relationship between Forward Propagation, Backward Propagation and iterative loops. Two representative data from each group of output data have been selected to show the fitting results. Group A and Group B show very optimistic fitting accuracy, while Group C output data still need to invest more time in the future to find a generalized solution to improve the fitting accuracy due to its too large order of magnitude difference.

5.1 Neural network algorithm

For the problem of finding the minimum of the Mean Square Error equation (Eq. (2.10)), this paper provides several ideas.

(i) The gradient descent method can be approximated as a first-order expansion of Taylor, which is very simple and intuitive, but it also means that there are more iterative steps, which is not very accurate in the face of complex structural database iterations. The gradient descent method works by calculating the gradient of the objective function at the current parameter position and then updating the parameters along the negative gradient direction. This gradually approaches the optimal solution.

(ii) The Gauss-Newton algorithm can be viewed as a second-order Taylor expansion, which is computed more accurately than a first-order matrix due to the second-order matrix information involved. The Gauss-Newton algorithm simplifies the computation and iteration time by using the first-order Jacobian matrix replacing the second-order Hessian matrix and optimizes the least-squares method so that it can be applied to nonlinear regression problems.

(iii) Levenberg-Marquardt algorithm combines the gradient descent method and Gauss-Newton algorithm by varying the values of the parameters during the iterative computation, thus allowing the algorithm to automatically switch between the these two algorithms. When moving away from the optimal solution, the LM algorithm mainly adopts the gradient descent method, which utilizes a large step size to quickly approximate the optimal solution. When close to the optimal solution, the Gauss-Newton algorithm is utilized with small update steps in order to accurately converge to the optimal solution.

The LM algorithm is the algorithm used in the cases of this paper, which can balance the fitting accuracy and fitting time well and performs well in the fitting results of this paper.

5.2 Iterative loops for neural networks

This paper describes in detail how Forward Propagation and Backward Propagation allow neural networks to form a closed-loop computational process.

(i) Except for the first loop in which the weights and biases are randomly generated, the essence of each Forward Propagation is to import the input data of the current loop combined with the updated weights and biases of the previous loop to calculate the output data of the current loop, thus preparing for the Backward Propagation. To illustrate the computational process of Forward Propagation, this paper also describes the rules for the computation of neuron, weight and bias.

(ii) Backward Propagation is the core part of neural network fitting, which is essentially a comparison of the fitting values calculated by this Forward

Propagation with the target data, and Back Propagation along the output layer to the input layer by calculating the loss function while updating the weights and bias of the current loop.

The result of neural network fitting is to obtain an array of weights and bias matrices. Since the next HCOG simulation was run on the server using the Fortran language, I used the weights and bias data obtained from the neural network fitting and reconstructed the neural network using Fortran to replace the HCOG database.

5.3 Summary of fitting results and memory size reduction

From the fitting results, the fitting data of Group A and Group B are satisfying the fitting requirements with errors within 10%, while the data of Group C still need to continue to find ways to improve them.

(i) The commonality between the data in Group A and Group B is that the order of magnitude maxes out within 5, which is the acceptable limit for neural network fitting. For data within this range, the neural network gives highly accurate fitting results even in the hard-to-fit region around 0, which is very exciting. For the Group A data, only a normalization preprocessing of the data is required to get a fitting within 5% error. For Group B data, since all of them are mass fractions of chemical components with values between 0 and 1, and most of them are small, some of them need to be multiplied by 10^5 on top of the normalization pre-processing in order to avoid the early termination of the fitting process due to the large initial fitting error.

(ii) For Group C data, their order of magnitude difference will reach 7 or 8, which is beyond the resolution of the neural network. Near 0, the neural network cannot give a satisfactory fitting value. Although we can bind the position coordinates in the reactor with the HCOG database and consider modifying the fitting results, it is too inefficient and not a general solution for databases with different structures. For areas where the target value is not 0, the fitting error is barely maintained at

around 10%. For the preprocessing and postprocessing of Group C data, I will provide some ideas below.

(iii) For Group C data, their order of magnitude difference will reach 7 or 8, which is beyond the resolution of the neural network. Near 0, the neural network cannot give a satisfactory fitting value. Although we can bind the position coordinates in the reactor with the HCOG database and consider modifying the fitting results, it is too inefficient and not a general solution for databases with different structures. For areas where the target value is not 0, the fitting error is barely maintained at around 10%. For the preprocessing and postprocessing of Group C data, I will provide some ideas below.

(iv) The size of the HCOG is 2.29GB, and the fitted neural network size reconstructed in Fortran is 1.69MB, with a difference of up to 1300 times in size between the two. If the fitting error is within an acceptable range, using neural networks to replace databases can greatly save storage space.

5.4 Future work

At present, I have completed the classification and extraction of binary data using Fortran, and they will be directly imported into MATLAB in CSV file type for neural network fitting. We can modify the matrix parameters in the Fortran file according to the weights and biases in the fitting results. The final Fortran file can realize the extraction and export of target data, and the fitting and export of fitting results. In the actual simulation process, only this Fortran file is needed, without the participation of the target database and MATLAB file. This means that the simulation will not only be limited to Windows systems, but Linux systems can also meet the requirements.

The fitting of group C data is like a dark cloud in the sky, which has been bothering me. In the past six months, I have tried various methods in order to find a general solution to improve the fitting error when the data magnitude difference is more than 5. At present, I have summarized some methods, but I still haven't completely solved this problem.

(i) No matter what data structure is aimed at, the pre-processing of data normalization is necessary, which greatly reduces the complexity and fitting

difficulty of the data structure. It should be noted that when building a neural network yourself, don't forget to denormalize the fitting results before outputting.

(ii) For the target data with a small value represented by the mass fraction, it is necessary to enlarge the database value as a whole. As mentioned earlier, there are two types of termination flags for the iterative process of neural network fitting. In addition to the specified times of fittings, the fitting error is also an important standard for determining whether the fitting will continue. The initial weights and biases of the neural network are randomly generated. If the target value is too small, the neural network is likely to terminate the iteration early due to the large fitting error of the first fifty times of iteration.

(iii) Import the coordinate information of the reactor as output data into the HCOG database. We can use PARAVIEW to visualize the HCOG database. After knowing the parameters of the reactor, the position information of each node becomes a known quantity. We can manually modify the fitting results based on the reaction characteristics of different regions in the reactor. But this method only targets a single database and requires specific analysis of specific issues.

(iv) The challenge with Group C data lies in the large order of magnitude difference. Now I'm trying to target a certain output data, such as *Y_ C2H2*, I only perform neural network fitting on the data within the range of 0 to 10^{-3}, and then apply the fitted neural network to the entire range. This raises a question: Should we fit to large values or small values? How do we select only nodes with *Y_C2H2* in the range of 0 to 10^{-3}, but it can ensure that these nodes cover the entire reactor and reflect the distribution trend inside the entire reactor? I am currently unable to provide an answer, which will be my main work in the future.

References:

[1] McCulloch, W.S., Pitts, W. A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics 5, 115–133 (1943).

[2] LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. Nature 521, 436–444 (2015).

[3] José M. Bermúdez, Ana Arenillas, Rafael Luque, J. Angel Menéndez,An overview of novel technologies to valorise coke oven gas surplus,Fuel Processing Technology,Volume 110,2013,Pages 150-159.

[4] N. Carlini and D. Wagner, "Towards Evaluating the Robustness of Neural Networks," 2017 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 2017, pp. 39-57, doi: 10.1109/SP.2017.49.

[5] Ning Qian,On the momentum term in gradient descent learning algorithms,Neural Networks,Volume 12, Issue 1,1999,Pages 145-151,ISSN 0893-6080.

[6] H. Iiduka, "Appropriate Learning Rates of Adaptive Learning Rate Optimization Algorithms for Training Deep Neural Networks," in IEEE Transactions on Cybernetics, vol. 52, no. 12, pp. 13250-13261, Dec. 2022, doi: 10.1109/TCYB.2021.3107415.

[7] H. Hindi, "A tutorial on convex optimization," Proceedings of the 2004 American Control Conference, Boston, MA, USA, 2004, pp. 3252-3265 vol.4, doi: 10.23919/ACC.2004.1384411.

[8] Bottou, L. (2012). Stochastic Gradient Descent Tricks. In: Montavon, G., Orr, G.B., Müller, KR. (eds) Neural Networks: Tricks of the Trade. Lecture Notes in Computer Science, vol 7700. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-35289-8_25.

[9] Huo, Z., & Huang, H. (2017). Asynchronous Mini-Batch Gradient Descent with Variance Reduction for Non-Convex Optimization. Proceedings of the AAAI Conference on Artificial Intelligence, 31(1). https://doi.org/10.1609/aaai.v31i1.10940.

[10] P. E. GILL , W. MURRAY, Quasi-Newton Methods for Unconstrained Optimization, IMA Journal of Applied Mathematics, Volume 9, Issue 1, February 1972, Pages 91–108, https://doi.org/10.1093/imamat/9.1.91.

[11] Yuan, Yx. Recent advances in trust region algorithms. Math. Program. 151, 249–281 (2015). https://doi.org/10.1007/s10107-015-0893-2.

[12] Madsen, K. A root-finding algorithm based on Newton's method. BIT 13, 71–75 (1973). https://doi.org/10.1007/BF01933524.

[13] M. Uzair and N. Jamil, "Effects of Hidden Layers on the Efficiency of Neural networks," 2020 IEEE 23rd International Multitopic Conference (INMIC), Bahawalpur, Pakistan, 2020, pp. 1-6, doi: 10.1109/INMIC50486.2020.9318195.

[14] Nielsen, Michael A. Neural networks and deep learning. Vol. 25. San Francisco, CA, USA: Determination press, 2015.

ACKNOWLEDGMENTS

One year and two months in Japan are coming to an end, and it seems like I just arrived at Fukuoka airport last week, so I can't help but feel that time has flown by.

I am very grateful to my advisor, Prof. Watanabe of Thermal Science and Energy Laboratory for his selfless help. At the beginning of my master's program, I knew nothing about deep learning, and Prof. Watanabe took great pains to tutor me so that I could slowly start to get started. During the two years of graduate school, I made many mistakes, and every time Prof. Watanabe guided me calmly.

Thanks to Dr. Panlong Yu, who has helped me a lot in my life, and I wish him good luck in his future work.

Thank you to all the students in Thermal Science and Energy Laboratory and other international students at Kyushu University. I was unable to go to Japan for the first six months because of COVID-19, and I asked myself many times in the middle of the night, "Do I have to study abroad?" But after meeting many Japanese and international friends, I truly believe that it was all worth it, and if I had another chance, I would choose Japan and Kyushu University without hesitation.

Finally, I am very grateful to Kyushu University for providing me with a platform with a broad vision, and I have had a very good research life, which will be a good memory for the rest of my life.

I will come back.